UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

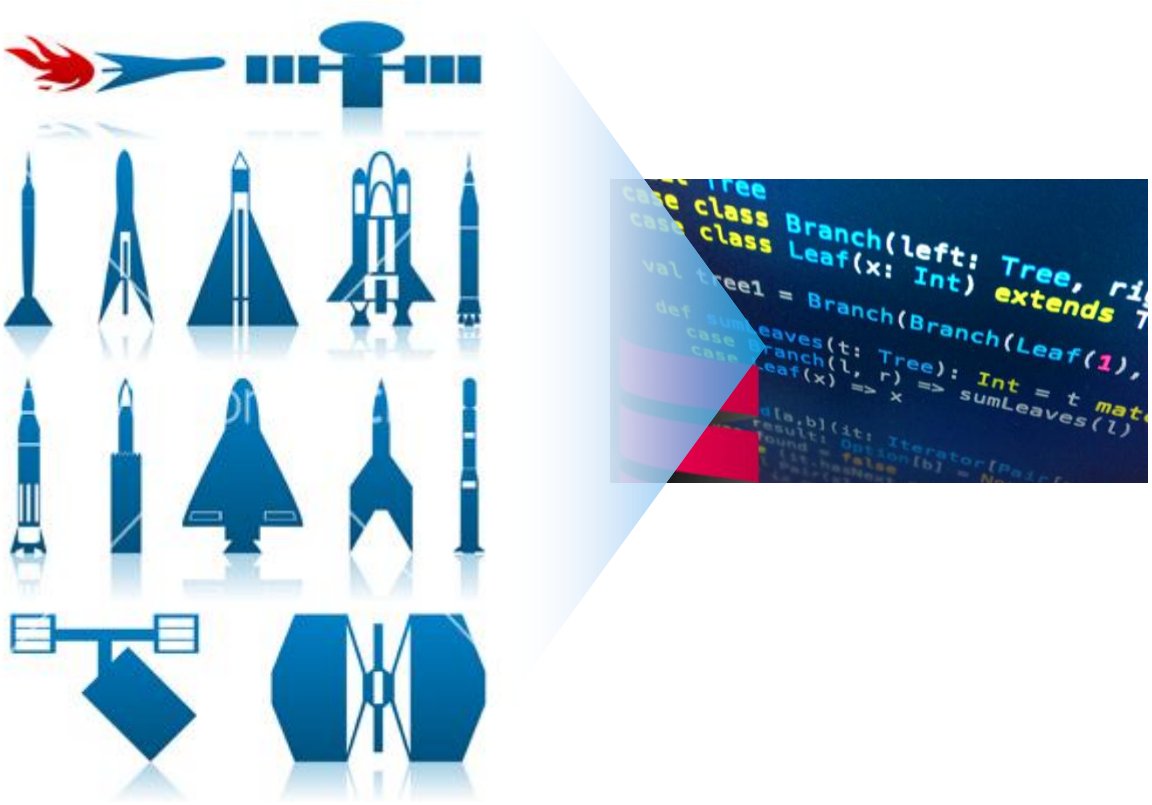
MÁSTER UNIVERSITARIO EN INVESTIGACIÓN EN INGENIERÍA DE SOFTWARE Y SISTEMAS INFORMÁTICOS

ITINERARIO DE INGENIERÍA DE SOFTWARE

CÓDIGO 31105128

TRABAJO FIN DE MASTER

ArchE: Evaluación de Arquitecturas y Toma de Decisiones. Aplicación al Análisis de Arquitecturas Software en el Sector Aeronáutico



Estudiante: Jose Antonio Miranda Rodera

Profesor: José Félix Estívariz López

Curso: 2014/2015

Convocatoria: Junio - 2015

UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

MÁSTER UNIVERSITARIO EN INVESTIGACIÓN EN INGENIERÍA DE SOFTWARE Y SISTEMAS INFORMÁTICOS

ITINERARIO DE INGENIERÍA DE SOFTWARE

CÓDIGO 31105128

ArchE: Evaluación de Arquitecturas y Toma de Decisiones. Aplicación al Análisis de Arquitecturas Software en el Sector Aeronáutico

Trabajo Fin de Máster Tipo A

Estudiante: Jose Antonio Miranda Rodera

Profesor: José Félix Estivariz López

Curso: 2014/2015

Convocatoria: Junio - 2015

CALIFICACIÓN:

AUTORIZACIÓN:

RESUMEN

El diseño de una arquitectura software no sólo ha de tener en cuenta los requisitos funcionales que dicha arquitectura debe de cumplir; también se deben de tener en cuenta otro tipo de requisitos, denominados de atributos de calidad, que dirigen el diseño para que cumpla otros requisitos adicionales; así, la arquitectura tendrá en cuenta no sólo requisitos del cliente, sino que cubrirá aspectos como la modificabilidad, el rendimiento, la escalabilidad o la seguridad.

Existen herramientas que ayudan al arquitecto software a comprobar si una arquitectura es capaz de cumplir con los atributos de calidad que se especifican para la misma, como ArchE. Dicha herramienta es también capaz de generar propuestas de mejora o tácticas, que modifican parcialmente esta arquitectura para poder cumplir mejor los requisitos de atributos de calidad.

ArchE funciona bien con arquitecturas convencionales, como por ejemplo, las basadas en Modelo-Vista-Controlador, con el software de aplicación distribuido en servidores globales y aplicaciones locales en hardware móvil o estaciones de trabajo. La pregunta sería ahora: ¿es válida esta herramienta para analizar otro tipo de arquitecturas no convencionales? ¿Puede ArchE ayudar a un arquitecto software a diseñar otro tipo de arquitecturas?

En este trabajo fin de máster se aborda dicha problemática aplicada a un campo concreto: las arquitectura software aeroespaciales, que son arquitecturas muy específicas y sometidas a unos rigurosos mecanismos de diseño. En este trabajo se proponen las dos tipos de arquitecturas más frecuentes en el campo aeroespacial, como son las arquitecturas software de simulación y las arquitecturas de software embebido en equipos de aviónica.

Utilizando ArchE, se analizarán ambos tipos de arquitectura, se presentarán los resultados obtenidos, y se analizará si las especificaciones técnicas de ArchE son suficientes para realizar un análisis cualitativo adecuado o bien se necesita implementar otros marcos de razonamiento para que ArchE pueda ser capaz de analizar con éxito dichas arquitecturas software aeronáuticas y aeroespaciales.

El resultado final se incluirá en las conclusiones del trabajo, así como posibles trabajos futuros.

PALABRAS CLAVES

ArchE, atributos de calidad, tácticas, marcos de razonamiento, arquitectura software, diseño dirigido por atributos, software aeroespacial, IMA, particiones.

ABSTRACT

The design of a software architecture does not have only to take into account the functional requirements that such architecture must fulfill but also it has to take into account other type of requirements, called quality attributes requirements, that drive the design to fulfill additional requirements; in this way, the architecture will take into account not only customer requirements but other aspects like modifiability, performance, scalability or security.

There are tools to help the software architect if an architecture is able to fulfill the quality attributes specified for it, like ArchE. Such tool is also able to make improvement proposals, so called tactics, which partially modify this architecture to fulfill in a better way the quality attributes requirements.

ArchE works fine with conventional architectures, like for example, the ones based in Model-View-Controller, with the application software distributed in global servers and local applications in mobile hardware and work stations. The question now is: is this tool useful to analyze other non conventional architectures? Can ArchE help a software architect to design other type of architectures?

In this master thesis this issue is addressed, applied to a specific scope: the aerospace software architectures, which are very specific architectures, submitted to a strict design mechanisms. In this thesis two types of the most common architectures in the aerospace scope are proposed: the simulation software architectures and embedded software architecture of avionics equipment.

Using ArchE, both types of architecture will be analyzed; the achieved results will be presented; and the technical specifications of ArchE will be analyzed to find out if they are good enough to carry out a suitable qualitative analysis or on other hand another reasoning framework is required to be implemented, to get ArchE is able to analyze such aerospace sowftware architectures successfully.

The final results will be included in the theses conclusions, as well as possible future works.

KEYWORDS

ArchE, quality attributes, tactics, reasoning frameworks, software architecture, attribute-driven design, aerospace software, IMA, partitions.

AGRADECIMIENTOS

A mi padre, por creer siempre en mí, te llevamos siempre en nuestro corazón aunque ya no estés.

A mi madre, por ser tan luchadora y salir siempre adelante.

A Virginia, por tanta paciencia, comprensión y cariño que me ha dado durante este largo año, y por ser siempre mi apoyo.

A mi profesor de trabajo fin de máster, José Félix, por su apoyo y sabia dirección.

A todos, muchas gracias.

Índice de Contenido

Índice de Figuras

Índice de Tablas

1 CAPÍTULO 1 – INTRODUCCIÓN. OBJETIVOS

1.1 Introducción al Trabajo Fin de Máster

El desarrollo de las arquitecturas software no es sólo una cuestión puramente técnica; no se trata sólo de ver de qué manera se pueden implementar una serie de requisitos y de cumplirlos. Con la experiencia, los arquitectos software también se dieron cuenta de que tiene una gran importancia el cómo cumplir unos requisitos técnicos. Una arquitectura software puede cumplir los requisitos de diseño para los cuales ha sido pensada, pero su rendimiento puede ser pobre; o también puede ser difícil de modificar o de mantener. O quizás dicha arquitectura sea aceptable en un momento dado, pero no cuando el sistema crezca, o cuando haya que portar dicha arquitectura a otra plataforma. O puede que la arquitectura sea funcionalmente buena pero no cumpla unos requisitos mínimos de seguridad.

Por lo tanto, una arquitectura software no sólo debe de cumplir los requisitos funcionales; también debe de cumplir otros requisitos mucho más genéricos, que hagan que dicha arquitectura sea robusta bajo otros puntos de vista.

El estudio del cumplimiento de dichos requisitos es lo que ha dado lugar a la implementación de los requisitos de calidad. Dichos requisitos han de tenerse en cuenta al comienzo del diseño general de la arquitectura, puesto que pueden tener repercusiones técnicas a la hora de tomar las decisiones de diseño.

Para cumplir dichos requisitos de diseño, se recurre a las tácticas, que son decisiones de diseño encaminadas a controlar la respuesta del sistema o una parte del mismo a un estímulo proveniente de un requisito de calidad.

Resolver el cumplimiento de los requisitos de calidad no es tarea fácil, puesto que hay una gran variedad de los mismos y su análisis para un sistema completo requiere un gran esfuerzo. Para ayudar al arquitecto software, hay herramientas que implementan marcos de razonamiento que permiten evaluar un determinado requisito de calidad y ver si el sistema lo cumple o no, y en caso de que no lo cumpla, sugerir tácticas para mejorar la arquitectura. Es el caso de ArchE, herramienta que es el objeto de estudio de este trabajo.

Es habitual en este tipo de trabajos fin de máster el aplicar la herramienta ArchE sobre arquitecturas convencionales, basadas en el patrón arquitectónico Modelo-Vista-Controlador, sobre el cual ArchE puede proporcionar muy buenos resultados en la mejora del cumplimiento de los atributos de calidad. Pero ¿qué ocurriría si se intenta aplicar ArchE sobre una arquitectura no convencional, como por ejemplo, una arquitectura de software aeroespacial? Estas arquitecturas tienen sus propias características, debido a que su diseño debe estar altamente sujeto a normas de diseño y de calidad, debido a que tiene unos requisitos de seguridad (entendiendo seguridad por la traducción directa de la palabra inglesa *safety*, es decir seguridad para las personas o seguridad en vuelo, no seguridad informática) muy estrictos.

Este es pues el objeto de este trabajo fin de máster: intentar ver qué ocurre con estas arquitecturas software aeroespaciales cuando se intentan analizar con una herramienta como

ArchE que, en principio, no está pensada para las particularidades que tienen dichas arquitecturas.

En la primera parte del trabajo (capítulos 2, 3 y 4) se expondrá una introducción a los conceptos principales que se manejarán a lo largo de este estudio, como son: arquitecturas software, patrones software, atributos de calidad, tácticas, y métodos de análisis de arquitecturas software.

En la segunda parte del trabajo (capítulos 5 y 6) se hará una introducción de los conceptos más importantes de las arquitecturas software aeroespaciales, como son: principios de diseño de software aeronáutico, normas de calidad en el diseño de software aeronáutico, introducción a las arquitecturas software IMA (*Integrated Modular Avionics*).

En la tercera parte del trabajo (capítulos 7 y 8) se presentará el análisis de dos casos prácticos de arquitecturas software aeroespaciales, que se corresponden a las más típicas: un simulador de vuelo, y un sistema de aviónica con software embebido y software de aplicación. Se analizará también cómo la herramienta ArchE analiza dichas arquitecturas, evaluando si la herramienta puede o no manejar las peculiaridades de dichas arquitecturas.

Por último, se presentarán unas conclusiones finales (capítulo 9) en las que se resuman los resultados obtenidos en la tercera parte del trabajo y se planteen líneas de investigación futuras.

Adicionalmente, se incluirán en este trabajo tres anexos:

- el Anexo 1 contendrá información sobre la herramienta ArchE: cómo trabaja la herramienta, sus principios teóricos, sus marcos de razonamiento y las capacidades que proporciona.
- el Anexo 2 contendrá una completa guía de instalación de la herramienta, ya que se ha observado durante este trabajo que hay ciertas dificultades en algunos pasos; en este anexo se pretende ofrecer una guía muy clara de los pasos a seguir.
- el Anexo 3 contendrá una guía de usuario de ArchE, para ayudar a los arquitectos software que se quieran iniciar en el uso de esta herramienta, exponiendo sus funcionalidades básicas, además de una serie de notas prácticas importantes y limitaciones a tener en cuenta sobre la herramienta.

Nota adicional 1: para la realización de este trabajo se han consultando diversas fuentes bibliográficas. Las referencias a dichas fuentes se añaden en el texto entre corchetes [ ], así como las referencias de las figuras incluidas. En el capítulo 10 se incluye una lista con todas las fuentes bibliográficas consultadas.

Nota adicional 2: para la realización de los diagramas UML de los capítulos 7 y 8 se ha utilizado la herramienta Software Ideas Modeler, versión 7.50.5300.41802.

1.2 Objetivos del Trabajo Fin de Máster

1.2.1 Objetivos Generales

- Adquirir conocimientos precisos sobre los métodos de análisis de arquitecturas software basados en atributos de calidad, y presentarlos en este trabajo a modo de resumen.
- Presentar una introducción a las arquitecturas software aeroespaciales y sus características específicas.
- Familiarizarse con la herramienta ArchE: capacidades y limitaciones, modo de uso, instalación, etc.

1.2.2 Objetivos Específicos

- Analizar dos modelos de arquitecturas aeroespaciales con la herramienta ArchE, en unos determinados escenarios.
- Presentar los resultados obtenidos para dichas arquitecturas y ver si hay margen para la mejora basada en las tácticas propuestas por ArchE.
- Analizar si las tácticas que propone ArchE son aplicables o no a los casos prácticos.
- Analizar si ArchE puede ser capaz de manejar dichas arquitecturas o si tiene alguna limitación a la hora de ejecutar sus marcos de razonamiento sobre las mismas.
- Si ArchE tiene alguna limitación a la hora de analizar alguna de las arquitecturas, proponer mejoras en la herramienta para que se pueda adaptar y sirva para realizar dichos análisis de una forma adecuada.

2 CAPÍTULO 2 - ARQUITECTURA SOFTWARE: INTRODUCCIÓN

2.1 Definición de Arquitectura Software

El concepto de Arquitectura software es relativamente nuevo, y sólo tiene unas décadas de vida. Este concepto no es fácil de definir si se aplica al software, en contraste con otras disciplinas de ingeniería, como la construcción, aeronáutica, e incluso el diseño hardware, en las cuales la existencia de una arquitectura es obvia.

Se intentará dar en esta sección una visión desde varias perspectivas de lo que es la arquitectura software.

Según [1], la arquitectura software de un sistema de computación es la estructura o estructuras del sistema, la cual incluye elementos software, las propiedades visibles externamente de estos elementos, y las relaciones entre todos ellos y con su entorno.

Esta sería una definición de arquitectura software como estructura; su definición como disciplina sería: una arquitectura software es aquella que cubre las actividades arquitectónicas y las decisiones relacionadas acerca del diseño e implementación de una arquitectura software.

Por lo tanto, según [1], arquitectura software es un concepto que es en sí mismo la suma de otros dos conceptos: uno como estructura, y otro como disciplina. Si se tiene en cuenta que aquí se habla no de una arquitectura software aislada sino de una arquitectura de un sistema completo (que consta de software y de hardware), habría que matizar que la arquitectura de un sistema como estructura es equivalente a la ya mencionada arquitectura software pero teniendo en cuenta que tiene elementos software y hardware; respecto a la arquitectura de sistema como disciplina, la definición es igual que la anteriormente mencionada.

La arquitectura software es parte de la arquitectura de un sistema y estructura los bloques software que constituyen dicho sistema. En la Figura 1 se puede ver la relación entre ambas:

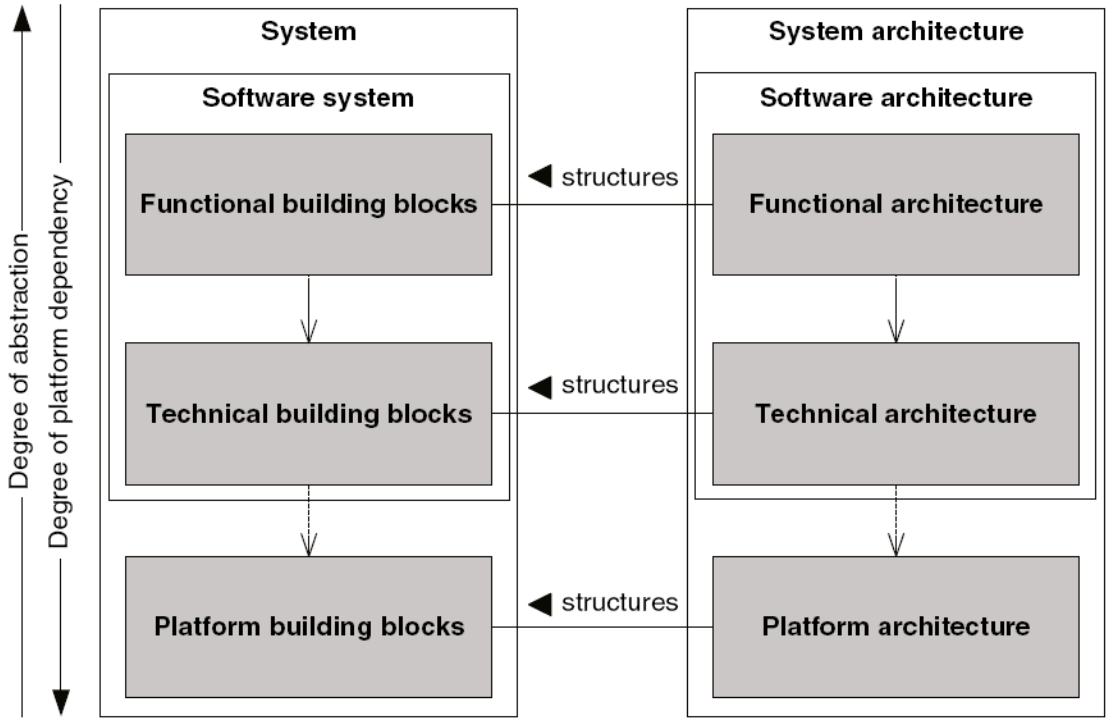


Figura 1. Componentes de una arquitectura y sus relaciones. [1]

Se pueden clarificar algunos puntos de la definición dada anteriormente. En [2] se especifica que una arquitectura software define un sistema en términos de componentes computacionales e interacciones entre todos esos componentes. Dichos componentes pueden ser clientes y servidores, bases de datos, filtros y capas en un sistema jerárquico. Las interacciones entre componentes a ese nivel de diseño pueden ser simplemente una llamada a procedimiento y un acceso a variables compartidas; pero también puede ser tan complejo como un protocolo de cliente-servidor, protocolos de acceso a bases de datos o multidifusión de evento asíncrono.

Además, [2] destaca que la arquitectura no sólo consiste en especificar la estructura y topología del sistema, sino que además muestra la correspondencia entre los requisitos del sistema y los elementos del sistema construido, proporcionando por tanto alguna justificación a las decisiones de diseño. Por ejemplo, se pueden citar propiedades como capacidad, rendimiento, consistencia y compatibilidad de componentes.

De una manera más general, los modelos arquitectónicos clarifican las diferencias estructurales y semánticas entre componentes e interacciones. Estos modelos arquitectónicos a menudo pueden ser compuestos para definir sistemas más grandes.

Cuando se habla de arquitectura software, no hay que olvidar el componente de rigor que implica implícitamente, así como las técnicas usadas para capturar y expresar características de diseño arquitectónico. La arquitectura software implica [3]:

- El conjunto de especificaciones de requisitos derivados de interacciones con los responsables del producto software

- Las representaciones funcionales de los comportamientos software y las interacciones entre usuarios, operadores y sistemas externos.
- La disposición física o estructural de los bloques componentes de software y la estrategia para combinar estos elementos software en un único producto integrado.

Los elementos de la arquitectura software, el entorno de computación y las relaciones y dependencias que existen entre todos estos elementos están identificados en la Figura 2: [3]

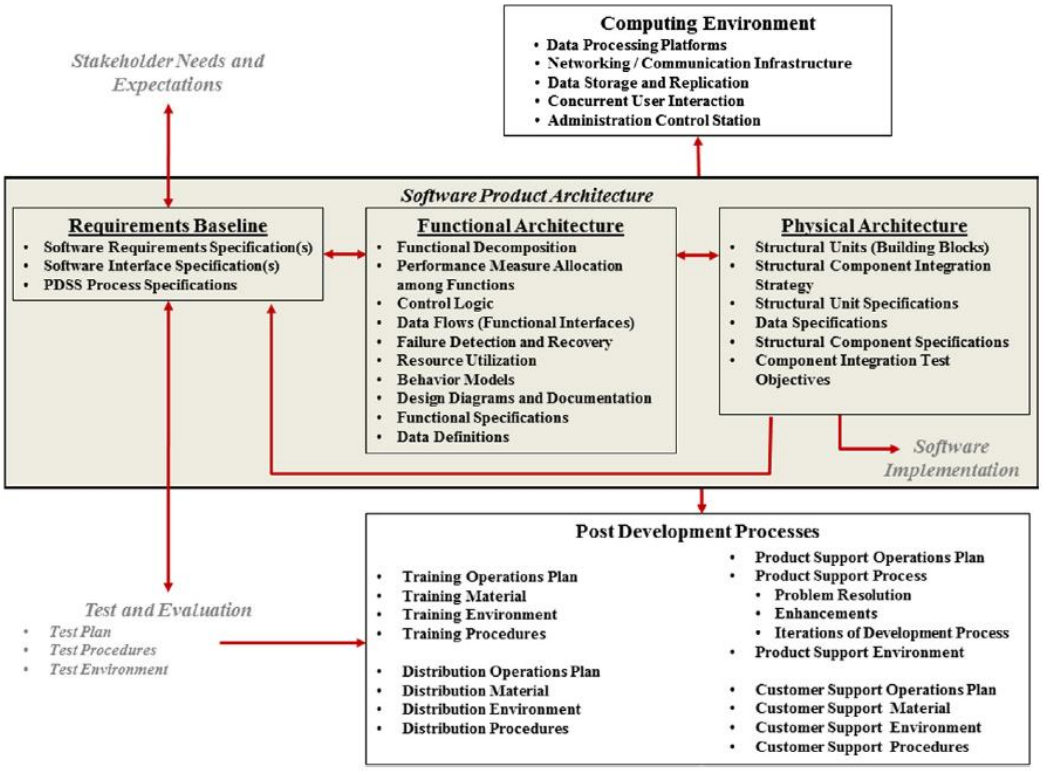


Figura 2. Elementos de la arquitectura software. [3]

2.2 Características Fundamentales de una Arquitectura Software

De lo visto en las diferentes definiciones de arquitectura software, se puede concluir que la arquitectura software tiene una serie de características fundamentales que se describen a continuación, según [4].

2.2.1 La Arquitectura Define una Estructura

Gran parte del diseño de un sistema está concentrado en cómo se puede particionar una aplicación en un conjunto interrelacionado de componentes, módulos, objetos o cualquier unidad de software que realiza una tarea.

Para realizar esta partición, se tendrán que tener en cuenta los requisitos de aplicación y las restricciones.

En este proceso de partición, el arquitecto software asignará responsabilidades a cada componente, que a su vez definirán las tareas que dicho componente podrá realizar dentro de la aplicación.

Un concepto clave a tener en cuenta es la optimización que lleve a minimizar las dependencias entre componentes, entendiendo como dependencia el hecho de que un cambio en un componente afecte a otros.

2.2.2 La Arquitectura Especifica la Comunicación entre Componentes

Al dividir una aplicación en componentes, es necesario establecer un modelo de comunicación entre los mismos. Para ello, es muy común el definir una serie de patrones de arquitecturas (que se analizarán más adelante), cada uno de los cuales ya establece una estructura y un modelo de comunicaciones entre componentes. La elección de un patrón u otro dependerá de los requisitos de la aplicación que han de satisfacerse.

2.2.3 La Arquitectura Analiza Requisitos No Funcionales

Los requisitos no funcionales son aquellos que no aparecen en los casos de uso. Más que decir qué hace la aplicación, dicen cómo la aplicación proporciona la funcionalidad requerida.

Hay tres áreas diferentes de requisitos no funcionales:

- *Restricciones técnicas:* restringen las opciones de diseño especificando ciertas tecnologías que la aplicación debe usar.
- *Restricciones de negocio:* restringen las opciones de diseño pero por motivos de negocio, no por razones técnicas.
- *Atributos de calidad:* definen requisitos de una aplicación en términos de escalabilidad, disponibilidad, facilidad de cambio, portabilidad, usabilidad, prestaciones, etc.

2.2.4 La Arquitectura es una Abstracción

Cualquier descripción de arquitectura debe emplear la abstracción, para ser entendible por todos los miembros del equipo de diseño y desarrollo. Esto implica que los detalles superfluos son suprimidos o ignorados para centrar la atención en los temas más importantes o críticos. Esto se hace describiendo los componentes de la arquitectura como cajas negras, especificando sólo sus propiedades externamente visibles.

2.3 Vistas

Según [4], una arquitectura software representa un artefacto de diseño complejo. Hay por tanto diversas maneras de describir y comprender una arquitectura, basada en las siguientes cuatro vistas:

- *Vista lógica:* describe los elementos más significativos de la arquitectura y las relaciones entre ellos. Para ello utiliza diagramas de clases o equivalentes.
- *Vista de procesos:* enfocado a describir los elementos concurrentes y de comunicación de una arquitectura. En particular, se describen componentes multihilo o replicados, así como mecanismos de comunicación síncronos o asíncronos.
- *Vista física:* representa cómo los procesos principales y componentes se mapean en el hardware de aplicación. Por ejemplo, cómo se distribuyen las bases de datos y los servidores web de una aplicación a lo largo de un determinado número de máquinas servidores.

- *Vista de despliegue:* captura la organización interna de los componentes software, típicamente cómo están sostenidos por un entorno de desarrollo o herramienta de gestión de configuración.

Otra manera de enfocarlo en [4] es la propuesta del *Software Engineering Institute* (SEI) de capturar un modelo de arquitectura mediante tres vistas diferentes:

- *Módulo:* este es una visión estructural de la arquitectura, que comprende los módulos de código, tales como clases, paquetes y subsistemas en el diseño. También captura la descomposición de módulos, asociaciones, agregaciones y herencia.
- *Componente y conector:* esta vista describe los aspectos de comportamiento de la arquitectura. Los componentes típicamente son objetos, hilos o procesos, y los conectores describen cómo interactúan los componentes.
- *Asignación:* esta vista muestra cómo se mapean los procesos en la arquitectura al hardware, y cómo se comunican usando redes y/o bases de datos. También captura una vista del código fuente en los sistemas de gestión de configuración, y quien dentro del grupo de desarrollo tiene la responsabilidad para cada módulo.

En la Figura 3 se representa esta arquitectura: [5]

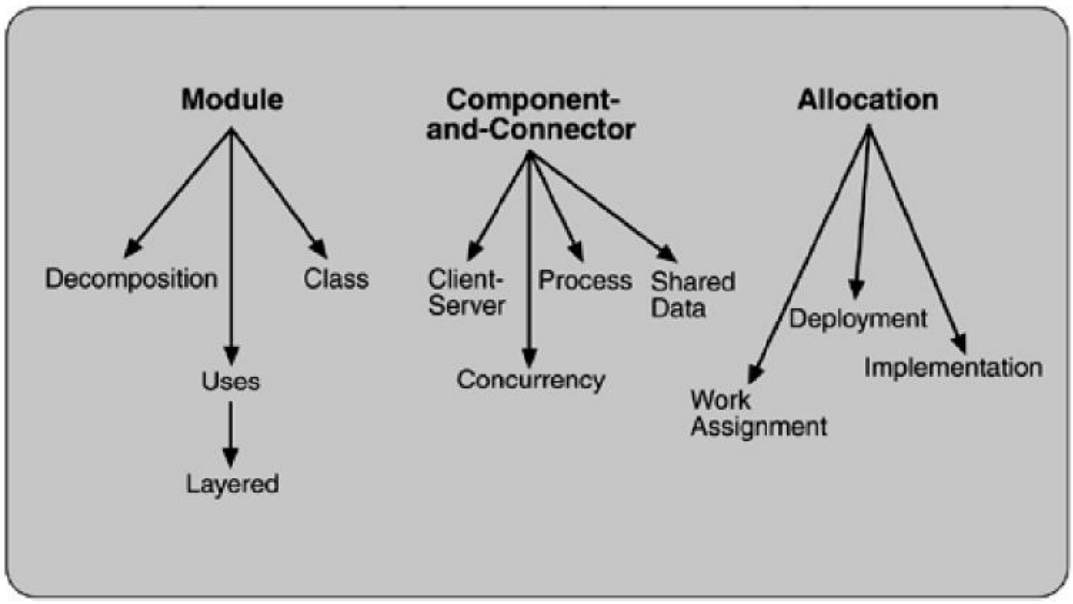


Figura 3. Estructuras de Arquitecturas Software comunes. [5]

2.4 Patrones y Estilos Arquitectónicos

En este apartado se introducen una serie de conceptos muy utilizados a la hora de establecer la arquitectura de un sistema software.

Se empieza este apartado dando una serie de definiciones importantes para entender el resto de apartados.

2.4.1 Patrones Software

Según [6], el propósito de un patrón software es compartir una solución probada y ampliamente aplicable a un problema de diseño particular en un formato estándar que permita hacerlo fácilmente reusado. Los patrones de software deben de proporcionar los siguientes puntos de información:

- *Nombre:* utilizar un nombre lleno de significado conforme a la solución que proporciona.
- *Contexto:* describe las situaciones en las cuales el patrón puede aplicar.
- *Problema:* es una clara declaración del problema que el patrón resuelve y las condiciones que se necesitan cumplir para que el patrón sea aplicado de manera efectiva.
- *Solución:* el núcleo del patrón es una descripción de la solución al problema que el patrón aborda.
- *Consecuencias:* una clara declaración de los resultados y los inconvenientes que puede tener el patrón como resultado de su aplicación.

2.4.2 Estilos, Patrones y Dialectos

En [6] describen una organización de los patrones software en tres grupos: estilos de arquitectura, patrones de diseño y dialectos de lenguajes. A continuación se definen los tres:

- *Estilos de Arquitectura:* expresan un esquema de organización estructural fundamental para los sistemas software. Proporcionan un conjunto de tipos predefinidos de elementos, especifican sus responsabilidades, e incluyen reglas y directivas para organizar las relaciones entre ellos. En definitiva, proporcionan un conjunto de principios organizativos para el sistema como un todo, más que los detalles de una sola parte del sistema.
- *Patrones de diseño:* proporcionan un esquema para refinar los elementos de un sistema software o las relaciones entre ellos. Describen una estructura comúnmente recurrente de elementos de diseño interconectados que solucionan un problema general de diseño dentro de un contexto particular. Un patrón de diseño representa una entrada al diseño software detallado del sistema y guía al diseñador software para organizar sus unidades de diseño software (tales como clases y procedimientos) de forma apropiada.
- *Dialectos de Lenguaje:* son patrones de bajo nivel específicos de un lenguaje de programación. Describen cómo implementar aspectos particulares de elementos o las relaciones entre ellos mediante el uso de características de un lenguaje dado. Proporcionan una guía al programador cuando implementa software en un lenguaje específico y se escribe normalmente para ayudar a prevenir trampas comunes con el lenguaje o para ilustrar características únicas que necesitan ser aprendidas.

Se va a analizar en el resto de esta sección las categorías del grupo más importante en lo relativo a la arquitectura software: los patrones o estilos de arquitectura.

2.4.3 Categorías de los Estilos de Arquitectura

Según [2], [7] y [8], las categorías para los estilos de arquitectura son las siguientes:

2.4.3.1 "From Mud to Structure"

En este estilo los patrones en esta categoría ayudan a evitar un mar de componentes u objetos. Soportan una descomposición controlada de una tarea general del sistema en subtareas cooperantes. Esta categoría incluye las siguientes:

2.4.3.1.1 Patrón de Capas

Ayudan a estructurar aplicaciones que pueden ser descompuestas en grupos de subtareas, cada una de las cuales está en un nivel particular de abstracción. Las capas se organizan de forma jerárquica, y cada capa proporciona servicio a la capa por encima y sirve como capa cliente a la capa inferior. En la Figura 4 se muestra un ejemplo: [8]

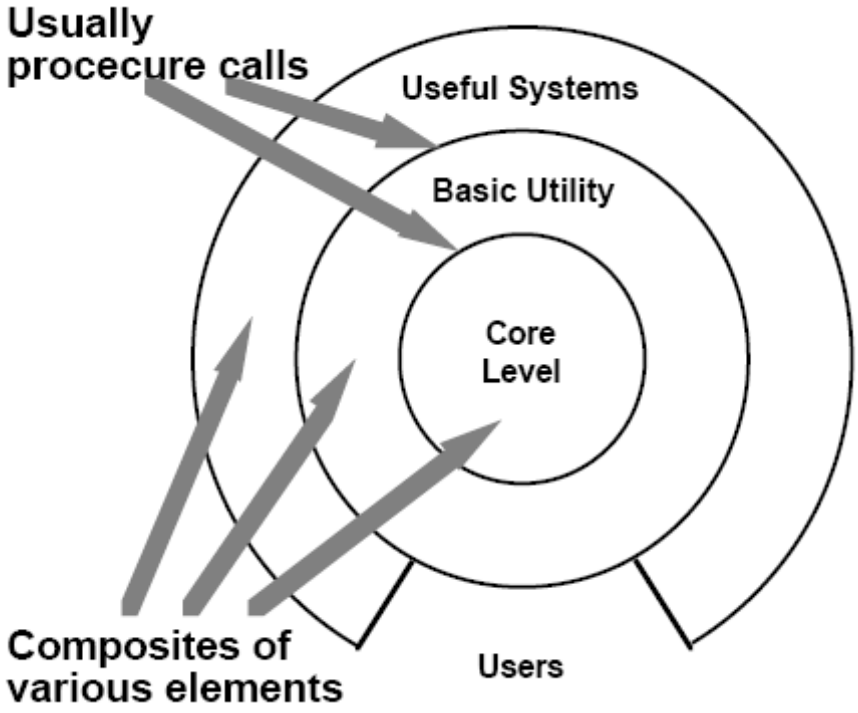


Figura 4. Sistemas en Capas. [8]

2.4.3.1.2 Patrón de Tuberías y Filtros

Proporcionan una estructura para los sistemas que procesan un flujo de datos. Cada paso de procesamiento se encapsula en un componente filtro. Los datos se pasan a través de tuberías entre filtros adyacentes. Recombinando los filtros, se permite construir familias de sistemas relacionados. En otras palabras, los componentes leen flujos o tramas de datos en sus entradas y producen flujos de datos en sus salidas, entregando una instancia completa del resultado en un orden estándar. Se aplica una trasformación local a los flujos de entrada y se computan incrementalmente de forma que la salida comienza antes de que la entrada se consuma. Los componentes son filtros y los conectores son tuberías. En la Figura 5 se muestra un ejemplo: [8]

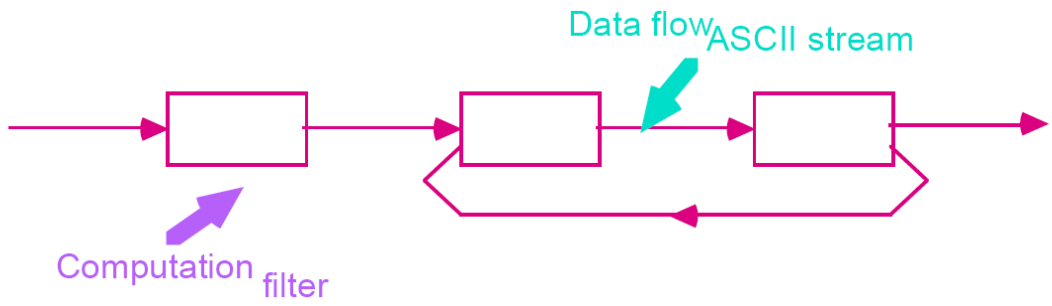


Figura 5. Tuberías y Filtros. [8]

2.4.3.1.3 El Patrón Pizarra o *Blackboard*

Es útil para problemas para los cuales no se conocen estrategias de solución determinísticas. En este patrón pizarra varios subsistemas especializados ensamblan su conocimiento para construir soluciones parciales o aproximadas.

2.4.3.2 Abstracción de Datos y Organización Orientada a Objetos [8]

En este estilo las representaciones de los datos y sus operaciones primitivas asociadas están encapsulados en tipos de datos abstractos u objetos. Los componentes en este estilo son los objetos, que interactuar a través de invocaciones a funciones o procedimientos. Hay dos aspectos muy importantes a considerar en este estilo:

- Un objeto es responsable de preservar la integridad de su representación.
- La representación está oculta de los otros objetos.

En la Figura 6 se muestra este estilo de arquitectura: [8]

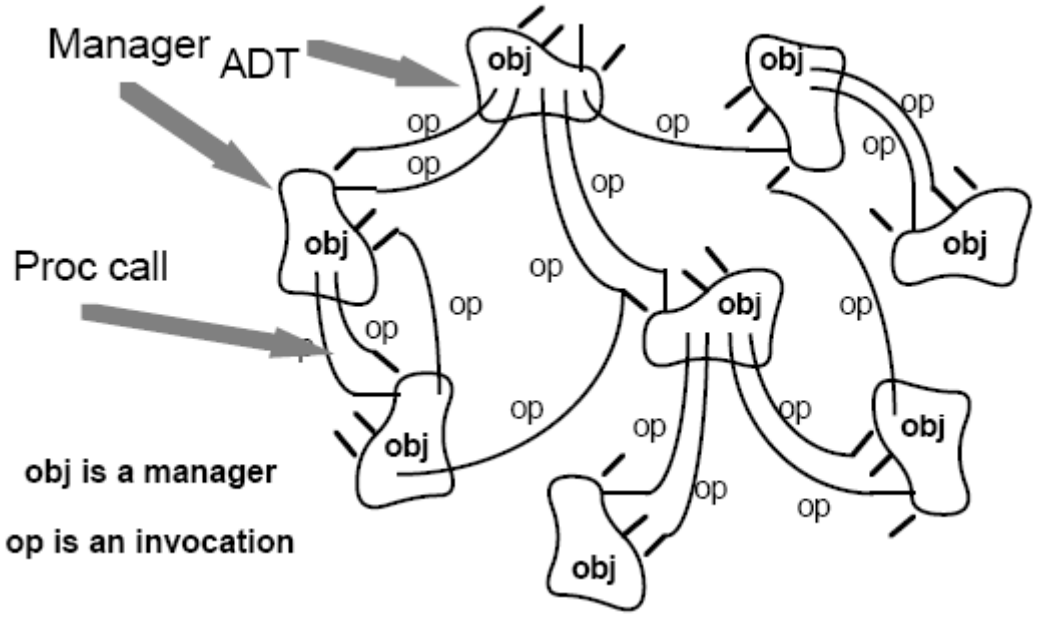


Figura 6. Objetos y tipos de datos abstractos. [8]

2.4.3.3 Invocación Implícita Basada en Eventos [8]

La idea de este estilo es que en vez de invocar un procedimiento directamente, un componente puede anunciar o difundir uno o más eventos. Otros componentes del sistema pueden registrar un interés en un evento mediante la asociación de un procedimiento con el evento. Cuando el evento es anunciado el sistema por sí mismo invoca todos los procedimientos que han sido registrados por el evento. Así, un anuncio de evento implícitamente provoca la invocación de los procedimientos en otros módulos.

2.4.3.4 Intérpretes Dirigidos por Tablas [8]

En una organización de intérpretes se fabrica una máquina virtual en software. Un intérprete incluye el pseudo-programa que es interpretado y el motor de interpretación. El pseudo-programa incluye el programa en sí mismo y el estado de ejecución del intérprete. El motor de interpretación incluye tanto la definición del intérprete como el estado actual de su ejecución. Así, un intérprete generalmente tiene cuatro componentes: un motor de interpretación para hacer el trabajo, una memoria que contiene el pseudo-código a ser interpretado, una representación del estado de control del motor de interpretación, y una representación del estado actual del programa que está siendo simulado. En la Figura 7 se muestra la descripción de un intérprete: [8]

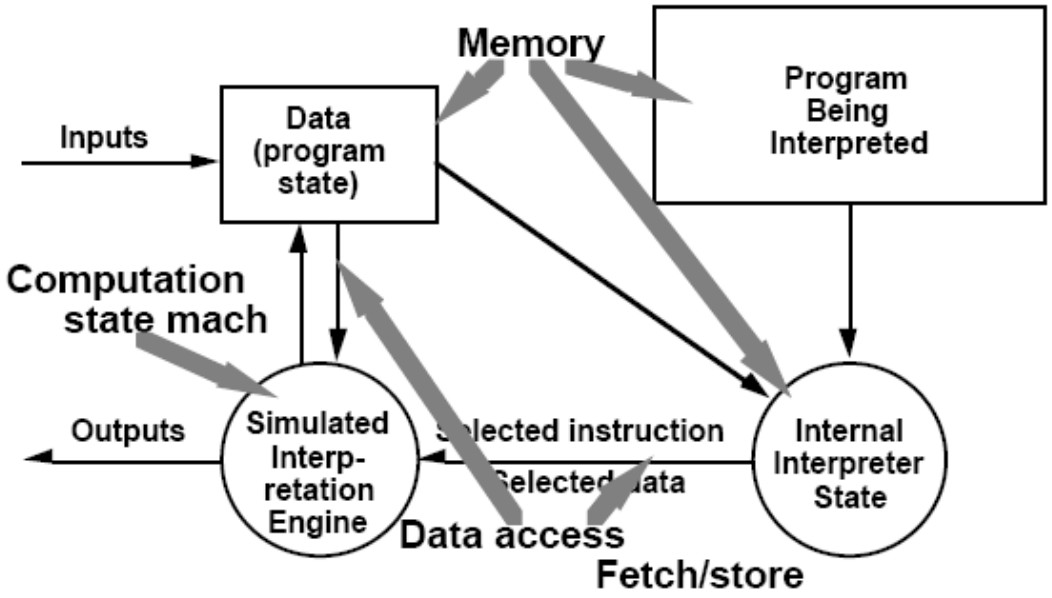


Figura 7. Intérprete. [8]

Los intérpretes se usan comúnmente para construir máquinas virtuales que cierran el hueco entre el motor de computación esperado por las semánticas del programa y el motor de computación disponible en hardware.

2.4.3.5 Sistemas Distribuidos [7]

Dentro de este patrón es de destacar el llamado “Broker”, que puede usarse para estructurar sistemas de software distribuidos con componentes desacoplados que interactúan mediante invocaciones de servicio remoto. Un componente bróker es responsable de coordinar la

comunicación, como por ejemplo reenviando peticiones, así como transmitiendo resultados y excepciones.

2.4.3.6 Sistemas Interactivos [7]

El núcleo principal de estos sistemas está basado en los requisitos funcionales para el sistema, que usualmente permanecen estables. Sin embargo, las interfaces de usuario a menudo están sujetas a cambios y adaptaciones. Esto requiere una arquitectura que soporte la adaptación de las partes de la interfaz de usuario son causar efectos mayores en la funcionalidad específica de aplicación o en el modelo de datos subyacente al software. Dentro de estos patrones interactivos cabe destacar dos:

2.4.3.6.1 Patrón Modelo-Vista-Controlador (MVC)

Divide una aplicación interactiva en tres componentes. El modelo contiene la funcionalidad principal y los datos. La vista muestra información al usuario. El controlador maneja las entradas del usuario. Las vistas y el controlador constituyen conjuntamente la interfaz de usuario. Un mecanismo de propagación del cambio asegura consistencia entre la interfaz de usuario y el modelo.

2.4.3.6.2 Patrón Presentación-Abstracción-Control (PAC)

Define una estructura para sistemas software interactivos en forma de una jerarquía de agentes cooperantes. Cada agente es responsable de un aspecto específico de la funcionalidad de la aplicación y consta de tres componentes: presentación, abstracción y control. Esta subdivisión separa los aspectos de interacción humano-máquina del agente de su núcleo funcional y su comunicación con otros agentes.

2.4.3.7 Sistemas Adaptativos [7]

Patrones que soportan la extensión de aplicaciones y su adaptación a las tecnologías que evolucionan y a los requisitos funcionales cambiantes. Hay dos patrones principales:

2.4.3.7.1 Patrón Microkernel

Aplica a sistemas software que deben poderse adaptar a requisitos cambiantes del sistema. Separa un núcleo mínimo funcional de la funcionalidad extendida y de las partes específicas del cliente.

2.4.3.7.2 Patrón de Reflexión

Proporciona un mecanismo para cambiar estructura y comportamiento de sistemas software dinámicamente. Soporta la modificación de aspectos fundamentales, tales como estructuras tipo y mecanismos de llamada a funciones. En este patrón, una aplicación se divide en dos partes. Un metanivel proporciona información sobre las propiedades seleccionadas del sistema y hace al software consciente de sí mismo. Un nivel base incluye la lógica de aplicación. Su implementación se construye en el metanivel. Los cambios a la información que se conservan en el metanivel afectan al comportamiento de subsiguientes niveles base.

3 CAPÍTULO 3 – ATRIBUTOS DE CALIDAD EN ARQUITECTURAS SOFTWARE

3.1 Atributos de Calidad

Siguiendo la definición dada en [4], los requerimientos de atributos de calidad deben ser específicos sobre cómo una aplicación debería de alcanzar una determinada necesidad, es decir, cómo se alcanzan los requisitos funcionales de una aplicación.

Se tiene por tanto que tener en consideración dichos atributos de calidad, no para alcanzar unos determinados requisitos funcionales, sino alcanzarlos atendiendo a una serie de parámetros que medirán cómo de bueno es el sistema. En [5] se afirma que para alcanzar dichos parámetros de calidad, se deben de considerar durante el diseño, implementación y despliegue. Para [5], hay tres clases de atributos de calidad:

- Calidades del sistema: hay una serie de parámetros que medirán cómo de bueno es el sistema, con respecto a: disponibilidad, modificabilidad, rendimiento, seguridad, testabilidad, usabilidad, escalabilidad, integración y portabilidad.
- Calidades del negocio: por ejemplo, *time to market*. Dichas calidades se ven afectadas por la arquitectura
- Calidades relacionadas con la arquitectura en sí misma

3.1.1 Calidades del Sistema

Según [5], hay tres problemas fundamentales a la hora de abordar los atributos de calidad del sistema:

- Las definiciones proporcionadas para un atributo de calidad no son operacionales
- Es difícil a menudo clasificar los atributos de calidad en relación a los diferentes aspectos funcionales del sistema
- Se manejan diferentes definiciones en la comunidad software para los atributos.

Para solventar el último problema, es necesario poner en común las diferentes visiones que se tienen sobre los atributos de calidad. Para solventar los dos primeros problemas, se recurre a lo que se ha denominado escenarios de atributos de calidad.

3.1.1.1 Escenarios de Atributos de Calidad

Un escenario de atributo de calidad es un requisito específico de un atributo de calidad, que consta de seis partes: [5]

Parte	Definición
Fuente del estímulo	Alguna entidad (humano, sistema computador, otro actuador) que ha generado el estímulo.
Estímulo	Es una condición que necesita ser considerada cuando llega al sistema.
Entorno	El estímulo ocurre en ciertas condiciones. El sistema puede estar por ejemplo en una condición de sobrecarga o puede estar ejecutándose cuando el estímulo ocurre, o alguna condición puede ser verdadera.
Artefacto	Es la parte del sistema que es estimulada. Puede ser la totalidad del sistema.
Respuesta	Es la actividad que se ejecuta después de la llegada del estímulo.
Medida de la respuesta	Cuando la respuesta ocurre, debería ser medible de alguna manera de forma que el requisito pueda ser testeado.

Tabla 1. Definición de Escenarios de Atributos de Calidad.

Se distinguen en escenarios de atributos de calidad generales (que son independientes del sistema) de los concretos (que son específicos de un sistema particular en consideración).

En la Figura 8 se muestran las diferentes partes de un escenario de atributo de calidad: [5]

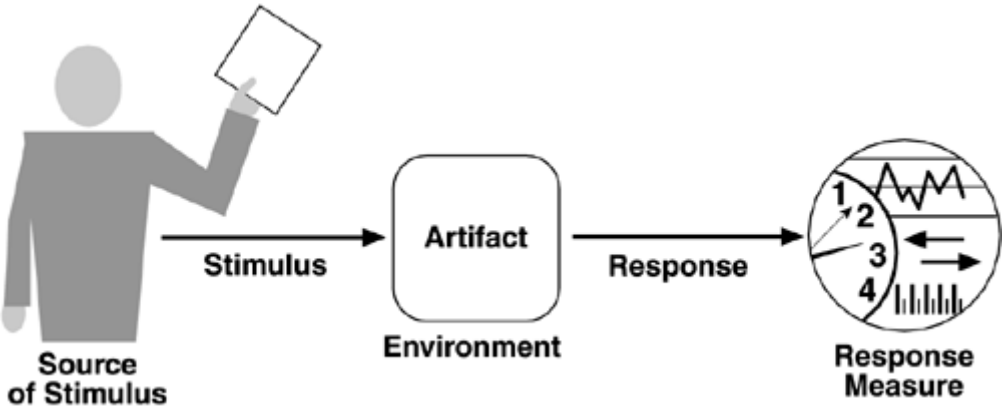


Figura 8. Partes del Atributo de Calidad. [5]

3.1.1.2 Descripción General de Atributos de Calidad

Según [4] y [5], los atributos de calidad se clasifican en diferentes categorías, descritas brevemente a continuación:

3.1.1.2.1 Disponibilidad

La disponibilidad está asociada al hecho de que un sistema puede fallar y provocar consecuencias asociadas al fallo. Un fallo del sistema ocurre cuando el sistema no proporciona más un servicio consistente con su especificación. Tal fallo es observable por los usuarios del sistema.

Las diferentes áreas a considerar en el atributo disponibilidad son: cómo un fallo del sistema se puede detectar, con qué frecuencia puede ocurrir, qué ocurre después del fallo, cuánto tiempo se permite al sistema estar fuera de servicio, qué consecuencias para la seguridad tiene un fallo, cómo se pueden prevenir los fallos y qué clase de alertas se requieren después de un fallo.

3.1.1.2.2 Modificabilidad

La modificabilidad está relacionada con el coste del cambio, en base a dos conceptos:

- ¿Qué puede el artefacto cambiar?: las funciones del sistema, la plataforma (hardware, sistema operativo…), el entorno dentro del cual el sistema opera, las capacidades del sistema, etc.
- ¿Cuándo se hace el cambio y quien lo hace? Por ejemplo, si se hace durante la implementación (cambiando el código fuente), durante la compilación, durante la configuración, ejecución, etc. También quien lo hace puede ser: desarrollador, usuario final, administrador del sistema, etc…

Un ejemplo de escenario de atributo de calidad – modificabilidad se muestra en la Figura 9: [5]

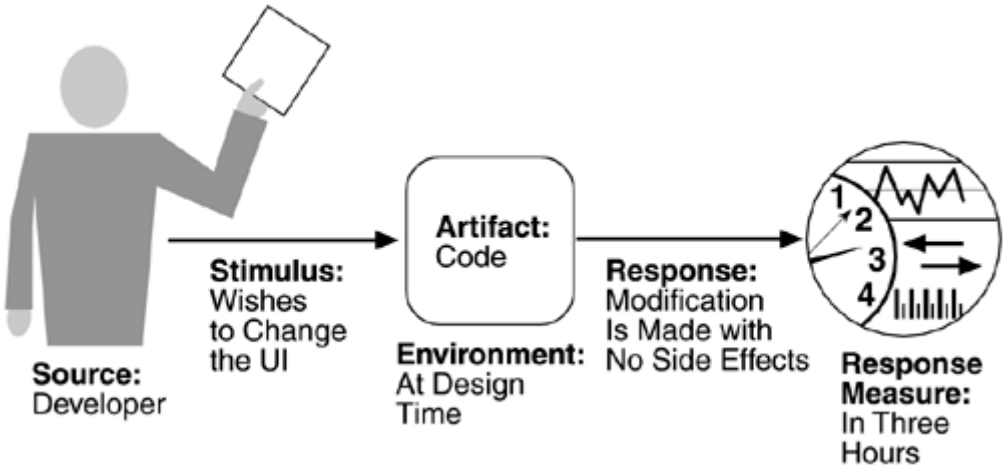


Figura 9. Escenario de Atributo de Calidad – Modificabilidad. [5]

3.1.1.2.3 Rendimiento

El rendimiento está relacionado con el tiempo. El atributo de calidad de rendimiento define una métrica que declara la cantidad de trabajo que una aplicación debe realizar en un tiempo dado, y el tiempo límite que debe ser cumplida para una correcta operación. Es decir, este atributo mide la respuesta del sistema frente a eventos que llegan y a los que tiene que responder en un determinado tiempo asignado.

Un ejemplo de escenario de atributo de calidad – rendimiento se muestra en la Figura 10: [5]

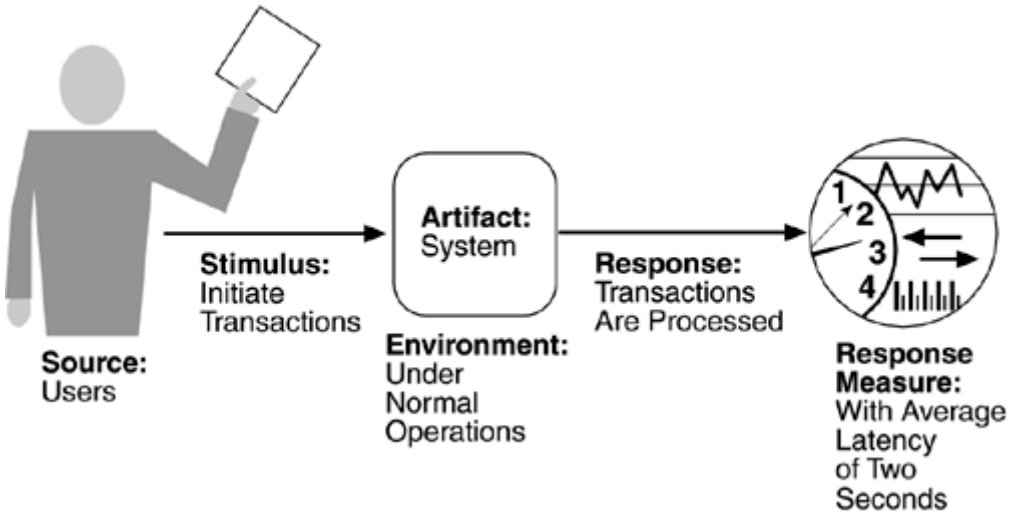


Figura 10. Escenario de Atributo de Calidad – Rendimiento. [5]

3.1.1.2.4 Seguridad

La seguridad es una medida de la capacidad del sistema para resistir a usos no autorizados mientras sigue proporcionando sus servicios a los usuarios legítimos. Un intento de romper la seguridad se denomina ataque y puede tomar un gran número de formas. Puede ser intentos no autorizados de acceder a datos o servicios o modificar datos, o puede intentar denegar servicios a usuarios legítimos.

3.1.1.2.5 Testabilidad

La testabilidad del software se refiere a la facilidad con la cual el software se puede hacer para demonstrar sus fallos a través de pruebas. Es cómo de fácil o difícil es de testear una aplicación. Las decisiones de diseño tempranas pueden afectar de manera significativa la cantidad de casos de test que se requieren. Como regla, cuanto más complejo es un diseño, más difícil es de testear. Por ejemplo, si se introducen componentes pretesteados en lugar de código propio, se reduce el esfuerzo de test.

3.1.1.2.6 Usabilidad

Está relacionada con cómo de fácil es para el usuario acometer una determinada tarea y la clase de soporte al usuario que proporciona el sistema. Se puede descomponer en las siguientes áreas:

- Aprendizaje de características del sistema.
- Usando un sistema de manera eficiente
- Minimizando los impactos de errores.
- Adaptando el sistema a las necesidades del usuario
- Incrementando la confianza y satisfacción.

3.1.1.2.7 Escalabilidad

La escalabilidad es cómo de bien una solución a un determinado problema trabajará cuando el tamaño del problema crezca. Es decir, cómo un diseño puede hacer frente a algunos aspectos de los requisitos de la aplicación cuando éstos se incrementen en tamaño.

3.1.1.2.8 Integración

La integración está relacionada con la facilidad con la cual una aplicación puede ser incorporada de manera útil dentro de un contexto de aplicación más amplio. El valor de una aplicación o componente puede ser frecuentemente incrementada de manera significativa si su funcionalidad o datos pueden ser usados en formas que el diseñador no había anticipado al comienzo.

3.1.1.2.9 Portabilidad

Responde a la pregunta de: ¿puede una aplicación ser ejecutada fácilmente en una plataforma hardware/software diferente de aquella para la cual fue desarrollada inicialmente? La portabilidad depende de las elecciones de la tecnología software usada para implementar la aplicación, y de las características de las plataformas en las cuales se necesita ser ejecutada.

3.1.2 Calidades del Negocio

Según [5], hay algunas metas de calidad de negocios que frecuentemente se han de adaptar a la arquitectura del sistema. Entre otras, se pueden considerar las siguientes:

- Tiempo hasta el mercado (*Time to market*).
- Coste y beneficio.
- Tiempo de vida proyectado del sistema.
- Mercado objetivo.
- Calendario de lanzamiento.
- Integración con sistemas heredados.

3.1.3 Calidades de la Arquitectura

Son calidades relacionadas directamente con la arquitectura en sí, y que son importantes de alcanzar. En [5] se consideran tres:

- Integridad conceptual: es el tema adyacente o la visión que unifica el diseño del sistema a todos los niveles. La arquitectura debería de hacer cosas similares de maneras similares.
- Correcta y completa: características de calidad fundamentales para una arquitectura, para permitir a todos los requisitos del sistema y restricciones de recursos de tiempo de ejecución ser cumplidos.
- Constructibilidad: permite al sistema ser completado por el equipo disponible en un tiempo determinado y estar abierto a ciertos cambios según los progresos del desarrollo. Se refiere a la facilidad de construir el sistema deseado y se alcanza de manera arquitectónica poniendo atención a la descomposición en módulos, asignando dichos módulos de manera juiciosa a los equipos de desarrollo, y limitando las dependencias entre módulos (y por tanto entre equipos). La meta es maximizar el paralelismo que se puede dar en el desarrollo.

3.2 Alcanzando la Calidad a través de Tácticas

En el apartado anterior se vio una colección de atributos de calidad y escenarios, pero no se vio cómo alcanzarlos. Para alcanzar dichos atributos de calidad, se recurre a las tácticas.

Las tácticas, según [5], son decisiones de diseño encaminadas a alcanzar atributos de calidad. Una táctica es una decisión de diseño que influye sobre el control de la respuesta de un atributo de calidad. Una colección de tácticas constituye lo que se denomina una estrategia arquitectónica.

En la Figura 11 se muestra la relación entre el estímulo y la respuesta a través de una táctica:

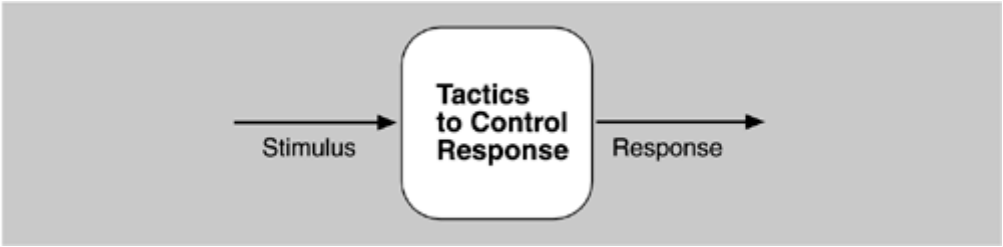


Figura 11. Las Tácticas intentan controlar las respuestas a los Estímulos. [5]

Cada táctica es una opción de diseño para el arquitecto, pero también hay que considerar que:

- Las tácticas pueden refinar otras tácticas.
- Los patrones empaquetan tácticas.

En las siguientes secciones se resumen las tácticas para cada atributo de calidad, según lo definido en [5].

3.2.1 Tácticas de Disponibilidad

El objetivo de las tácticas de disponibilidad se muestra en la Figura 12:

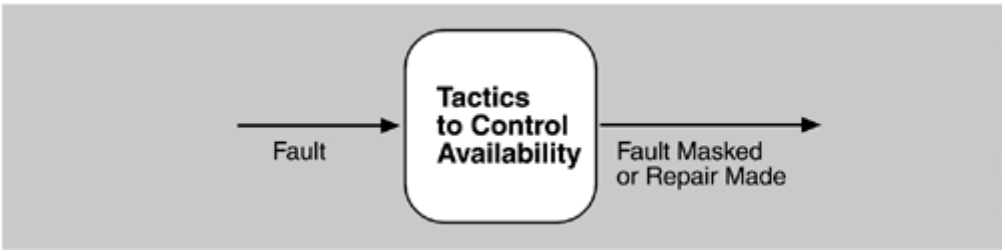


Figura 12. Objetivo de las tácticas de disponibilidad. [5]

Las tácticas utilizadas para disponibilidad se muestran en la Figura 13:

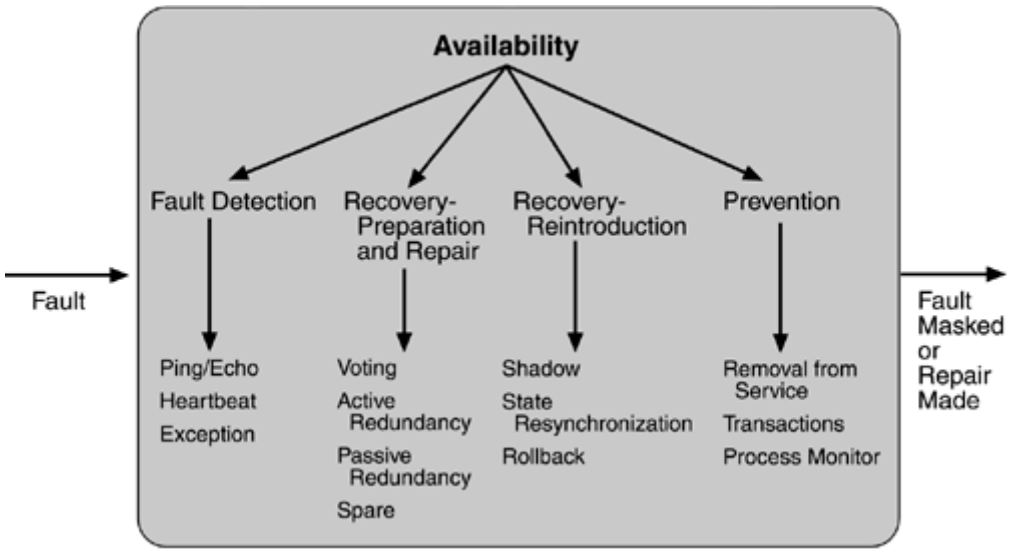


Figura 13. Resumen de las tácticas de disponibilidad. [5]

3.2.1.1 Detección de Fallos

Se resumen en tres tácticas:

- Ping/echo: un componente manda un ping y espera recibir un eco dentro de un tiempo predefinido desde el componente bajo escrutinio.
- Latido: un componente emite un mensaje "latido" periódicamente y otro componente lo escucha, si el latido falla, se supone que el componente que ya no late ha fallado.
- Excepciones: cuando se reconoce uno de los tipos de fallo que se puedan producir.

3.2.1.2 Recuperación de Fallos

- Votación: varios procesos se ejecutan a la vez en procesadores redundantes, cada uno de los cuales toman entradas equivalentes y computan un único valor de salida que se envía a un votante. Si el votante detecta comportamientos desviados de uno de los procesadores, se asume que está fallado.
- Redundancia activa: existen múltiples componentes redundantes trabajando en el mismo estado y uno sólo se utiliza. Si ese falla, se cambia a uno de los que funcionan en pocos milisegundos.
- Redundancia pasiva: un componente primario responde a eventos e informa a los otros componentes (en *standby*) de las actualizaciones de estado que deben de hacer. Si el primario falla, se cambia a uno en *standby* asegurándose de que el estado está actualizado.
- Repuesto: una plataforma de computación de repuesto en *standby* se configura para reemplazar muchos componentes fallados diferentes.
- Operación en la sombra: un componente que ha fallado previamente puede ser ejecutado "en la sombra" para comprobar que imita el comportamiento de los demás componentes, antes de restaurarlo al servicio
- Resincronización del estado: consiste en un mensaje único conteniendo el estado del componente que se ha de restaurar en las tácticas de redundancia activa y pasiva, con incrementos graduales del estado, para simplificar la restauración.

- *Checkpoint/rollback*: un punto de chequeo es una grabación de un estado consistente creado periódicamente o en respuesta a eventos específicos.

3.2.1.3 Prevención de Fallos

Se resumen en las siguientes tácticas:

- Eliminación del servicio: la táctica elimina un componente del sistema de la operación para impedir y anticipar posibles fallos.
- Transacciones: usadas para prevenir que cualquier data se vea afectado si un paso en un proceso falla. Es un lote de sucesivos pasos secuenciales de manera que se pueda deshacer el lote completo al instante.
- Monitor de procesos: cuando se detecta un fallo en un proceso, el monitor puede eliminarlo y crear una nueva instancia de él.

3.2.2 Tácticas de Modificabilidad

El objetivo de las tácticas de modificabilidad se muestra en la Figura 14:

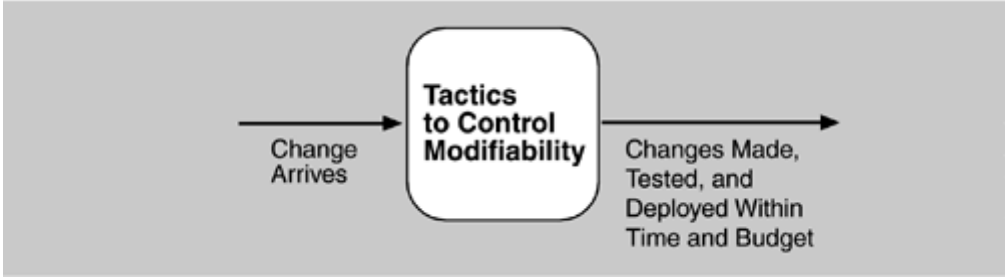


Figura 14. Objetivo de las Tácticas de Modificabilidad. [5]

Las tácticas utilizadas para modificabilidad se muestran en la Figura 15:

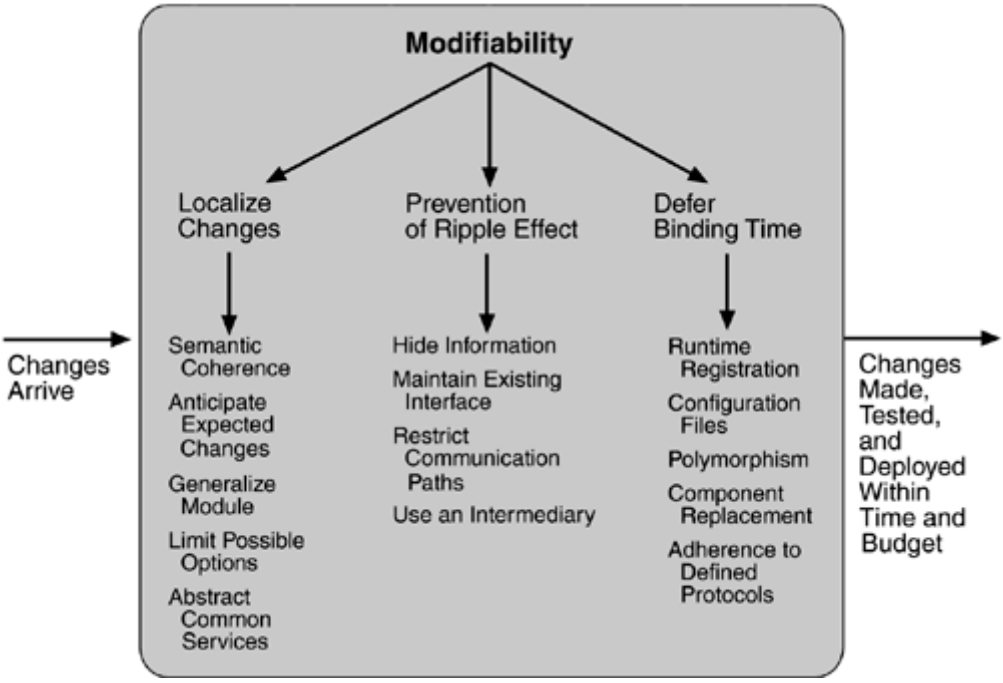


Figura 15. Resumen de las Tácticas de Modificabilidad. [5]

3.2.2.1 Localizar Cambios/Modificaciones

Se identifican cinco tácticas:

- Mantener coherencia semántica: se refiere a las relaciones entre responsabilidades en un módulo; el objetivo es asegurar que todas estas responsabilidades trabajen juntas sin una excesiva dependencia en otros módulos.
- Anticipar cambios esperados: relacionada con minimizar los efectos de los cambios, y usualmente utilizada junto con la coherencia semántica.
- Generalizar el módulo: hacer un módulo más general le permite calcular un mayor amplio rango de funciones basadas en la entrada.
- Limitar posibles opciones: restringir las posibles opciones reducirá el efecto de las modificaciones.

3.2.2.2 Prevenir Efectos de Propagación

Un efecto de propagación de una modificación es la necesidad de hacer cambios a módulos no directamente afectados por la misma.

Las tácticas para prevenir efectos de propagación serán:

- Ocultar información: ocultar información es la descomposición de las responsabilidades de una entidad en trozos más pequeños y elegir qué información ha de ser privada y cual pública. La información pública está disponible a través de interfaces especificadas.
- Mantener interfaces existentes: si un modulo depende del nombre y firma d otro, si se mantiene la interfaz y la sintaxis de este último se mantendrá al primero invariable.
- Restringir caminos de comunicación: restringir los módulos con los cuales un modulo dado comparte datos.
- Usar un intermediario: si un modulo B tiene una dependencia de A diferente a la semántica, es posible insertar un intermediario entre B y A que gestione las actividades asociadas con la dependencia.

3.2.2.3 Diferir "binding time"

"*Binding*" en tiempo de ejecución significa que el sistema ha sido preparado para el enlazado o *linking* (es decir, desde que se hace un cambio hasta que se despliega, se testea, se distribuye) y que todos los pasos de testeado y distribución han sido completados.

Las tácticas asociadas serían:

- El registro del tiempo de ejecución soporta operaciones de *plug-and-play*.
- Los ficheros de configuración son parámetros en el arranque.
- El polimorfismo permite un *linking* atrasado de las llamadas a los métodos.
- El reemplazo de componentes permite cargar el tiempo de *linking.*
- La adherencia a los protocolos definidos permite *linking* de tiempo de ejecución de procesos independientes.

3.2.3 Tácticas de Rendimiento

El objetivo de las tácticas de rendimiento se muestra en la Figura 16:

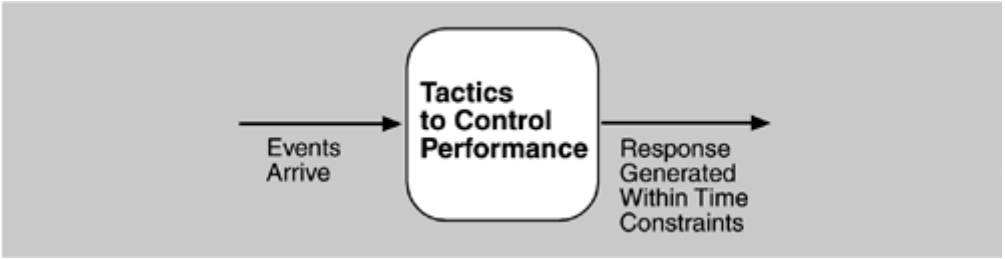


Figura 16. Objetivo de las Tácticas de Rendimiento. [5]

Las tácticas utilizadas para rendimiento se muestran en la Figura 17:

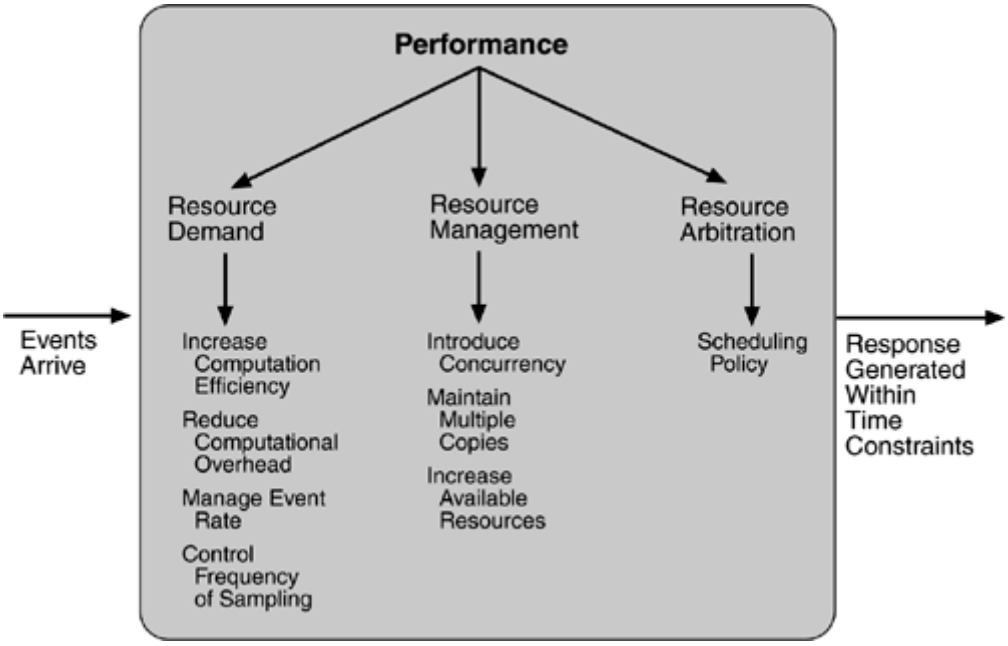


Figura 17. Resumen de las Tácticas de Rendimiento. [5]

Las tácticas se clasifican en tres grupos:

3.2.3.1 Demanda de Recursos

Una táctica para reducir la latencia es reducir los recursos requeridos para procesar una trama de eventos:

- Incrementar la eficiencia computacional: aplicando algún algoritmo mejorado.
- Reducir el sobrecoste computacional: reducir el uso de intermediarios.
- Gestionar tasa de eventos: reduciendo la frecuencia de muestreo a la cual las variables de entorno son monitorizadas.
- Controla la frecuencia de muestreo
- Limitar los tiempos de ejecución
- Limitar los tamaños de las colas.

3.2.3.2 Gestión de Recursos

Algunas tácticas de gestión de recursos son:

- Introducir concurrencia: procesando diferentes flujos de eventos en diferentes hilos.
- Mantener múltiples copias de datos o cálculos: el propósito de las replicas es reducir la contención que ocurriría si todos los cálculos tienen lugar en un servidor central.
- Incrementar los recursos disponibles: procesadores más rápidos, incrementar numero de procesadores, redes más rápidas, etc., reducen la latencia.

3.2.3.3 Arbitraje de Recursos

Establecer una política de planificación, en la cual se asignará primeramente una prioridad, y a continuación se despacharan las tareas.

Dentro de las políticas de planificación cabe destacar:

- FIFO (First-in/First-out)
- Planificación de prioridad fijada
- Planificación de prioridad dinámica
- Planificación estática.

3.2.4 Tácticas de Seguridad

El objetivo de las tácticas de seguridad se muestra en la Figura 18:

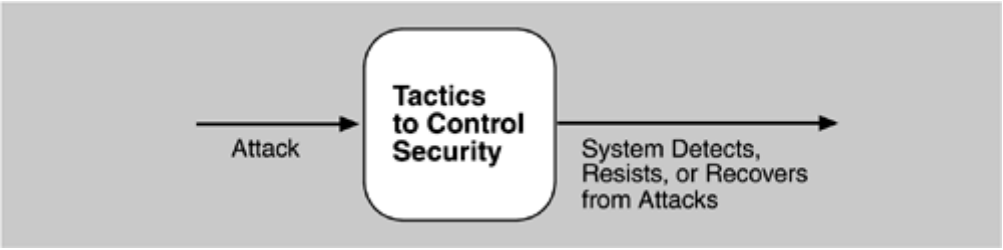


Figura 18. Objetivo de las Tácticas de Seguridad. [5]

Las tácticas utilizadas para seguridad se muestran en la Figura 19:

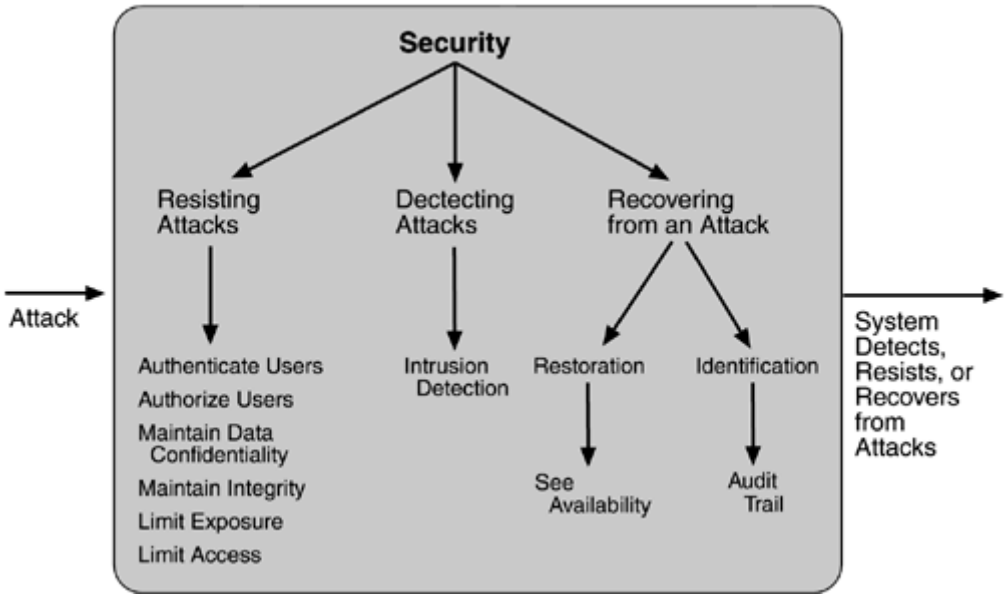


Figura 19. Resumen de las Tácticas de Seguridad. [5]

Las tácticas se agrupan en 3 categorías:

3.2.4.1 *Resistir Ataques*

Algunas tácticas serían:

- Autenticar usuarios
- Autorizar usuarios
- Mantener la confidencialidad de los datos
- Mantener integridad
- Limitar la exposición, entendiéndose por asignar los servicios a los hosts de forma que haya servicios limitados en cada host.
- Limitar el acceso, por medio de firewalls

3.2.4.2 *Detectar Ataques*

A través de un sistema de detección de intrusos, como por ejemplo, comparando el patrón de tráfico de red en una base de datos.

3.2.4.3 *Recuperación Después de un Ataque*

Las tácticas que se usan son la restauración para volver al estado anterior, y mantener un rastro de auditoría, que es una copia de cada transacción aplicada a los datos en el sistema además de identificar la información.

3.2.5 Tácticas de Testabilidad

El objetivo de las tácticas de testabilidad se muestra en la Figura 20:

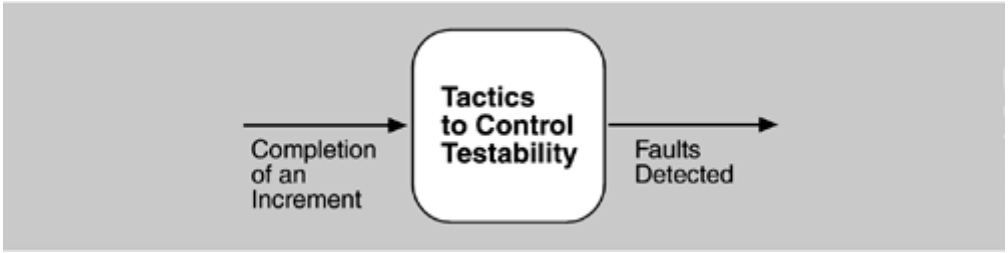


Figura 20. Objetivo de las Tácticas de Testabilidad. [5]

Las tácticas utilizadas para testabilidad se muestran en la Figura 21:

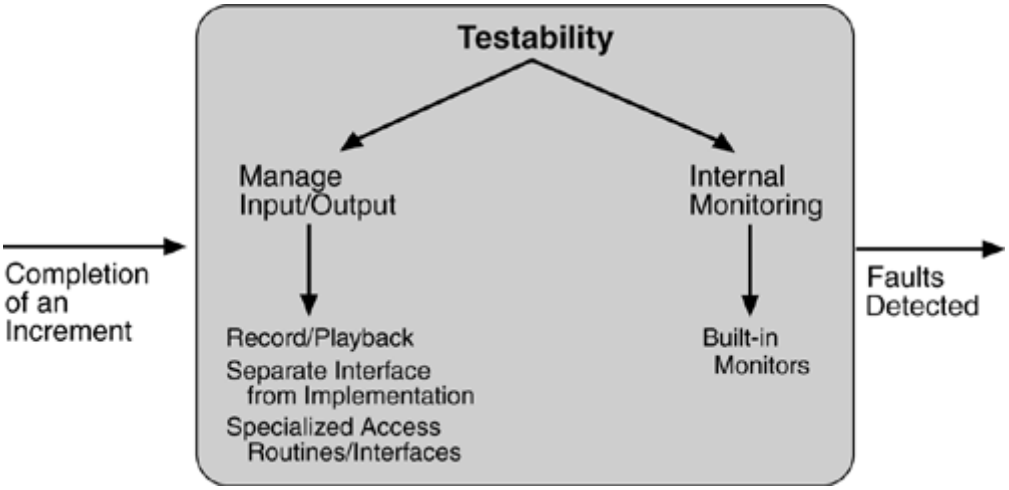


Figura 21. Resumen de las Tácticas de Testabilidad. [5]

Las tácticas usadas se agrupan en dos categorías:

3.2.5.1 Gestionar Entradas/Salidas

Hay tres tácticas:

- Grabación/reproducción: se refiere a capturar información que cruza por una interfaz y usándola dentro del armazón de los tests.
- Separar la interfaz de la implementación: permite sustituir las implementaciones para varios propósitos de tests.
- Especializar interfaces/rutas de acceso: permite capturar o especificar los valores de las variables para un componente a través de un armazón de test de forma independiente de su ejecución normal.

3.2.5.2 Monitorización Interna

Basándose en su estado interno, un componente puede implementar tácticas para soportar el proceso de tests, a través de monitores *built-in*; a través de una interfaz el componente hace accesible información sobre su estado, carga, capacidad o seguridad.

3.2.6 Tácticas de Usabilidad

El objetivo de las tácticas de usabilidad se muestra en la Figura 22:

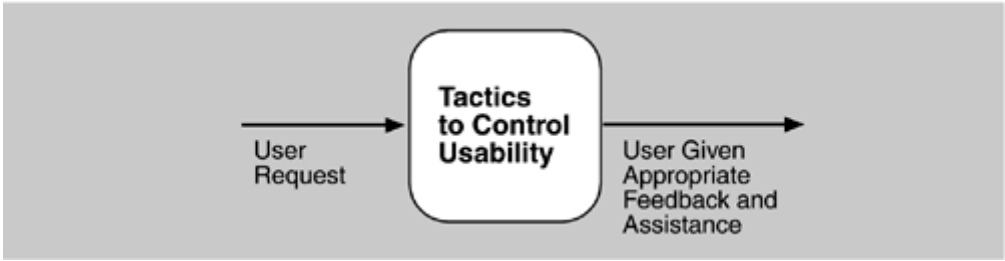


Figura 22. Objetivos de las Tácticas de Usabilidad. [5]

Las tácticas utilizadas para usabilidad se muestran en la Figura 23:

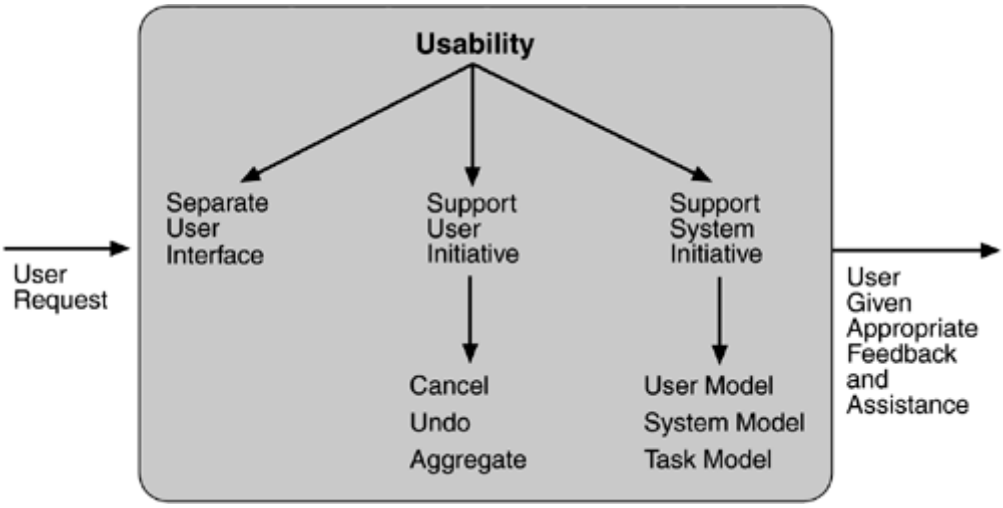


Figura 23. Resumen de las Tácticas de Usabilidad. [5]

Hay dos grupos diferentes de tácticas:

3.2.6.1 Tácticas en Tiempo de Ejecución

Una vez que el sistema se está ejecutando, la usabilidad se mejora dando al usuario un *feedback* de lo que el sistema está haciendo y proporcionando al usuario la habilidad para lanzar comandos basados en usabilidad como cancelar, deshacer, agregar o mostrar múltiples vistas.

Si es el sistema el que toma la iniciativa, debe de depender de cierta información (un modelo) sobre el usuario, la tarea que se está realizando o el estado del sistema en sí mismo. Esto da lugar a tres tácticas adicionales.

- Mantener un modelo de la tarea: para determinar el contexto para q el sistema pueda tener alguna idea de lo que el usuario quiere hacer y proporcionar así alguna guía
- Mantener un modelo del usuario: determina el conocimiento del usuario sobre el sistema, el comportamiento del usuario, etc.
- Mantener un modelo del sistema: determina el comportamiento esperado del sistema para así dar un *feedback* apropiado al usuario.

3.2.6.2 Tácticas en Tiempo de Diseño

Como las interfaces de usuario se revisan típicamente durante el proceso de test, se proponen varias tácticas que son un refinamiento de la táctica de modificabilidad de coherencia semántica:

- Separar la interfaz de usuario del resto de la aplicación: ya que la interfaz de usuario se espera que cambie muy a menudo durante el desarrollo y después de la implementación, mantener el código de la interfaz de usuario separado localizara los cambios en él. Los patrones de arquitectura software desarrollados para implementar esta táctica son:
  - Modelo-Vista-Controlador
  - Presentación-Abstracción-Control
  - Seeheim
  - Arch/Slinky

3.3 Diseño Dirigido por Atributos

Una vez vistos los atributos de calidad y las tácticas, en este punto es interesante cómo se pueden aplicar ambos al diseño de una arquitectura software. El SEI desarrolló la metodología *Attribute-Driven Design* (ADD) [13], que es un método pensado para definir una arquitectura software en la cual el proceso de diseño está basado en los requisitos de atributos de calidad del software. ADD sigue un proceso de diseño recursivo que descompone un sistema o elementos del sistema aplicando tácticas arquitectónicas y patrones que satisfacen los requisitos directores. Esencialmente, ADD sigue un ciclo "Planifica, Haz y Comprueba" o como más conocido es en inglés, "*Plan, Do and Check*":

- *Plan:* Se consideran los atributos de calidad y las restricciones de diseño para seleccionar qué tipos de elementos se usarán en la arquitectura
- *Do:* Se instancian elementos para satisfacer los requisitos de atributos de calidad así como los requisitos funcionales.
- *Check:* El diseño resultante se analiza para determinar si los requisitos se cumplen.

Este proceso se repite hasta que todos los requisitos arquitectónicamente significativos se cumplen.

En la Figura 24 y la Figura 25 se muestran el proceso que sigue este método:

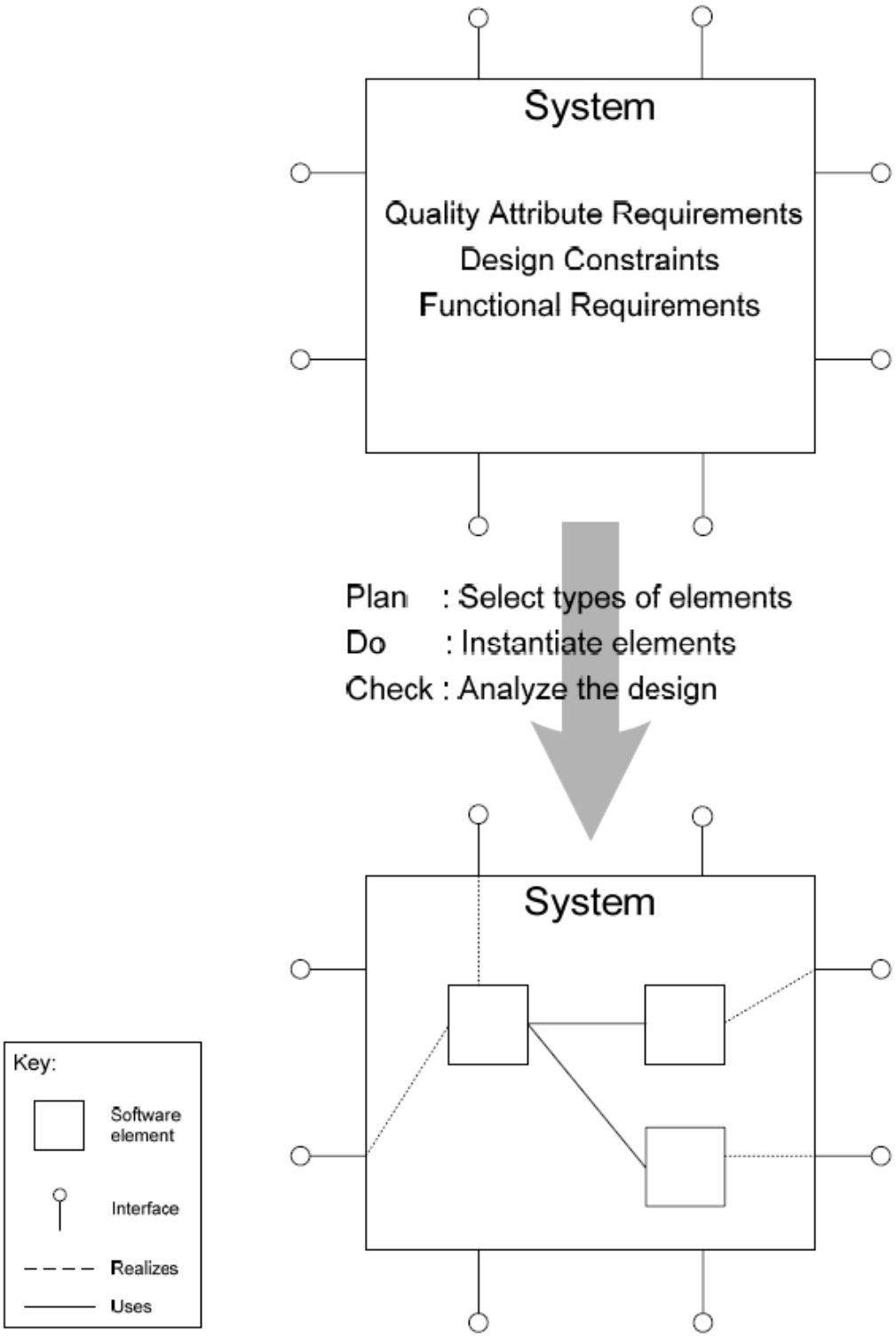


Figura 24. Metodología *Plan, Do & Check* aplicada al Diseño de Arquitectura Software. [13]

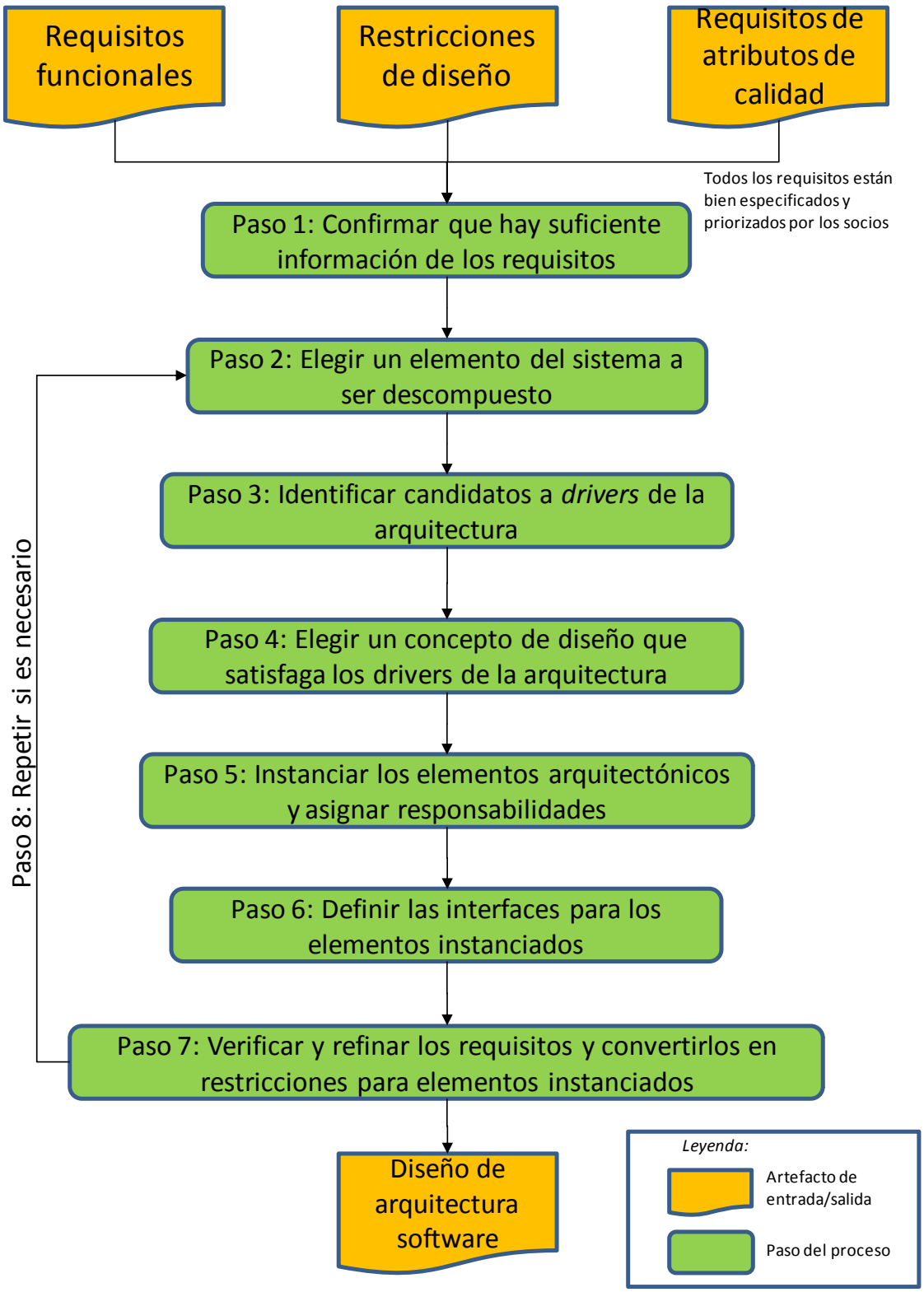


Figura 25. Pasos del Método ADD. [13]

4 CAPÍTULO 4 – MÉTODOS DE ANÁLISIS DE ARQUITECTURAS SOFTWARE

4.1 Introducción a la Metodología de Análisis de Arquitecturas

En los capítulos anteriores se ha presentado el concepto de arquitectura software y se han introducido los principios para realizar una evaluación de la misma, como los atributos de calidad y escenarios relacionados.

En este capítulo se van a presentar de manera breve algunos de los métodos utilizados en la evaluación de las arquitecturas software, cuyo objetivo es analizar si los escenarios de calidad han sido tenidos en cuenta a la hora de diseñar dicha arquitectura.

4.2 Clasificación de los Principales Métodos

En [9] se propone la siguiente clasificación:

- Métodos de evaluación de arquitecturas software basada en escenario

Esta es la categoría que se desarrollará en este capítulo. Entre los métodos utilizados, destacan: SAAM, ATAM, ALPSM and ALMA, SBAR, SALUTA, SAAMCS y ESAAMI.

- Evaluación de la Arquitectura Software basada en Modelos matemáticos.

Hay dos tipos principales:

  - Análisis de fiabilidad basado en la arquitectura software. Es un método que ayuda a identificar los componentes críticos del software y cuantificar su influencia en la fiabilidad del sistema. [11]
  - Análisis de rendimiento basado en la arquitectura software. Es un método que se basa en algoritmos matemáticos para determinar las prestaciones de una arquitectura software. Se basa generalmente en el modelo *layered queuing network*.

- Evaluación de la Arquitectura Software basada en Métricas

Hay dos métodos principales:

  - Evaluación temprana: QuADAI
  - Evaluación tardía: TLC, FA, MNS, SSC.

4.3 Perspectiva de los Principales Métodos de Análisis.

En [10] se presenta un resumen de los principales métodos de análisis de arquitecturas software, que son los siguientes:

4.3.1 Método de Análisis de Arquitectura Basado en Escenarios (SAAM)

El SAAM o *Scenario-Based Architecture Analysis Method* [10] tiene como objetivo verificar las hipótesis básicas y los principios de una arquitectura contra los documentos que describen sus propiedades deseadas de una aplicación, así como realizar una estimación de los riesgos.

La técnica de evaluación se basa en escenarios de atributos de calidad. SAAM evalúa un escenario investigando qué elementos arquitectónicos se ven afectados por dicho escenario.

En la Figura 26 se muestra un esquema con las entradas y las actividades de este método

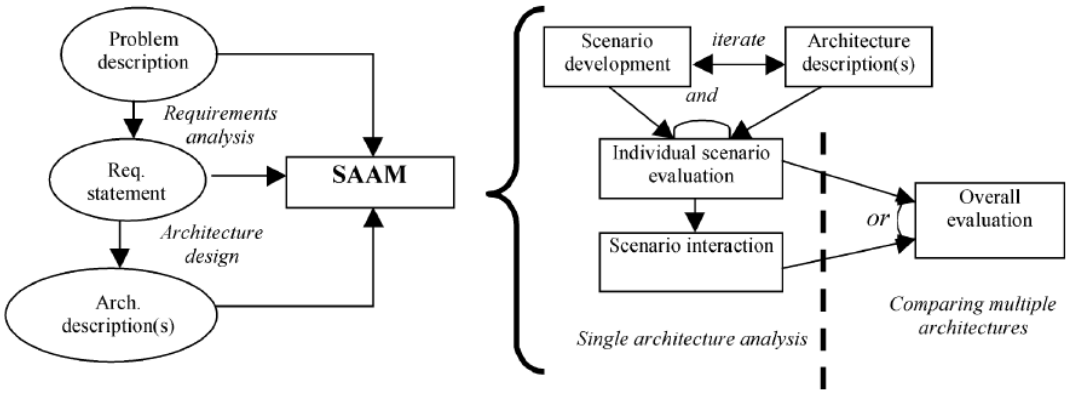


Figura 26. Entradas y Actividades del SAAM. [10]

4.3.2 SAAM Fundado en Escenarios Complejos (SAAMCS)

Método ampliado del anterior en el cual se considera que la complejidad de los escenarios es el factor de estimación de riesgos más importante. [10]

La técnica de evaluación busca escenarios que sean complejos de realizar. El atributo de calidad analizado por SAAMCS es la flexibilidad.

En la Figura 27 se muestran las entradas y actividades de SAAMCS.

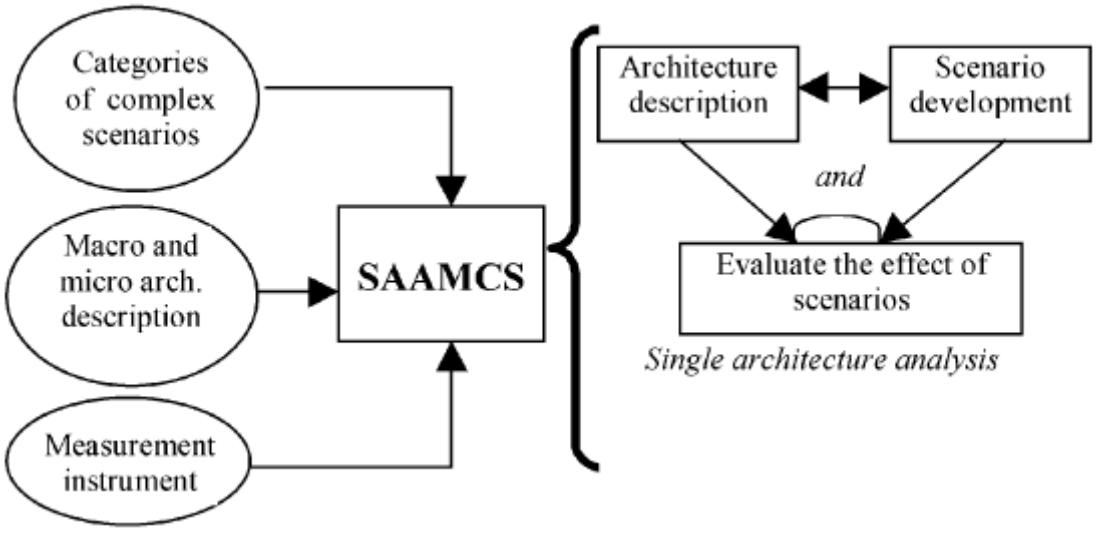


Figura 27. Entradas y Actividades de SAAMCS. [10]

4.3.3 Extendiendo SAAM Mediante la Integración en el Dominio (ESAAMI)

El objetivo principal de este método es considerar sólo la descripción del problema, declaración de requisitos y descripción de la arquitectura. [10]

ESAAMI es una combinación de los conceptos analíticos y de reutilización, y se alcanza integrando SAAM en el proceso de desarrollo específico de dominio y basado en el reutilización de la arquitectura, aplicada a un nuevo sistema.

En la Figura 28 se muestran las entradas a ESAAMI:

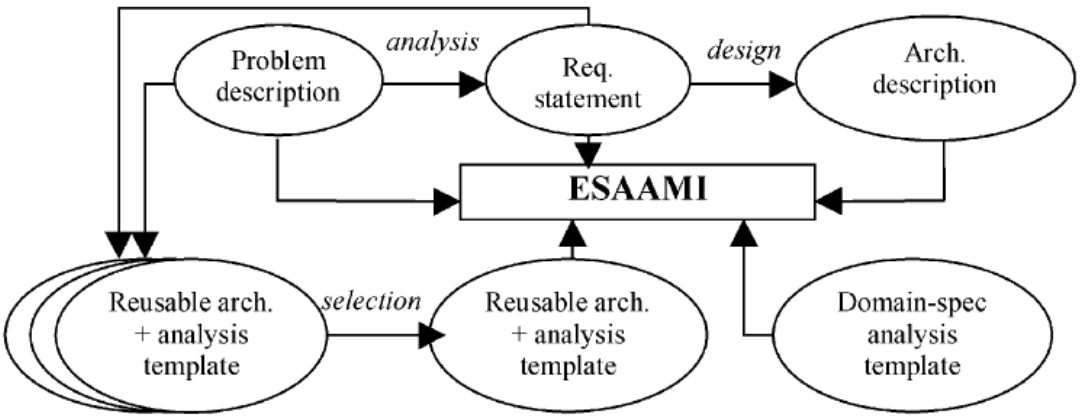


Figura 28. Entradas a ESAAMI. [10]

4.3.4 Método de Análisis de Arquitectura Software para Evolución y Reusabilidad (SAAMER)

El *Software Architecture Analysis Method for Evolution and Reusability* (SAAMER) [10] es una extensión de SAAM y tiene como objetivo analizar cómo un sistema podría soportar cada uno de los objetivos de calidad o niveles de riesgo en una evolución o una reutilización de la arquitectura. Por tanto, evolución y reusabilidad son los nuevos atributos de calidad.

SAAMER considera las siguientes vistas de la arquitectura como críticas: estática, mapa, dinámica y recursos. La vista estática integra y extiende SAAM para abordar la clasificación y generalización de los componentes de un sistema y las funciones y conexiones entre componentes. La vista dinámica es apropiada para la evaluación del aspecto de comportamiento, para validar que el control y la comunicación sean manejados de una determinada manera. El mapeo entre componentes y funciones podría revelar los aspectos de cohesión y acoplamiento del sistema.

4.3.5 Método de Análisis de Compensación de la Arquitectura

El método *Architecture Trade-Off Analysis Method* (ATAM) [10] fue desarrollado por el SEI (*Software Engineering Institute*), para estimar las consecuencias de las decisiones arquitectónicas considerando los requisitos de atributos de calidad y los objetivos de negocio. [12]. ATAM reconoce la necesidad de un acuerdo o compensación entre múltiples atributos de calidad software cuando la arquitectura software de un sistema se especifica y antes de que el sistema sea desarrollado [10].

Para describir la arquitectura, ATAM considera que el espacio de la arquitectura está restringido por los sistemas heredados, la interoperabilidad y los fallos de los proyectos previos. Además, ATAM requiere diversas vistas: una vista dinámica, que muestre cómo los sistemas se comunican; una vista de sistema, que muestra cómo el software se ha asignado al hardware; y una vista fuente, que muestra cómo los componentes y los sistemas están compuestos de objetos. Una serie de gráficos con mensajes de secuencias muestran las interacciones y escenarios en tiempo de ejecución.

ATAM utiliza escenarios como técnica de evaluación. Aparte de los escenarios basados en los típicos atributos de calidad, añade los siguientes: casos de uso, en los cuales se muestra típicos casos de uso del sistema; escenarios de crecimiento, que cobren cambios anticipadamente; y escenarios de exploración, que cubren cambios extremos que se esperan puedan estresar al sistema.

Las actividades del método se dividen en cuatro fases, como se muestra en la Figura 29: [10]

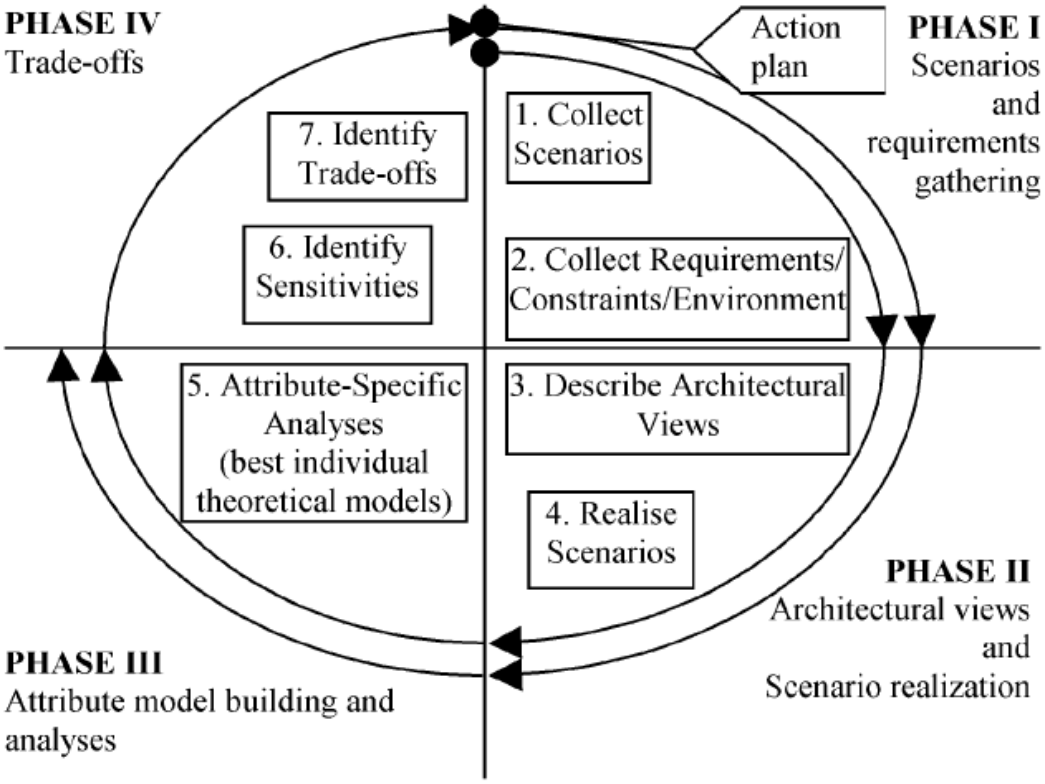


Figura 29. Fases de ATAM. [10]

El flujo conceptual de ATAM se muestra en la Figura 30: [12]

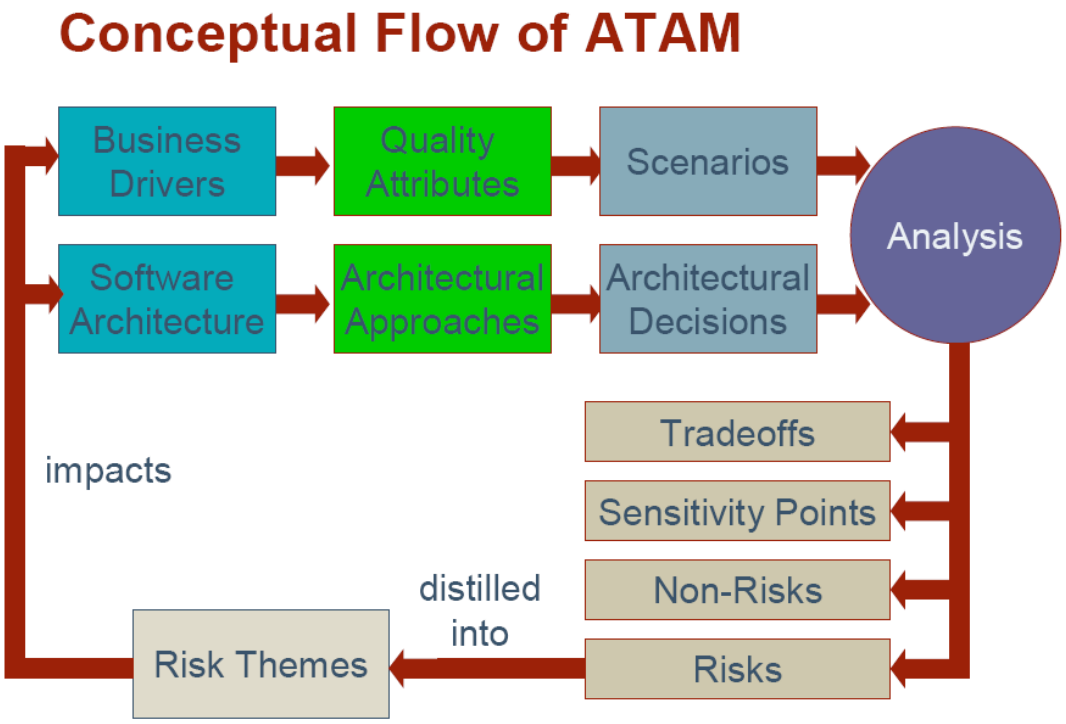


Figura 30. Flujo conceptual de ATAM. [12]

El método ATAM encajaría de la siguiente manera en un desarrollo basado en la arquitectura, tal y como explica [12]: (Figura 31)

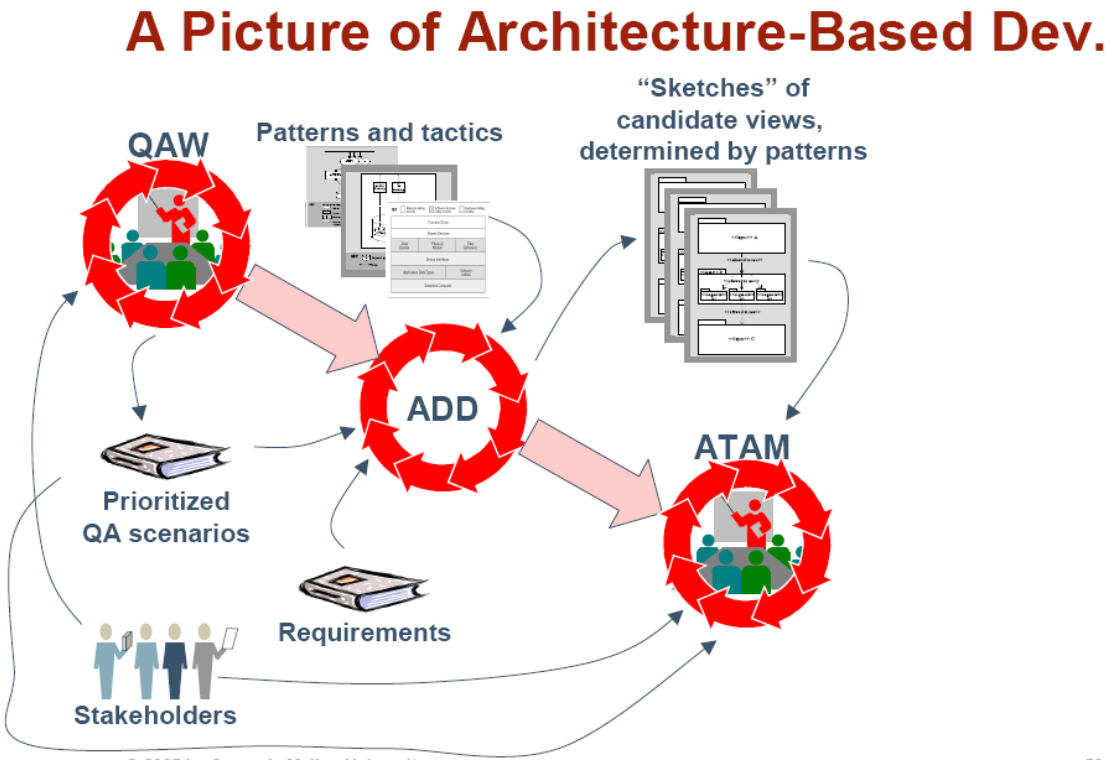


Figura 31. Proceso de Desarrollo Basado en la Arquitectura. [12].

En donde hay un ciclo de negocio de arquitectura, que ayuda a identificar los factores de casos de negocio, que ayudarán a dar forma a la arquitectura; un grupo de trabajo de atributos de calidad o *Quality Attribute Workshop* (QAW), que es el primer camino para implicar a los diferentes socios; los Escenarios de atributos de calidad o QA Scenarios, que es la forma de capturar los requisitos de atributos de calidad; y por último, el Diseño Dirigido por Atributos o *Atribute driven design* (ADD), que es un método para diseñar una arquitectura que cumpla sus requisitos funcionales y de atributos de calidad.

4.3.6 Reingeniería de la Arquitectura Basada en Escenarios (SBAR)

El *Scenario-Based Architecture Reengineering* o SBAR estima el potencial de la arquitectura para alcanzar los requisitos de calidad del software. Para ello, utiliza técnicas de escenarios, simulación, modelos matemáticos y razonamiento basado en la experiencia.

En la Figura 32 se muestran las actividades del SBAR: [10]

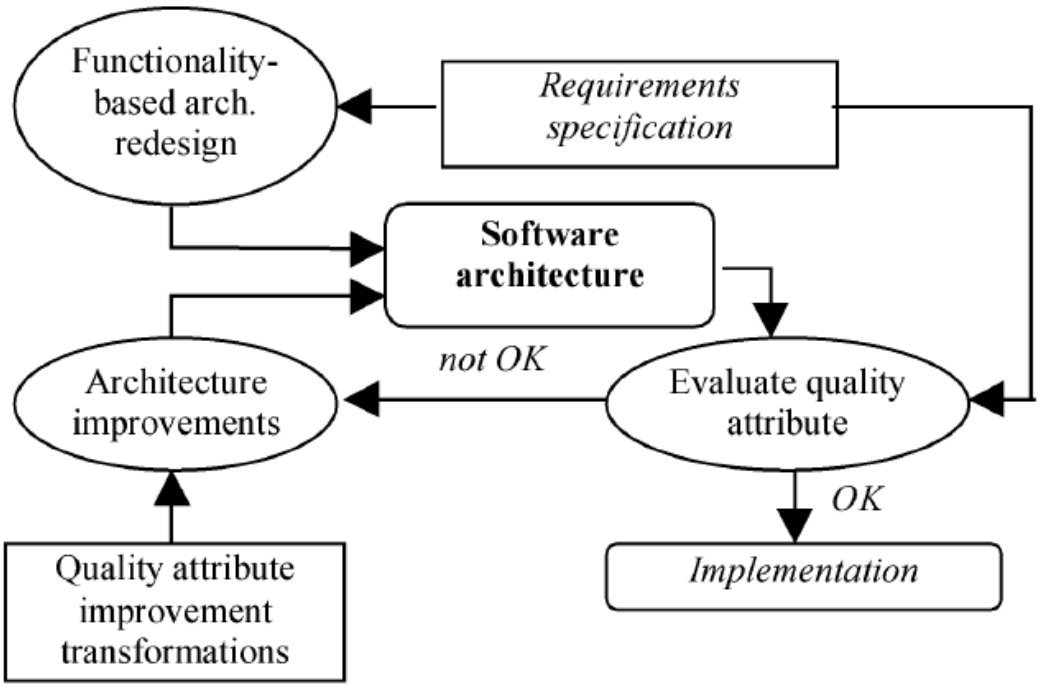


Figura 32. Actividades en SBAR. [10]

4.3.7 Predicción del Mantenimiento de Software a Nivel de Arquitectura (ALPSM)

El método *Architecture Level Prediction of Software Maintenance* o ALPSM [10] analiza la mantenibilidad de un sistema software mirando el impacto de los escenarios a nivel arquitectura software. Para ello, el método define un perfil de mantenimiento, como un conjunto de escenarios de cambio que representan tareas de mantenimiento adaptativas Por tanto, un escenario de cambio describe una cierta tarea de mantenimiento.

ALPSM consta de seis pasos fundamentales:

1. La identificación de las categorías de las tareas de mantenimiento
2. La síntesis de escenarios
3. La asignación de un peso a cada escenario
4. La estimación del tamaño de todos los elementos
5. Realizar el scripting (o guión) de los escenarios
6. Calcular el esfuerzo de mantenimiento previsto.

En la Figura 33 se muestra las entradas y el resultado de ALPSM: [10]

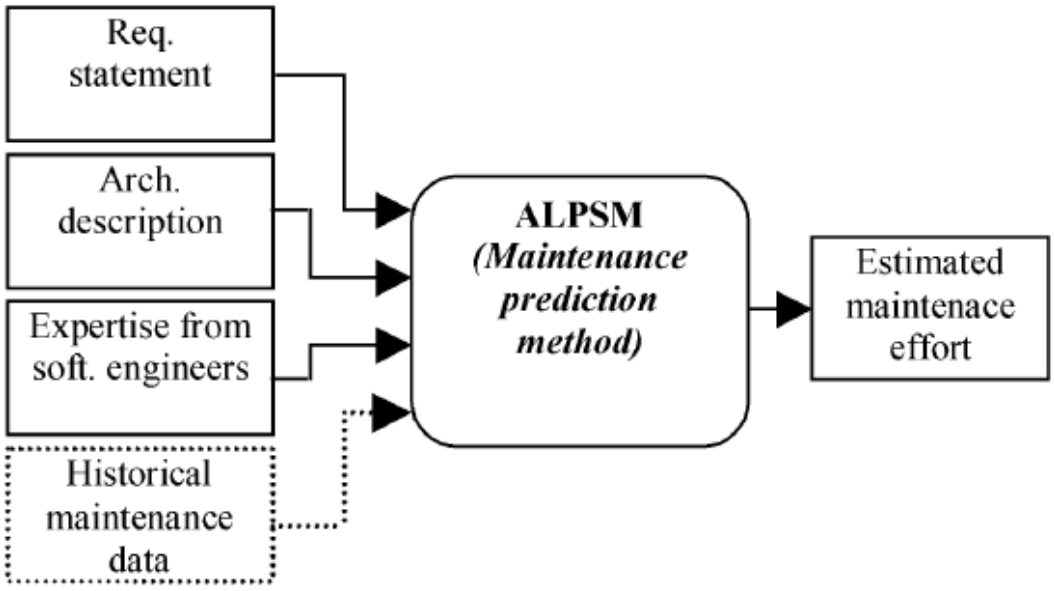


Figura 33. Entradas y resultado de ALPSM. [10]

4.3.8 Modelo de Evaluación de Arquitectura Software (SAEM)

El *Software Architecture Evaluation Model* o SAEM, tal y como explica [10], intenta definir métricas de calidad basada en la técnica Objetivo-Pregunta-Métrica o *Goal-Question-Metric* (GQM). La meta de las métricas es descubrir si ciertos atributos cumplen los valores especificados en la especificación de calidad para cada característica software.

La especificación de calidad está dividida en dos categorías: externa e interna. La categoría externa expresa la visión del usuario y la interna expresa la visión del desarrollador.

En cuanto a las actividades del método SAEM, esta técnica proporciona un modelo de evaluación de la calidad basado en colección de datos, medidas y análisis de los resultados. Los requisitos de calidad especificados se mapean a los atributos internos que estarán presentes en la arquitectura software, basados en el conocimiento de los expertos.

5 CAPÍTULO 5 – INTRODUCCIÓN AL DISEÑO SOFTWARE AERONÁUTICO

5.1 Introducción

En este capítulo se introducirán los principales conceptos a tener en cuenta en el desarrollo software: el proceso de desarrollo, los estándares de calidad software, la comparación entre dichos estándares y los utilizados actualmente en cualquier proyecto software, y las restricciones de diseño que se imponen debido a dichos estándares, lo que afectará como se verá en posteriores capítulos a la arquitectura software que se pretende evaluar.

El estándar de calidad y diseño software utilizado en la industria aeronáutica es la RTCA DO-178C, *Software Considerations in Airborne Systems and Equipment Certification*. [14]

5.2 Procesos de Desarrollo Software

En esta sección se describen brevemente los procesos de desarrollo software y sus objetivos principales. La descripción detallada es la que proviene de [14].

Dichos procesos se pueden clasificar en:

- Proceso de Requisitos Software
- Proceso de Diseño Software
- Proceso de codificación Software
- Proceso de Integración.

Los procesos de desarrollo software producen uno o más niveles de requisitos software, los requisitos de alto nivel se producen directamente a través del análisis de los requisitos del sistema y de su arquitectura. Usualmente, estos requisitos de alto nivel son desarrollados a lo largo del proceso de diseño software, produciendo de este modo uno o más sucesivos niveles más bajos de requisitos. Sin embargo, si el código fuente se genera directamente desde los requisitos de alto nivel, entonces los requisitos de alto nivel se consideran también directamente implementados sin información extendida.

El desarrollo de una arquitectura software implica toma de decisiones sobre la estructura del software. Durante el proceso de diseño software, la arquitectura software está definida y los requisitos de bajo nivel se desarrollan. Los requisitos de bajo nivel son requisitos de software desde los cuales el código fuente puede ser directamente implementado sin información más detallada.

Cada proceso de desarrollo software puede producir requisitos derivados. Los requisitos derivados son aquellos que no son directamente trazables a requisitos de más alto nivel. Los requisitos derivados son aquellos que no están directamente ligados a requisitos funcionales de más alto nivel, y que provienen de una necesidad específica de implementación de una solución particular, como por ejemplo:

- La necesidad de desarrollar un software de manejo de interrupciones para el computador destino elegido.
- La especificación de una tasa de iteración de un monitor periódico cuando no esté especificado por los requisitos de sistema asignados al software.

- La implementación de límites de escala cuando se use aritmética de punto fijo.

Los requisitos de alto y bajo nivel pueden incluir requisitos derivados. Los efectos de los requisitos derivados en los requisitos relativos a seguridad (entendiendo por seguridad o en inglés *safety* la relativa a hacer posible unas condiciones de vuelo seguro, no se está refiriendo a seguridad informática; a lo largo de todo este capítulo, el término seguridad hará referencia a *safety*) están determinados por el proceso de estimación de seguridad del sistema.

En la Figura 34 se muestra un esquema del proceso de desarrollo software de la DO-178C, descrita en [16]:

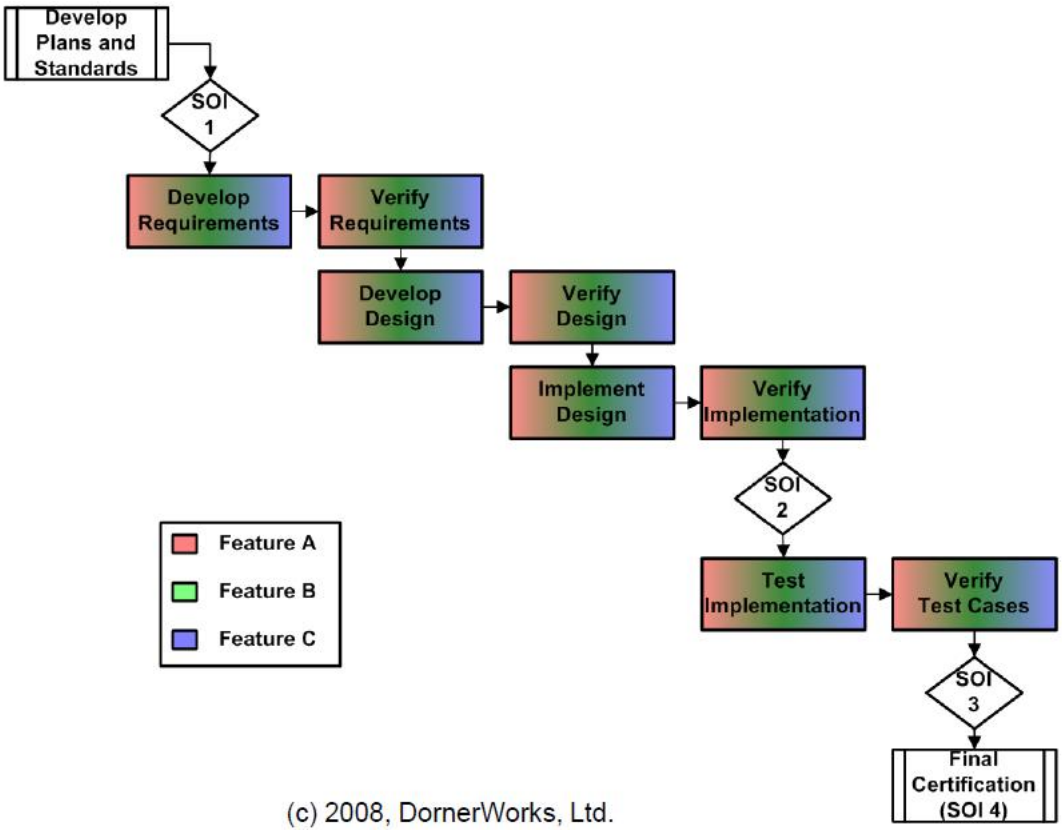


Figura 34. Proceso en Cascada de la DO-178C. [16]

5.2.1 Proceso de Requisitos de Software.

El proceso de requisitos software usa las salidas del proceso de ciclo de vida del sistema para desarrollar los requisitos de alto nivel de software. Estos requisitos de alto nivel incluyen requisitos funcionales, de prestaciones, de interfaces y relativos a seguridad.

Los objetivos del proceso de requisitos software son los siguientes:

1.- Los requisitos de alto nivel se desarrollarán.
2.- Los requisitos derivados de alto nivel se establecerán y se tendrán en cuenta para el proceso de evaluación de seguridad del sistema.

5.2.2 Proceso de Diseño de Software.

Los requisitos de alto nivel de software se refinan a través de una o más iteraciones en el proceso de diseño software para desarrollar la arquitectura software y los requisitos de bajo nivel que pueden ser usados para implementar código fuente.

Los objetivos del proceso de diseño software son los siguientes:

1.- La arquitectura software y los requisitos de bajo nivel son desarrollados a partir de los requisitos de alto nivel.
2.- Los requisitos derivados de bajo nivel se le proporcionan al proceso de evaluación de seguridad.

5.2.3 Proceso de Codificación de Software.

En el proceso de codificación de software, el código fuente es implementado a partir de la arquitectura software y los requisitos de bajo nivel.

El objetivo principal del proceso de codificación de software es el siguiente:

El código fuente es desarrollado de forma que sea trazable, verificable, consistente y que implemente correctamente los requisitos de bajo nivel.

5.2.4 Proceso de Integración

El computador que ejecutará el software (*target computer*) y el código fuente y código objeto a partir del proceso de codificación software se usan con el compilador, el enlazador (*linking*) y la carga de datos en el proceso de integración para desarrollar el sistema o equipo integrado embarcado.

El objetivo principal del proceso de integración es el siguiente:

El código objeto ejecutable se carga en el hardware destino para realizar la integración hardware/software.

5.3 Trazabilidad del Proceso de Desarrollo Software

Una parte muy importante del proceso de desarrollo software que pertenece a la parte de validación software y que da una idea de la calidad del mismo es la trazabilidad, que consiste en establecer unos enlaces entre los requisitos de alto nivel y la solución final software implementada, a nivel incluso de código fuente. Esto permite asegurar que todos los requisitos de alto nivel, pasando también por los de bajo nivel, han sido tenidos en cuenta en la implementación de la solución, posibilitando la verificación de la implementación completa de los requisitos del sistema asignados al software, y además proporcionar visibilidad de los requisitos derivados que no están trazados directamente a los requisitos del sistema. Entre las actividades de este proceso de trazabilidad se encuentran:

- Datos de trazabilidad, mostrando una asociación bidireccional entre los requisitos de sistema asignados a los requisitos software y los requisitos de alto nivel.
- Datos de trazabilidad, mostrando una asociación bidireccional entre los requisitos de alto nivel y los requisitos de bajo nivel

- Datos de trazabilidad, mostrando una asociación bidireccional entre los requisitos de bajo nivel y el código fuente.

5.4 Proceso de Pruebas Software

Probar el software embarcado tiene dos objetivos complementarios: un objetivo es demostrar que el software satisface sus requisitos. El segundo objetivo es demostrar con un alto grado de confianza que los errores que pudieran conducir a unas condiciones de fallo inaceptables, como se determina en el proceso de evaluación de seguridad del sistema, han sido eliminados.

En la Figura 35 se muestra un diagrama de las actividades de pruebas software. Los objetivos de los tres tipos de pruebas en la figura son:

1.- Pruebas de integración hardware/software: verificar la correcta operación del software en el entorno del target computer.
2.- Pruebas de integración software: verificar la interrelación entre los requisitos software y los componentes y verificar la implementación de los requisitos software y los componentes software dentro de la arquitectura software.
3.- Pruebas de bajo nivel: verificar la implementación de los requisitos de bajo nivel software.

Para satisfacer los objetivos de las pruebas de software:

a. Los casos de test deberían estar basados principalmente en los requisitos de software.
b. Los casos de test deberían estar desarrollados para verificar la correcta funcionalidad y para establecer condiciones que revelen errores potenciales.
c. El análisis de la cobertura de los requisitos software debería determinar qué requisitos software no fueron probados.
d. El análisis de cobertura estructural debería determinar qué estructuras software no fueron ejercitadas.

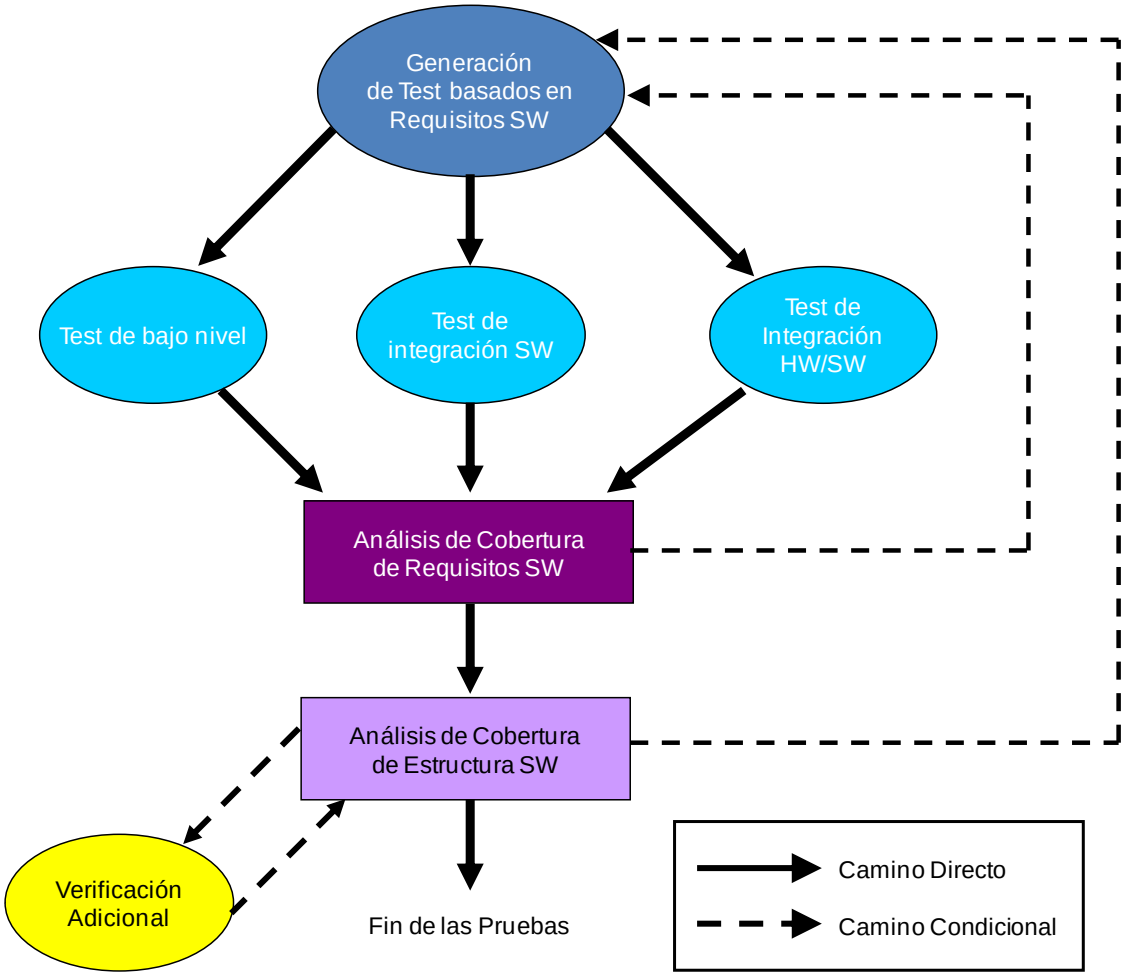


Figura 35. Actividades de Pruebas Software. [14]

5.5 Niveles de Diseño DAL

5.5.1 Categorías de las Condiciones de Fallo

Al diseñar un equipo de aviónica, y en general cualquier sistema, estructura o parte de una aeronave, se le asigna un nivel de diseño según la importancia del elemento aeronáutico sobre la seguridad del avión, y sobre todo, de las consecuencias que un fallo total o parcial en ese elemento puedan tener en la seguridad en vuelo de la aeronave.

El Nivel de Aseguramiento de Diseño o *Design Assurance Level* (DAL) es el nivel asignado al diseño del elemento aeronáutico, en este caso, un computador de aviónica, y es determinado a partir del proceso de estimación de seguridad (*safety*) y el análisis de peligros, examinando los efectos de una condición de fallo en el sistema. Las condiciones de fallo están divididas en categorías, según sus efectos en la aeronave, tripulación y pasajeros. Las categorías son, según [14], las siguientes:

a. *Catastrophic*: condiciones de fallo que impedirían un vuelo continuado seguro y aterrizaje
b. *Hazardous*: condiciones de fallo que reducirían la capacidad de la aeronave o la capacidad de la tripulación para hacer frente a condiciones de operación adversa al extremo que implicarían:

(1) una gran reducción en los márgenes de seguridad o capacidades funcionales,
(2) angustia física o mayor carga de trabajo tal que la tripulación de vuelo no podría confiar en realizar sus tareas de forma precisa o completa, o
(3) efectos adversos en los ocupantes, incluyendo daños serios o potencialmente mortales a un pequeño número de esos ocupantes.

c. *Major*: condiciones de fallo que reducirían la capacidad de la aeronave o la capacidad de la tripulación para enfrentarse con condiciones de operación adversas hasta el extremo de que implicarían, por ejemplo, una reducción significante en los márgenes de seguridad o capacidades funcionales, un incremento significativo en la carga de trabajo de la tripulación o en condiciones de perjudicar la eficiencia de la tripulación, o incomodidad a los ocupantes, incluyendo posiblemente daños.
d. *Minor*: Condiciones de fallo que no reducirían significativamente la seguridad de la aeronave, y las cuales implicarían acciones de la tripulación que están al alcance de sus capacidades. Las condiciones de fallo *Minor* pueden incluir, por ejemplo, una ligera reducción en los márgenes de seguridad o capacidades funcionales, un ligero incremento en la carga de trabajo de la tripulación, tales como cambios en el plan de vuelo rutinario, o algunas incomodidades a los ocupantes.
e. *No Safety effect*: Condiciones de fallo que no afectan la capacidad operacional de la aeronave o incremente la carga de trabajo de la tripulación.

5.5.2 Definición del Nivel de Software

Según estas condiciones de fallo, se asigna un nivel al hardware y al software; este nivel está basado en la contribución del HW/SW a las condiciones potenciales de fallo como están determinadas por el proceso de estimación de seguridad del sistema. El nivel HW/SW implica que el nivel de esfuerzo requerido para demostrar cumplimiento con los requisitos de certificación varía con la categoría de la condición de fallo. Las definiciones de los niveles SW (e igualmente para HW) son: [14]

a. Nivel A: Software cuyo comportamiento anómalo, como lo muestra el proceso de evaluación de seguridad del sistema, causaría o contribuiría a un fallo de la función del sistema que resulte en una condición de fallo *catastrophic* para la aeronave.
b. Nivel B: Software cuyo comportamiento anómalo, como lo muestra el proceso de evaluación de seguridad del sistema, causaría o contribuiría a un fallo de la función del sistema que resulte en una condición de fallo *hazardous* para la aeronave.
c. Nivel C: Software cuyo comportamiento anómalo, como lo muestra el proceso de evaluación de seguridad del sistema, causaría o contribuiría a un fallo de la función del sistema que resulte en una condición de fallo *major* para la aeronave
d. Nivel D: Software cuyo comportamiento anómalo, como lo muestra el proceso de evaluación de seguridad del sistema, causaría o contribuiría a un fallo de la función del sistema que resulte en una condición de fallo *minor* para la aeronave
e. Nivel E: Software cuyo comportamiento anómalo, como lo muestra el proceso de evaluación de seguridad del sistema, causaría o contribuiría a un fallo de la función del sistema sin efecto en la capacidad de operación de la aeronave o la carga de trabajo del piloto.

5.5.3 Determinación del Nivel de Software

El proceso de evaluación de seguridad del sistema determina, según [14], el nivel de software apropiado para los componentes software de un sistema particular basado en la condición de fallo que puede resultar en un comportamiento anómalo del software. El impacto de ambos, la pérdida de función y el malfuncionamiento, deben ser analizados. Los factores externos tales como condiciones ambientales adversas así como estrategias de arquitectura pueden considerarse cuando se identifique la condición de fallo y se determine el nivel de software.

Si el comportamiento anómalo de un componente software contribuye a más de una condición de fallo, entonces al componente software se le debe asignar el nivel de software asociado con la condición de fallo más severa a la cual el software puede contribuir, incluyendo condiciones de fallo combinadas.

5.6 Consideraciones Arquitectónicas

Se pueden adoptar estrategias en la arquitectura software para limitar el impacto de fallos, o detectar fallos y proporcionar respuestas del sistema aceptables para contenerlos. Estas técnicas arquitecturales típicamente se identifican durante el diseño del sistema, y vienen descritas en [14], tal y como se detalla a continuación.

5.6.1 Particionamiento

El particionamiento es una técnica que proporciona aislamiento entre componentes software para contener y/o aislar fallos y reducir potencialmente el esfuerzo del proceso de verificación del software. El particionamiento entre componentes software se puede alcanzar asignando recursos hardware únicos a cada componente (es decir, sólo se ejecuta un componente software en cada plataforma hardware en un sistema). Alternativamente, se pueden hacer provisiones de particionamiento para permitir a múltiples componentes software correr en la misma plataforma hardware. Sea cual sea el método, se tienen que tener en cuenta las siguientes consideraciones para los componentes software particionados:

a. No se debe permitir a un componente software particionado que contamine el código de otro componente software particionado, entrada/salida o áreas de almacenamiento de datos.
b. Se debe permitir que un componente software particionado consumir recursos compartidos del procesador sólo durante su periodo de ejecución planificado.
c. Los fallos del hardware asignados a un componente software particionado no deben causar efectos adversos en otros componentes software particionados.
d. Cualquier software que proporcione particionamiento debe tener el mismo o superior nivel software que el nivel más alto asignado a cualquiera de los componentes software particionados.
e. Cualquier hardware que proporcione particionamiento debe ser evaluado por el proceso de evaluación de seguridad del sistema para asegurar de que no afectará de forma adversa a la seguridad.

Los procesos de ciclo de vida del software deben tener en cuenta las consideraciones de diseño de particionamiento. Esto incluye la extensión y el ámbito de las interacciones

permitidas entre los componentes particionados y si la protección se implementa mediante hardware o por una combinación de hardware y software.

En capítulos posteriores se verá que esta condición de particionamiento tendrá una influencia muy grande en la definición de las arquitecturas objeto de estudio. El particionamiento incluye una fuerte restricción a la arquitectura software que tendrá gran influencia a la hora de diseñar los diferentes componentes software y sobre todo su dependencia e interacciones.

5.6.2 Software Disimilar de Versiones Múltiples

Es una técnica de diseño que implica producir dos o más componentes de software que proporcionan la misma función de forma que pueden evitar algunas fuentes de errores comunes entre componentes.

5.6.3 Monitorización de Seguridad

La monitorización de seguridad es un medio de protección contra condiciones de fallo específicas por medio de la monitorización directa de una función para fallos que podrían resultar en una condición de fallo. La monitorización de funciones se puede implementar en hardware, software o una combinación de ambos.

A través del uso de técnicas de monitorización, el nivel del software del software monitorizado puede asignarse a un nivel de software asociado con la pérdida de la función del sistema relacionada. Para permitir esta asignación, hay tres atributos importantes que la monitorización debería cumplir:

a. Nivel de software: al software de monitorización de seguridad se le asigna el nivel de software asociado con la categoría de condición de fallo más severa de la función monitorizada.
b. Cobertura de fallo del sistema: la evaluación de la cobertura de fallo del sistema de un mecanismo de monitorización asegura que el diseño del mecanismo de monitorización y su implementación son tales que los fallos que intenta detectar serán detectados bajo todas las condiciones necesarias.
c. Independencia de la función y del mecanismo de monitorización: el mecanismo de monitorización y el mecanismo de protección no se vuelven inoperativos por el mismo fallo que cause la condición de fallo.

5.7 Estándares de Calidad de Procesos Software

5.7.1 Introducción

Para concluir este capítulo, se van a analizar dos estándares de procesos software y modelos de capacidad y madurez muy implementados en la industria; uno está orientado a procesos software convencionales, y el otro a procesos software aeronáuticos. Después de su descripción, se hará una breve comparación entre ambos.

5.7.2 Visión General de SW-CMM

Según [15], el marco SW-CMM se inició en 1986 por el Software Engineering Institute (SEI). Su misión principal consiste en categorizar la madurez de los procesos de una compañía en cinco

niveles. En el ámbito del SW-CMM, un proceso software se puede definir como un conjunto de actividades, métodos, prácticas y transformaciones que las personas usan para desarrollar y mantener software y los productos asociados. Los niveles de madurez que da CMM, del 1 al 5, indican la efectividad general de las prácticas de ingeniería software de la compañía. El incremento de un nivel a otro se basa en alcanzar los atributos establecidos del nivel inferior previo.

Los niveles se describen como sigue: [15]

- Nivel 1: inicial – el proceso software se caracteriza como ad hoc, e incluso ocasionalmente como caótico. Hay pocos procesos definidos, y el éxito depende del esfuerzo individual.
- Nivel 2: repetible – los procesos de gestión de proyectos están establecidos para realizar el seguimiento de coste, planificación y funcionalidad. La disciplina de proceso necesaria tiene lugar para repetir éxitos tempranos en proyectos con aplicaciones similares.
- Nivel 3: definido – los procesos software para actividades de gestión e ingeniería están documentadas, estandarizadas e integradas en un proceso software estándar para la organización. Todos los proyectos usan versión aprobada y adaptada del proceso software de la organización para desarrollar y mantener software.
- Nivel 4: Gestionado – se recogen medidas detalladas del proceso software y calidad del producto. Ambos, el proceso software y los productos son comprendidos y controlados de forma cuantitativa.
- Nivel 5: Optimización – Se establece una mejora de proceso continua mediante *feedback* cuantitativo desde el proceso y desde ideas innovativas y tecnologías.

Cada nivel de madurez tiene áreas claves de proceso asociadas (KPAs) que describen los atributos de ingeniería software que deben estar presentes para satisfacer ese particular nivel (ver Tabla 2). Como el nivel de madurez aumenta, la calidad y la productividad aumentan y el riesgo de proyectos fracasados e impredecibles disminuye. Dentro de cada KPA hay metas/objetivos que deben ser alcanzados para satisfacer el KPA. Cada KPA está evaluado usando los siguientes componentes:

- Actividades: tareas a ser completadas para alcanzar el KPA con éxito.
- Compromisos: requisitos de la organización impuestos para asegurar que se realicen las actividades.
- Capacidades: todo aquello que permita a la organización cumplir los compromisos.
- Medida de implementación: el proceso de monitorizar las actividades.
- Verificación de la implementación: el proceso de asegurar que el KPA está siendo alcanzado de manera apropiada.

Maturity Levels	Key Process Areas
5 – Optimizing (focus on continuous improvement)	• Process Change Management • Technology Innovation • Defect Prevention
4 – Managed (focus on product & process quality)	• Quality Management • Process Measurement & Analysis
3 – Defined (focus on engineering process)	• Peer Reviews • Intergroup Coordination • Software Product Engineering • Integrated Software Management • Training Program • Organization Process Definition • Organization Process Focus
2 – Repeatable (focus on project management)	• Software Configuration Management • Software Quality Assurance • Software Sub-contract Management • Software Project Tracking & Oversight • Software Project Planning • Software Requirements Management
1 – Initial (focus on individual)	• None

Tabla 2. Aéreas de Proceso Claves para SW-CMM. [15]

Cada KPA está definido por un conjunto de prácticas clave (tales como políticas, procedimientos y actividades) que deben ponerse en práctica antes de que el KPA se alcance.

5.7.3 Visión General de la DO-178C/ED-12C

Como se detalla en [15], la DO-178/ED-12 fue desarrollada por la comunidad internacional de aviación civil en 1982. Fue revisado en 1985 para añadir más detalle. En 1992, el estándar DO-178B/ED-12B se completó y se ha convertido en el estándar de software para software embarcado en aeronaves en productos de aviación civil. El documento DO-178/ED-12 y todas sus revisiones fueron esponsorizadas por RTCA y EUROCAE, con la implicación de expertos de aviación, software y certificación de todo el mundo. El actual estándar es el DO-178C/ED-12C, desde enero de 2012.

La DO-178C está enfocada en los aspectos software del desarrollo de sistemas. Como parte de las tareas de ingeniería de sistemas, una estimación de seguridad de sistemas debe ser realizada antes de que la DO-178C pueda ser aplicada al esfuerzo del desarrollo del software. Una estimación de seguridad de sistemas es un proceso para identificar los peligros, las condiciones de fallo que conduzcan a dichos peligros, y los efectos de las estrategias de mitigación. La tarea de estimación de seguridad determina un nivel de software basado sobre la contribución del software a las condiciones de fallo potencial definidas en el proceso de estimación de seguridad del sistema. Los cinco niveles de software, de la A a la E, se resumen en la sección 5.5.2 y en la Tabla 3:

Failure Condition Category	Description	SW Level
Catastrophic	Failure conditions which would prevent continued safe flight and landing of the aircraft.	A
Hazardous	Failure condition which would reduce the capability of the aircraft or the ability of the crew to cope with adverse operation conditions to the extent that there would be: (1) a large reduction in safety margins or functional capabilities, (2) physical distress or higher workload such that the flight crew could not be relied on to perform their tasks accurately or completely, or (3) adverse effects on occupants including serious or potential fatal injuries to a small number of occupants.	B
Major	Failure conditions which would reduce the capability of the aircraft or the ability of the crew to cope with adverse operation conditions to the extent that there would be, for example, a significant reduction in safety margins or functional capabilities, as significant increase in crew workload or in conditions impairing crew efficiency, or discomfort to occupants, possibly including injuries.	C
Minor	Failure conditions which would not significantly reduce aircraft safety, and which would involve crew actions that are well within their capabilities.	D
No Effect	Failure conditions which do not affect the operational capability of the aircraft or increase crew workload.	E

Tabla 3. Niveles Software DO-178C/ED-12C. [15]

Estos niveles software definen diferentes grados de rigor. En el Anexo A de la DO-178C se listan los objetivos que se deben cumplir para cada nivel software específico. Estos niveles software definen un número de atributos deseables para los procesos de desarrollo y verificación software. Las diferencias en rigor se determinan por el número de objetivos que necesitan ser satisfechos, si un objetivo específico ha de ser satisfecho con independencia, y la formalidad del control de configuración de los datos software (documentación, códigos fuentes, etc.) producidos durante el desarrollo. Por ejemplo, el número de objetivos para cada nivel software se lista a continuación:

- Nivel A: 66 objetivos
- Nivel B: 65 objetivos
- Nivel C: 58 objetivos
- Nivel D: 28 objetivos
- Nivel E: 0 objetivos

La DO-178C se divide en actividades de desarrollo y procesos integrales. Las actividades de desarrollo incluyen planificación, requisitos, diseño, codificación e integración. Los procesos integrales incluyen verificación, gestión de la configuración, aseguramiento de la calidad y certificación. Los procesos integrales están superpuestos en cada una de las actividades de desarrollo (por ejemplo, verificación, gestión de configuración, aseguramiento de la calidad y certificación están aplicadas a cada actividad de desarrollo).

Los objetivos de la DO-178C se listan en el Anexo A de dicho estándar y están organizados alrededor de las actividades de desarrollo y los procesos integrales previamente descritos. Hay diez tablas en el Anexo A con objetivos-el tema de cata tabla se lista a continuación:

- Tabla A-1: Proceso de planificación software
- Tabla A-2: Proceso de desarrollo software
- Tabla A-3: Verificación de resultados del proceso de requisitos software
- Tabla A-4: verificación de resultados del proceso de diseño software
- Tabla A-5: Verificación de resultados de los procesos de codificación e integración software.
- Tabla A-6: Pruebas de los resultados del proceso de integración
- Tabla A-7: Verificación de los resultados del proceso de verificación.

Para ilustrar un ejemplo de la aplicación de la DO-178C y la estructura de las tablas del Anexo A, se utiliza el objetivo 1 de la tabla A-4 (Figura 36). El primer conjunto de columnas contiene información acerca de los objetivos de la DO-178C: número de objetivo, descripción y la referencia al párrafo de la DO-178C en donde ese objetivo se detalla. El siguiente conjunto de columnas con los encabezados A, B, C, D muestran la aplicabilidad de ese objetivo particular al nivel de software. Por ejemplo, el objetivo 1 es aplicable a los niveles A, B y C; sin embargo, no necesita ser cumplido para un software nivel D. Si el círculo que indica aplicabilidad está relleno, entonces ese objetivo debe ser cumplido con independencia. Las siguientes series de columnas describen los resultados producidos como evidencia de que el objetivo se ha cumplido. La columna “Descripción” lista donde se encuentran los datos. La columna “Ref.” identifica el párrafo dentro del capítulo 11 de la DO-178C que detalla los atributos de los datos de software. Las últimas cuatro columnas correlacionan el rigor de la gestión de configuración del resultado particular con el nivel de software asociado. La categoría 1 de control requiere más actividades de gestión de configuración que la categoría de control 2. Por ejemplo, la categoría de control 1 requiere informe de problemas y control de los cambios, mientras que la categoría de control 2 requiere sólo control de cambios.

	Objective		Applicability by SW Level				Output		Control Category by SW level			
	Description	Ref.	A	B	C	D	Description	Ref.	A	B	C	D
1	Low-level requirements comply with high-level requirements.	6.3.2a	I	I	m		Software Verification Results	11.14	2	2	2	

Figura 36. Parte de la Tabla A-4 de la DO-178C. [14], [15]

La evaluación de la implementación de la DO-178C se realiza a través de revisiones in situ y/o revisiones en escritorio del personal de EASA (European Aviation Safety Agency), representantes designados de ingeniería, y/o miembros del equipo de desarrollo software. La evaluación valora los datos para determinar si los objetivos listados en el Anexo A de la DO-178C se cumplen.

5.7.4 Comparación entre SW-CMM

Según [15], aunque ambos, SW-CMM y DO-178C son estándares de aseguramiento de la calidad, sus objetivos primarios son un poco diferentes. SW-CMM apunta no sólo al proceso de desarrollo sino también a su gestión y refinamiento. Cuanto mejor definido, gestionado y mejorado sea un proceso, mejor será la calidad del software y mayor la probabilidad de que el desarrollo esté dentro de coste y planificación.

El objetivo de la DO-178C es únicamente aseguramiento del diseño. Los objetivos específicos de la DO-178C se identifican para proporcionar el aseguramiento requerido por la criticidad del nivel. Para niveles A y B, las clases de errores de desarrollo software específicos están bajo observación para ser eliminados del código a través de estrictas actividades de verificación. Aunque las consideraciones de gestión, coste y planificación pueden ser importantes para los desarrolladores, no es el ámbito de las autoridades de certificación, que sólo se interesan por los aspectos de aseguramiento del diseño del desarrollo software.

Por lo tanto, SW-CMM no puede usarse como medio de cumplimiento aceptable de la DO-178C. Sin embargo la implantación exitosa en la empresa de SW-CMM en niveles de madurez 2 y 3 es una garantía bastante aceptable de que dicha empresa puede ser capaz de cumplir el estándar DO-178C. La mejor estrategia es integrar los procesos SW-CMM y DO-178C para evitar sobrecostes.

6 ARQUITECTURAS SOFTWARE AEROESPACIALES Y AERONÁUTICAS

6.1 Introducción a las Arquitecturas Software Aeroespaciales y Aeronáuticas

En este capítulo se repasarán las arquitecturas software más importantes utilizadas en la aeronáutica y en la industria aeroespacial. Se introducirán los conceptos de arquitecturas federadas y arquitecturas IMA, que incluyen también una descripción de los sistemas hardware donde van implementadas, y además se analizarán las particularidades de las arquitecturas software de estos sistemas, como son las particiones en espacio y tiempo.

6.2 Tipos de Arquitecturas de Sistemas Software

Hay dos tipos principales de sistemas de arquitecturas SW, según [17]:

- Arquitecturas federadas
- Arquitecturas IMA (*Integrated Modular Avionics*)

6.2.1 Arquitecturas Federadas

En [17] se describen dichas arquitecturas, que se explican a continuación.

Las arquitecturas federadas responden al principio de una función, un computador. Por lo tanto, esto supone que para cualquier función nueva que se quiera implementar, se necesita tener una caja o computador dedicado. Esto presenta dos límites:

- Por una parte, el peso: un computador tiene un chasis, tarjetas, y alimentación. Al no poder combinar ninguno de estos elementos con otras funciones, esto supone un aumento de peso.
- Mantenimiento: un número elevado de cajas supone un incremento en las tareas de mantenimiento.

En este tipo de arquitectura de sistemas, para implementar cada función, se necesitaría un conjunto independiente de sensores, unidades de procesamiento y actuadores, de manera que se producen unas circunstancias:

- Cada función viene implementada en un sistema *"stand-alone"* que tiene sus propias interfaces a sensores y actuadores
- Los datos no se comparten, por tanto existen particiones intrínsecas de funciones, que están localizadas en un determinado sistema.
- Las interfaces suelen ser punto a punto, como los buses de comunicaciones ARINC 429, y señales discretas y analógicas.

Las arquitecturas federadas hacen uso de funciones de aviónica distribuidas que se empaquetan como unidades autocontenidas:

- Un armario de aviónica de equipo totalmente dedicado, que aloja una determinada aplicación
- Cada uno de los equipos es una Unidad Reemplazable en Línea, o LRU.

Esta arquitectura está caracterizada por ser relativamente fácil de mantener (excepto por lo citado anteriormente) siempre y cuando se entienda el mantenimiento por reemplazar una determinada caja; también tiene una estructura simple de interfaces y de cables y buses de interconexión.

Esta arquitectura es típica de sistemas de aviónica más antiguos provenientes de los años 80 y 90, como el programa Eurofighter (Figura 37).

Figura 37. Eurofighter Typhoon.

En la Figura 38 se muestra un ejemplo de dicha arquitectura: [18]

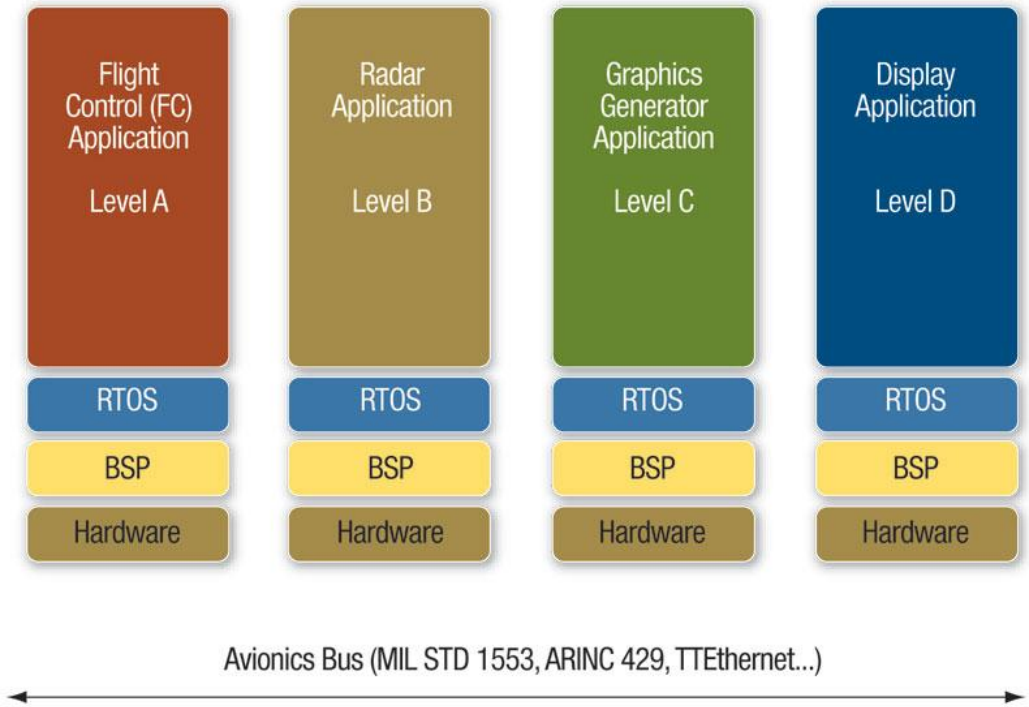


Figura 38. Ejemplo de Arquitectura Federada. [18]

Esta arquitectura tiene una serie de desventajas:

- Cada una de las cajas tiene una función específica, con un hardware y software desarrollado específicamente
  - Por lo tanto, cada sistema está desarrollado desde cero (más o menos), sin poderse reutilizar tecnología
  - El hardware de los computadores sufren problemas de obsolescencia.
- El peso y el consumo se ven incrementados, pues cada unidad lleva su correspondiente chasis, lo cual supone problemas de protección frente al ambiente y disipación de calor.
- Añadir nuevas funcionalidades supone un rediseño de carácter mayor, tests de integración y trabajos adicionales de verificación.
- Tener un amplio rango de computadores supone problemas logísticos (mantener un amplio y variado catálogo de piezas de repuesto)
- Para compartir los datos entre los diversos sistemas se necesita un sistema de bus centralizado, lo cual añade complejidad al diseño e incrementa el peso.

6.2.2 Arquitecturas IMA

El concepto IMA se basa en el estándar ARINC 651, (descrito en la referencia [19]) y se forma alrededor del concepto de un computador potente con un sistema operativo que permite procesamiento independiente de software de aplicación, mientras mantiene una partición robusta entre los módulos de software para cada una de las funciones más críticas. El computador está alojado en módulos hardware, formando un subsistema con un diseño de chasis común, compartiendo procesamiento tolerante a fallos, fuentes de alimentación centralizadas e interfaces de avión flexibles.

El sistema IMA, tal y como describe [19], comprende un número de *cabinets* o armarios de aviónica que contienen equipo para realizar la mayoría del procesamiento y entradas/salidas (I/O) localmente a sensores, actuadores e indicadores. Los *cabinets* están conectados entre sí y con los actuadores, sensores e indicadores vía bus de datos ARINC 629. En las Figura 39 y Figura 40 se muestra la arquitectura del sistema IMA:

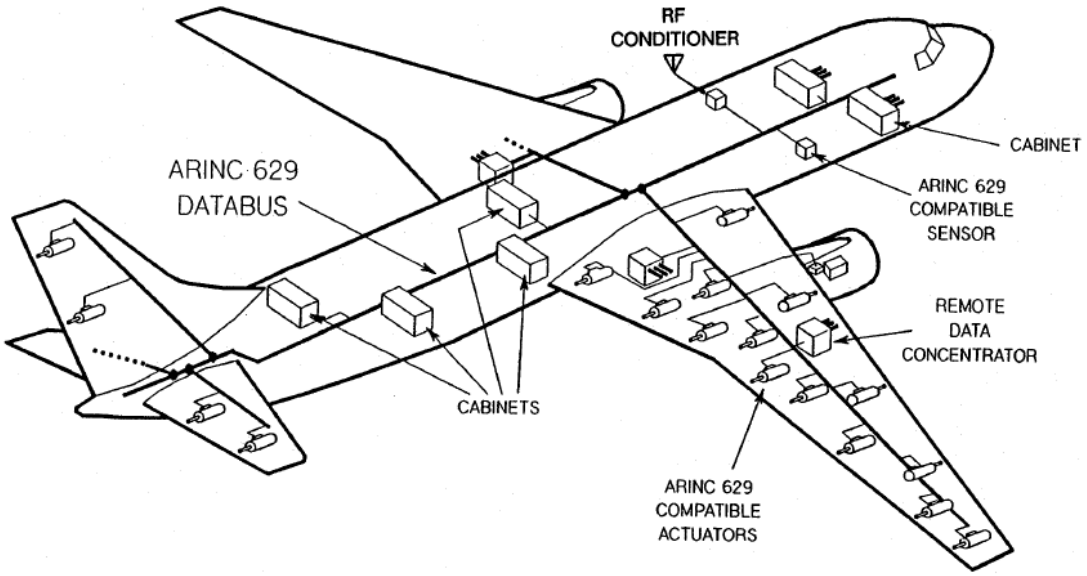


Figura 39. Descripción Funcional de la Arquitectura IMA. [19]

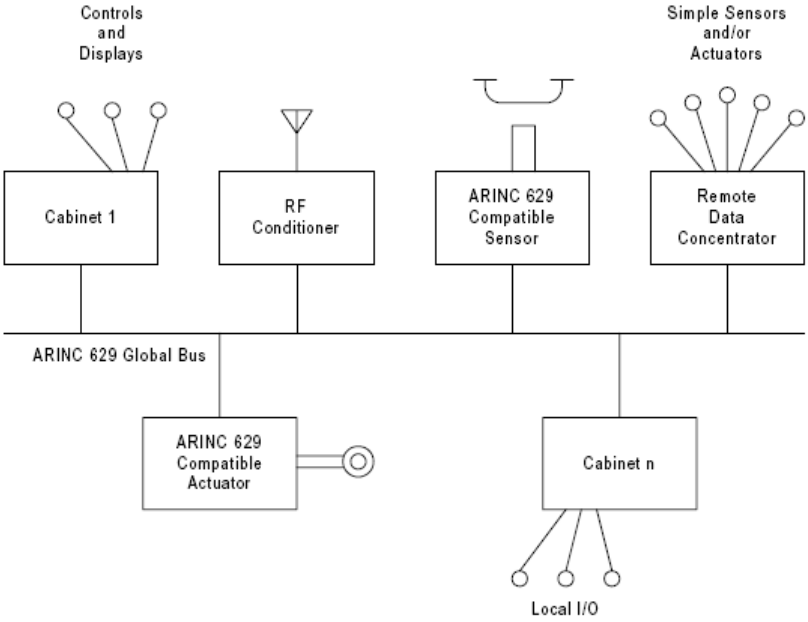


Figura 40. Visión General de la Arquitectura IMA. [19]

La arquitectura IMA es típica de los aviones comerciales más modernos, como los Airbus o Boeing. En la Figura 41 se muestra el A380 de Airbus, que implementa los conceptos de IMA.

Figura 41. Airbus A380.

En cada *cabinet* hay una serie de módulos procesadores que integran aplicaciones de diferente nivel de criticidad, cuya arquitectura se detallará más adelante, pero como adelanto y por comparación con la arquitectura federada, en la Figura 42 se puede ver cómo sería su disposición:

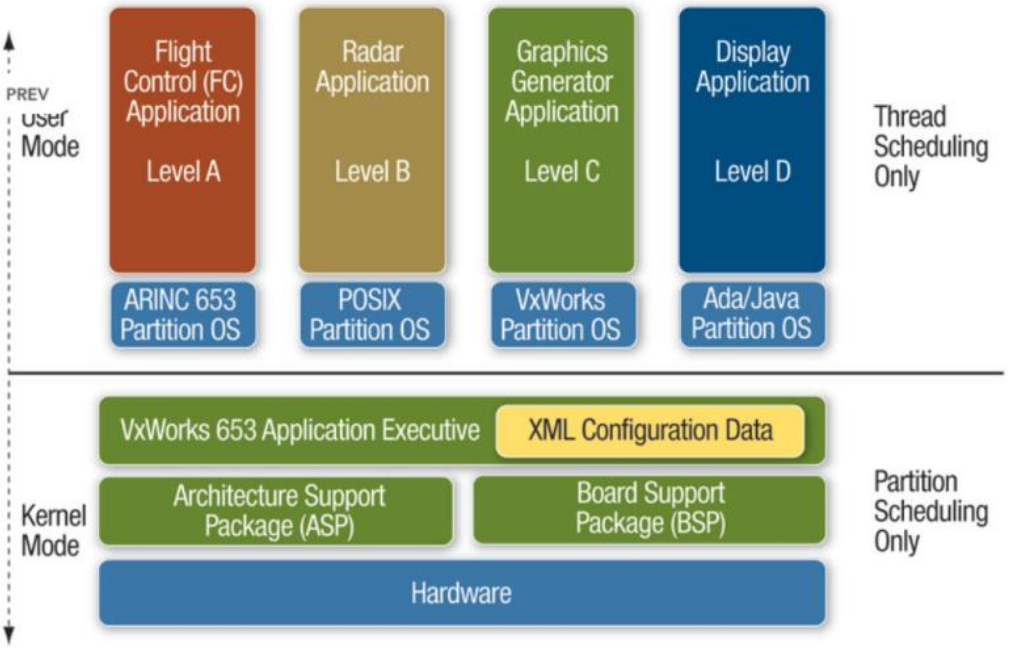


Figura 42. Ejemplo de Arquitectura Integrada. [18]

6.2.2.1 Componentes del Sistema IMA

Los componentes hardware de un sistema IMA incluyen, según [19], los siguientes elementos:

6.2.2.1.1 Cabinets

Su función es proporcionar los recursos de computación e interfaces necesarias para todo el software de aplicación que reside en el *cabinet*.

Hay tres elementos primarios que pueden considerarse como parte de un *cabinet*: el chásis, el *backplane assembly* (es una especie de placa base con conectores y bus de interconexión en donde van ensamblados los módulos; por su difícil traducción se mantendrá el término inglés en el resto del trabajo), y los módulos funcionales.

- Chasis del cabinet y *backplane assembly*

El chasis proporciona el soporte mecánico y eléctrico para instalar un grupo de módulos funcionales, además de constituir la interfaz entre los módulos y el fuselaje.

El *backplane assembly* constituye la interfaz entre los módulos funcionales y el resto de la aviónica. Está dividido en tres áreas: la primera implementa la interfaz entre los cables del avión a los conectores físicos del *backplane*; la segunda implementa el bus del *backplane*, compatible con la norma ARINC 659; la tercera implementa la alimentación eléctrica.

En la Figura 43 se muestra una vista general de un *cabinet*:

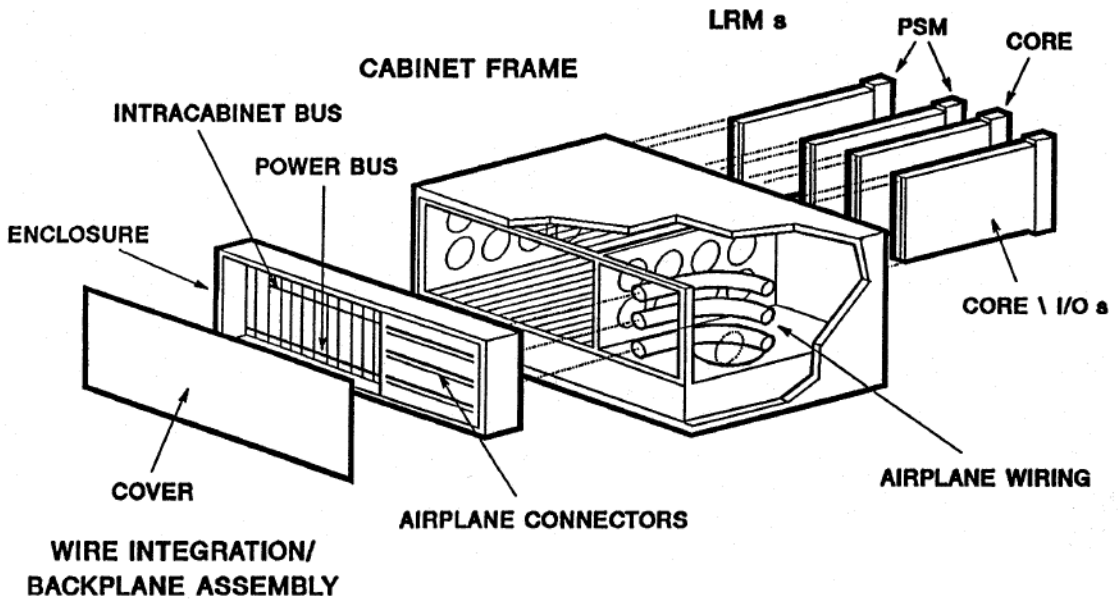


Figura 43. Arquitectura Física del Cabinet. [19].

- Módulos Reemplazables en Línea (Line Replaceable Modules, LRMs)

Los módulos funcionales se empaquetan como LRMs dentro del cabinet. Según su función, se pueden clasificar como:

1) Procesador principal
2) I/O (Input/Output) estándar
3) I/O especiales
4) Módulo de suministro electrico
5) Puente de buses
6) Puerta de enlace (Gateway)
7) Memoria masiva

El número y distribución de unidades de módulos funcionales en cualquier *cabinet* depende de la localización, número de funciones, el tipo, y el nivel de redundancia/tolerancia a fallos requerida.

Estos módulos deberían de ser intercambiables, por lo tanto es necesaria una rigurosa definición de los aspectos estáticos de las interfaces, incluyendo definición de conectores, pin-out y características de las señales.

En la Figura 44 se muestra un ejemplo de la arquitectura de módulos funcionales:

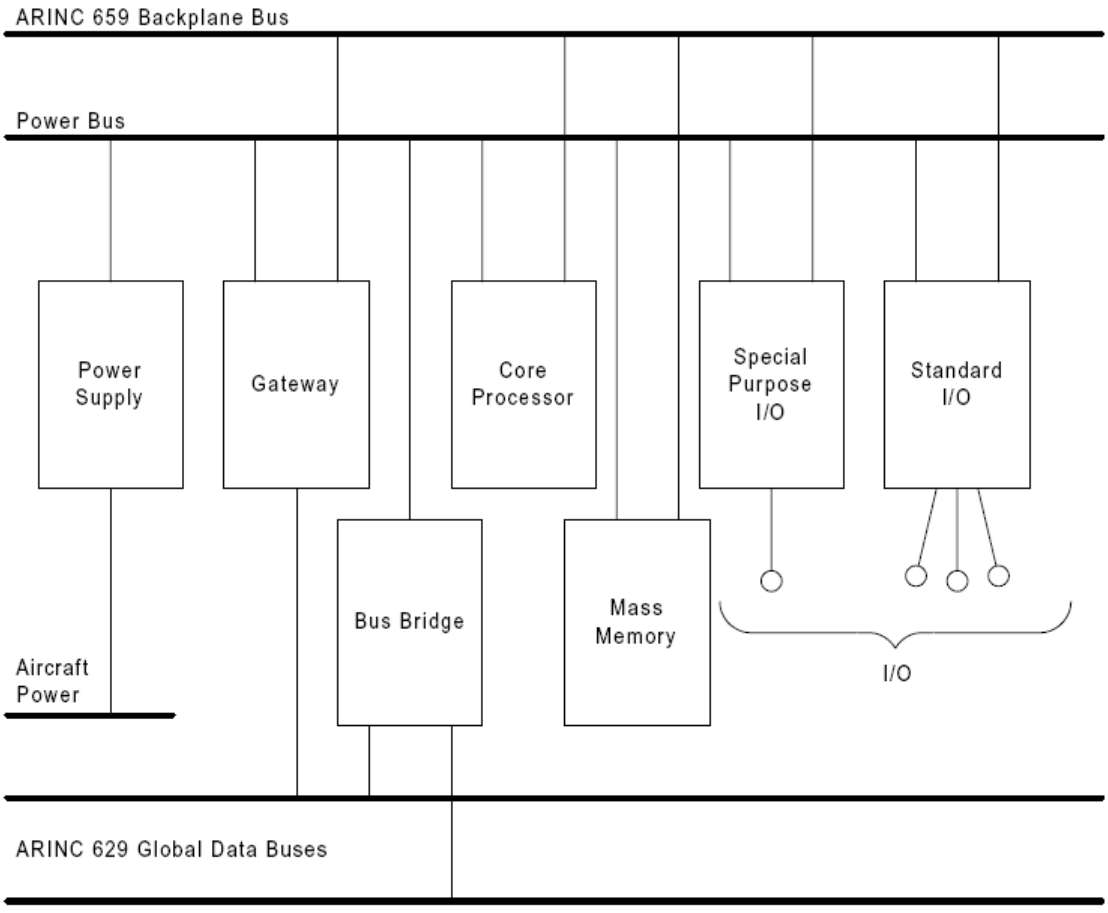


Figura 44. Arquitectura Funcional del Cabinet. [19]

Los módulos tipo procesador principal proporcionan la potencia de computación del *cabinet*, que puede contener uno o varios. El software de aplicación puede ir cargado en cualquiera de estos procesadores.

Los módulos de I/O actúan como transductores de señales analógicas o discretas en datos digitales que son trasferidas a los módulos de procesamiento via bus *backplane*.

Los módulos *gateways* o puentes de buses servirán para transformar señales de buses de diferentes tipos.

Los módulos I/O especiales se utilizan con señales muy específicas.

Los módulos de alimentación eléctrica sirven para alimentar los módulos del *cabinet*, realizando una transformación eléctrica entre la alimentación de la aeronave y la requerida por cada módulo.

6.2.2.1.2 Buses de Datos (ARINC 629, ARINC 429)

El bus ARINC 629 se utiliza para la comunicación entre los diferentes *cabinets* de la aeronave. Es un bus de datos serie bidireccional.

Otros buses muy extendidos para la comunicación entre dispositivos de aviónica es el ARINC 429, que es un bus de datos punto a punto de TX/RX.

6.2.2.1.3 Dispositivos Compatibles con ARINC 629

Son dispositivos que sirven para interactuar con el mundo exterior, como por ejemplo, sensores, actuadores e indicadores, que a su vez son compatibles con el bus de datos ARINC 629 para la comunicación con los *cabinets*.

6.2.2.1.4 Concentradores de Datos Compatibles con ARINC 629

Su función es convertir información recibida de diferentes formatos (señales analógicas, discretas e incluso otros tipos de buses digitales) al formato ARINC 629 para la comunicación con otros *cabinets*. También funciona de manera inversa, transformando datos en formato ARINC 629 a otro tipo de señal.

6.2.2.2 Ejemplo de Arquitectura IMA

Un ejemplo de esta arquitectura IMA (de las muchas que se pueden crear) se muestra en la Figura 45:

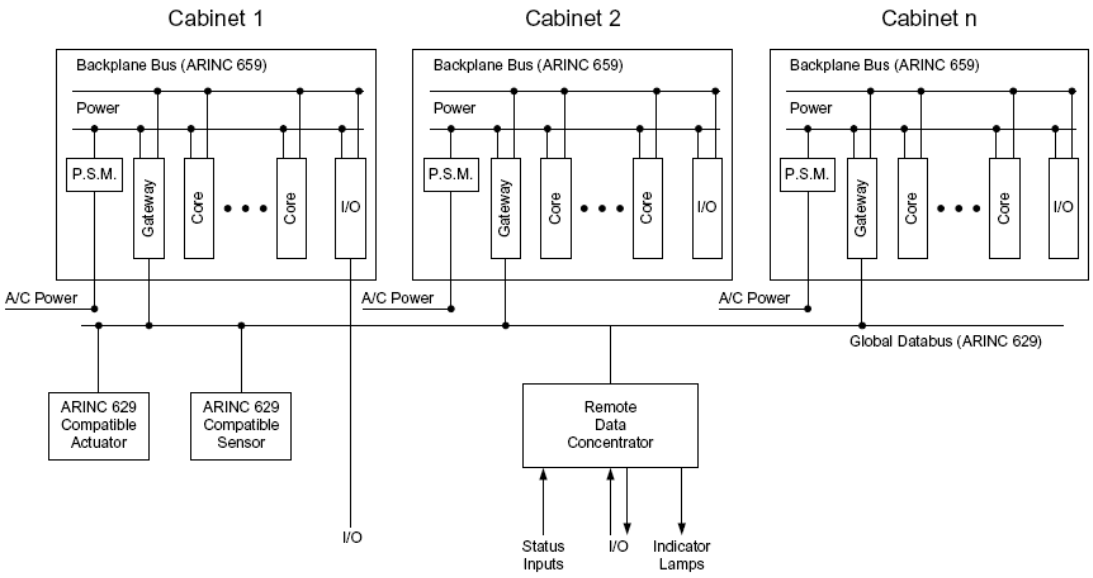


Figura 45. Ejemplo de Arquitectura IMA. [19]

Esta arquitectura se basa en el concepto de procesamiento centralizado, usando un módulo de procesamiento muy potente para proporcionar una función de computación genérica. Para completar la arquitectura, se usan módulos de alimentación eléctrica, módulos de I/O, y sensores/actuadores compatibles con ARINC 629. Los módulos dentro del *cabinet* se conectan via bus ARINC 659.

La arquitectura del módulo de procesamiento, que dará lugar al estudio de arquitectura software posterior en este capítulo, se muestra en la Figura 46:

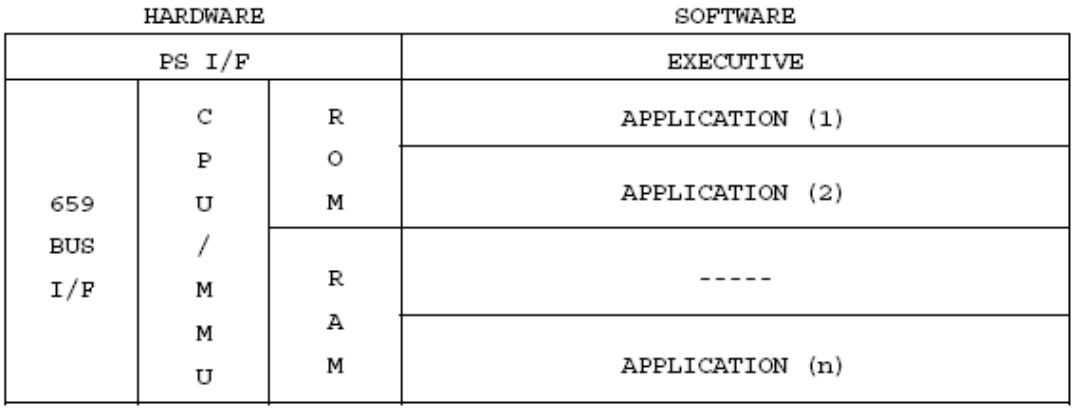


Figura 46. Arquitectura Hardware/Software del Módulo IMA. [19]

Este módulo incluye:

1) Interfaz de fuente de alimentación
2) Interfaz bus de datos ARINC 629
3) Interfaz de bus *backplane* ARINC 659
4) Procesador central y unidad de gestión de memoria
5) Memoria no volátil para almacenar programas y datos.
6) RAM para ejecutar programas de aplicación

El software está basado en una estructura de ejecutivo único/múltiples aplicaciones. El ejecutivo único es el responsable de planificar, controlar y ejecutar aplicaciones.

La gestión de memoria se emplea para proporcionar un particionamiento robusto e impenetrable entre las funciones asignadas al procesador. Como resultado, las tareas de aplicación no tienen ningún efecto en la función de gestión de memoria. El hardware de gestión de memoria está implementado en el procesador. Las tablas de gestión de memoria no necesitan ser modificadas dinámicamente, y por tanto pueden estar almacenadas en memoria no volátil.

6.3 El concepto de Arquitectura Software IMA

6.3.1 Introducción a la Arquitectura Software IMA

La funcionalidad del sistema de aviónica dentro del concepto IMA la proporciona el software de aplicación, que en sí mismo ofrece una gran flexibilidad para mejorar la funcionalidad.

La arquitectura software general se describe en [19] y se muestra en la Figura 47:

Los componentes del sistema son:

- Software de aplicación, que realiza la funcionalidad del sistema de aviónica
- Núcleo software, que proporciona un entorno estándar y común en el cual se ejecuta el software de aplicación.

El núcleo software está dividido en dos partes:

a. Sistema Operativo (SO): gestiona respuestas lógicas a las demandas del software de aplicación. Sus funciones incluyen la asignación del tiempo de procesamiento, canales de comunicación y recursos de memoria. El SO mapea las peticiones de las aplicaciones a los mecanismos lógicos a nivel de sistemas, y proporciona la interfaz lógica uniforme a las aplicaciones.
b. Sistema de Interfaz Hardware (HIS): gestiona los recursos hardware físicos en nombre del SO. El HIS mapea las peticiones lógicas del SO sobre la configuración física particular del módulo de procesamiento central.

Por otro lado, la especificación ARINC 653, descrita en [21], define una interfaz de usuario común a nivel de aplicación software, que minimiza los efectos de la implementación hardware subyacente. A continuación se exponen los conceptos más importantes de [21].

El hardware proporciona los medios físicos de acceso al sistema IMA vía bus *backplane* junto con la gestión de memoria y otras funciones de particionamiento, que aseguran que las aplicaciones no puedan interferir entre ellas o con el SO.

El software que compone el HIS mapea la implementación particular de hardware sobre el SO. Por tanto, es único con respecto a dicha implementación hardware.

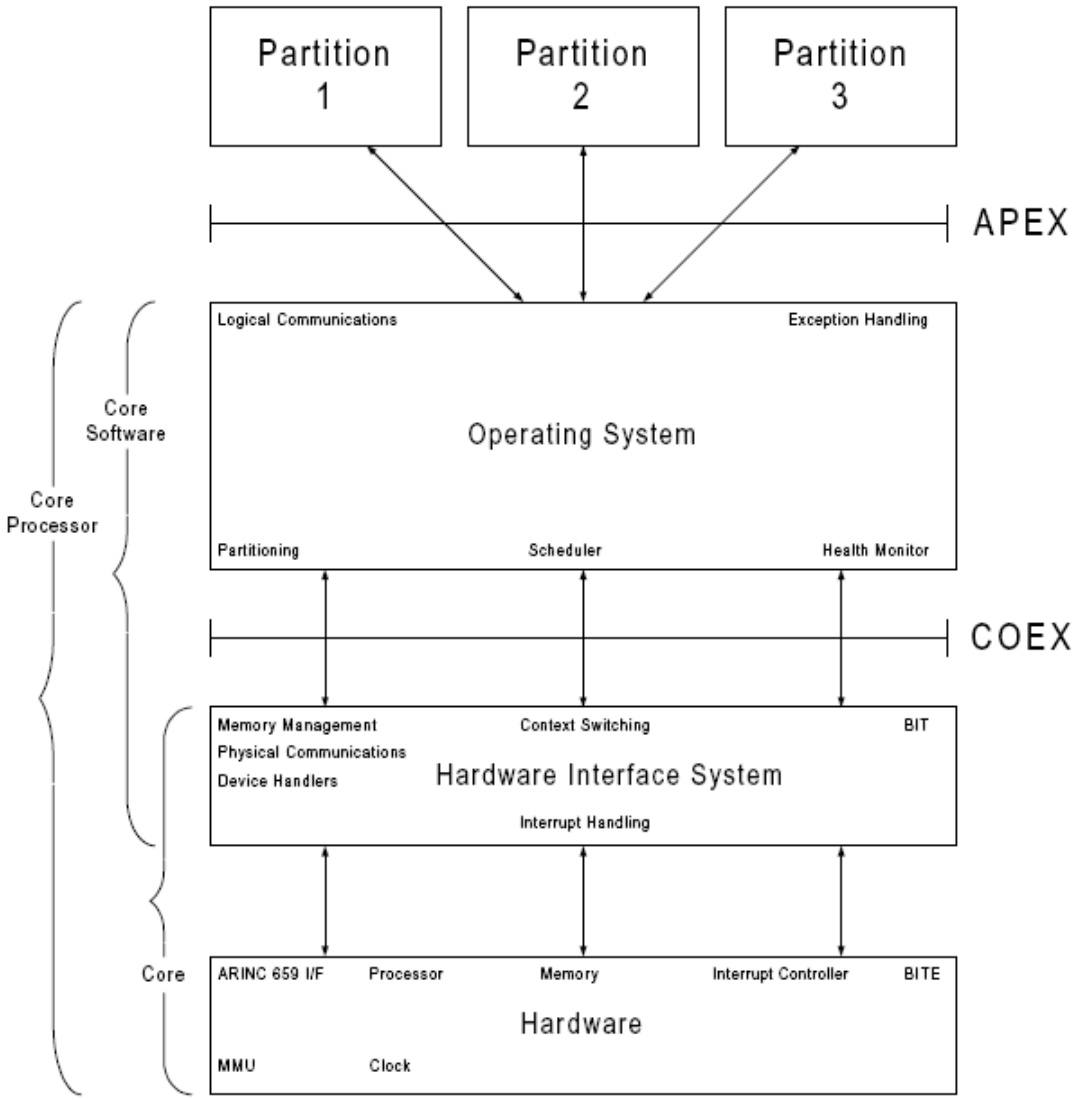


Figura 47. IMA Software Architecture. [19]

El proceso de descomponer el software de aplicación en unidades bien particionadas debe hacerse de manera que proporcione mayores beneficios que el simple hecho de tener el sistema dividido en unidades más fácilmente gestionables. Por ejemplo, los criterios de particionamiento deben hacerse pensando en tener agrupados elementos que tengan mayor probabilidad de ser modificados, o agrupar aquellos que sean dependientes de características temporales específicas, y así. Es más, para aquellos elementos susceptibles de ser modificados, se pueden a su vez dividir en elementos cuya funcionalidad deba ser mejorada de aquellos cuya implementación sea específica del entorno físico en el que se van a ejecutar.

Es necesario así mismo establecer un sistema de particiones entre las aplicaciones software lo suficientemente robusto para asegurar que se crean auténticas “paredes de ladrillo” entre aplicaciones, especialmente cuando estas aplicaciones puedan tener diferentes niveles de criticidad software.

Es también necesario establecer las especificaciones detalladas para las aplicaciones software, con el objetivo de definir las interfaces entre cada una de las aplicaciones software y los

recursos hardware, así como definir la manera en que los recursos hardware estarán disponibles.

La portabilidad de las aplicaciones software requiere la estandarización de la interfaz entre las aplicaciones software y el hardware del procesador central. Dicha interfaz puede ser una combinación de hardware y software, que represente un entorno común para el software de aplicación. Esto puede claramente dividirse en dos áreas de interés:

- El método lógico de gestión de recursos e intercomunicación
- Los medios de mapear estos métodos lógicos en el hardware físico.

En la Figura 48 se puede ver con más detalle determinadas partes de la arquitectura interna de cada elemento: [20]

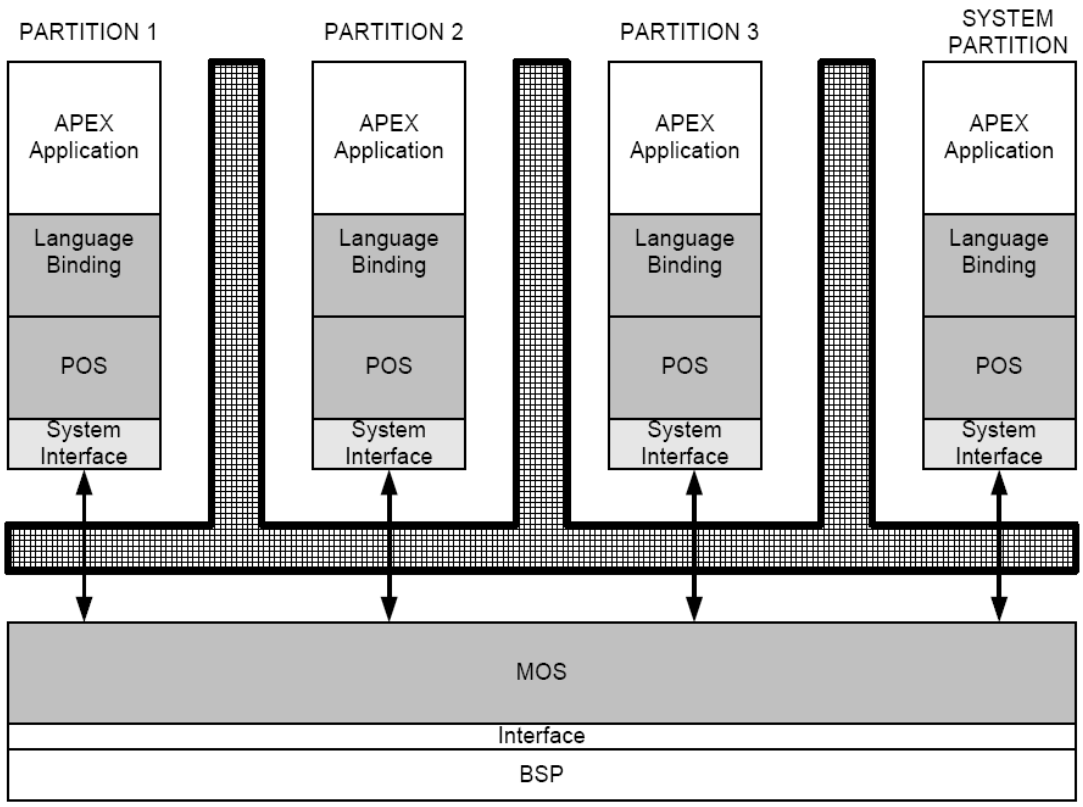


Figura 48. Diseño de Memoria de Aplicaciones Múltiples. [20]

Tal y como se define en [20], dentro de cada partición, se puede observar que existe el código ejecutable de la aplicación, el *Language Binding* o API que permite el uso de librerías contenidas en el POS desde la aplicaciones; el POS o *Partition Operating System*, un kernel incluído en la partición para gestionar el funcionamiento de la misma a nivel local; el MOS o *Module Operating System*, que es el sistema operativo del módulo; La interfaz, que en este caso sería la COEX, definida más adelante; y el BSP o *Boarding Support Package*, equivalente al HIS. El BSP tiene mucha importancia, pues es el software que aísla el SO del hardware del computador y permite que el SO resida en diferentes arquitecturas hardware. El BSP inicializa el procesador, los dispositivos y la memoria; realiza varios chequeos de memoria y muchas otras cosas. Una vez que la inicialización se haya completado, el BSP puede aún funcionar para realizar manipulaciones de cache de bajo nivel. Mientras que el SO puede ser desarrollado

como un sistema operativo de propósito general, para ser usado potencialmente un muchos diferentes procesadores, el BSP está típicamente muy altamente customizado para cada aplicación específica, configuración de hardware de computador, y conjunto de recursos.

El área rayada de la figura representa el mecanismo de protección que prohíbe o controla estrictamente las operaciones de una partición a otra y desde cualquier operación al SO. Para impedir que las aplicaciones estén completamente aisladas, se introducen unos mecanismos de comunicación para permitir enviar información en una secuencia controlada entre particiones y desde particiones a un dispositivo de I/O. Estos mecanismos se verán más adelante.

6.3.2 Elementos Principales de la Arquitectura Software

6.3.2.1 Funciones Software

Las funciones software que están implementadas en un procesador central IMA están representadas en la Figura 47, y son las siguientes:

- Software de aplicación
- Sistema Operativo (SO)

En los siguientes apartados se describen con más detalle dichas funciones.

6.3.2.2 Interfaces Software

La arquitectura IMA permite que el software de aplicación, el software del sistema operativo y el núcleo hardware sean desarrollados independientemente por diferentes suministradores. Para ello, se definen dos interfaces software claves, según [19]:

- APEX: La interfaz Aplicaciones\Ejecutivo
- COEX: La interfaz núcleo hardware y HIS/Ejecutivo

En este contexto de definición de interfaces, el sistema operativo de IMA se refiere como el "ejecutivo".

6.3.2.2.1 Interfaz APEX

La interfaz APEX [19] define el entorno común para que los programas de aplicación se comuniquen con el SO. La definición de APEX consiste básicamente en llamadas de procedimiento al sistema.

Todas las comunicaciones, incluyendo peticiones, comandos, respuestas, datos de I/O, etc, entre las aplicaciones entre sí, y las aplicaciones y el SO, se realizan mediante dicha interfaz de mensajes definida de manera rigurosa. Las particiones entre aplicaciones y entre las aplicaciones y el SO son controladas por mecanismos hardware estándares, y por tanto, si una aplicación cumple con el estándar APEX, se puede asegurar que se comunicará correctamente con el SO.

La interfaz APEX se define en el estándar ARINC 653 [21]. En posteriores apartados se detallarán las características de dicha interfaz.

6.3.2.2.2 Interfaz COEX

La interfaz COEX [19] define una frontera compartida por el sistema operativo y las funciones específicas del hardware de bajo nivel.

La interfaz COEX especifica las capacidades del hardware necesarias para dar soporte al SO. Este soporte incluye la inicialización, gestión de memoria, interfaz de bus e interrupciones.

La interfaz COEX está diseñada de acuerdo a una especificación rigurosa que incluye la interfaz entre el SO y las primitivas de bajo nivel que mapean las funciones sobre el hardware físico.

6.3.2.3 Software de Aplicación

El software de aplicación, según [19], es aquel que realiza una función de aviónica específica. Está especificado, desarrollado y verificado al nivel de criticidad apropiado con la función del sistema.

El diseño de software modular permite la implementación de particiones software para aislar las funciones de aviónica dentro de un entorno hardware común. Dicho software de aplicación debería ser independiente del hardware. Es necesario que el software dentro de una partición esté completamente aislado de otras particiones de manera que una partición no pueda causar un fallo en otra partición.

Para asegurar la integridad de las particiones, las imágenes cargadas en las particiones deben de estar construidas de manera estática y separada. Deben de estar completamente aisladas como módulos de programas independientes sin ninguna interdependencia con otros módulos de programas. El código de la partición puede cambiar los atributos de planificación de procesos para cambiar la ejecución o el estado de un proceso dentro de esa partición.

De esta forma, la integridad de una aplicación no se ve comprometida por el comportamiento de otra aplicación, si la otra aplicación se considera más o menos crítica.

Todas las comunicaciones entre aplicaciones deberían ser realizadas a través del sistema operativo, cuyos mecanismos aseguran que no hay ninguna violación de dicha interfaz, y que no hay ninguna aplicación que o bien monopoliza un recurso o bien deja a otra suspendida permanentemente esperando una petición entre aplicaciones.

Se deben también implementar mecanismos para la gestión de I/O de, por ejemplo, datos provenientes de sensores, de manera que esto mecanismos de gestión estén fuera del software de aplicación, haciendo independiente de este modo el desarrollo de dicho software de aplicación de la implementación hardware específica de la aviónica de una aeronave.

Por último, cabe destacar que el software de aplicación se invoca por el planificador del sistema operativo de una manera determinista. La frecuencia y prioridad del planificador requerido para aplicaciones cíclicas debería estar definido en el SO.

Las aplicaciones deben invocar a los procedimientos de la interfaz APEX para todos los procesos.

6.3.2.4 Sistema Operativo

El sistema operativo (SO), según [19], tiene varias funciones principales dentro de la arquitectura IMA. Su papel principal es mantener la integridad funcional en la planificación y el *dispatching* (cesión de CPU para su uso por procesos de una aplicación) de programas de aplicación. Se debe demostrar que el nivel de determinismo temporal no se ve afectado por la agregación de otras aplicaciones al sistema. Son funciones del SO asegurar el aislamiento de las particiones, asignar tiempo de procesamiento a las particiones, y realizar el *dispatch* de los procesos para su ejecución.

Un método de conseguir esto es implementar un método de división del tiempo en trozos, y separar las aplicaciones en grupos de "tiempo estricto" y "planificadas". Cada aplicación de tiempo estricto se asegura una cantidad específica de tiempo de procesamiento en cada trozo de tiempo, de manera que puede realizar un cierto número de algoritmos definidos en cada periodo de tiempo asignado. Si una aplicación sobrepasa su período de tiempo entonces es interrumpida por el SO. El planificador proporciona una cantidad de tiempo suficiente para cada trozo de tiempo para trabajar eficientemente. Una aplicación de tiempo estricto no puede ser interrumpida por una interrupción software.

El SO debe ser capaz de reconocer procesos tanto periódicos como aperiódicos, y planificar y hacer el *dispatchintg* de todos los procesos.

El SO también gestiona la asignación de recursos lógicos y físicos en nombre de las aplicaciones. Es responsable de la gestión de la memoria y las comunicaciones. Como gestor de todos los recursos físicos en este entorno multitarea, el SO monitoriza las peticiones de recursos, y controla el acceso a dichos recursos para aquellos recursos que no puedan ser usados de forma concurrente por más de una aplicación. El SO tiene acceso a toda la memoria, las interrupciones y los recursos hardware.

El SO también proporciona una función de monitorización de salud (*health monitoring*) para detectar fallos en la integridad del hardware y software. Como dicha función es específica al diseño del procesador central, es recomendable que esta función esté dividida en una parte residente en el SO y una tabla de estrategia de recuperación.

El SO está delimitado por la interfaz APEX para permitir intercambiabilidad del software de aplicación, y también por la COEX, para permitir intercambiabilidad del núcleo de hardware.

En el apartado 6.3.3.6 se detallan los tres grandes sistemas operativos que soportan particiones en software aeroespacial.

6.3.3 Análisis Detallado de la Arquitectura Software IMA

Hasta este apartado, se ha dado una visión general del sistema IMA y sus diferentes componentes. En este nuevo apartado, se describirá con más detalle las principales características de la arquitectura software IMA, así como una descripción de la interfaz APEX. Se describirán los principales requisitos de dicha interfaz entre el software de aplicación y el SO, así como la lista de servicios que permiten al software de aplicación controlar la planificación, comunicación y la información de estado de sus elementos internos de procesamiento. No es objeto de este trabajo el describir toda la interfaz APEX, que viene

totalmente detallada en el estándar ARINC 653 [21], sino de dar una descripción de sus objetivos y funciones principales.

6.3.3.1 Particionamiento IMA y Descomposición Software

El propósito principal de un módulo núcleo en un sistema IMA es implementar una o más aplicaciones de aviónica y permitir la ejecución independiente de estas aplicaciones. Esto se puede conseguir fácilmente mediante el particionamiento, es decir, una separación funcional de las aplicaciones de aviónica, generalmente para conseguir aislamiento y contención de fallos (evitar que un fallo ocurrido en una partición pueda propagarse a otras) y facilitar la verificación, validación y certificación.

La unidad de particionamiento se conoce como partición. Una partición es básicamente lo mismo que un programa en un entorno de aplicación única: consta de datos, su propio contexto, atributos de configuración, etc. Para aplicaciones mayores, el concepto de particiones múltiples relativo a aplicaciones únicas está contemplado.

Como ya se ha visto en la introducción a la arquitectura software IMA, el software que reside en la plataforma hardware viene descrito en [20] y consta de:

a. Particiones de aplicación, que son las porciones software específicas a las aplicaciones de aviónica que están implementadas en el módulo núcleo. Este software está especificado, desarrollado y verificado de acuerdo con el nivel de criticidad apropiado para la aplicación de aviónica. Estas particiones está sujetas a un particionamiento robusto en espacio y tiempo, y están restringidas a usar sólo llamadas a la interfaz al sistema tipo ARINC 653.
b. Un kernel del SO que proporciona una API y unos comportamientos bien definidos, y soporta un entorno común y estándar en el cual el software de aplicación se ejecuta. Esto puede incluir interfaces hardware tales como drivers de dispositivos y funciones *built-in-test* (función de autotest incluida en el sistema).
c. Particiones del sistema, que son particiones que requieren interfaces fuera del alcance de los servicios APEX. Estas particiones pueden realizar funciones tales como gestión de las comunicaciones desde dispositivos hardware o esquemas de gestión de fallos. Estas particiones de sistema son opcionales y específicas a la implementación del módulo núcleo.
d. Funciones específicas del sistema que pueden incluir interfaces tales como drivers de dispositivos, depurado de fallos y funciones *buil-in-test*.

La interfaz APEX [21] puede verse desde el punto de vista del software de aplicación como una especificación de lenguaje de alto nivel (HOL, High Order Language), y desde el punto de vista del SO como una definición de parámetros y mecanismos de entrada (por ejemplo, llamadas de excepciones). APEX puede incluir una capa que traduzca de la especificación HOL a los mecanismos de entrada apropiados; dicha traducción depende de la implementación del SO, plataforma hardware e incluso el compilador usado para el software de aplicación. Si se utilizan rutinas de librerías, deben ser incluidas dentro del código de la aplicación para preservar un particionamiento robusto.

En la Figura 49 se muestra la relación entre los componentes del módulo núcleo: [22]

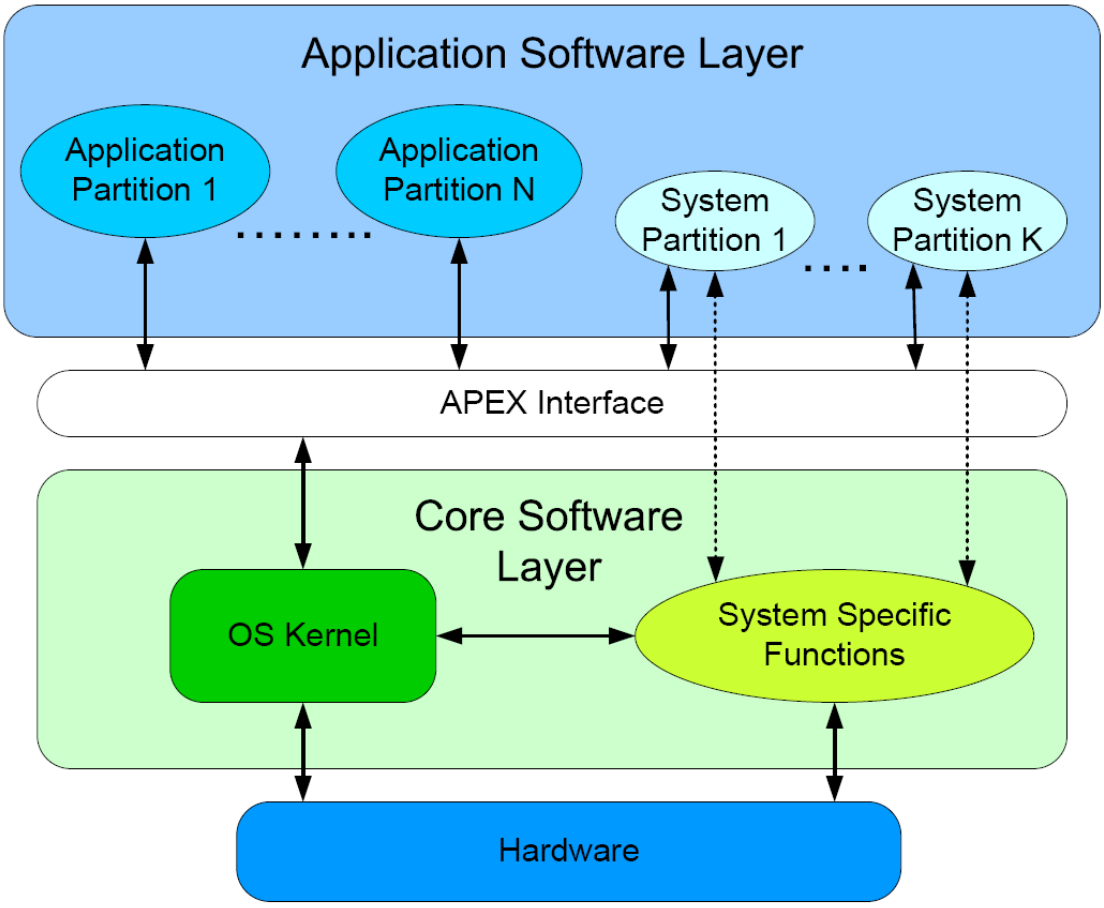


Figura 49. Relaciones entre Componentes del Módulo Principal. [22]

La interfaz APEX, por lo tanto, proporciona un entorno lógico común para el software de aplicación. Este entorno permite a las aplicaciones desarrolladas de manera independiente poder ejecutarse juntas en el mismo hardware.

EL principal objetivo de la interfaz APEX es proporcionar una interfaz de propósito general entre el software de aplicación y el SO dentro de IMA, tal y como se describe en [21]. Esta interfaz tiene múltiples ventajas, entre ellas están:

- Satisfacer los requisitos de tiempo real comunes de diferentes versiones de Ada, que es el lenguaje de programación por excelencia en aplicaciones software aeronáuticas.
- Desacoplar el software de aplicación de la arquitectura del procesador real. Por tanto, los cambios hardware serán trasparentes al software de aplicación.
- La especificación de la interfaz APEX es independiente del lenguaje utilizado. Esta es una condición necesaria para soportar software de aplicación escrito en deferentes lenguajes de alto nivel.
- Portabilidad: el software de aplicación desarrollado para una determinada plataforma puede portarse fácilmente a otra.
- Reusabilidad: la interfaz APEX facilita la producción de código de aplicación reusable para sistemas IMA.
- Modularidad: al eliminar dependencias hardware y software, la interfaz APEX reduce el impacto en el software de aplicación de los cambios en todo el sistema.

- Integración de software de criticidades múltiples: la interfaz PAPEX soporta la integración de múltiples aplicaciones que tengan diferentes niveles de criticidad.

6.3.3.2 Descripción Detallada de la Arquitectura Software IMA

En la Figura 50 se muestra la arquitectura general de un sistema IMA.

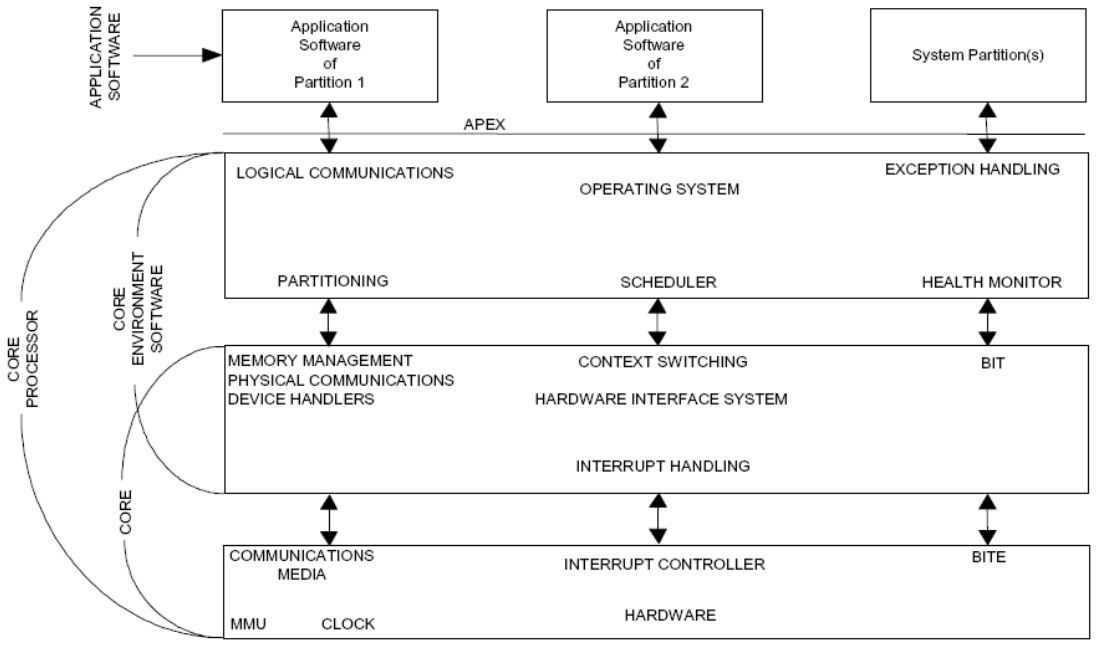


Figura 50. Arquitectura de Software de Sistema IMA. [21].

Cada módulo núcleo puede contener uno o más procesadores individuales. La arquitectura del módulo núcleo tiene influencia en el SO pero no en la interfaz APEX usada por el software de aplicación.

Los procesos dentro de una partición pueden ser distribuidos sobre múltiples procesadores. Para alcanzar la funcionalidad requerida y conforme a las restricciones de tiempo especificadas, los procesos que constituyen una partición pueden operar de manera concurrente. El SO proporciona servicios para controlar y soportar el entorno operacional para todos los procesos dentro de una partición.

Cada partición consta de uno o más procesos de ejecución concurrente, compartiendo acceso a los recursos del procesador, basado en los requisitos de la aplicación. Todos los procesos son identificables de manera única, teniendo atributos que afectan a la planificación, sincronización y ejecución general.

Para asegurar la portabilidad, la comunicación entre particiones es independiente de la localización de la fuente y del destino. Una aplicación que mande o reciba un mensaje a/de otra aplicación, no contendrá información explícita de la localización de dicha aplicación, almacenándose dicha información en tablas de configuración.

El SO, a nivel de módulo núcleo, gestiona las particiones y la comunicación entre particiones (ya sea comunicación dentro del mismo módulo o entre módulos). A nivel de partición, el SO

gestiona los procesos dentro de una partición, así como la comunicación entre los procesos de dicha partición (comunicación dentro de la partición).

6.3.3.3 Funcionalidad del Sistema

6.3.3.3.1 Hardware

Como se detalla en [21], para aislar múltiples particiones en un entorno de recursos compartidos, el hardware proporciona al SO la habilidad para restringir espacios de memoria, tiempo de procesamiento y acceso a las I/O para cada partición individual.

La generación de interrupción temporal para las particiones es deterministica. Cualquier interrupción requerida por el hardware es servida por el SO. El particionamiento en tiempo no se ve perturbada por el uso de otras interrupciones.

De manera general, las condiciones que cumple el procesador son:

- El procesador proporciona suficiente poder de procesamiento para cumplir los requisitos de tiempo en el caso peor.
- El procesador tiene acceso a los recursos de memoria e I/O requeridos.
- El procesador tiene acceso a los recursos de tiempo para implementar los servicios de tiempo.
- El procesador proporciona un mecanismo para trasferir control al SO si la partición intenta realizar una operación invalida.
- El procesador proporciona operaciones atómicas para implementar construcciones de control de procesamiento. Estas operaciones inducirán algo de *jitter* en las porciones de tiempo, y se espera que tengan efectos nulos en la planificación.

6.3.3.3.2 Sistema Operativo: Gestión de las Particiones

Es el concepto fundamental de la filosofía ARINC 653, según el cual las funciones residentes en un módulo núcleo están particionadas con respecto al espacio (particionamiento de memoria) y tiempo (particionamiento temporal), tal y como se describe en [21].

El SO proporciona particionamiento robusto, el cual permite a particiones con diferentes niveles de criticidad ejecutarse en el mismo módulo núcleo, sin afectarse unas a otras espacial o temporalmente. El particionamiento temporal y espacial se explica en los siguientes párrafos.

- ***Particionamiento Temporal:***

Las particiones son planificadas con una base fija y cíclica. El SO mantiene un plazo de ejecución principal de duración fija, el cual se repite periódicamente a través de la operación de *runtime* del módulo. Las particiones se activan asignándoles una o más ventanas dentro de este plazo de ejecución principal. Así, se utiliza una metodología de planificación deterministica por la cual a las particiones se les asigna una predeterminada cantidad de tiempo para acceder a los recursos del procesador de manera ininterrumpida. La tabla de configuración para la planificación de una partición definirá el plazo de tiempo principal y el orden de activación de las ventanas de tiempo para cada partición.

- ***Particionamiento Espacial:***

Cada partición tiene asignadas unas predeterminadas áreas de memoria, según los requisitos que tengan, y varían en tamaño y derechos de acceso. El particionamiento de memoria se consigue prohibiendo accesos de memoria fuera de las áreas de memoria definidas para una partición. Por ejemplo, una partición no tendría derechos de escritura en áreas de memoria no asignadas a la misma.

En el capítulo 8 se analizará más en detalle con ejemplos el particionamiento espacial y temporal.

6.3.3.3.2.1 Definición de Atributos de una Partición

Los atributos de una partición habilitan al SO para controlar y mantener la operación de la partición.

Hay de dos tipos: [21]

- Atributos fijos
  - Identificador
  - Requisitos de memoria: define los límites de memoria para una partición
  - Período: define el período de activación de la partición
  - Duración: la cantidad de tiempo de proceso asignada a la partición dentro del período
  - Nivel de criticidad
  - Requisitos de comunicación: recopila aquellas particiones y/o dispositivos con los cuales la partición se comunicará.
  - Tabla de Monitor de Salud de la partición: recoge instrucciones para el *Health Monitoring* sobre las acciones requeridas.
  - Punto de entrada: dirección de reinicio de la partición
  - Partición del sistema: identifica si la partición es de sistema
- Atributos variables
  - Nivel de bloqueo
  - Modo de operación: el modo de operación de la partición
  - Condición de inicio: condición por la cual la partición es iniciada

6.3.3.3.2.2 Modos de la Partición

Cada partición, durante su ejecución, puede encontrarse en uno de estos modos: [21]

- IDLE: la partición no está ejecutando ninguno de sus procesos dentro de su ventana de tiempos, ni está inicializada.
- NORMAL: el planificador de procesos está activo, y todos los procesos han sido creados para ser ejecutados.
- COLD_START: la fase de inicialización está en progreso y el planificador de procesos está inhibido.
- WARM_START: similar al COLD_START pero el entorno inicial puede ser diferente, por ejemplo no hay necesidad de copiar código desde la memoria no volátil a la RAM.

6.3.3.3.2.3 Planificación de Particiones

En base a la configuración de las particiones dentro del módulo núcleo, requisito/disponibilidad de recursos generales, y requisitos de la partición específicos, se genera una planificación de activación basada en el tiempo, e identifica las ventanas de partición asignadas a particiones individuales. Cada partición es planificada según su respectiva ventana.

Las principales características del modelo de planificador de particiones según [21] son:

- La unidad de planificación es una partición
- Las particiones no tienen prioridad
- El algoritmo de planificación es predeterminado, repetitivo con una periodicidad fija, y es configurable sólo por el integrador del sistema. Al menos una ventana de partición se asigna a cada partición durante cada ciclo.
- El SO a nivel del módulo núcleo controla exclusivamente la asignación de recursos a la partición

6.3.3.3.3 Gestión de Procesos

Una partición contiene uno o más procesos que combinan dinámicamente para proporcionar las funciones asociadas a dicha partición. Dichos procesos pueden operar de manera concurrente, por tanto la partición puede soportar múltiples procesos. [21]

Una aplicación requiere ciertas capacidades de planificación del SO para controlar de forma precisa la ejecución de sus procesos de manera que se satisfagan los requisitos de la aplicación. Los procesos pueden ser diseñados para una ejecución periódica o aperiodica. La ocurrencia de un fallo puede requerir que algunos procesos sean reinicializados o terminados, y se requiere un método para impedir la replanificación de procesos, para acceder de manera segura a los recursos que demandan acceso mutuamente exclusivo.

Un proceso es por tanto una unidad de programación contenida dentro de una partición que se ejecuta concurrentemente con otros procesos de la misma partición. Consta del programa ejecutable, aéreas de datos y pila, contador de programa, y otros atributos, tales como prioridad y tiempo límite.

El SO a nivel de partición es responsable de gestionar los procesos individuales dentro de una partición

El acceso a las funciones de gestión de procesos se realiza vía utilización de los servicios APEX.

Cabe destacar que, al igual que las particiones, los procesos tienen una serie de atributos y de estados para su gestión y control. Dichos atributos no se describirán en este trabajo, pues es información de muy bajo nivel que queda fuera del objetivo principal del estudio.

6.3.3.3.3.1 Control de Procesos

Los procesos son creados e inicializados durante la inicialización de la partición. Una partición es capaz de reinicializar o terminar sus procesos en cualquier momento, y también es capaz de impedir que un proceso se convierta en elegible para recibir los recursos del procesador. [21]

Cada proceso tiene un nivel de prioridad, y su comportamiento puede ser síncrono o asíncrono. Ambos tipos de procesos pueden coexistir dentro de una partición. Cualquier proceso puede ser interrumpido de la pila de ejecución por otro proceso que tenga un nivel de prioridad mayor.

La ventaja de crear los procesos y mecanismos al inicializar el sistema es que se pueden establecerse que los procesos requeridos para una partición tendrán siempre los recursos necesarios, espacio de pila, etc. Además, la función de gestión de memoria del SO es mucho más directa. Al definir los requisitos de gestión de memoria de una partición y sus procesos durante la inicialización, el sistema puede proporcionar protección a nivel de proceso de los recursos.

Otra ventaja considerable de crear los procesos solo durante la inicialización del sistema es que se evita de este modo la excesiva fragmentación de memoria.

6.3.3.3.3.2 Planificación de Procesos

Las principales características del modelo de planificación de procesos usados a nivel de partición son: [21]

a. La unidad de planificación es un proceso APEX. El SO arbitra la competición que se produce dentro de una partición cuando varios procesos quieren tomar el control del procesador
b. Cada proceso tiene una prioridad
c. El algoritmo de planificación está basado en la prioridad de los procesos. El SO gestiona la ejecución de los procesos dentro de la partición.
d. La planificación de procesos periódicos y aperiódicos está soportada por el SO.
e. Todos los procesos dentro de una partición comparten los recursos asignados a dicha partición

6.3.3.3.4 Gestión de Tiempo

El SO proporciona división del tiempo en períodos para planificación de particiones, límite de ejecución, periodicidad, y retardos para planificación de procesos, y *time-outs* para la comunicación entre particiones y dentro de la partición con el objetivo de gestionar el tiempo. Estos mecanismos se definen a través de atributos o servicios. [21]

A cada proceso se le asigna una capacidad de tiempo, que representa el tiempo de respuesta que se le da al proceso para satisfacer sus requisitos de procesamiento.

En tanto que un proceso realice todo su procesamiento sin usar su capacidad de tiempo por completo, el tiempo límite se cumple. Si su procesamiento requiere más de la capacidad de tiempo, el tiempo límite ha fallado.

6.3.3.3.5 Asignación de Memoria

Las particiones, y por tanto sus espacios de memoria asociados, se definen durante la configuración del sistema. No hay servicios de asignación de memoria en la interfaz APEX. [21]

6.3.3.3.6 Comunicación entre Particiones

La comunicación entre particiones se puede dar entre particiones que estén en el mismo módulo núcleo o en diferentes módulos. [21]

Toda comunicación entre particiones se realiza mediante mensajes. Un mensaje se define como un bloque continuo de datos de longitud finita. Un mensaje se envía desde una fuente única a uno o más destinos. Esta comunicación via mensajes esta soportada por la interfaz APEX.

El mecanismo para establecer un enlace entre particiones a través de mensajes es el canal. Un canal define un enlace lógico entre una fuente y uno o más destinos.

Las particiones tienen acceso a los canales vía puntos de acceso definidos llamados puertos. Un canal consta de uno o más puertos y de los recursos asociados.

Los canales y los puertos se definen completamente en el momento de la configuración del sistema.

6.3.3.3.7 Comunicación dentro de las Particiones

Para la comunicación entre los diferentes procesos dentro de una partición, se establecen una serie de mecanismos, como buffers, blackboards, semáforos y eventos. Los dos primeros son para la comunicación entre procesos, y los dos últimos son para la sincronización entre procesos. [21]

6.3.3.3.8 Flujo de Datos entre Particiones y Mecanismos de Protección

Los buffers usados para escribir y leer los mensajes no deben ser visibles a ninguna partición individual, pues esto comprometería el particionamiento en espacio. Se usan varios diseños para implementar este requisito. [20]

6.3.3.3.8.1 Copia Directa a través del Kernel

En esta aproximación, tal y como se describe en [20], el kernel actúa en nombre del escritor. Como se muestra en la Figura 51, el mensaje se pone en la memoria asignada a la partición que escribe en el instante de tiempo correspondiente a la llamada de escritura de mensaje. Cuando el mensaje es leído por la partición receptora, el kernel lo transfiere desde el área del buffer de escritura al área del buffer de lectura.

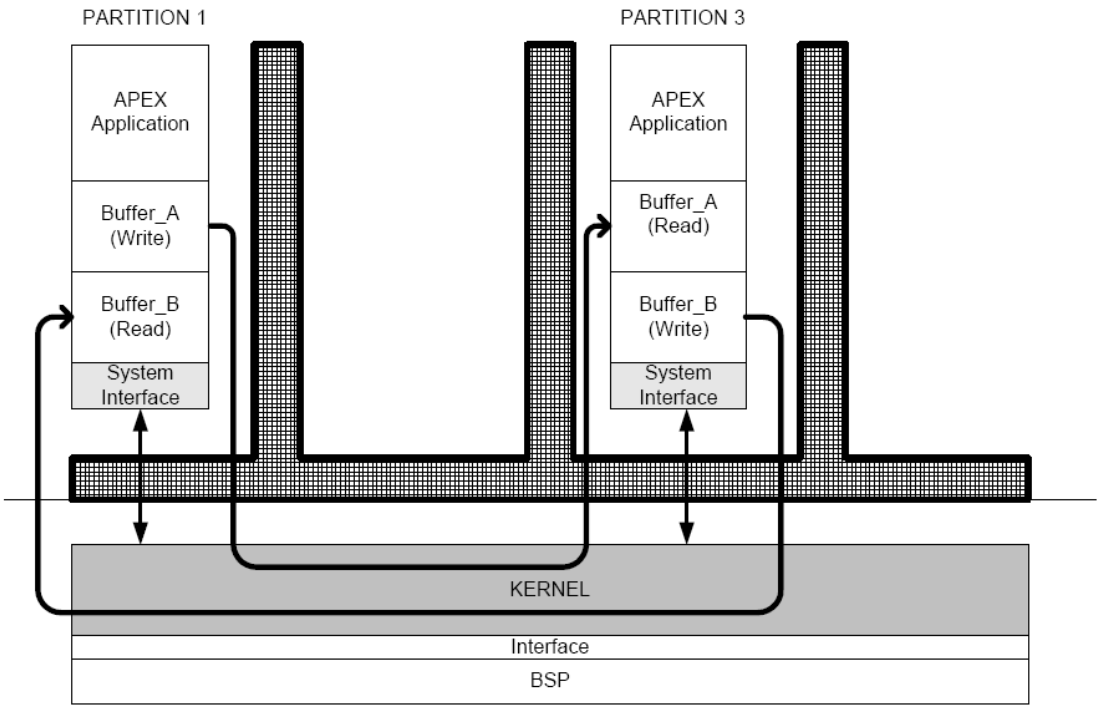


Figura 51. El Kernel copia desde el Espacio de una Partición al de otra directamente. [20]

6.3.3.3.8.2 Copia Indirecta a través del Kernel

Esta implementación descrita en [20] es más preferible cuando el kernel implementa el MOS y la partición contiene el POS. En la Figura 52 se muestra esta aproximación. El mensaje es trasferido desde la partición a un buffer en el kernel, y después es trasferido desde este buffer intermedio al buffer destino. La ventaja de esta aproximación es que el comportamiento de la gestión del buffer está muy simplificado.

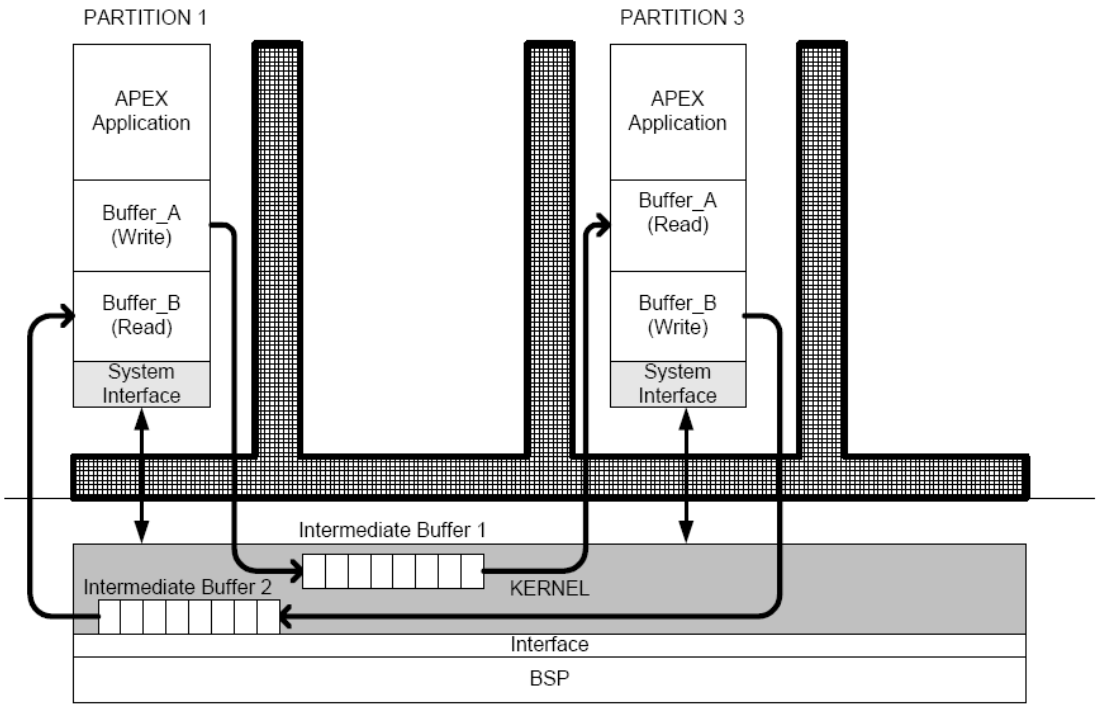


Figura 52. Copias en Buffer a través del Buffer del Kernel. [20]

6.3.3.3.8.3 Copia Cero Síncrona

En este modelo descrito en [20], los buffers se sitúan en un área de memoria que se remapea cuando una partición se activa. En el ejemplo de la Figura 53, cuando la partición 1 está corriendo, un área de lectura/escritura se convierte en direccionable para contener el Buffer_A, y un área de sólo lectura se mapea para contener el Buffer_B. Cuando la partición 3 se activa, las protecciones de paginado de memoria se invierten en las áreas comunes. Este modelo tiene la ventaja de que los mensajes grandes no necesitan ser copiados físicamente.

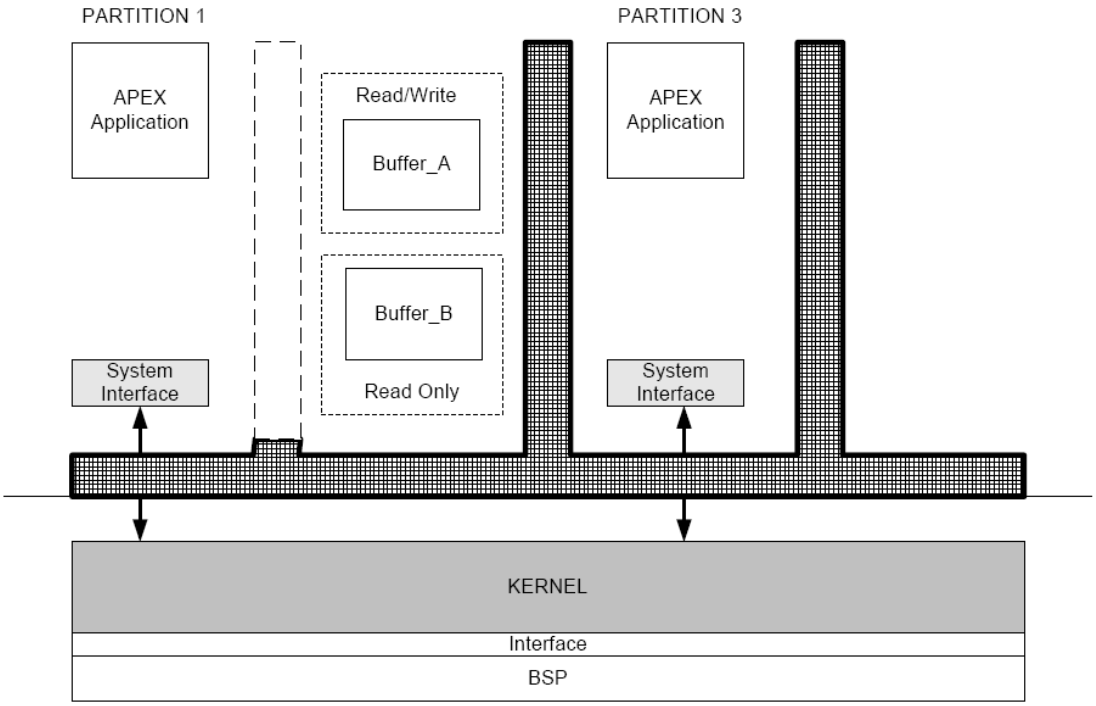


Figura 53. Acceso a través de Manipulación del Mecanismo de Protección. [20]

6.3.3.3.8.4 Copia Cero Asíncrona.

Este modelo descrito en [20] es similar al anterior, sólo que el mecanismo de protección sólo se usa cuando hay una petición de lectura de mensaje. En el ejemplo anterior, un mensaje se puede enviar desde la partición 1 al Buffer_A. un cambio a la partición 3 no resultaría en que la memoria sea remapeada para el área del buffer. Si el código en la partición 3 intentase leer el mensaje del buffer_A, el hardware lanzaría una excepción de acceso a memoria. Esta excepción es interceptada por el kernel, el cual chequea la tabla de configuración del sistema para ver si el acceso a memoria debería ser permitido. Si el acceso es permitido, entonces la memoria seria remapeada de forma que la lectura desde el buffer continuaría.

6.3.3.4 Monitor de Salud

El monitor de salud (HM, *Health Monitor*) descrito en [21] es la función del SO responsable de la monitorización y el informe de fallos de hardware y software de aplicación y del SO. El HM ayuda a aislar fallos e impedir que dichos fallos puedan propagarse. Componentes del HM están contenidos dentro de los siguientes elementos software:

- Núcleo del SO
- Particiones de aplicación
- Particiones del sistema.

6.3.3.5 Consideraciones acerca de la Configuración

La configuración es implementada por el integrador de todo el sistema, que asignara particiones a los módulos núcleos. La configuración resultante permitirá a cada partición el acceso a sus recursos requeridos y asegurara que los requisitos de disponibilidad e integridad de cada aplicación se satisfacen. Por tanto, los requisitos de tiempo, uso de memoria e interfaces externas deben ser conocidos por el integrador del sistema, para cada partición que tenga q ser integrada. [21]

La información de configuración requerida por el integrador es:

- Requisitos de memoria
- Periodo
- Duración
- Identidad de los mensajes a ser enviados por la partición
- Identidad de los mensajes a ser recibidos por la partición

La localización de las particiones en los diferentes módulos dictamina las rutas de los mensajes y por tanto el integrador debe también proporcionar el mapeo entre los nodos en los caminos de los mensajes.

Las tablas de configuración son requeridas por el SO para asegurar que el sistema está completo durante la inicialización, y para habilitar las comunicaciones entre particiones. Estas tablas de configuración son aéreas de datos estáticos accedidos por el SO. Estas no pueden ser accedidas directamente por las aplicaciones, y no son parte del SO.

Para describir los datos de configuración, se utiliza XML.

Hay diferentes tablas de configuración; las más importantes son:

- Tablas de configuración para la inicialización del sistema
  El SO asegura de que la configuración de las particiones es correcta mediante la referencia a una tabla de configuración, que contendrá los identificadores de la partición que residen en un módulo, los requisitos de memoria de cada partición y el número y tamaño de los puertos requeridos por la partición
- Tablas de configuración para comunicación entre particiones
  Cuando el SO detecta que se ha escrito un mensaje en un puerto entonces refiere a una tabla de configuración que contiene el mapeado desde el puerto identificado

hasta el siguiente puerto dentro del canal, este puerto puede ser un puerto en otra partición del módulo o una dirección en la memoria entre módulos.

- Tablas de configuración para Monitor de Salud
  El HM usa tablas de configuración para manejar los fallos que se produzcan, estas tablas son la tabla HM de sistema, la tabla HM de módulo y la tabla HM de partición. Para todas ellas se define la acción de recuperación ante un fallo o error crítico.

6.3.3.6 Algunos Ejemplos de Implementaciones IMA

En este apartado se verán algunos ejemplos de implementaciones de arquitectura IMA a través de soluciones comerciales de sistemas operativos.

6.3.3.6.1 VxWorks 653 Partition Operating System

Desarrollado por Wind River, soporta el estándar ARINC 653 con particiones espaciales y temporales, como se verá más adelante, además de soporte a aplicaciones heterogéneas, como se ve en la Figura 54: [23]

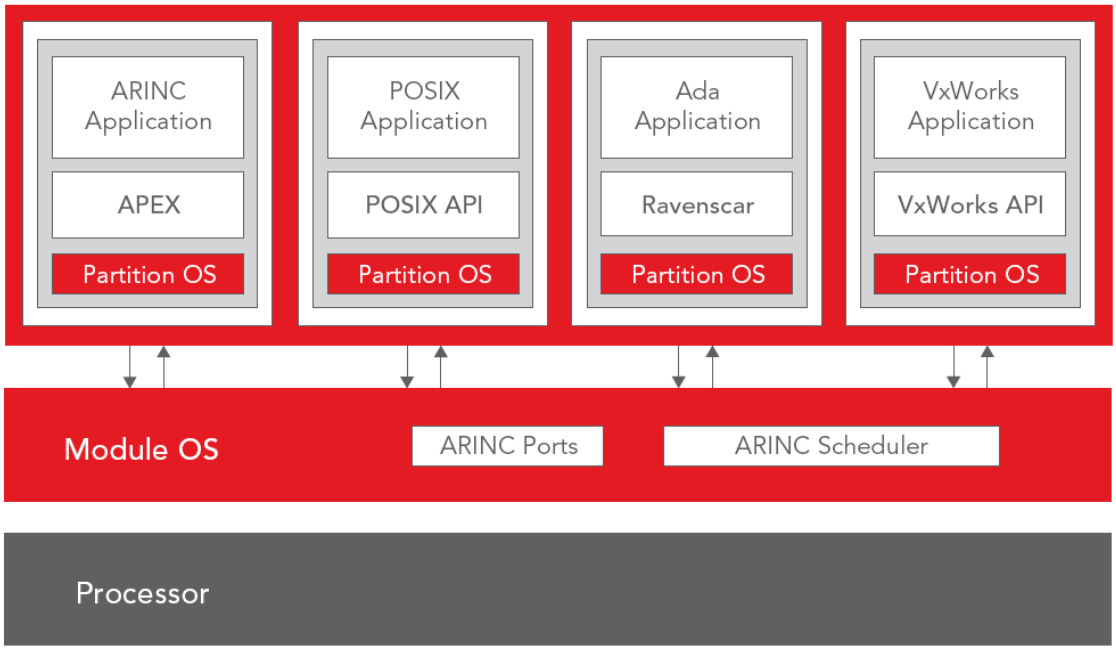


Figura 54. Soporte a Aplicaciones Heterogéneas del SO VxWorks 653. [23]

6.3.3.6.2 LynxOS-178

Sistema Operativo en tiempo real con soporte ARINC 653, particionado en tiempo y espacio, basado en estándares abiertos estilo UNIX. En la Figura 55 se puede ver su arquitectura: [24]

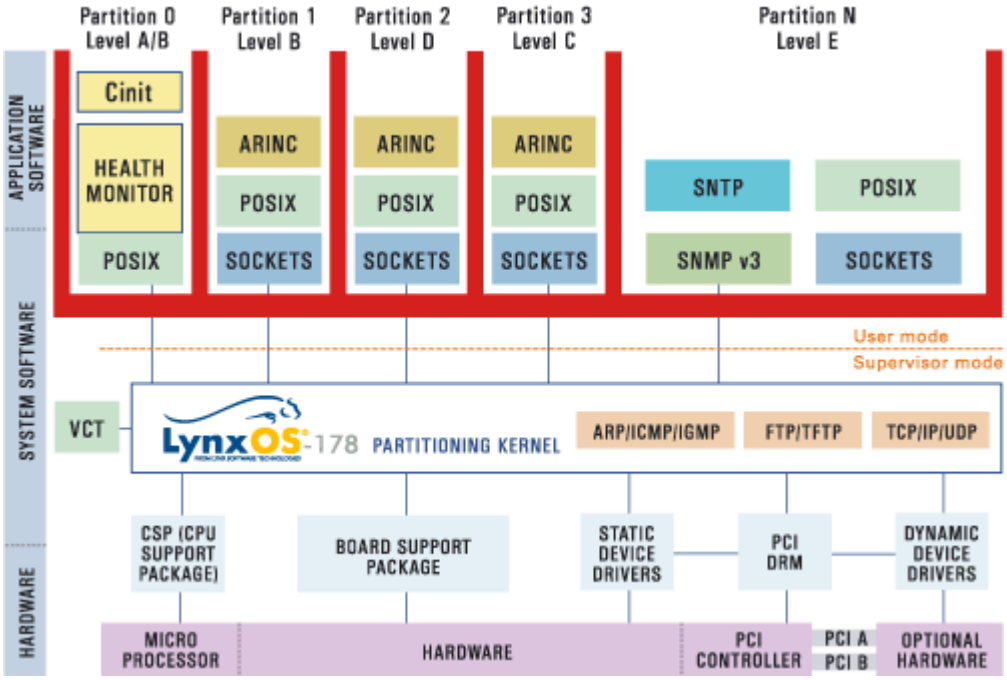


Figura 55. Arquitectura ARINC 653 con el SO LynxOS-178. [24]

6.3.3.6.3 INTEGRITY-178B Operating System

Desarrollado por Green Hills, al igual que los anteriores, ofrece soporte completo a ARINC 653 APEX interface.

En la Figura 56 se puede ver su arquitectura: [25]

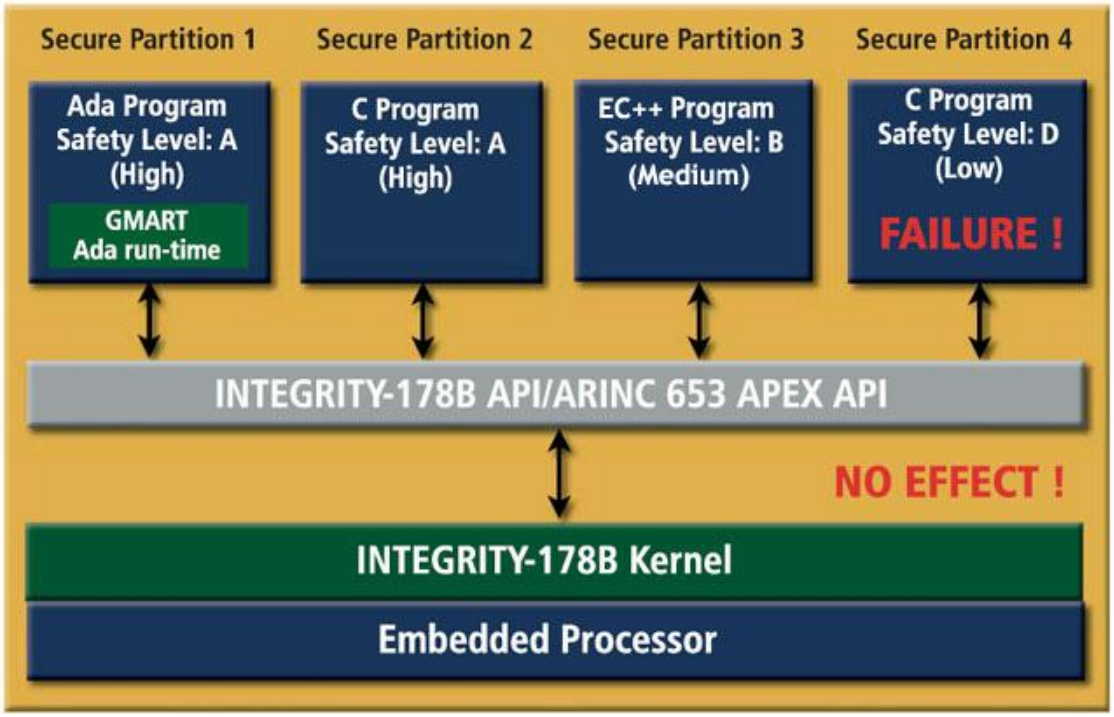


Figura 56. Arquitectura IMA ARINC 653 con el SO Integrity-178B. [25]

6.3.3.6.4 Arquitectura del Sistema AIR

Es una solución sencilla, destallada en [22], para proporcionar la funcionalidad ARINC 653 que falta en los kernels RTOS (*Real Time Operating Systems*) comerciales, tales como los *Real Time Executive for Multiprocessors Systems* o RTEMS. Esta solución implica encapsular dichas funciones en componentes con una interfaz bien definida y añadirlas a la arquitectura básica del sistema operativo.

En la Figura 57 se puede observar en detalle esta arquitectura. [22]

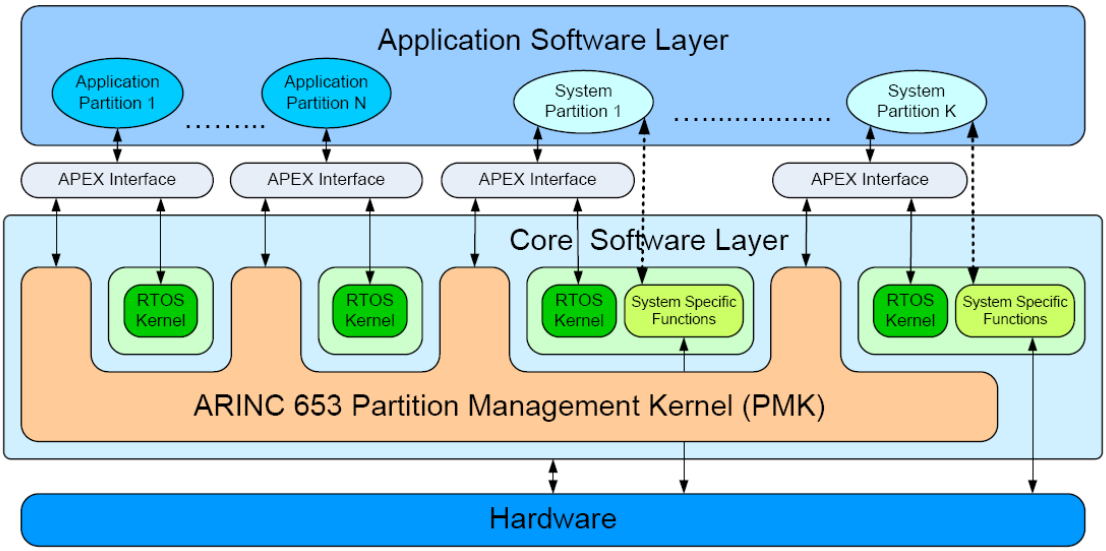


Figura 57. Arquitectura del Sistema AIR. [22]

En esencia, la arquitectura AIR preserva la independencia del hardware y el RTOS definida dentro del ámbito de la especificación ARINC 653 [21]. Hay un módulo específico que se necesita añadir al kernel del RTOS (por ejemplo, a un RTEMS), llamado AIR Partition Management Kernel (PMK), que incluye las siguientes funciones:

- Planificador de particiones AIR: asigna ventanas de tiempo a las particiones para que éstas utilicen los recursos e infraestructura del sistema (procesador, acceso a memoria, etc.). Asegura la segregación temporal usando un único planificador cíclico fijo.
- Despachador de particiones AIR: es el responsable de salvar el contexto de ejecución de la partición que se está ejecutando y de restaurar dicho contexto para la partición que la sucede. Asegura la gestión de todas las provisiones requeridas para garantizar la segregación espacial.
- Módulo de Comunicación entre particiones AIR: permite el intercambio de información entre las diferentes particiones sin violar las restricciones de segregación espacial.

Otro componente fundamental está relacionado con la interface APEX (APplication Executive) definida en ARINC 653., que define para cada partición del sistema un conjunto de servicios en estricta conformidad con el estándar ARINC 653. Está diseñada tanto como sea posible mapeando los servicios ARINC 653 dentro de las nativas y/o primitivas POSIX del RTOS.

Un aspecto a destacar de esta arquitectura es el particionamiento robusto y la componibilidad, descritos en [22]; el particionamiento robusto comprende la protección del espacio de direccionamiento de memoria de cada partición, que es proporcionado por mecanismos específicos de protección de memoria, usualmente implementados en una unidad de gestión de memoria hardware (*Memory Management Unit*, MMU). Ello también requiere una protección funcional relativa a la gestión de niveles de privilegios y restricciones a la ejecución de instrucciones con privilegios. Aunque hay espacio para mejoras, existen un conjunto básico de tales mecanismos en la arquitectura Intel IA-32 y, como extensión, en el núcleo del procesador SPARC LEON.

La especificación del estándar ARINC 653 [21] restringe el tiempo de proceso asignado a cada partición, de conformidad con los parámetros de configuración dados. La planificación de las particiones definida por el estándar ARINC 653 es estrictamente determinística sobre el tiempo. Cada partición tiene una ventana temporal fija en la cual tiene el control sobre los recursos de procesamiento. Cada partición está planificada sobre unas bases cíclicas fijas.

En la arquitectura AIR, la segregación temporal está asegurada por el planificador de particiones AIR. Esto deja espacio para la componibilidad temporal de aplicaciones.

Para asegurar flexibilidad y modularidad, en vez de modificar el planificador del RTOS para extenderlo hasta el concepto de particionamiento, la estrategia seguida en la arquitectura AIR utiliza una instancia del planificador nativo del RTOS para planificación de procesos dentro de cada partición. No se necesita una modificación trascendental a la funcionalidad del planificador de procesos del RTOS para su integración en el sistema AIR. Tal estrategia de planificador en dos niveles jerárquicos asegura el desacoplamiento del planificador de procesos y particiones, permitiendo así el uso de diferentes sistemas operativos en diferentes particiones (por ejemplo, RTEMS, eCos, etc.)

7 CAPÍTULO 7 – ANÁLISIS DE ARQUITECTURAS SOFTWARE CON ARCHE

7.1 Introducción

En este capítulo se presentará el análisis realizado con la herramienta ArchE sobre dos modelos de arquitecturas de software aeronáutico: la primera de ellas es un simulador de vuelo, y la segunda es una arquitectura de software de un equipo de aviónica de una aeronave.

La herramienta ArchE es un asistente para el análisis y mejora de arquitecturas software basado en reglas, el cual utiliza modelos de atributos de calidad como marcos de razonamiento [36], [37]. En concreto, utiliza dos marcos de razonamiento:

- Marco de Modificabilidad: mide el impacto de la modificación de un componente de la arquitectura en otros componentes con los cuales tiene algún tipo de relación
- Marco de Rendimiento: mide la respuesta del sistema ante un evento externo, y concluye en si el sistema es planificable o no bajo el punto de vista de la ejecución programada de tareas en un determinado intervalo de tiempo.

En el Anexo 1, Anexo 2 y Anexo 3 se describe la herramienta ArchE con más detalle, además de enumerar las limitaciones que tiene y sus capacidades, junto con una pequeña guía de usuario para poder manejar los marcos de razonamiento con éxito.

7.2 Simulador de Vuelo

7.2.1 Introducción a los Simuladores de Vuelo

Un simulador de vuelo es un dispositivo cuyo objetivo es simular el funcionamiento de una aeronave en cualquier condición de vuelo, para permitir a los pilotos entrenar los procedimientos de operación de dicha aeronave.

Para dicho fin, un simulador de vuelo consta, básicamente, de los siguientes elementos:

- Una réplica de la cabina de la aeronave a simular, incluyendo todos los controles hardware y pantallas multifunción que presentan la información al piloto
- Un sistema visual, que representa el entorno aéreo y terrestre por el cual la aeronave va a realizar su vuelo.
- Modelos de software de todos los sistemas de la aeronave, así como del modelo de vuelo, simulaciones de entorno, etc.
- Sistema de movimiento, que ejecuta los movimientos de la aeronave en respuesta a las acciones del piloto, desplazando la cabina sobre tres ejes de rotación (alabeo, guiñada y cabeceo).
- Sistema de generación de audio, que genera los sonidos de los motores, condiciones ambientales, comunicaciones de voz del piloto, etc.
- Posición de instructor: sistema hardware/software utilizado por el instructor para generar un escenario de vuelo para el vuelo de entrenamiento. Este escenario incluye: posición de inicio de la aeronave, condiciones iniciales, generación de fallos de sistemas, etc.

En la Figura 58 se muestra un esquema simplificado de los distintos elementos descritos.

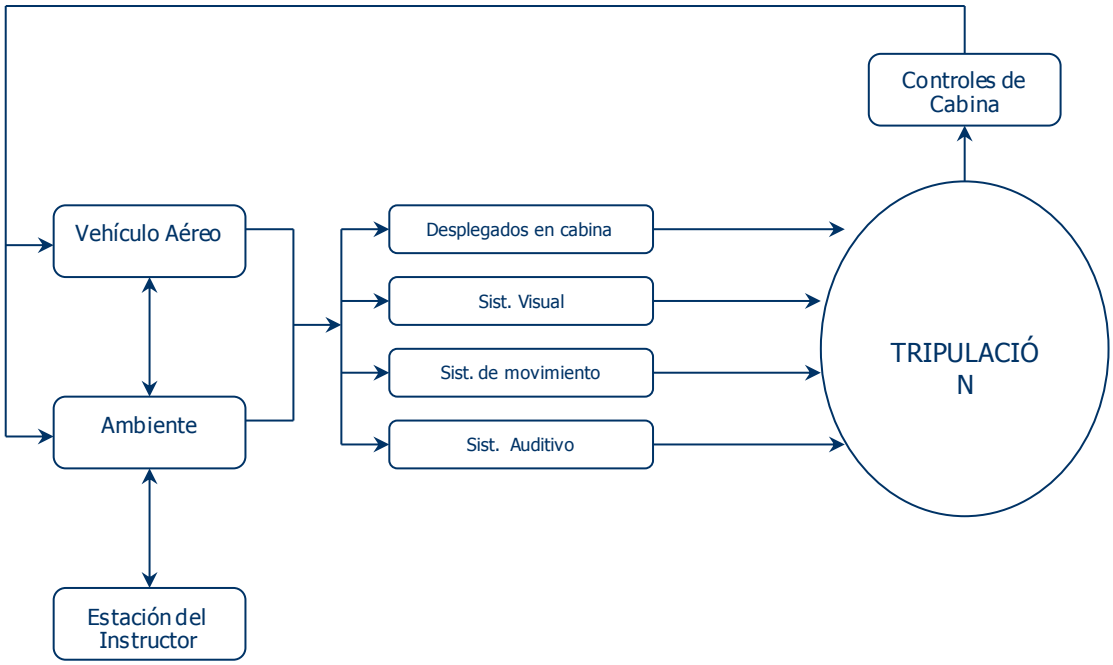


Figura 58. Elementos de un Simulador de Vuelo. [26]

En la Figura 59 se muestra una disposición más real de los diferentes elementos de un simulador de vuelo: ([28] [29] [30] [31] [32] [33] [34] [35])

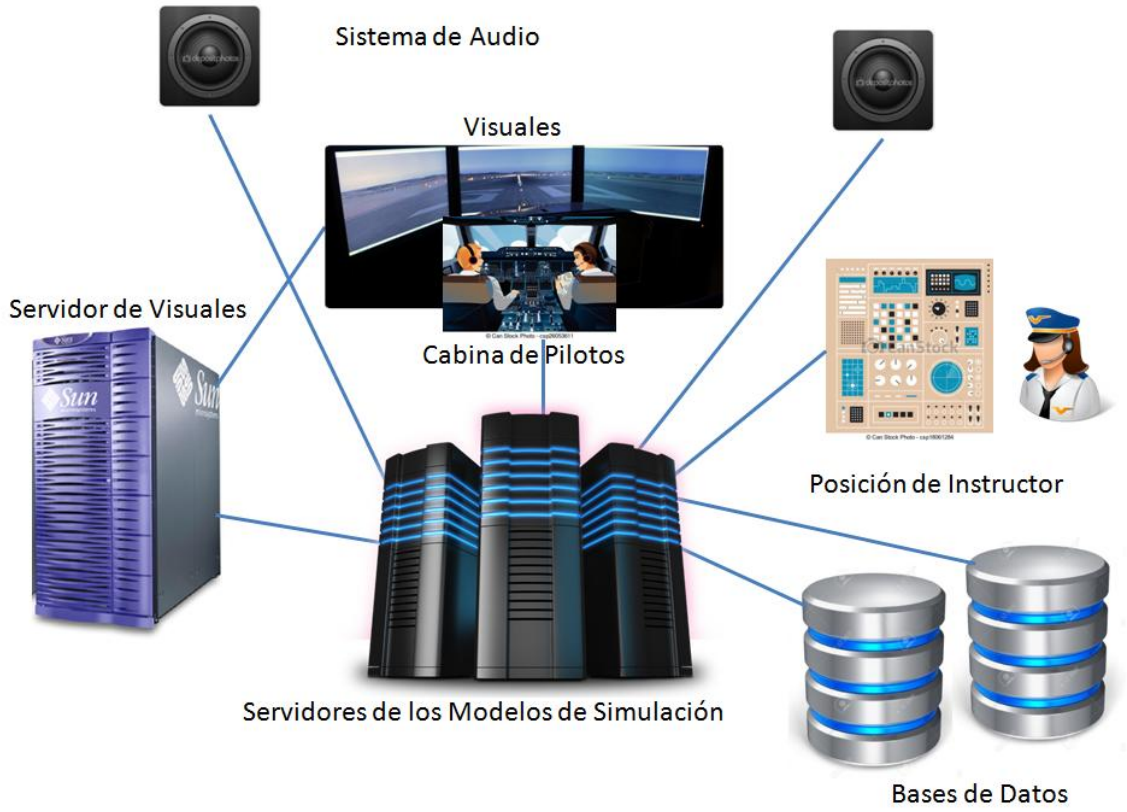


Figura 59. Disposición real de un Simulador de Vuelo.

7.2.2 Arquitectura y Diseño de un Simulador de Vuelo

7.2.2.1 Casos de Uso y Actores principales

En esta sección se definen los casos de uso del simulador de vuelo. El diagrama de casos de uso se muestra en la Figura 60:

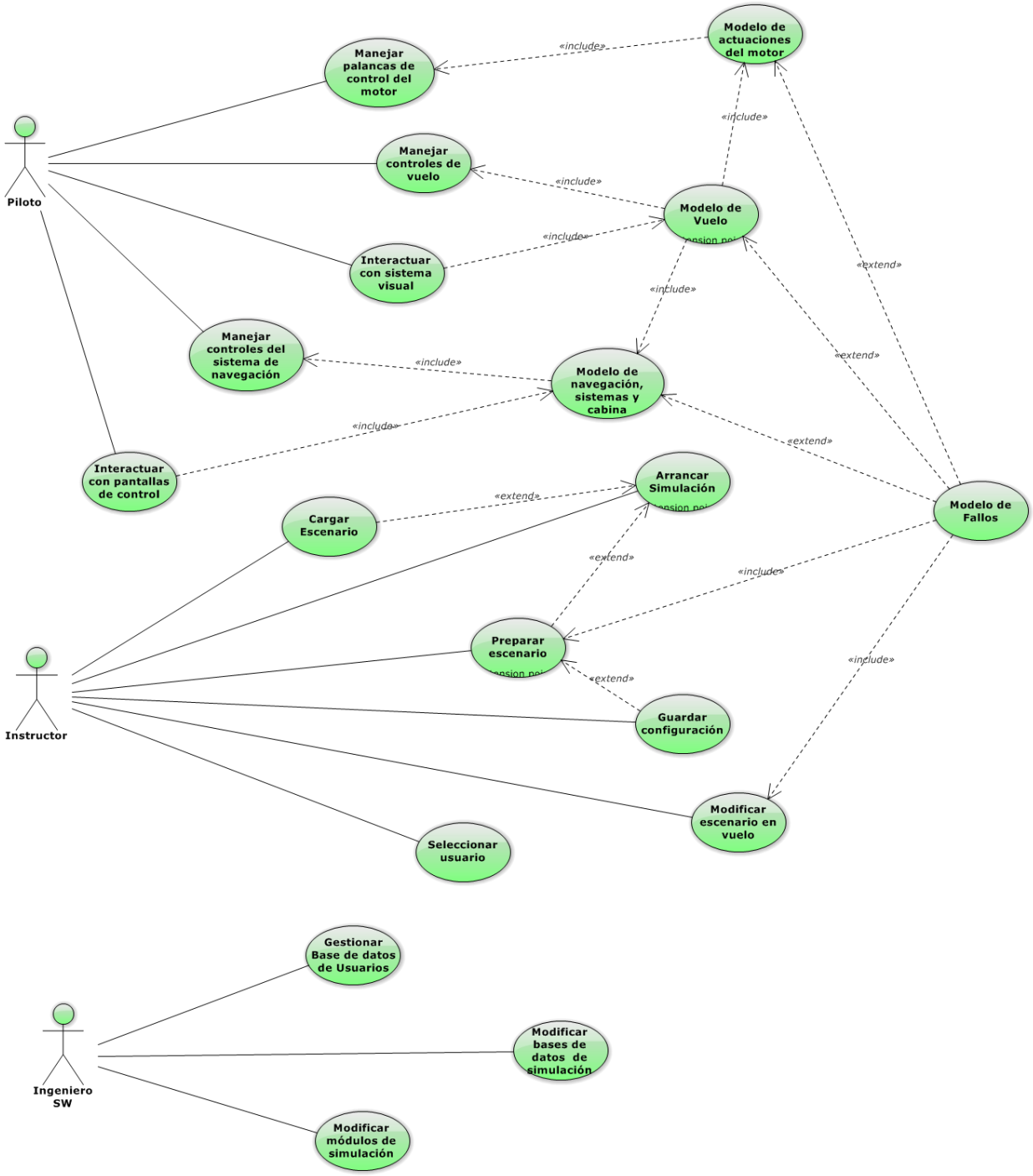


Figura 60. Diagrama de Casos de Uso del Simulador de Vuelo.

A continuación se describen los casos de uso.

CU-01	Manejar Palancas de control del motor
Actor	Piloto
Descripción	Controles en cabina. HW físico. Modelo SW que parametriza la posición de las palancas del motor. Con dicha información, el modelo de actuaciones del motor calculará la potencia del motor.
Condiciones iniciales	La simulación debe estar arrancada.
Condiciones finales	El piloto obtiene la información del estado de los motores en las pantallas de control. El aumento o disminución de la potencia de los motores influyen en controles de vuelo y sistema visual.
Secuencia de ejecución	1.- El piloto interactúa con las palancas de control del motor 2.- El piloto obtiene el *feedback* del modelo de actuaciones del motor.

Tabla 4. CU-01 – Manejar Palancas de Control del Motor.

CU-02	Manejar controles de vuelo
Actor	Piloto
Descripción	Controles en cabina. HW físico. Modelo SW que parametriza la posición de las palancas de control de vuelo. Con dicha información, junto con el modelo de actuaciones del motor y el modelo de navegación, el sistema calculará la posición y actitud del avión, su velocidad, trayectoria, etc.
Condiciones iniciales	La simulación debe estar arrancada. El piloto debe interactuar con el joystick de control.
Condiciones finales	El piloto obtiene la información de la posición y actitud del avión tanto por el movimiento proporcionado por el sistema visual como por el estado de posición en las pantallas de control.
Secuencia de ejecución	1.- El piloto interactúa con las palancas de control de vuelo 2.- El piloto obtiene el *feedback* del modelo de vuelo, a través fundamentalmente del sistema visual.

Tabla 5. CU-02 - Manejar Controles de Vuelo.

CU-03	Interactuar con sistema visual
Actor	Piloto
Descripción	El sistema visual es el responsable del cálculo de toda la representación gráfica de la simulación, incluyendo el propio avión, el escenario o zona geográfica por donde vuela, los elementos con los cuales el avión puede interactuar, las condiciones climatológicas, etc. Una vez calculado en tiempo real, el sistema visual será el encargado de mostrárselo al piloto en la pantalla de proyección.
Condiciones iniciales	La simulación debe estar arrancada.
Condiciones finales	Se actualiza la representación gráfica de forma continua.
Secuencia de ejecución	Se ejecuta en tiempo real y de forma continua.

Tabla 6. CU-03 - Interactuar con sistema visual.

CU-04	Manejar controles del sistema de Navegación
Actor	Piloto
Descripción	Controles en cabina. HW físico. Modelo SW que parametriza la posición de los controles de los sistemas de navegación. Con dicha información, junto con el modelo de actuaciones del motor, modelo de navegación y el modelo de controles de vuelo, el sistema calculará la posición y actitud del avión, su velocidad, trayectoria, etc.
Condiciones iniciales	La simulación debe estar arrancada. El piloto debe interactuar con los controles de navegación en cabina.
Condiciones finales	El piloto obtiene la información de ruta, posición y actitud del avión en las pantallas multifunción de la cabina.
Secuencia de ejecución	1.- El piloto interactúa con los controles de navegación. 2.- El piloto obtiene el *feedback* del modelo de navegación a través de las pantallas multifunción.

Tabla 7. CU-04 - Manejar controles del sistema de Navegación.

CU-05	Interactuar con pantallas de control
Actor	Piloto
Descripción	Controles en cabina. HW físico. Modelo SW que modela la información de navegación, sistemas, alertas de la aeronave, para mostrarlas en las pantallas multifunción de cabina.
Condiciones iniciales	La simulación debe estar arrancada. Las pantallas deben estar encendidas y los modelos de simulación deben de estar iniciados y proporcionando parámetros.
Condiciones finales	El piloto puede ver la información de los sistemas y de navegación en las pantallas.
Secuencia de ejecución	1.- El piloto interactúa con los controles de las pantallas, seleccionando los formatos que desea ver. 2.- El piloto obtiene el *feedback* de los modelos.

Tabla 8. CU-05 - Interactuar con pantallas de control.

CU-06	Modelo de actuaciones del motor
Actor	Otros módulos
Descripción	El modelo de actuaciones del motor calculará, a partir de los *inputs* de los controles de las palancas de gases, el modelo de combustible dentro del modelo de sistema, condiciones de entorno ambiental, altitud, etc, la potencia del motor y/o empuje (dependiendo de si se trata de un turbohélice o motor de reacción).
Condiciones iniciales	La simulación debe estar arrancada. Los controles de palancas del motor deben proporcionar una posición.
Condiciones finales	La potencia/empuje de los motores es calculada y proporcionada al resto de los modelos.
Secuencia de ejecución	1.- El piloto interactúa con las palancas de control de los motores 2.- El modelo de actuaciones de motor proporciona la potencia/empuje.

Tabla 9. CU-06 - Modelo de actuaciones del motor.

CU-07	Modelo de vuelo
Actor	Otros modelos
Descripción	El modelo de vuelo, a partir de las curvas aerodinámicas de la aeronave, los controles de vuelo y el modelo de actuaciones del motor, calcularás los modelos de fuerzas que gobernarán la trayectoria de la aeronave.
Condiciones iniciales	La simulación debe estar arrancada. Los modelos que interactúan con el de vuelo deben de estar inicializados y proporcionando parámetros.
Condiciones finales	Se obtienen las fuerzas que actúan sobre la aeronave y la posición de la misma.
Secuencia de ejecución	1.- Los modelos proporcionan los datos al modelo de vuelo 2.- El modelo de vuelo proporciona la posición de la aeronave.

Tabla 10. CU-07 - Modelo de vuelo.

CU-08	Modelo de navegación, sistemas y cabina
Actor	Otros modelos.
Descripción	El modelo de Navegación, sistemas y cabina proporciona las siguientes funciones: - Establecer la ruta de la aeronave mediante un plan de vuelo - A partir de la posición calculada por el modelo de vuelo, genera los *inputs* para los instrumentos de vuelo de cabina para mostrar la información al piloto - Simula el comportamiento del resto de sistemas de la aeronave: navegación, aviónica, eléctrico, hidráulico, combustible, luces, etc. - Controla el estado de los sistemas de cabina.
Condiciones iniciales	La simulación debe estar arrancada. Los modelos que interactúan con éste deben de estar inicializados y proporcionando parámetros.
Condiciones finales	El modelo de navegación, sistemas y cabina proporciona *inputs* para el resto de modelos.
Secuencia de ejecución	1.- Los modelos proporcionan los datos al modelo navegación, sistemas y cabina 2.- El modelo proporciona inputs para otros modelos.

Tabla 11. CU-08 - Modelo de navegación, sistemas y cabina.

CU-09	Modelo de fallos
Actor	Otros modelos
Descripción	El modelo de fallos es un módulo software que genera fallos en los sistemas de la aeronave, a petición del instructor.
Condiciones iniciales	La simulación debe estar arrancada. Los modelos que interactúan con éste deben de estar inicializados y proporcionando parámetros. El Instructor ha preparado el escenario y/o ha modificado el escenario en vuelo.
Condiciones finales	El modelo de fallos calcula los nuevos parámetros y las condiciones de fallo y se las proporciona al modelo afectado por el fallo.
Secuencia de ejecución	1.- El instructor ha preparado el escenario y/o ha modificado el escenario en vuelo. 2.- El modelo de fallos introduce los fallos en el modelo afectado.

Tabla 12. CU-09 - Modelo de fallos.

CU-10	Cargar escenario
Actor	Instructor
Descripción	El instructor prepara un escenario ya creado y lo carga en el simulador.
Condiciones iniciales	La simulación debe estar sin arrancar.
Condiciones finales	El escenario se ha cargado con éxito.
Secuencia de ejecución	1.- El instructor manipula el escenario desde la posición de instructor. 2.- El escenario aparece cargado y ya se puede inicializar la simulación.

Tabla 13. CU-10 - Cargar escenario.

CU-11	Preparar escenario
Actor	Instructor
Descripción	El instructor crea un nuevo escenario, introduciendo los parámetros de inicialización del vuelo simulado, así como los fallos que se producirán durante el vuelo.
Condiciones iniciales	La simulación debe estar sin arrancar.
Condiciones finales	El escenario se ha cargado con éxito y se ha cargado en el simulador.
Secuencia de ejecución	1.- El instructor manipula el escenario desde la posición de instructor. 2.- El escenario aparece cargado y ya se puede inicializar la simulación.

Tabla 14. CU-11 - Preparar escenario.

CU-12	Modificar escenario en vuelo
Actor	Instructor
Descripción	El instructor modifica el escenario actual, para introducir algún fallo o bien reinicializar la simulación en otro punto geográfico o de tiempo.
Condiciones iniciales	La simulación debe estar arrancada.
Condiciones finales	El escenario se ha modificado con éxito y la simulación prosigue.
Secuencia de ejecución	1.- El instructor manipula el escenario desde la posición de instructor. 2.- El escenario ha sido modificado en determinados parámetros sin afectar al resto de parámetros de la simulación.

Tabla 15. CU-12 - Modificar escenario en vuelo.

CU-13	Guardar configuración
Actor	Instructor
Descripción	Una vez creado un escenario, o bien terminada una simulación, esta función permite guardar en un fichero los parámetros para poder reproducir el escenario, analizarlo y/o cargarlo posteriormente.
Condiciones iniciales	La simulación debe estar sin arrancar.
Condiciones finales	El escenario queda guardado y disponible para su posterior análisis o para cargarlo de nuevo.
Secuencia de ejecución	1.- El escenario ha sido creado o modificado; o la simulación ha terminado. 2.- El instructor guarda el escenario en un fichero.

Tabla 16. CU-13 - Guardar configuración.

CU-14	Arrancar simulación
Actor	Instructor
Descripción	Esta función permite arrancar la simulación, una vez que un escenario ha sido cargado o creado, y el perfil de un piloto ha sido seleccionado.
Condiciones iniciales	La simulación debe estar sin arrancar.
Condiciones finales	La simulación comienza a funcionar.
Secuencia de ejecución	1.- El instructor carga un escenario o bien lo crea. 2.- A continuación, el instructor selecciona el perfil de un piloto de la base de datos. 3.- El instructor configura el resto de parámetros necesarios para arrancar la simulación (por ejemplo, tiempo de simulación, equipos reales o simulados cuando es un *Full Flight Simulator*, etc)

Tabla 17. CU-14 - Arrancar simulación.

CU-15	Seleccionar usuario
Actor	Instructor
Descripción	El instructor debe seleccionar un piloto de una base de datos.
Condiciones iniciales	La simulación debe estar sin arrancar. El instructor debe acceder a la base de datos de pilotos desde una aplicación.
Condiciones finales	El perfil del piloto ha sido cargado con éxito.
Secuencia de ejecución	1.- El instructor abre la aplicación, y accede a la base de datos, seleccionando el alumno piloto que va a recibir el entrenamiento. 2.- El instructor carga el perfil y cierra la base de datos. El perfil está listo para ser usado como parte del escenario de la simulación.

Tabla 18. CU-15 - Seleccionar usuario.

CU-16	Gestionar base de datos de usuarios
Actor	Ingeniero Software
Descripción	Mediante esta funcionalidad es posible crear, modificar o eliminar los perfiles de la base de datos de los pilotos que utilizarán el simulador, para registrar sus estadísticas, datos personales, etc.
Condiciones iniciales	La simulación puede estar arrancada. El servidor no estará disponible para seleccionar un perfil una vez que se haya procedido a la modificación.
Condiciones finales	El perfil ha sido creado/modificado.
Secuencia de ejecución	1.- El ingeniero software inicia sesión en el servidor 2.- El ingeniero software accede a la base de datos tipo MySQL y allí procede a realizar los cambios necesarios. 3.- Los cambios se graban y el ingeniero software sale de la sesión. 4.- El servidor ha de reflejar los cambios en un determinado período de tiempo, antes de que estén disponibles.

Tabla 19. CU-16 - Gestionar base de datos de usuarios.

CU-17	Modificar bases de datos de simulación
Actor	Ingeniero Software
Descripción	Mediante esta funcionalidad es posible crear o modificar los parámetros correspondientes a las bases de datos utilizadas por el simulador. Por ejemplo, bases de datos de terrenos, mapas, aeropuertos, otras aeronaves que pueden aparecer en vuelo, etc.
Condiciones iniciales	La simulación debe estar sin arrancar, ya que los modelos de simulación cogen los datos del servidor.
Condiciones finales	La base de datos ha sido creada/modificada.
Secuencia de ejecución	1.- El ingeniero software abre sesión en el servidor 2.- El ingeniero software accede a la base de datos tipo MySQL y allí procede a realizar los cambios necesarios. 3.- Los cambios se graban y el ingeniero software sale de la sesión. 4.- El servidor ha de reflejar los cambios en un determinado período de tiempo, antes de que estén disponibles.

Tabla 20. CU-17 - Modificar bases de datos de simulación.

CU-18	Modificar modelos de simulación
Actor	Ingeniero Software
Descripción	Mediante esta funcionalidad es posible crear, modificar o eliminar los modelos de simulación del servidor de simulación.
Condiciones iniciales	La simulación debe estar sin arrancar.
Condiciones finales	El modelo de simulación ha sido creado/modificado/eliminado.
Secuencia de ejecución	1.- El ingeniero software inicia sesión en el servidor de simulación. 2.- El ingeniero software accede a la base de datos de los modelos de simulación y allí procede a realizar los cambios necesarios. 3.- Los cambios se graban y el ingeniero software sale de la sesión. 4.- El servidor ha de reflejar los cambios en un determinado período de tiempo, antes de que estén disponibles.

Tabla 21. CU-18 - Modificar modelos de simulación.

Los actores principales se muestran en la Tabla 22: (la tabla no muestra los actores ni casos de uso correspondientes a interacciones entre modelos)

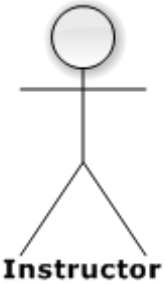
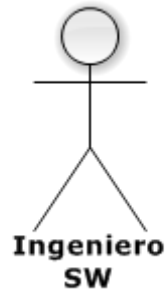
ACTOR	FUNCION PRINCIPAL	CASO DE USO
	Interactuar en el simulador con los controles de cabina, visual, pantallas multifunción, etc.	- Manejar palancas de control del motor - Manejar Controles de vuelo - Interactuar con sistema visual - Manejar controles del sistema de navegación - Interactuar con pantallas de control
	Preparar y cargar escenarios, seleccionar pilotos, y arrancar la simulación	- Cargar escenario - Preparar escenario - Modificar escenario en vuelo - Guardar configuración - Seleccionar usuario - Arrancar simulación
	Gestionar bases de datos de pilotos, de simulación, y gestionar los modelos de simulación.	- Gestionar base de datos de usuarios - Modificar bases de datos de simulación - Modificar módulos de simulación

Tabla 22. Actores Principales y Casos de Uso.

7.2.2.2 Funcionalidades del Sistema

Se detallan a continuación en la Tabla 23 las diferentes funcionalidades del sistema, que básicamente coincidirán con las descritas anteriormente:

Función	Descripción
Manejar Palancas de control del motor	Es un input para el modelo de actuaciones del motor
Manejar controles de vuelo	Es un input para el modelo de vuelo
Manejar controles del sistema de Navegación	Es un input para el modelo de navegación, sistemas y cabina.
Interactuar con sistema visual	Proporciona al piloto una recreación visual del entorno exterior de simulación
Interactuar con pantallas de control	Proporciona al piloto una recreación visual del estado de los sistemas, representada dentro de las pantallas multifunción de control
Modelo de actuaciones del motor	Calcula las actuaciones de los motores (potencia/empuje)
Modelo de vuelo	Calcula la posición de la aeronave en base a inputs de otros modelos.
Modelo de navegación, sistemas y cabina	Calcula el estado de los sistemas de navegación, sistemas y cabina, a partir de inputs de otros modelos.
Modelo de fallos	Introduce fallos en los sistemas, a partir de la configuración del instructor.
Cargar escenario	Carga un escenario de simulación
Preparar escenario	Crea un escenario de simulación
Modificar escenario en vuelo	Modifica un escenario de simulación previamente creado.
Guardar configuración	Almacena en un fichero los parámetros necesarios para reproducir un escenario.
Arrancar simulación	Inicializa la simulación a partir de un escenario y un perfil de piloto
Seleccionar usuario	Selecciona un perfil de piloto de una base de datos.
Gestionar base de datos de usuarios	Permite modificar la base de datos de perfiles de usuarios.
Modificar bases de datos de simulación	Permite modificar los parámetros de las bases de datos utilizados por los modelos de simulación
Modificar modelos de simulación	Permite modificar los propios modelos de simulación.

Tabla 23. Funcionalidades del Simulador de Vuelo.

7.2.2.3 Modelo de Arquitectura del Simulador de Vuelo

En la Figura 61 se muestra el diagrama de despliegue del simulador de vuelo:

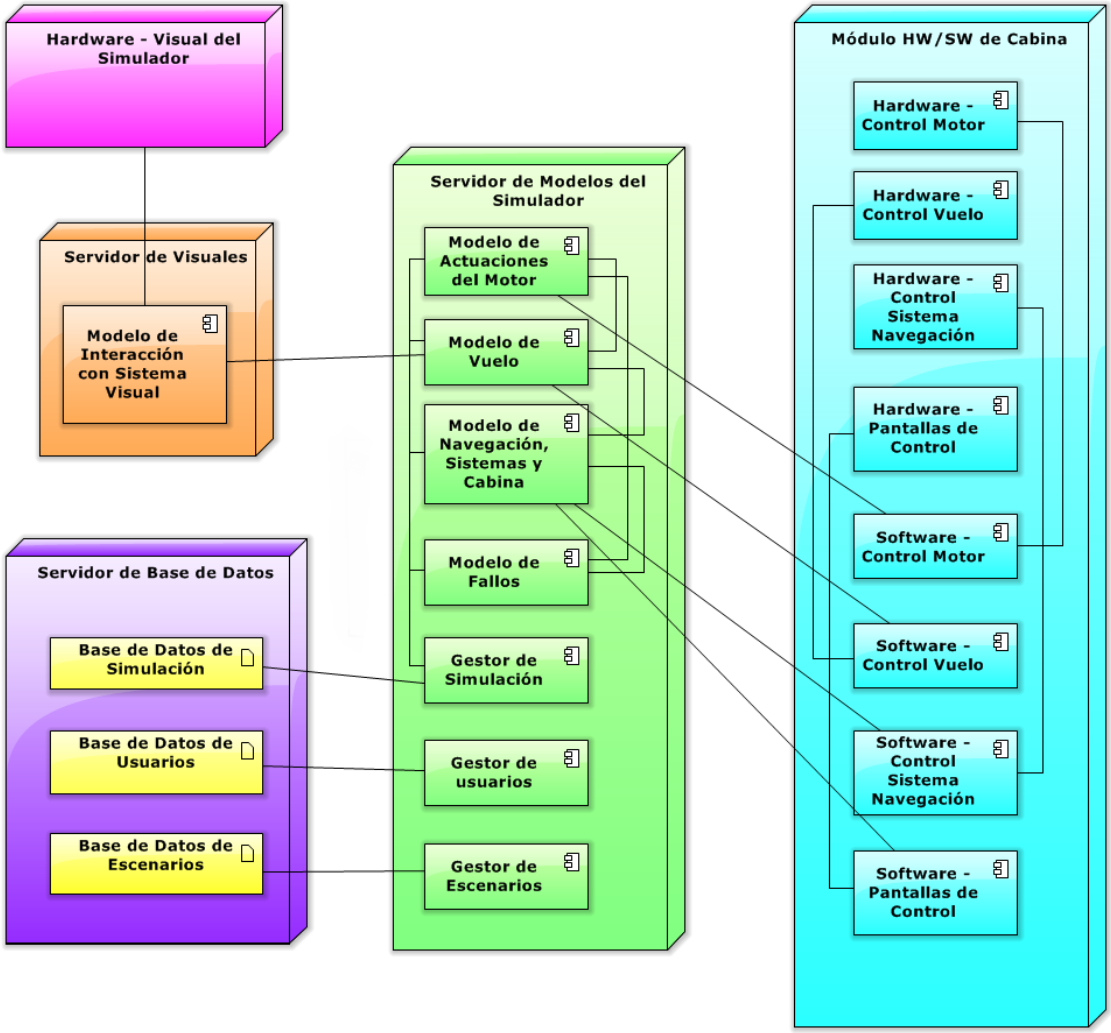


Figura 61. Diagrama de Despliegue del Simulador de Vuelo

7.2.2.4 Escenarios de Calidad

Los escenarios de calidad propuestos para analizar la arquitectura del simulador son todos ellos de modificabilidad, y son los siguientes:

- Escenario 1

M1 - Modificaciones aerodinámicas han de introducirse en el avión, afectando al modelo de vuelo. Estos cambios han de poder implementarse en un tiempo no superior a 20 días.

Elemento	Atributo/Valor
Estímulo	Realizar una nueva modificación aerodinámica
Fuente del Estímulo	Ingeniero SW
Entorno	En tiempo de diseño o mantenimiento
Artefacto	Sistema
Respuesta	La modificación se implementa en el modelo de vuelo
Medida de la Respuesta	En 20 días

Tabla 24. Escenario de Modificabilidad M1.

- Escenario 2

M2 - Nueva aviónica se introduce en el simulador, afectando al modelo de navegación, modelo de manejo de controles del sistema de navegación y modelo de interacción con pantallas de control. El tiempo estimado de implementación es de 15 días.

Elemento	Atributo/Valor
Estímulo	Realizar una nueva modificación de aviónica
Fuente del Estímulo	Ingeniero SW
Entorno	En tiempo de diseño o mantenimiento
Artefacto	Sistema
Respuesta	La modificación se implementa en el modelo de navegación, modelo de manejo de controles del sistema de navegación y modelo de interacción con pantallas de control
Medida de la Respuesta	En 15 días

Tabla 25. Escenario de Modificabilidad M2.

- Escenario 3

M3 - Modificar un perfil de usuario para añadir nuevas características afectara al modulo gestionar base de datos de usuario. Esto puede incluir operaciones de actualización (agregar, dar de baja usuarios), sincronización con la base de datos (1 día). La duración total estimada es de 4 días.

ELEMENTO	ATRIBUTO/VALOR
Estímulo	Realizar una modificación en perfiles de usuario.
Fuente del Estímulo	Ingeniero SW
Entorno	En tiempo de diseño o mantenimiento
Artefacto	Sistema
Respuesta	La modificación se implementa en el modelo de gestión de base de datos de usuario
Medida de la Respuesta	En 4 días

Tabla 26. Escenario de Modificabilidad M3.

- Escenario 4

M4 - La inclusión de un nuevo fallo afectaría al modulo de modelo de fallos, modulo preparar escenario y modulo modificar escenario en vuelo. Dicha modificación debe ser incluida en 10 días.

ELEMENTO	ATRIBUTO/VALOR
Estímulo	Inclusión de un nuevo fallo
Fuente del Estímulo	Ingeniero SW
Entorno	En tiempo de diseño o mantenimiento
Artefacto	Sistema
Respuesta	Modificación implementada dentro del modelo de fallos
Medida de la Respuesta	En 10 días

Tabla 27. Escenario de Modificabilidad M4.

7.2.3 Análisis de la Arquitectura con ArchE

7.2.3.1 Definición de Funciones

El primer paso es introducir las funciones en ArchE: (Figura 62)

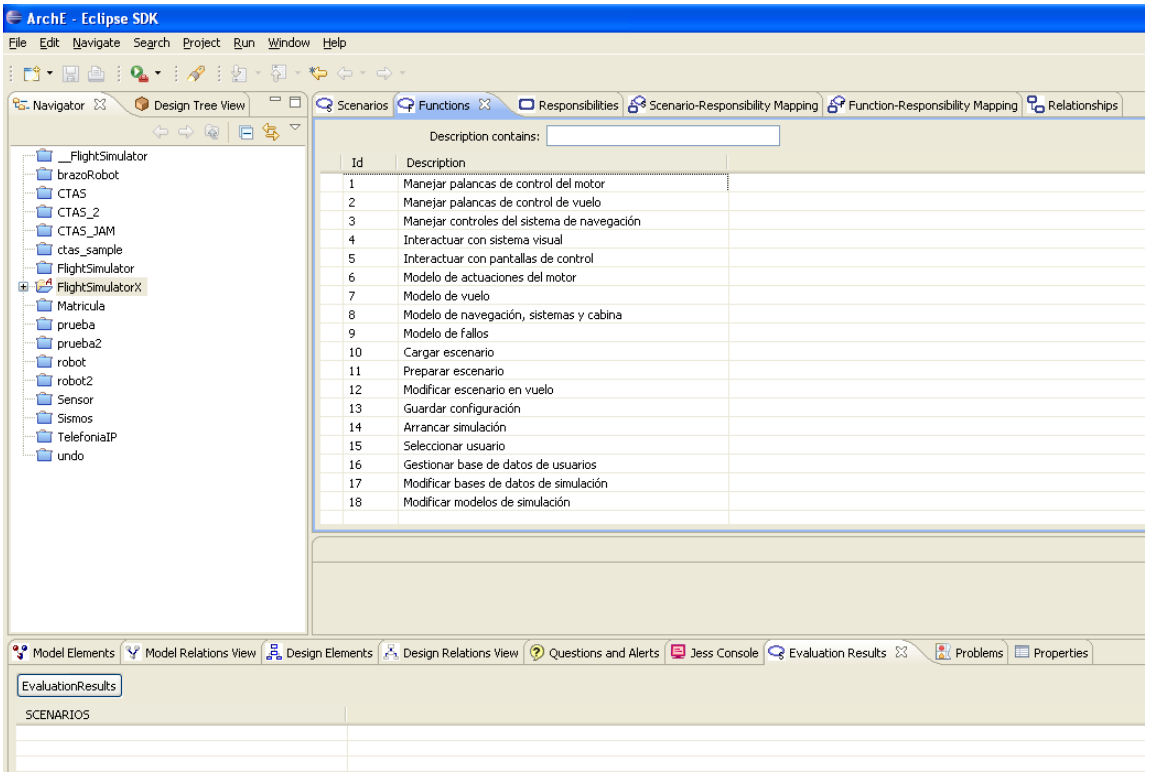


Figura 62. Funciones del Simulador en ArchE.

7.2.3.2 Definición de Responsabilidades

Una vez introducidas las funciones, es el propio ArchE quien se encarga de crear responsabilidades y mapearlas a las funciones: (Figura 63)

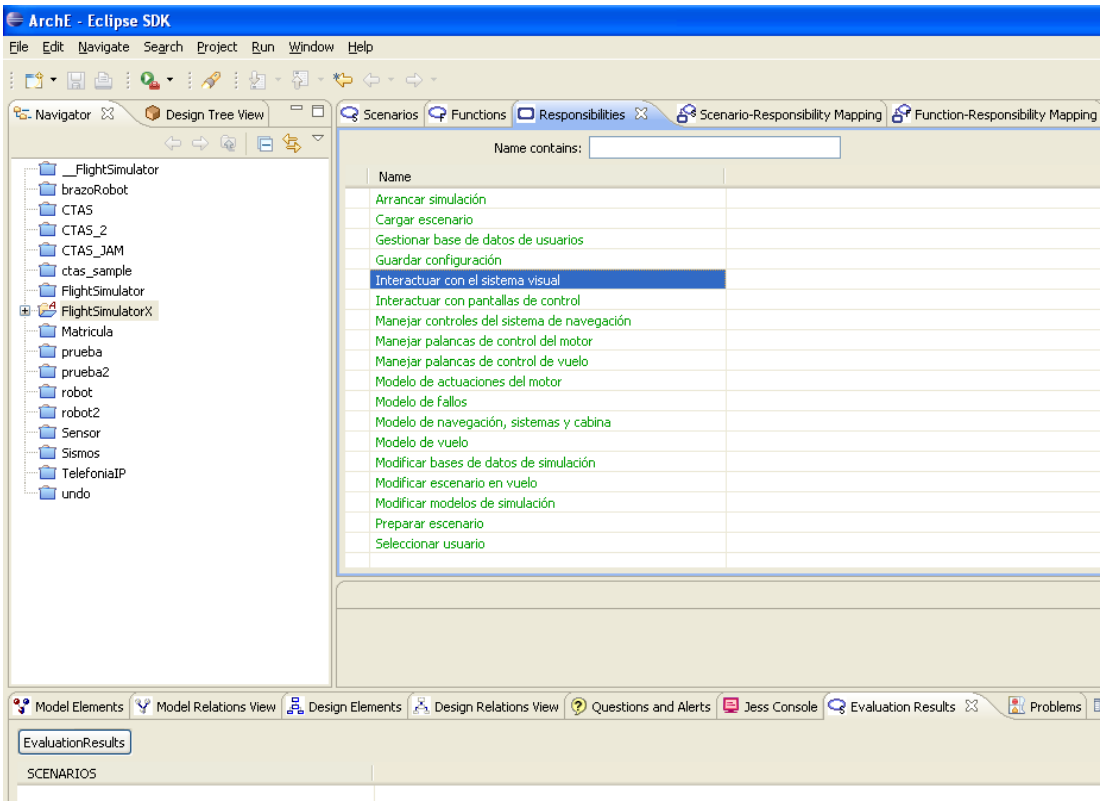


Figura 63. Responsabilidades del Simulador en ArchE.

A continuación, Arche muestra el mapeado Función-Responsabilidad: (Figura 64)

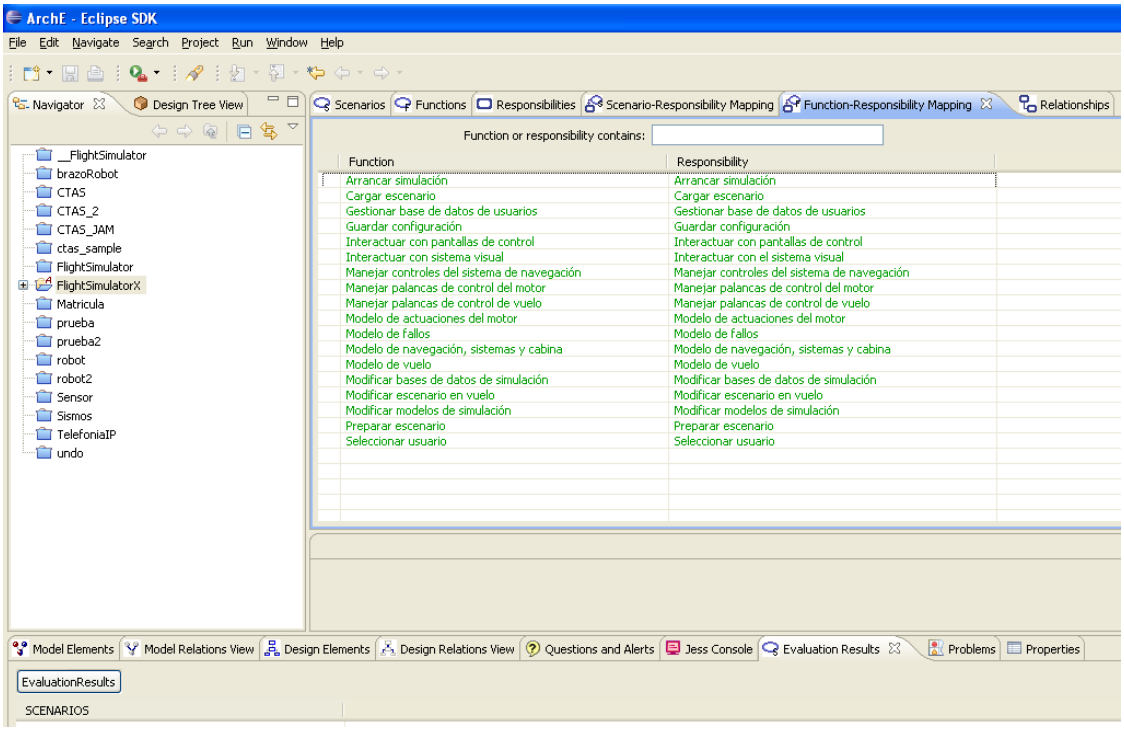


Figura 64. Mapeado Funciones-Responsabilidades en ArchE.

7.2.3.3 Definición de Relaciones

Las relaciones del marco de modificabilidad tienen que ver con cómo afecta la modificación de una responsabilidad a otras con las que guarda algún tipo de relación; este tipo de relación se denomina en ArchE relación de dependencia. En la Tabla 28 se muestran las relaciones de dependencia entre las responsabilidades del simulador de vuelo:

	Child Responsability																		
Parent Responsability	Manejar palancas de control del motor	Manejar palancas de control de vuelo	Manejar controles del sistema de navegación	Interactuar con sistema visual	Interactuar con pantallas de control	Modelo de actuaciones del motor	Modelo de vuelo	Modelo de navegación, sistemas y cabina	Modelo de fallos	Cargar escenario	Preparar escenario	Modificar escenario en vuelo	Guardar configuración	Arrancar simulación	Seleccionar usuario	Gestionar base de datos de usuarios	Modificar bases de datos de simulación	Modificar modelos de simulación
Manejar palancas de control del motor						D												
Manejar palancas de control de vuelo							D											
Manejar controles del sistema de navegación								D										
Interactuar con sistema visual																		
Interactuar con pantallas de control																		
Modelo de actuaciones del motor							D		D									
Modelo de vuelo				D					D									
Modelo de navegación, sistemas y cabina					D		D		D									
Modelo de fallos																		
Cargar escenario																		
Preparar escenario									D				D					
Modificar escenario en vuelo									D									
Guardar configuración																		
Arrancar simulación										D	D							
Seleccionar usuario																		
Gestionar base de datos de usuarios																		
Modificar bases de datos de simulación																		
Modificar modelos de simulación																		

Tabla 28. Relaciones de Dependencia entre las Responsabilidades del Simulador de Vuelo.

A continuación se introducen dichas relaciones en ArchE (Figura 65). Antes de definir las relaciones entre las distintas responsabilidades, hay que asegurarse de que el marco de razonamiento ChangeImpact Modifiability esté iniciado. Si no fuera así, no se podría seleccionar el tipo de relación “Dependency”.

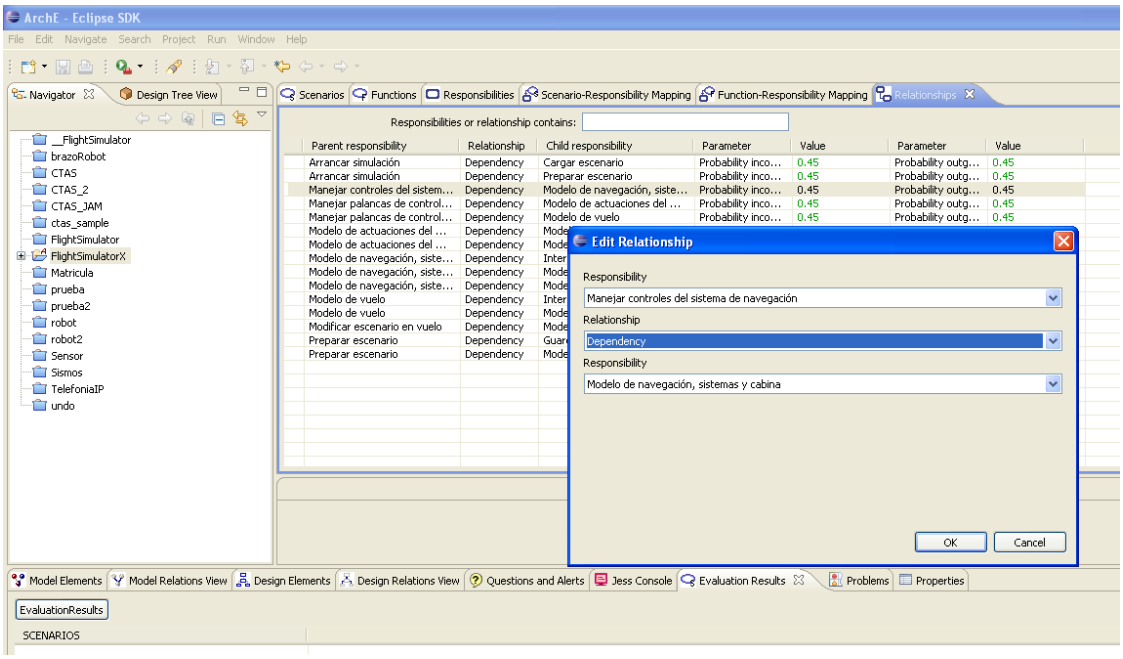


Figura 65. Relaciones de Dependencia del Simulador de Vuelo en ArchE.

7.2.3.4 *Definición de Escenarios*

El siguiente paso es introducir los diferentes escenarios en ArchE, teniendo la precaución de que el marco de razonamiento ChangeImpact Modifiability esté iniciado. Si no fuera así, no se podría seleccionar el tipo de escenario: (Figura 66)

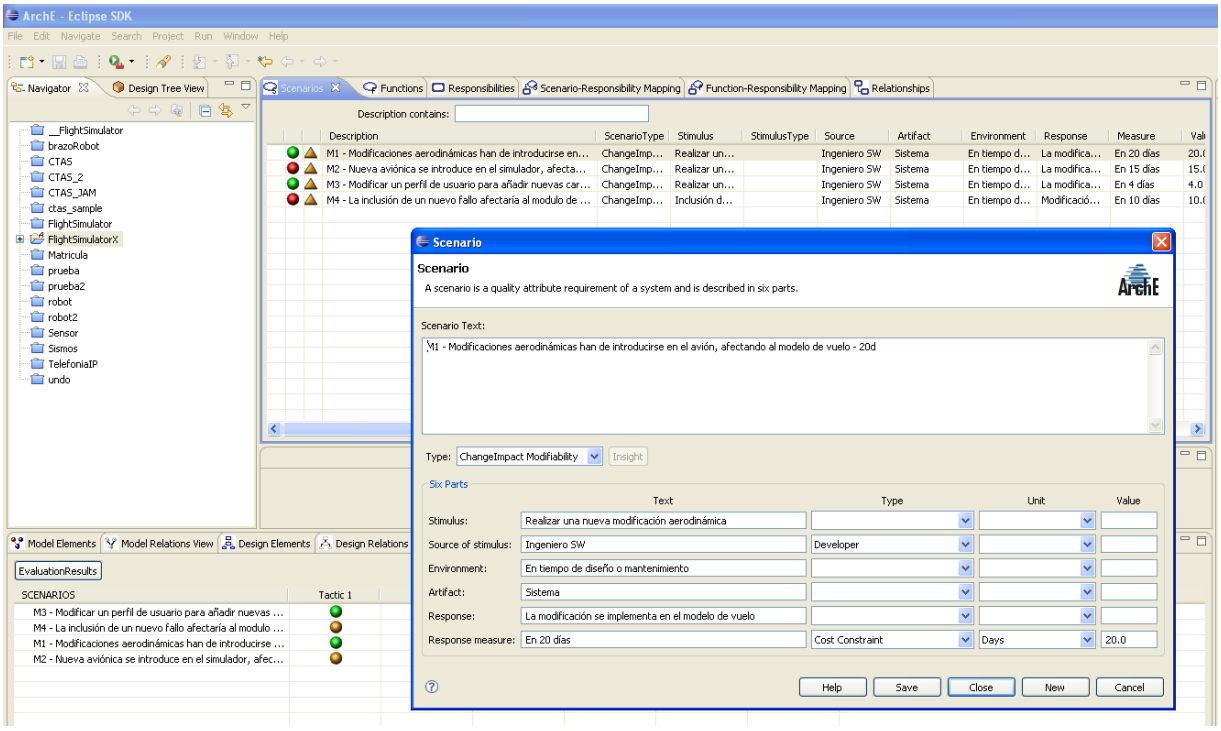


Figura 66. Introducción de Escenarios del Simulador de Vuelo en ArchE.

7.2.3.5 Mapeo de Escenarios a Responsabilidades

El último paso para poder iniciar el análisis de arquitectura con ArchE es mapear cada escenario con la responsabilidad impactada. En la Tabla 29 viene el mapeado de Escenarios y responsabilidades, y en la Figura 67 dicho mapeado introducido en ArchE:

	Escenarios			
Responsabilidades	**M1**	**M2**	**M3**	**M4**
Manejar palancas de control del motor				
Manejar palancas de control de vuelo				
Manejar controles del sistema de navegación		X		
Interactuar con sistema visual				
Interactuar con pantallas de control		X		
Modelo de actuaciones del motor				
Modelo de vuelo	X			
Modelo de navegación, sistemas y cabina		X		
Modelo de fallos				X
Cargar escenario				
Preparar escenario				X
Modificar escenario en vuelo				X
Guardar configuración				
Arrancar simulación				
Seleccionar usuario				
Gestionar base de datos de usuarios			X	
Modificar bases de datos de simulación				
Modificar modelos de simulación				

Tabla 29. Mapeado de Responsabilidades y Escenarios del Simulador de Vuelo.

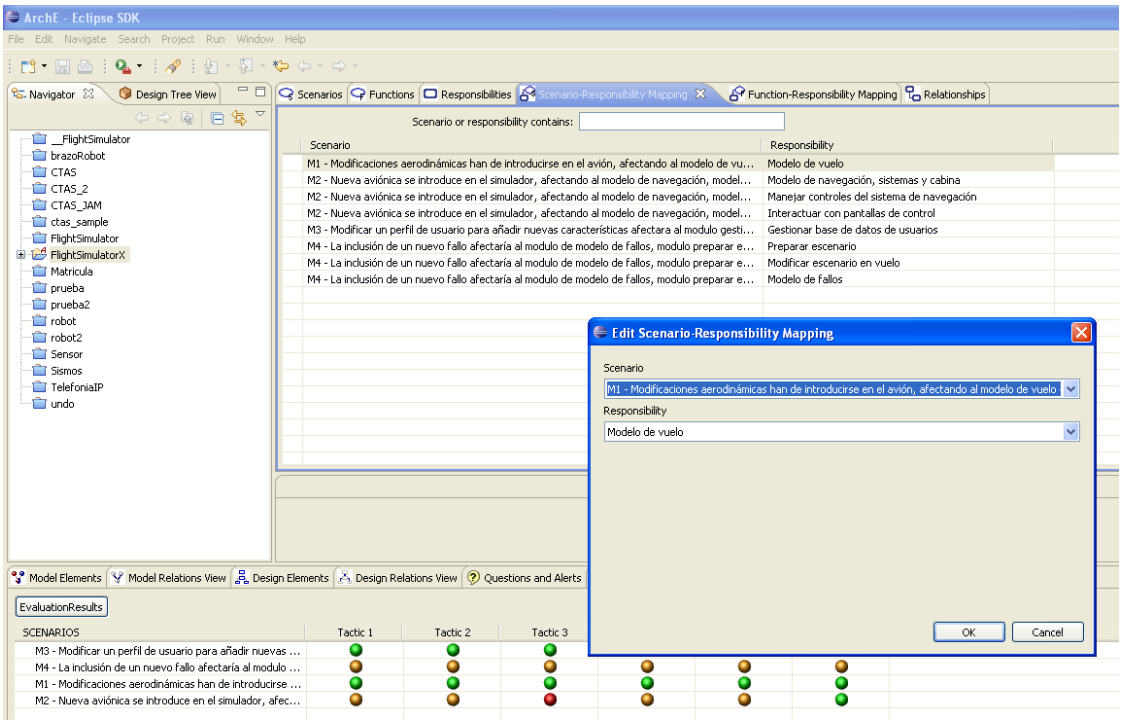


Figura 67. Mapeado de Responsabilidades y Escenarios del Simulador de Vuelo en ArchE.

7.2.3.6 *Análisis de ArchE*

En una primera iteración, ArchE muestra el siguiente análisis de la Figura 68:

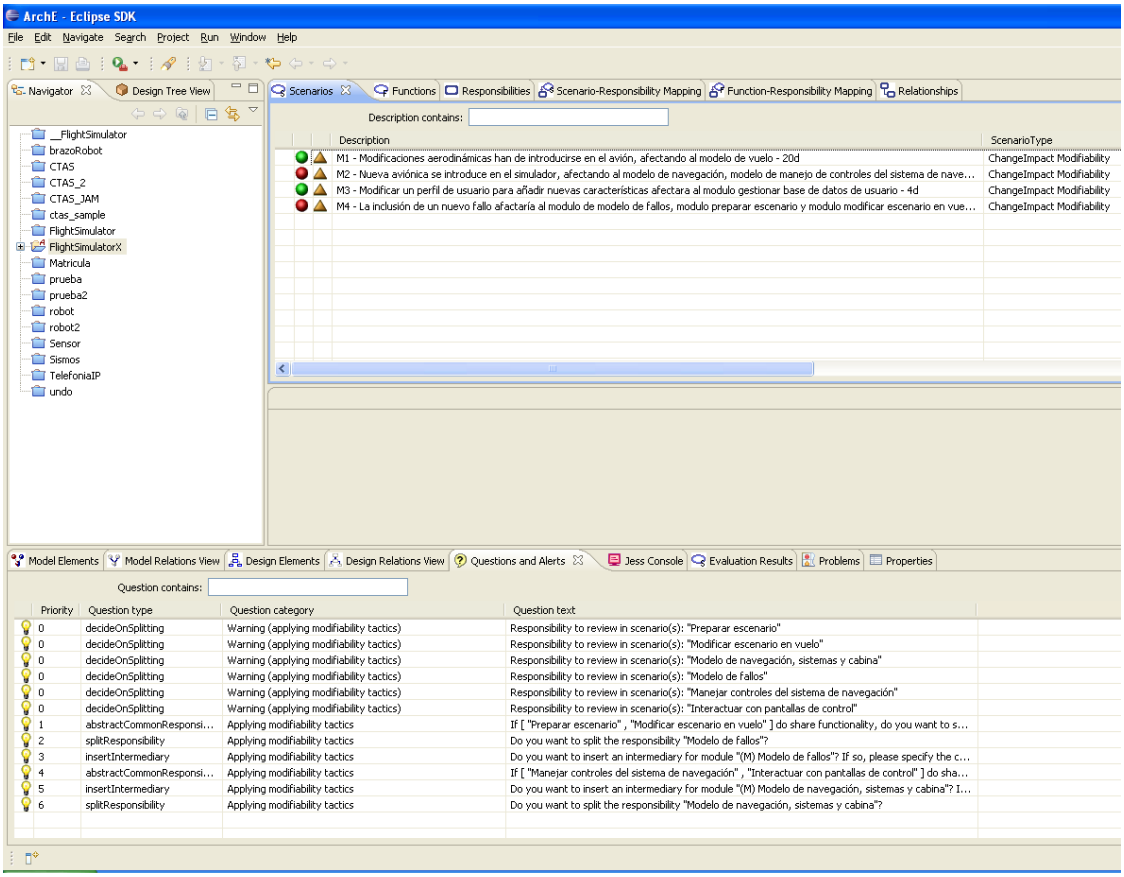


Figura 68. Análisis Inicial de Arche - Simulador.

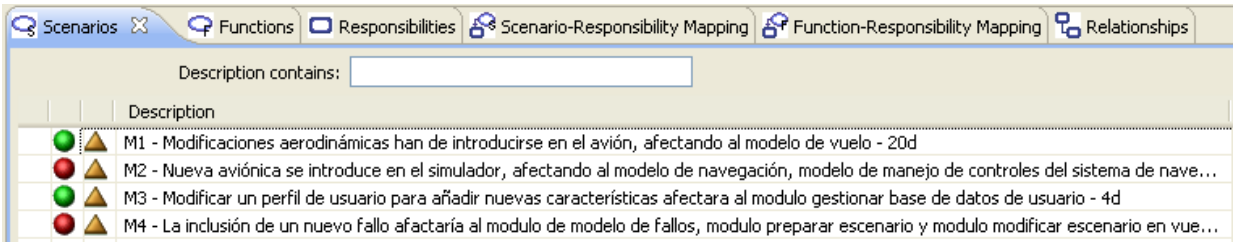


Figura 69. Resultado en los Escenarios de Modificabilidad – Simulador de Vuelo.

Se puede observar que se cumplen dos escenarios, M1 y M2, ambos marcados con un círculo verde. Sin embargo, M2 y M4 están marcados con un círculo rojo, lo cual significa que no se cumplen.

ArchE también muestra un esquema de relaciones entre responsabilidades similar al modelo ULM (Figura 70):

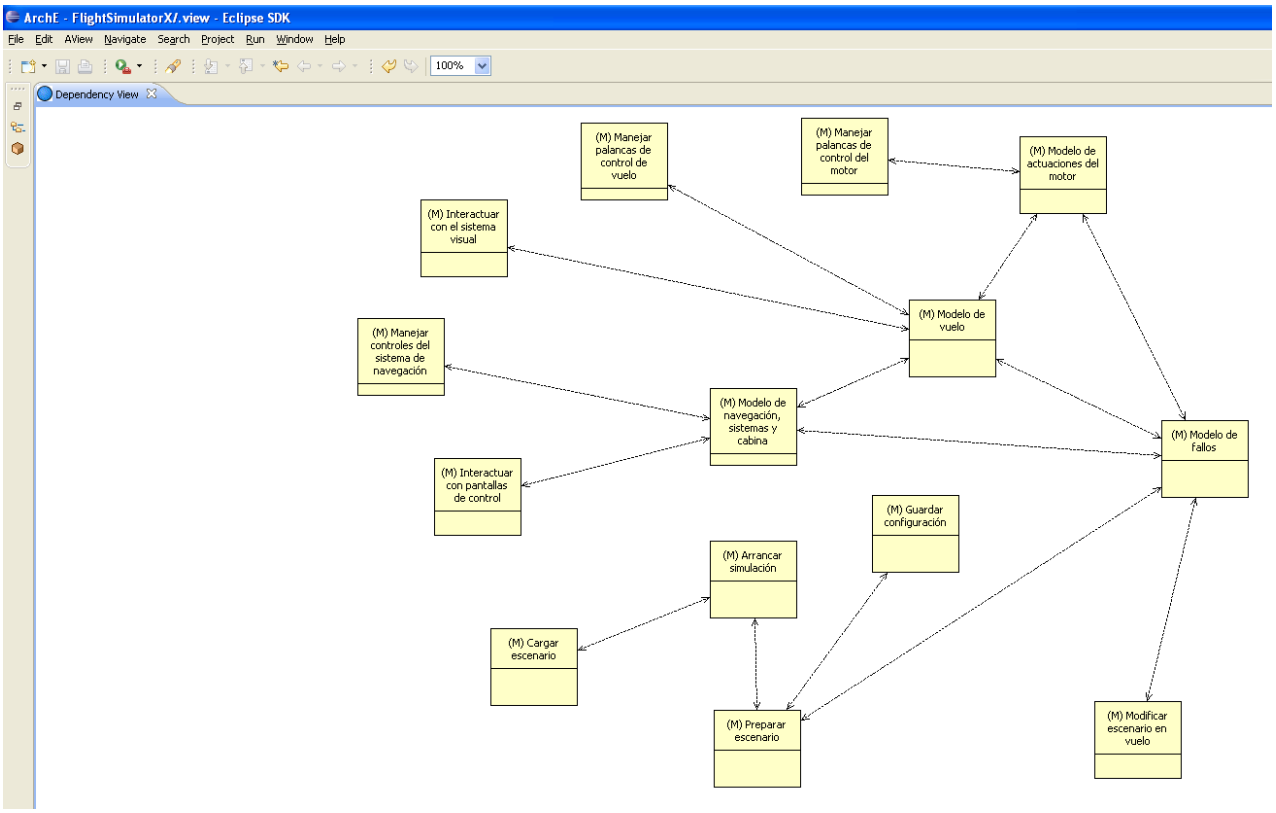


Figura 70. Vista de Dependencias de ArchE – Simulador.

El análisis de tácticas que realiza ArchE muestra lo siguiente: (Figura 71)

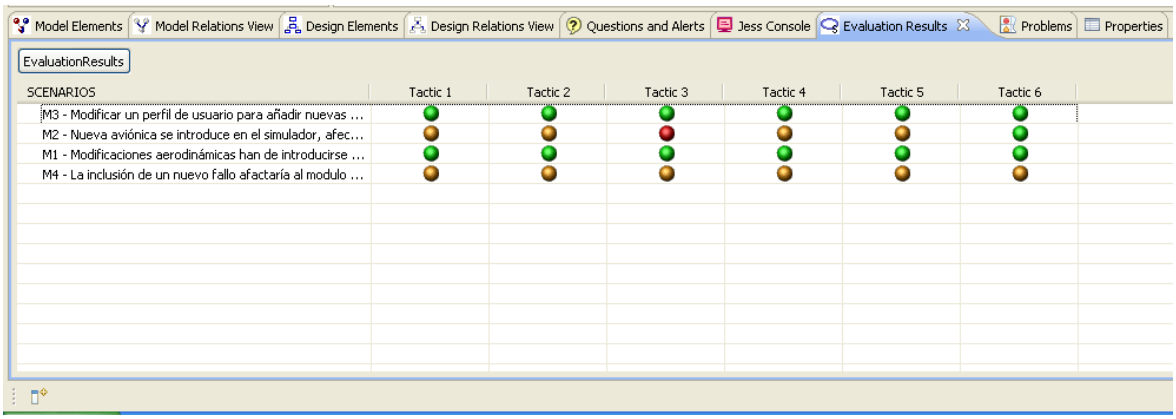


Figura 71. Análisis de Tácticas de ArchE - Simulador.

Las tácticas con un círculo verde significan que su aplicación supondrá una mejora en el escenario. La táctica 6 es la que parece producir una mejora en varios escenarios, en concreto en M1, M2 y M3.

Para intentar cumplir más escenarios, se revisa la lista de preguntas que proporciona ArchE: (Figura 72)

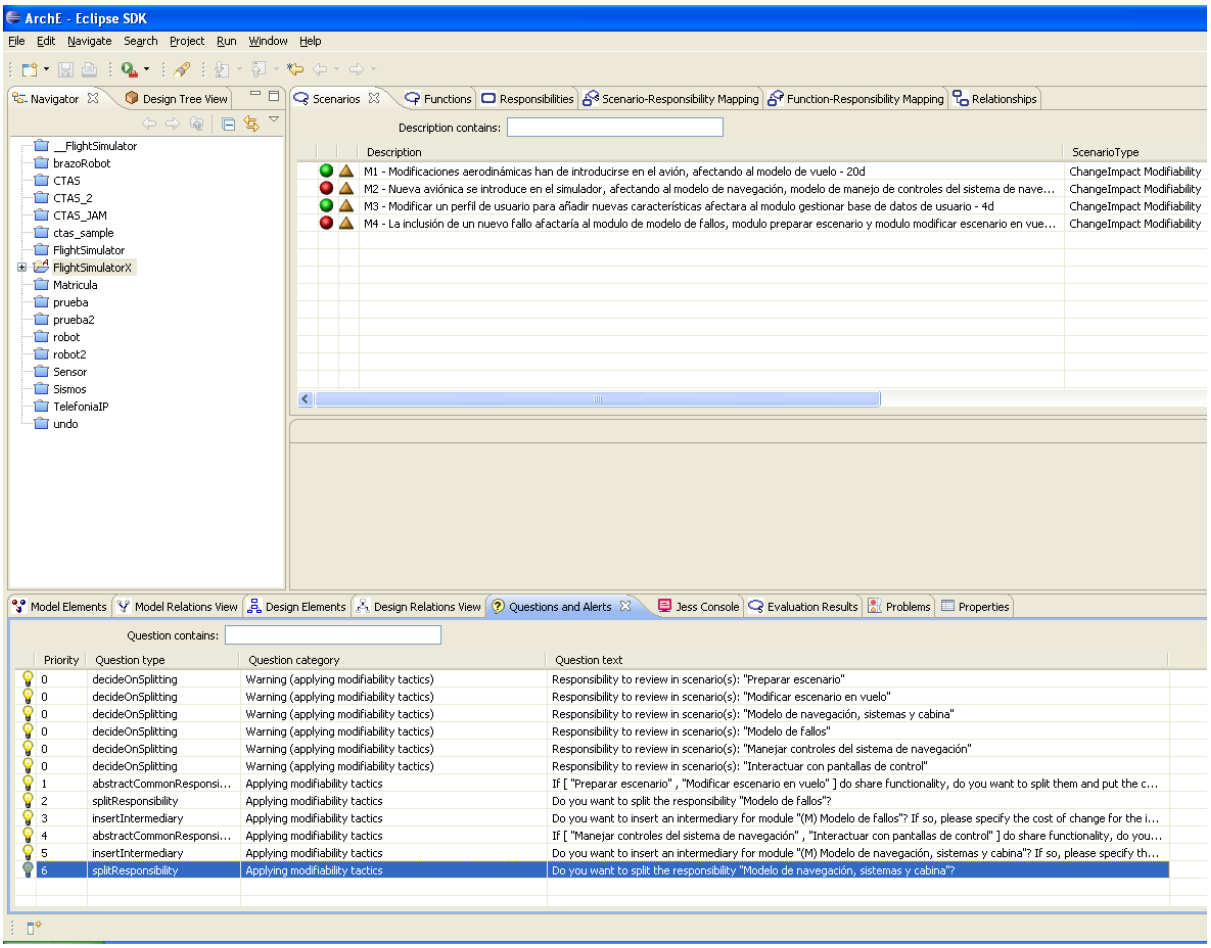


Figura 72. Lista de preguntas de ArchE - Simulador.

Se observa que la pregunta de prioridad 6 aconseja separar las responsabilidades del modelo de navegación, sistemas y cabina: (Figura 73)

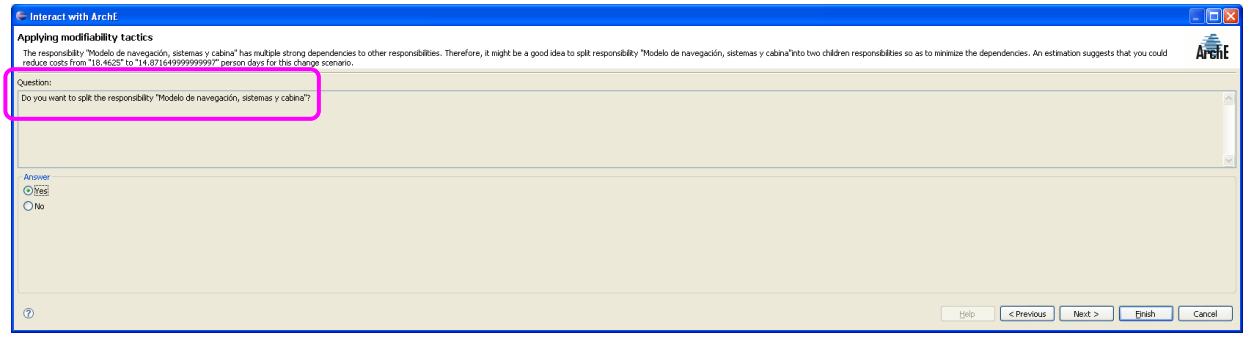


Figura 73. Elección de Preguntas para Mejoras en la Arquitectura.

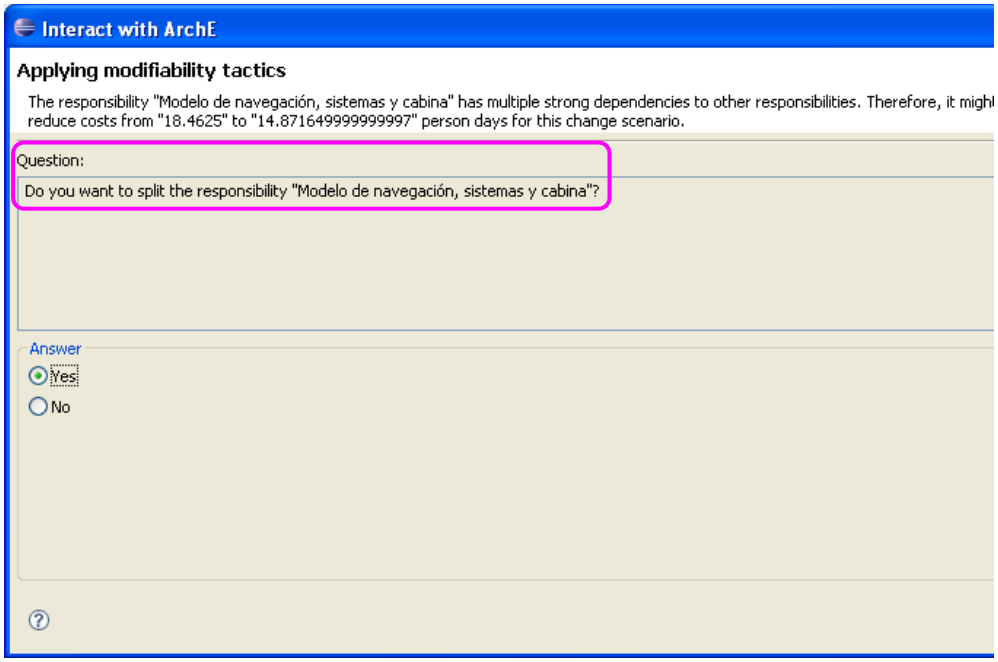


Figura 74. Elección de Preguntas para Mejoras en la Arquitectura (ampliado).

Al aplicar esta táctica, se obtiene el siguiente resultado: (Figura 75)

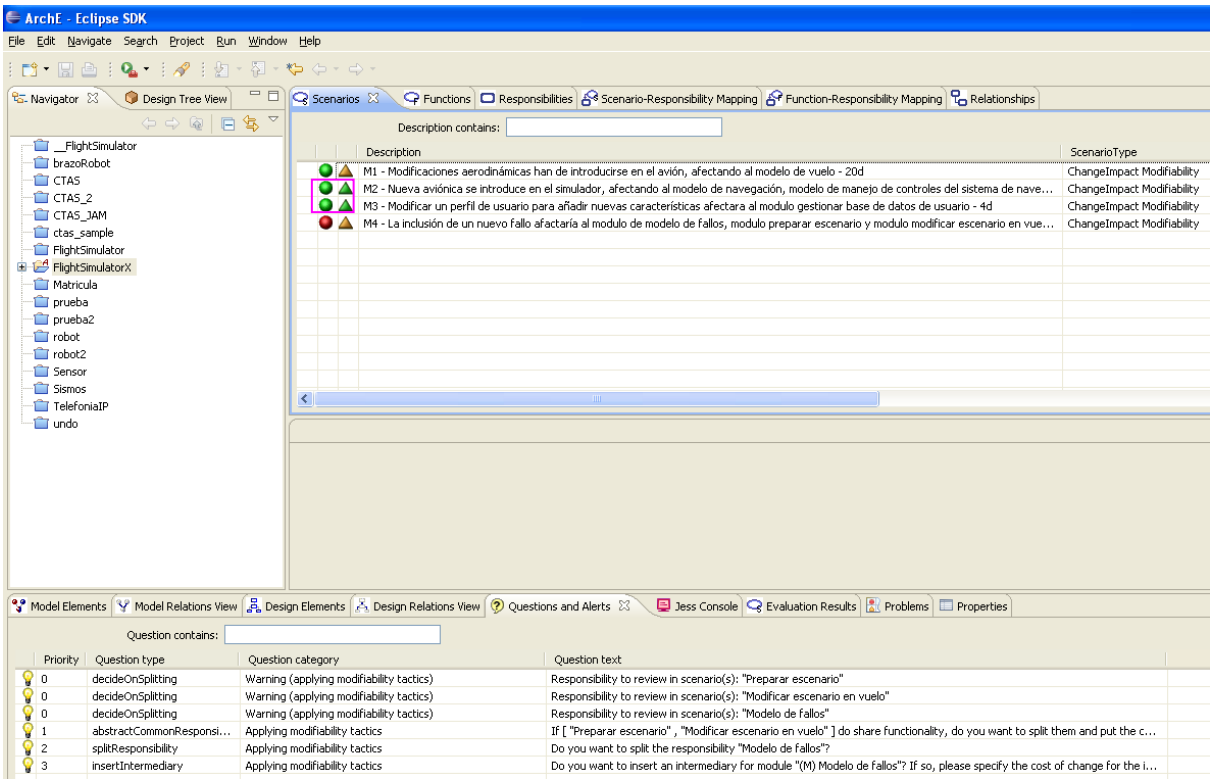


Figura 75. Aplicando Tácticas en Arche - Simulador.

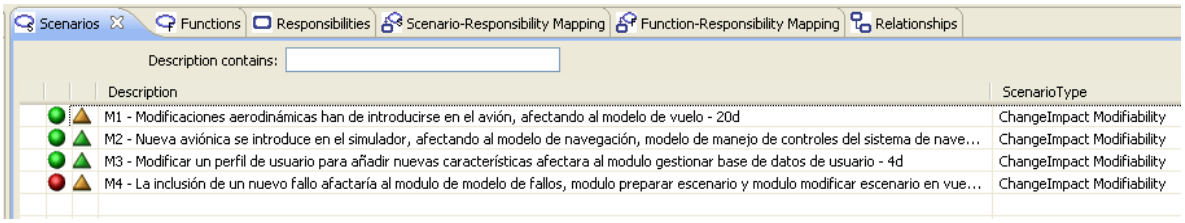


Figura 76. Aplicando Tácticas en Arche – Simulador (ampliado).

Lo cual se traduce en una mejora del escenario M3 y el cumplimiento del escenario M2.

Al aplicar la táctica, el diagrama de dependencias ha sido modificado: (Figura 77)

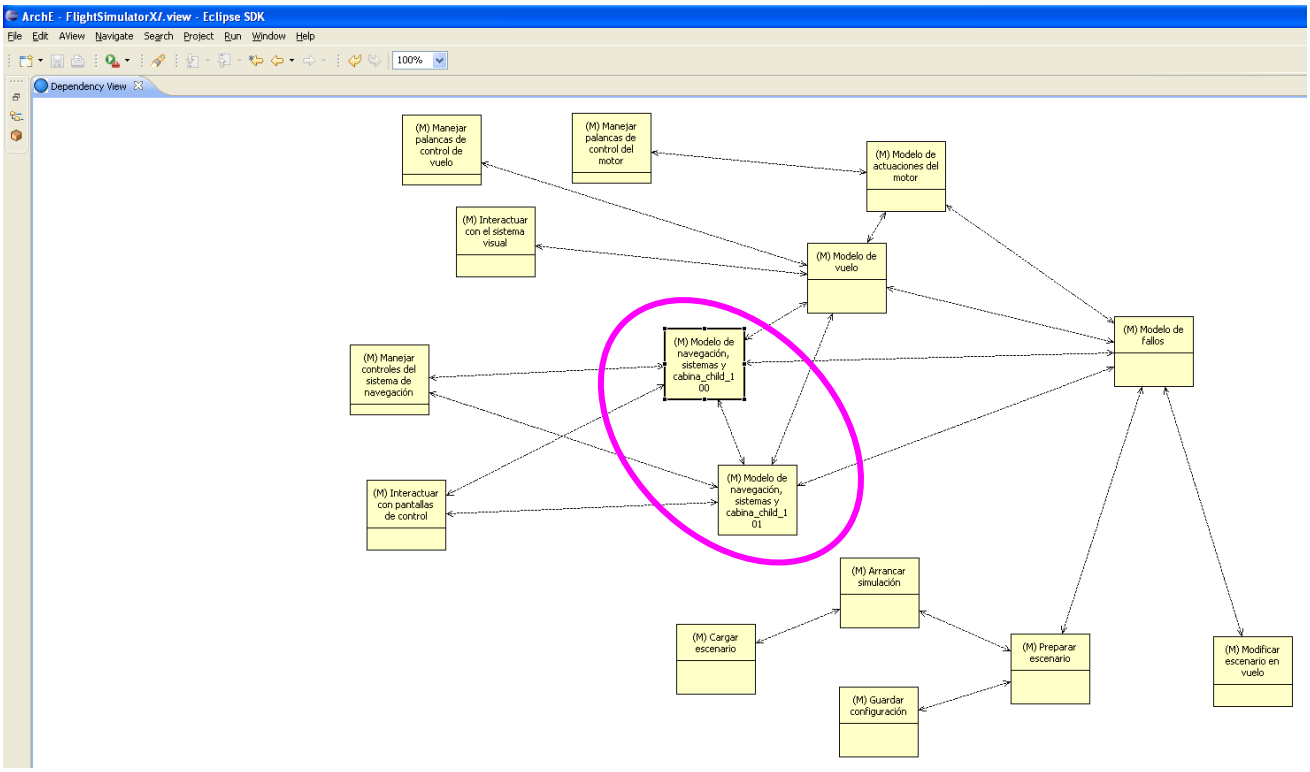


Figura 77. Diagrama de Dependencias Modificado tras aplicar táctica.

Lo cual tiene sentido, dado el elevado número de sistemas de aviónica, navegación, sistemas y cabina que hay, se puede separar en dos partes bien diferenciadas: por un lado se pueden poner sistemas de navegación, vigilancia y comunicaciones, y por otro lado, se pueden poner sistemas generales, sistemas de presentación de datos en cabina, otros sistemas de aviónica, etc. La sugerencia de ArchE, sin tener conocimientos acerca de la arquitectura de un sistema de aviónica, puede hacer sugerencias sobre la arquitectura software y la distribución de funciones.

El nuevo análisis de ArchE muestra el siguiente resultado: (Figura 78)

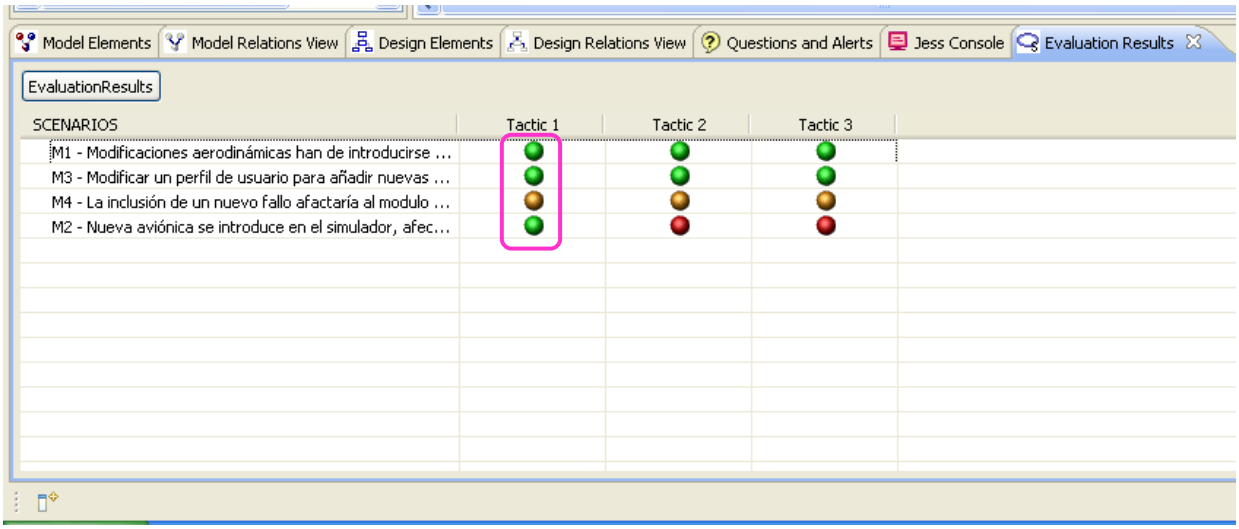


Figura 78. Nuevo análisis de Arche - Simulador.

Si se analizan las sugerencias de ArchE, se observa que la táctica 1 puede provocar una mejora en los escenarios M1, M2 y M3: (Figura 79)

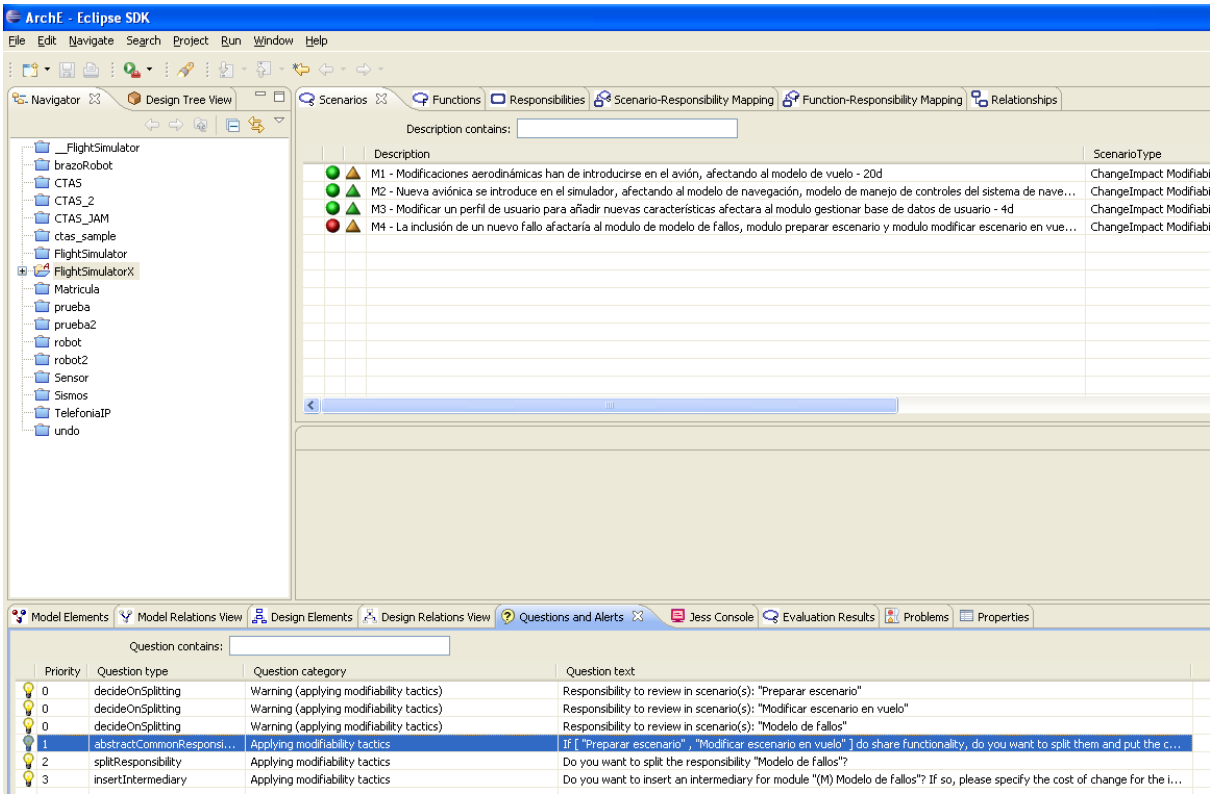


Figura 79. Nueva Lista de Preguntas de ArchE - Simulador.

La sugerencia de ArchE a la pregunta de prioridad 1 es la siguiente: (Figura 80)

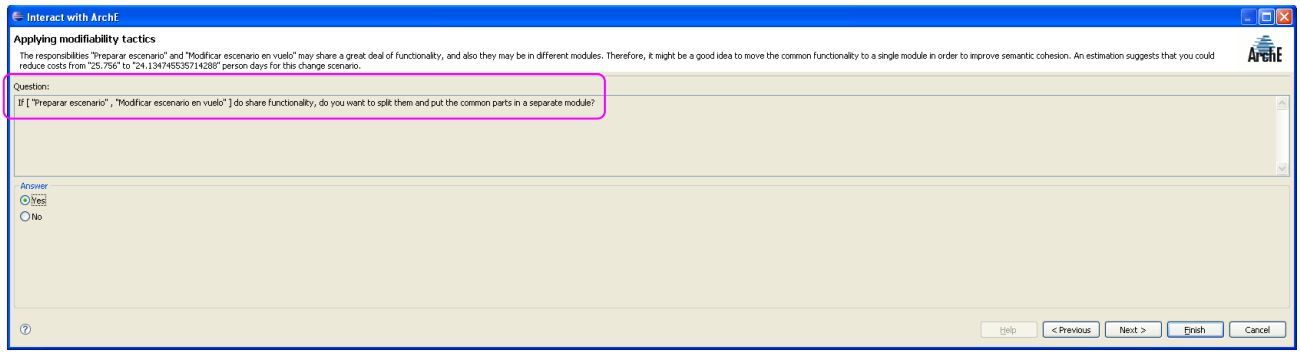


Figura 80. Sugerencias ArchE.

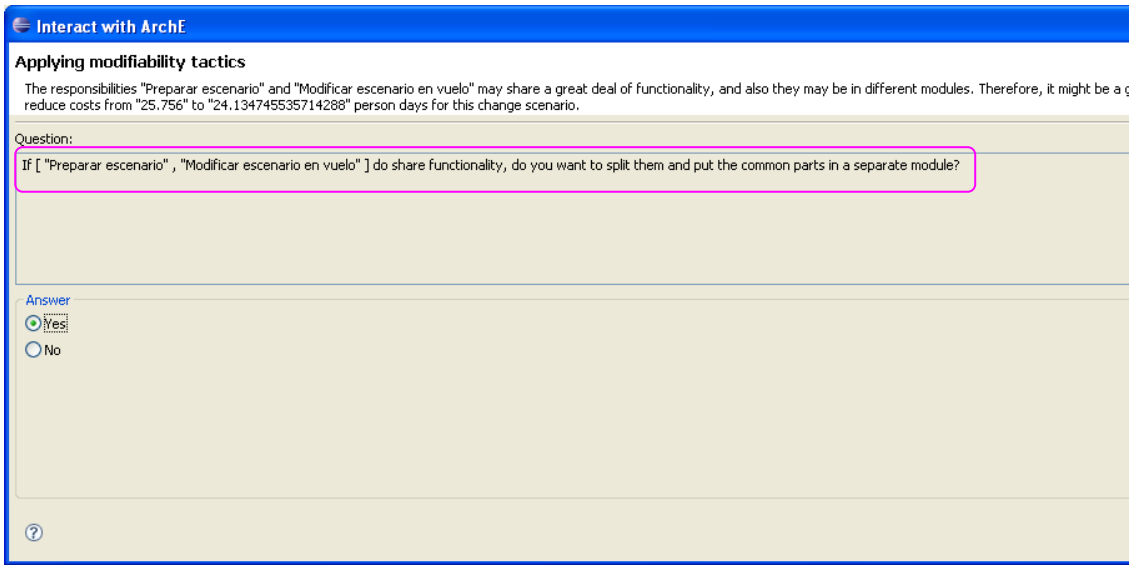


Figura 81. Sugerencias ArchE (ampliada)

Al aceptar la sugerencia y aplicar la táctica, el resultado obtenido es el siguiente: (Figura 82)

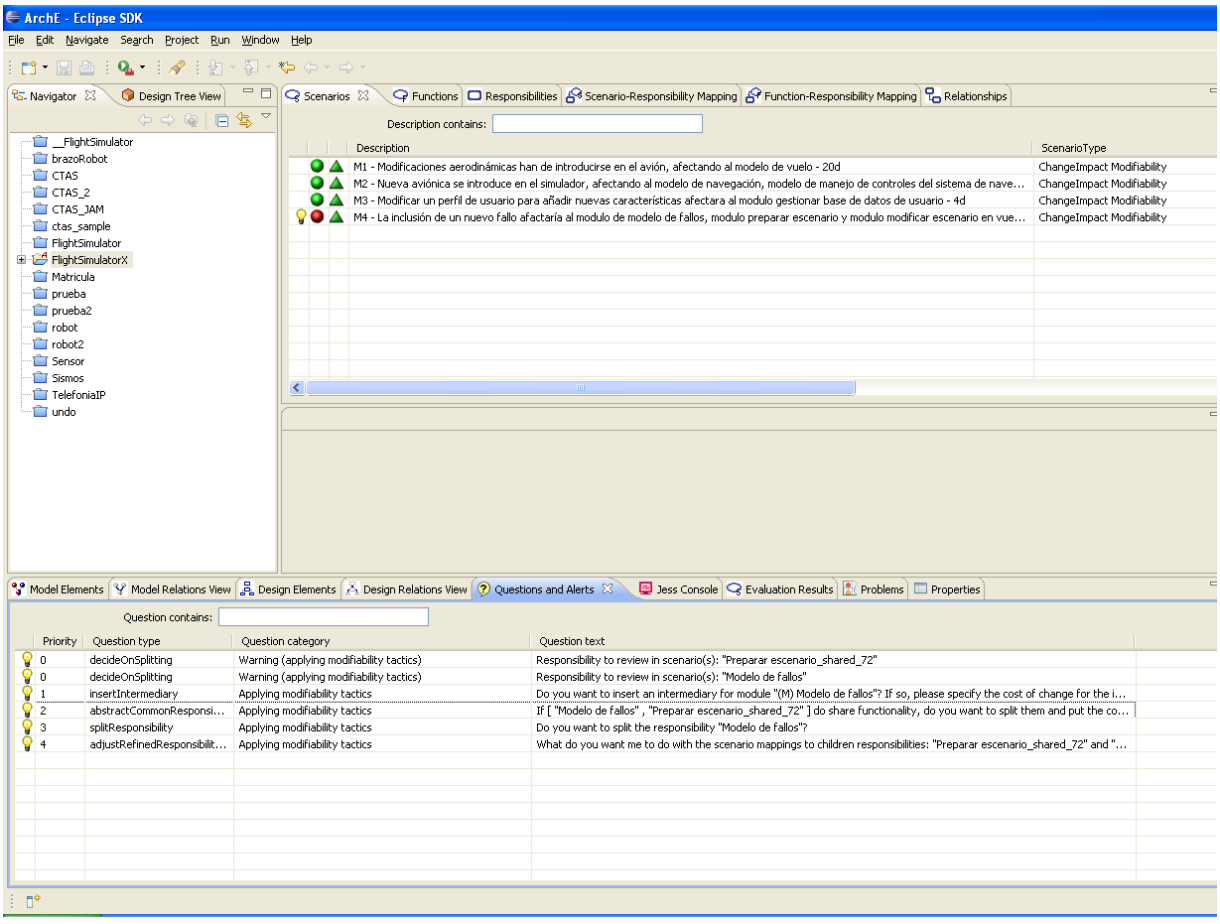


Figura 82. Aplicación de una nueva Táctica – Simulador.

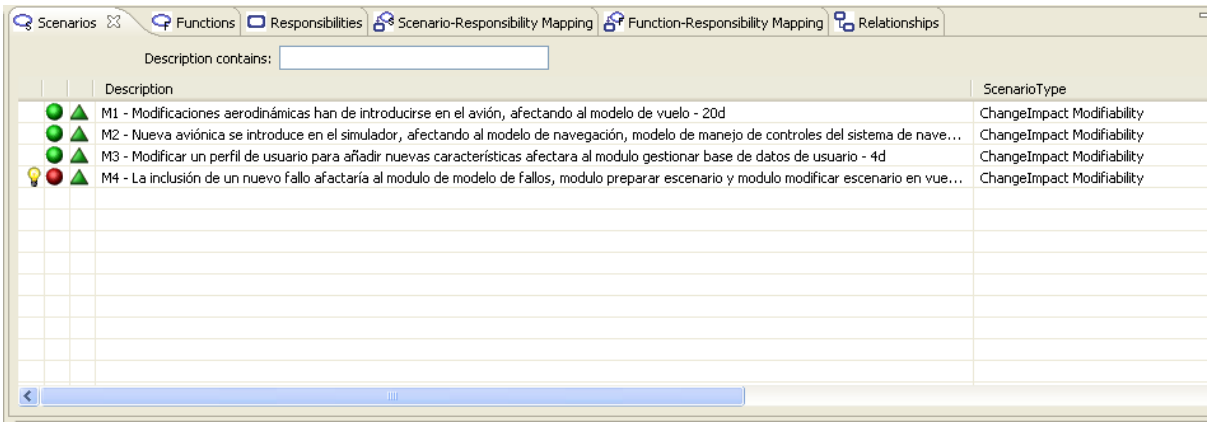


Figura 83. Aplicación de una nueva Táctica – Simulador (ampliada).

De aplicar esa táctica se obtiene una mejora en M1, M2 y M3. El nuevo diagrama de dependencias quedaría de la siguiente manera: (Figura 84)

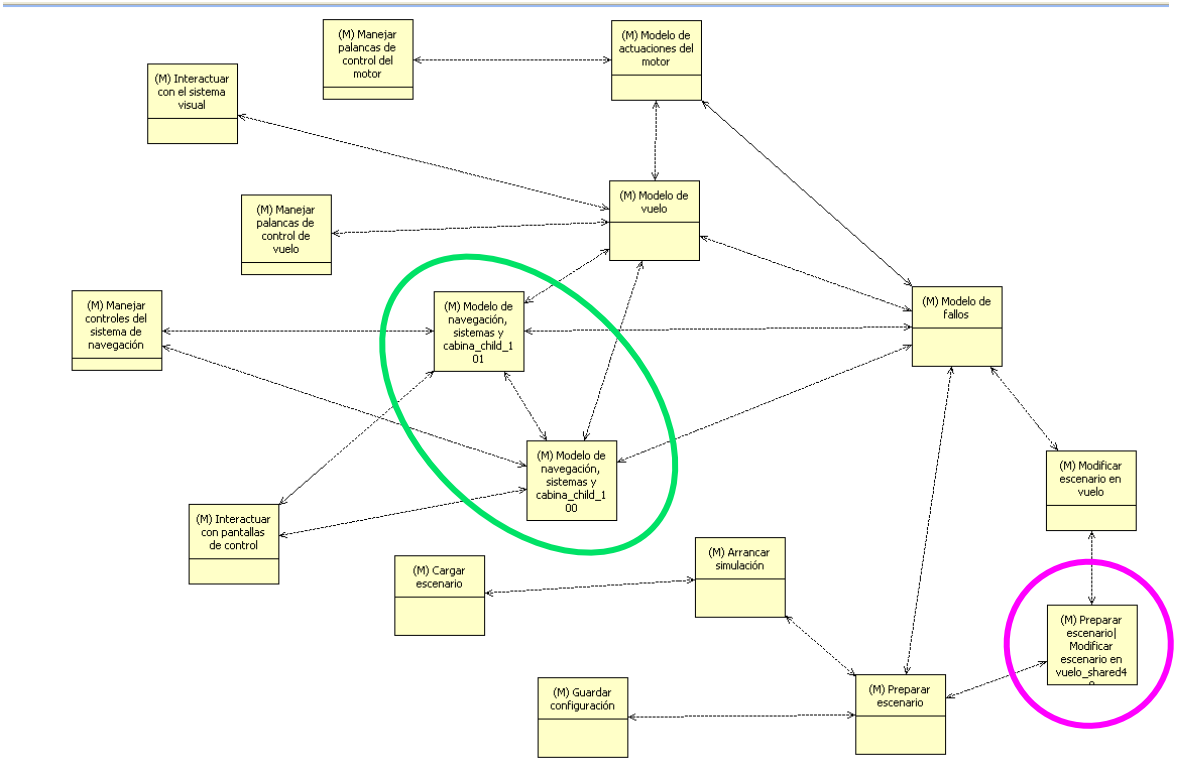


Figura 84. Diagrama de Dependencias Final - Simulador.

Las sucesivas tácticas no van a conseguir mejorar más el diseño de la arquitectura, especialmente en el escenario M4, que es el que aún no se cumple, por lo tanto aquí se detiene la iteración de ArchE. (Figura 85)

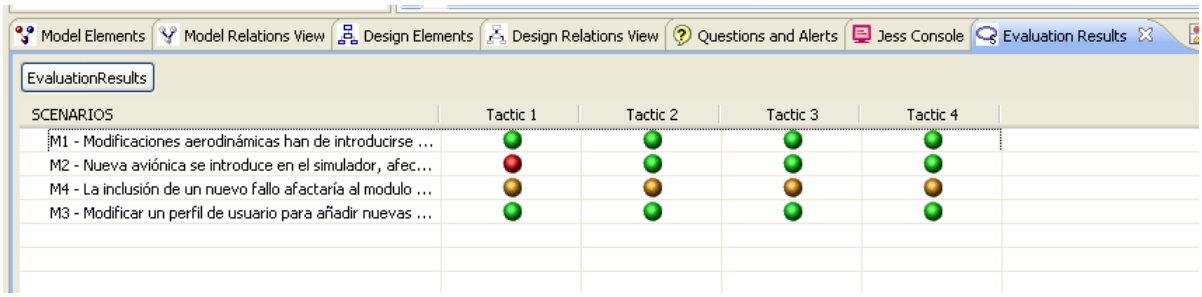


Figura 85. Últimas Tácticas de Arche sobre el Simulador.

7.3 Sistema de Aviónica con Software Empotrado

7.3.1 Arquitectura del Sistema de Aviónica

Un Sistema de aviónica con software empotrado está constituido por un hardware con uno o varios procesadores, interfaces de entrada y salida, dispositivos programables tipo FPGA, etc; y un sistema software, con un sistema operativo y aplicaciones software, tal y como se vió en el capítulo 6.

7.3.1.1 Arquitectura Hardware

En la parte hardware, el sistema estará compuesto de una arquitectura como la de la Figura 86:

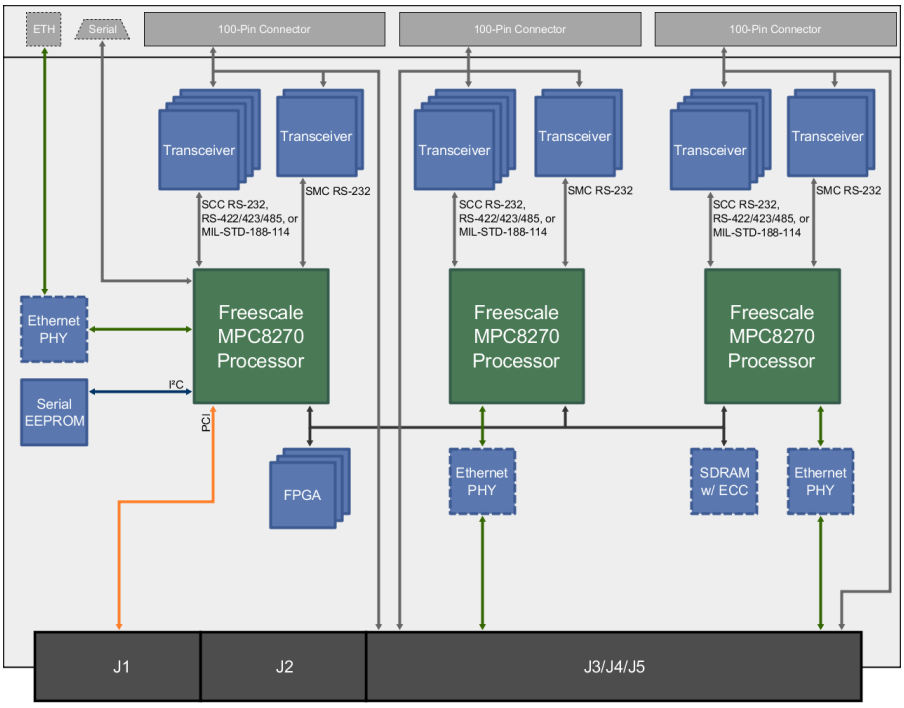


Figura 86. Arquitectura Hardware del Sistema de Aviónica. [27]

En la Figura 87 se puede ver la implementación física de dicha arquitectura:

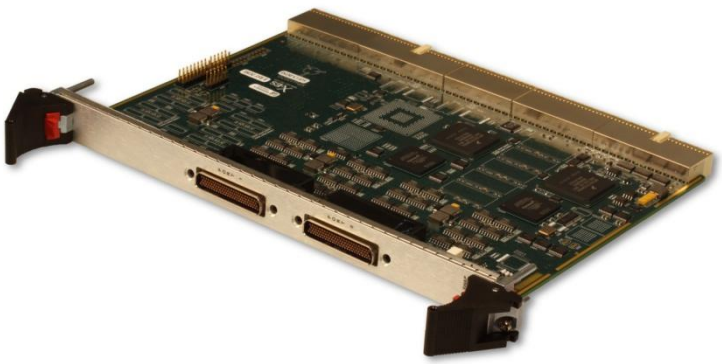


Figura 87. Placa Hardware del Sistema de Aviónica. [27]

7.3.1.2 Arquitectura Software

En la parte software, un sistema de aviónica modular de este tipo es bastante complejo; sin embargo, se pueden identificar los siguientes módulos principales en la Tabla 30:

#	Módulo	Definición
01	Boot	Inicializa el Módulo planificador y el módulo BITE para que estos a su vez controlen el resto de los módulos.
02	Planificador	Módulo que gestiona el orden de activación del conjunto de módulos del sistema y el tiempo que estarán activos.
03	Carga de Datos	Módulo que se encarga de implementar el protocolo de carga de los diferentes módulos de software
04	Gestor de interfaces	Módulo que controla los datos I/O hacia interfaces externas del sistema
05	BITE	Módulo encargado de gestionar el envío de mensajes de mantenimiento y resultados de tests internos a través de interfaces externas
06	Monitorización	Módulo encargado de detectar cualquier malfuncionamiento en los módulos operacionales
07	NAV MOD	Módulo operacional base, encargado de gestionar la configuración y los modos de funcionamiento del resto de módulos operacionales
08	IR MOD	Módulo operacional encargado de realizar los cálculos de posición y velocidad mediante técnicas basadas en sensores inerciales y la extrapolación de dichos datos a lo largo del tiempo
09	ADR MOD	Módulo operacional encargado de realizar los cálculos de datos de aire, tales como velocidad del aire, presión, altitud, actitud del avión, etc, mediante técnicas basadas en sensores de datos de aire
10	GPS MOD	Módulo operacional encargado de realizar los cálculos de datos de posición y tiempo GPS, mediante técnicas basadas en la recepción de señales GPS
11	HYBRID MOD	Módulo operacional encargado de obtener parámetros híbridos a partir de datos provenientes del módulo IR, GPS y ADR, mediante técnicas basadas en filtros Kalman
12	TIME MOD	Módulo operacional encargado de calcular la referencia de tiempos y proporcionarla al resto de sistemas del avión, mediante técnicas basadas en el tiempo recibido por los satélites GPS y la extrapolación de dicho tiempo

Tabla 30. Módulos Componentes de la Arquitectura Software.

Debido a la complejidad del diseño, no se establecerán únicamente funcionalidades y/o responsabilidades, sino módulos que realizan multitud de funciones, y de éstas se escogerán algunas para establecer los escenarios de calidad. Por tanto, se hace la hipótesis de que las funcionalidades son equivalentes a los módulos.

El esquema de arquitectura debido a las relaciones entre los diferentes módulos (según el intercambio de datos o llamadas a procedimientos y funciones que puedan hacer) es el siguiente: (Figura 88)

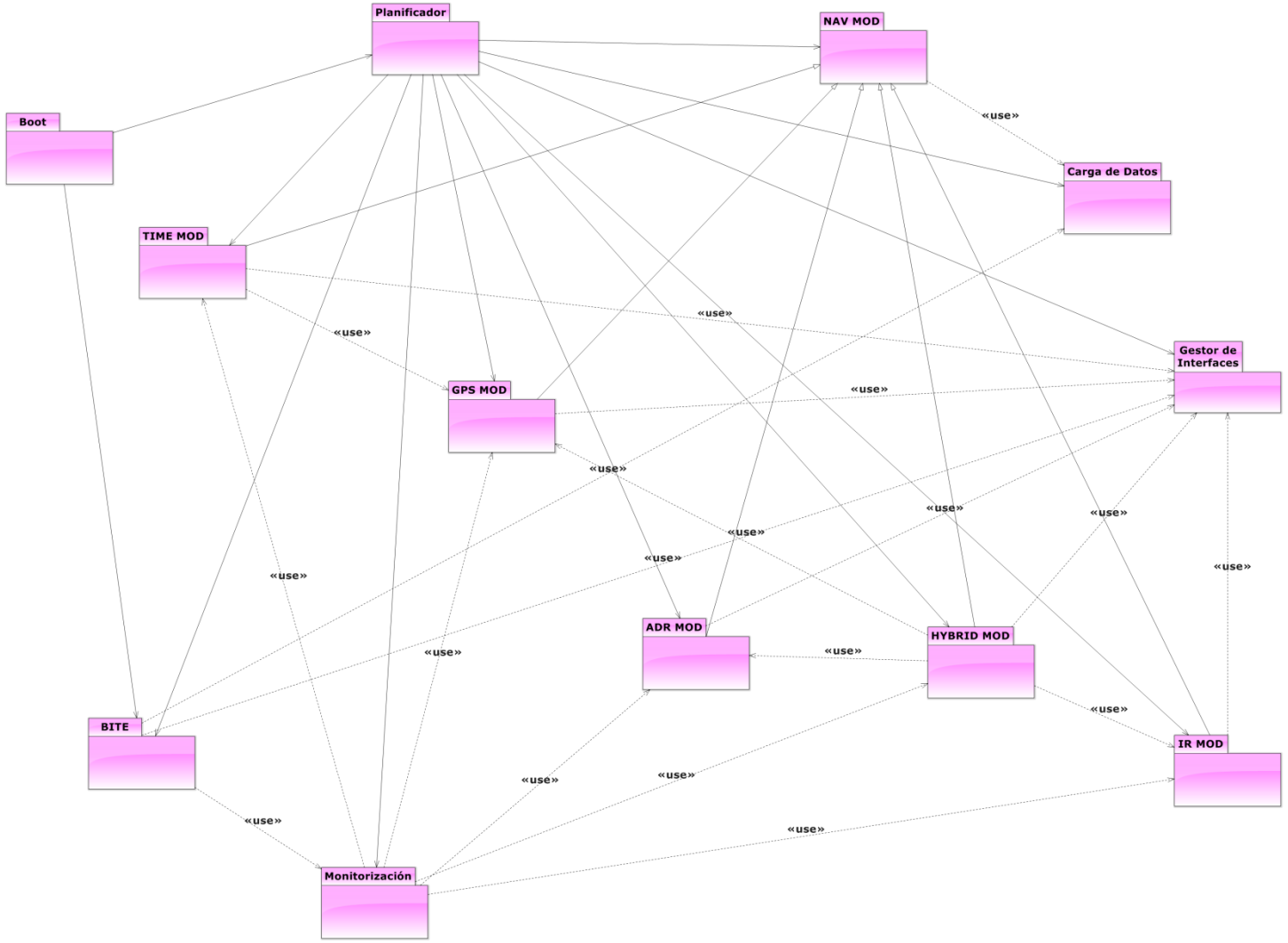


Figura 88. Arquitectura Software – Sistema de Aviónica.

7.3.1.3 Escenarios de Calidad

Los escenarios de calidad propuestos son de rendimiento y de modificabilidad, y son los siguientes:

- Escenario 1

P1 – Los datos de posición son enviados desde diferentes módulos al módulo híbrido. El período de recepción máximo de todos los datos de cada uno de los módulos es de 100 ms. El módulo híbrido debe ser capaz de sacar datos antes de 90 ms. El escenario afectará a los módulos de IR MOD, GPS MOD, ADR MOD y HYB MOD

Elemento	Atributo/Valor	Tipo	Unidad	Valor
Estímulo	Recepción datos posición	Periódico	ms	100
Fuente del Estímulo	Módulos IR, GPS y ADR	Sistema	-	
Entorno	En condiciones normales	En condiciones normales	-	
Artefacto	Sistema	Sistema	-	
Respuesta	Se procesa la posición	Latencia de la Tarea	-	
Medida de la Respuesta	Antes de 90 ms	Caso peor	ms	90

Tabla 31. Escenario de Rendimiento P1.

- Escenario 2

P2 – El módulo de tiempo tiene que proporcionar datos de referencia de tiempo a todos los sistemas con una cadencia de 60 ms. Dichos datos de tiempo estarán disponibles en el módulo GPS con un período de 80 ms. El escenario afectará a los módulos de GPS MOD y TIME MOD

Elemento	Atributo/Valor	Tipo	Unidad	Valor
Estímulo	Recepción datos GPS	Periódico	ms	80
Fuente del Estímulo	GPS	Sistema	-	
Entorno	En condiciones normales	En condiciones normales	-	
Artefacto	Sistema	Sistema	-	
Respuesta	Se procesa el tiempo	Latencia de la Tarea	-	
Medida de la Respuesta	Antes de 60 ms	Caso peor	ms	60

Tabla 32. Escenario de Rendimiento P2.

- Escenario 3

M1 - Modificar un módulo operacional, como pueda ser el caso del MOD IR, para agregar nuevas funcionalidades o bien un nuevo filtro de datos, afectaría también al NAV MOD. El coste de la modificación se estima en 25 días

Elemento	Atributo/Valor
Estímulo	Realizar una modificación en el IR MOD.
Fuente del Estímulo	Ingeniero SW
Entorno	En tiempo de diseño o mantenimiento
Artefacto	Sistema
Respuesta	La modificación se implementa en el IR MOD
Medida de la Respuesta	En 25 días

Tabla 33. Escenario de Modificabilidad M1.

- Escenario 4

M2 - La modificación del protocolo de carga de software en el equipo afectaría a los módulos de carga de datos, módulo BITE y NAV MOD. El coste se estima en unos 20 días.

Elemento	Atributo/Valor
Estímulo	Modificación del protocolo de carga de software
Fuente del Estímulo	Ingeniero SW
Entorno	En tiempo de diseño o mantenimiento
Artefacto	Sistema
Respuesta	Modificación implementada dentro del modelo de carga de datos
Medida de la Respuesta	En 20 días

Tabla 34. Escenario de Modificabilidad M2.

7.3.2 Análisis de la Arquitectura con ArchE

7.3.2.1 Definición de Funciones y Responsabilidades

Como ya se ha comentado anteriormente, las funciones y las responsabilidades van a ser equivalentes en este caso a los módulos software definidos en la Tabla 30. Una vez introducidos en Arche, quedarán como se refleja en la Figura 89:

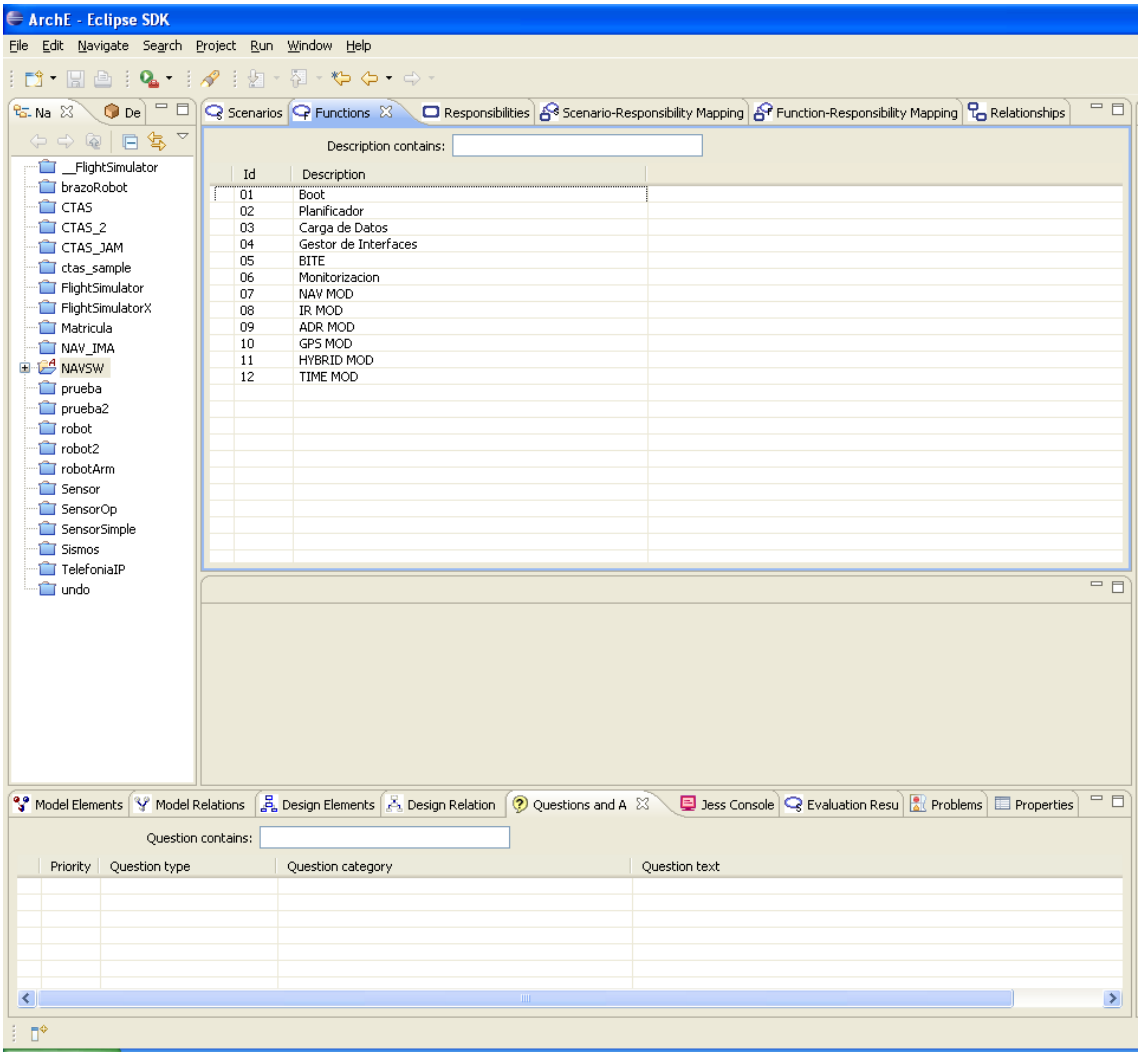


Figura 89. Funciones del Software del Sistema de Aviónica en ArchE.

A continuación, ArchE genera automáticamente las Responsabilidades, y el mapeo funciones-responsabilidades, como se muestra en la Figura 90 y en la Figura 91:

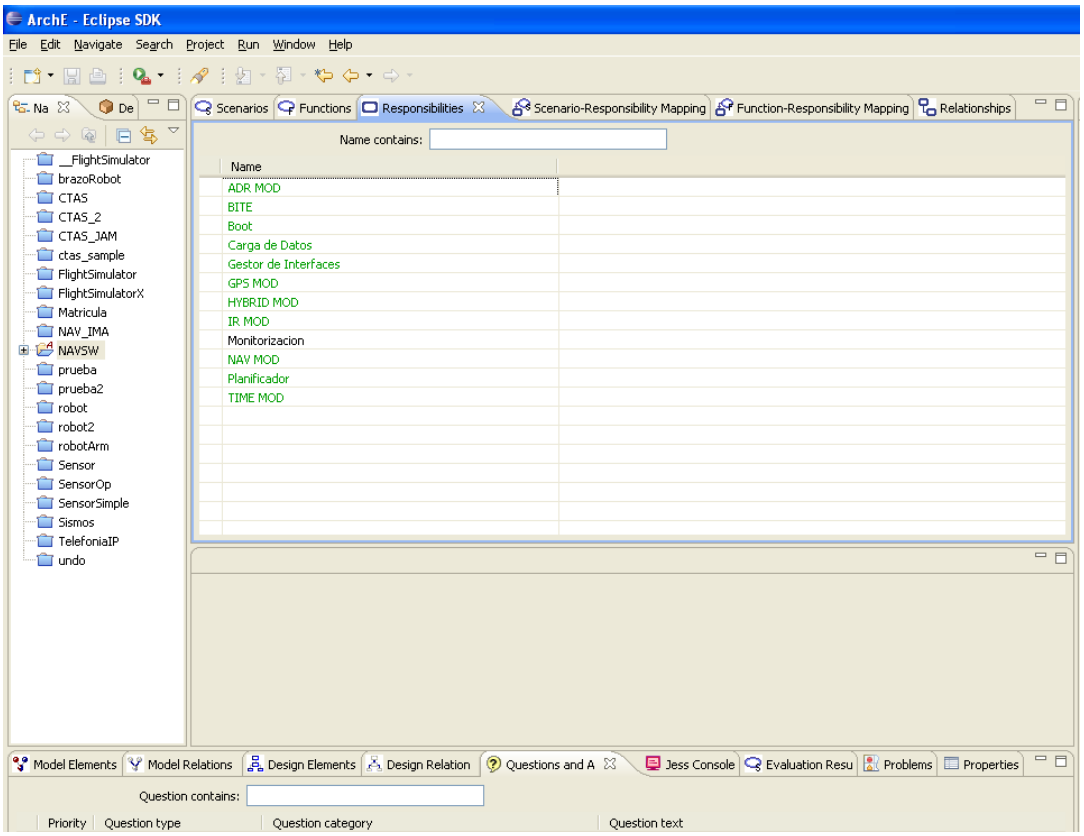


Figura 90. Responsabilidades del Software del Sistema de Aviónica en ArchE.

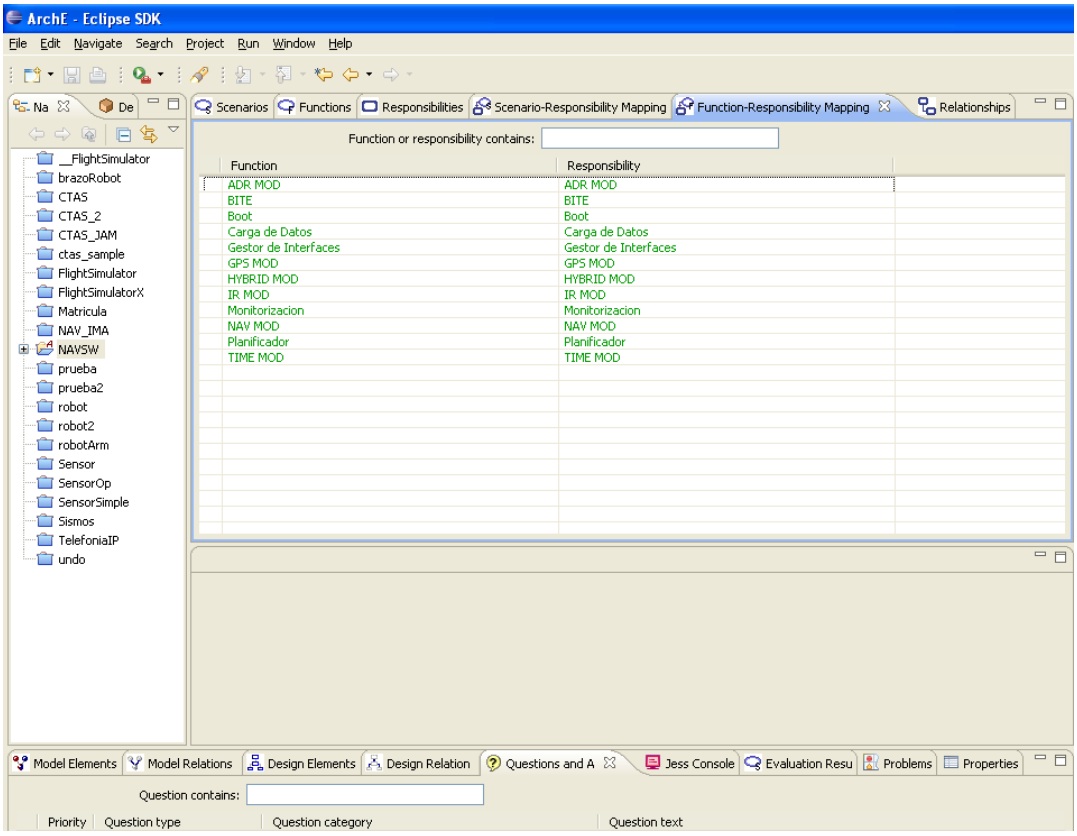


Figura 91: Mapeo Funciones-Responsabilidades del Software del Sistema de Aviónica.

7.3.2.2 Definición de Relaciones

Las relaciones entre las diferentes responsabilidades son las siguientes:

- Relaciones de dependencia entre responsabilidades

Al igual que en el ejemplo anterior, representan el impacto de la modificabiliad de un componente sobre otros. En la Tabla 35 se muestran las diferentes relaciones de dependencia:

	Child Responsability											
Parent Responsability	Boot	Planificador	Carga de datos	Gestor de Interfaces	BITE	Monitorización	NAV MOD	IR MOD	ADR MOD	GPS MOD	HYBRID MOD	TIME MOD
Boot		D			D							
Planificador			D	D	D	D	D	D	D	D	D	D
Carga de Datos					D		D					
Gestor de interfaces					D			D	D	D	D	D
BITE												
Monitorización					D			D	D	D	D	D
NAV MOD								D	D	D	D	D
IR MOD											D	
ADR MOD											D	
GPS MOD											D	D
HYBRID MOD												
TIME MOD												

Tabla 35. Relaciones de Dependencia de las Responsabilidades – Sistema de Aviónica.

- Relaciones de Reacción

Se utilizan para analizar el marco de razonamiento de rendimiento. Representan que, para medir el efecto de la ejecución de un determinado evento en la planificación de las diversas tareas del sistema, una responsabilidad se ejecuta antes que otra.

En la Tabla 36 se muestran las diferentes relaciones de reacción:

	Child Responsability											
Parent Responsability	Boot	Planificador	Carga de datos	Gestor de Interfaces	BITE	Monitorización	NAV MOD	IR MOD	ADR MOD	GPS MOD	HYBRID MOD	TIME MOD
Boot		R			R							
Planificador			R	R	R	R	R	R	R	R	R	R
Carga de Datos												
Gestor de interfaces												
BITE												
Monitorización					R							
NAV MOD												
IR MOD											R	
ADR MOD											R	
GPS MOD											R	R
HYBRID MOD												
TIME MOD												

Tabla 36. Relaciones de Dependencia de las Responsabilidades – Sistema de Aviónica.

7.3.2.3 Definición de Escenarios

Se introducen los escenarios en ArchE, tal y como muestra la Figura 92:

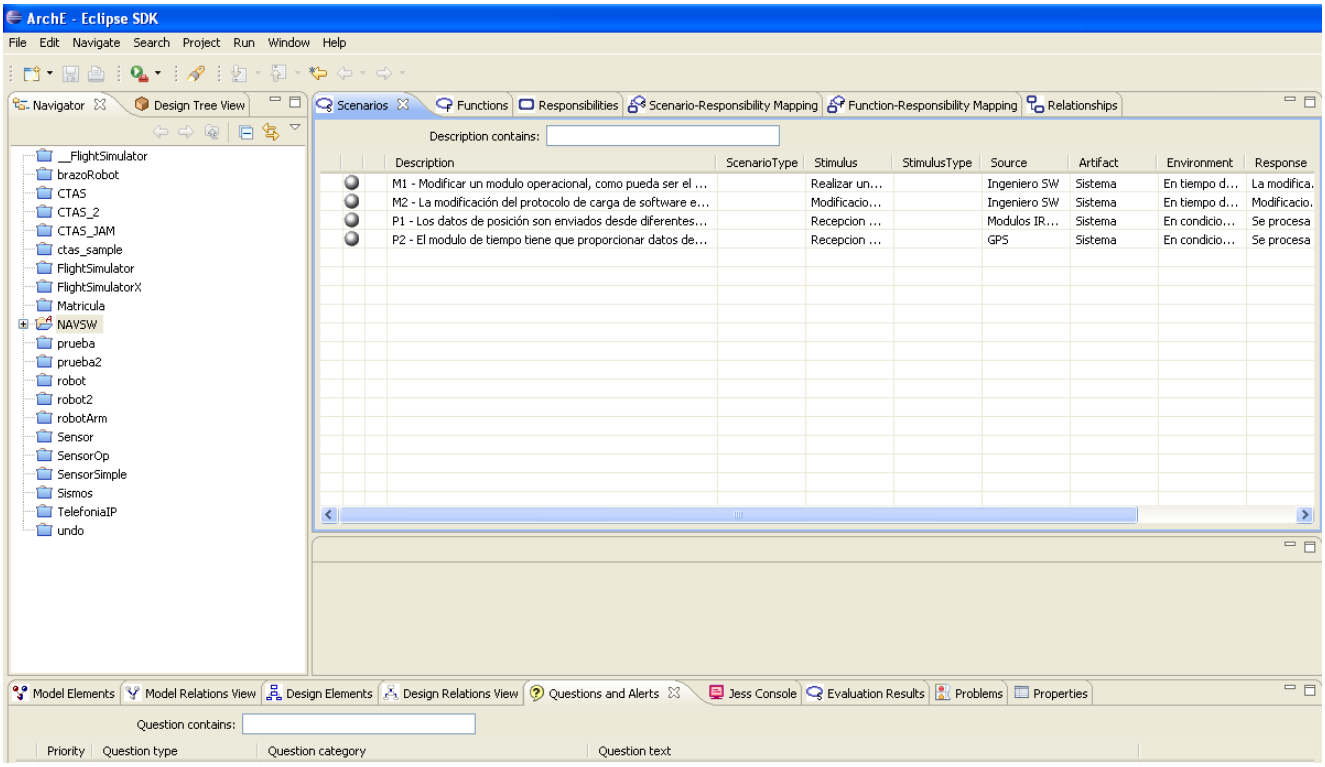


Figura 92. Escenarios en ArchE – Sistema de Aviónica.

El escenario M1 quedaría de la siguiente manera: (Figura 93)

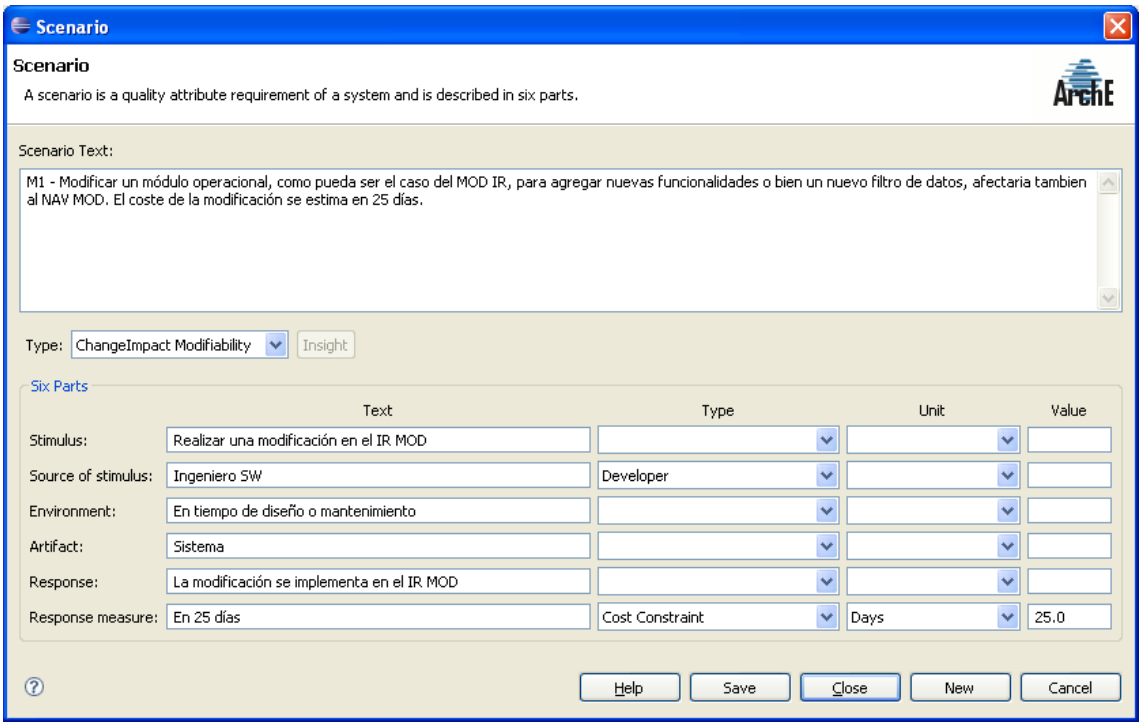


Figura 93. Escenario M1 del Software del Sistema de Aviónica.

El escenario M2 quedaría de la siguiente manera: (Figura 94)

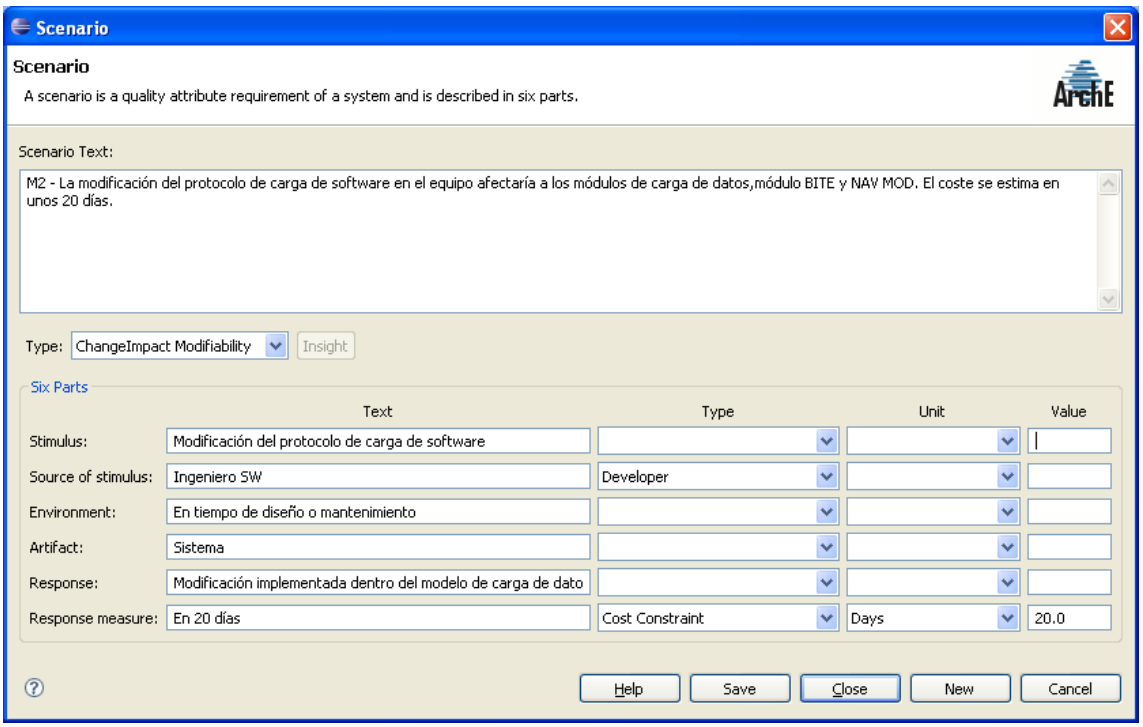


Figura 94. Escenario M2 del Software del Sistema de Aviónica.

El escenario P1 quedaría de la siguiente manera: (Figura 95)

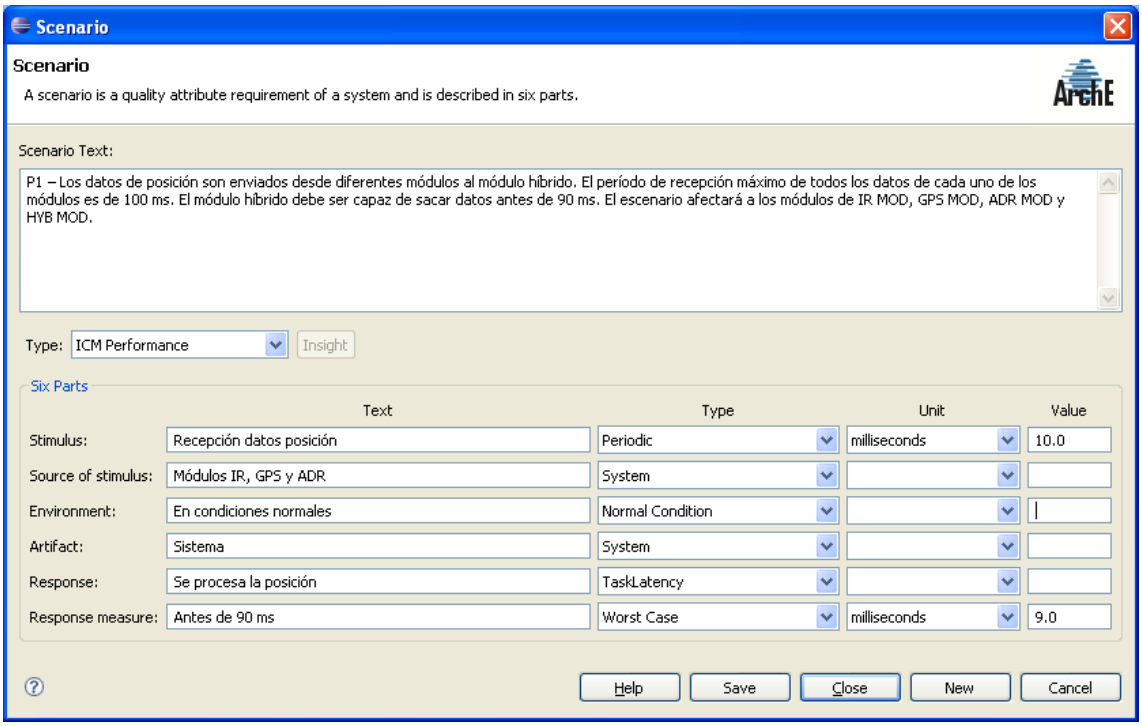


Figura 95. Escenario P1 del Software del Sistema de Aviónica.

El escenario P2 quedaría de la siguiente manera: (Figura 96)

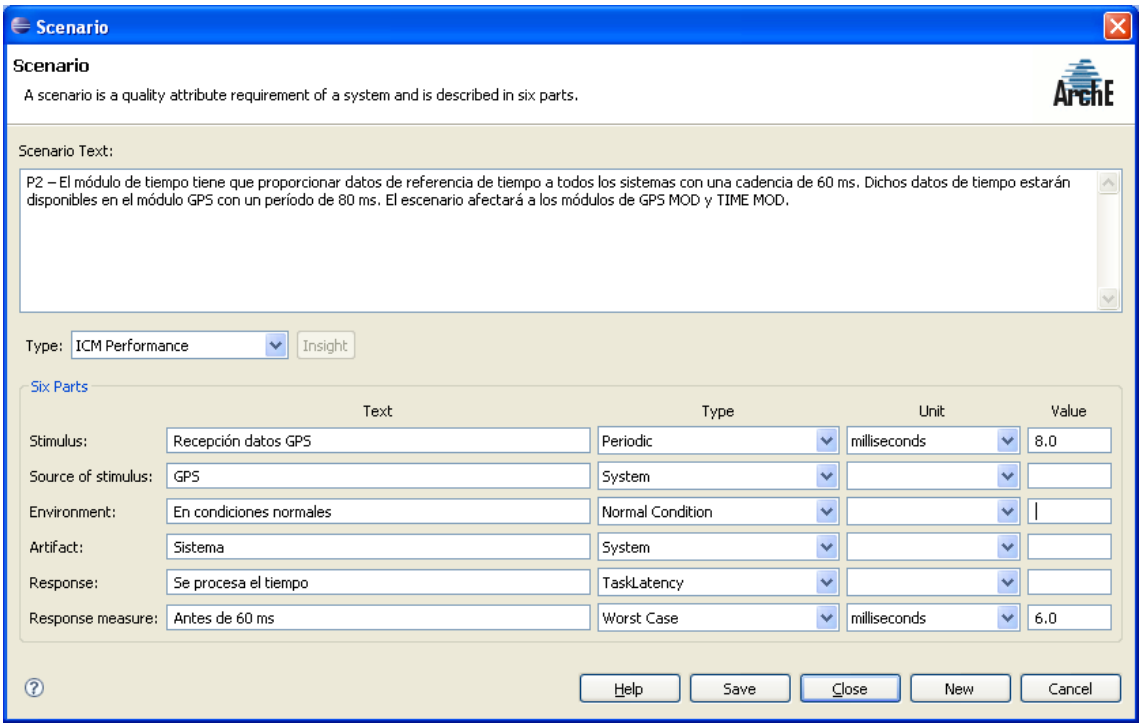


Figura 96. Escenario P2 del Software del Sistema de Aviónica.

7.3.2.4 Mapeo de Escenarios a Responsabilidades

Se mapean los escenarios a las responsabilidades, según la siguiente Tabla 37:

	Escenarios			
Responsabilidades	**M1**	**M2**	**P1**	**P2**
NAV MOD	X	X		
IR MOD	X		X	
HYBRID MOD			X	
GPS MOD			X	X
TIME MOD				X
ADR MOD			X	
BITE		X		
Carga de Datos		X		

Tabla 37. Mapeado de Escenarios a Responsabilidades.

Se introduce el mismo mapeado en ArchE: (Figura 97)

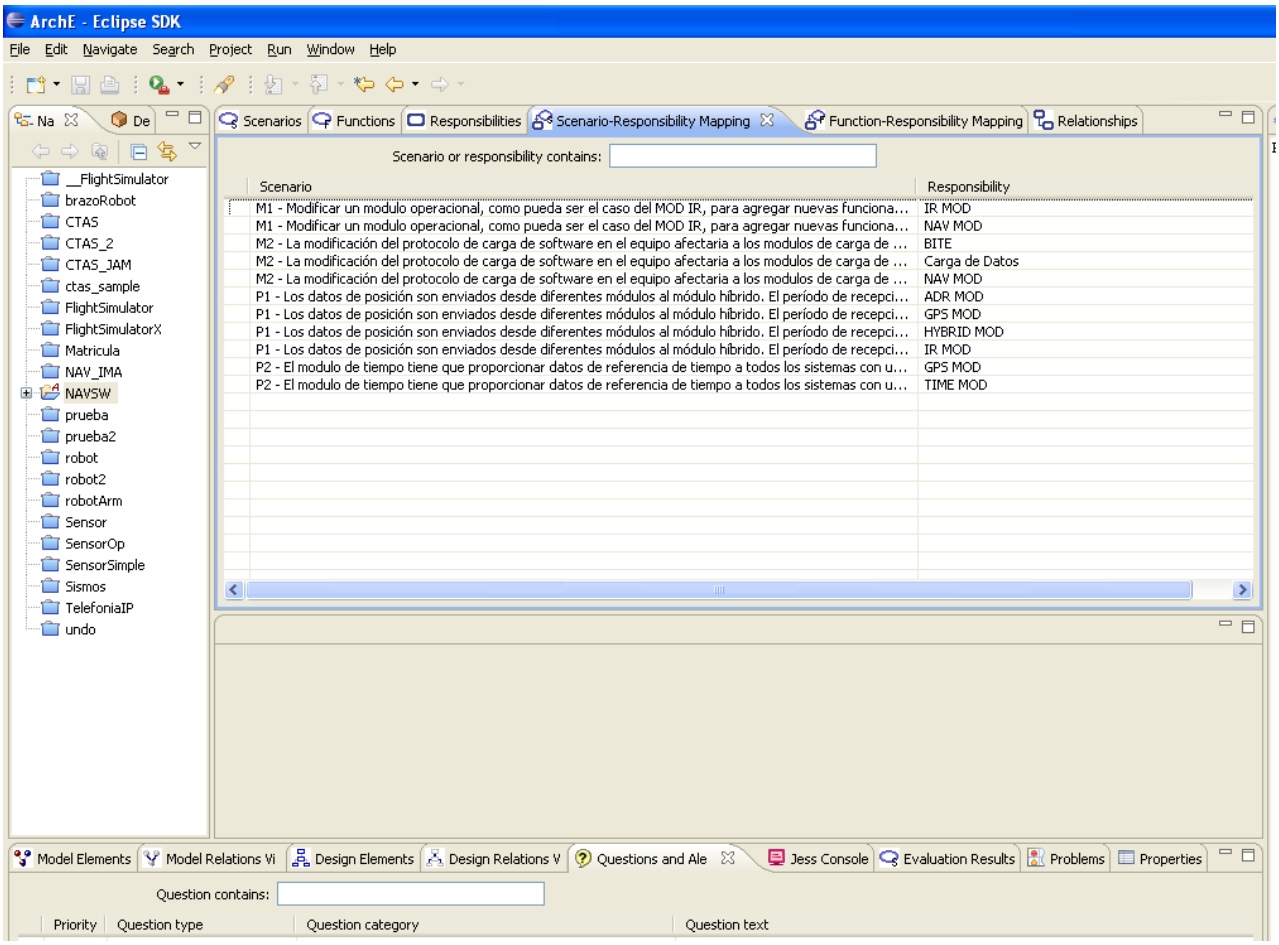


Figura 97. Mapeado de Escenarios y Responsabilidades en ArchE – Sistema de Aviónica.

7.3.2.5 Análisis de ArchE

Una vez Introducidos las responsabilidades y los escenarios, se arrancan los marcos de razonamiento y se obtiene el siguiente resultado: (Figura 98, Figura 99, Figura 100, Figura 101, Figura 102 y Figura 103)

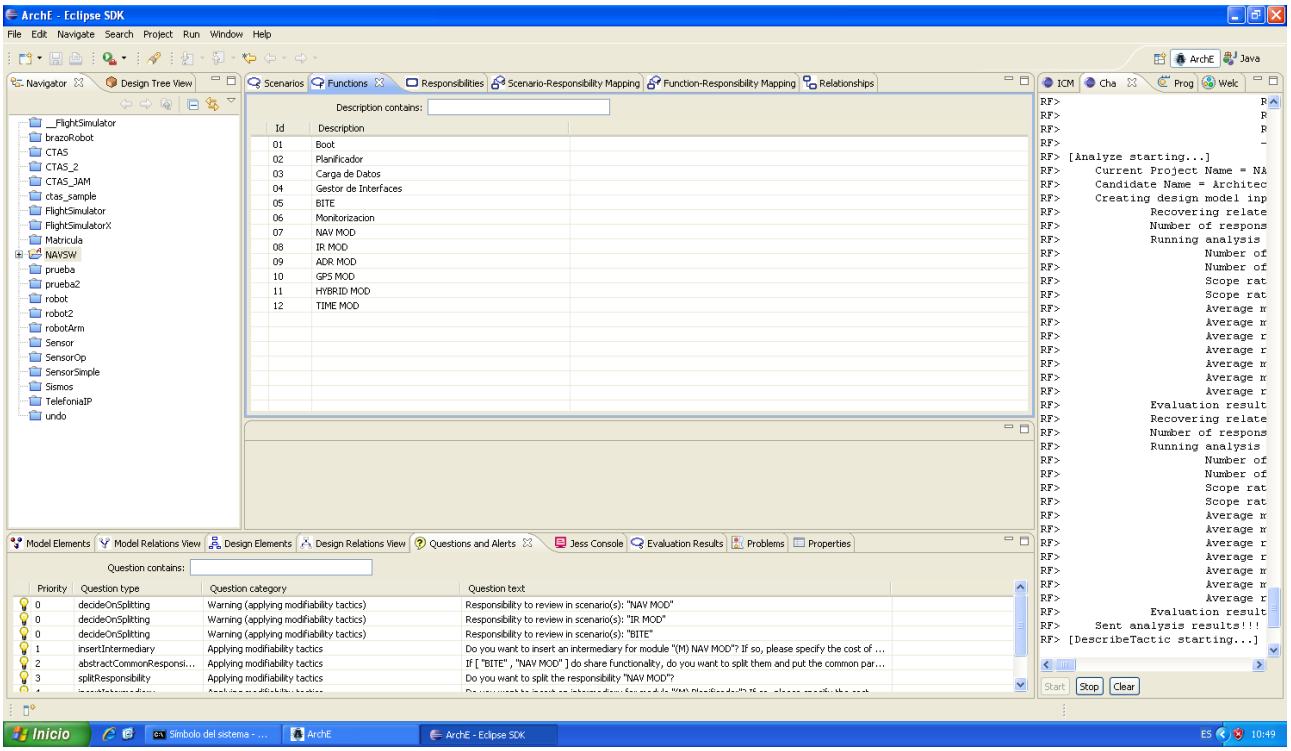


Figura 98. Funciones del Software del Sistema de Aviónica con los Marcos de Razonamiento Activos.

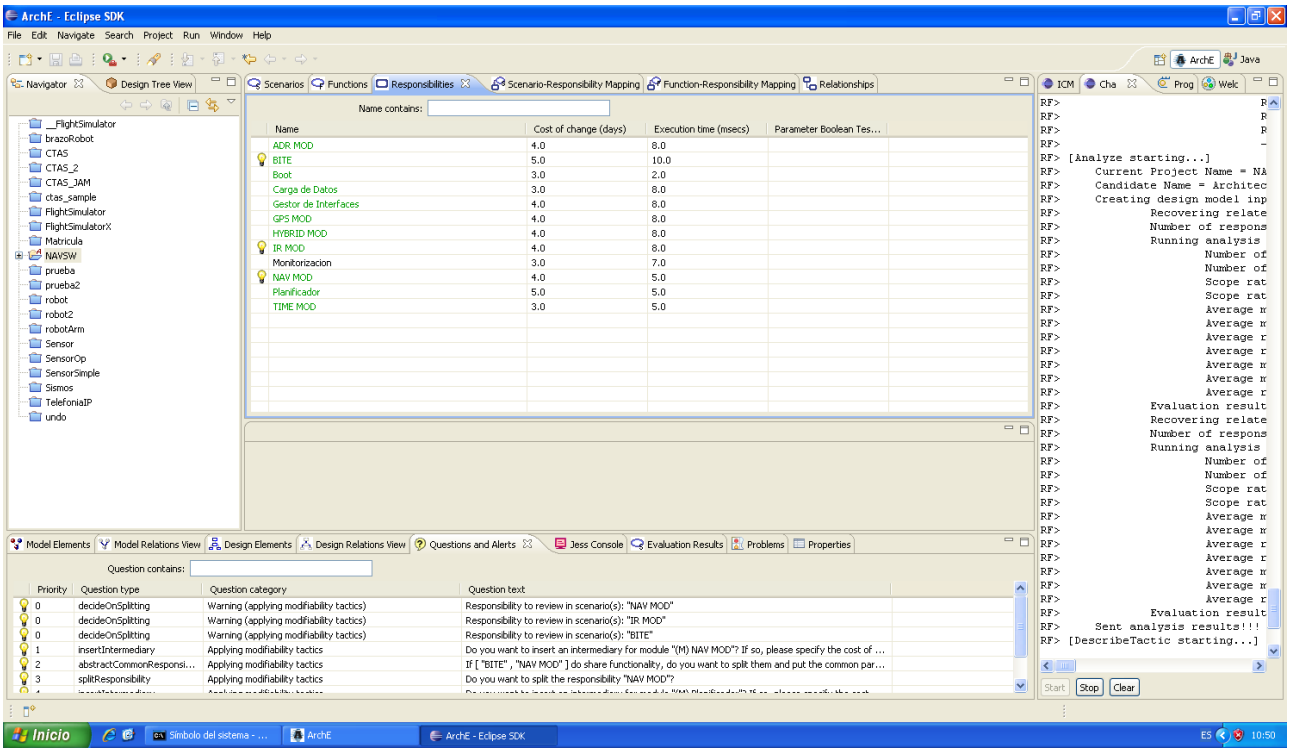


Figura 99. Responsabilidades del Software del Sistema de Aviónica con los Marcos de Razonamiento Activos

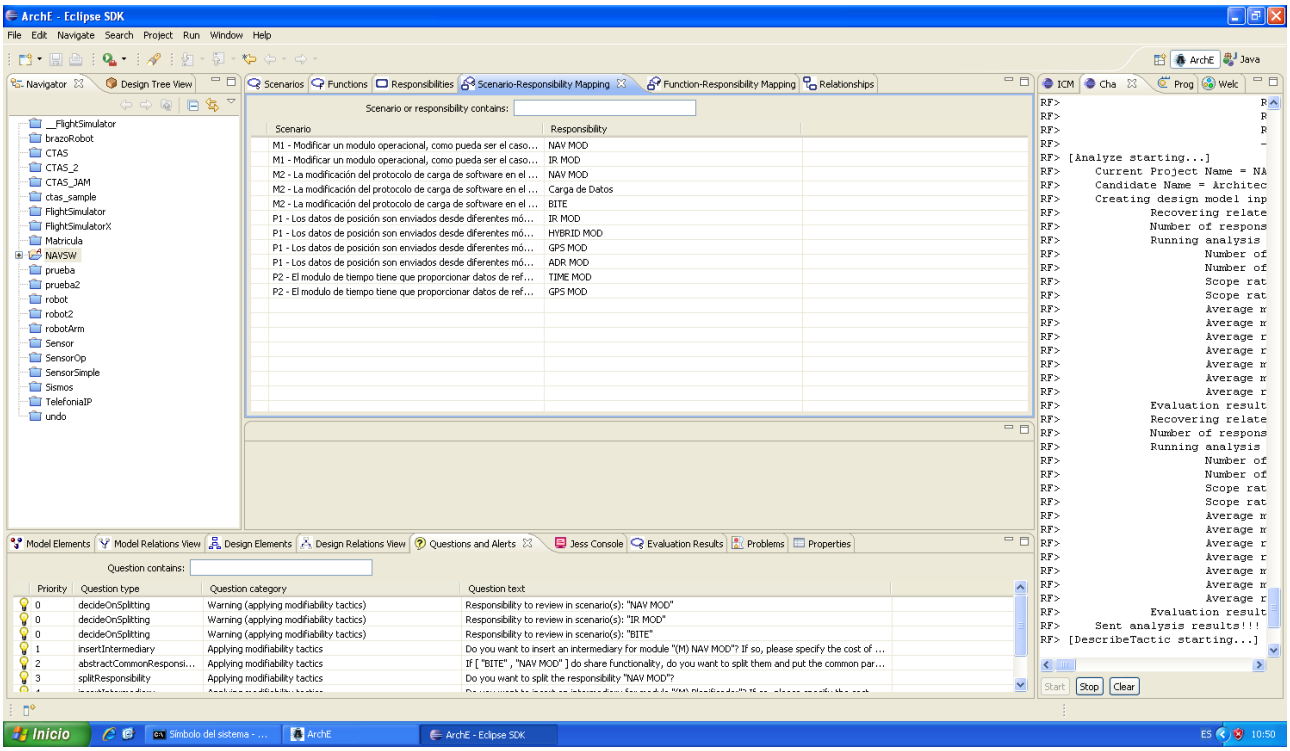


Figura 100. Mapeo Escenarios-Responsabilidades del Software del Sistema de Aviónica con los Marcos de Razonamiento Activos.

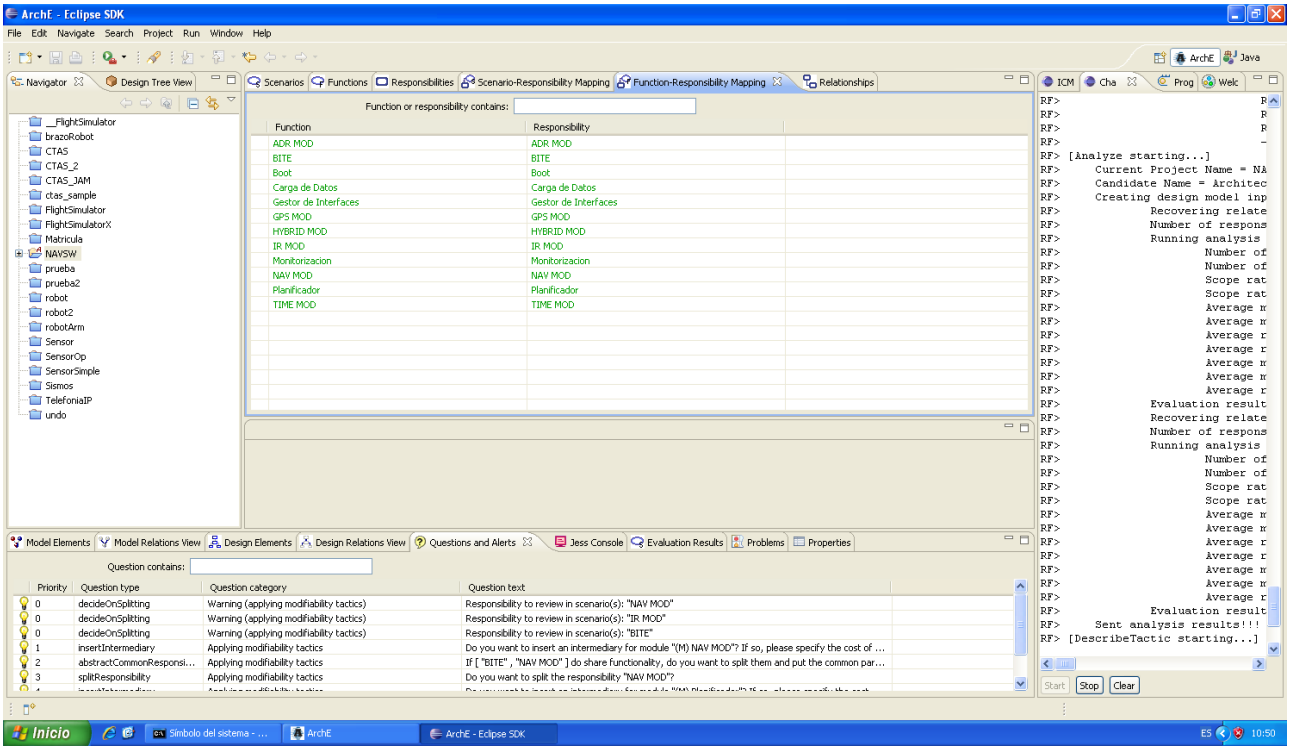


Figura 101. Mapeo Funciones-Responsabilidades del Software del Sistema de Aviónica con los Marcos de Razonamiento Activos

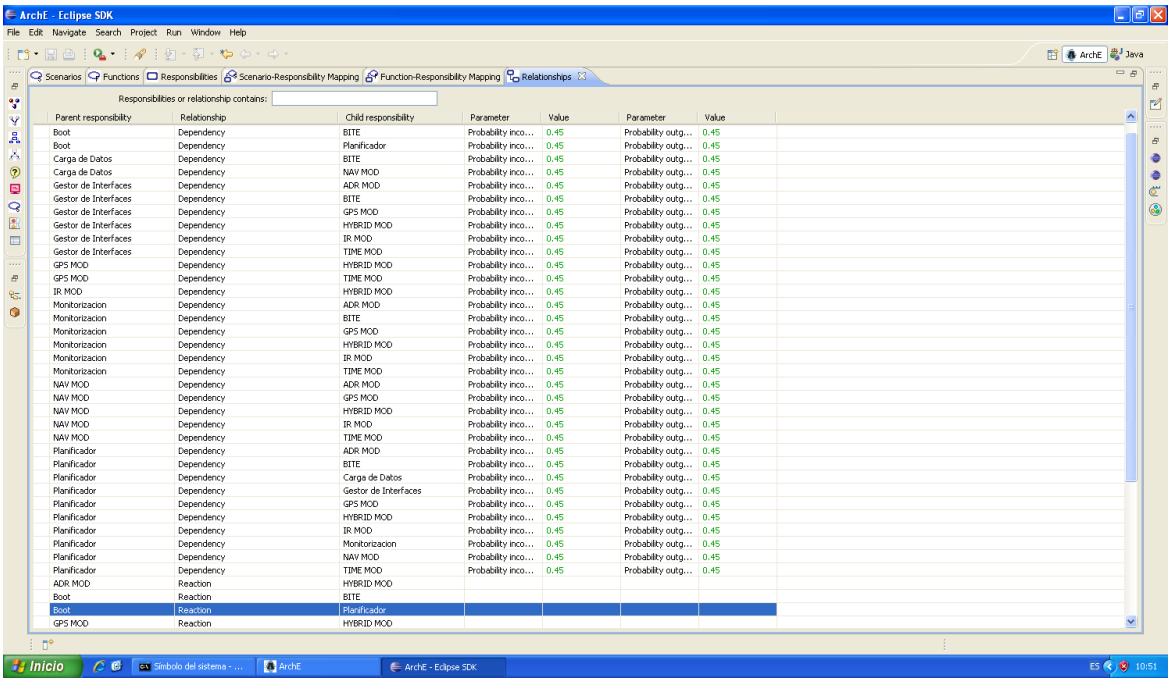


Figura 102. Relaciones entre Responsabilidades del Software del sistema de Aviónica – I.

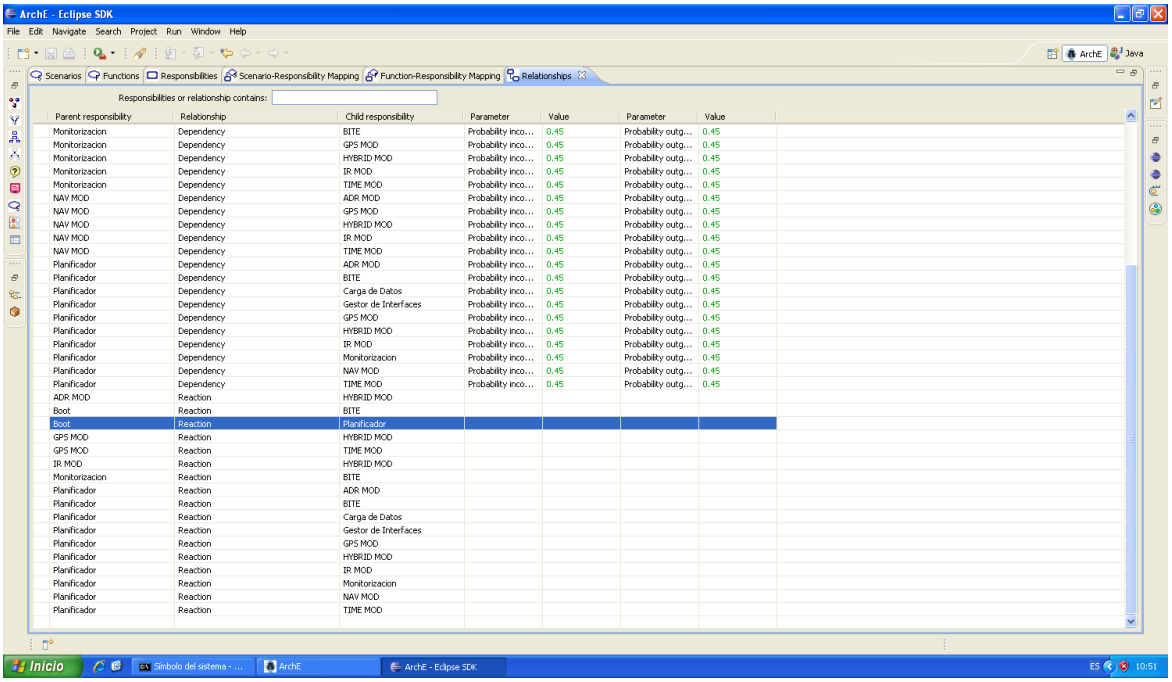


Figura 103. Relaciones entre Responsabilidades del Software del sistema de Aviónica – II.

Hay que hacer notar que, debido a una limitación de ArchE, los tiempos de ejecución de las responsabilidades han de estar en el intervalo [0 – 10] milisegundos; como esta medida no es aplicable al sistema de aviónica, que maneja una escala superior, en general entre [0 – 500] milisegundos, para poder analizar el sistema, se han escalado todas las magnitudes dividiéndolas por 10, tanto los tiempos de ejecución como los tiempos de los escenarios; por ejemplo, 150 ms serían 15 ms en la escala de ArchE.

Como se puede ver en la Figura 104, los escenarios de performances no se cumplen en absoluto y el sistema completo no es planificable:

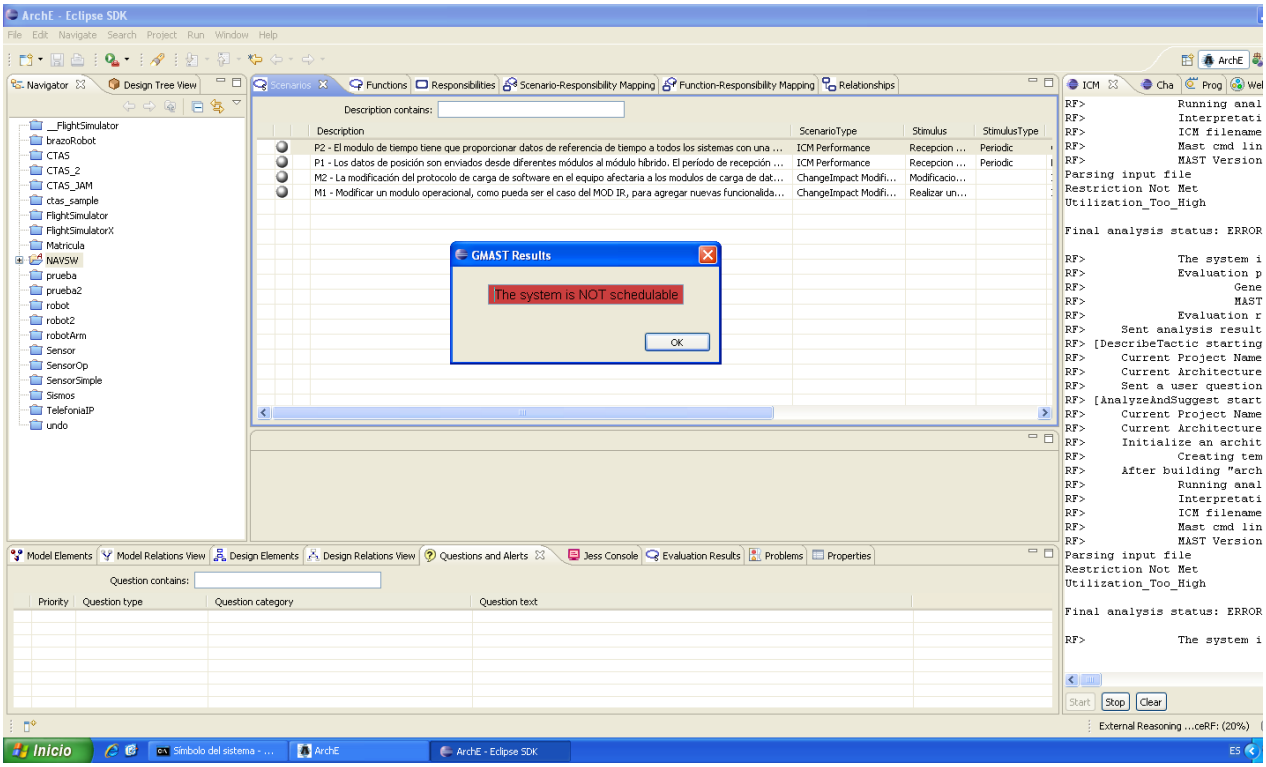


Figura 104. Ejecución inicial de ArchE.

Los escenarios no se cumplen, ni los de rendimiento ni los de modificabilidad: (Figura 105)

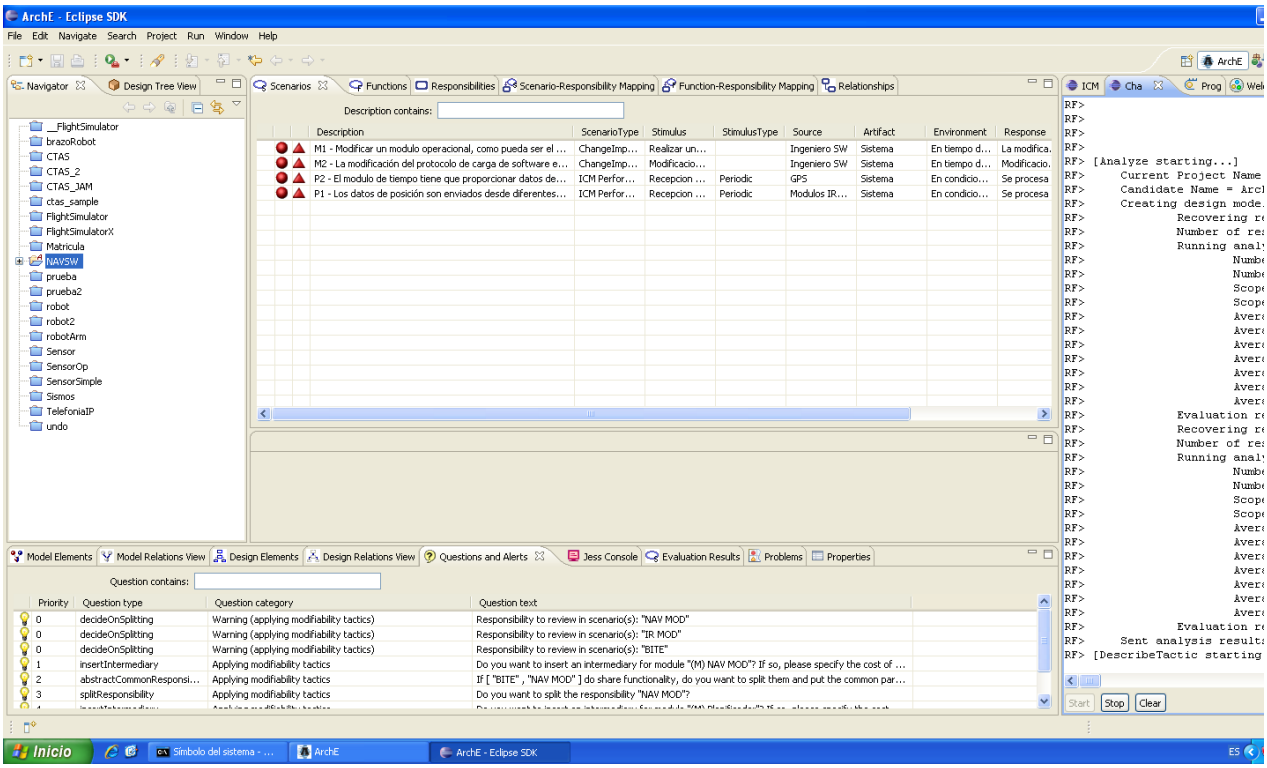


Figura 105. Resultado de la evaluación de escenarios de ArchE - Sistema de Aviónica.

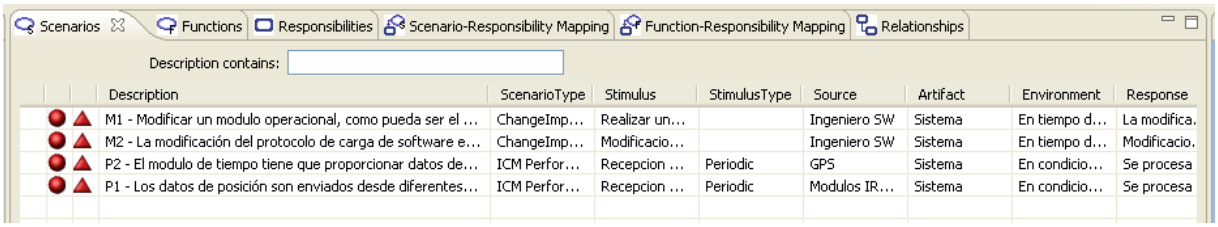


Figura 106. Resultado de la evaluación de escenarios de ArchE - Sistema de Aviónica (ampliada)

Como puede verse en la imagen, ambos marcos de razonamiento están activos y analizando la arquitectura: (Figura 107 y Figura 108)

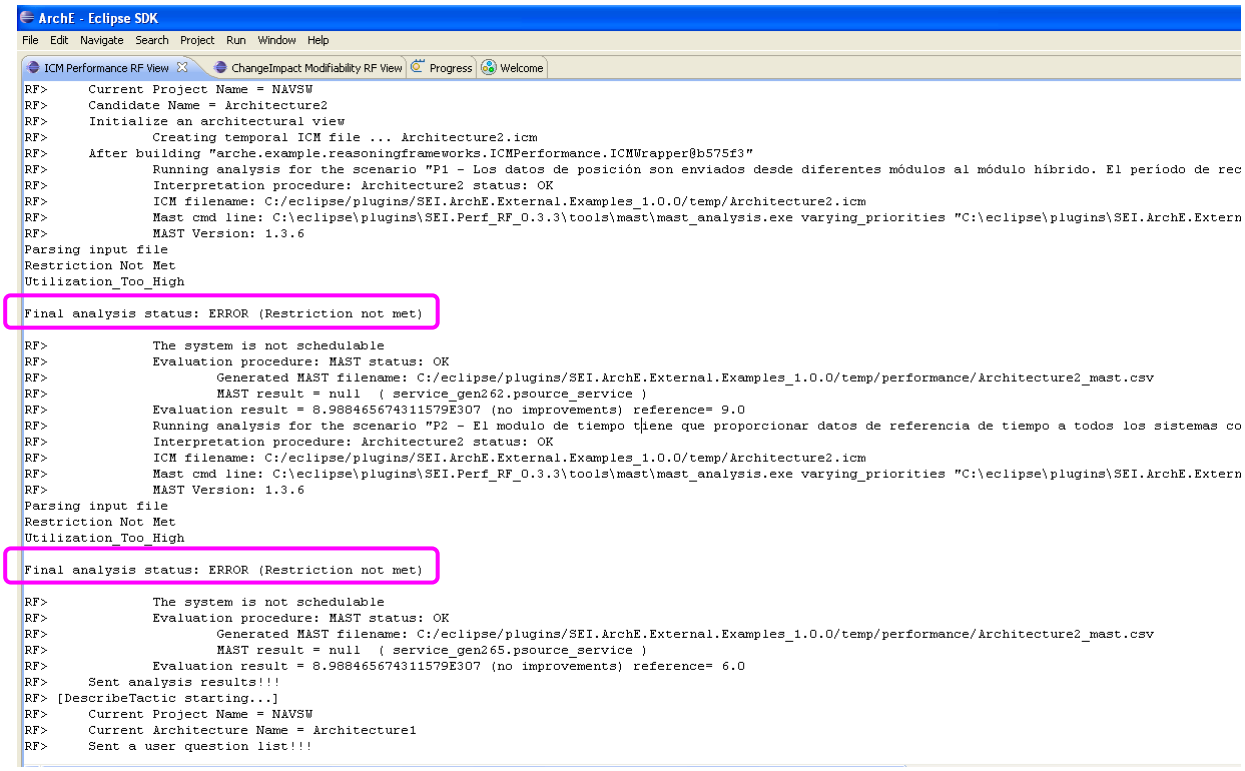


Figura 107. Marco de Razonamiento ICM Performance Activo.

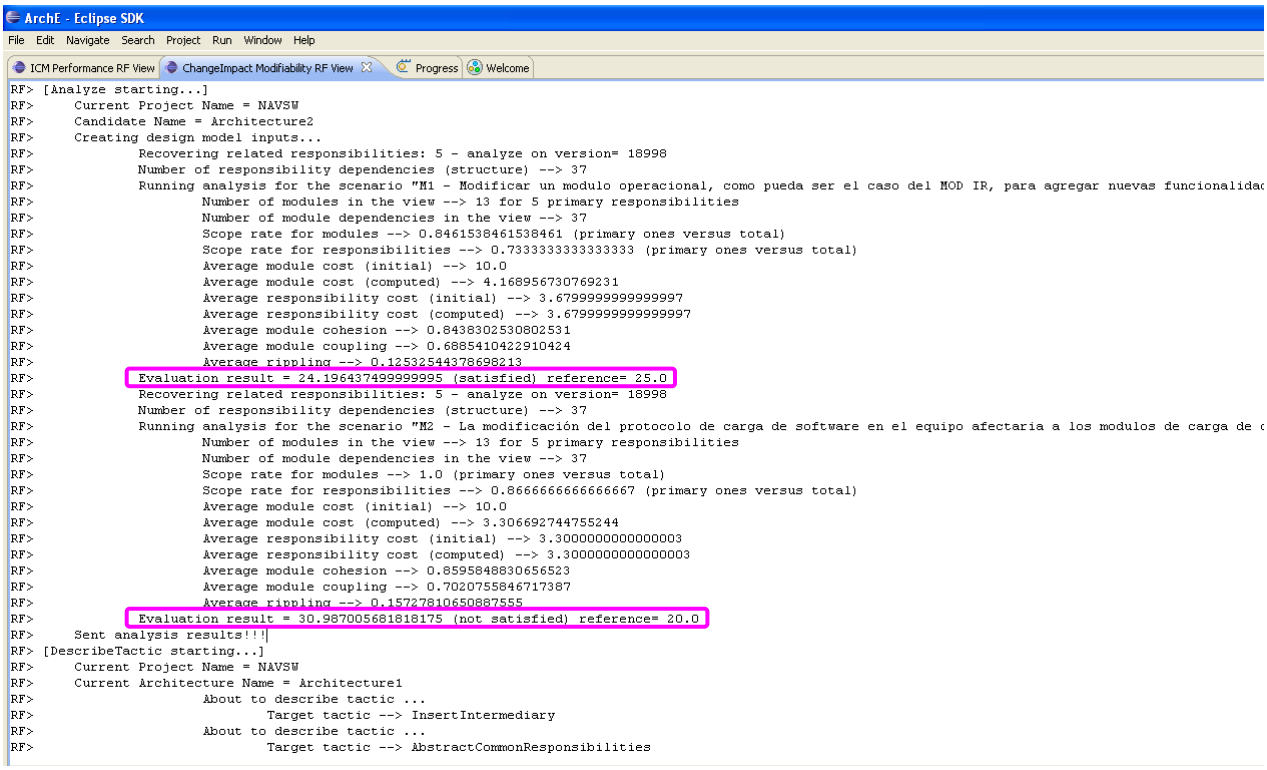


Figura 108. Marco de Razonamiento ChangeImpact Modifiability Activo.

Como el marco ICM Performance no propone ninguna táctica para cumplir los escenarios, hay que cambiar los parámetros iniciales, prestando atención a lo que indican las ventanas de los marcos de razonamiento.

Se modifican parámetros de tiempos de ejecución de responsabilidades y de escenarios; la reducción del tiempo de ejecución de las responsabilidades ayuda a cumplir las *deadlines* de los tiempos de respuesta marcados por los escenarios. El aumento de los períodos y de los tiempos de respuesta de los escenarios ayuda a cumplir la tarea. (Figura 109, Figura 110 y Figura 111)

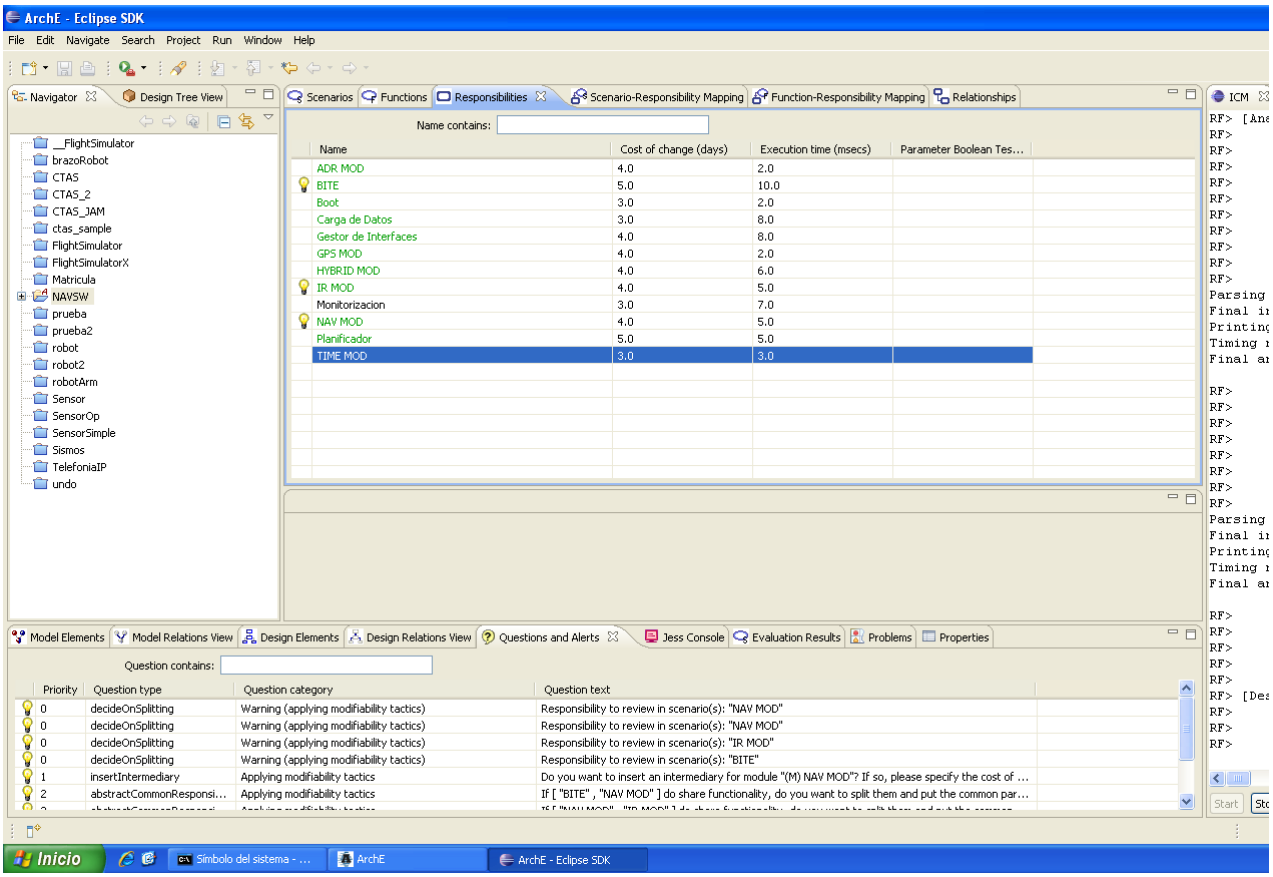


Figura 109. Modificación de los Tiempos de Ejecución.

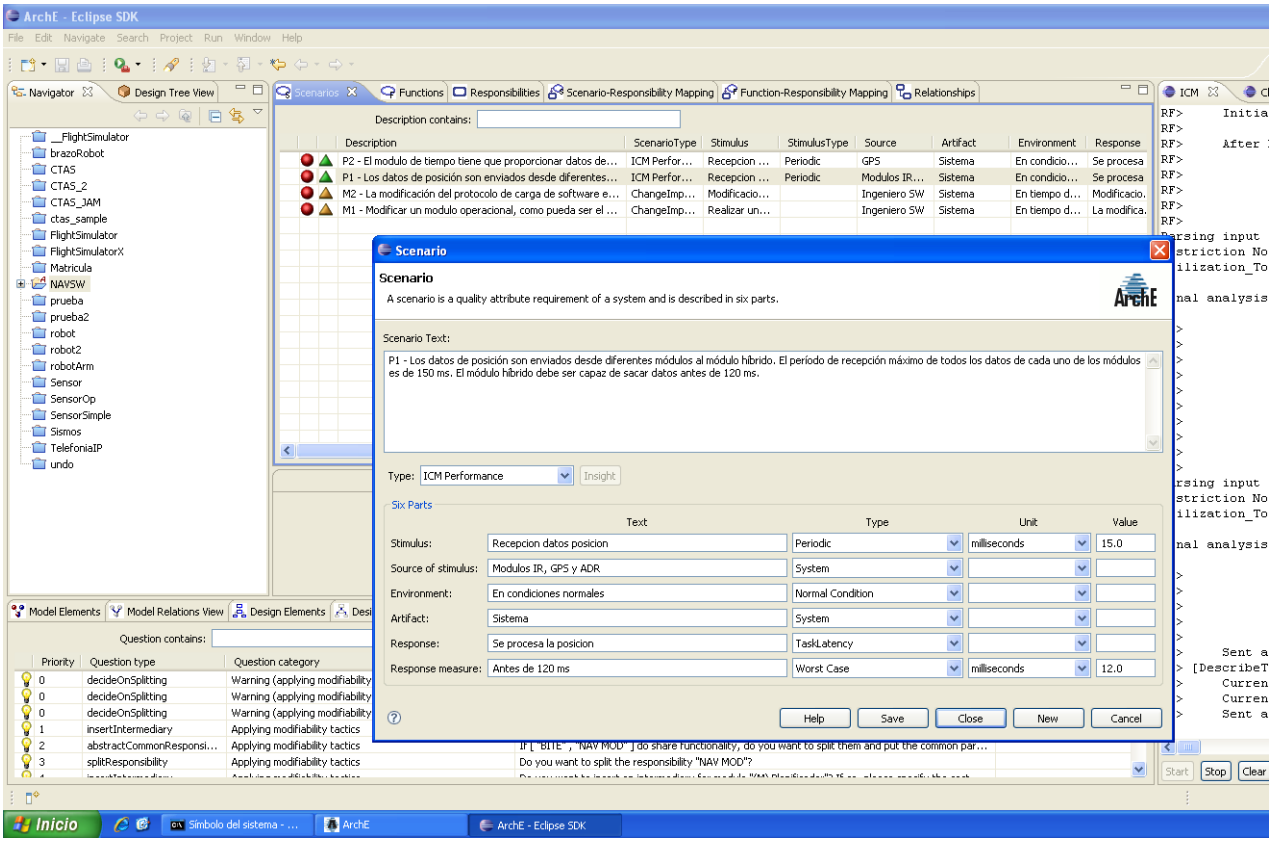


Figura 110. Modificación del Escenario P1.

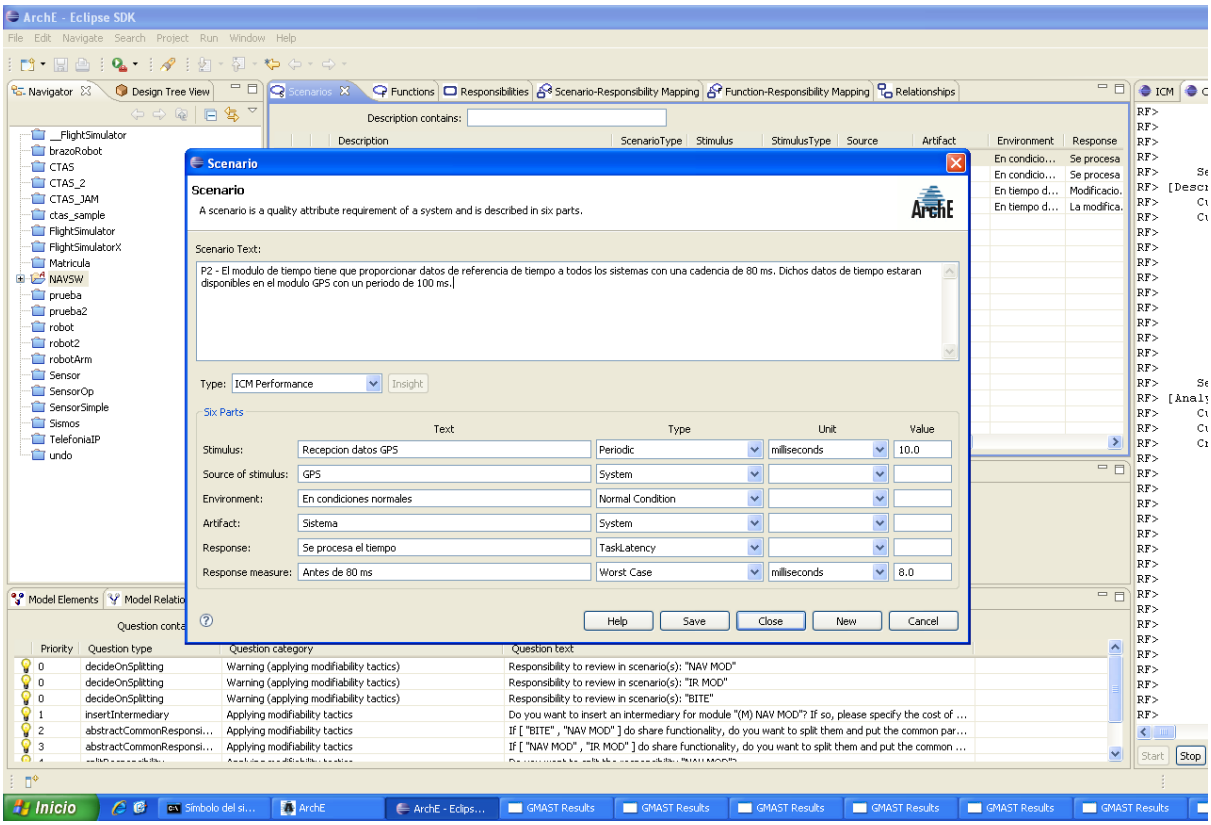


Figura 111. Modificación del Escenario P2.

Una vez modificados de nuevo, se lanzan los marcos de razonamiento de nuevo, y esta vez sí se van a cumplir los escenarios, tal y como se muestra en las ventanas de la aplicación MAST: (Figura 112)

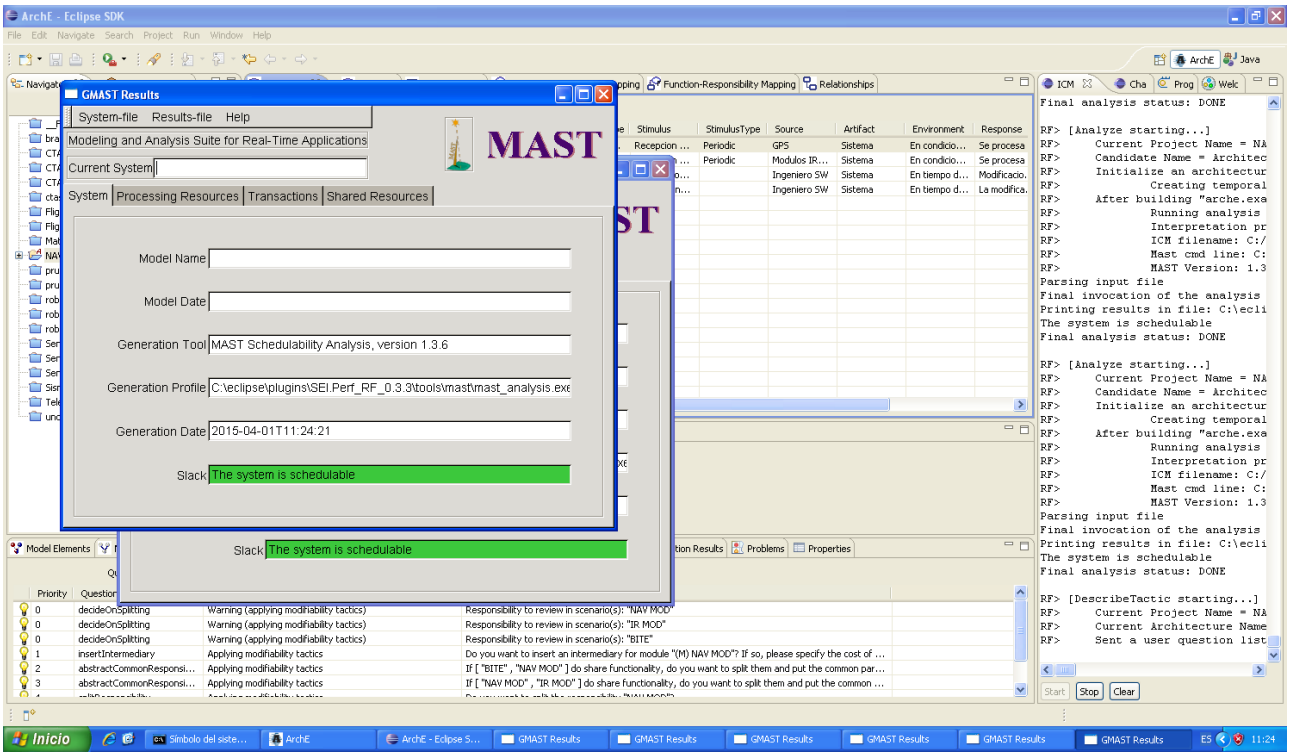


Figura 112. Ventana MAST que indica cumplimiento de Escenario.

En esta ventana de MAST se muestra como para los dos escenarios, las *deadlines* se cumplen: (Figura 113)

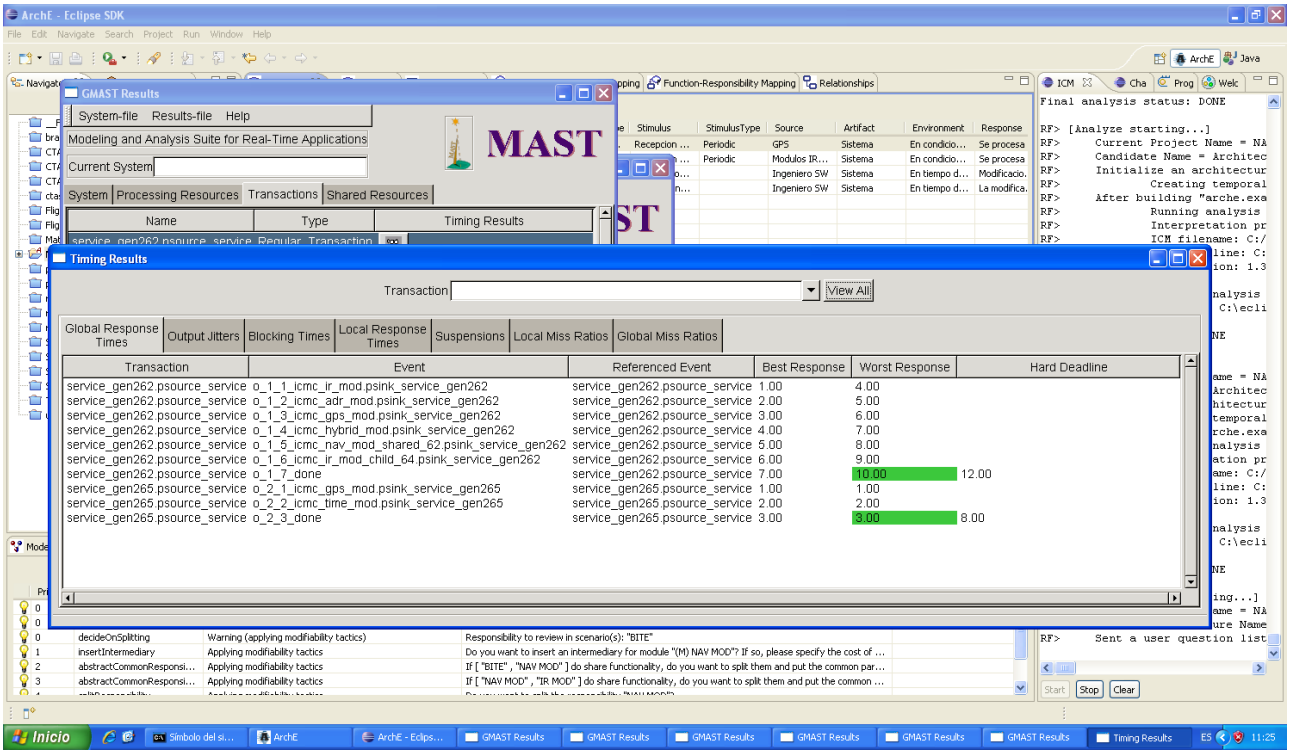


Figura 113. Ventana Ampliada de MAST.

En la ventana del marco de Razonamiento ICM Performance también se ve cómo se cumplen los escenarios: (Figura 114)

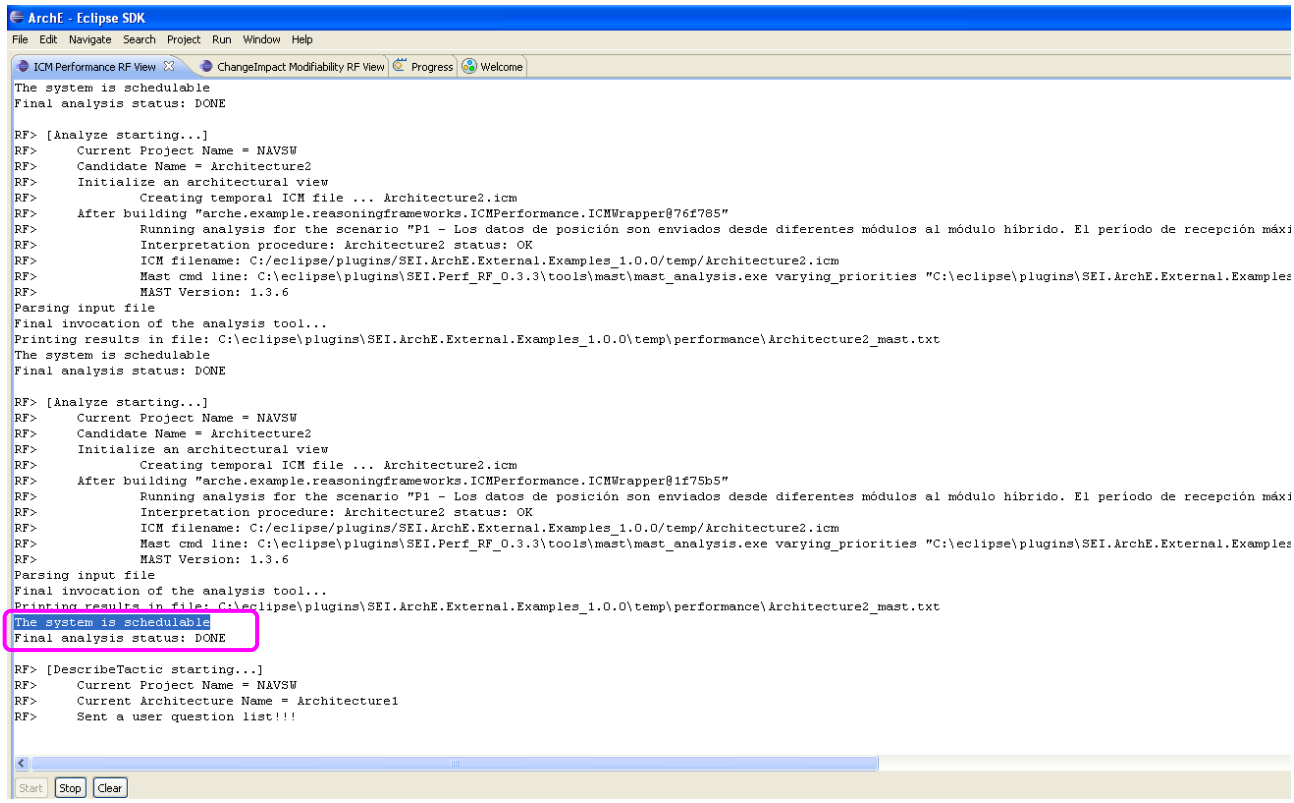


Figura 114. Ventana del marco de Razonamiento ICM Performance con Sistema Planificable.

Aunque exista el problema de configuración en la herramienta ArchE (los escenarios de rendimiento no se muestran en verde aunque se cumplan), el sistema es planificable: (Figura 115)

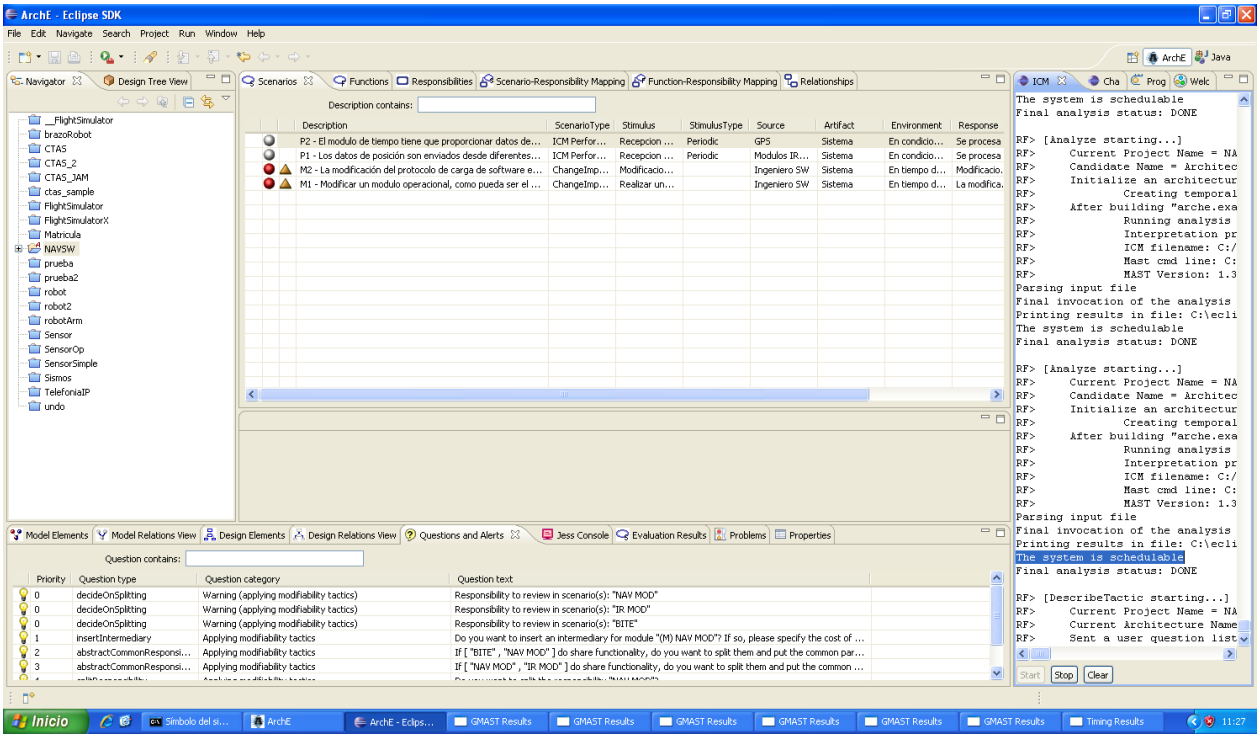


Figura 115. Escenarios de Rendimiento P1 y P2 Cumplidos.

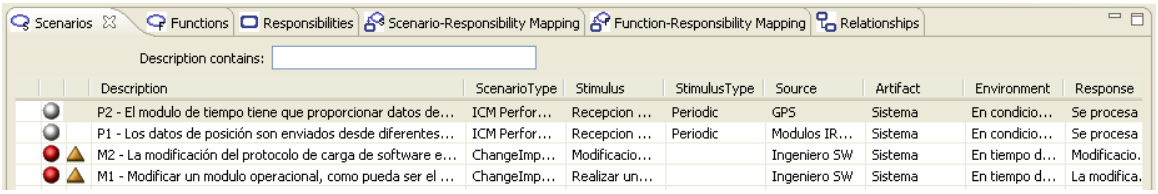


Figura 116. Escenarios de Rendimiento P1 y P2 Cumplidos (ampliado).

Añadimos un nuevo escenario de *performances* (Tabla 38), para así ver la influencia que puede tener sobre el sistema ya planificado. Para ello, el nuevo escenario afectará a módulos diferentes de los anteriores:

- Escenario 5

P3 – El módulo de BITE tiene que proporcionar el estado del sistema bajo petición exterior o bien de manera periódica cada 80 ms. El tiempo límite para que se cumpla la tarea es de 60 ms. Esta tarea implica realizar la monitorización de cada módulo; por tanto, los módulos de BITE y monitorización forman parte de la tarea.

Elemento	Atributo/Valor	Tipo	Unidad	Valor
Estímulo	Proporcionar estado del sistema	Periódico	ms	80
Fuente del Estímulo	Sistema de Mantenimiento	Sistema	-	
Entorno	En condiciones normales	En condiciones normales	-	
Artefacto	Sistema	Sistema	-	
Respuesta	Se procesa el estado del sistema y se transmite.	Latencia de la Tarea	-	
Medida de la Respuesta	Antes de 60 ms	Caso peor	ms	60

Tabla 38. Escenario de Rendimiento P3.

El mapeado de escenarios a responsabilidades quedaría de la siguiente manera: (Tabla 39)

	Escenarios				
Responsabilidades	**M1**	**M2**	**P1**	**P2**	**P3**
NAV MOD	X	X			
IR MOD	X		X		
HYBRID MOD			X		
GPS MOD			X	X	
TIME MOD				X	
ADR MOD			X		
BITE		X			X
Carga de Datos		X			
Monitorización					X

Tabla 39. Mapeado Escenarios/Responsabilidades con P3.

En Arche el nuevo escenario quedaría de la siguiente manera: (Figura 117)

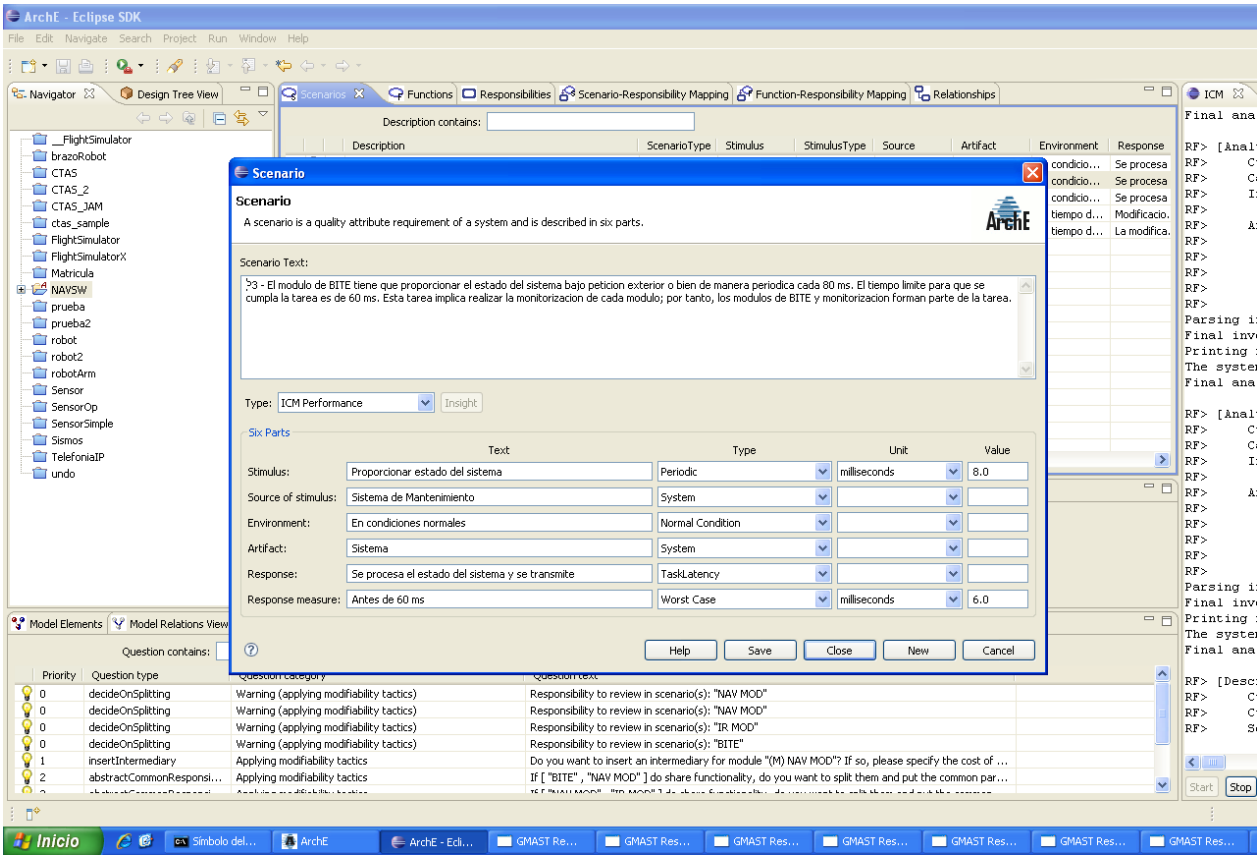


Figura 117. Nuevo Escenario P3 en ArchE.

El mapeado en ArchE quedaría de la siguiente manera: (Figura 118)

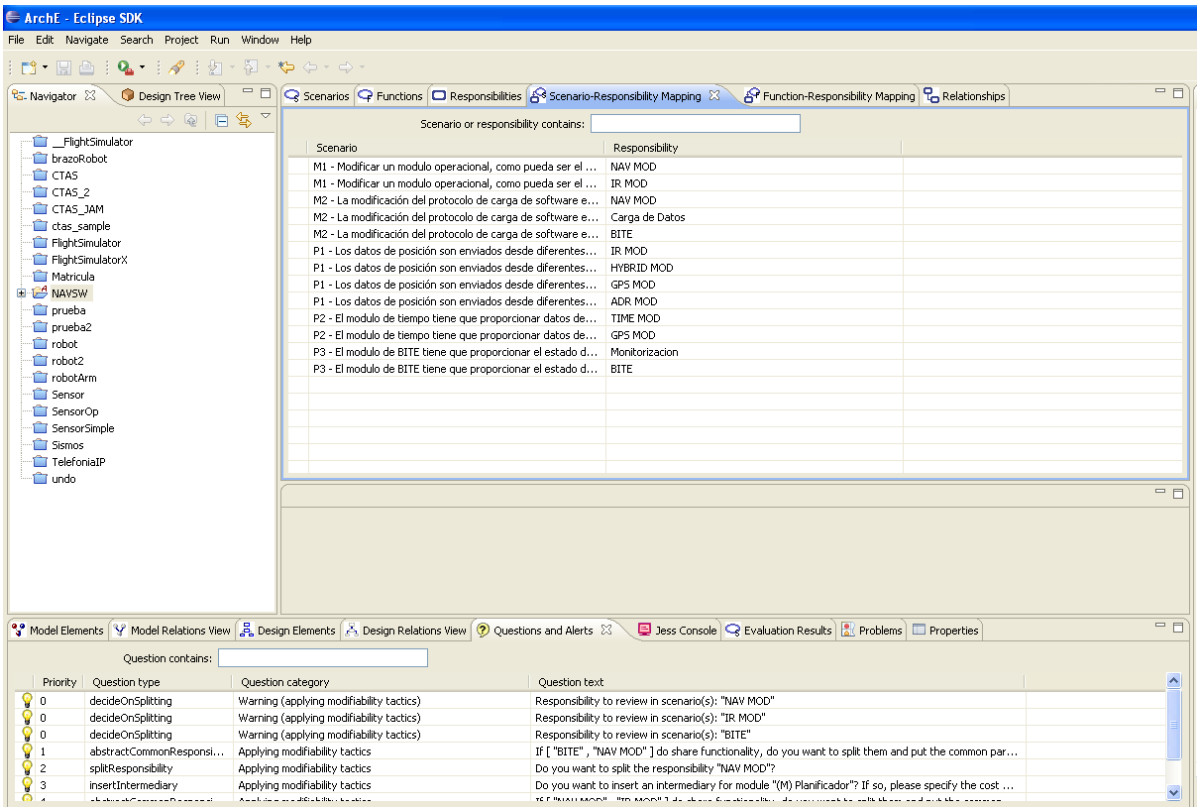


Figura 118. Mapeado Escenarios/Responsabilidades con P3 en Arche.

ArchE en este caso nos indica que de nuevo y debido al nuevo escenario, el sistema deja de ser planificable: (Figura 119 y Figura 121)

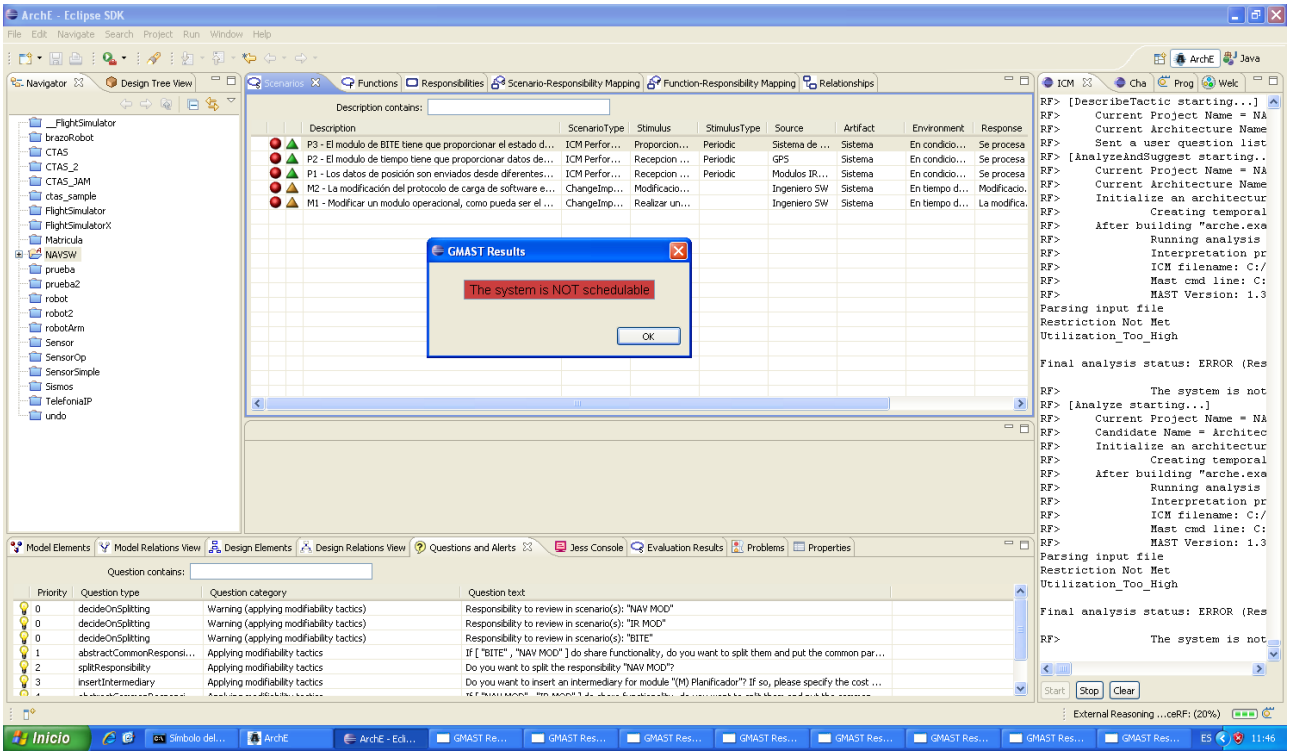


Figura 119. Añadiendo P3 el sistema no es planificable.

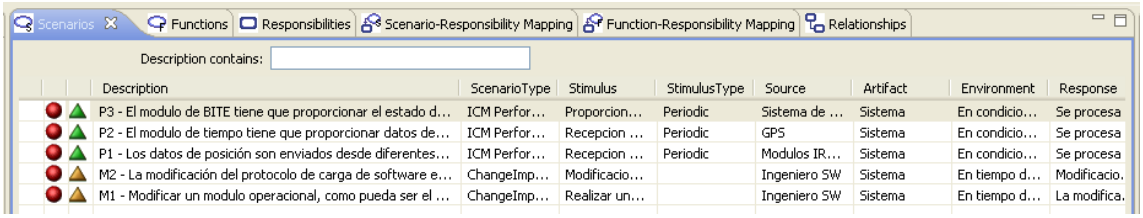


Figura 120. Resultado en Escenarios añadiendo P3 - el sistema no es planificable

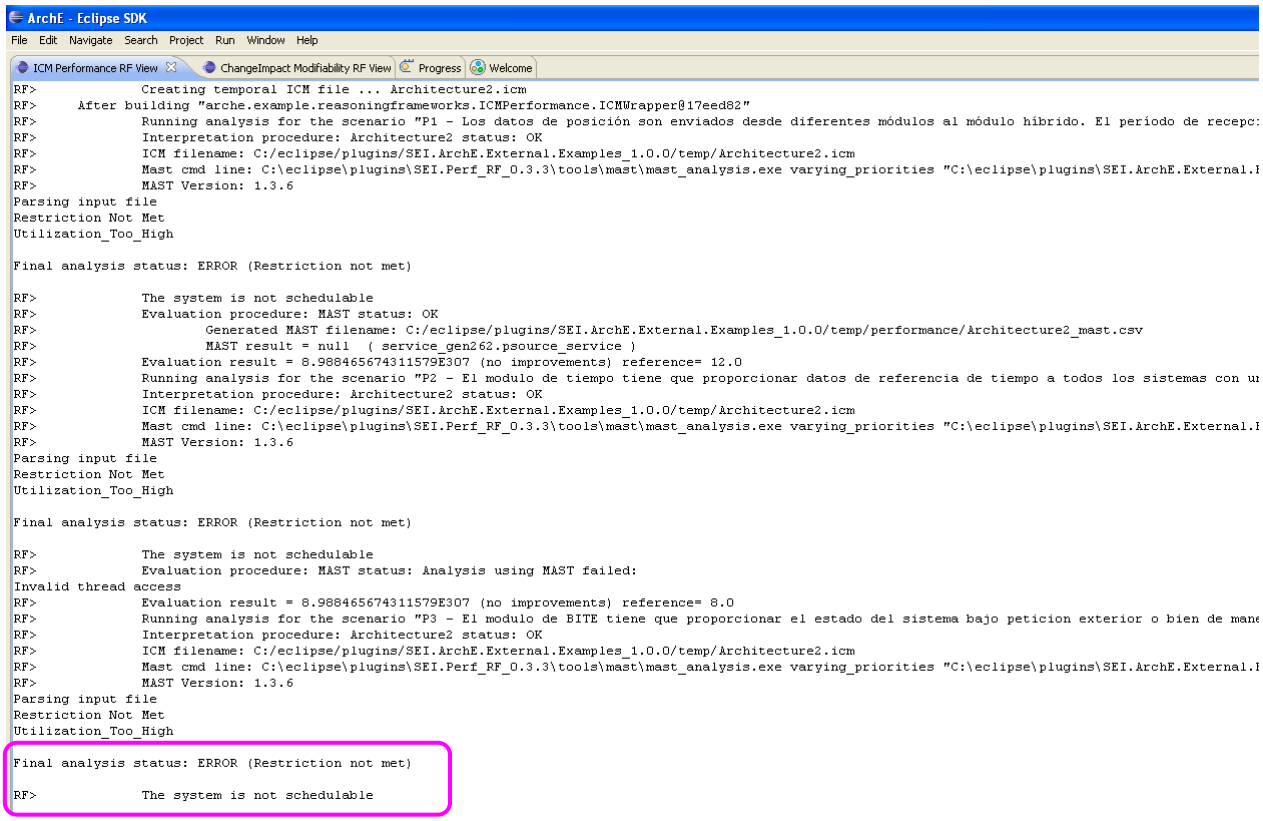


Figura 121. Ventana de ICM Performance. Sistema no planificable.

Como se puede observar, la inclusión del nuevo escenario afecta a la planificabilidad de los otros dos, y ahora las restricciones de cada uno no se cumplen; por tanto, habrá que volver a actuar sobre los parámetros:

Ahora, las funciones de BITE y monitorización han de tener que realizarse en un tiempo de 8 y 5 ms respectivamente. (Figura 122)

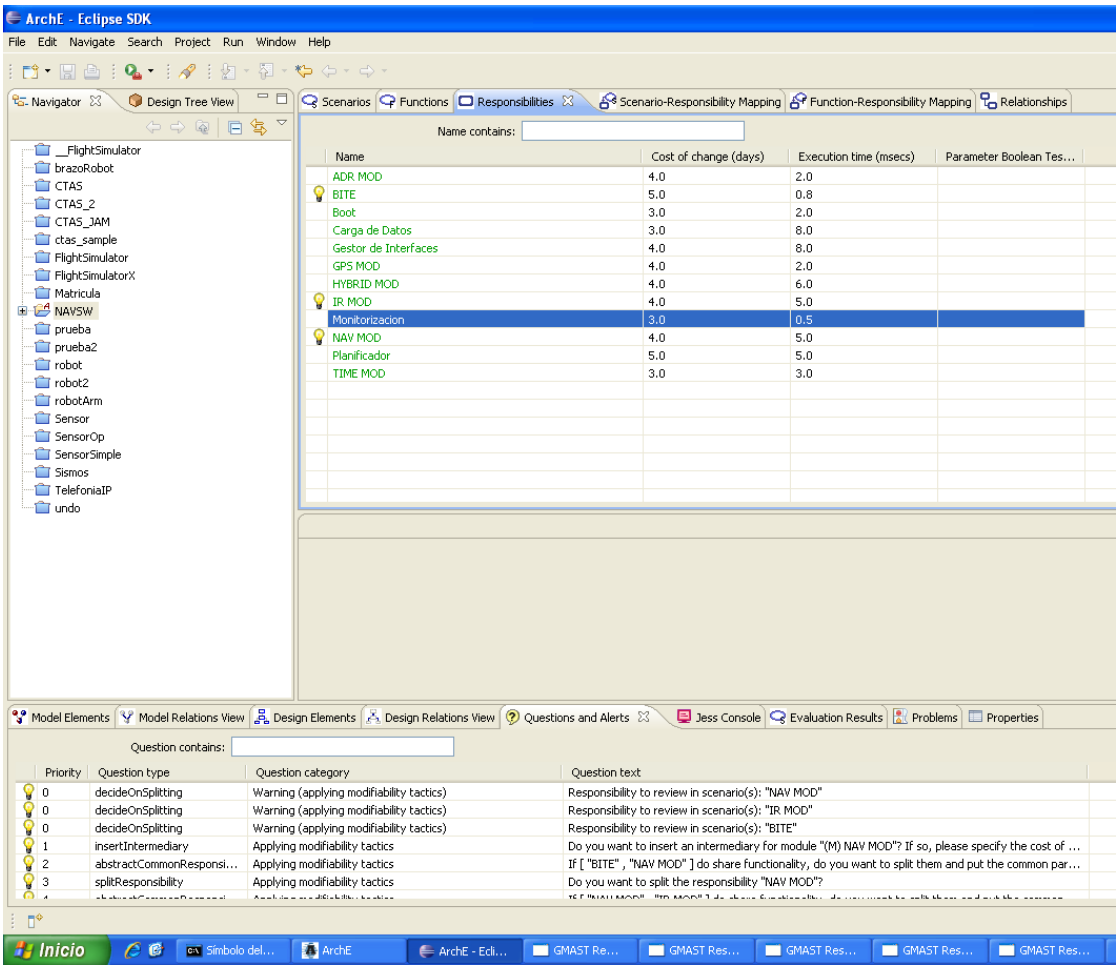


Figura 122. Modificación de tiempo de ejecución.

Y el escenario P3 ha de pasar a unos parámetros de período 500 ms y tiempo de respuesta de 400 ms, lo cual es asumible dado que no es una función crítica para que el sistema desempeñe su fin, aunque sí es muy importante para saber el estado de dicho sistema. Por lo tanto, es asumible dicho “delay”: (Figura 123)

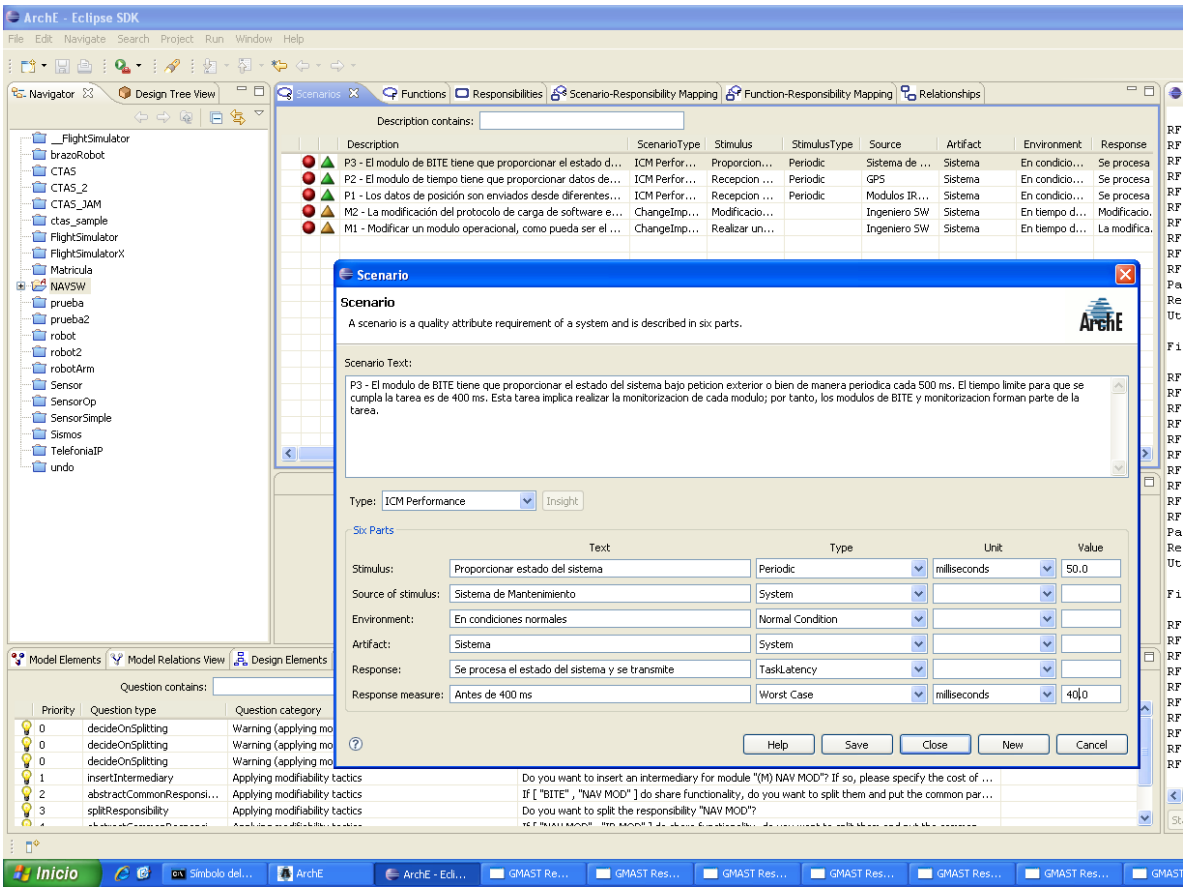


Figura 123. Modificación del Escenario P3.

Con esta modificación, se puede observar que se cumplen algunos escenarios pero no todos; (Figura 124)

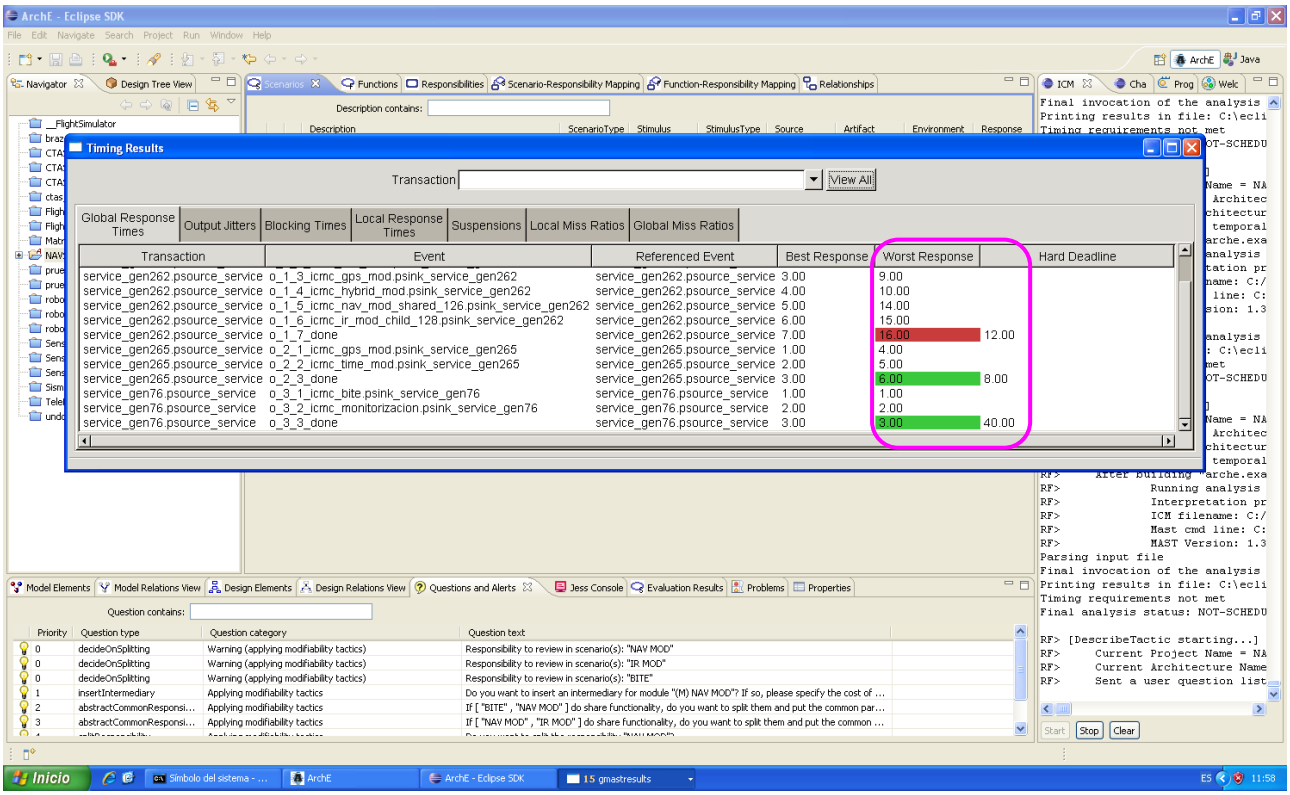


Figura 124. Análisis de Arche con P3 modificado.

El escenario P1 ahora no se cumple: (Figura 125)

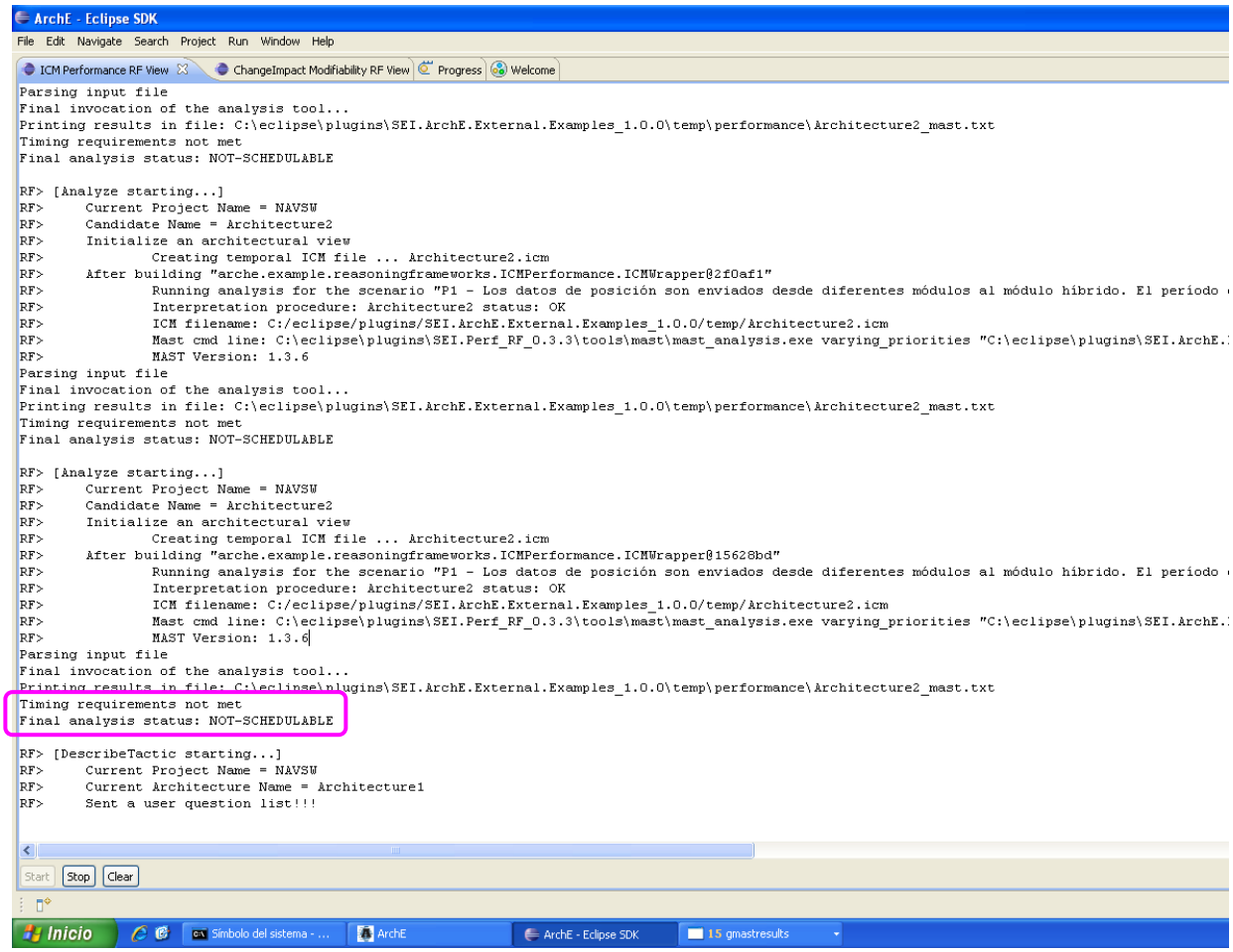


Figura 125. Ventana ICM Performance.

De nada sirve seguir relajando las condiciones del escenario P3, como se puede observar en la Figura 126 y la Figura 127:

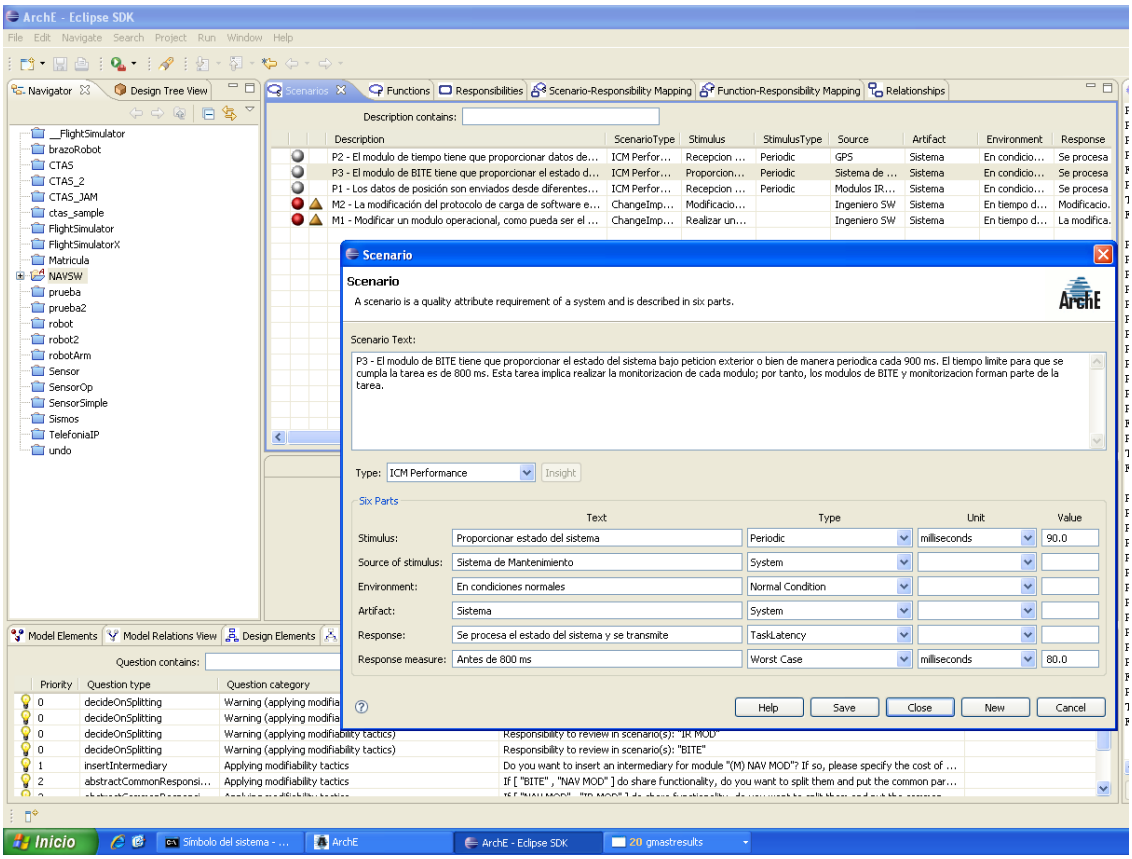


Figura 126. P3 ha alcanzado su límite de relajación de condiciones.

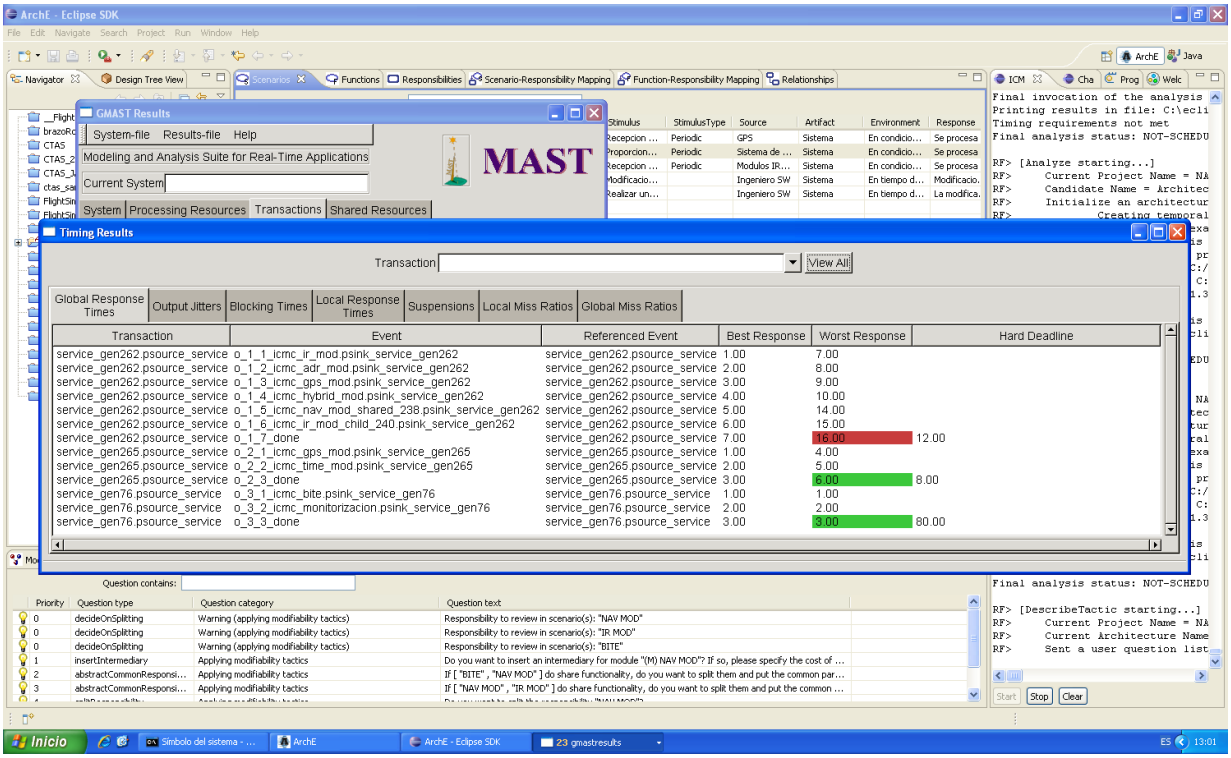


Figura 127. Análisis MAST de P3 cuando alcanza su límite de relajación de condiciones.

La última modificación, por tanto, es una relajación del escenario P1, como se ve en la Figura 128 y en la Figura 129:

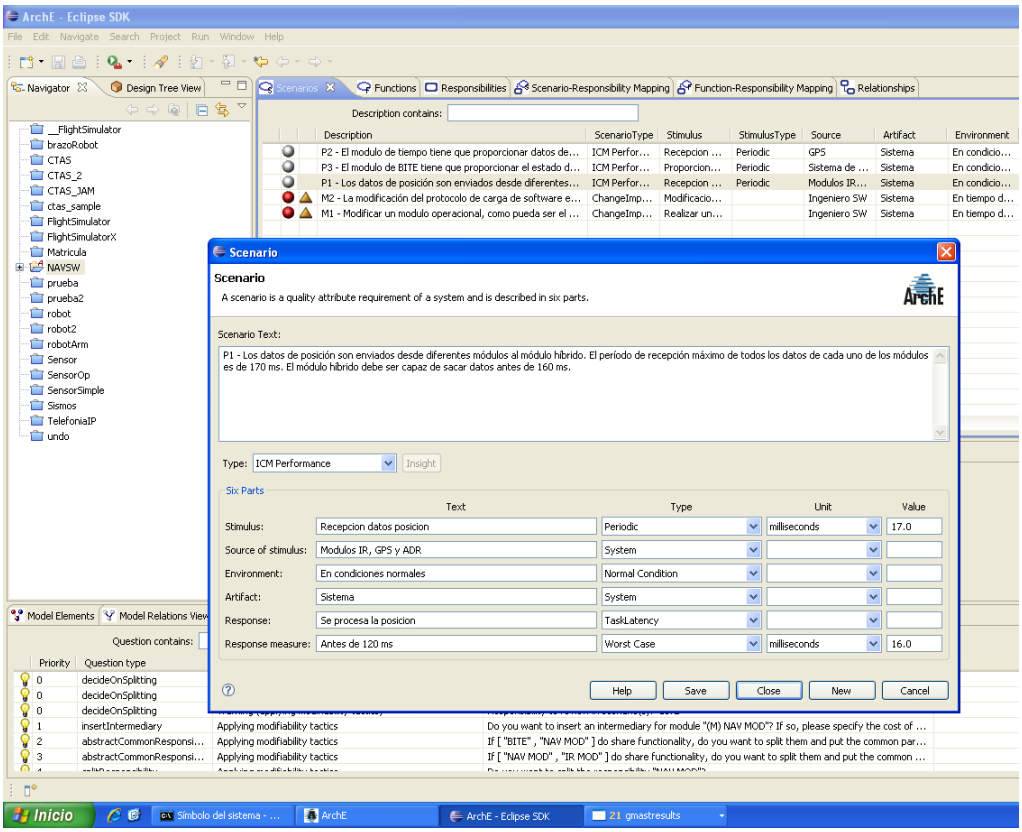


Figura 128. Las condiciones de P1 se relajan.

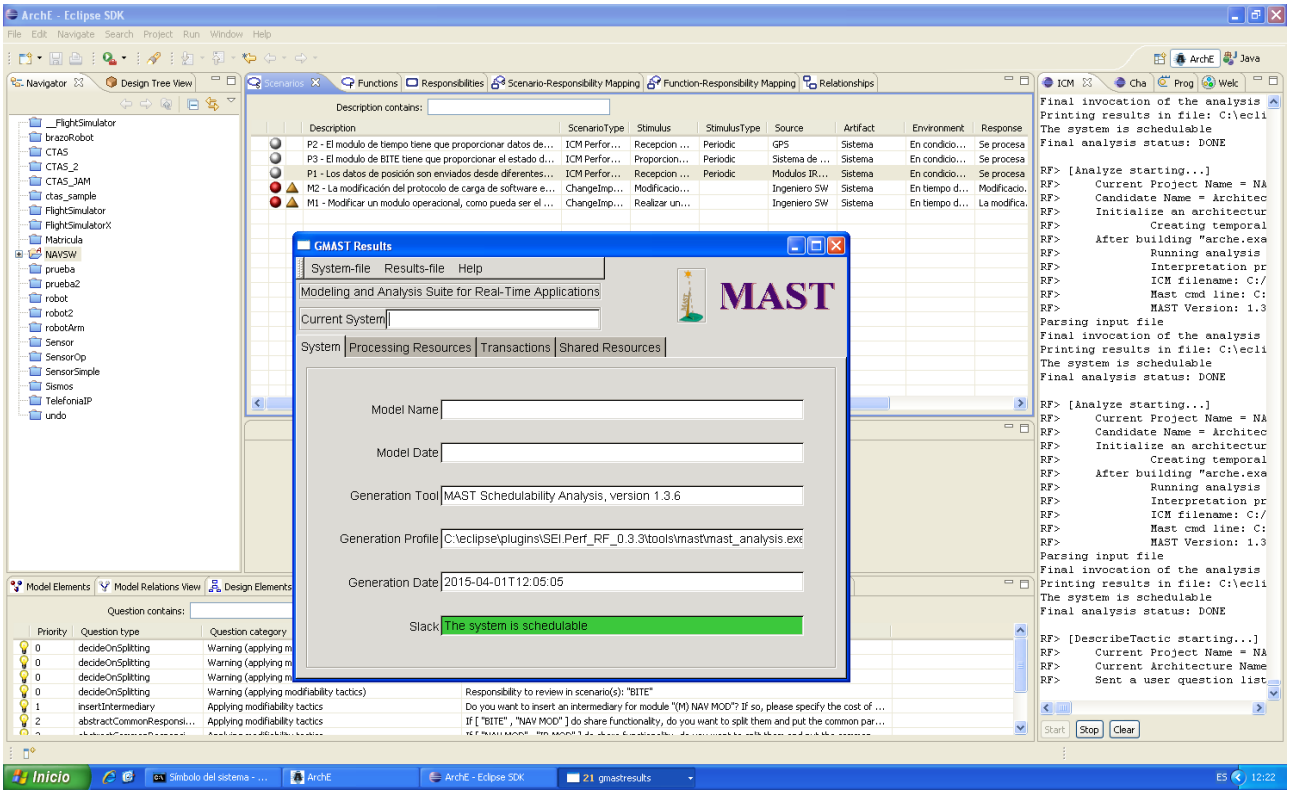


Figura 129. Análisis de MAST para condiciones de P1 relajadas.

Para el caso anterior (deadline de P3 = 400 ms, escalado a 40 ms en ArchE): (Figura 130)

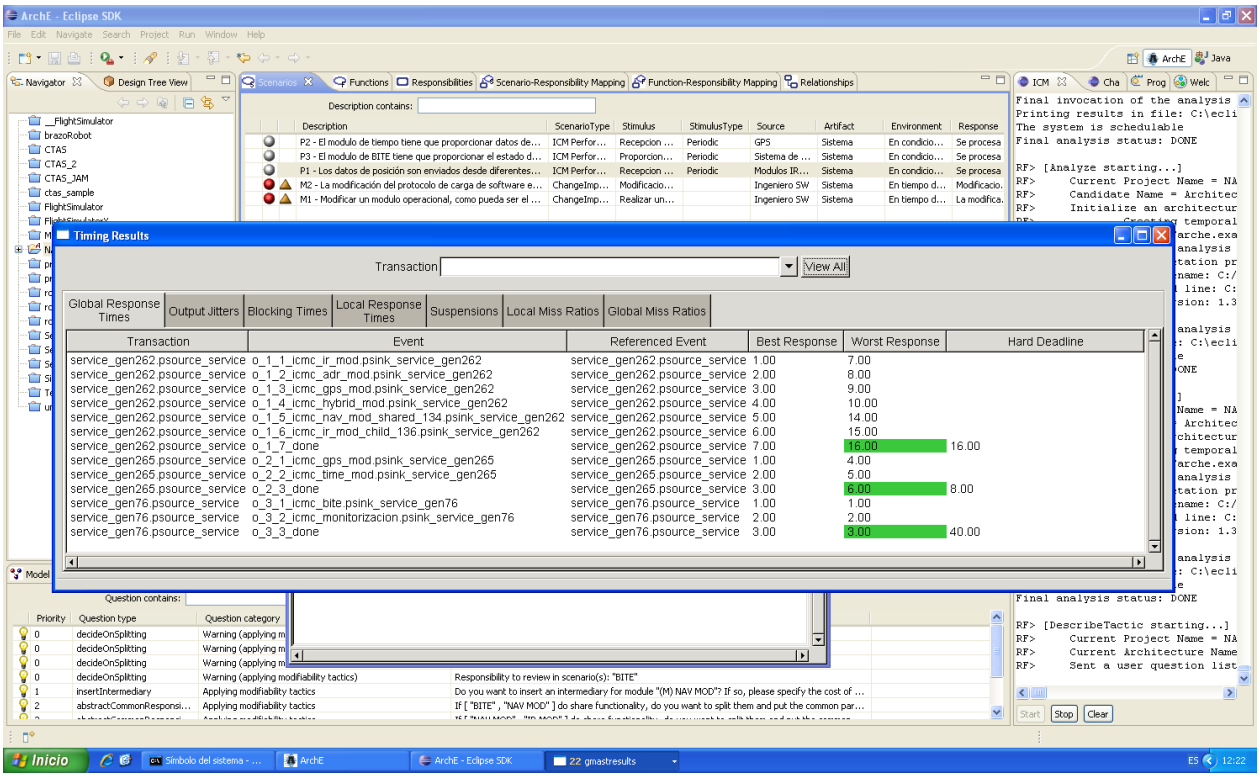


Figura 130. P3 con *deadline* de 400 ms después de relajar P1.

Y para el actual (Figura 131), vemos que no hay ya diferencia, una vez modificado P1:

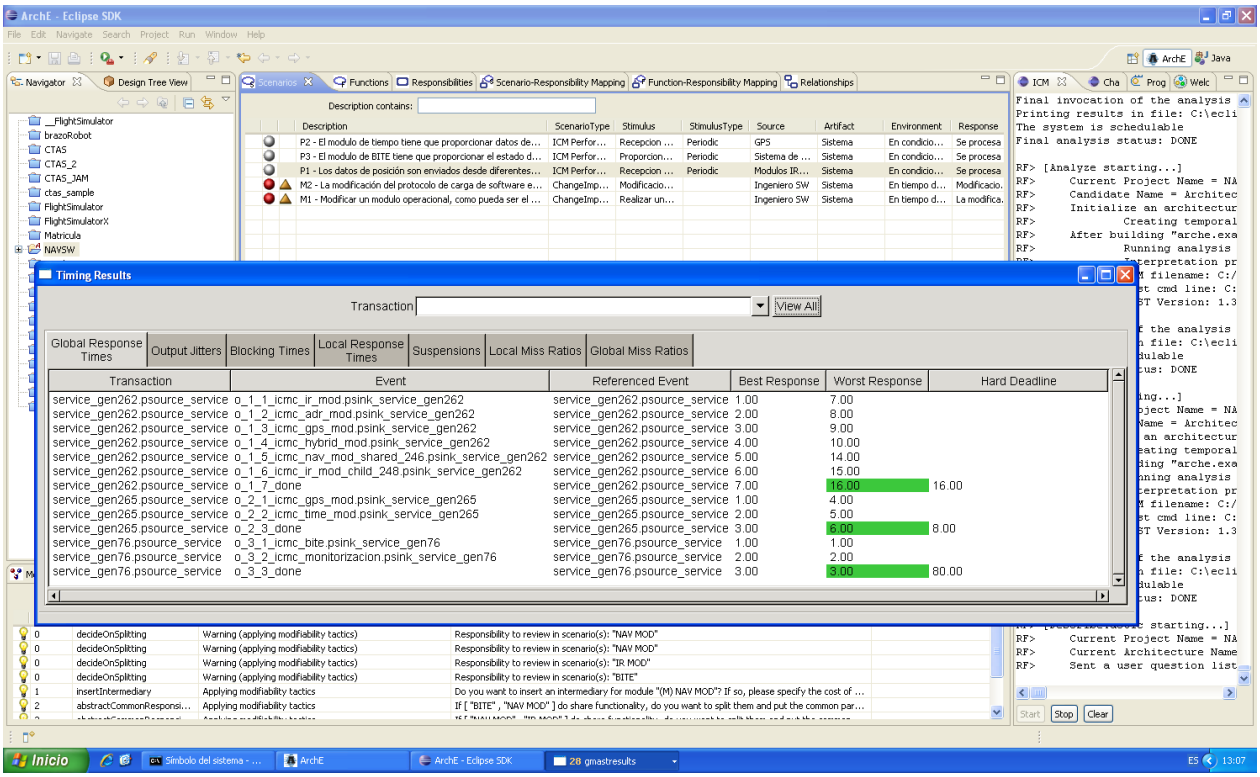


Figura 131. P3 con *deadline* de 800 ms después de relajar P1.

Se observa en la Figura 132 que el sistema es planificable:

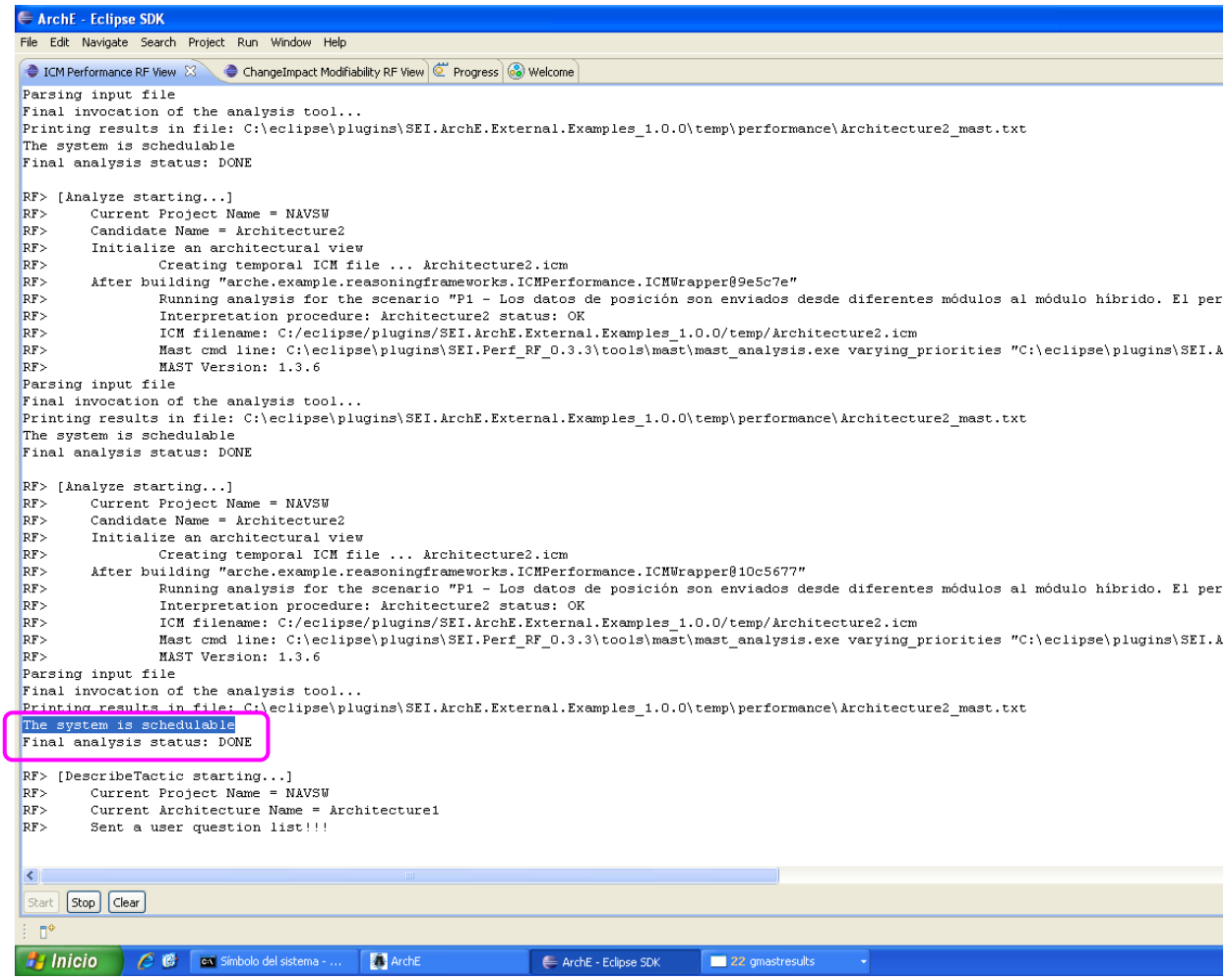


Figura 132. Ventana ICM Performance con P1 y P3 cumpliéndose.

Analizando los resultados, ArchE, y en concreto, MAST, muestran que el verdadero cuello de botella para la planificación es P1. Una vez relajado este escenario, se puede relajar P3. Se ve claramente en las anteriores imágenes el margen de mejora que hay con una pequeña modificación de P1. Modificando P3, también se conseguiría planificar el sistema (Figura 133):

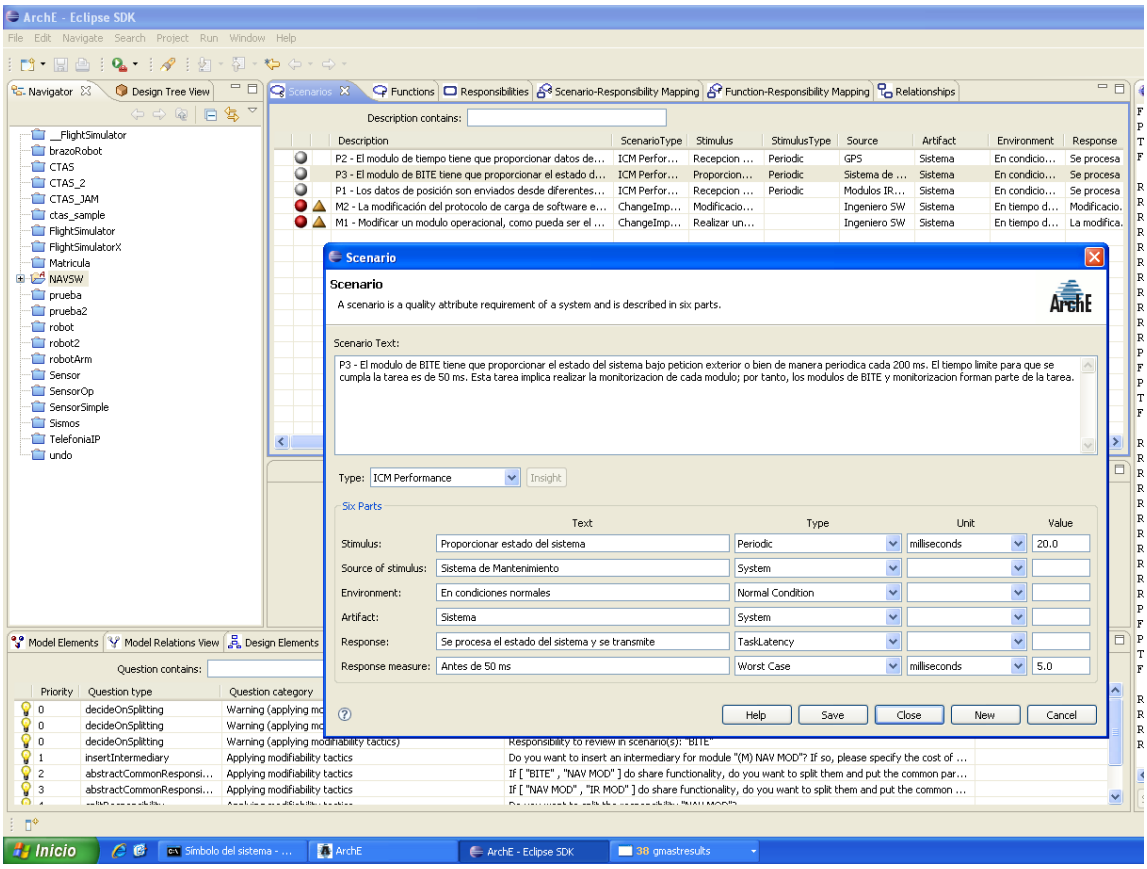


Figura 133. Nueva Modificación de P3 con deadline 50 ms.

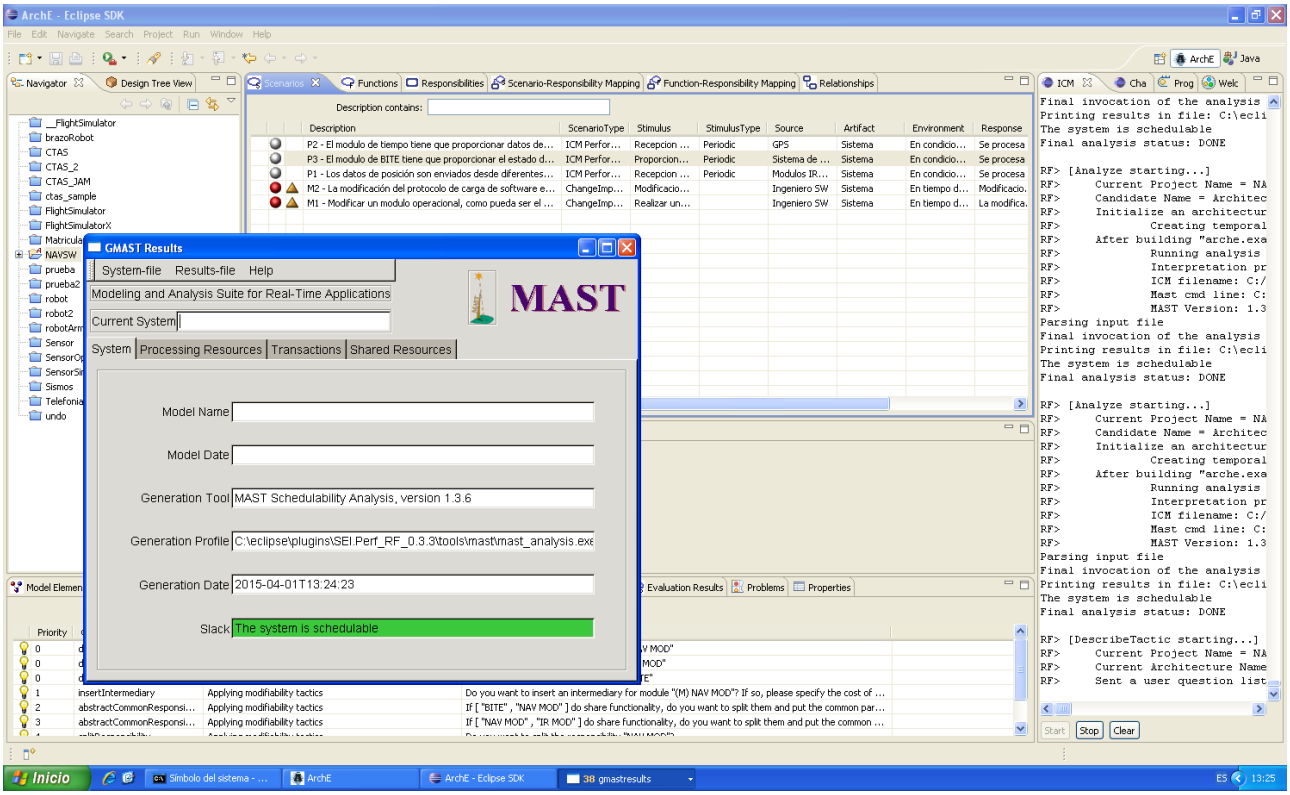


Figura 134. Análisis de MAST: Sistema planificable con *deadline* de P3 en 50 ms.

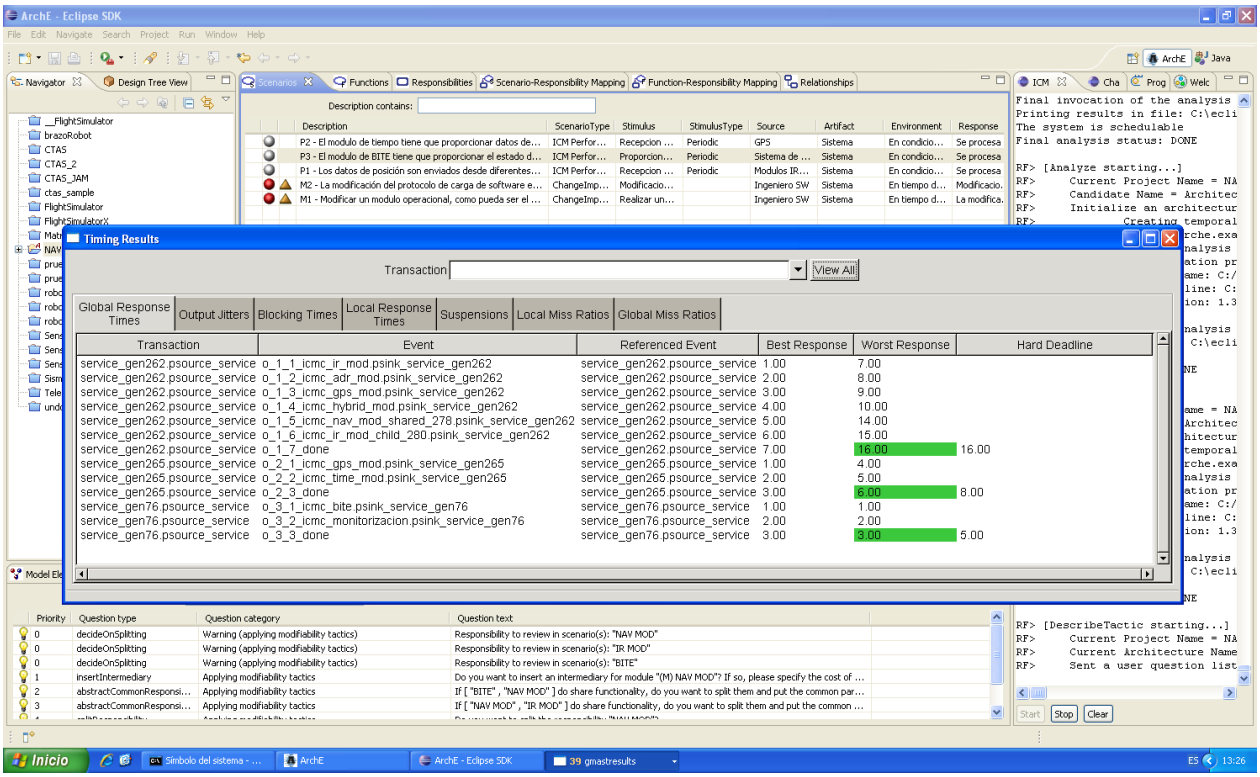


Figura 135. Análisis de MAST detallado: Sistema planificable con *deadline* de P3 en 50 ms.

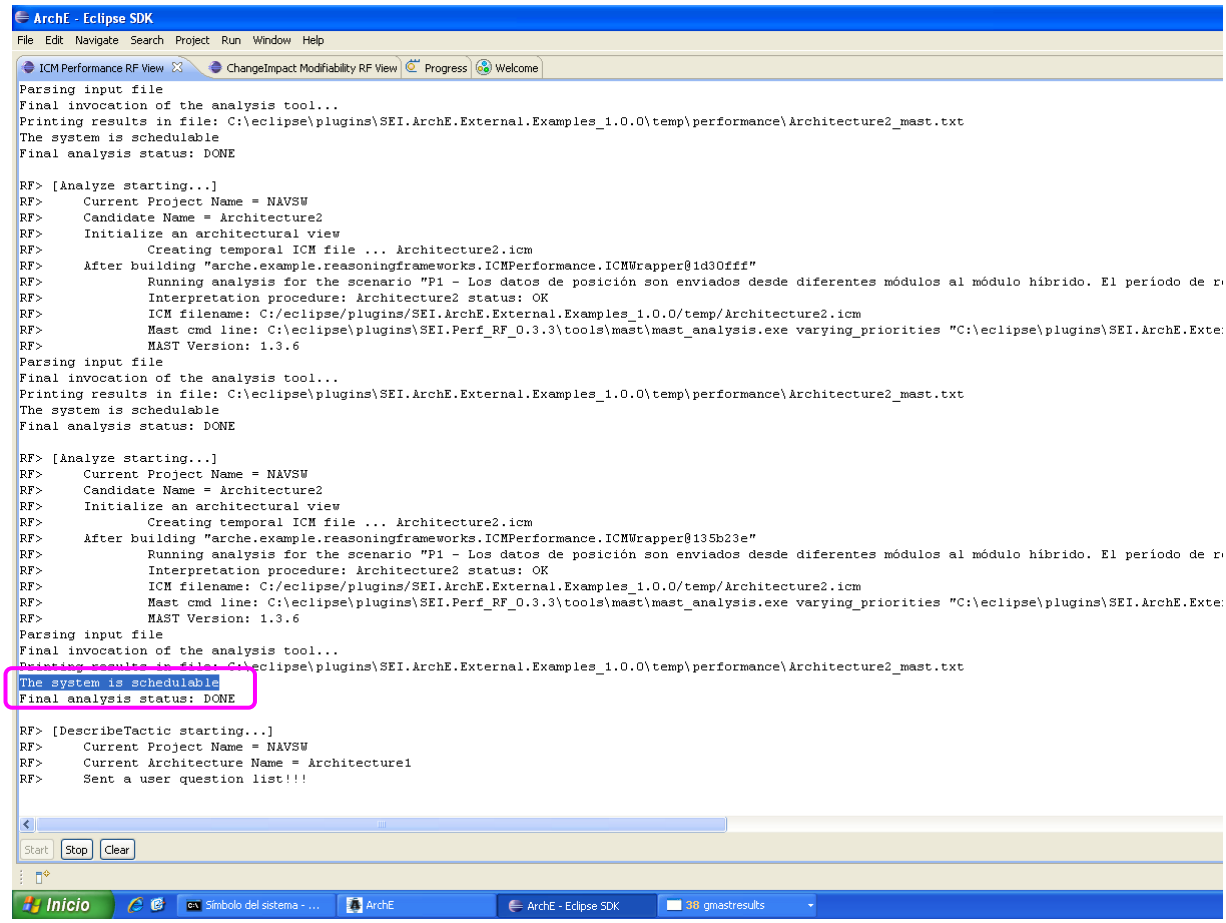


Figura 136. Ventana ICM Performance con P1, P2 y P3 cumpliéndose.

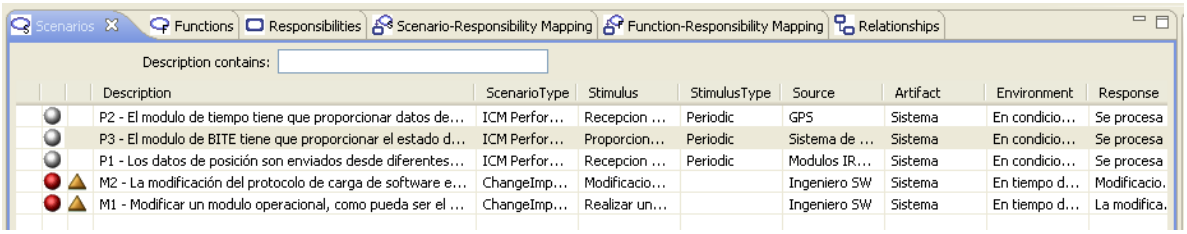


Figura 137. Escenarios de Performance Planificables.

En la Tabla 40 se muestra un resumen de las diferentes iteraciones que se han realizado. El color amarillo de las celdas muestra los parámetros que han sido modificados respecto de la iteración anterior:

Responsibility	Time (ms)	it0	Status	it1	Status	it2	Status	it3	Status	it4	Status	it5	Status	it5	Status
Boot	20	20		20		20		20		20		20		20	
Planificador	50	50		50		50		50		50		50		50	
Carga de Datos	80	80		80		80		80		80		80		80	
Gestor de interfaces	80	80		80		80		80		80		80		80	
BITE	100	100		100		100		8		8		8		8	
Monitorización	70	70		70		70		5		5		5		5	
NAV MOD	50	50		50		50		50		50		50		50	
IR MOD	80	80		50		50		50		50		50		50	
ADR MOD	80	80		20		20		20		20		20		20	
GPS MOD	80	80		20		20		20		20		20		20	
HYBRID MOD	90	90		60		60		60		60		60		60	
TIME MOD	50	50		30		30		30		30		30		30	
P1 - Period	100	100		150		150		150		150		170		170	
P1 - Response Measure	90	90	NOK	120	OK	120	NOK	120	NOK	120	NOK	160	OK	160	OK
P2 - Period	80	80		100		100		100		100		100		100	
P2 - Response Measure	60	60	NOK	80	OK	80	NOK	80	OK	80	OK	80	OK	80	OK
P3 - Period						80		500		900		900		200	
P3 - Response Measure						60	NOK	400	OK	800	OK	800	OK	50	OK

Tabla 40. Iteraciones del Sistema con ICM Performance.

Respecto a los escenarios de modificabilidad, ArchE propone un diseño de la arquitectura basada en las relaciones de dependencia entre responsabilidades de la siguiente manera: (Figura 138)

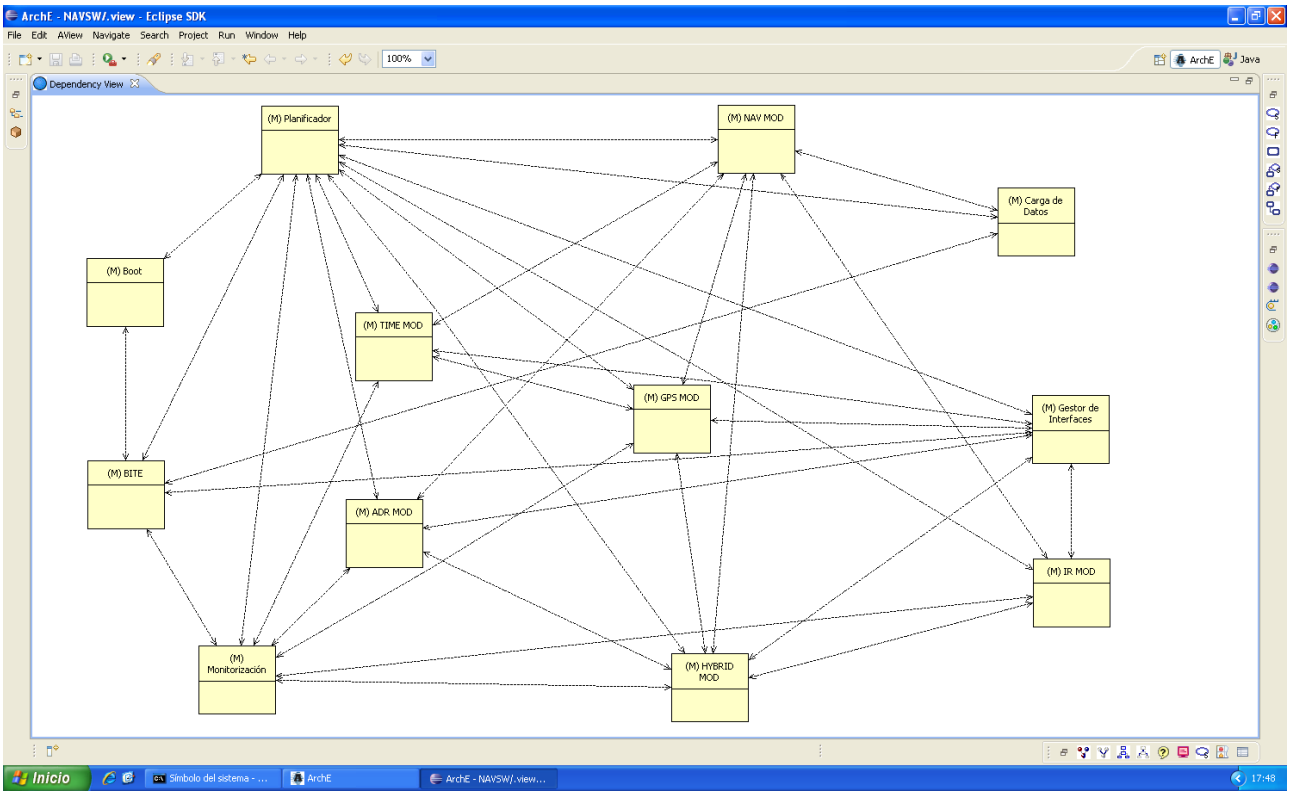


Figura 138. Vista de Dependencias ArchE - Arquitectura Software del Sistema de Aviónica.

Del análisis del marco ChangeImpact Modifiability, se puede observar que el escenario M1 está cerca de poderse cumplir, mientras que el M2 queda bastante lejos. Esto se debe, sin duda, al fuerte acoplamiento que hay entre módulos (Figura 139):

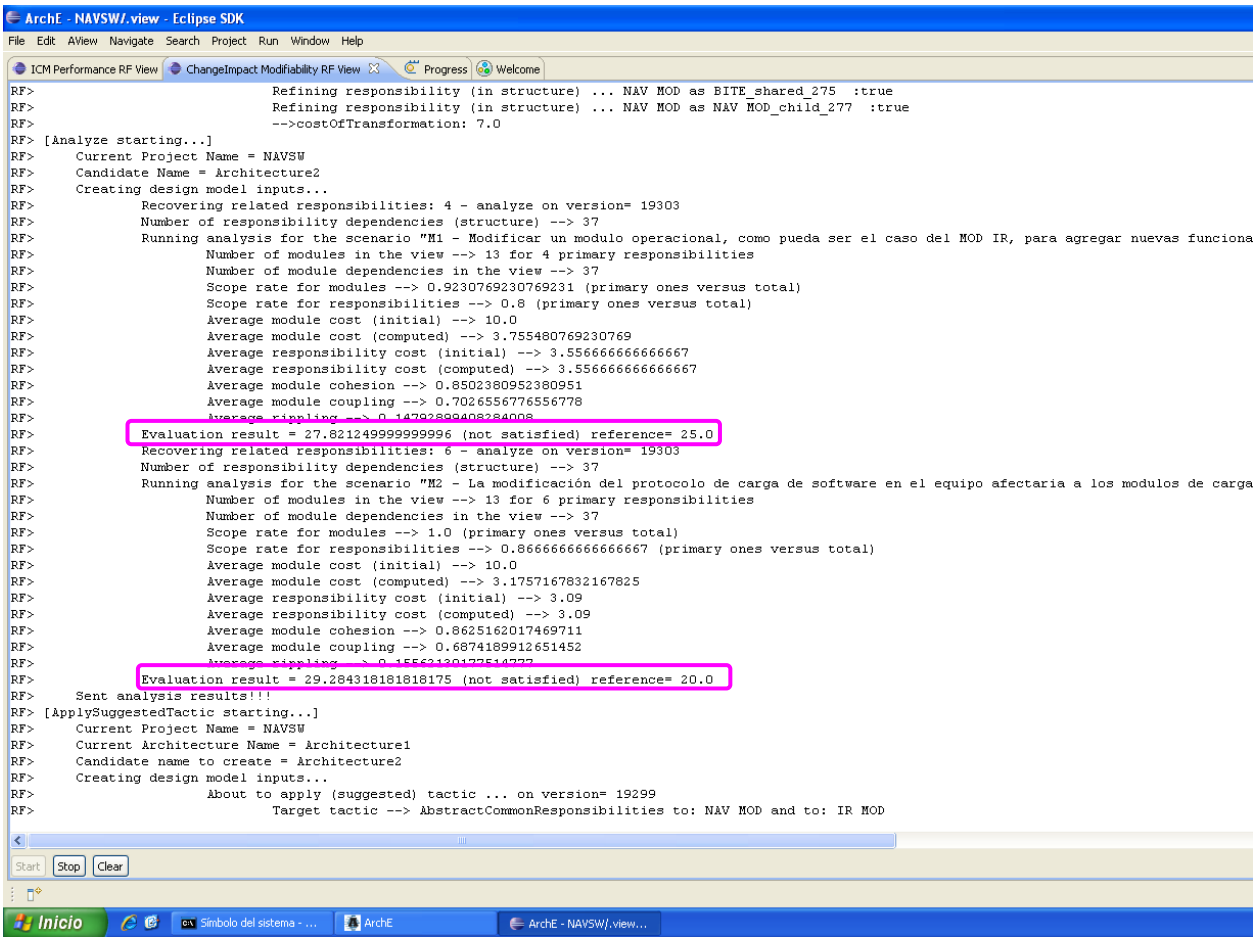


Figura 139. Marco ChangeImpact Modifiability con M1 y M2 sin cumplirse.

De la pestaña “Evaluation Results” (Figura 140 y Figura 141) se puede observar que, de todas las tácticas propuestas por ArchE, la táctica 1 y la 3 son las que ofrecen una gran mejora en el escenario M1.

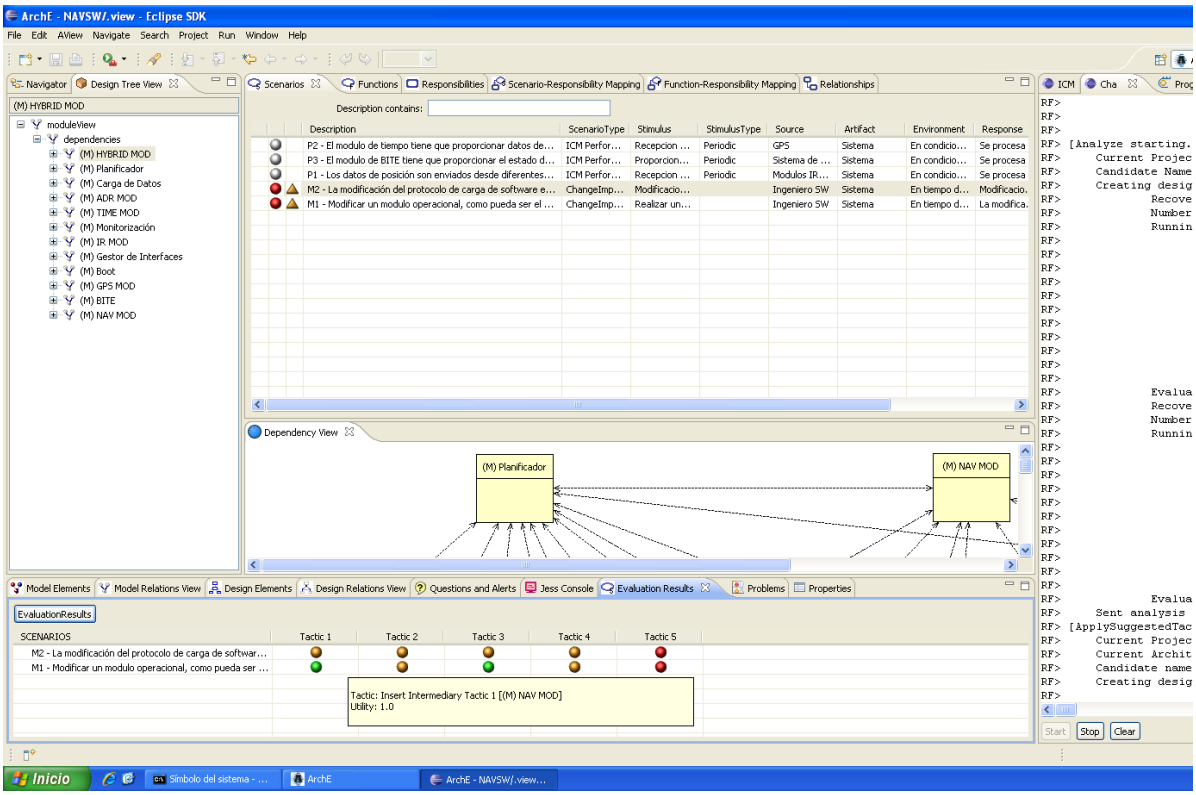


Figura 140. Evaluation Results para M1 y M2.

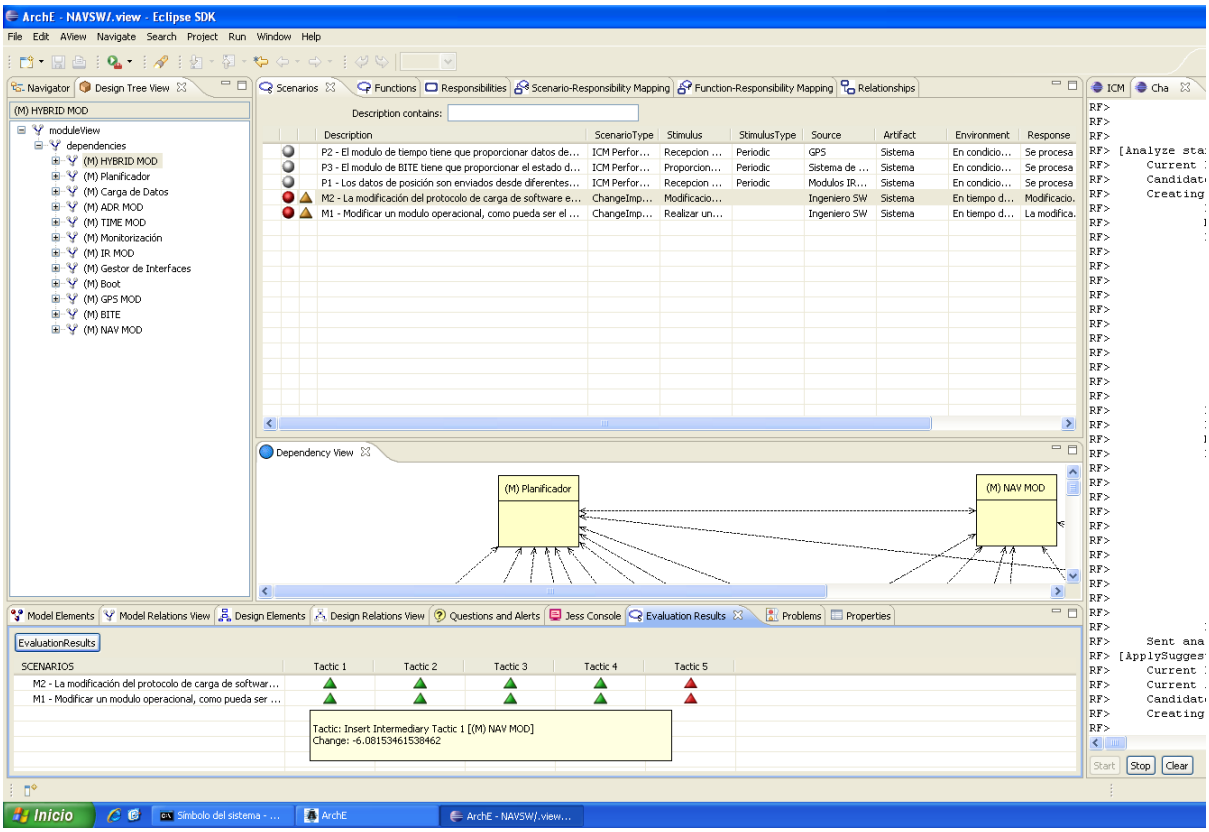


Figura 141. Evaluation Results para M1 y M2.

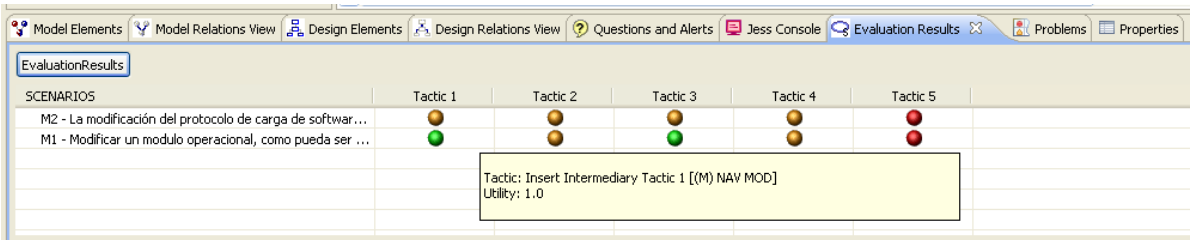


Figura 142. Resultados de la Evaluación para M1 y M2 (ampliado).

ArchE ofrece una serie de tácticas para mejorar los escenarios M1 y M2 (Figura 143):

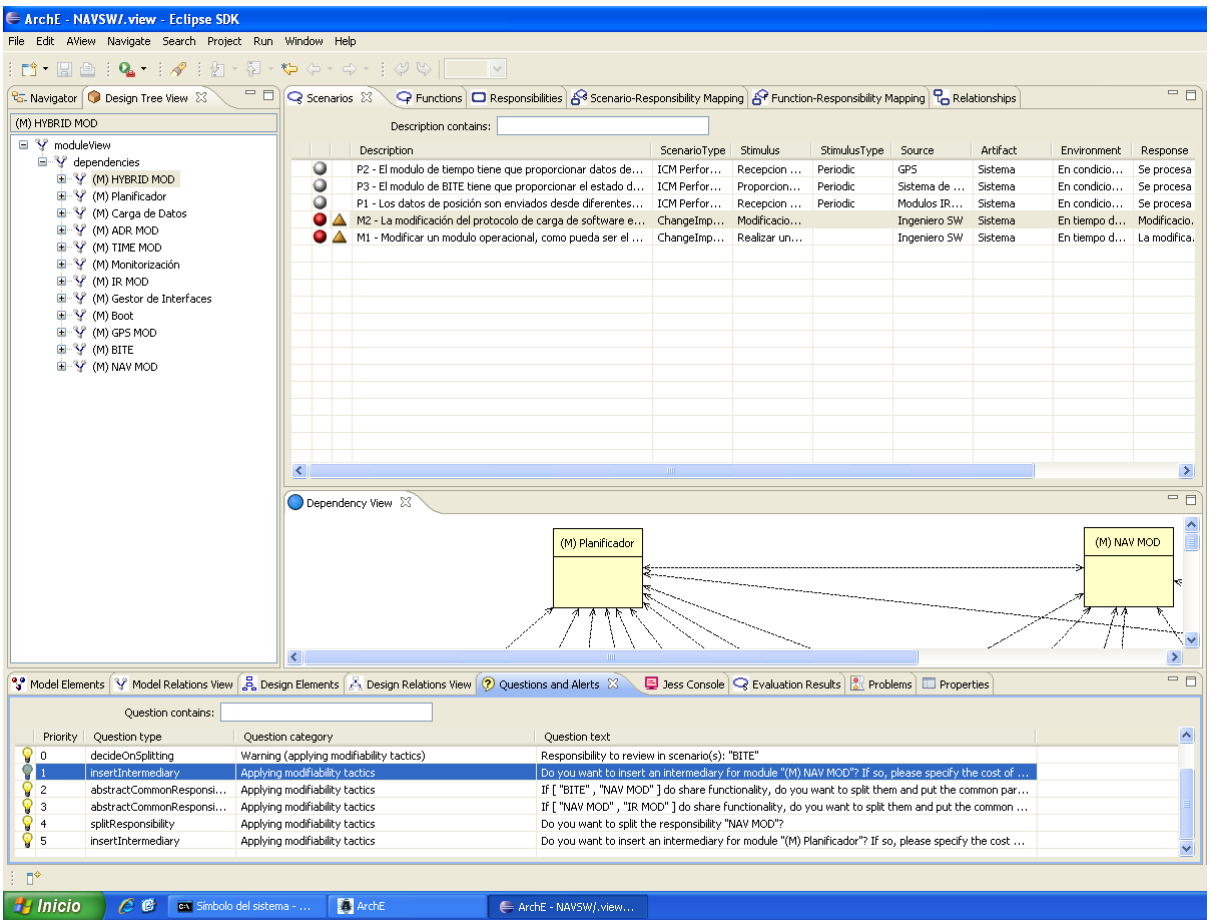


Figura 143. Tácticas de Mejora propuestas por AchE para M1 y M2.

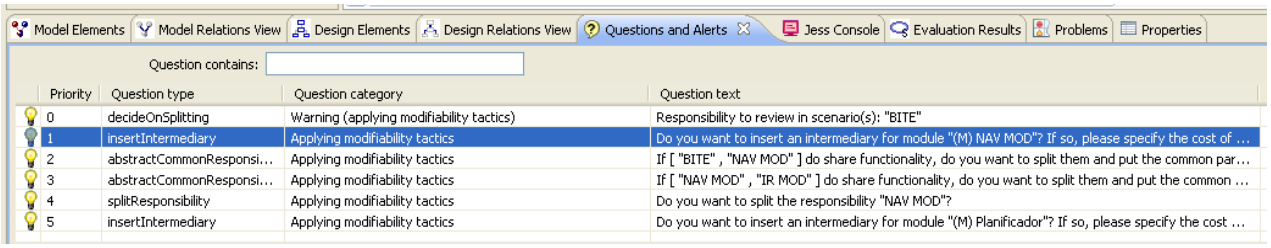


Figura 144. Tácticas de Mejora propuestas por AchE para M1 y M2 (ampliado)

Analizando las diferentes tácticas, puede deducirse que la táctica 1 tiene sentido, pues propone introducir un intermediario para el módulo NAV MOD, lo cual se traduciría en una reducción del acoplamiento entre responsabilidades.

La táctica 2 no tiene demasiado sentido, pues los módulos BITE y NAV MOD no tienen demasiado en común. No hay que olvidar que ArchE no conoce nada respecto a la funcionalidad del sistema, de ahí que haya que interpretar todas sus sugerencias y ver si tienen sentido en el contexto en el que el sistema se desenvuelve.

La táctica 3 podría tener sentido, pues los módulos NAV MOD e IR MOD sí que comparten funcionalidad, pero no tendría sentido seguir esta sugerencia, pues precisamente NAV MOD ha de compartir funcionalidad con todos los módulos funcionales, de ahí el sentido de su existencia.

La táctica 4 no produciría una gran mejora, por lo tanto no se tiene en cuenta. Y la táctica 5 no produciría ninguna mejora, por lo tanto se descarta.

Se aplica la táctica 1 y se analiza lo que ocurre: (Figura 145)

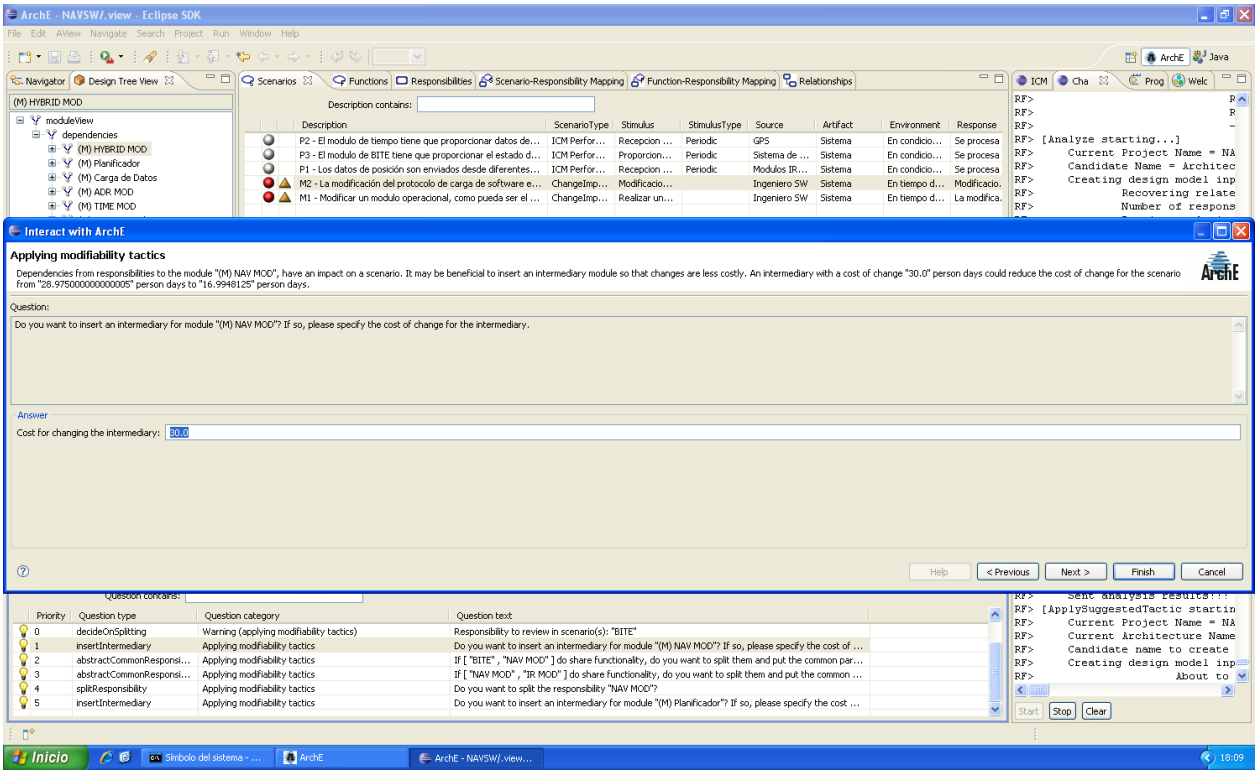


Figura 145. Aplicación de la Táctica 1 a la Arquitectura.

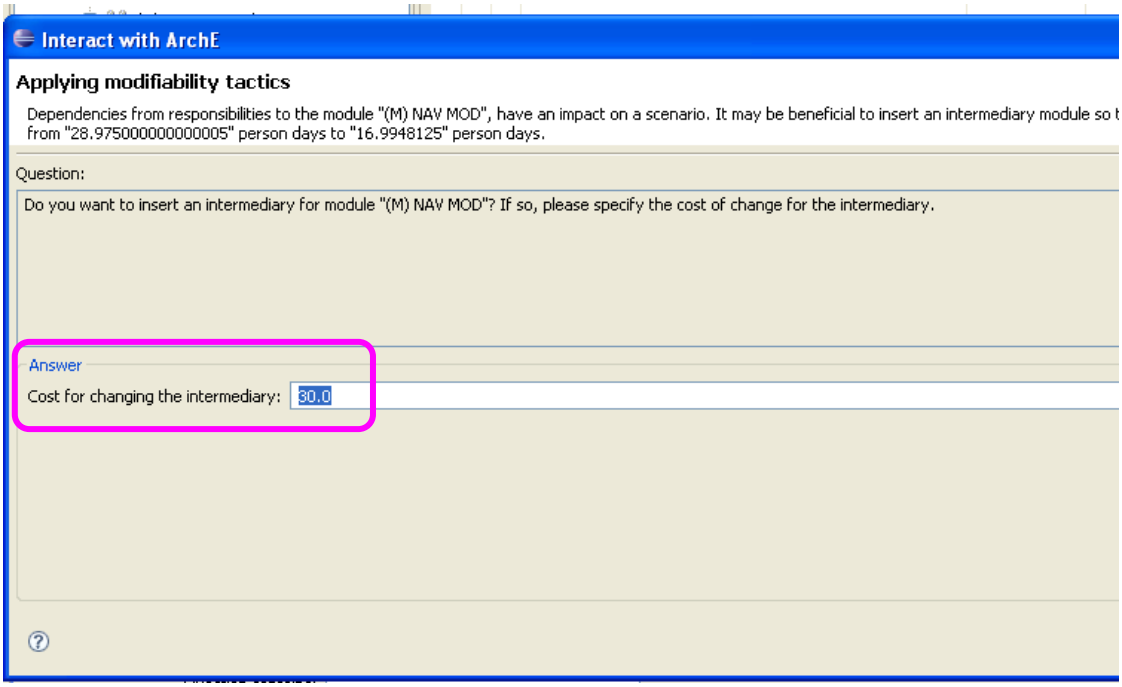


Figura 146. Aplicación de la Táctica 1 a la Arquitectura (ampliada).

Como se puede observar en la Figura 147, después de aplicar esta táctica, el escenario M1 se cumple. Además, se siguen cumpliendo los escenarios de performance

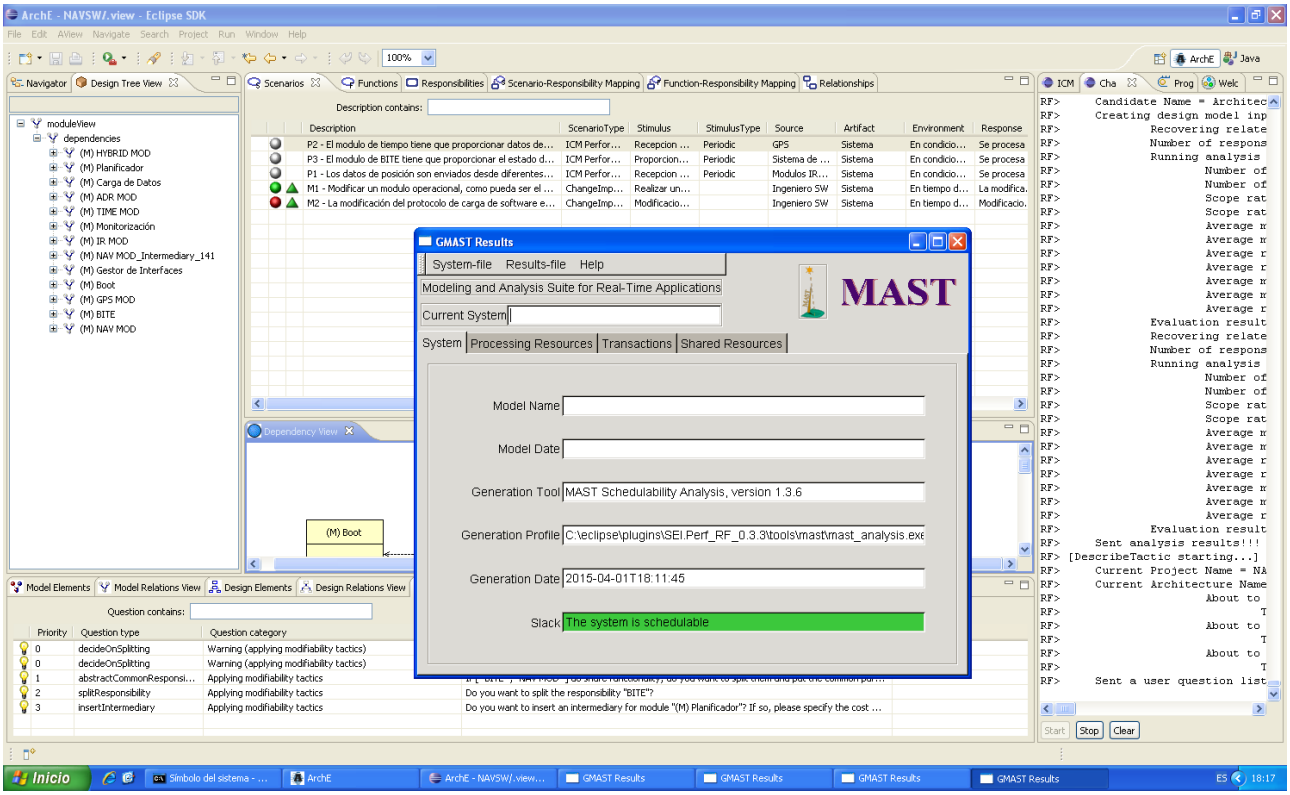


Figura 147. Influencia en la Táctica 1 sobre los Escenarios basados en ICM Performance.

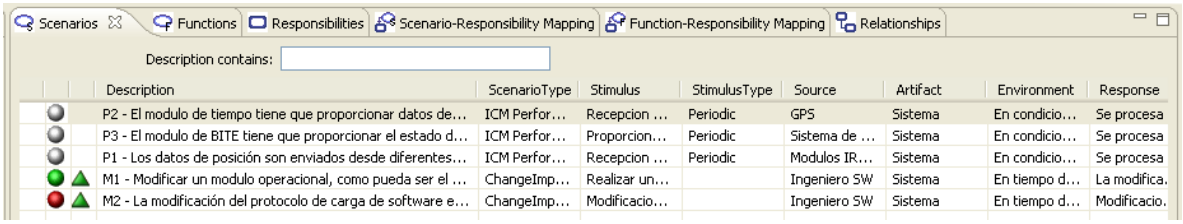


Figura 148. M1 se cumple después de aplicar la táctica 1.

La introducción de un nuevo módulo intermediario no ha provocado que se deje de cumplir la planificación global del sistema, aunque ahora se esté más cerca de las deadlines: (Figura 149)

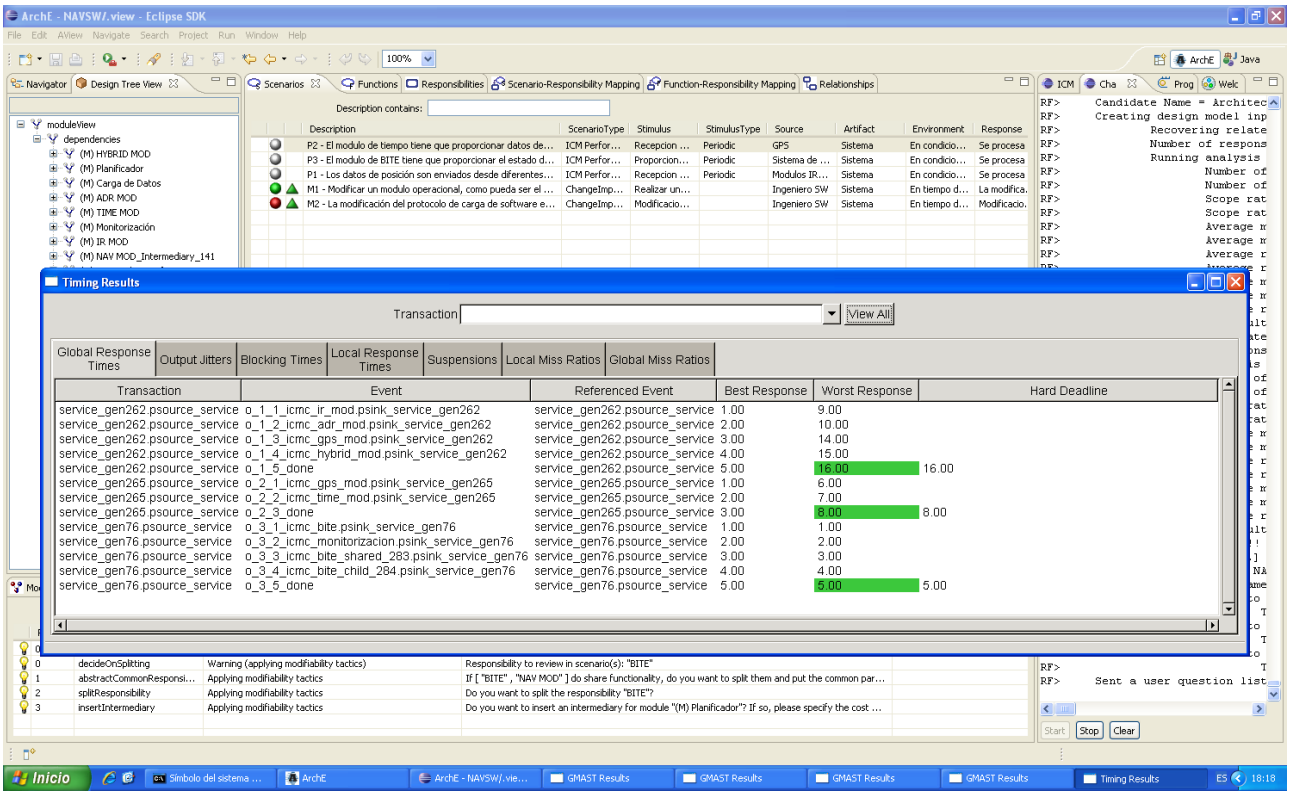


Figura 149. Análisis detallado MAST de los escenarios basados en ICM Performance.

Se puede observar cómo se cumple el M1 y el M2 mejora considerablemente, quedándose muy cerca de cumplirse también (Figura 150):

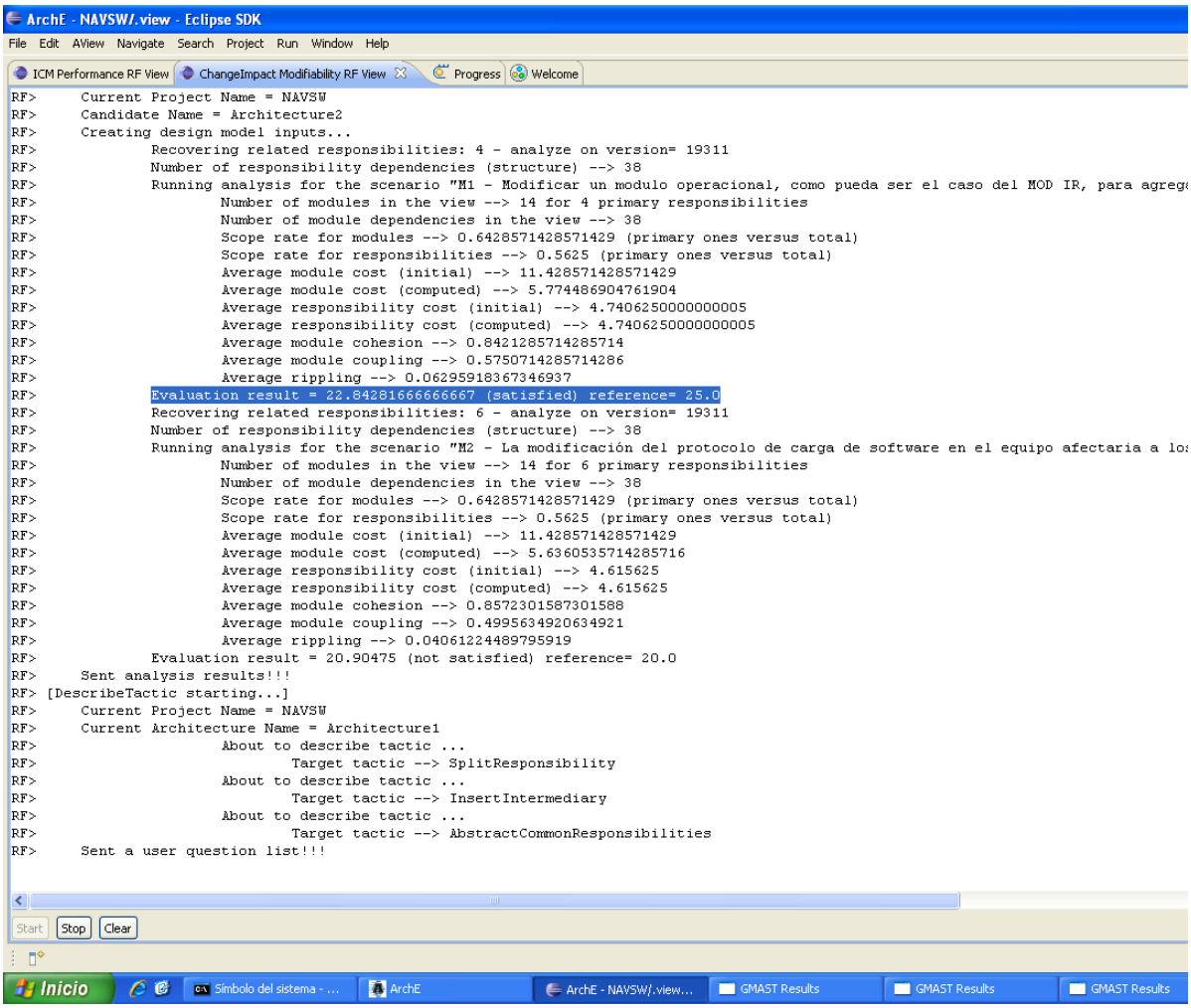


Figura 150. Ventana del Marco de Modificabilidad tras aplicar la Táctica 1.

La arquitectura propuesta por ArchE quedaría de la siguiente manera (Figura 151):

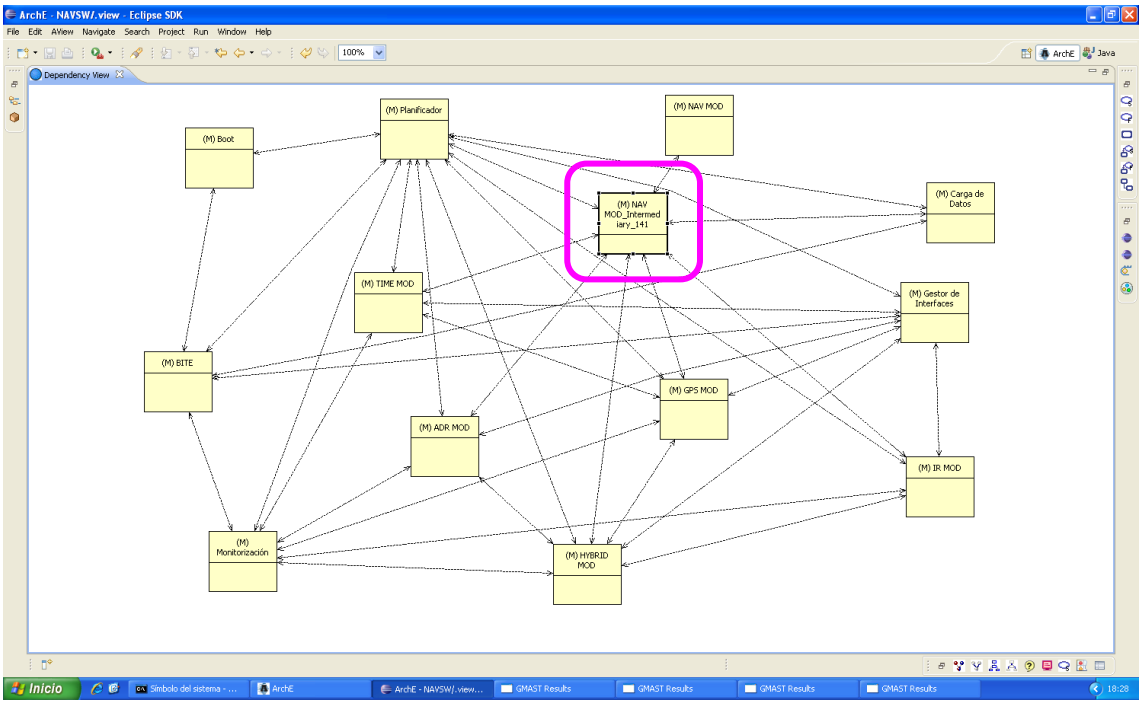


Figura 151. Nueva Vista de Dependencia tras aplicar la Táctica 1.

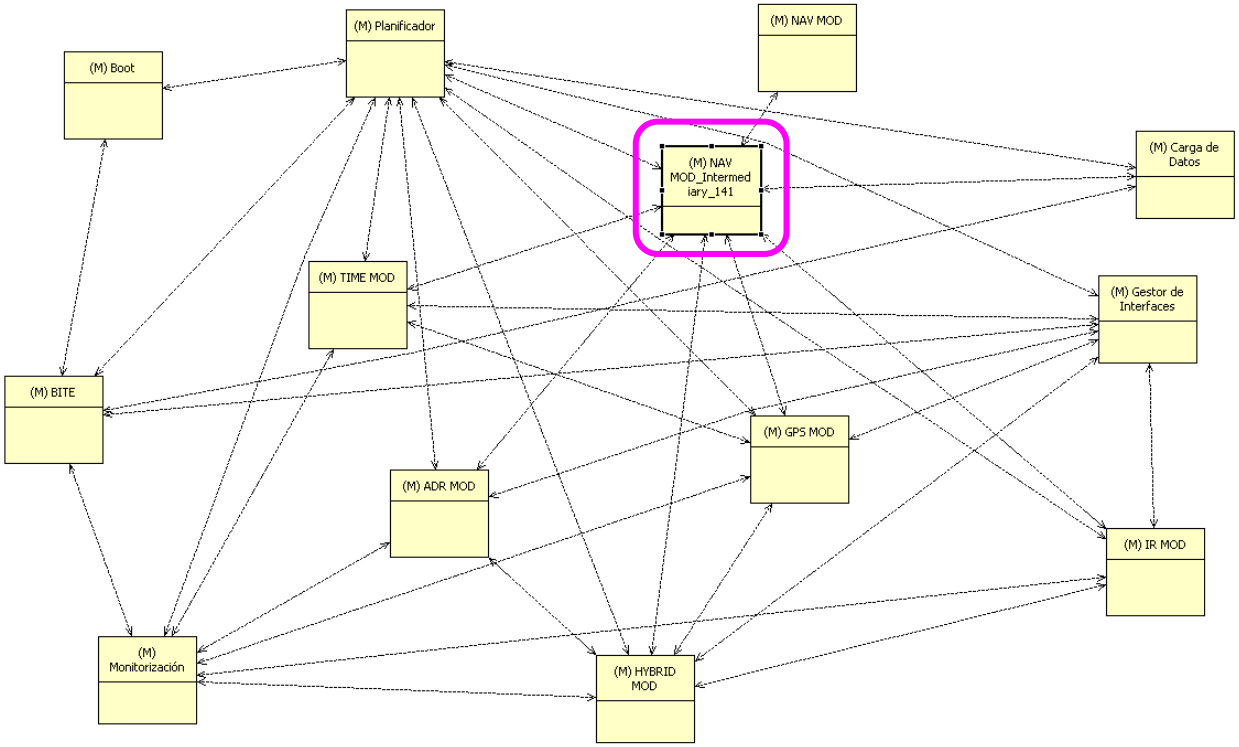


Figura 152. Nueva Vista de Dependencia tras aplicar la Táctica 1 (ampliada).

De esta manera, se podría separar la parte del módulo de NAV MOD que se ha de modificar siempre que se toque otro módulo funcional (NAV MOD_intermediary_141) de la parte que no es necesario modificar.

Las tácticas propuestas ahora por ArchE seguirían mejorando M1 pero no M2, por lo que hay que buscar otro tipo de soluciones (Figura 153):

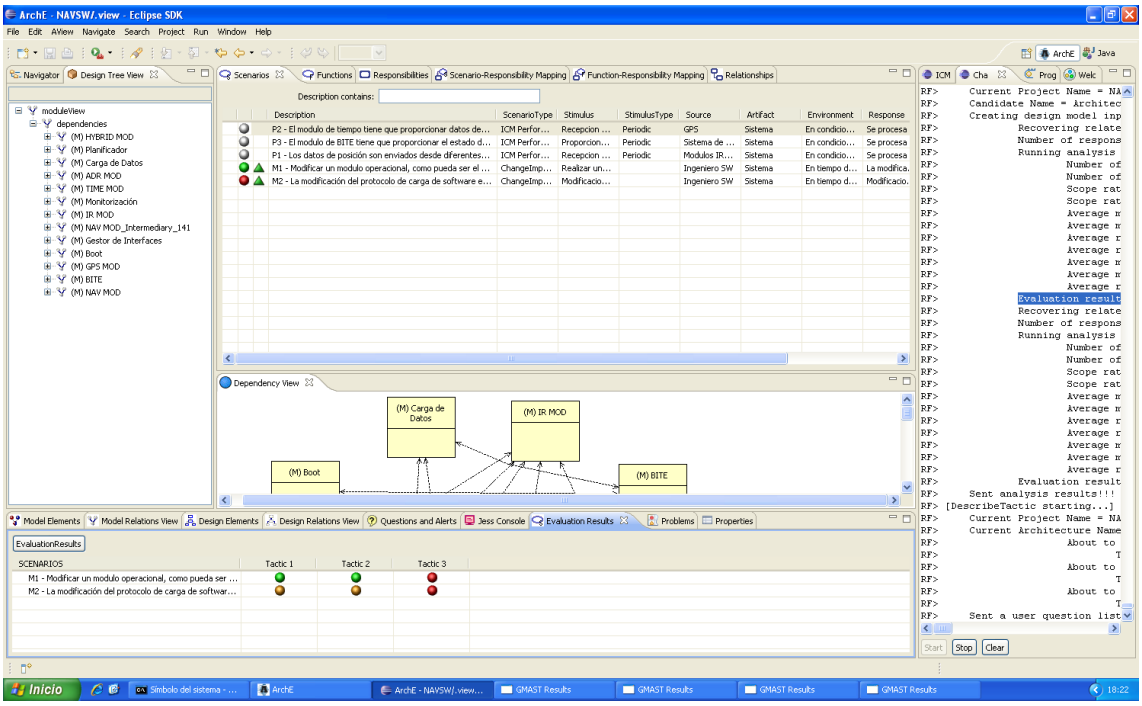


Figura 153. Resultado en M1 y M2 de aplicar la Táctica 1.

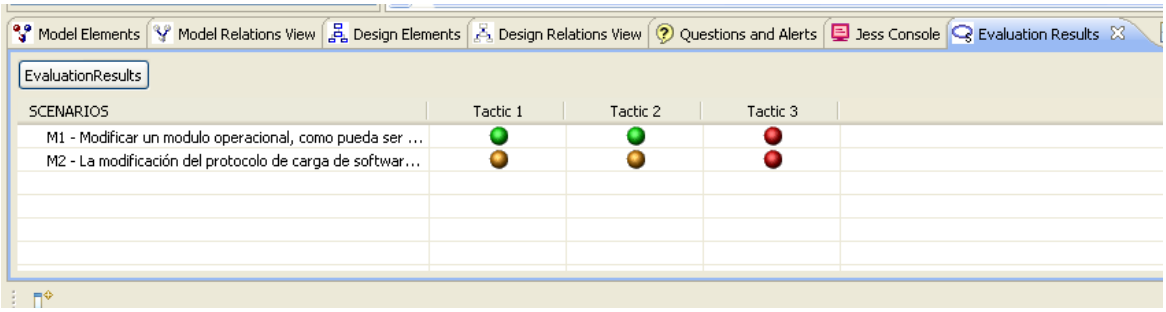


Figura 154. Resultado en M1 y M2 de aplicar la Táctica 1 (ampliado)

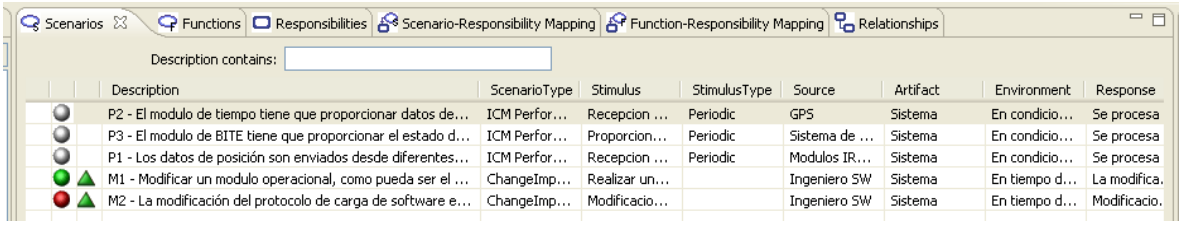


Figura 155. Resultado en los Escenarios M1 y M2.

Para mejorar M2 y cumplir el escenario, hay dos estrategias una vez quedan descartadas la aplicación de tácticas:

- Reducir coste del cambio de los módulos directamente afectados por el escenario (carga de datos, modulo BITE y NAV MOD).
- Relajar el escenario M2.

Si se aplica la primera estrategia, se obtienen los siguientes resultados (Figura 156 y Figura 157):

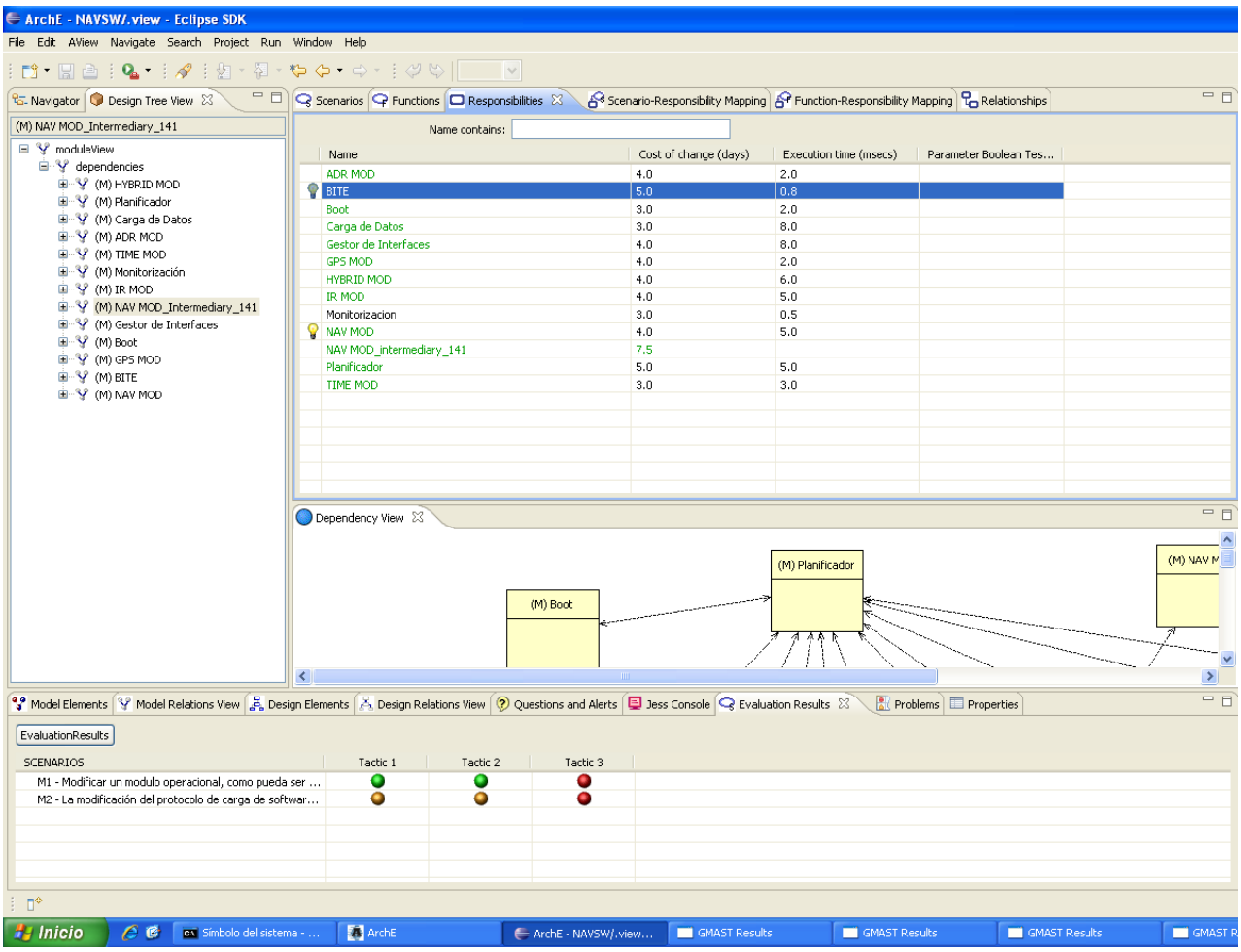


Figura 156. Coste del Cambio de los Módulos.

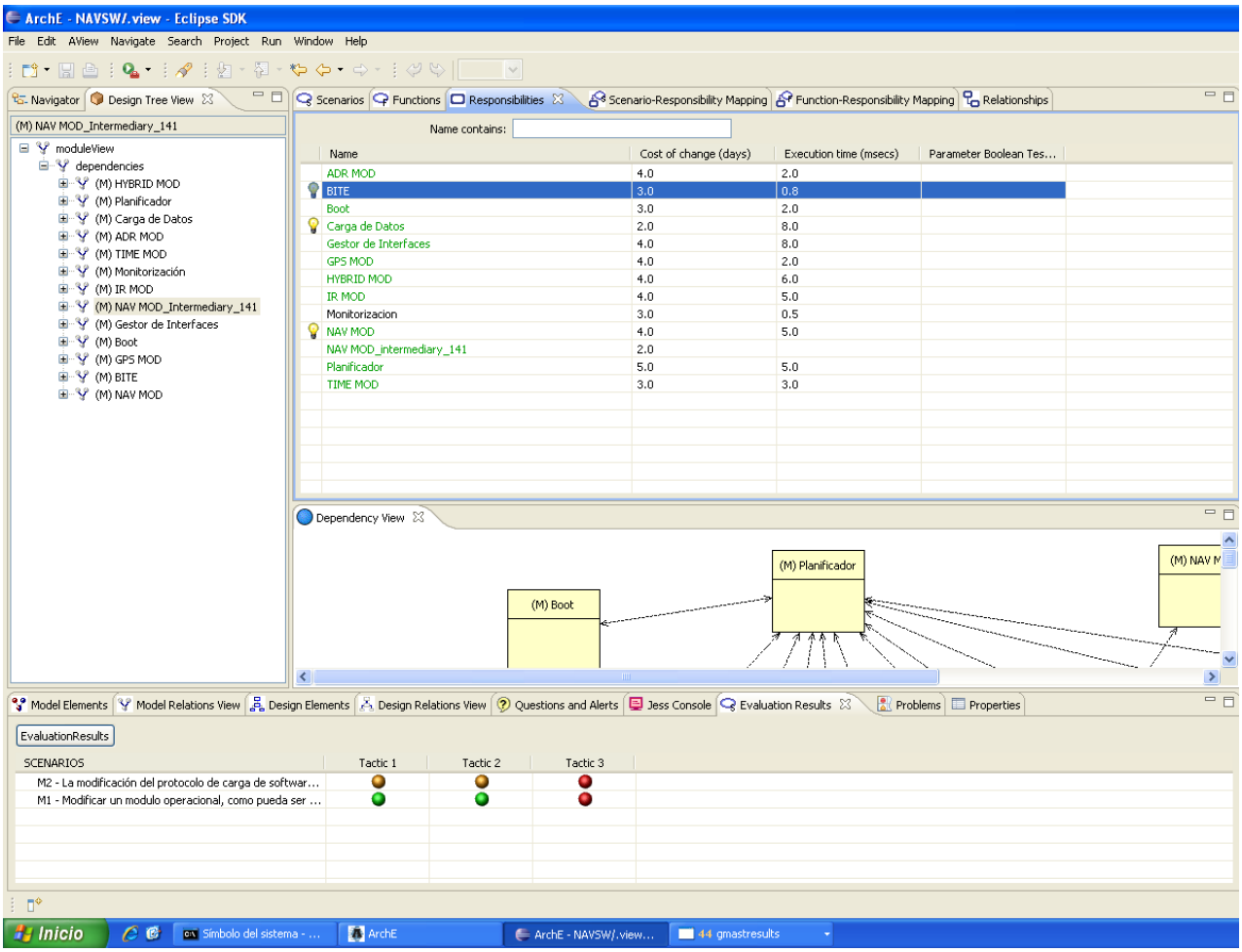


Figura 157. Reducción del Coste del Cambio de los Módulos.

Después de la reducción, ArchE recalcula el impacto en los escenarios de modificabilidad (Figura 158):

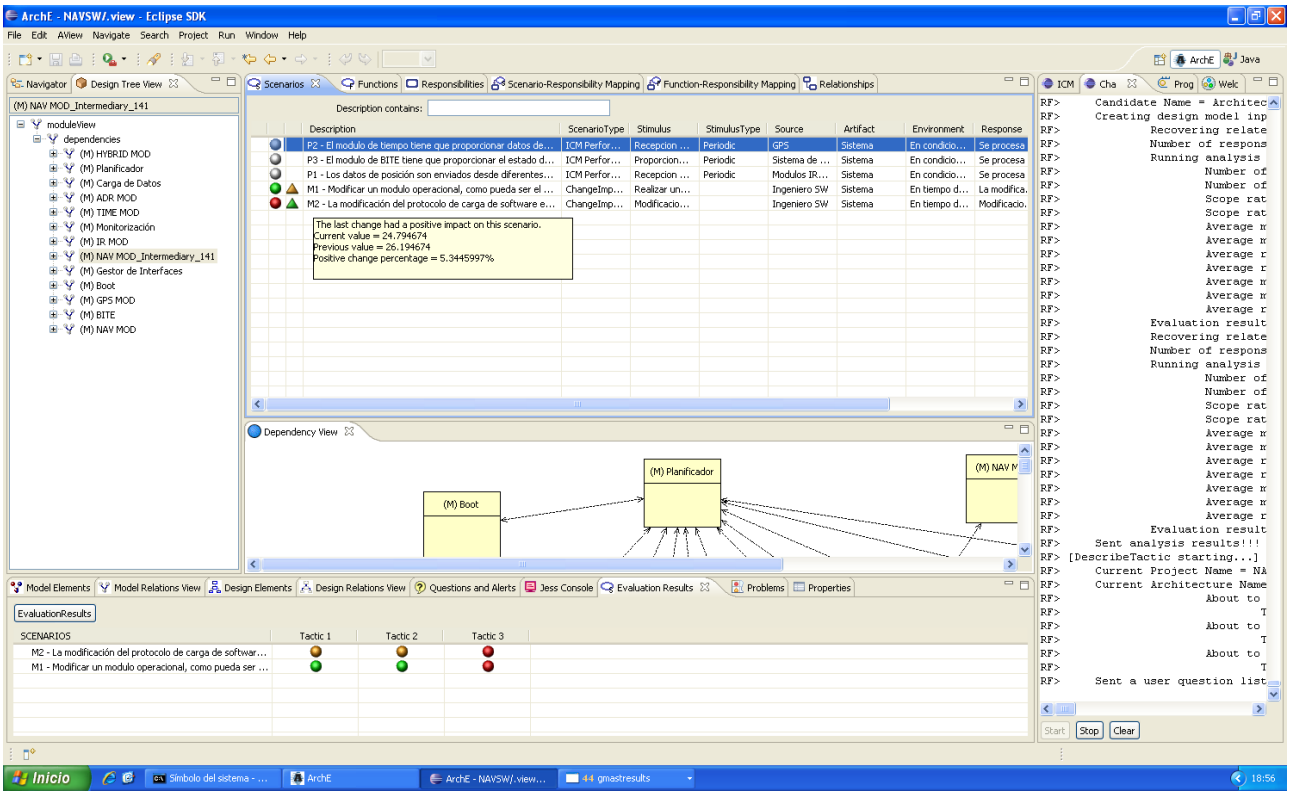


Figura 158. Impacto en los Escenarios de la Reducción en el Coste del Cambio en los Módulos.

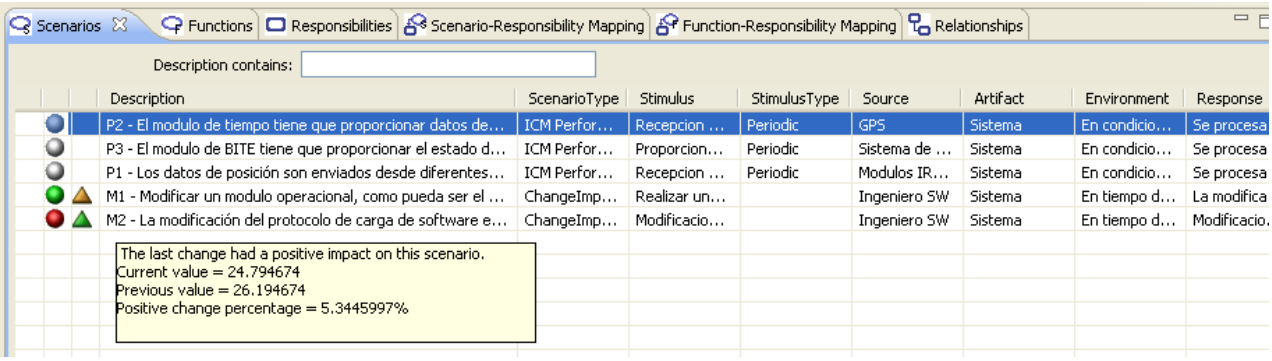


Figura 159. Impacto en los Escenarios de la Reducción en el Coste del Cambio en los Módulos (ampliado)

Ya no es posible rebajar más el coste del cambio. Por lo tanto, se aplicará la estrategia segunda, es decir, relajar el escenario cuanto sea posible y pasar de 20 a 25 días (Figura 160):

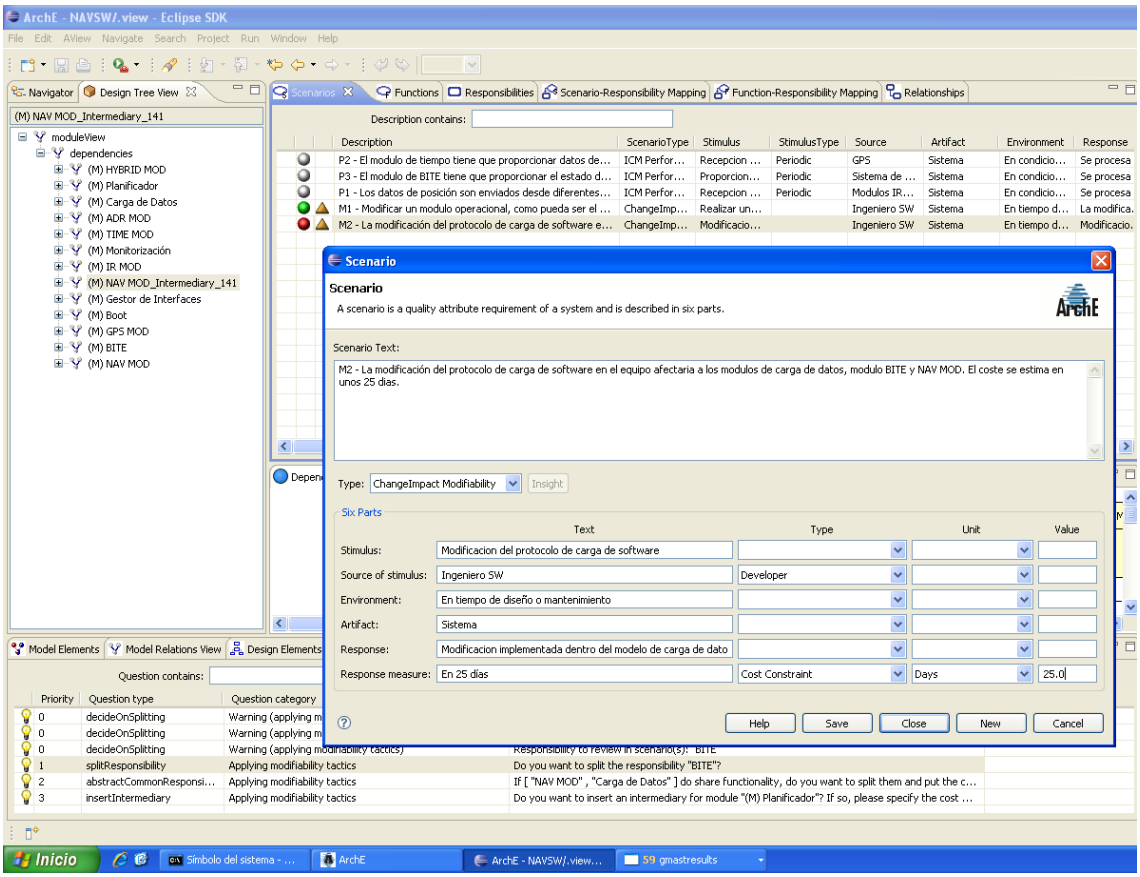


Figura 160. Relajación del Escenario M2 – de 20 a 25 días.

Se puede observar que con este último cambio sí se cumple el escenario (Figura 161 y Figura 163):

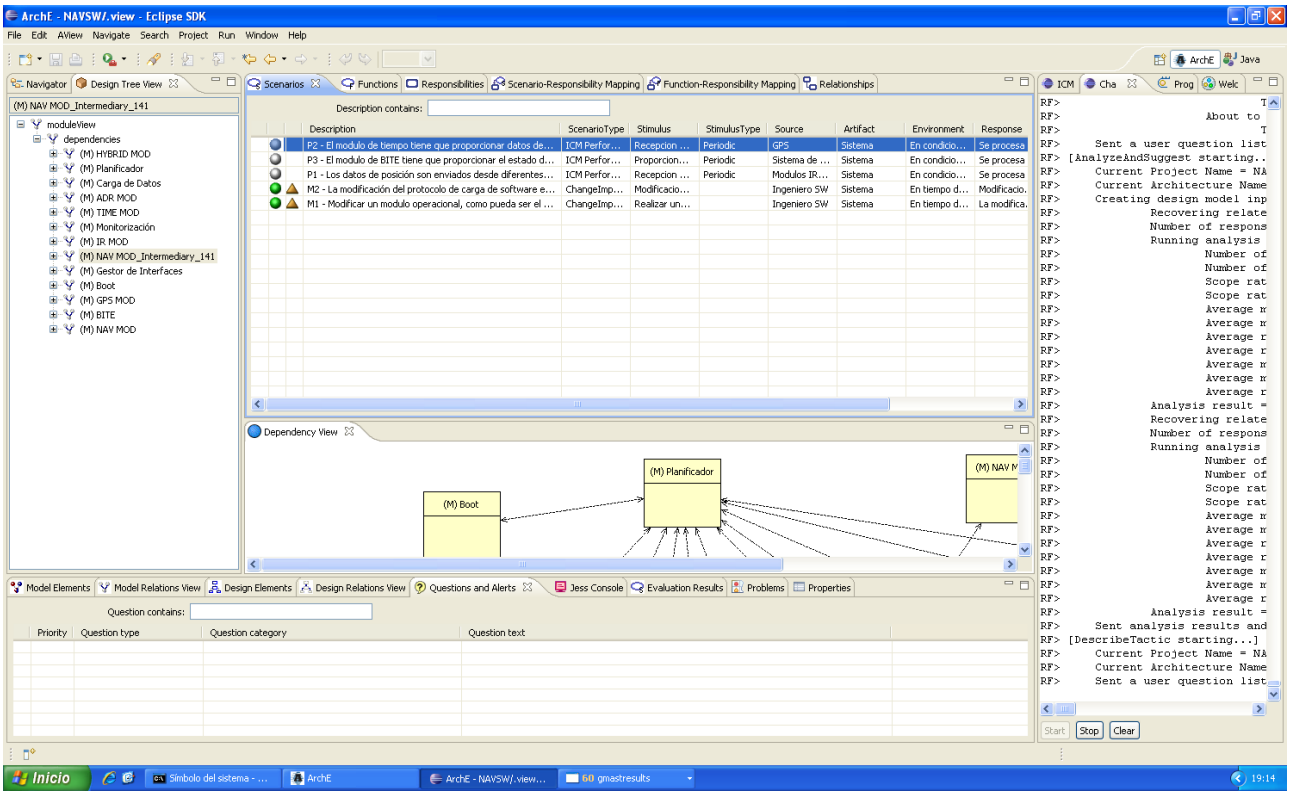


Figura 161. Impacto en el escenario M2 de la relajación de las condiciones.

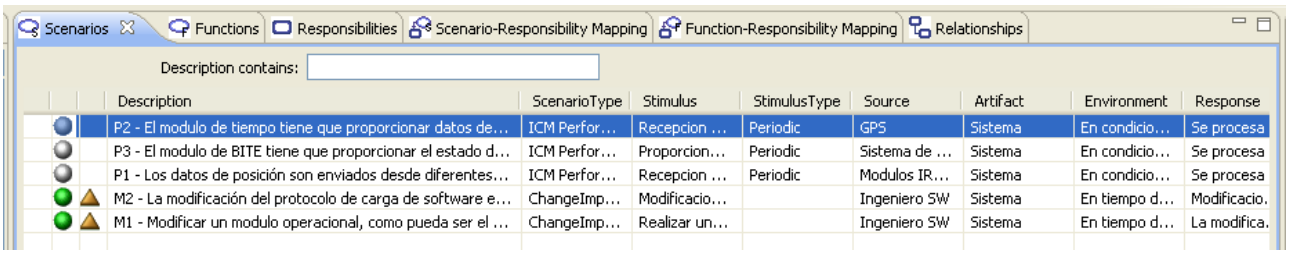


Figura 162. Impacto en el escenario M2 de la relajación de las condiciones (ampliado).

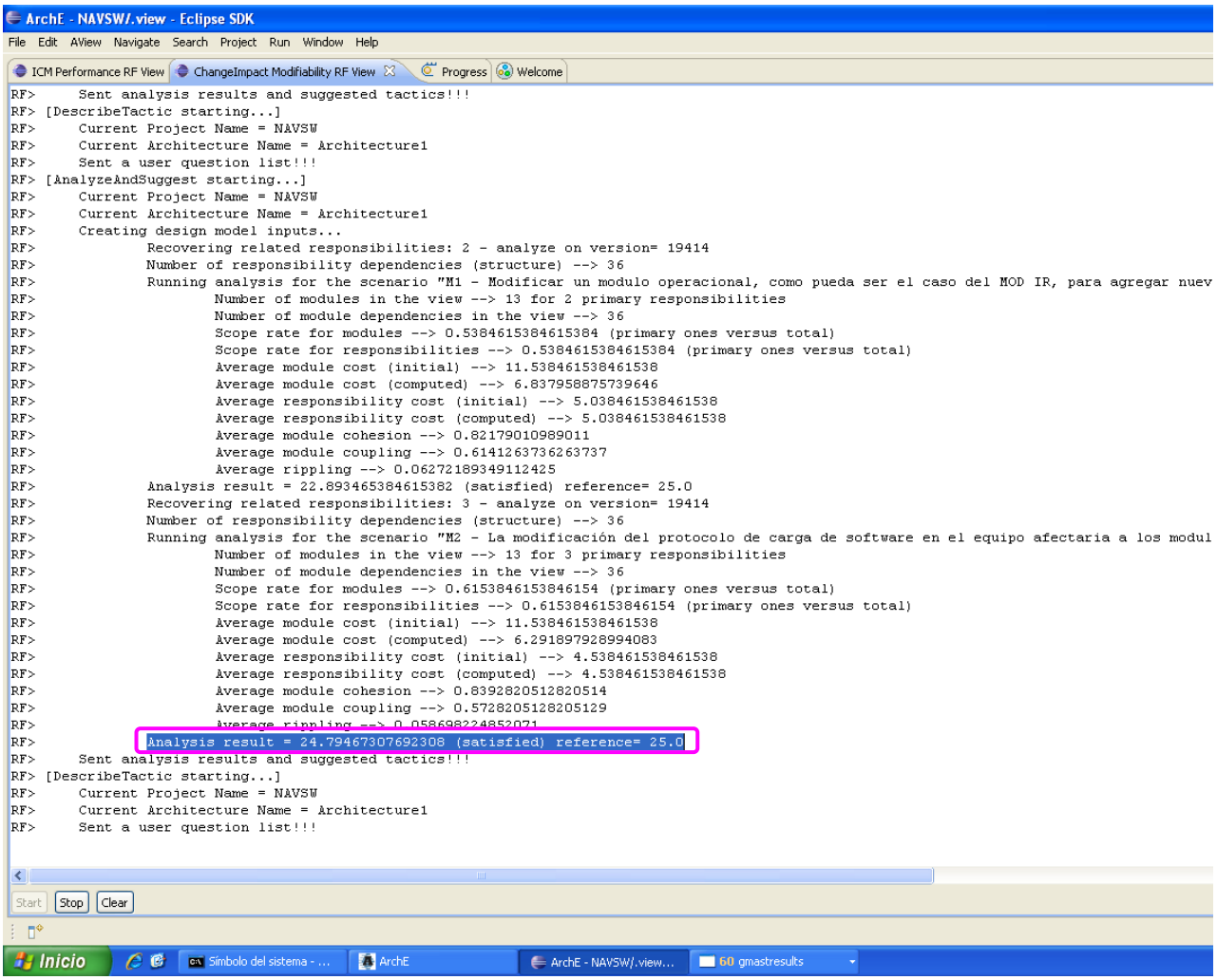


Figura 163. Ventana del Marco de ChangeImpact Modifiability después de relajar el Escenario M2.

ArchE ha sugerido una serie de tácticas, y con estas se ha conseguido una mejora en los escenarios. Después de exprimir al máximo las posibles mejoras que ArchE propone y de comprobar que no es posible mejorar más, lo que ArchE le muestra al arquitecto software es que la herramienta ha explorado los límites de la arquitectura. Al arquitecto no le queda otro remedio que plantearse si los escenarios que ha planteado no eran demasiado exigentes, vistas las dependencias que tienen los módulos de su arquitectura entre sí, y llegado a este caso, intentar relajar dichos escenarios, siempre y cuando esté dentro de la lógica y de lo posible el hacerlo.

8 ARCHE COMO ASISTENTE DE ARQUITECTURAS SW IMA

8.1 Introducción. Objetivos

En el capítulo 6 se ha estudiado una arquitectura típica de software embarcado. Pero no se han tenido en cuenta las peculiaridades de dicha arquitectura, ya que ArchE no es capaz de manejar ciertas características arquitectónicas, como las particiones en espacio y tiempo.

En este capítulo se va a detallar qué sería necesario implementar en ArchE para poder analizar mejor este tipo de arquitecturas.

8.2 Arquitectura Software Real del Sistema de Aviónica

Lo primero que hay que definir cuando se está diseñando una arquitectura de aviónica modular es el número de particiones y su DAL (tal y como se definió el concepto de DAL en el capítulo 5). En la Tabla 41 se puede ver dicha asignación para cada módulo:

Módulo	DAL	Partición	#
Boot	A	K	1
Planificador	A	K	2
Monitorización	C	K	3
Logical Comms.	A	K	4
Carga de datos	C	SA	5
Gestor de interfaces	B	SA	6
BITE	D	SA	7
IR MOD	A	SA	8
ADR MOD	A	SA	9
GPS MOD	A	SA	10
HYBRID MOD	A	SA	11
TIME MOD	B	SA	12

Tabla 41. Módulos Software y su DAL y Partición asociados.

- DAL: A, B, C, D, E
- Partición: K (Kernel); SA (Standalone); SH (Shared).

Como puede observarse, hay varios tipos diferentes de particiones. Según sea el tipo de partición asignado a cada módulo, la arquitectura variará.

Si un modulo tuviera un estado de partición como SH, se podría meter con otro módulo del mismo nivel de DAL en una sola partición, siendo así dos procesos distintos que podrían comunicarse entre sí.

Si un módulo forma parte del Kernel (K), se mantiene separado del resto, aunque se puede comunicar directamente, tanto con otros elementos tipo K como con otros elementos tipo SA.

Si un módulo está considerado SA, hay que implementarlo en una partición exclusiva y su única via de comunicación con otros elementos tipo SA es via kernel a través de Logical Comms.

Se pueden apreciar las primeras diferencias con la arquitectura inicialmente planteada:

- Hay determinados módulos que ahora están clasificados como Kernel, pues será dicho kernel del SO su lugar de ejecución.
- Hay módulos que están clasificados con un determinado nivel de desarrollo y ocupando una partición; por tanto, estos no podrán comunicarse directamente con módulos situados en otras particiones, y tendrán que hacerlo a través de un módulo adicional, situado en el kernel, y denominado Logical Communications.
- El módulo NAV MOD, que englobaría la parte común, ya no tendría sentido en un esquema rígido tan sumamente particionado, y su funcionalidad se integraría en cada uno de los módulos IR MOD, ADR MOD, GPS MOD y TIME MOD.
- La comunicación entre módulos del kernel y particiones sí puede ser directa; no así entre particiones entre sí.

En la Figura 164 se puede apreciar cómo sería la arquitectura de este modelo según la norma ARINC 653

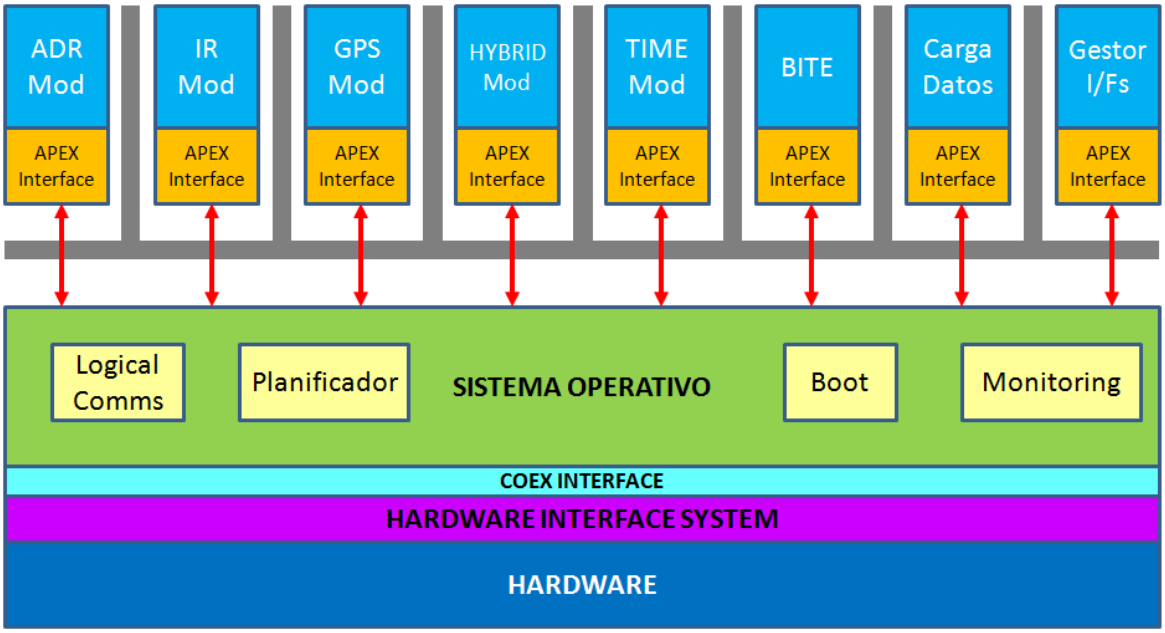


Figura 164. Arquitectura Software IMA real del Sistema de Aviónica.

Se puede ver en la Figura 165 cómo quedaría la arquitectura real, según el modelo UML:

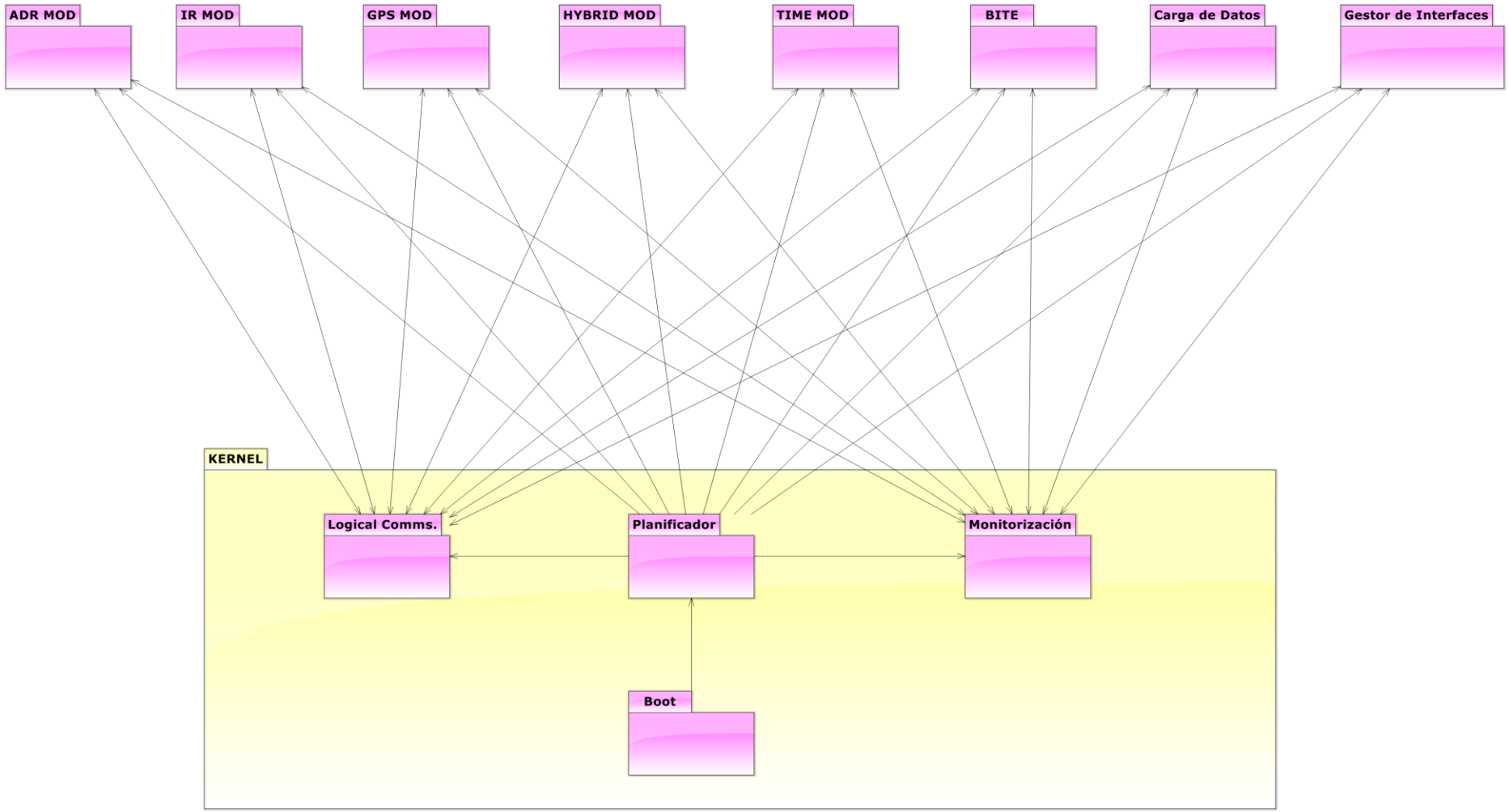


Figura 165. Arquitectura UML de Software del Equipo Embarcado.

8.3 Definición de los Nuevos Marcos de Razonamiento en ArchE

Se plantean aquí dos nuevos marcos de razonamiento. El primero de ellos está orientado al establecimiento de particiones asociadas a uno o varios módulos y el establecimiento de comunicaciones entre diversas particiones. El segundo de ellos está orientado a la planificación temporal del sistema.

8.3.1 Marco de Razonamiento *Space Partitioning*.

Para que ArchE pueda razonar sobre dicho esquema de arquitectura particionado, se necesitaría incluir un nuevo marco de razonamiento, que tuviera en cuenta:

- DAL asociado a cada módulo y tipo de partición de los mismos.
- Comunicación entre las diferentes particiones, definiendo para ello canales de comunicación y puertos.
- Incompatibilidades de módulos de diferentes DAL en una misma partición.
- Posibilidad de que módulos del mismo DAL puedan compartir una misma partición; en este caso, se redefinirían como procesos.

Un ejemplo dado por la ARINC 653 [21] se muestra en la Figura 166 . En dicha figura se asignan nombres a los puertos y canales. Además se puede observar las condiciones iniciales de partida en cuanto a período de repetición de las particiones y tiempo de ejecución. Toda esta información se codificará en ficheros XML dentro del software del sistema de aviónica. En el apartado siguiente se volverá a incidir en este ejemplo para la definición de las ventanas de tiempo.

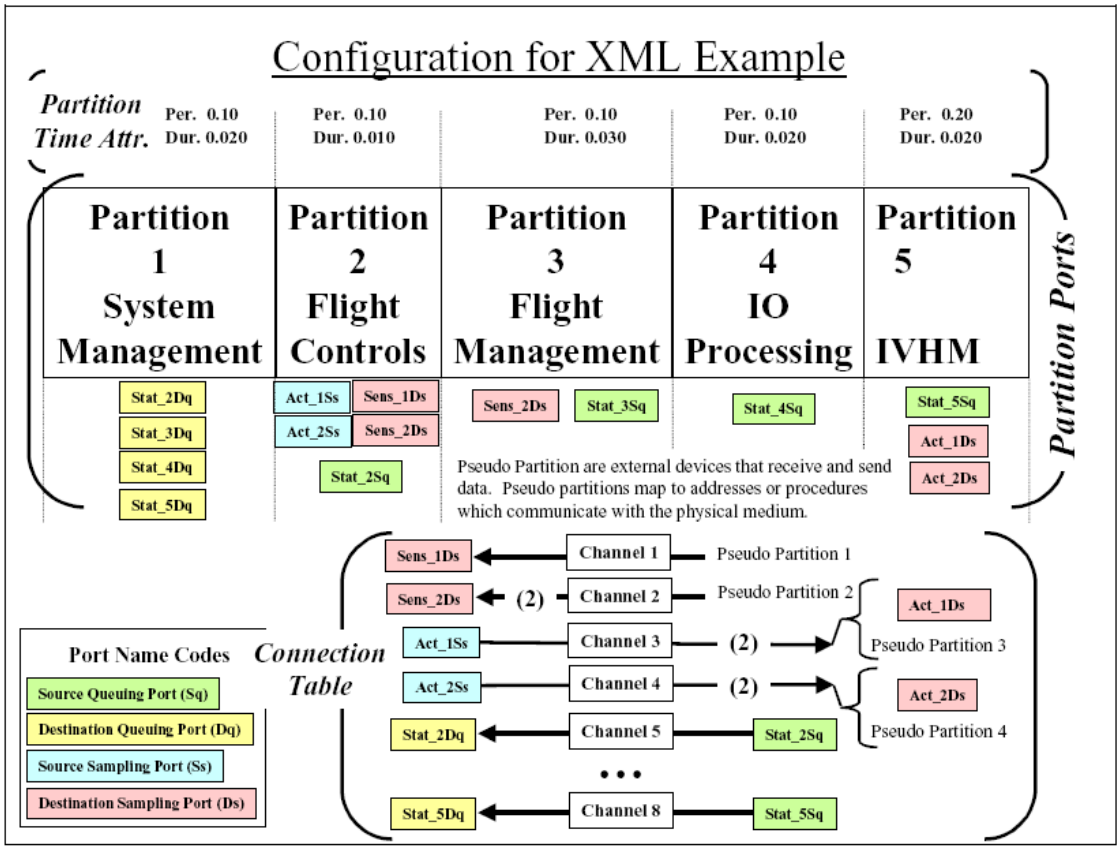


Figura 166. Configuración para el fichero XML de ejemplo. [21]

Se definiría, así mismo, un nuevo tipo de relaciones entre responsabilidades (en este caso, entre módulos):

- logicComm (Logical Communications), para establecer una comunicación lógica entre particiones. La primera responsabilidad será el origen o *source*, y la segunda será el destino o *destination.*
- processComm (Process Communication), para establecer una comunicación lógica entre procesos, dentro de una misma partición.

Se definirá un escenario general único, que se mapeará a todas las responsabilidades, y que una vez analizado, dará la arquitectura final. Esto se hace para seguir la filosofía general de ArchE en cuanto a relaciones entre escenarios y responsabilidades.

Aprovechando las ventajas de la interfaz gráfica de ArchE, el sistema quedaría de la siguiente manera: (Figura 167)

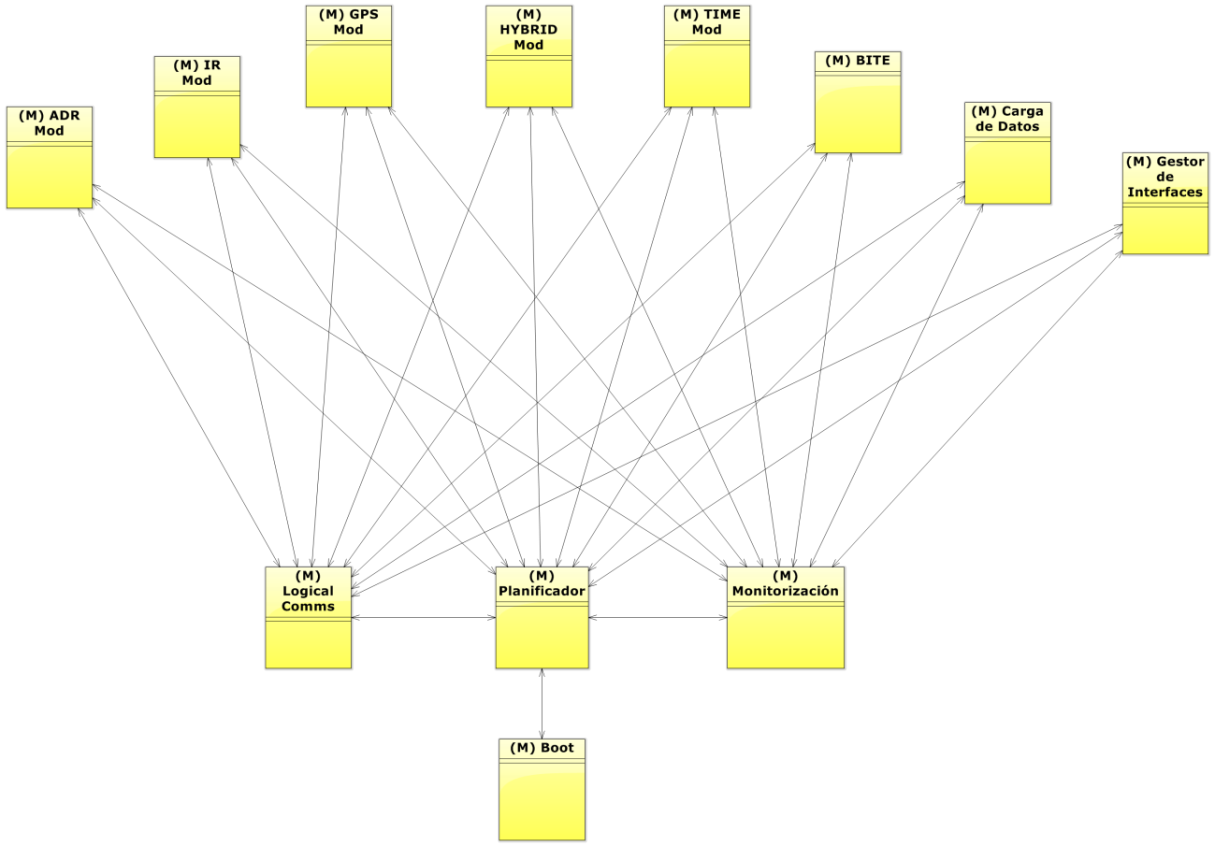


Figura 167. Esquema de Arquitectura IMA de ArchE.

Una vez finalizada la primera iteración, ArchE podrá realizar sugerencias; por ejemplo:

- si el diseñador ha cometido un error (en este caso el escenario no se cumpliría) y ha establecido una relación de processComm entre dos módulos de distinto DAL (y por tanto situados en diferentes particiones), ArchE sugiriría:
  - Convertir la relación de processComm en logicComm, lo cual sí está permitido.
  - Sugerir si la aplicación de menor DAL debe ser convertida al DAL mayor; en este caso, ambas serían convertidas en procesos y se ejecutarían en la misma partición.
- Para todas las relaciones logicComm, ArchE preguntaría sobre el tipo de relación más apropiado: *queuing* o *sampling*.
  - *queuing*: en este modo, cada instancia nueva de un mensaje puede llevar datos diferentes y por tanto no está permitido sobrescribir mensajes previos durante la trasferencia.
  - *sampling*: en este modo mensajes sucesivos llevan datos idénticos pero actualizados. No hay colas y un mensaje permanece en el puerto *source* hasta que es transmitido o sobrescrito por una nueva instancia del mensaje.
- Para todas las relaciones processComm, Arche preguntaría al diseñador sobre el tipo de relación más apropiado: *buffer* o *blackboard*.
  - *buffer*: es el mismo concepto de queuing, pero aplicado a comunicación entre procesos.
  - *blackboard*: es el mismo concepto de sampling, pero aplicado a comunicación entre procesos.

El marco de razonamiento establecería los atributos de los puertos, designando:

- identificador de la partición que tiene acceso al puerto
- nombre del puerto
- modo de trasferencia (*sampling*/*queuing*)
- dirección de trasferencia (*source*/*destination*)

Con toda esta información, ArchE iteraría de nuevo y sacaría la lista definitiva de canales de comunicaciones entre particiones, y la lista de puertos de cada partición, tal y como se muestra en la Figura 168:

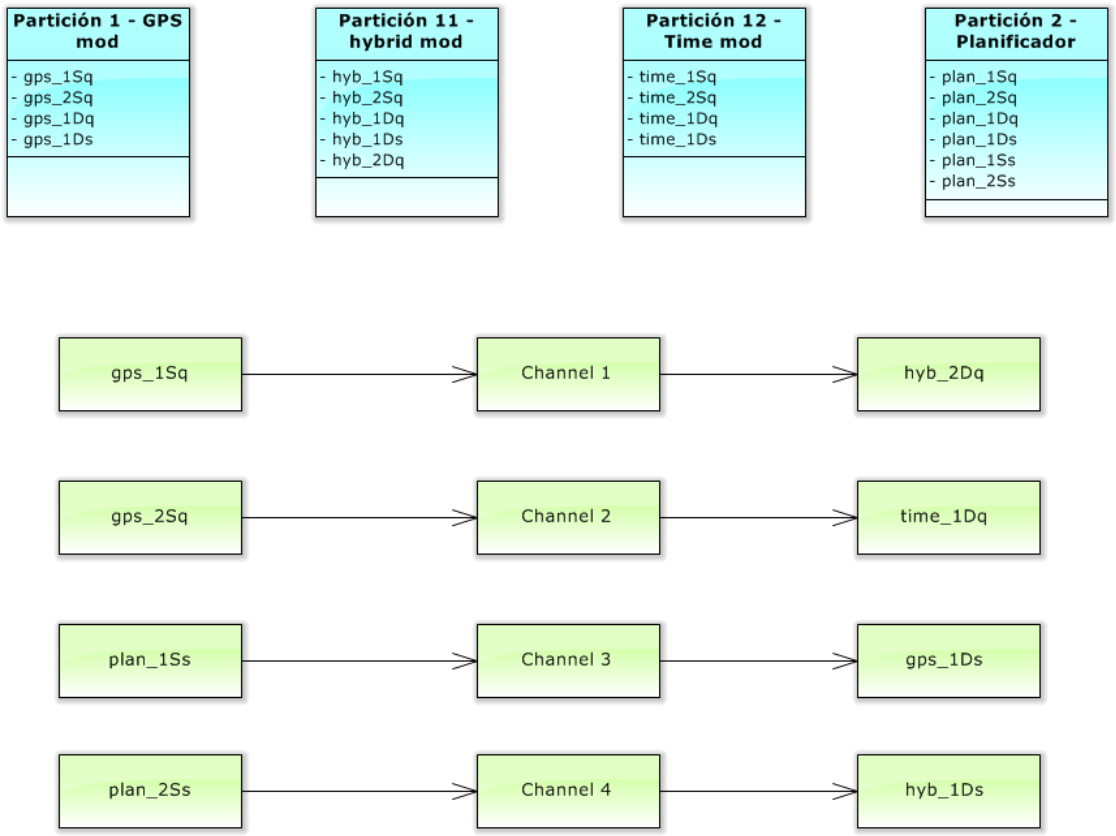


Figura 168. Particiones y sus Canales y Puertos de Comunicaciones asociados.

La interfaz de ArchE para introducir el DAL y el tipo de partición quedaría de la siguiente manera (Figura 169 y Figura 170)

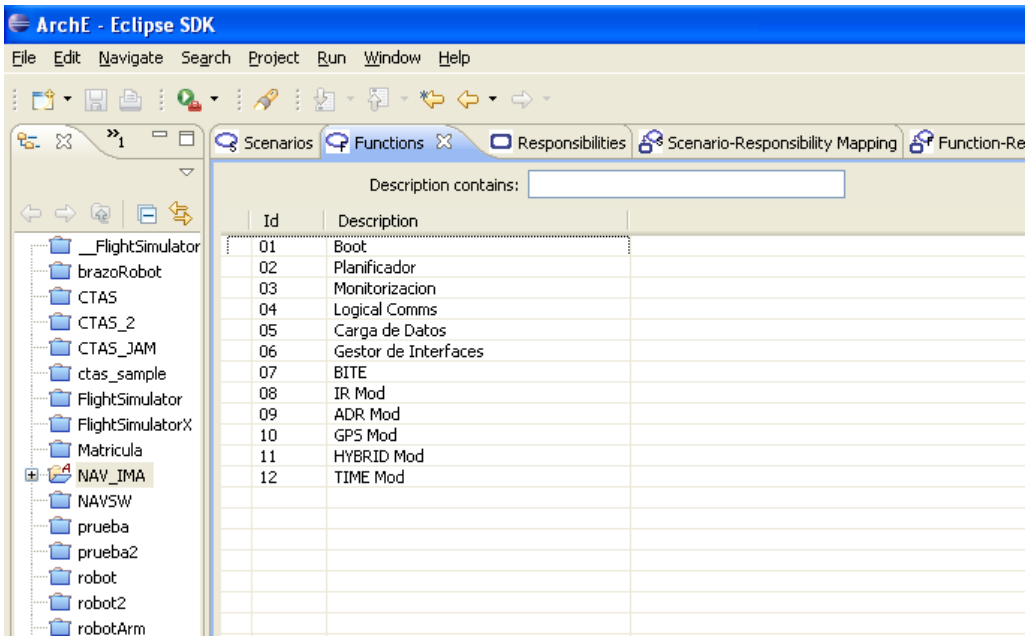


Figura 169. Funciones de la Arquitectura IMA.

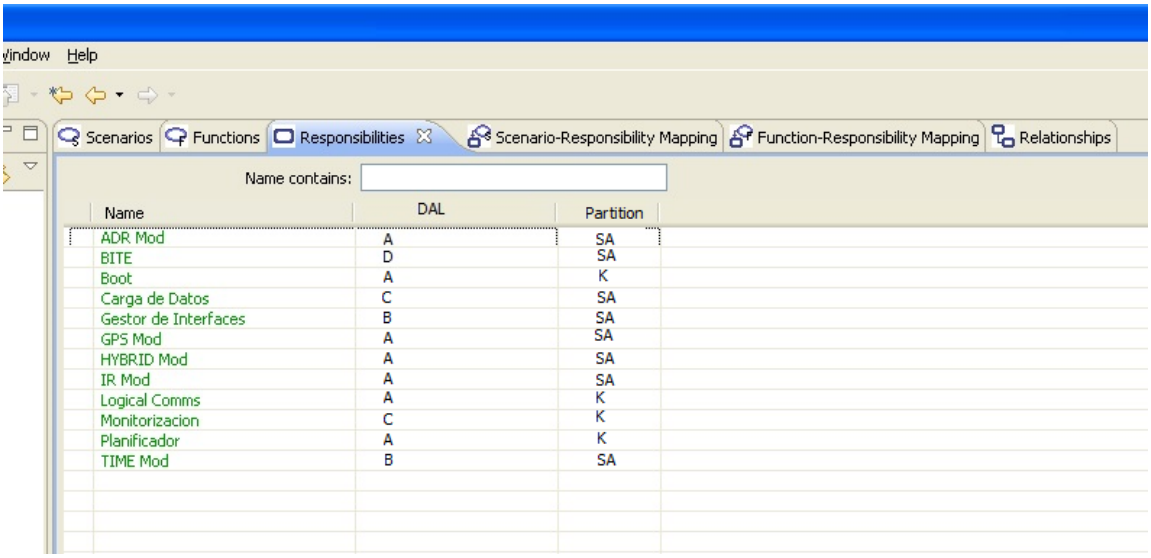


Figura 170. Asignación de Particiones y DAL en ArchE.

Una vez introducidas las responsabilidades, se establecerán las nuevas relaciones entre ellas: (Figura 171 y Figura 172)

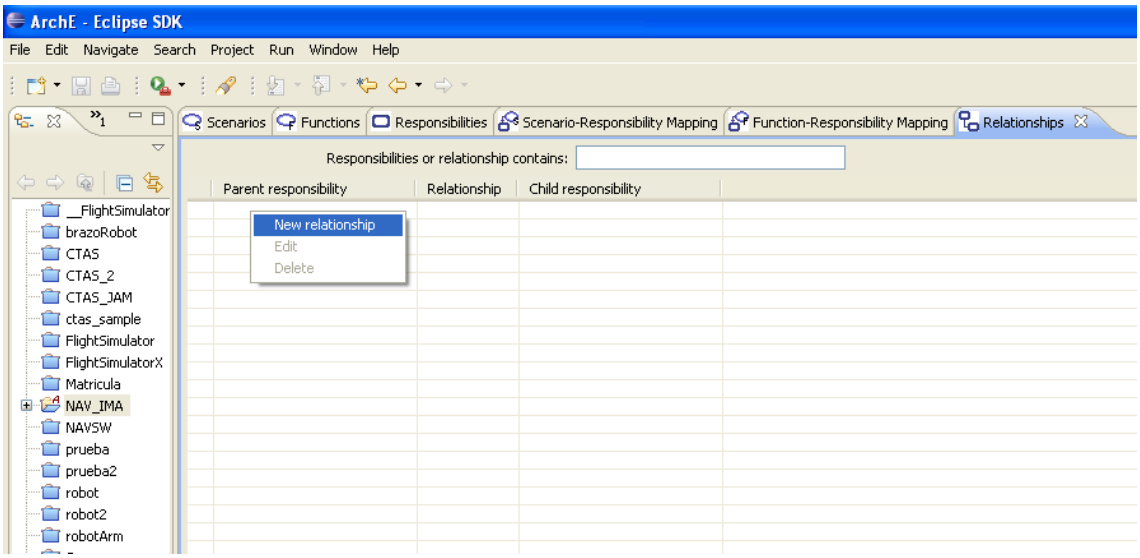


Figura 171. Nuevo Menú para Relaciones.

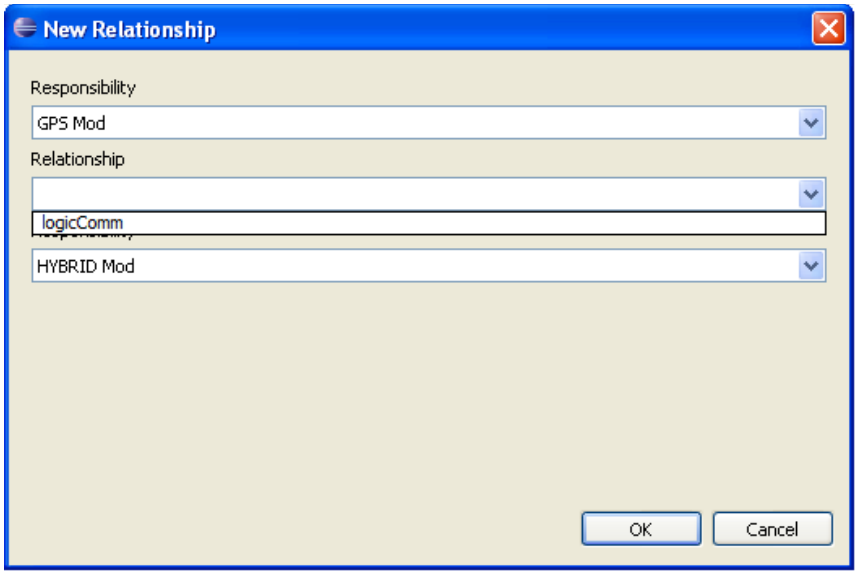


Figura 172. Selección de nuevas relaciones.

En la Figura 173 se ve cómo quedarían las nuevas relaciones; las comunicaciones entre particiones y planificador/monitorización son directas, no a través de Logical Comms, por eso el campo en ArchE aparece vacío.

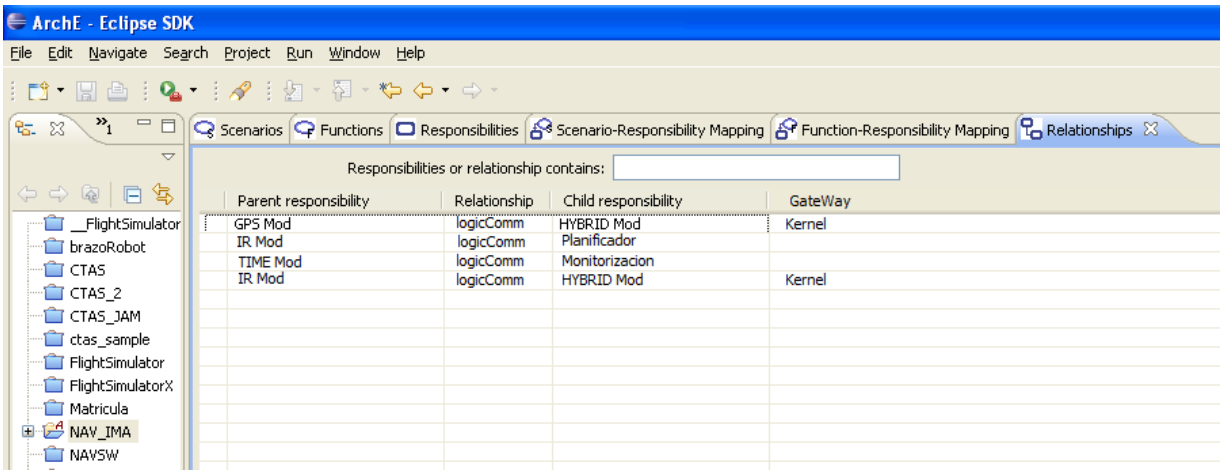


Figura 173. Relaciones logicComm y tipo de Gateway.

8.3.2 Marco de Razonamiento *Time Partitioning*

Este segundo marco de razonamiento está muy ligado con el de *space partitioning*, ya que si no se establece una comunicación entre módulos, no se puede iterar en busca de *performances*.

A pesar de que este nuevo marco de razonamiento guarda muchas similitudes con *ICM Performance*, hay ciertas características que los diferencia. La mayor diferencia es que ahora se definen para cada partición el período y la duración del tiempo de ejecución, y el instante de inicio de la ventana de ejecución, y es algo que no se puede variar. En el *ICM performance* se definía la duración de las responsabilidades, pero no se asignaba ninguna ventana a las mismas. Además, los escenarios tenían en cuenta estímulos externos, con un período y una *deadline*. Aquí, las *deadlines* vienen dadas por la duración de la ventana de tiempos.

Se definen varios parámetros para cada partición:

- *Windows offset*: el tiempo de inicio de la ventana.
- *Windows duration*: el tiempo de duración de la ventana, que corresponde al tiempo de ejecución de la partición.
- *Windows ID*: la identificación de la ventana relativa a la partición y a la trama principal considerada. La trama principal se definirá a continuación.
- *Partition Time Attributes*:
  - *Partition period*: es el período de repetición de la partición.
  - *Partition duration*: es el tiempo de ejecución de la partición.

La trama principal es, en este contexto, aquella planificación de ventanas de ejecución cuyo período mínimo o superperíodo es tal que permite la ejecución al menos una vez de la partición de período de repetición mayor. El superperíodo de dicha trama principal se elige, por tanto, igual al período de repetición mayor de entre todos los períodos de repetición de todas las particiones.

Es decir, se define un período de repetición de cada partición, y se asigna un tiempo a dicha partición para que se ejecute. Una vez sobrepasado este tiempo, el planificador dará paso a la siguiente partición, según la planificación que se haya realizado.

Por lo tanto, para que ArchE pudiera procesar este tipo de marco de razonamiento, habría que añadir a cada responsabilidad (en este caso, partición), los valores de período y duración. El marco de razonamiento cogería el período mayor de entre todos los asignados a las particiones, y crearía una trama principal. Dentro de esta trama, se planificarían el resto de las particiones, teniendo en cuenta sus períodos.

Los escenarios propondrían un estímulo, que afectase por ejemplo a varias particiones, y crearía una tarea, con una *deadline* impuesta por el estímulo que se tendría que cumplir, teniendo en cuenta:

- Las diferentes ventanas de ejecución de las particiones
- El tiempo del superperíodo o trama.

Por lo tanto, el escenario y el marco de razonamiento en este caso se parecen bastante al *ICM Performance*, pero modificado, para ser adaptado al contexto de la norma ARINC 653. Además, dicho marco está íntimamente relacionado con el anterior de *space partitioning*, pues en dicho marco se definan las particiones, y además se definen las comunicaciones tanto en el interior de las particiones (entre procesos) como entre particiones. Por lo tanto, sin el anterior marco, este otro no tiene ningún sentido.

A continuación se incluyen dos ejemplos de cómo este marco de razonamiento establecería las diferentes ventanas de ejecución de las diferentes particiones:

En el primer ejemplo, descrito en [21] y visto ya en la sección anterior, la asignación de las cinco particiones a las ventanas de particiones se muestra en la Figura 174. Ya que la partición de período de repetición mayor define el período de la trama principal o superperíodo, para este ejemplo sería de 200 milisegundos. Los requisitos de ventana definirían el tiempo de inicio (*offset*) de cada ventana y su duración para todo el superperíodo.

El tiempo de inicio de la ventana inicial es 0 ms; el tiempo de inicio de la siguiente ventana es el tiempo de inicio de la ventana inicial más el tiempo de duración de la ventana inicial; y así sucesivamente hasta completar la trama principal de superperíodo 200 ms. Como se puede ver, la partición 4 tiene dos ventanas para cada período (100 ms) y cuatro ventanas dentro del superperiodo de la trama principal. Las particiones 1, 2 y 3 tienen dos ventanas y la partición 5 tiene una ventana dentro del superperiodo de la trama principal.

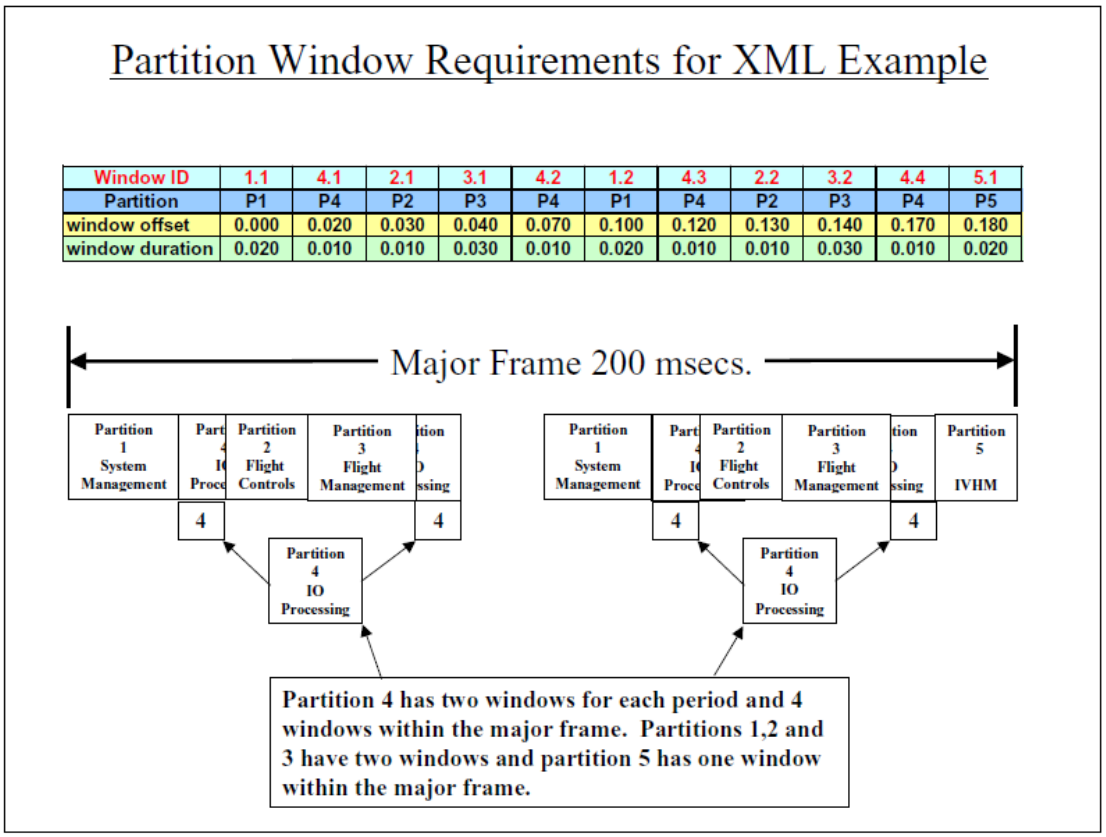


Figura 174. Requisitos de Tiempo de Ventana de las Particiones. [21]

En el segundo ejemplo (Figura 175), descrito en [17], se puede observar que el tiempo de la trama principal (*Mayor Frame* o MAF) o superperiodo corresponde de nuevo al período mayor de las tres particiones, es decir, 100 ms; y el período de repetición (*Minor Frame* o MIF) es el máximo común divisor de los períodos de las particiones, es decir, 25 ms.

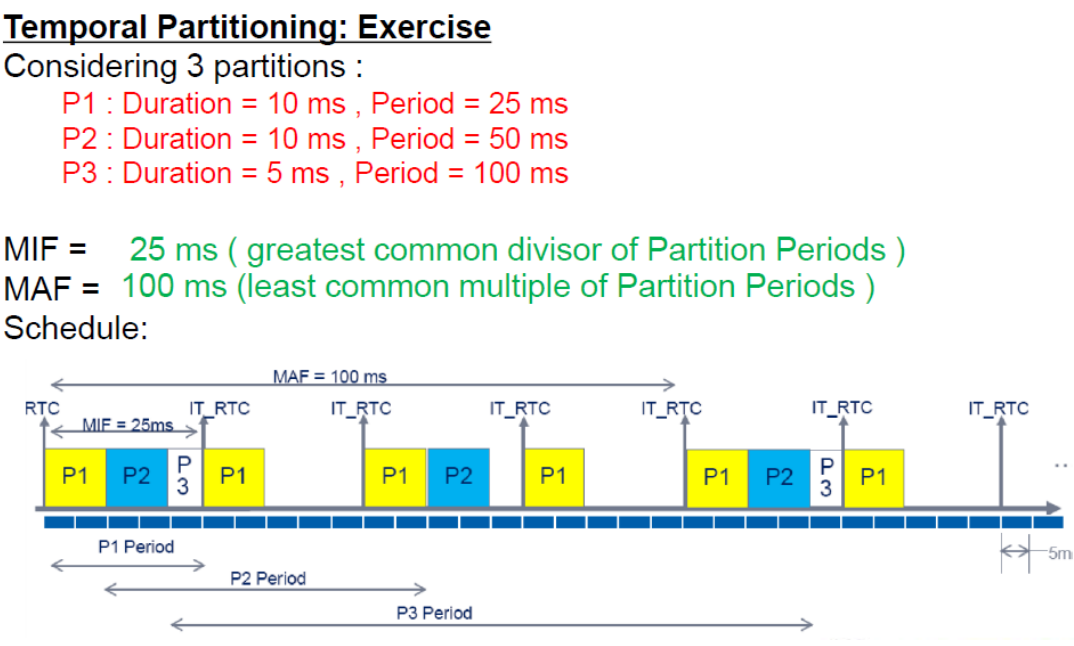


Figura 175. Distribución de Tiempos de Ventanas de las Particiones. [17]

Para el ejemplo objeto de este trabajo, no se ha representado la distribución temporal de ventanas de tiempo, dada su complejidad, pero se calcularía de la forma vista en los ejemplos anteriores.

Las ventanas de Arche quedarían de la siguiente manera: (Figura 176)

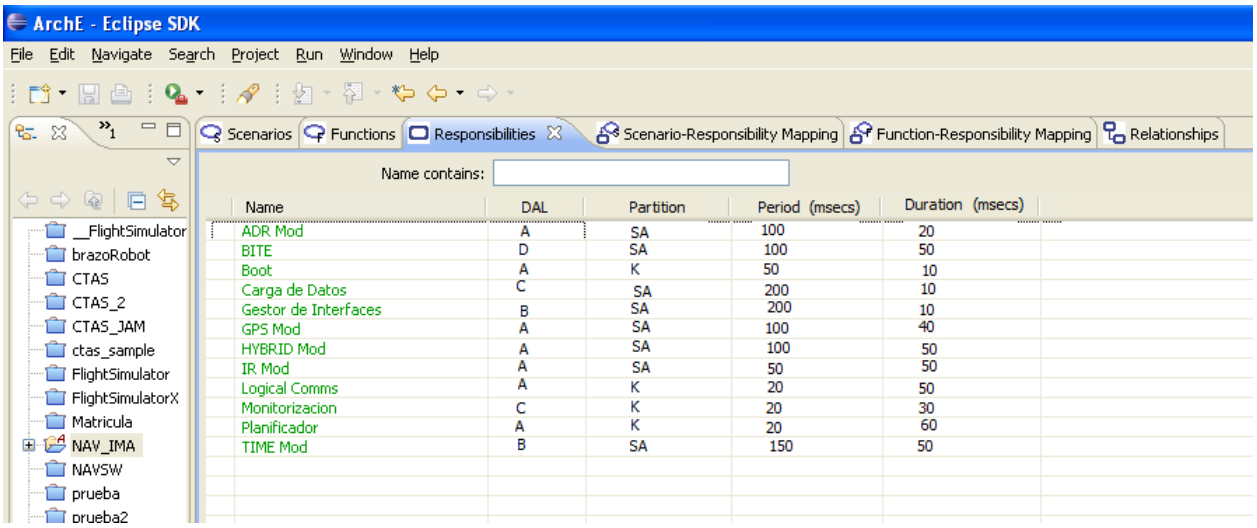


Figura 176. Responsabilidades con DAL, Tipo de Partición, Período y Duración.

Un posible escenario podría ser el siguiente:

El módulo GPS, ADR e IR mandan la posición en un período de 120 ms. La posición la calcula el módulo híbrido cada 90 ms.

Es decir, el escenario es el mismo que antes, pero cambia la forma de calcularlo. ArchE tendría que planificar todo el escenario, teniendo en cuenta las ventanas de tiempo de las particiones, su período de repetición, el período de la trama principal o superperiodo, etc.

Otras diferencias con *ICM Performance* serían:

- el establecimiento de canales de comunicación entre todos los implicados. Por ejemplo, si previamente no se hubiera establecido un canal de comunicación entre alguna de las particiones, el escenario directamente no se cumpliría y ArchE propondría revisar los canales de comunicación establecidos.
- *ICM Performance* calcularía si el sistema es planificable sin tener en cuenta las ventanas de cada partición. No está claro, por ejemplo, la prioridad que asigna a cada proceso. En este caso, se asigna a cada partición una ventana estricta de tiempo, acorde con los períodos de cada partición y su tiempo de ejecución, y una vez ejecutada dicha ventana, se pasa a la siguiente partición.

8.4 Análisis de la Arquitectura IMA con Arche

Para terminar con el estudio de la arquitectura IMA con la herramienta ArchE, se va a analizar la arquitectura real IMA software del sistema de aviónica, intentando "traducir" las relaciones entre particiones al lenguaje que ArchE entiende, y teniendo en cuenta que habrá limitaciones que ArchE no podrá manejar.

Lo primero que hay que hacer es establecer nuevas relaciones de dependencia y reacción entra las particiones, que se muestran en las Tabla 42 y Tabla 43.

	Child Responsability											
Parent Responsability	Boot	Planificador	Carga de datos	Gestor de Interfaces	BITE	Monitorización	IR MOD	ADR MOD	GPS MOD	HYBRID MOD	TIME MOD	Logical Comms
Boot		D										
Planificador			D	D	D	D	D	D	D	D	D	D
Carga de Datos					D							D
Gestor de interfaces					D		D	D	D	D	D	D
BITE						D						D
Monitorización					D		D	D	D	D	D	
IR MOD										D		D
ADR MOD										D		D
GPS MOD										D	D	D
HYBRID MOD												D
TIME MOD												D
Logical Comms			D	D	D		D	D	D	D	D	

Tabla 42. Relaciones de Dependencia de la Arquitectura IMA.

	Child Responsability											
Parent Responsability	Boot	Planificador	Carga de datos	Gestor de Interfaces	BITE	Monitorización	IR MOD	ADR MOD	GPS MOD	HYBRID MOD	TIME MOD	Logical Comms
Boot		R										
Planificador			R	R	R	R	R	R	R	R	R	R
Carga de Datos												R
Gestor de interfaces												R
BITE												R
Monitorización			R	R	R		R	R	R	R	R	
IR MOD										R		R
ADR MOD										R		R
GPS MOD										R	R	R
HYBRID MOD												R
TIME MOD												R
Logical Comms			R	R	R		R	R	R	R	R	

Tabla 43.Relaciones de Reacción de la Arquitectura IMA.

Lo primero que se observa es que, a pesar de que hay dependencias debido a que los módulos *Boot*, Planificador, Monitorización y *Logical Comms* con otros módulos y entre sí, dado que estos módulos realizarán llamadas a procedimientos, intercambiarán datos, etc., no se deberían de considerar para los escenarios de modificabilidad, puesto que dichos módulos forman parte del Kernel, que no se modifica ya que como se vio en el capítulo 6, el kernel es un elemento COTS (*Commercial Off-The-Shelf*) que es proporcionado por un suministrador independiente y no se puede modificar.

Otra observación muy relevante es el hecho de que, para modelar las comunicaciones entre particiones, el módulo *Logical Comms* se tiene que tener en cuenta dos veces; primero como receptor de datos de la partición fuente, y segundo como emisor hacia la partición destino; por ejemplo, si la partición IR MOD desea mandar datos a la HYBRID MOD, ha de hacerlo a través de *Logical Comms*; por tanto, por la definición del marco de rendimiento, habrá que establecer una relación de reacción, primero entre IR MOD y *Logical Comms*, y después entre *Logical Comms* y HYBRID MOD. Las consecuencias de esta relación se analizarán cuando se establezcan los escenarios, aunque todo parece indicar que habrá un "cuello de botella" en el módulo Logical Comms.

Los escenarios para comprobar la arquitectura son los mismos ya establecidos en el capítulo 7. La única variación se ha introducido en el M1, ya que ahora al desaparecer el NAV Mod, se necesitaría que el escenario afecte a más módulos, para que sea más representativo e interesante; se supone que en este caso, la modificación del IR MOD podría afectar al HYBRID MOD, con el que tiene una gran relación. Además, también cambia el hecho de que en las comunicaciones entre particiones, habrá que incluir al módulo *Logical Comms*: (Tabla 44)

	Escenarios				
Responsabilidades	**M1**	**M2**	**P1**	**P2**	**P3**
IR MOD	X		X		
HYBRID MOD	X		X		
GPS MOD			X	X	
TIME MOD				X	
ADR MOD			X		
BITE		X			X
Carga de Datos		X			
Monitorización					X
Logical Comms			X	X	

Tabla 44. Mapeo Escenarios-Responsabilidades en IMA.

Las funcionalidades, una vez introducidas en Arche, quedarían así: (Figura 177)

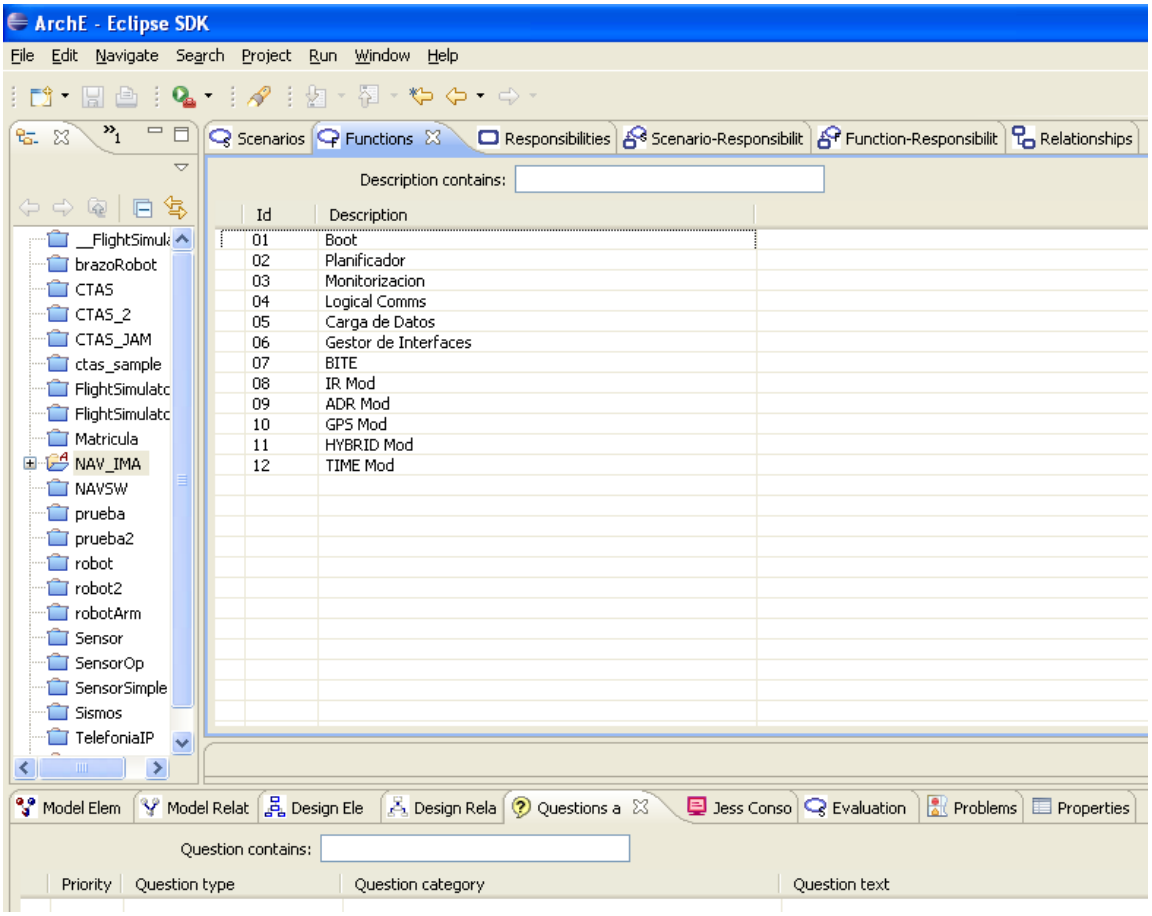


Figura 177. Funciones de IMA en ArchE.

Las responsabilidades, junto con su coste de cambio y tiempos de ejecución, quedarían así: (Figura 178)

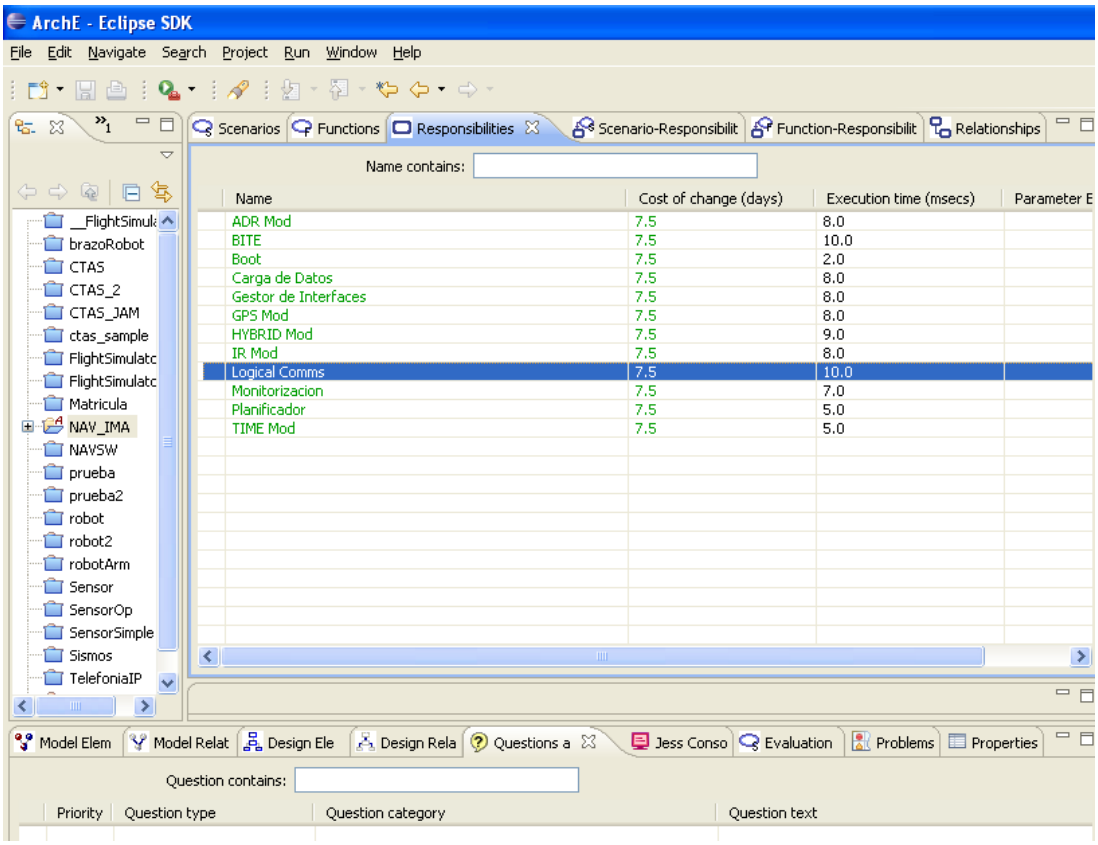


Figura 178. Responsabilidades de IMA con sus parámetros.

El mapeo de funciones-responsabilidades quedaría así: (Figura 179)

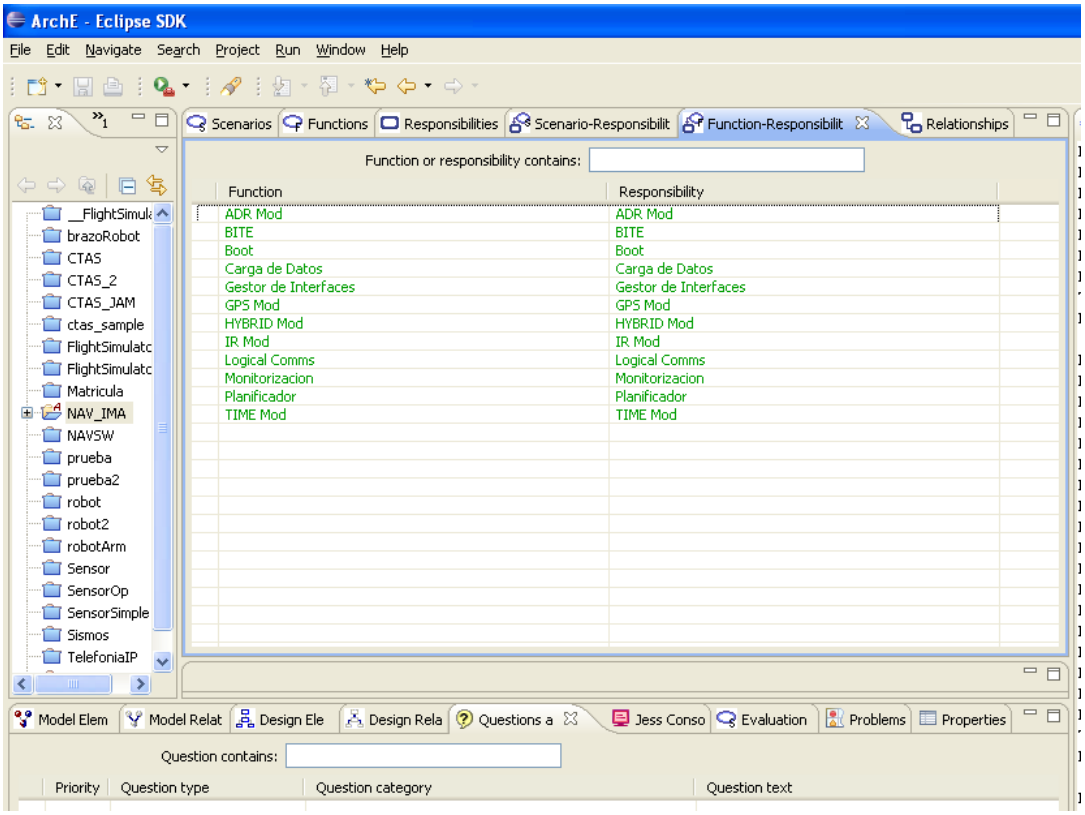


Figura 179. Mapeo Funciones-Responsabilidades de IMA.

Las relaciones quedarían de la siguiente manera:

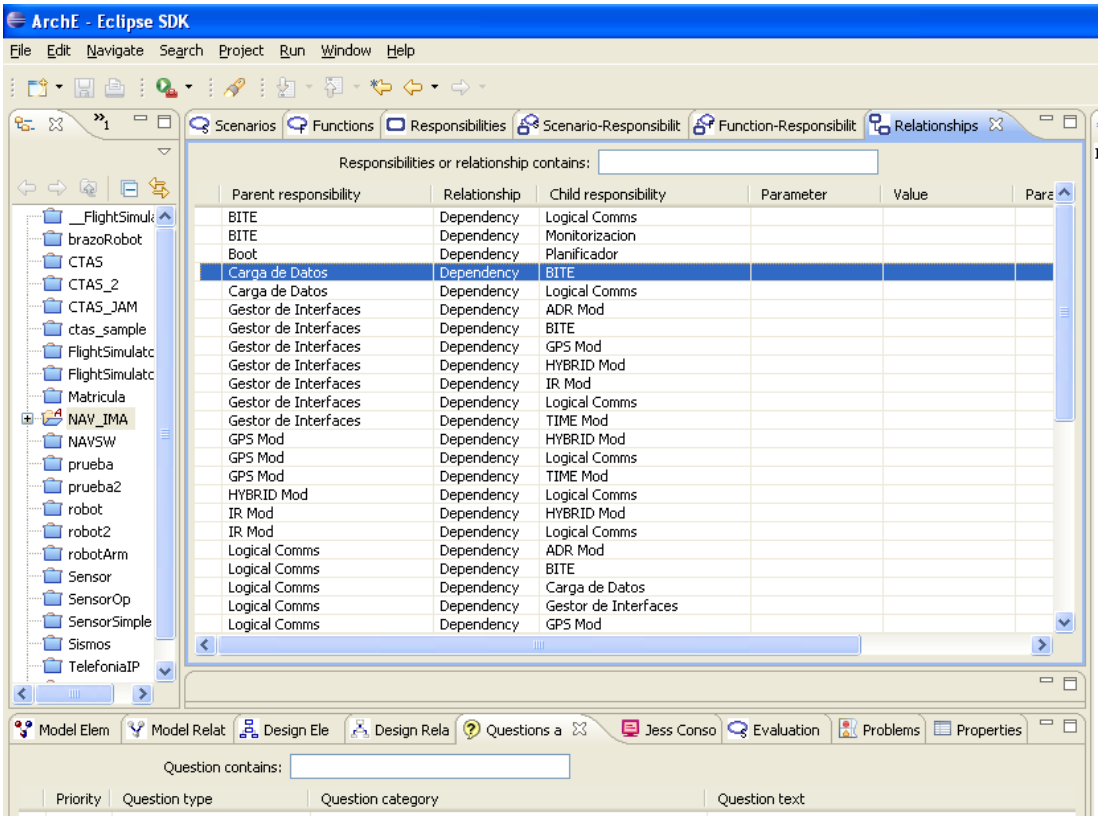


Figura 180. Relaciones de IMA – I.

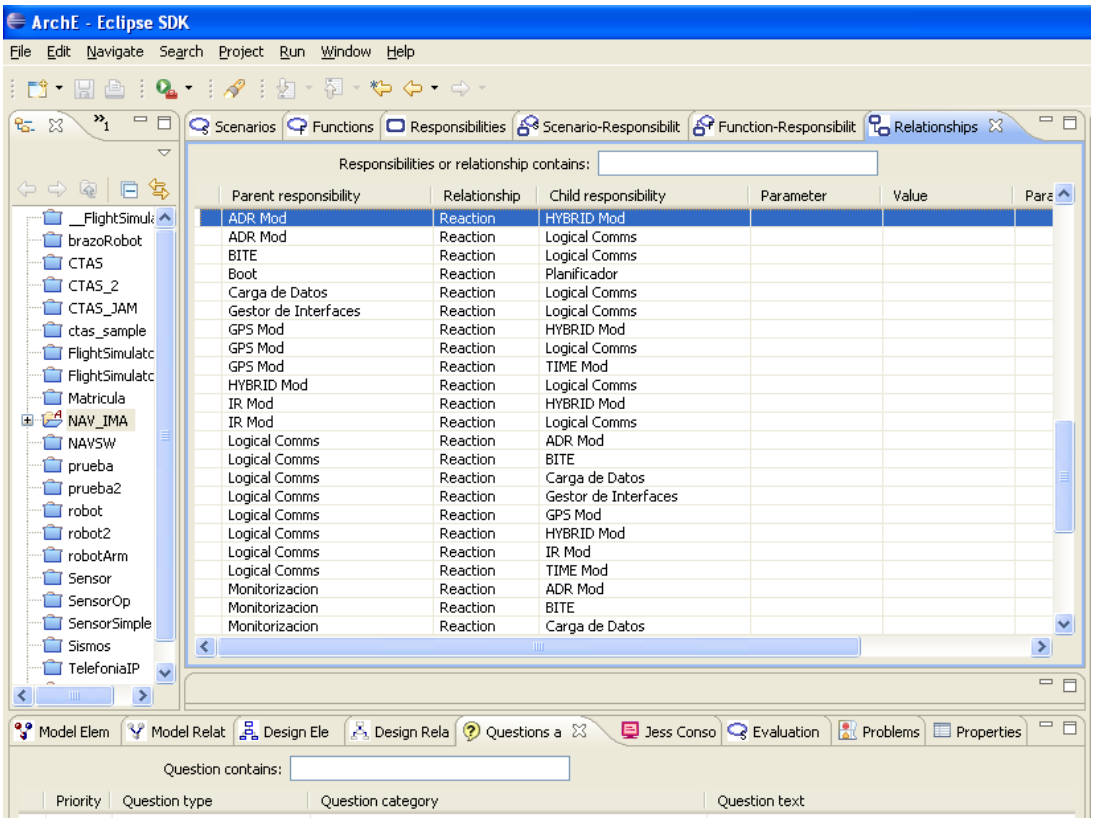


Figura 181. Relaciones de IMA – II.

Se introducen los mismos escenarios del capítulo 7: (Figura 182, Figura 183, Figura 184, Figura 185 y Figura 186)

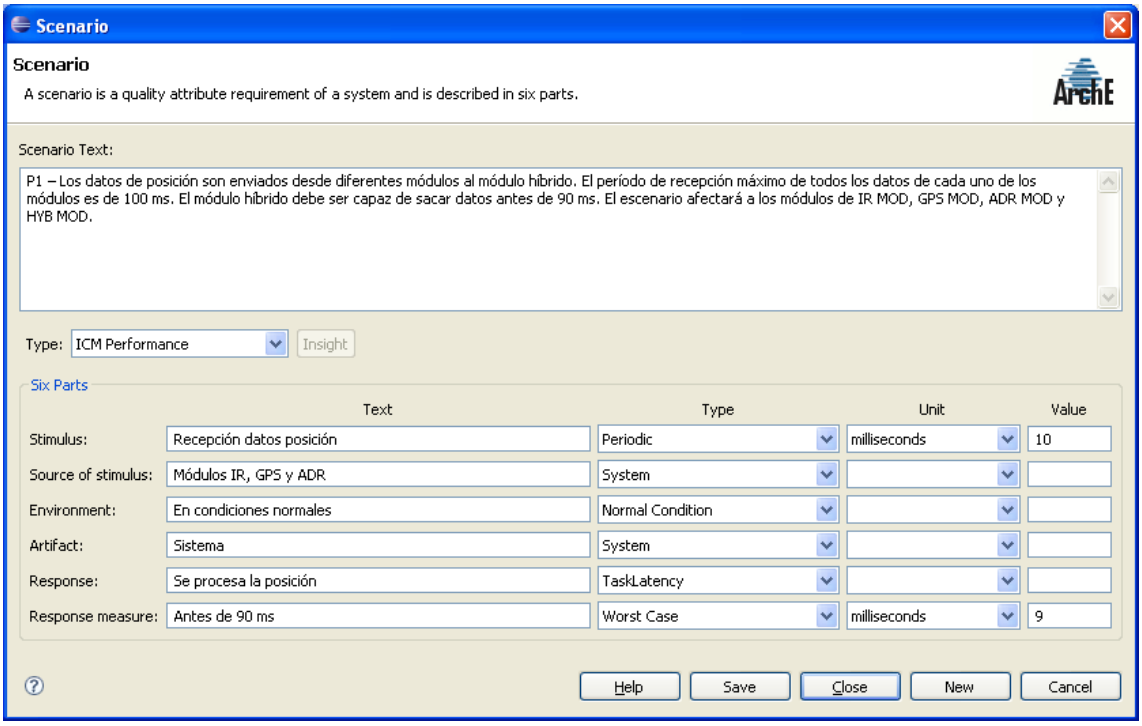


Figura 182. Escenario P1 para IMA.

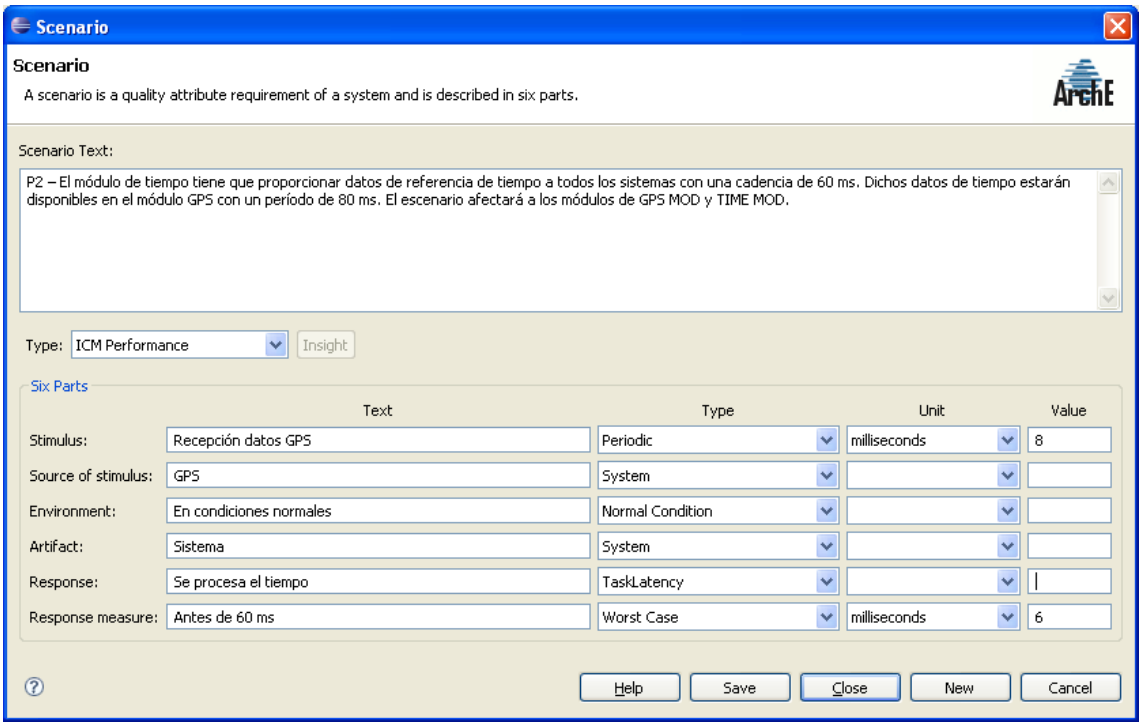


Figura 183 Escenario P2 para IMA.

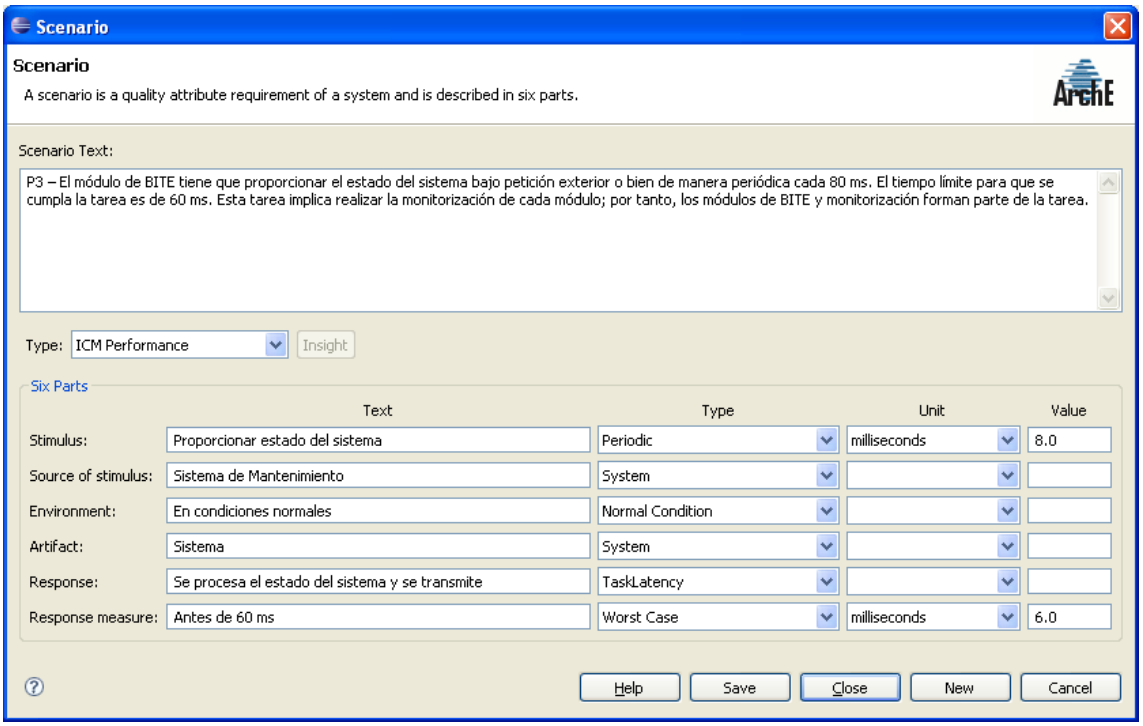


Figura 184. Escenario P3 para IMA.

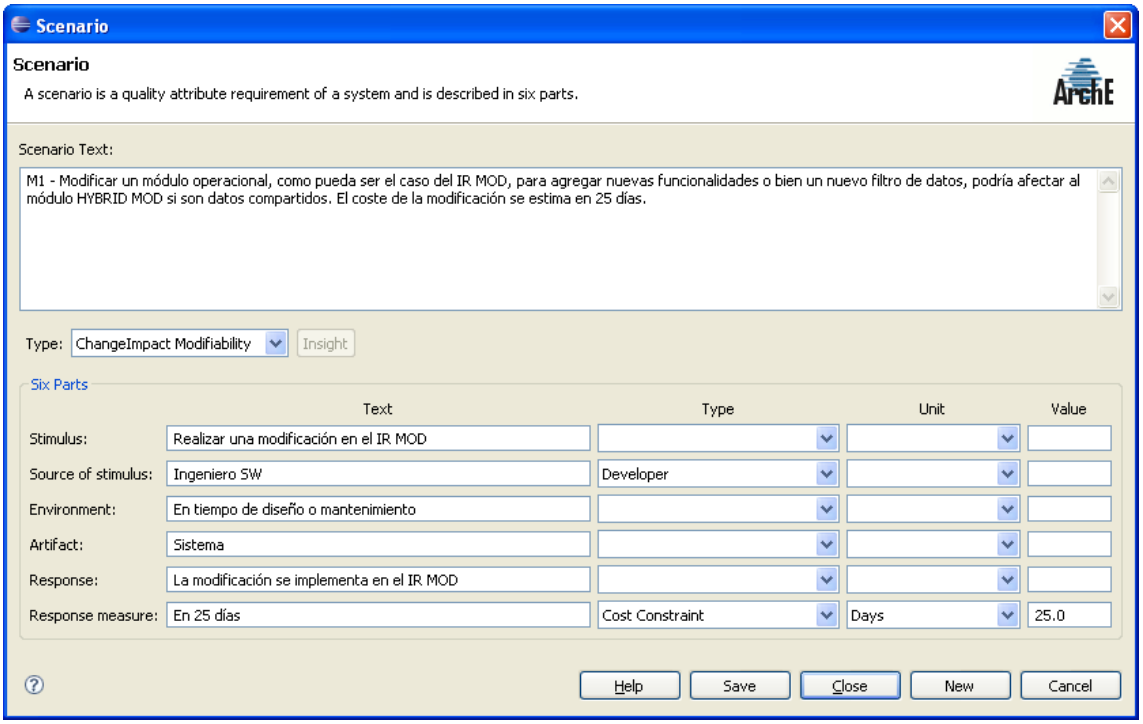


Figura 185. Escenario M1 para IMA.

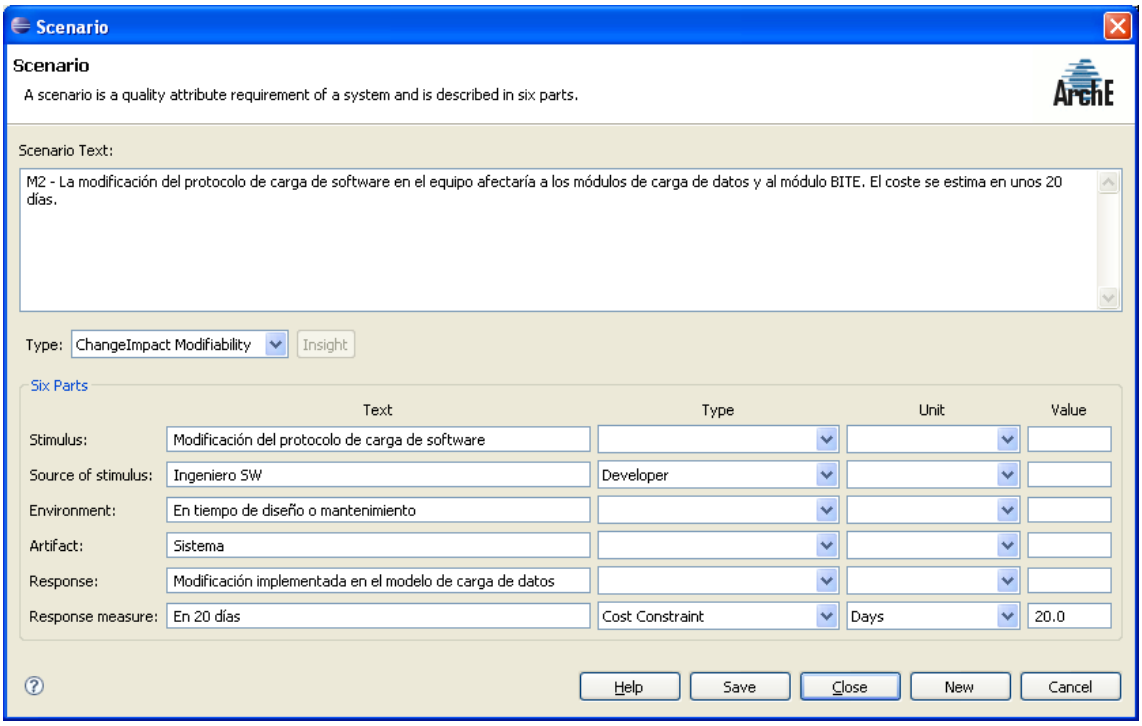


Figura 186. Escenario M2 para IMA.

El mapeo Escenarios-Responsabilidades quedaría así: (Figura 187)

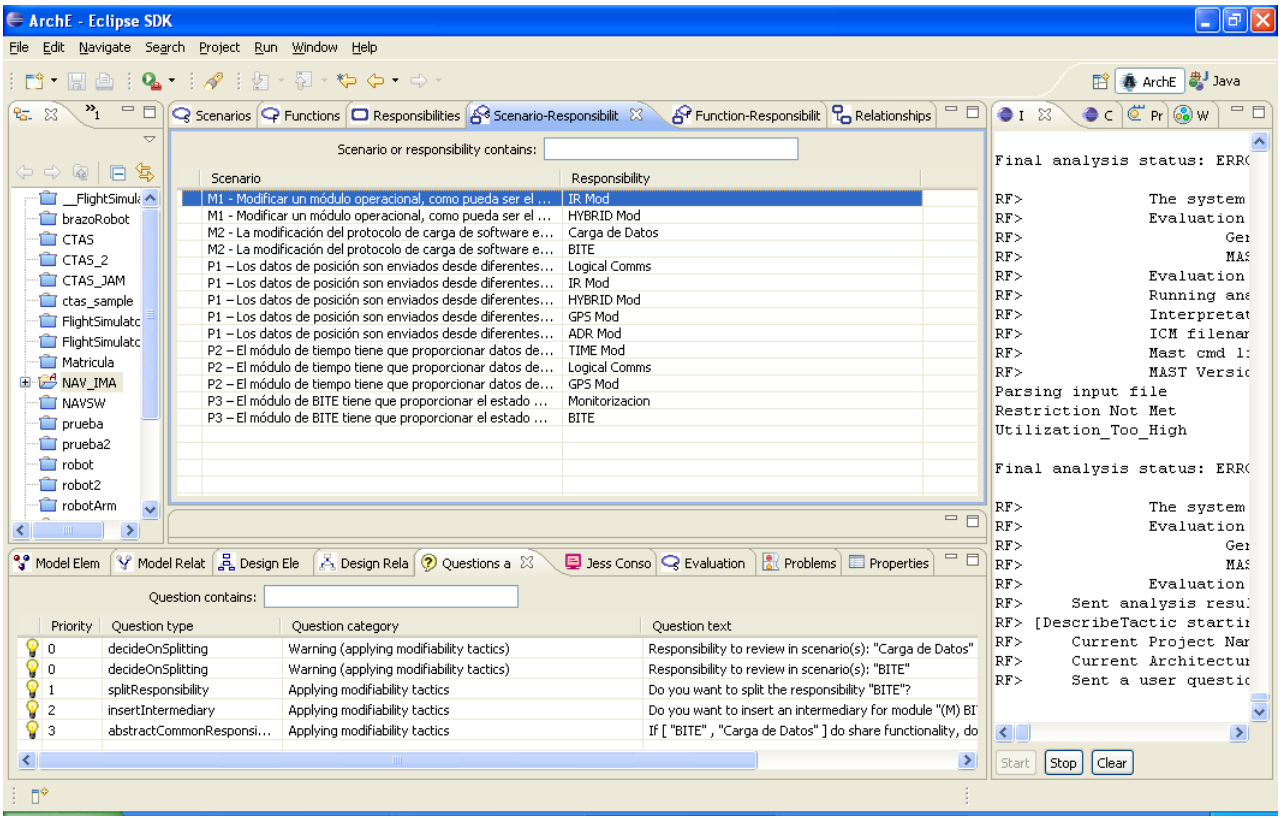


Figura 187. Mapeo Escenarios-Responsabilidades para IMA.

Los marcos de razonamiento de ArchE dan el siguiente resultado en la primera iteración: (Figura 188)

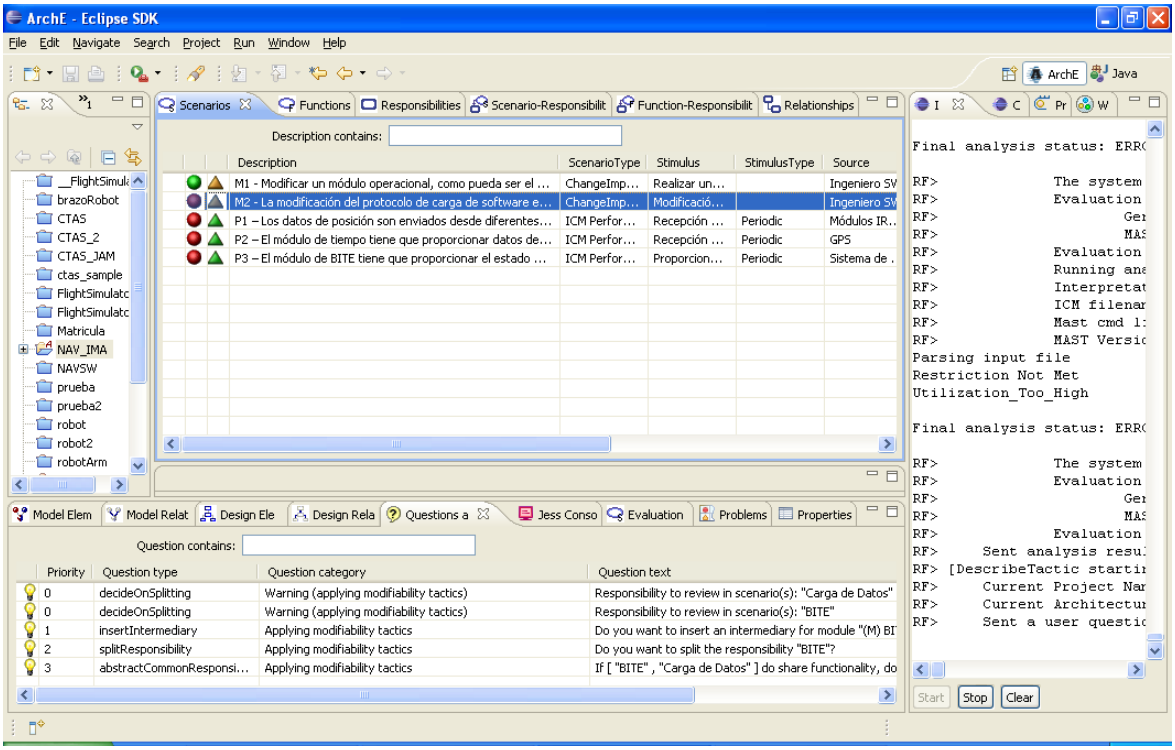


Figura 188. Estado inicial de los escenarios en IMA.

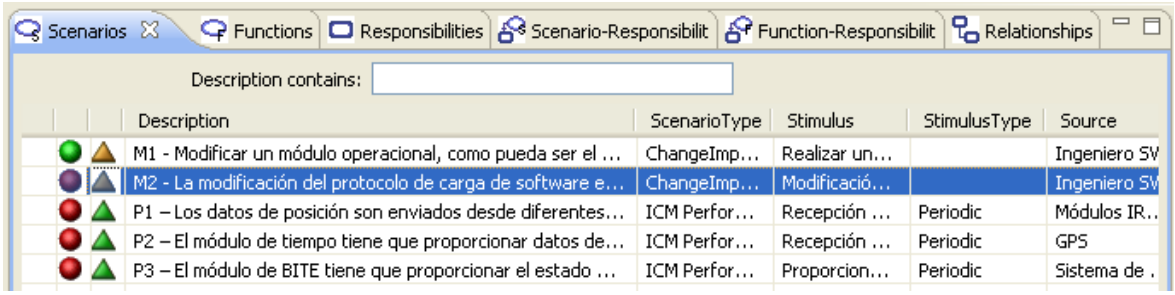


Figura 189. Estado inicial de los escenarios en IMA (ampliado).

Se analizan a continuación los diferentes escenarios:

- **Modifiability.**

Aparéntenme, los escenarios de modificabilidad se analizan de la misma manera que una arquitectura cualquiera: (Figura 190)

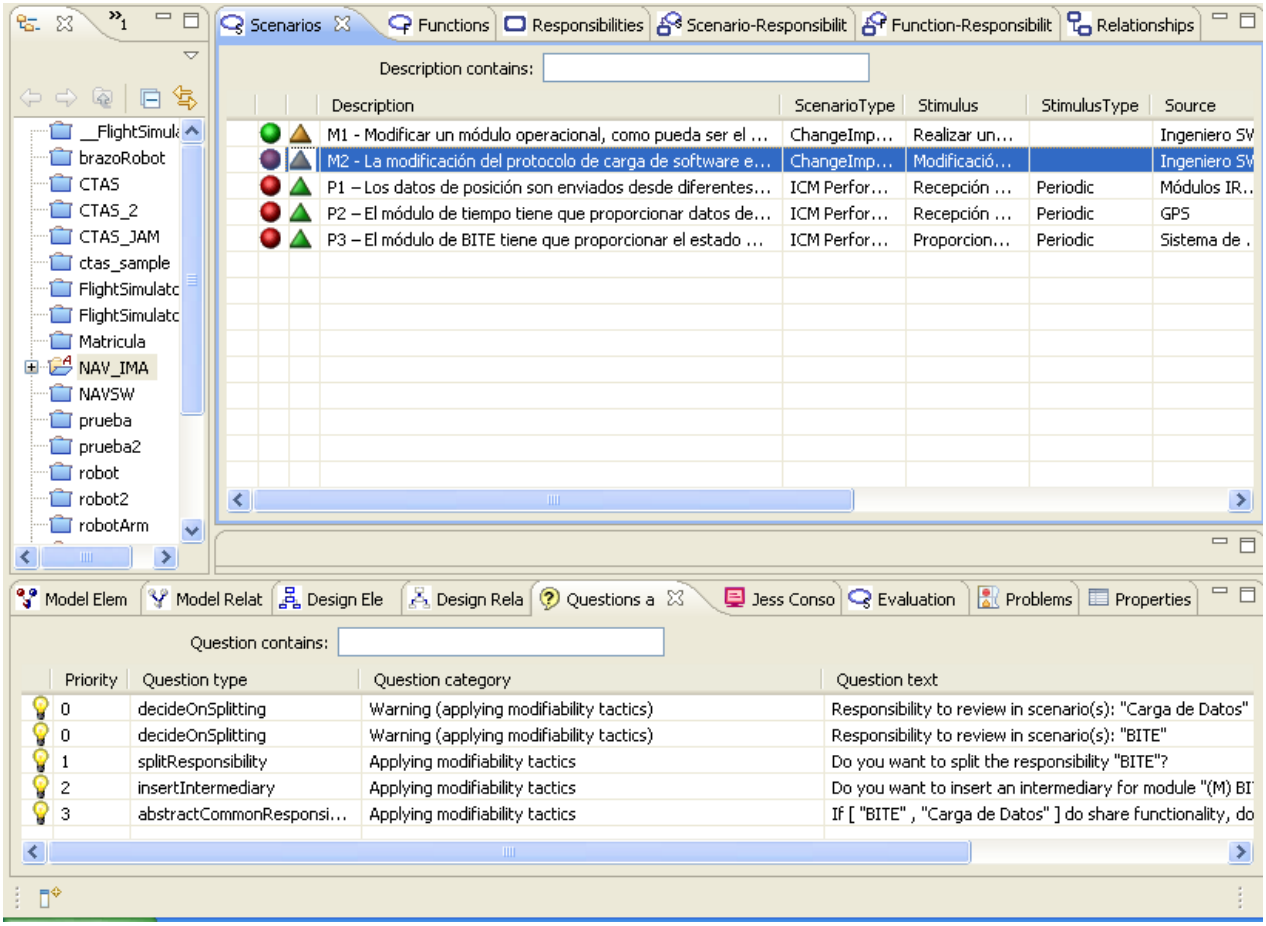


Figura 190. Escenarios y Tácticas de IMA.

ArchE sugiere tácticas para intentar que se cumpla M2. Sin embargo, si nos fijamos en la arquitectura de dependencias que sugiere ArchE para el sistema (Figura 191), lo que viene a ser el equivalente al diagrama UML, se puede observar que la dependencia entre módulos (o ya en este caso se puede hablar de particiones) que intercambian datos o llamadas a procedimientos no está bien modelada: fijémonos por ejemplo en los módulos IR Mod y HYBRID Mod; ambos intercambian datos, por lo tanto hay una dependencia entre ellos; pero la arquitectura IMA no recoge esta dependencia, ya que la llamada a procedimientos o el intercambio de datos se hace a través de *Logical Comms* (a través de la línea verde que los conecta). Sin embargo, ArchE modela una dependencia entre dichos módulos de manera directa (a través de la línea roja) y además con *Logical Comms*, lo cual entra en contradicción.

Además, la modificabilidad no se puede aplicar a los módulos que forman parte del kernel, es decir, a *Logical comms*, Planificador, Monitorización y *Boot*. ArchE no puede entender esto porque no está informado de ello, y con la configuración actual no se puede indicar. Es decir, el actual marco de razonamiento de *Modifiability* entendería que, si hay un cambio en IR Mod,

que se comunica con HYBRID Mod a través de *Logical Comms*, el cambio se propagaría a través de toda la ruta de comunicación, incluido el módulo *Logical Comm*, cosa que en IMA no es cierta porque no se puede producir.

En una arquitectura IMA real, si por ejemplo se modificara IR Mod, podría desde luego tener impacto en otras particiones que intercambiaran datos con dicho módulo, pero:

- La arquitectura en particiones se hace para conseguir la máxima independencia entre módulos. Para ello se introdujo el concepto de interfaz APEX en el capítulo 6.
- Si hay alguna modificación en algún módulo, por ejemplo en alguna variable interna o procedimiento, el cambio se propagará a las particiones que utilicen dicha variable o procedimiento, pero la interfaz APEX a través de la cual la partición se comunica, y el módulo kernel que realiza la función de puerta de enlace entre particiones no se modificarán. Esto no se refleja en el diseño UML de la arquitectura, por lo tanto no son tenidas en cuenta por ArchE.

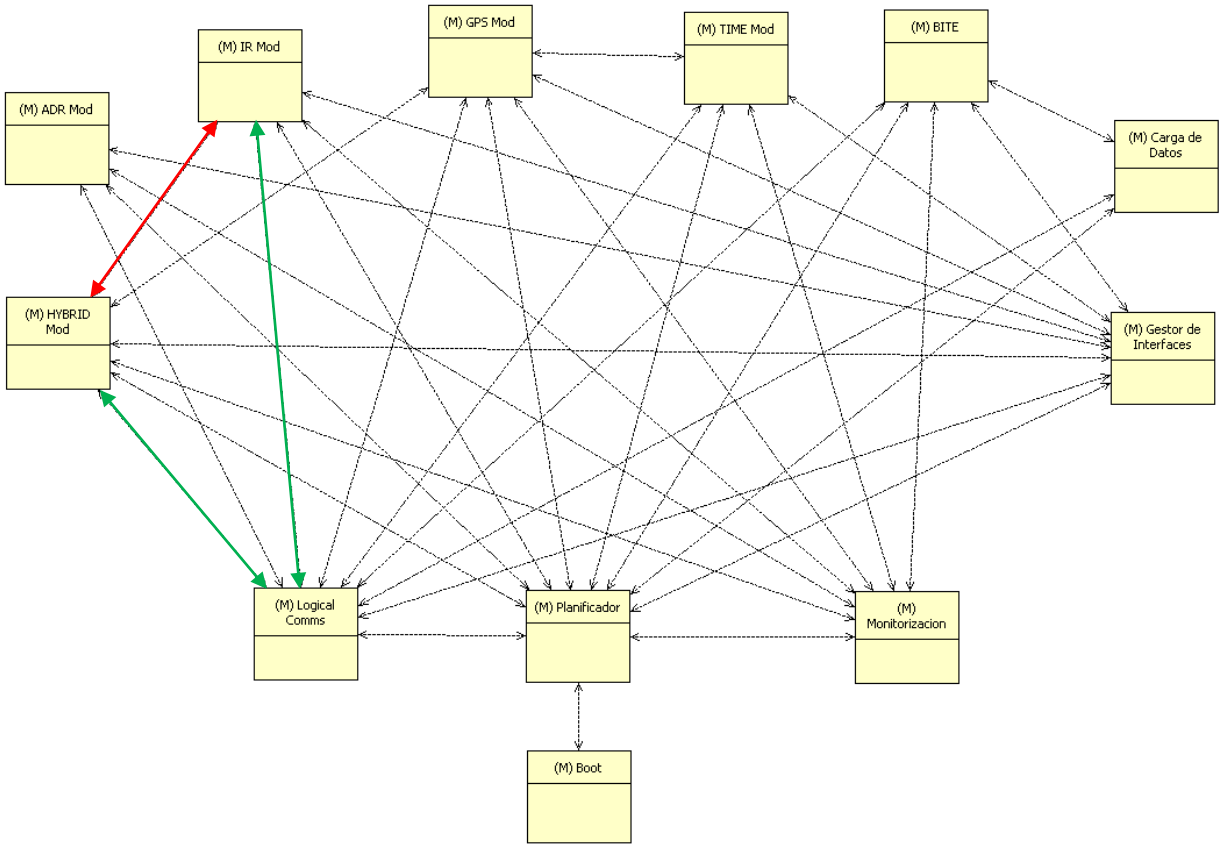


Figura 191. Arquitectura IMA generada por ArchE.

Por estos motivos, aunque ArchE sea capaz de analizar dependencias y modificabilidad, realmente no es capaz de entender las peculiaridades de la arquitectura IMA.

- **ICM Performance**

Respecto al marco de rendimiento, la primera iteración muestra que el sistema no es planificable: (Figura 192)

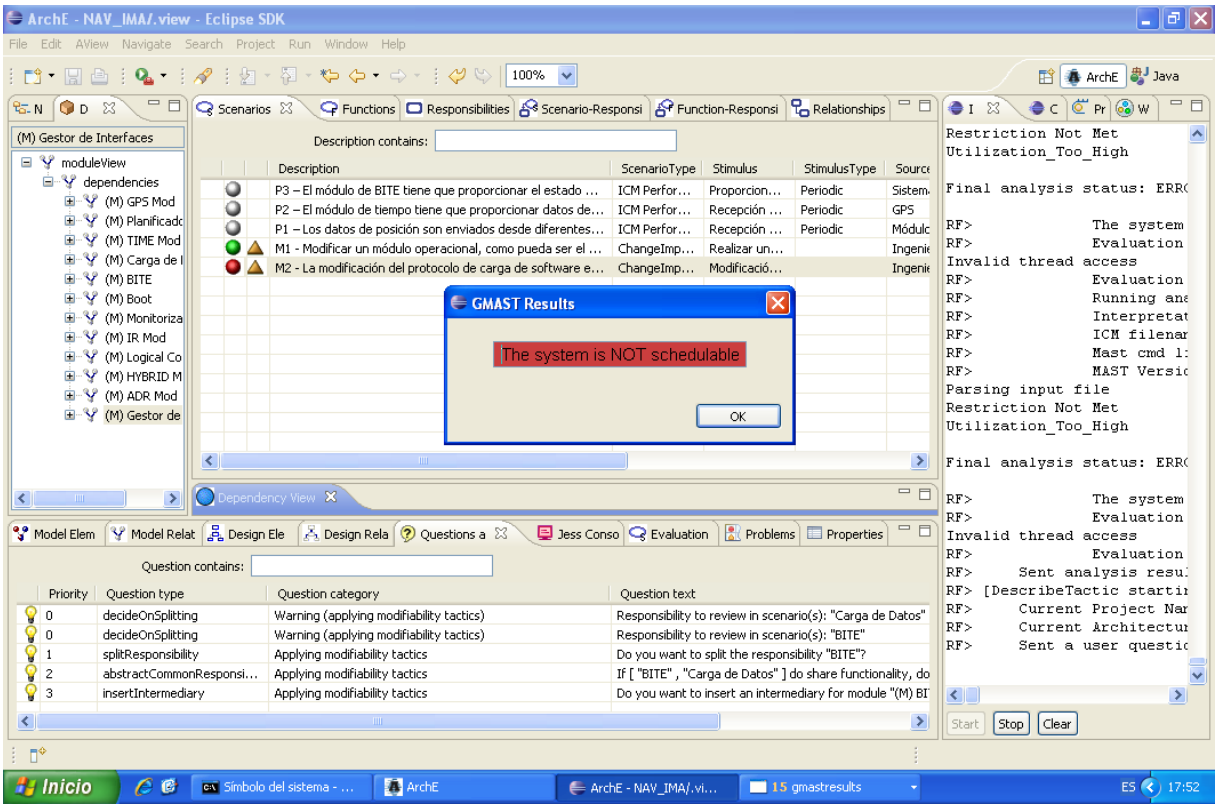


Figura 192. Primera iteración de ICM Performance para IMA.

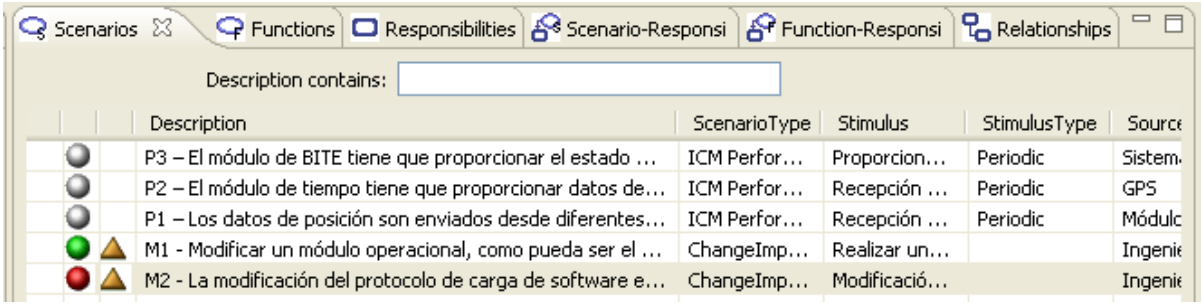


Figura 193. Primera iteración de ICM Performance para IMA – Escenarios.

El problema parece estar en P3: (Figura 194)

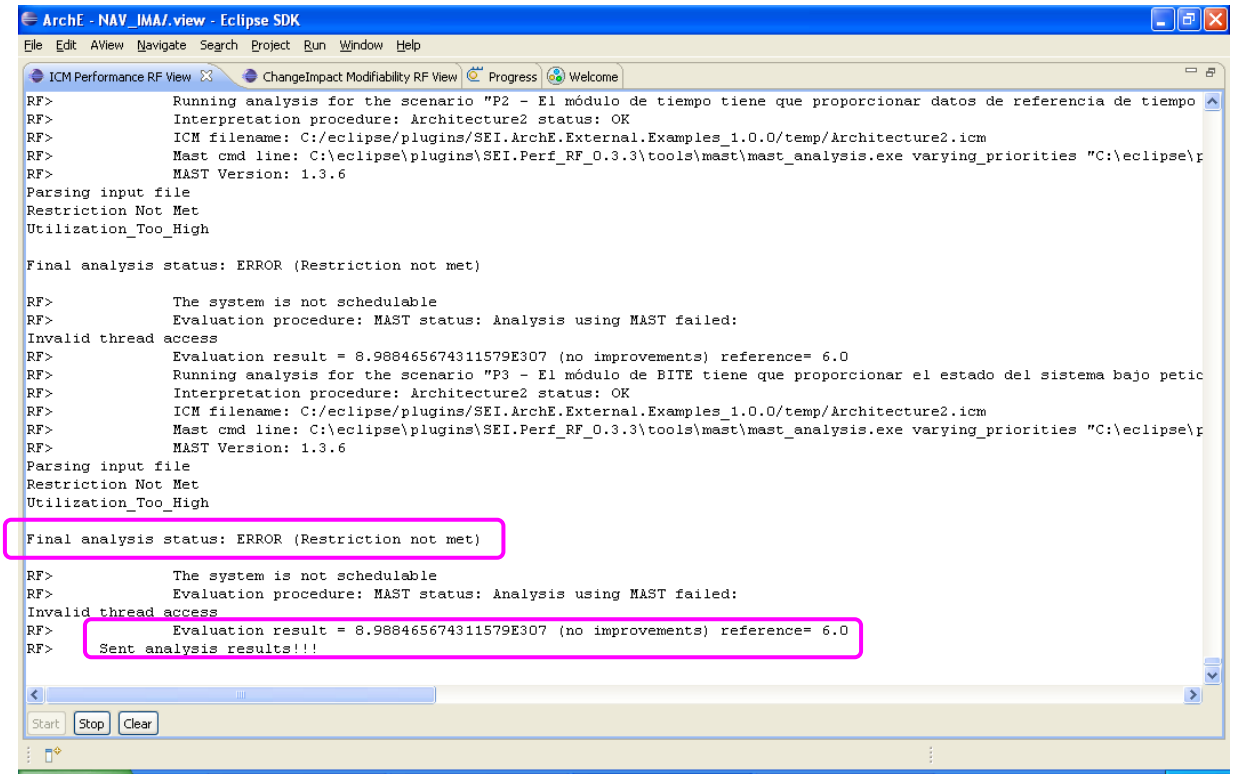


Figura 194. Ventana de ICM Performance señala a P3.

Se modifican ligeramente el período y la deadline de P3: (Figura 195)

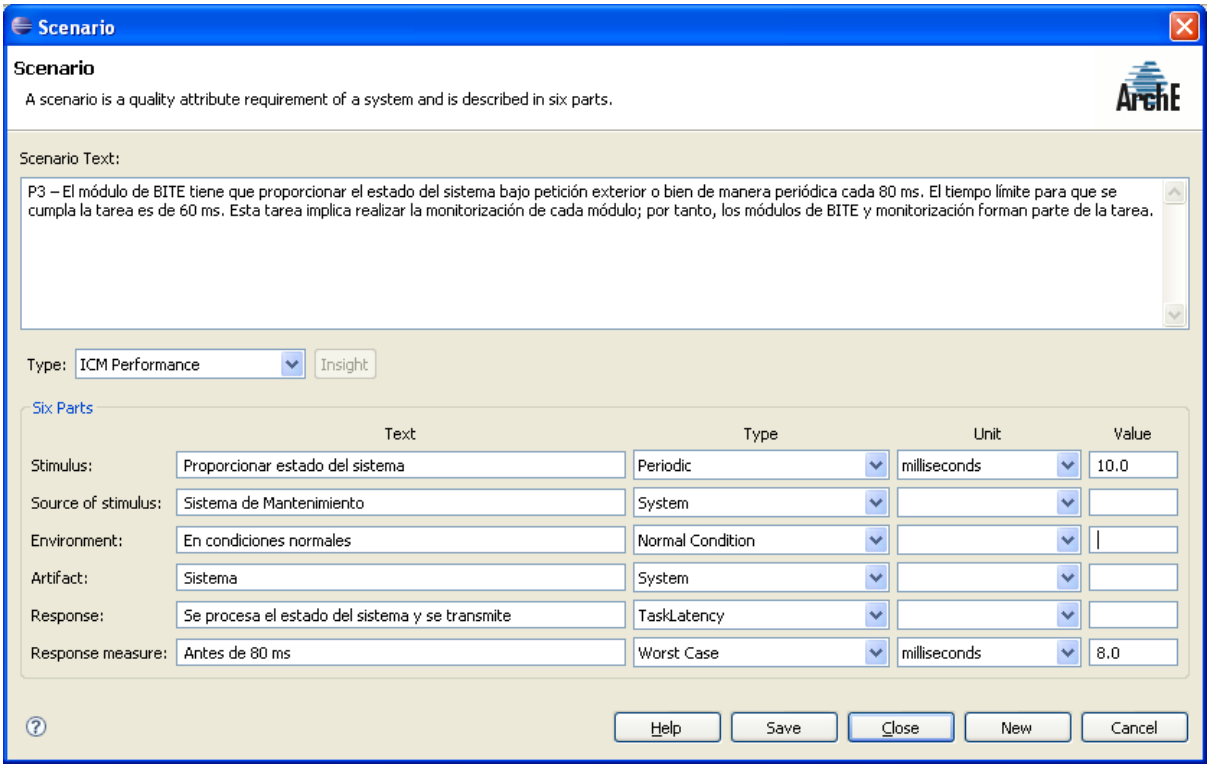


Figura 195. Modificación del escenario P3.

Pero después, el no cumplimiento de restricciones se extiende a los de más escenarios; después de modificar todas las *deadlines*, se obtiene un resultado sorprendente: a pesar de cumplir las restricciones, MAST declara al sistema como no planificable: (Figura 196)

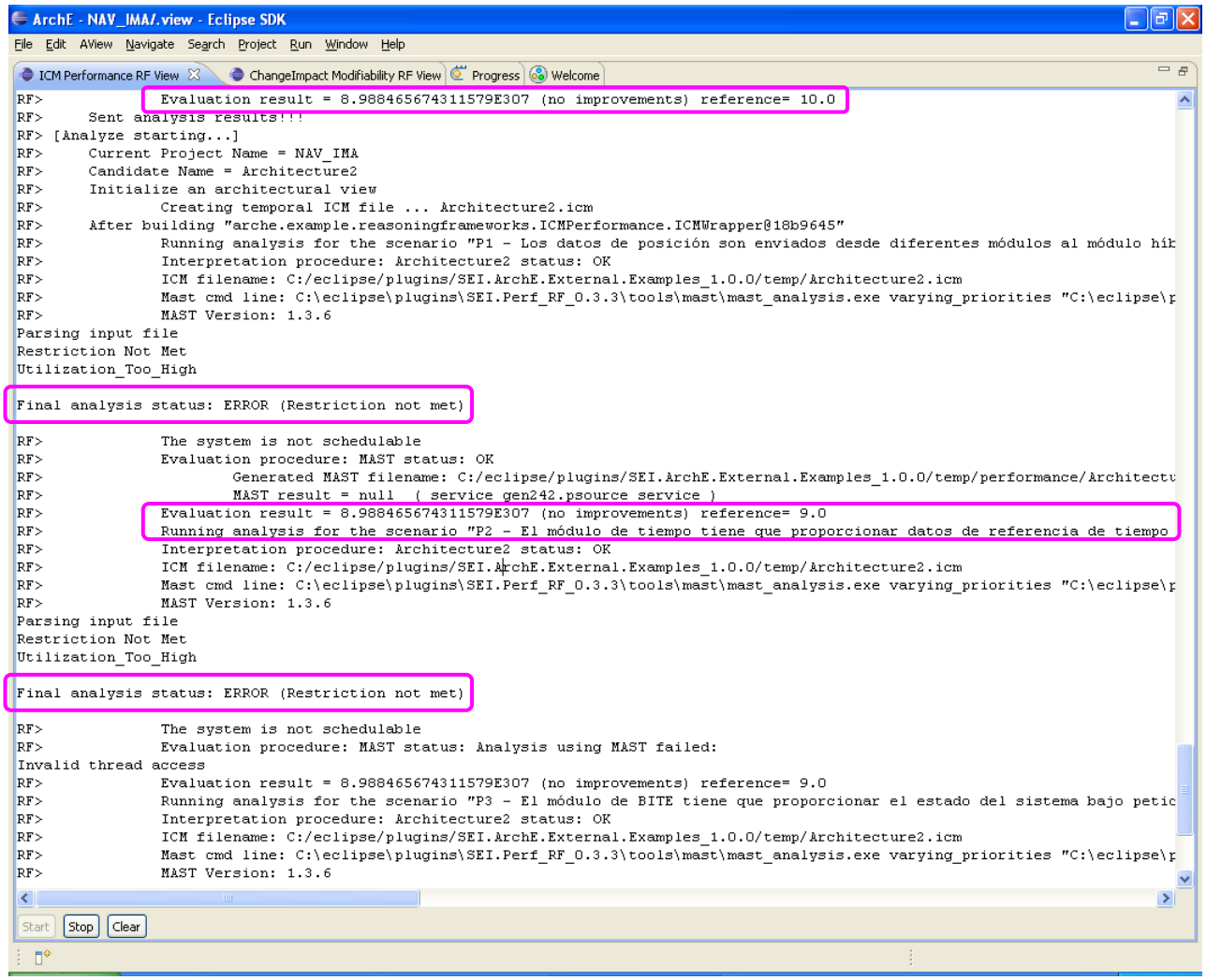


Figura 196. Ventana ICM Performance con resultados contradictorios.

Después de relajar todos los escenarios, se empiezan a obtener resultados más optimistas: (Figura 197)

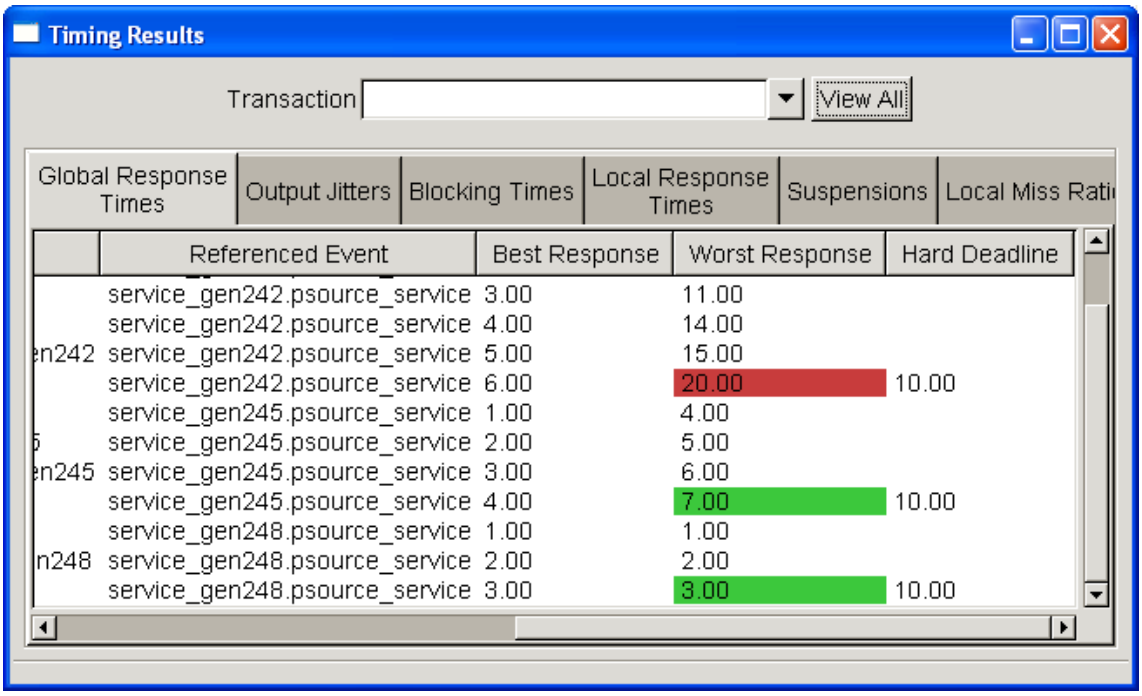


Figura 197. Ventana MAST muestra que P1 no cumple su deadline.

Si se relaja el escenario P1, se obtiene: (Figura 198)

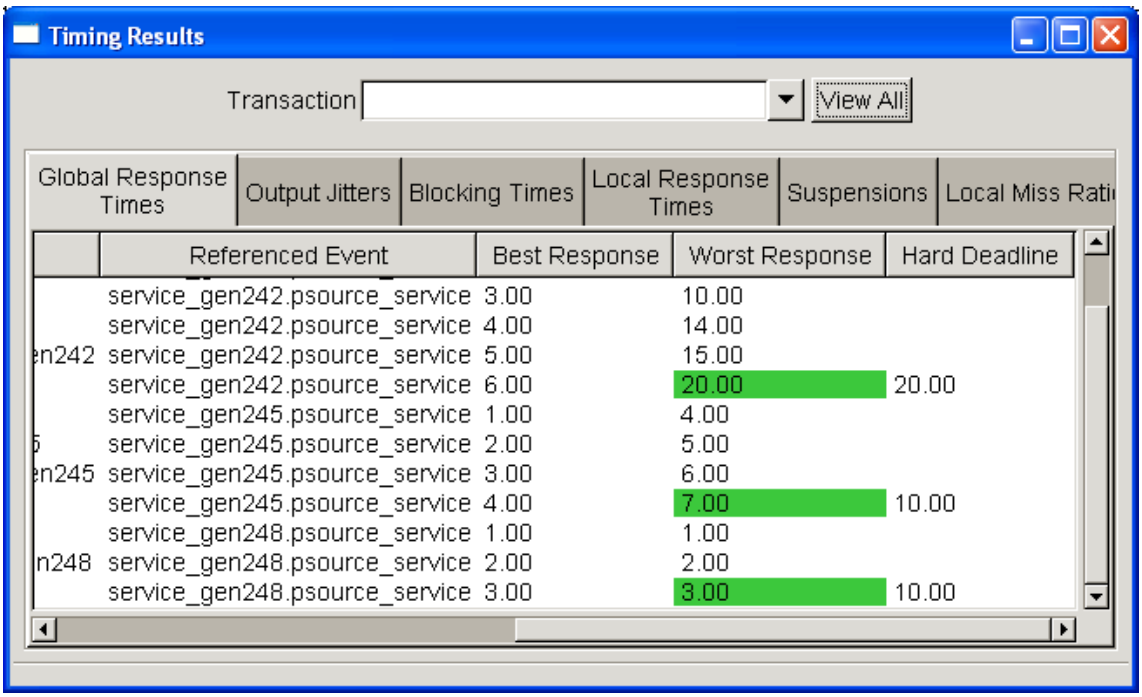


Figura 198. Ventana MAST muestra que todos los escenarios cumples su deadline

Pero por otro lado, MAST arroja resultados contradictorios; también sigue diciendo que el sistema no es planificable: (Figura 199 y Figura 200)

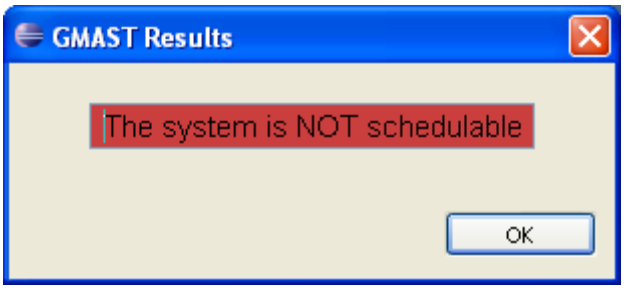


Figura 199. Resultado de MAST – Sistema no planificable.

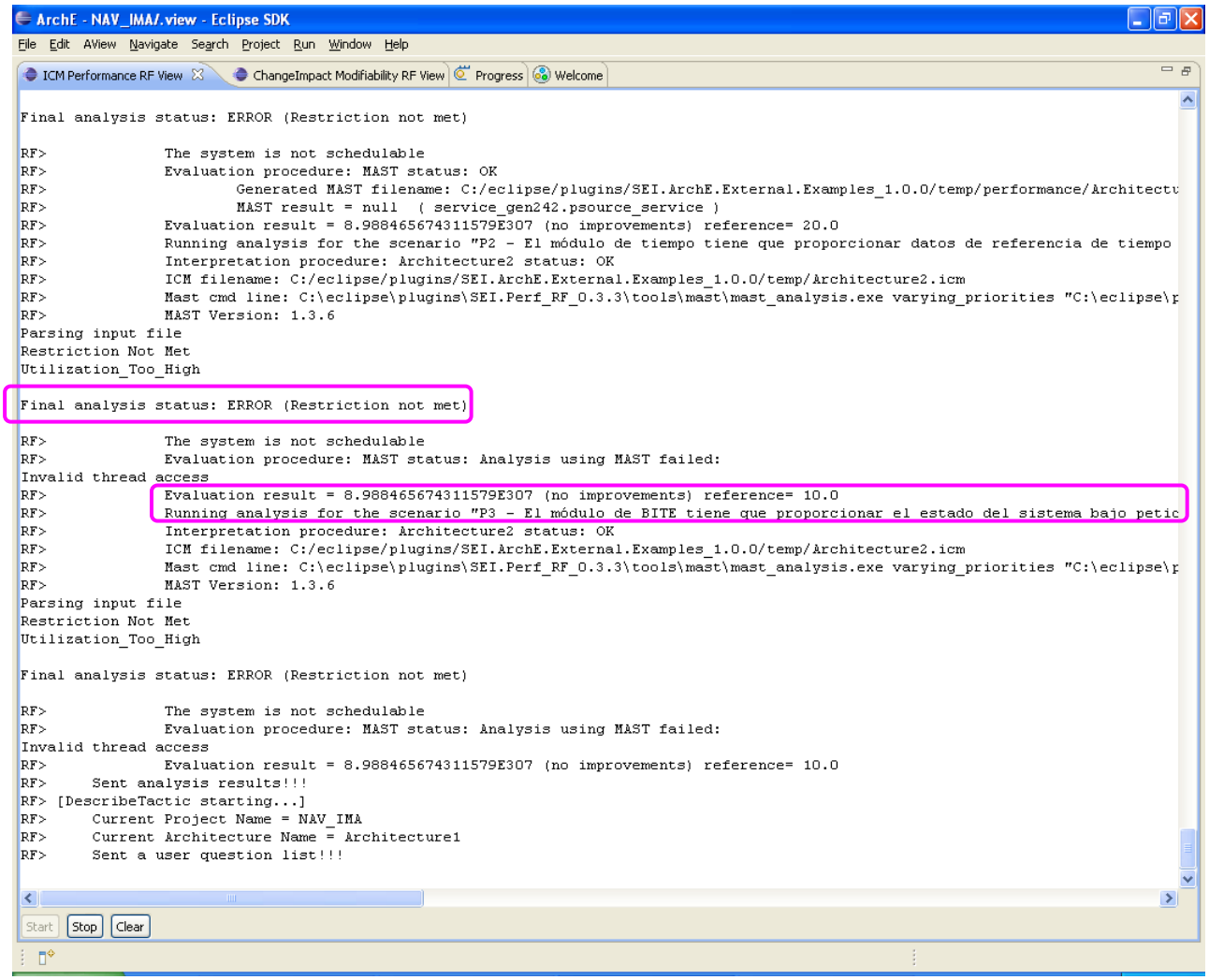


Figura 200. Ventana ICM Performance muestra que el sistema no es planificable.

Si se ajustan los tiempos de ejecución de las responsabilidades al valor del ejemplo de sistema de aviónica no IMA, se obtiene de nuevo que el sistema no es planificable: (Figura 201)

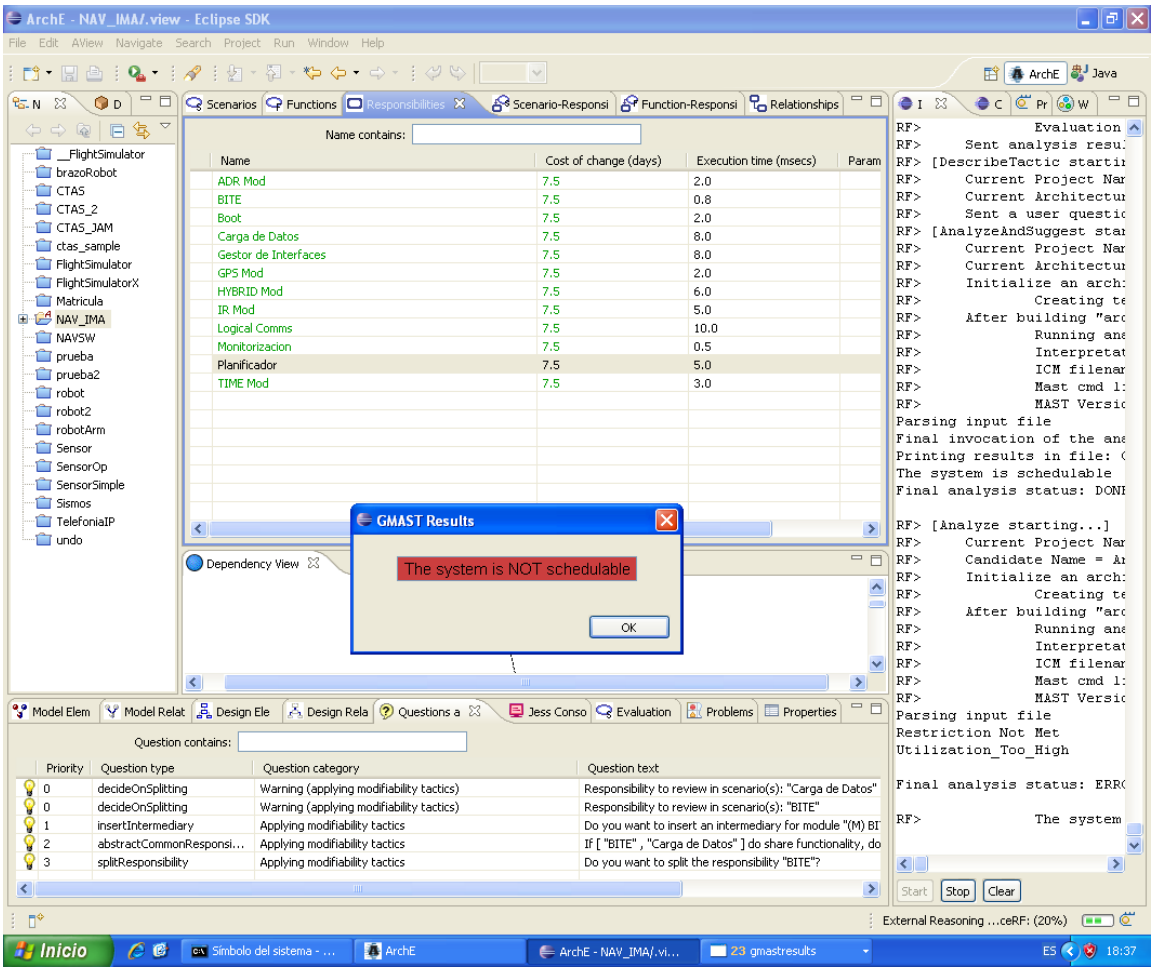


Figura 201. Tiempos de ejecución de las responsabilidades de IMA ajustados.

Sin embargo, las transacciones aparecen como cumplidas: (Figura 202)

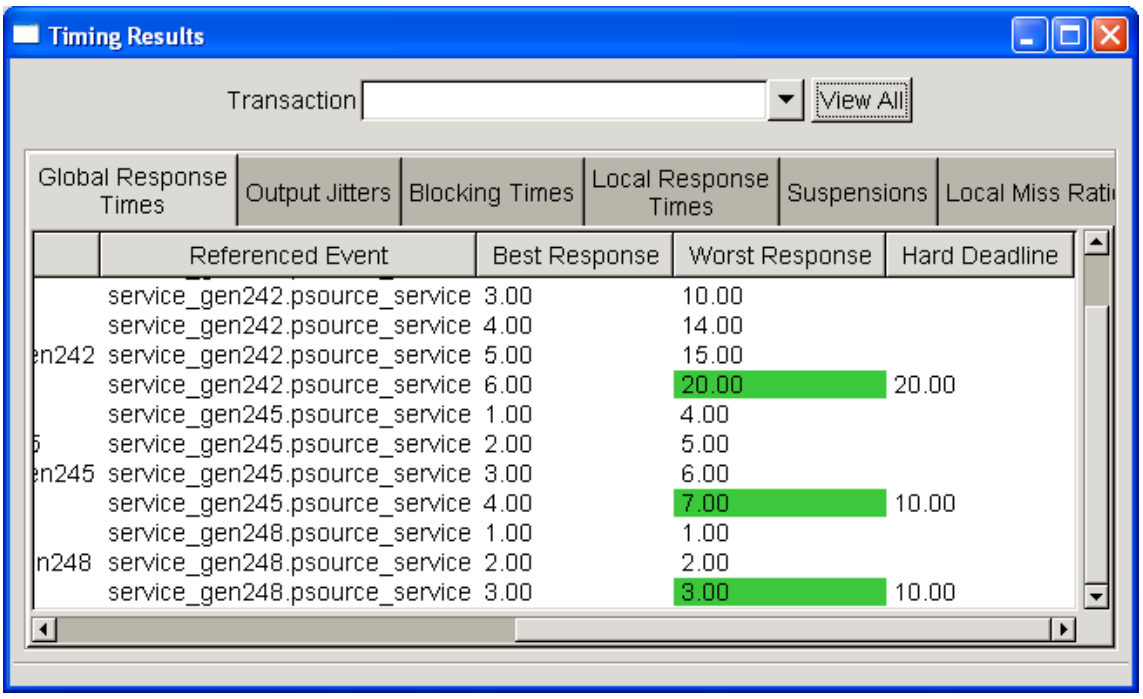


Figura 202. Ventana MAST muestra que los escenarios cumplen sus deadlines.

Pero la ventana de ICM Performance sigue insistiendo en que el sistema no es planificable: (Figura 203)

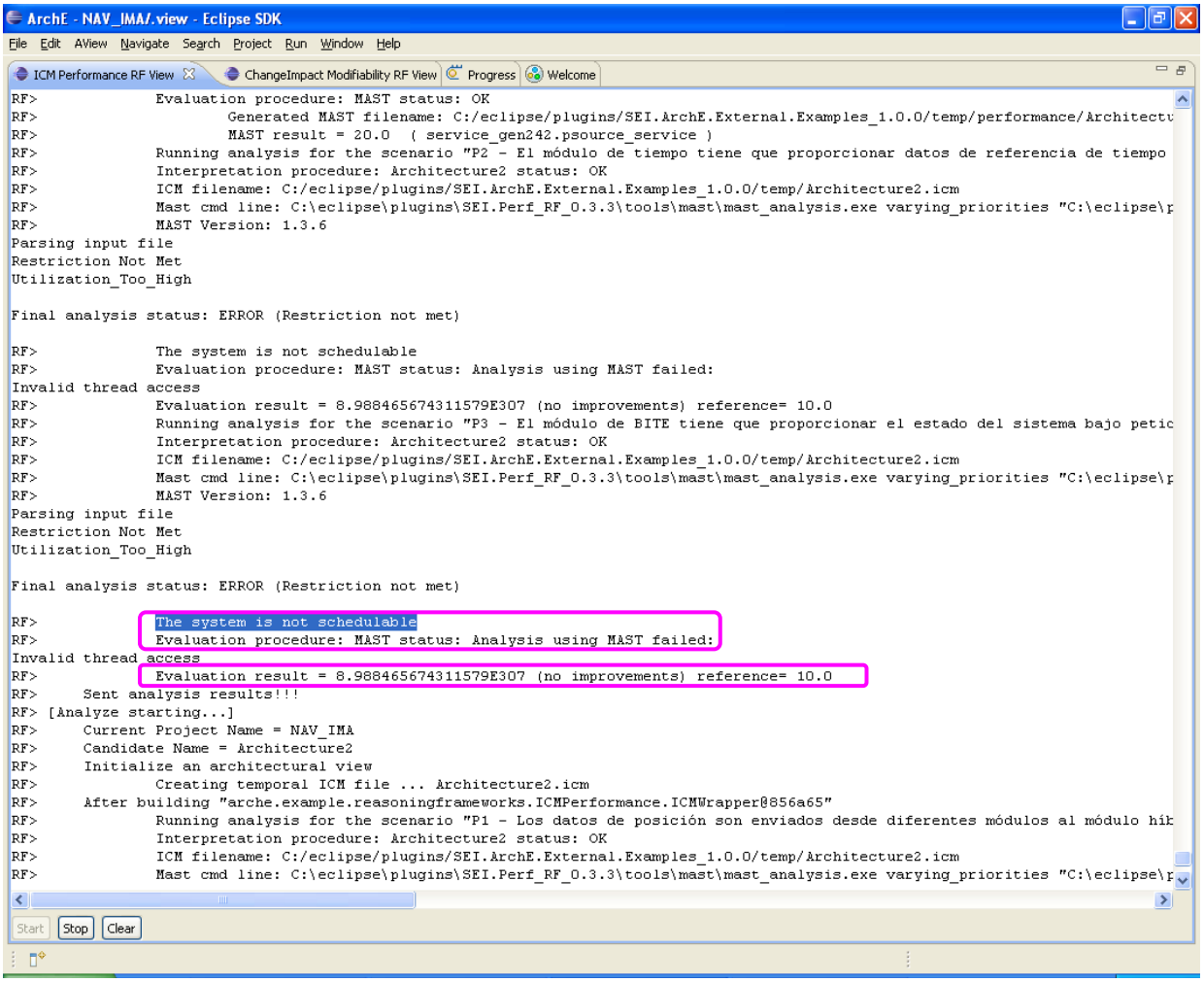


Figura 203. Ventana ICM Performance muestra como contradicción que el sistema IMA no es planificable.

Esta aparente contradicción, que podría ser generada por la inclusión en la nueva arquitectura del elemento del kernel *Logic Comms*, en realidad tiene otro problema: si se observa la ventana del ICM Performance, uno de los avisos que da es Utilization_Too_High. Observando el período de P2, se ha fijado en 100 ms, justo igual que su deadline; si se aumenta dicho período a 150 ms, se obtiene lo siguiente: (Figura 204 y Figura 205)

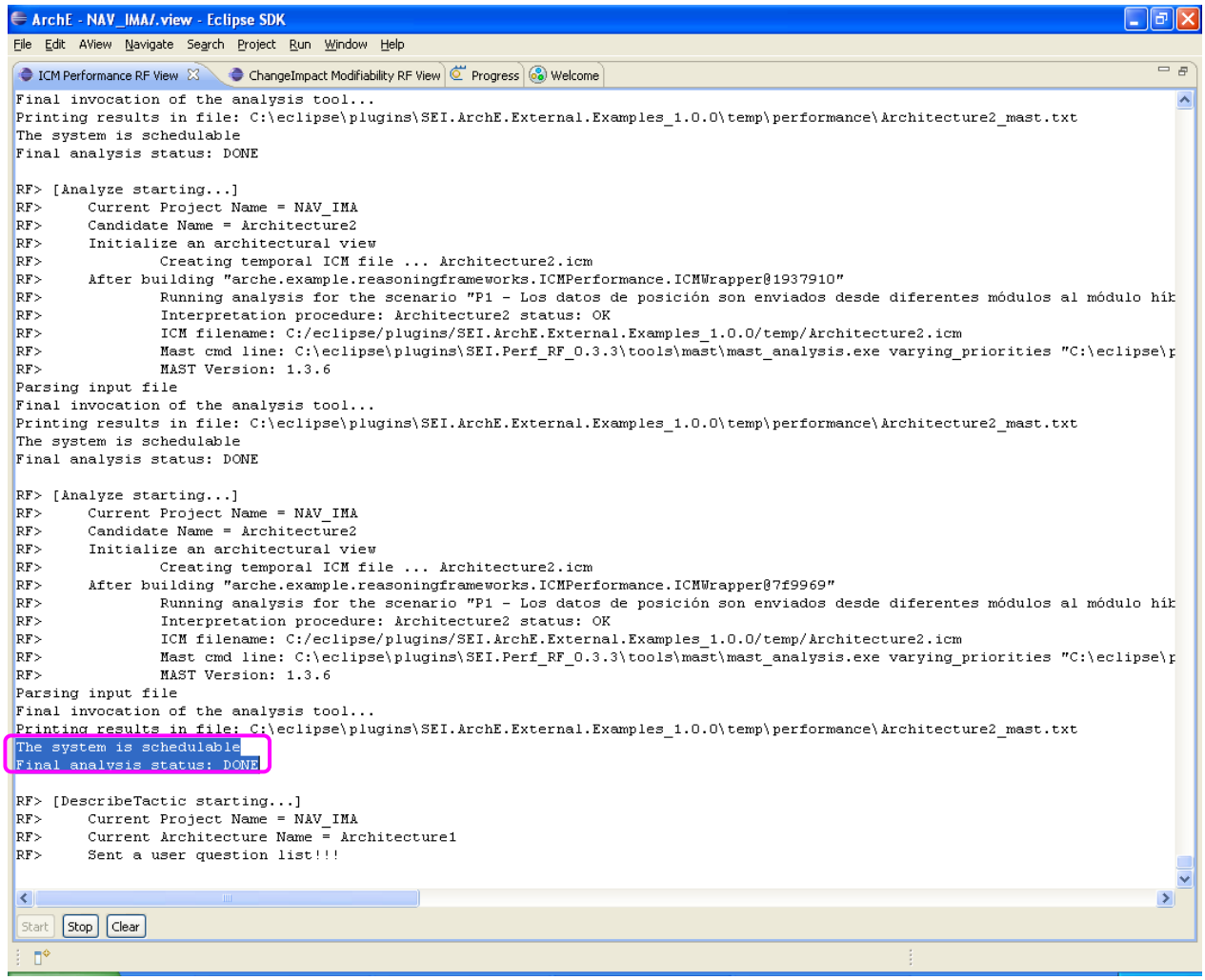


Figura 204. El Sistema IMA es planificable después de ajustar el período de P2.

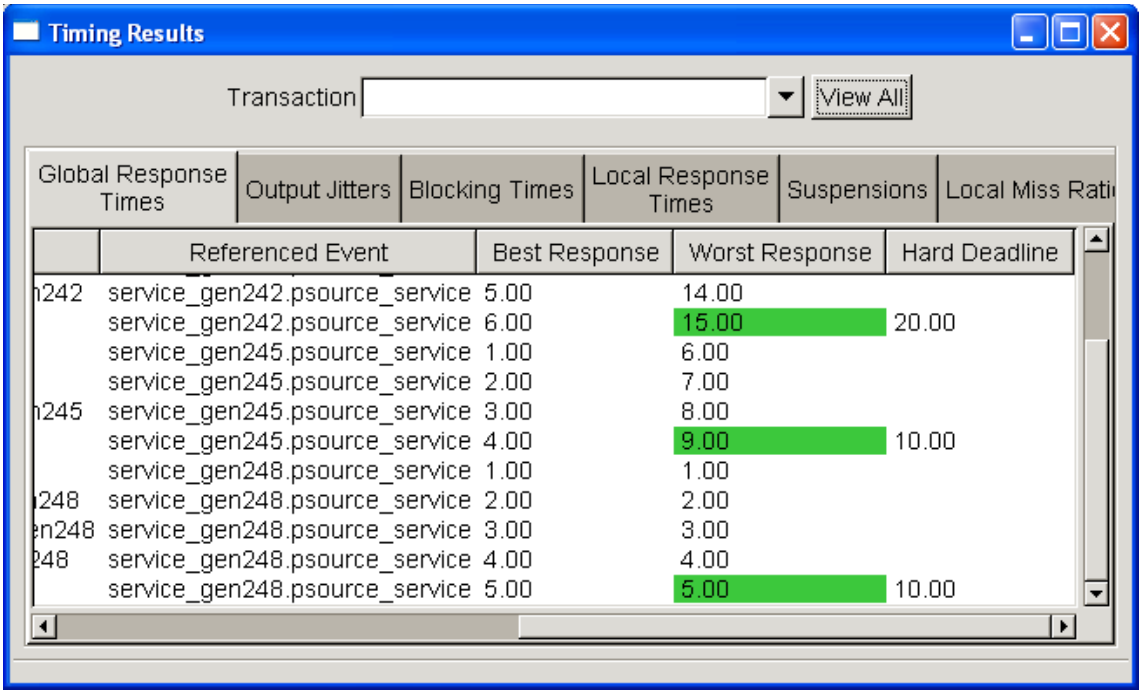


Figura 205. Ventana MAST muestra todos los escenarios cumpliendo su deadline.

El sistema ahora sí es planificable. Hay cierto margen de mejora y se pueden bajar las deadlines: (Figura 206)

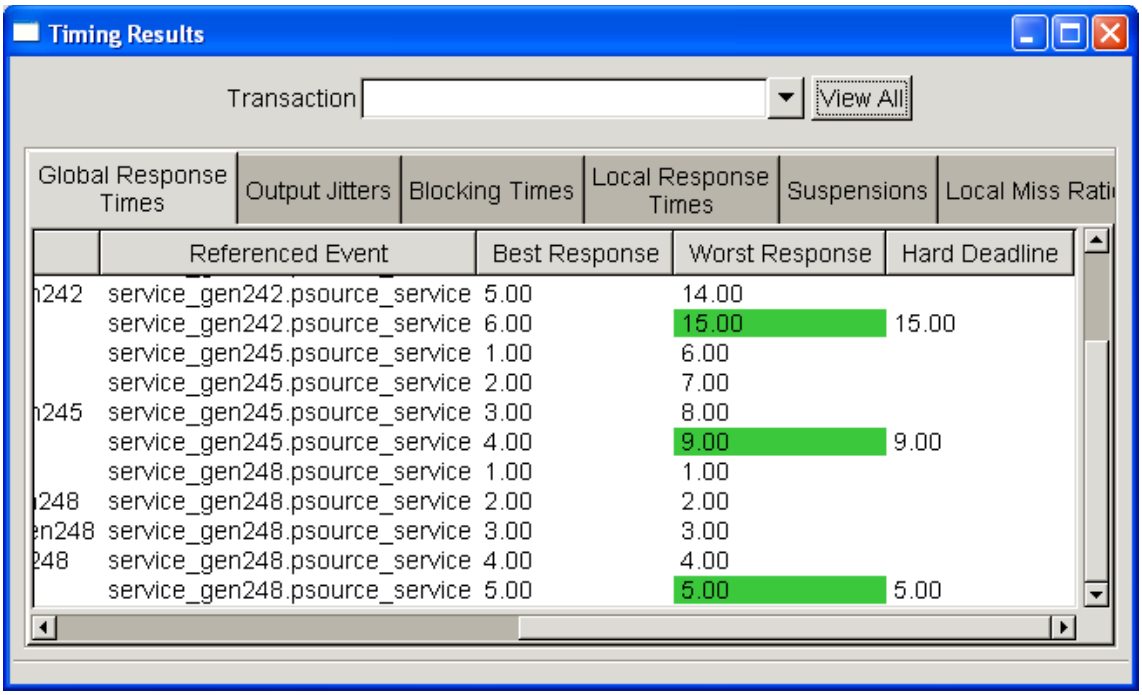


Figura 206. Ventana MAST con las deadlines de los escenarios ajustados al límite.

Y el sistema sigue siendo planificable: (Figura 207)

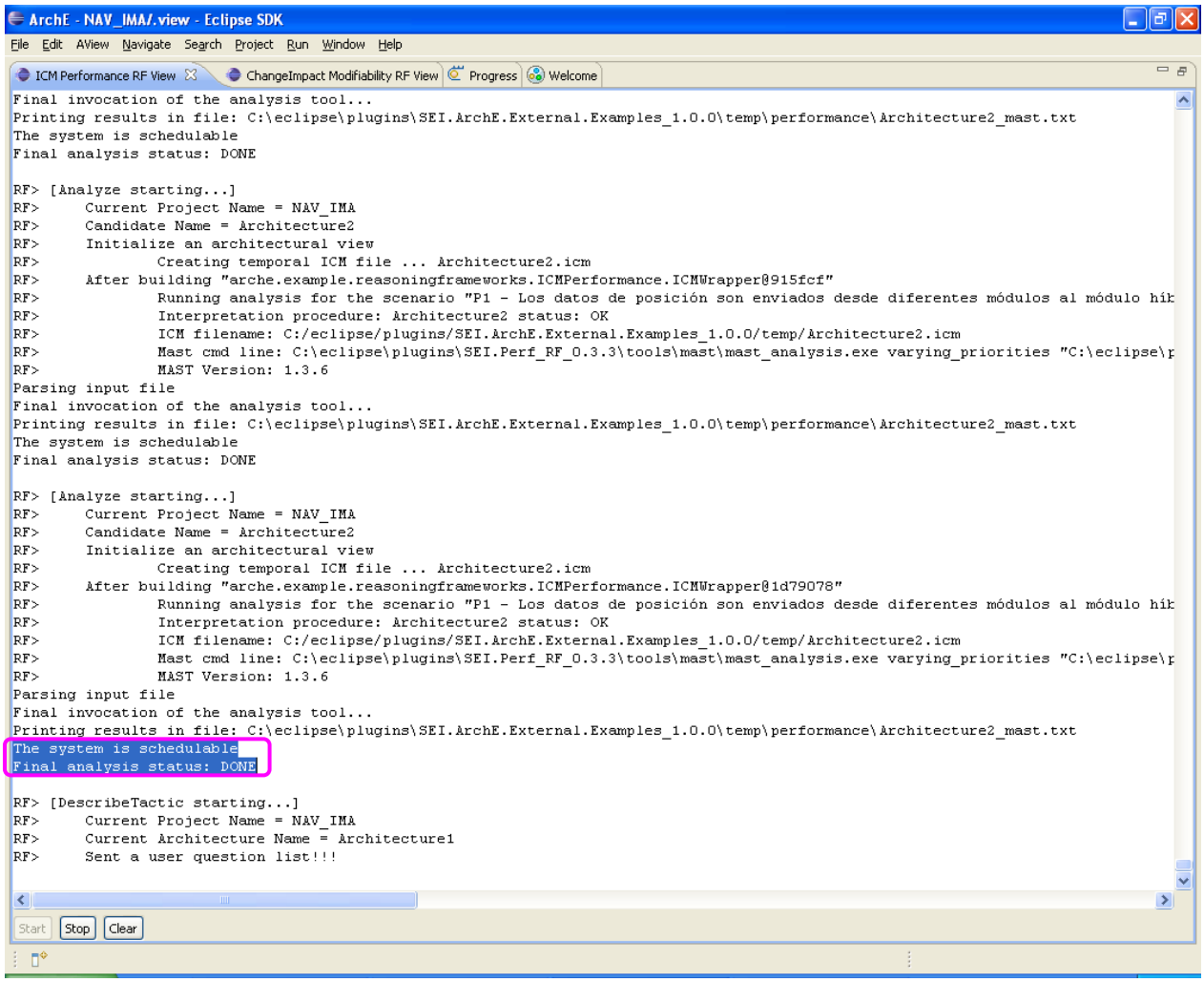


Figura 207. Ventana ICM Performance muestra de nuevo que el sistema IMA es planificable.

Si se comparan los resultados de la arquitectura IMA con los obtenidos por MAST para la arquitectura convencional, se observa lo siguiente: (Tabla 45)

	Arquitectura IMA										Arquitectura convencional	
Responsibility	**Time (ms)**	**it0**	**Status**	**it1**	**Status**	**it2**	**Status**	**it3**	**Status**	**it4**	**Status**	**it_final**	**Status**
Boot	20	20		20		20		20		20			
Planificador	50	50		50		50		50		50			
Carga de Datos	80	80		80		80		80		80			
Gestor de interfaces	80	80		80		80		80		80			
BITE	100	100		100		8		8		8			
Monitorización	70	70		70		5		5		5			
IR MOD	80	80		80		50		50		50			
ADR MOD	80	80		80		20		20		20			
GPS MOD	80	80		80		20		20		20			
HYBRID MOD	90	90		90		60		60		60			
TIME MOD	50	50		50		30		30		30			
P1 - Period	100	100		250		250		250		250		170	
P1 - Response Measure	90	90	OK	200	OK	200	OK	200	OK	150	OK	160	OK
P2 - Period	80	80		100		100		150		150		100	
P2 - Response Measure	60	60	OK	100	OK	100	OK	100	OK	90	OK	80	OK
P3 - Period	80	80		150		150		150		150		200	
P3 - Response Measure	60	60	NOK	100	OK	100	OK	100	OK	50	OK	50	OK
System			NOK		NOK		OK		OK		OK		OK

Tabla 45. Comparación Arquitectura IMA con Arquitectura Convencional para las diferentes iteraciones.

Se ve que el sistema vuelve de nuevo a ser planificable, pero a cambio de aumentar considerablemente los períodos de los escenarios y sus respectivas *deadlines* en algunos casos, si bien manteniéndolos en otros. Este es el efecto de la inclusión de un nuevo elemento en todas las transacciones de datos, *Logical Comms*, que lógicamente desbarata un poco los resultados obtenidos con anterioridad.

La conclusión es que se puede ver que ArchE es, efectivamente, capaz de analizar una arquitectura IMA desde el punto de vista de *ICM Performance* como si se tratase de una arquitectura convencional, teniendo en cuenta que en las transacciones entre las particiones habrá que incluir un elemento, el Logical Comms, que será una auténtica puerta de enlace, de forma que al definir las relaciones tipo *reaction*, una transacción se realizará de la siguiente forma:

IR MOD $\rightarrow$ *Logical Comms* $\rightarrow$ *HYBRID Mod.*

A priori, podía parecer que *Logical Comms* introduciría un cuello de botella a la hora de planificar el sistema, dado que todas las transacciones entre particiones pasan por ella. Los resultados muestran que no ha tenido tanta influencia como se esperaba, aunque se haya tenido que aumentar el período de algunas transacciones.

No ocurre lo mismo desde el punto de vista de ChangeImpact Modifiability, puesto que ArchE no es capaz de tener en cuenta que los elementos del Kernel, que están relacionados con las particiones, no se pueden modificar, aunque sí exista una interfaz que represente esta relación en el diagrama UML. Por tanto, esta especial "relación" entre por ejemplo una partición y Logical Comms, habría que traducírsela a ArchE para que estableciera dicho enlace en el diagrama de relaciones UML pero no lo tuviera en cuenta a la hora de calcular el impacto de una modificación de una partición en el resto de particiones con las que intercambia datos/llamadas a procedimientos. Dicho cambio se podría propagar de una partición a otra, pero no afectaría al elemento del kernel.

Aunque ArchE pueda realizar una evaluación de rendimiento de una arquitectura IMA, no hay que olvidar que no está tratando los tiempos de ejecución de las particiones de acuerdo al método de ventanas de tiempo establecido en la ARINC 653, y por tanto, hay que tomar estos resultados con las debidas reservas.

9 CAPÍTULO 9 – CONCLUSIONES. TRABAJOS FUTUROS

9.1 Conclusiones

Una vez realizado el análisis de las dos arquitecturas software aeroespaciales ejemplo, se pueden sacar varias conclusiones.

Acerca de la capacidad de análisis de ArchE sobre arquitecturas software de simuladores de vuelo, se puede concluir que ArchE es capaz de analizar dichas arquitecturas de la misma manera que otras basadas en Modelo-Vista-Controlador. La arquitectura de un simulador está basada en el uso de servidores distribuidos para contener las diferentes aplicaciones que componen un simulador de vuelo, y por tanto, su análisis va a ser muy similar a las de otros tipos de arquitecturas.

ArchE analizó correctamente esta arquitectura y propuso tácticas para mejorar escenarios no cumplidos. Algunas de dichas tácticas tenían sentido, es decir, ArchE sugiere tácticas desconociendo completamente la funcionalidad y la semántica de la arquitectura en cuestión. Es el arquitecto software quien debe de interpretar dichas tácticas. En este caso, las conclusiones que obtuvo ArchE eran de fácil interpretación y aplicación.

Respecto a las arquitecturas de software de aviónica, ArchE analizó dos casos distintos dentro de las mismas.

Por un lado, una arquitectura de software aeroespacial “convencional”, es decir, sin introducir los conceptos de particionado espacio-temporal. Para esta arquitectura, ArchE, al igual que en el caso anterior, supo proporcionar tácticas para la modificabilidad de la arquitectura, obteniendo mejoras que permitieron cumplir los escenarios; y para el caso de rendimiento, ArchE, a través de MAST, analizó las *deadlines* de los diferentes escenarios, identificando cuando se cumplían y cuando no.

Por otro lado, ArchE también analizó una arquitectura software aeroespacial tipo IMA, es decir, teniendo en cuenta la organización de los módulos en particiones diferentes con una comunicación no directa entre las mismas sino a través del kernel del sistema. En este caso, se obtuvieron resultados sensiblemente distintos.

En el caso de la modificabilidad, no fue posible establecer una relación entre el análisis de este tipo de arquitectura y una convencional, debido a que a ArchE le faltaba información acerca de la no posibilidad de modificar módulos del kernel cuando se modifica una partición y el cambio se propaga a través de toda la ruta de comunicaciones hasta la partición destino.

En el caso de rendimiento, de manera muy positiva ArchE fue capaz de realizar un estudio con resultados muy similares a los obtenidos en una arquitectura convencional, teniendo en cuenta que todas las comunicaciones pasan a través de un elemento del kernel y por tanto, hay que tenerlo en cuenta mediante su introducción en las relaciones de tipo *reaction* entre responsabilidades. Eso sí, ArchE no pudo tener en cuenta el método de las ventanas aplicables a las particiones IMA ARINC 653, sino que utilizó su propio marco de razonamiento para obtener la planificabilidad del sistema.

Respecto a la utilización de ArchE como herramienta de análisis de arquitecturas, se puede concluir que ArchE es una herramienta de fácil manejo que proporciona una ayuda eficaz a la hora de mejorar una arquitectura teniendo en cuenta los atributos de calidad que se quieren aplicar a la misma. Se ha detectado una serie de limitaciones, no sólo en cuanto a las arquitecturas IMA, que parece lo más lógico, puesto que ArchE no fue diseñado en principio para analizarlas, sino para las arquitecturas convencionales. Dichas limitaciones se recogen como parte del Anexo 3.

Entre las limitaciones más significativas, cabe destacar que ArchE a día de hoy sólo proporciona tácticas para *ChangeImpact Modifiability*, lo cual implica que para *ICM Performance*, es el arquitecto software quien debe investigar por qué el sistema no es planificable, a través de los resultados que muestra la herramienta MAST integrada en ArchE. Por otro lado, se ha observado que en *ICM Performance* se han de escalar los resultados, puesto que los tiempos de ejecución de las responsabilidades están limitados a 10 ms máximo (no así en los escenarios). Por último, debido posiblemente a un problema de configuración, ArchE no muestra los resultados de los escenarios que sí se cumplen con un círculo verde en la pantalla principal de escenarios, a pesar de que MAST muestra efectivamente que sí se cumplen. Esto no sucede en el caso contrario, cuando los escenarios no se cumplen ArchE sí muestra el correspondiente círculo rojo.

Como conclusión final puede afirmarse de que ArchE es una herramienta muy útil para el análisis de las arquitecturas, y que siguiendo su desarrollo y adaptándola con nuevos marcos de razonamiento, podría servir para analizar otros tipos de arquitecturas para las cuales no fue pensada en un principio.

9.2 Trabajos Futuros

Se han identificados dos líneas de investigación posibles para continuar este proyecto:

- Tal y como se vio en el capítulo 8, si se desea que ArchE pueda cumplir con los requisitos necesarios para el análisis de arquitecturas IMA no convencionales, se han de implementar nuevos marcos de razonamiento para conseguir que ArchE entienda el concepto de particionamiento en espacio y en tiempo.
  Así mismo, dentro el marco de razonamiento de particionamiento en tiempo, se ha de desarrollar un algoritmo que consiga planificar un escenario de ejecución con ventanas asignadas a diferentes particiones, cuando el número de particiones consideradas en el sistema sea superior a cinco, ya que en este caso la complejidad aumenta considerablemente.
- También se debe de pensar en un marco de razonamiento que sea capaz de indicar a ArchE que para una arquitectura IMA, las modificaciones en particiones se extienden a otras que utilicen sus variables o llamadas a procedimientos, pero no al kernel ni a la interfaz APEX.
- Por último, una revisión de ArchE y de sus limitaciones ya comentadas, como por ejemplo los tiempos de ejecución limitados a 10 ms, ayudaría a extender el uso de esta herramienta a un ámbito más extenso de arquitecturas software.

10 BIBLIOGRAFÍA

[01] Vogel, Olivier; Arnold, Ingo; Chughtai, Arif; Kehrer, Timo. *Software Architecture - A Comprehensive Framework and Guide for Practitioners*. Heidelberg (Germany): Springer, 2009. ISBN: 978-3-642-19735-2.

[02] Shaw, Mary; Garlan, David. *Software architecture: Perspectives on an emerging discipline*. Pearson, 1996. ISBN-10: 0131829572.

[03] Schmidt, Richard F. *Software Engineering - Architecture-driven Software Development*. Elsevier, 2013. ISBN 978-0-12-407768-3.

[04] Gorton, Ian. *Essential Software Architecture*. Springer, 2011 [2ª Edición]. ISBN 978-3-642-19175-6.

[05] Bass, Len; Clements, Paul; Kazman, Rick. *Software Architecture in Practice*. Addison-Wesley, 2003. ISBN: 0-321-15495-9.

[06] Rozanski, Nick; Woods, Eoin. *Software Systems Architecture*. Addison-Wesley, 2005. ISBN 0-321-11229-6.

[07] Buschmann, Frank; Meunier, Regine; Rohnert, Hans; Sommerlad, Peter; Stal, Michael. *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*. Wiley, 1995. ISBN: 978-0-471-95869-7.

[08] Garlan, David; Shaw, Mary. *An Introduction to Software Architecture*. CMU-CS-94-166. 1994.

[09] *Introduction to Software Architecture Evaluation*. Universidad Politécnica de Valencia, de: http://users.dsic.upv.es/~jagonzalez/IST/files/IntroductionArchitectureEvaluation.pdf

[10] Dobrica, L.; Niemela, E. *A Survey on Software Architecture Analysis Methods*. 2002. En: Software Engineering, IEEE Transactions on Software Engineering (Volume:28 , Issue: 7). IEEE. ISSN: 0098-5589

[11] Koziolek, Heiko; Schlich, Bastian; Bilich, Carlos. *A Large-Scale Industrial Case Study on Architecture-based Software Reliability Analysis*. 2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE).

[12] Clements, Paul. *Current Best Practices in Software Architecture Session 4: Evaluating Software Architectures*. 13rd October 2005. Software Engineering Institute. Carnegie Mellon University.

[13] Wojcik, Rob; Bachmann, Felix; Bass, Len; Clements, Paul; Merson, Paulo; Nord, Robert; Wood, Bill. *Attribute-Driven Design (ADD), Version 2.0*. November 2006. Technical Report. CMU/SEI-2006-TR-023. ESC-TR-2006-023.

[14] RTCA DO-178C *Software Considerations in Airborne Systems and Equipment Certification*. 2011

[15] Rierson, Leanna K. *Using The Software Capability Maturity Model For Certification Projects*. Washington. 1998. FAA.

[16] Buter, Andrew; Stienstra, Curt; VanderLeest, Steven H. *Agile for Aerospace*. DornerWorks. GLSEC 2008.

[17] Casals, David. *Aeronautical Software Course: Real Time Software Architectures*. Mission Systems Department.Airbus Defense and Space. 2014

[18] *System Development Modular Approach Eases Avionics Certification Challenges* - COTS Journal online, de: http://www.cotsjournalonline.com/articles/view/101451

[19] ARINC Report 651-1. *Design Guidance For Integrated Modular Avionics*. 1997

[20] DOT/FAA/AR-03/77. *Commercial Off-The-Shelf Real-Time Operating System and Architectural Considerations*. 2004. FAA.

[21] ARINC SPECIFICATION 653-1. *Avionics Application Software Standard Interface*. 2003.

[22] Rufino, José; Craveiro, Joao; *Robust Partitioning and Composability in ARINC 653 Conformant Real-Time Operating Systems*.

[23] Wind River VxWorks 653. 2010, de:

http://www.windriver.com/products/product-overviews/PO_VxWorks653_Platform_0210.pdf

[24] LynxOS-178 RTOS for DO-178B Software Certification, de:

http://www.lynx.com/products/real-time-operating-systems/lynxos-178-rtos-for-do-178b-software-certification/

[25] Green Hills Software - Safety Critical Products: INTEGRITY®-178B RTOS, de:

http://www.ghs.com/products/safety_critical/integrity-do-178b.html

[26] Mejia Alvarez, Pedro; Cova Suazo Nancy, Noemí; Pérez Reséndiz Marisol; *Arquitectura de Software*. CINVESTAV – IPN. Sección de Computación. De:

http://slideplayer.es/slide/1057374/

[27] XPort1020 Freescale MPC8270 Processor-Based Multi-Protocol Twelve-Port Serial 6U cPCI Module, de:

http://www.xes-inc.com/products/view/xport1020/

[27] http://www.xes-inc.com/products/view/xport1020/

[28] http://comps.canstockphoto.es/can-stock-photo_csp26053611.jpg

[29] http://www.futureplatone.com/img/simulador-vuelo-valencia.png

[30] http://www.fancyicons.com/free-icons/108/occupations/png/256/pilot_female_light_256.png

[31] http://comps.canstockphoto.es/can-stock-photo_csp18061284.jpg // http://st2.depositphotos.com/1429923/5516/v/950/depositphotos_55164017-Flat-illustration-of-expert-with-control-panel.-Analytics-and-management.jpg

[32] http://st2.depositphotos.com/1000244/5869/v/450/depositphotos_58693619-sound-speaker-icon.jpg

[33] http://ubuntuarte.com/wordpress/wp-content/uploads/2008/04/sun_server_familyubuntu.jpg

[34] http://www.sdm.es/Web/wp-content/uploads/2014/09/servidores.png

[35] http://previews.123rf.com/images/scanrail/scanrail1205/scanrail120500028/13877506-Hard-disk-and-database-icon-isolated-on-white-background-Stock-Photo.jpg

[36] Bachmann, Felix; Bass, Len; Klein, Mark; *Preliminary Design of ArchE: A Software Architecture Design Assistant*. September 2003. Technical Report. CMU/SEI-2003-TR.

[37] Bachmann, Felix; Bass, Len; Bianco, Philip; Klein, Mark. *Using ArchE in the Classroom: One Experience.* September 2007. Technical Note. CMU/SEI-2007-TN-001.

[38] Bachmann, Felix; Bass, Len; Klein, Mark. *Illuminating the Fundamental Contributors to Software Architecture Quality*. August 2002. Technical Report. CMU/SEI-2002-TR-025. ESC-TR-2002-025.

[39] Bachmann, Felix; Klein, Mark. *Methodical Design of Software Architecture Using an Architecture Design Assistant (ArchE)*. 2005 by Software Engineering Institute - Carnegie Mellon University. Pittsburgh.

[40] Bachmann, Felix; Bass, Len; Bianco, Phil. *Software Architecture Design with ArchE*. Marzo de 2007. Software Engineering Institute. Carnegie Mellon University. Pittsburgh.

[41] Bass, Len. *ArchE – An Architecture Design Assistant*. August 2, 2007. Software Engineering Institute. Carnegie Mellon University. Pittsburgh.

[42] Bianco, Phil; Diaz-Pace, Andres. *Current SEI SAT Initiative Technology Investigations*. May 1st, 2008. Software Engineering Institute. Carnegie Mellon University. Pittsburgh.

[43] Moreno, Gabriel A.; Hansen, *Jeffrey. Overview of the Lambda-* Performance Reasoning Frameworks*. February 2009. Technical Report. CMU/SEI-2008-TR-020. ESC-TR-2008-020.

[44] Diaz-Pace, Andres ; Kim, Hyunwoo; Bass, Len; Bianco, Phil; Bachmann, Felix. *Integrating Quality-attribute Reasoning Frameworks in the ArchE Design Assistant*. Software Engineering Institute. Carnegie Mellon University. Pittsburgh.

[45] Champagne, Roger; Gagné, Sébastien. *Towards automation of architectural tactics application – an example with ArchE*. 2011. Dept. of Software and IT Engineering. ÉTS (University of Québec). Montréal, Canada.

[46] Lee, Jinhee; Bass, Len. *Elements of a Usability Reasoning Framework*. September 2005. Software Architecture Technology Initiative. Technical Note. CMU/SEI-2005-TN-030.

[47] Clements, Paul; Bass, Len. Relating Business Goals to Architecturally Significant Requirements for Software Systems. May 2010. Technical Note. CMU/SEI-2010-TN-018.

[48] Gaitán Peña, Carlos Alberto. *Arquitecturas Software: Gestión de los atributos de calidad y su incorporación en ArchE para la mejora del análisis, evaluación y soporte en la toma de decisiones durante el desarrollo arquitectónico*. Enero de 2014. Trabajo Fin de Master del Máster Universitario En Investigación En Ingeniería De Software Y Sistemas Informáticos – UNED. Curso 2012/2013.

[49] Clements, Paul; Bachmann, Felix; Bass, Len; Garlan, David; Ivers, James; Little, Reed; Merson, Paulo; Nord, Robert; Stafford, Judith. *Documenting Software Architectures. Views and Beyond*. Pearson, 2011 [2ª Edición]. ISBN-10: 0321552687 / ISBN-13: 9780321552686.

[50] Malveau, Raphael; Mowbray, Thomas J. *Software Architect Bootcamp*. Prentice Hall, 2000. ISBN: 0-13-027407-0

11 GLOSARIO

ADD	**A**ttribute-**D**riven **D**esign
ALMA	**A**rchitecture-**L**evel **M**odifiability **A**nalysis
ALPSM	**A**rchitecture **L**evel **P**rediction of **S**oftware **M**aintenance
APEX	**A**pplication/**E**xecutive **I**nterface.
API	**A**pplication **P**rogramming **I**nterface
ArchE	**A**rchitecture **E**xpert
ARINC	**A**eronautical **R**adio, **Inc**orporated
ATAM	**A**rchitecture **T**rade-Off **A**nalysis **M**ethod
BSP	**B**oarding **S**upport **P**ackage
COTS	**C**ommercial **O**ff-**T**he-**S**helf
CPU	**C**entral **P**rocessing **U**nit
COEX	**Co**re/**Ex**ecutive Interface
DAL	**D**esign **A**ssurance **L**evel
EASA	**E**uropean **A**viation **S**afety **A**gency
ESAAMI	**E**xtending **SAAM** by **I**ntegration in the Domain
EUROCAE	**Euro**pean **O**rganization for **C**ivil **A**viation **E**quipment
FA	**F**actor **A**nalysis
GRMA	**G**eneralized **R**ate **M**onotonic **A**nalysis
HIS	**H**ardware **I**nterface **S**ystem
HM	**H**ealth **M**onitor
HOL	**H**igh **O**rder **L**anguage
ICM	**I**ntermediate **C**onstructive **M**odel
IMA	**I**ntegrated **M**odular **A**vionics
I/O	**I**nputs/**O**utputs
KPA	**K**ey **P**rocess **A**reas
LRM	**L**ine **R**eplaceable **M**odules

MAF	**Ma**yor **F**rame
MAST	**M**odeling and **A**nalysis **S**ui**t**e for Real-Time Applications
MIF	**Mi**nor **F**rame
MMU	**M**emory **M**anagemnet **U**nit
MOS	**M**odule **O**perating **S**ystem
PMK	**P**artition **M**anagement **K**ernel
POS	**P**artition **O**perating **S**ystem
POSIX	**P**ortable **O**perating **S**ystem **I**nterface UNI**X** Standard
QuADAI	**Qua**lity-**D**riven **A**rchitectural **I**mprovement
RF	**R**easoning **F**ramework
RTCA	**R**adio **T**echnical **C**ommission for **A**eronautics
RTEMS	**R**eal **T**ime **E**xecutive for **M**ultiprocessors **S**ystems
RTOS	**R**eal **T**ime **O**perating **S**ystems
SAAM	**S**cenario-Based **A**rchitecture **A**nalysis **M**ethod
SAAMCS	**S**cenario-Based **A**rchitecture **A**nalysis **M**ethod, **C**omplex **S**cenarios
SAAMER	**S**oftware **A**rchitecture **A**nalysis **M**ethod for **E**volution and **R**eusability
SAEM	**S**oftware **A**rchitecture **E**valuation **M**odel
SALUTA	**S**cenario-based **A**rchitecture **L**evel **U**sabili**T**y **A**nalysis
SBAR	**S**cenario-**B**ased **A**rchitecture **R**eengineering
SEI	**S**oftware **E**ngineering **I**nstitute
SO	**S**istema **O**perativo
SW-CMM	**S**oft**w**are **C**apability **M**aturity **M**odel
TLC	**T**race-based **L**oad **C**haracterization
TX/RX	**T**ransmission/**R**eception
WBA	**W**orst-Case, **B**locking and **A**synchrony
XML	e**X**tensible **M**arkup **L**anguage

12 ANEXO 1 - INTRODUCCIÓN A ARCHE

12.1 Conceptos Básicos de ArchE

ArchE (Architecture Expert), se define, tal y como viene recogido en [36], como un asistente que ayudará a los diseñadores software a generar una arquitectura que satisfaga los requisitos expresados como escenarios de atributos de calidad. Arche, tal y como dice [37], es una herramienta software para ayudar al diseñador a desarrollar arquitecturas que poseen niveles especificados de calidad requeridos.

ArchE tiene conocimiento de atributos de calidad pero no conoce ningún dominio del problema. Por consiguiente, ArchE puede ofrecer consejos sobre cómo satisfacer requisitos de atributos de calidad pero no conoce que significan estos consejos para el arquitecto con respecto al particular dominio del sistema.

ArchE es un sistema basado en reglas, en el cual los modelos de atributos de calidad son vistos como marcos. ArchE utiliza Jess (intérprete basado en reglas de Java). Además, ArchE está implementado como una aplicación Eclipse, el cual proporciona una familiarización inmediata con la interfaz de usuario y un concepto de operación para cualquiera que haya usado Eclipse.

En la Figura 208 se muestra el flujo general de ArchE:

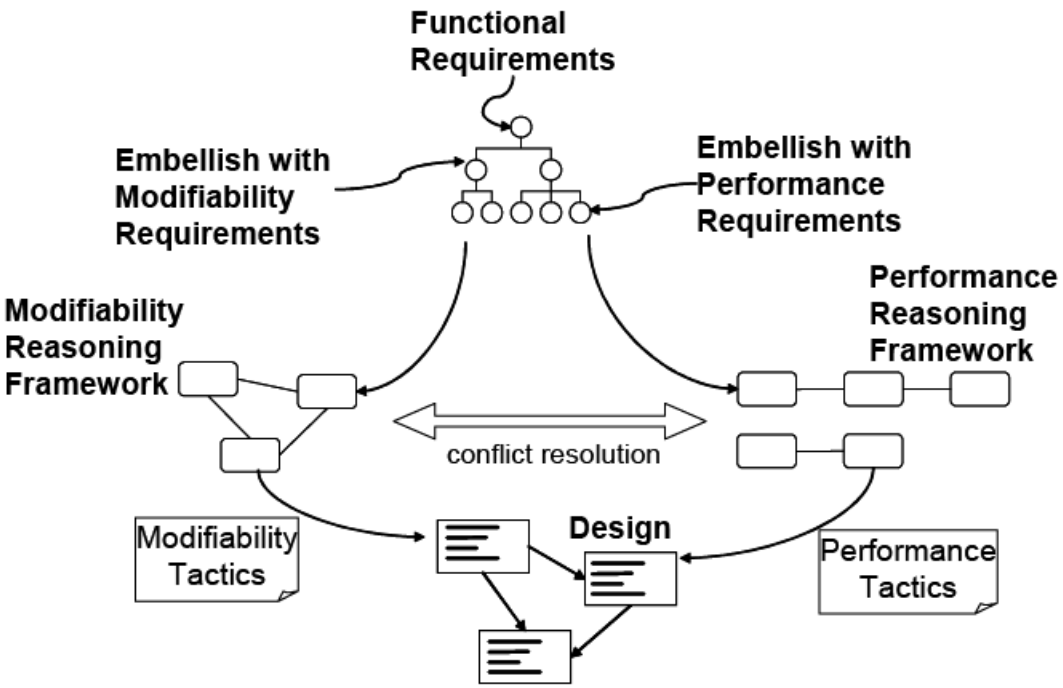


Figura 208. Flujo General de ArchE. [36]

Dentro de esta figura se recogen los conceptos más importantes que maneja ArchE, según [36]:

- *Escenarios de atributos de calidad:* tal y como se describieron en el capítulo 3, tienen una estructura que incluye seis conceptos: estímulo, fuente del estímulo, entorno, artefacto que es estimulado, respuesta y medida de respuesta.

- *Marcos de razonamiento:* es un cuerpo de conocimiento sobre un atributo de calidad particular, que incluye métodos para calcular la medida de la respuesta frente a un estímulo, dada una colección de parámetros independientes. Además, incluye tácticas arquitectónicas que permiten ajustar dichos parámetros independientes, para modificar y controlar el valor de los parámetros dependientes.
- *Responsabilidades:* son actividades asignadas al software que está siendo diseñado. Las responsabilidades son usadas por ArchE como medios para expresar requisitos funcionales, como una porción integral de los escenarios de atributos de calidad, y como medios de integrar los modelos producidos por diversos marcos de razonamientos de atributos de calidad.

Una vez definidos estos conceptos, Arche funciona de la siguiente manera: [36]

Dentro de ArchE los requisitos funcionales se presentan como grafos de responsabilidades y son complementados con requisitos de atributos de calidad, en la forma de escenarios de atributos de calidad.

Para cada atributo de calidad, existen marcos de razonamiento que convierten dichos escenarios de calidad en modelos específicos de atributos de calidad. Cada modelo representa un diseño que satisface los requisitos especificados para dicho atributo de calidad.

Los marcos de razonamiento resuelven conflictos entre diferentes modelos de atributos de calidad para crear un modelo global que satisfaga todos los requisitos de atributos de calidad. Este modelo global se convierte entonces en una representación arquitectónica del diseño.

En la Figura 209 [36] se muestra la estructura interna básica de ArchE, que es una estructura tipo *blackboard* donde el repositorio de datos es la *FactBase*, el motor de reglas es proporcionado por Jess, y cada conjunto de reglas es activado por los datos en el repositorio y sitúa nuevos datos en el conjunto de datos. Cada conjunto de datos se denomina módulo.

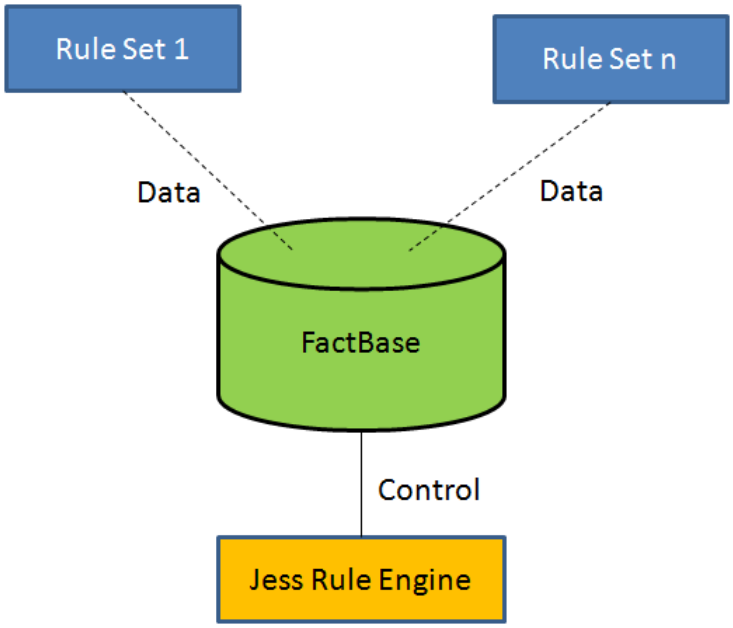


Figura 209. Arquitectura *Blackboard* de ArchE. [36]

En [40] describen los principios del diseño de arquitectura implementados en ArchE: (Figura 210)

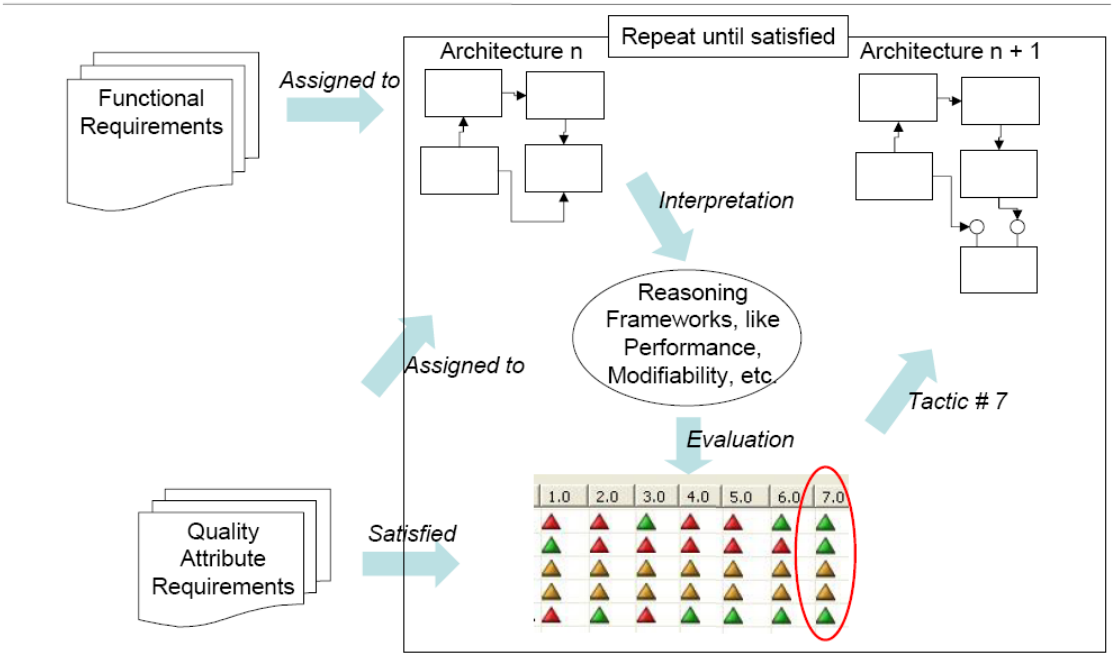


Figura 210. Principios de Diseño de Arquitectura en ArchE. [40]

Este principio de diseño se basa en los siguientes puntos:

- Establecer requisitos:
  - Requisitos funcionales
  - Dependencias entre requisitos
  - Requisitos de atributos de calidad
  - Diseño inicial conteniendo sólo el sistema
- Comprobar que el diseño es bueno, produciendo modelos de atributos de calidad que proporcionen información acerca de los atributos de calidad
  - Extraer información requerida del modelo desde el diseño (interpretación)
  - Correr el modelo para calcular los valores para los requisitos de atributos de calidad (evaluación)
- Intentar una mejora, mediante un conjunto de tácticas que mejoren la arquitectura
  - Interpretar el modelo para determinar posibles tácticas
  - Aplicar las tácticas al diseño mediante el cambio de elementos, relaciones y sus propiedades.

12.2 Operación de ArchE

12.2.1 Conceptos Claves de ArchE

En la Figura 211 se muestran los conceptos claves de ArchE: [36]

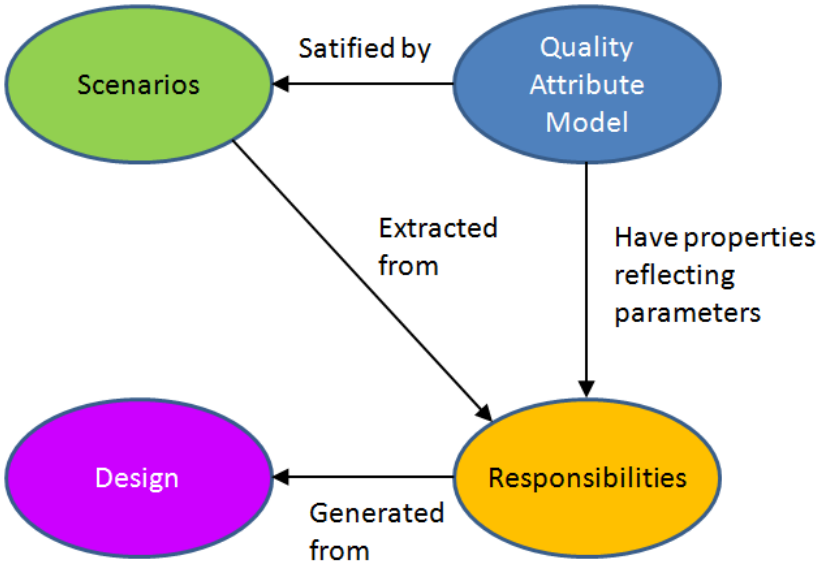


Figura 211. Conceptos Claves de Arche y sus Relaciones. [36]

- Escenarios: los requisitos de calidad para el sistema.
- Responsabilidades: definidas en la sección anterior, estas responsabilidades están ligadas a sus respectivas fuentes (por ejemplo, escenarios, requisitos o tácticas), incluyendo la relación entre los requisitos como se muestra en la Figura 211, y tienen parámetros que incluyen la asignación a los elementos arquitectónicos y propiedades que los diversos marcos de razonamiento necesitan, como tiempo de ejecución o coste de modificación.
- Modelo de Atributos de Calidad: es una instancia del marco de razonamiento completamente instanciada. Son los parámetros independientes para los marcos de razonamiento. Estos parámetros están ligados a sus fuentes (por ejemplo, los escenarios, especificaciones del diseñador, resultados de la aplicación de una táctica particular), así como a las responsabilidades a las cuales pertenecen.
- Diseño: una enumeración de elementos arquitectónicos, sus propiedades y sus relaciones.

12.2.2 Actividades Básicas de ArchE e Interacciones con el Usuario

Según [36], hay diferentes tipos de actividades que un usuario puede realizar en ArchE:

12.2.2.1 Interacciones Básicas

En cualquier punto de la preparación del diseño, el diseñador tiene la opción de salvar el estado actual del diseño, o restaurar el diseño de un estado previo salvado; especificar aspectos del diseño; e importar, exportar o modificar escenarios existentes o requisitos.

Dentro del apartado de especificar aspectos del diseño, hay múltiples tipos de actividades, que incluyen:

- Dar un nombre significativo al conjunto de los requisitos
- Especificar el valor de los parámetros, dentro de cualquier marco de razonamiento
- Especificar restricciones.

12.2.2.2 Adquirir Requisitos

El diseñador puede introducir requisitos, que ArchE transformará posteriormente en responsabilidades.

ArchE adquiere tanto los requisitos funcionales (que son traducidas a responsabilidades) como los requisitos de los escenarios de calidad.

12.2.2.3 Refinar Escenarios

ArchE refina los escenarios, identificando las seis partes en las que se descompone: estímulo, fuente del estímulo, entorno, artefacto que es estimulado, respuesta y medida de respuesta. Dichas partes son añadidas por el usuario en la interfaz de Eclipse.

12.2.2.4 Elegir Marco de Razonamiento

Arche debe determinar qué marco de razonamiento debe resolver un determinado escenario; la actual configuración de ArchE permite elegir entre dos opciones: *ICM Performance* y *ChangeImpact Modifiability*.

12.2.2.5 Construir Los Modelos de Atributos de Calidad.

Un modelo de atributo de calidad es una instancia completamente instanciada de un marco de razonamiento. Si ArchE está resolviendo múltiples escenarios simultáneamente, genera múltiples modelos de atributos de calidad, uno por cada escenario, que son consistentes.

Un modelo consiste en un conjunto de parámetros asociados con atributos de calidad.

Hay tres tipos de interacciones que pueden ocurrir entre ArchE y el diseñador durante la actividad de construir el modelo de atributos de calidad:

- Especificar parámetros: los parámetros dentro de un modelo de atributos de calidad son fijos o libres. Los parámetros pueden ser fijados por el estado actual del diseño o por la especificación del diseñador durante su actividad.
- Asistir a ArchE para elegir tácticas: el constructor del modelo de atributos de calidad explora posibles tácticas para generar una solución, y genera preguntas que deben ser respondidas por el diseñador.
- Reportar la incapacidad de ArchE para satisfacer escenarios particulares: si ArchE no es capaz de generar una solución para un escenario en particular, el diseñador es informado y las razones para esta incapacidad se presentan. El diseñador puede entonces modificar alguno de los escenarios y ArchE intentará generar el diseño de nuevo. El diseñador probablemente relajará la medida e respuesta pero puede también especificar alguno de los parámetros o cambiar el estímulo.

12.2.2.6 Construir el Diseño

Según se describe en [36], una vez que se crea un modelo, ArchE construye un diseño. Un diseño consiste en una serie de elementos arquitectónicos y sus propiedades.

La actividad de construcción del diseño es invocada por iniciativa de ArchE cuanto todos los escenarios se satisfacen o bien por iniciativa del diseñador. En cualquier caso, ArchE debe crear el diseño a partir del modelo existente.

La actividad de construcción del diseño puede fallar sólo cuando una restricción se especifica. Si no hay restricciones, el diseño debería ser alcanzable. Si un diseño falla, ArchE especifica una restricción en los valores de los parámetros y reintenta el diseño. El diseñador queda involucrado sólo cuando ArchE no puede determinar las restricciones.

Arche también propone, a parte del diseño, los requisitos de atributos de calidad no satisfechos por dicho diseño, y propone una serie de transformaciones arquitectónicas para mejorar el diseño con respecto a dichos requisitos de atributos de calidad, tal y como se define en [37].

El arquitecto selecciona una trasformación y proporciona información adicional para los nuevos elementos del diseño, tales como nombres significativos, tiempos de ejecución, o coste del cambio. Este proceso continua hasta que o todos los requisitos de calidad son satisfechos o ArchE no tenga más propuestas.

12.3 Marcos de Razonamiento en ArchE

Según [41], un marco de razonamiento o RF (*Reasoning Framework*) es un vehículo para encapsular el conocimiento de atributos de calidad y las herramientas necesarias para analizar el comportamiento de un sistema con respecto a algunos de esos atributos de calidad.

La razón para encapsular dicho conocimiento sobre atributos de calidad es para permitir el incorporar dicho conocimiento en ArchE sin requerir que los atributos de calidad sepan algo el uno del otro.

Según [40], un marco de razonamiento en Arche realiza las siguientes tareas:

- Traduce desde la descripción de la arquitectura al modelo de atributos de calidad (Interpretación).
- Evalúa los escenarios de atributos de calidad en términos del modelo (Evaluación).
- Propone tácticas para mejorar la arquitectura.

Se necesitan dos entradas para un marco de razonamiento en ArchE:

- La arquitectura actual.
- Los escenarios de atributos de calidad relevantes.

Y se proporcionan dos salidas:

- La evaluación de la arquitectura actual con respecto a los escenarios de atributos de calidad.
- La lista de potenciales tácticas para mejorar la arquitectura si al menos un escenario no se cumple.

Para hacer todo esto, los marcos de razonamiento de ArchE requieren una definición clara de los elementos arquitectónicos, relaciones y propiedades que pueden influir en un atributo de

calidad. La interpretación extrae esta información desde la arquitectura y crea un modelo a partir de ella.

Además, se requiere la existencia de una fórmula para realizar los cálculos con el modelo, para proporcionar información sobre el cumplimiento del atributo de calidad. Esta tarea es llevada a cabo por la evaluación.

Por último, ArchE requiere una clara definición de los posibles cambios a la arquitectura para conseguir un mejor cumplimiento de los atributos de calidad. Esta tarea se lleva a cabo por las tácticas.

Los marcos de razonamiento ArchE están basadas en las teorías explicadas en [38], y básicamente son dos: Modificabilidad y Rendimiento.

En [39] se proporcionan unos ejemplos acerca del funcionamiento práctico de los marcos de razonamiento de ArchE.

Según [41], los elementos de un marco de razonamiento se pueden observar en la Figura 212:

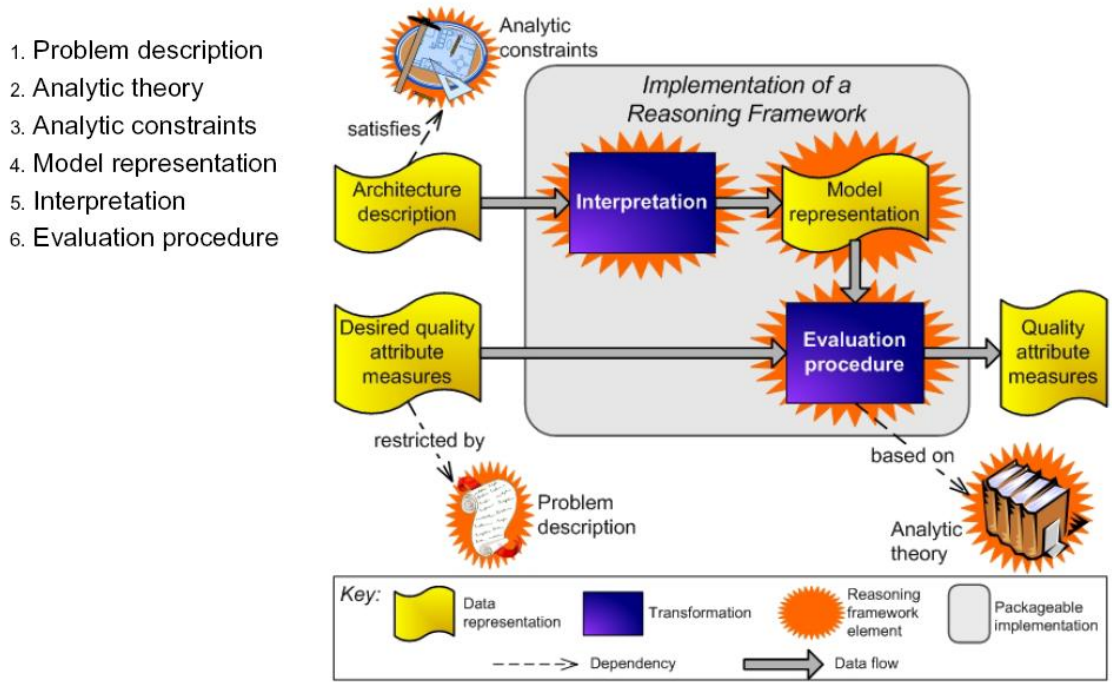


Figura 212. Elementos de un Marco de Razonamiento. [41]

Según [38], los principios fundamentales de dichos marcos de razonamiento son los siguientes:

12.3.1 Modificabilidad

Modificabilidad trata sobre el coste de hacer cambios. Intentar hacer una arquitectura modificable significa localizar cambios en una pequeña área de la misma. Lo peor que puede ocurrir para un arquitecto es que un pequeño cambio en una parte de la arquitectura afecte muchas otras partes.

Los cambios tienen tres aspectos:

- Qué se va a cambiar (interfaz de usuario, Sistema operativo, etc.)
- Quién debería hacer el cambio (desarrollador, administrador del sistema, etc.)
- Cuando se va a hacer el cambio (durante el desarrollo, instalación, cuando el sistema está corriendo, etc.)

12.3.1.1 Especificando Requisitos de Modificabilidad

Los cambios pueden clasificarse de de acuerdo a:

- Probabilidad: cómo de probable es que cierto cambio ocurra
- Frecuencia: cómo de a menudo cierto cambio ocurrirá.
- Dependencia: si el cambio está conectado a otro tipo diferente de cambio, es decir, si el cambio provoca a su vez otro cambio.

12.3.1.2 Elementos Primarios que Afectan a la Modificabilidad

Hay dos factores claves:

- El módulo por sí mismo. Un módulo es una unidad de implementación de software que proporciona una unidad coherente de funcionalidad, con su colección de interfaces y su conjunto de responsabilidades que definen las tareas a realizar. Otros módulos dependen de dichas tareas.
- De qué depende dicho módulo. Por ejemplo, el módulo B depende del A si un cambio en A requiere un cambio en el módulo B.

Otros factores que contribuyen a la modificabilidad son:

- Las responsabilidades se asignan a diferentes capas de una aplicación, para soportar la división entre la interfaz de usuario, la aplicación software en sí, y la abstracción hardware. Siempre y cuando se puedan ajustar en estas categorías, los cambios requeridos pueden ser hechos de manera más fácil.
- Cómo está definida la interfaz, es decir, qué es visible y qué está oculto tiene una gran influencia en la modificabilidad.
- La capa intermedia puede actuar como una intermediaria para prevenir cambios a las capas más bajas que se propaguen desde las capas más altas.

En la Figura 213 se muestra la división en capas de un módulo software: [38]

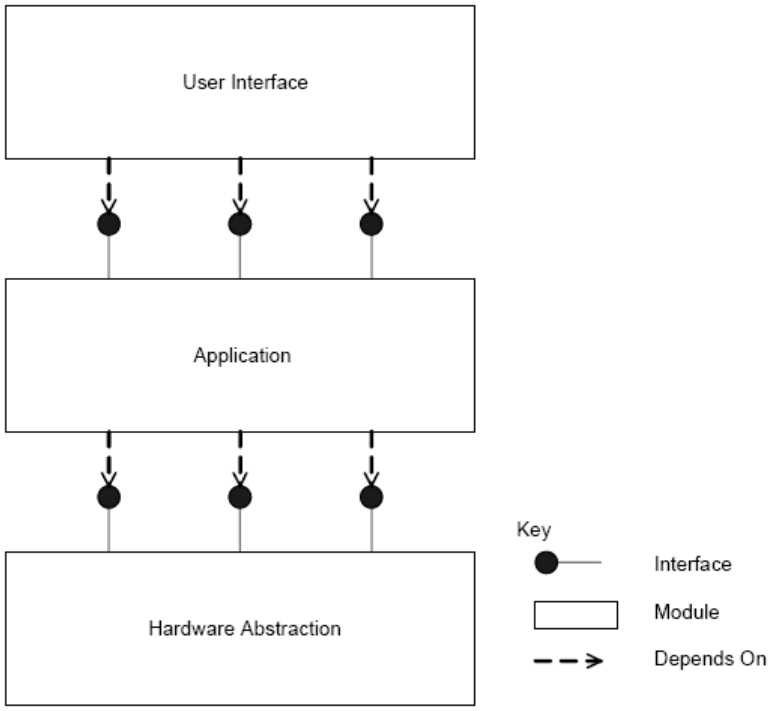


Figura 213. Ejemplo de Arquitectura en Capas. [38]

12.3.1.3 Tácticas de Modificabilidad

Se organizan en tres conjuntos:

- *Tácticas para localizar modificaciones esperadas.* Las responsabilidades que se asignan a los módulos tienen gran influencia en el coste de hacer cambios. Dependiendo de cómo se haga la asignación, un cambio específico puede afectar a un único módulo o a múltiples módulos. La meta de estas tácticas es afectar al menos número de módulos como sea posible por un cambio simple, presentando guías de cómo asignar responsabilidades.
- *Tácticas para restringir la visibilidad de las responsabilidades*. Si un módulo es afectado por un cambio, es importante saber si ese cambio será visible fuera de dicho módulo. Si es así, es posible que se requieran cambios en otros módulos.
- *Tácticas para impedir el efecto onda*. El efecto onda desde una modificación es la necesidad de hacer cambios a módulos que no están directamente afectados por dicha modificación. Esto ocurre debido a que la dependencia entre el módulo afectado por el cambio y otro que depende de él.

12.3.1.4 Análisis de Modificabilidad

El análisis se divide en tres partes:

- Determinar cómo de bien se han asignado las responsabilidades para localizar las modificaciones esperadas
- Determinar cómo de bien la información está oculta y si hay algún efecto onda
- Determinar las posibles dependencias entre módulos que pueden sufrir efectos del cambio en otros módulos.

12.3.1.5 Implementación del Marco de Razonamiento de Modificabilidad en ArchE

Según [40], la modificabilidad de una arquitectura depende de la asignación de funcionalidad a los módulos y de las dependencias entre módulos. La modificabilidad se mide en coste (esfuerzo) del cambio. Por lo tanto, la siguiente información debe estar disponible:

- Grafos de Responsabilidades: tienen dependencias, y pueden ser descompuestos
- Propiedades de responsabilidades: coste del cambio (asignado a cada responsabilidad)
- Propiedades de dependencia entre responsabilidades: aporta una idea sobre la fortaleza del acoplamiento
- Responsabilidades asignadas a los módulos.
- Escenarios de modificabilidad han de ser asignados a responsabilidades.

El trabajo del arquitecto es asignar costes del cambio a cada responsabilidad. No hay manera de que ArchE pueda conocer los valores iniciales. Si el arquitecto no asigna una medida de la fortaleza del acoplamiento para dependencias entre responsabilidades, entonces ArchE asume una probabilidad por defecto del 0.7. Esta probabilidad es una medida de cómo se va a propagar un cambio en una responsabilidad a otras responsabilidades en un módulo.

Es fácil construir el modelo a partir de la descripción de la arquitectura debido a la forma de la misma:

- Cada módulo tiene un coste del cambio que es la suma del coste del cambio de las responsabilidades asignadas.
- Cada módulo que no está descompuesto, se convierte en un nodo en el modelo.
- Cada nodo tiene un coste del cambio que es el coste del cambio del módulo
- Las dependencias entre responsabilidades determinan directamente las dependencias entre módulos.
- Las dependencias entre módulos se convierten en los arcos en el modelo que conecta los nodos.
- Cada dependencia de módulo tiene una fortaleza de acoplamiento, que se asigna a los arcos en el modelo.

En la Figura 214 se resume cómo ArchE crea un modelo de modificabilidad a partir de una arquitectura: [40]

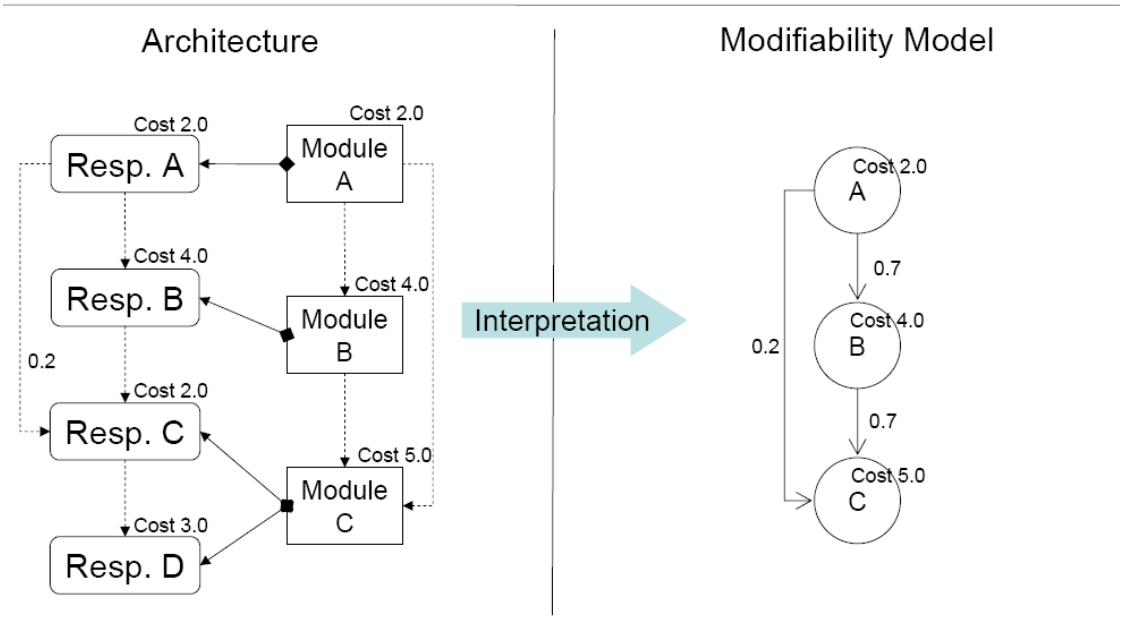


Figura 214. Marco de Razonamiento de Modificabilidad: Relación Arquitectura-Modelo. [40]

En [41] se describe cómo pueden estar acoplados los módulos:

- Altamente acoplados → alta probabilidad de propagación del cambio.
- Medianamente acoplados → probabilidad media de propagación del cambio.
- Bajo acoplamiento acoplados → baja probabilidad de propagación del cambio.

Respecto a las tácticas, [40] establece que las siguientes tácticas de modificabilidad están implementadas en ArchE:

- Encapsulación: reduce el acoplamiento
- Incrementar el nivel de abstracción: reduce el acoplamiento
- Insertar un intermediario: reduce el acoplamiento de algunas dependencias.
- Dividir responsabilidades: permite asignar nuevas responsabilidades a módulos y varía el coste del cambio.
- *Wrapper*: reduce todos los acoplamientos de salida.

12.3.2 Rendimiento

Según se detalla en [38], se detalla rendimiento como la habilidad de un sistema para asignar recursos a las peticiones de servicio de manera que:

- Se satisfagan los requisitos de tiempo
- Se proporcionan varios niveles de servicio

en presencia de:

- Peticiones en competencia
- Demanda variable; y
- Disponibilidad de recursos variable

12.3.2.1 Especificando Requisitos de Rendimiento

Los requisitos de rendimiento están especificados dentro de los escenarios de rendimiento que indican el estímulo al sistema y las respuestas requeridas del sistema a dichos estímulos. Los estímulos de rendimiento son flujos de eventos que el sistema tiene que procesar, y pueden ser clasificados como sigue:

- Periódicos: los intervalos entre llegadas son iguales.
- Estocásticos: los eventos llegan siguiendo alguna distribución probabilística.
- Esporádicos: los intervalos entre llegadas pueden no ser más pequeños que un mínimo especificado.

La respuesta a un escenario de rendimiento está usualmente especificado en términos de requisitos de latencia y tasa de procesamiento: aquellos para minimizar el *jitter*, asegurar que las relaciones de precedencia están adheridas; y proporcionar capacidad de reserva para crecimiento.

12.3.2.2 Elementos Primarios que Afectan al Rendimiento

La latencia es el tiempo que le lleva a un sistema responder a un estímulo incluyendo el tiempo gastado en esperar el tener acceso al recurso y usarlo. Los elementos que afectan a la latencia son:

- Recursos: unidades físicas con capacidad limitada que proporcionan servicios
- Demanda de recursos: generada por estímulos o eventos; un evento es una petición para realizar un cómputo, o lo que es lo mismo, para usar un recurso específico.
- Arbitraje: gestionar demandas competidoras por recursos, según ciertas reglas. Cuando múltiples eventos requieren atención al mismo tiempo, el sistema debe decidir cual atender primero, y como compartir el recurso.

Cuando se considera la latencia, es importante considerar el tiempo de ejecución y las fuentes del tiempo de espera, que incluyen:

- Un evento que está en cola detrás de otros eventos
- Un proceso que necesita acceder a datos compartidos, requiere acceso exclusivo mutuo, y está siendo accedido por otros procesos.
- Un recurso que requiere otros recursos para realizar una terea.
- Un proceso que está entregando el servicio requerido y q tiene que compartir el procesador con otros procesos.

La latencia total para que un recurso responda a un evento comprende el tiempo de ejecución, el overead (cualquier combinación de tiempo de computación, memoria, ancho de banda, o otros recursos que son requeridos para atender una tarea particular) *preemption time* (tiempo que una respuesta debe esperar mientras respuestas de prioridad mayor se están ejecutando), tiempo de bloqueo (tiempo que una respuesta de mayor prioridad debe esperar mientras que una respuesta de prioridad inferior hace uso de un recurso que no se puede liberar), tiempo de trasferencia y tiempo debido a la computación en serie.

Entre los factores que contribuyen a la latencia, están los siguientes:

- La atención en serie de peticiones lleva a paralizar recursos cuando la concurrencia física se puede aprovechar
- El overead utilizando servicios, los mensajes alineados y el transferir control introduce tiempo de ejecución extra.
- Los algoritmos, especialmente en sitios con alta demanda, tienen un impacto muy fuerte en el tiempo de ejecución
- El no tener en cuenta el criterio de arbitraje de recursos, tales como las *deadlines*, pueden llevar a un incremento de latencia para determinados eventos.

12.3.2.3 Tácticas de Rendimiento

En general, la latencia se ve afectada por el nivel y la naturaleza de la demanda de recursos. Esto incluye tasas de llegada de eventos, tiempo de ejecución resultante de la llegada de eventos, y el nivel de variabilidad en dicha llegada. También se ve afectada la latencia por las reglas para elegir entre demandas competidoras por un recurso. Así mismo, la incapacidad de usar un recurso también afecta a la latencia. Por tanto, las tácticas de rendimiento se agrupan en tres niveles:

- *Tácticas para gestionar la demanda de recursos*: controlan los tiempos de espera y de trasferencia mediante el control de las demandas de recursos
- *Tácticas para arbitraje entre demandas en conflicto*: estas tácticas controlan los tiempos de interrupción y espera cuando hay peticiones competidoras, y toman en consideración la importancia semántica o la urgencia (*deadlines*) cuando hay contención por recursos compartidos.
- *Tácticas para gestionar múltiples recursos*: estas tácticas habilitan que múltiples recursos sean usados de manera eficiente para asegurar que los recursos disponibles sean usados cuando se necesiten, por tanto controlando el tiempo de espera de los mismos.

12.3.2.4 Análisis de Rendimiento

Se consideran en el análisis del rendimiento dos posibilidades:

- Gestión de la demanda.

Para el análisis de la gestión de la demanda, se consideran colas FIFO, con un proceso simple, una cola simple y un flujo simple de mensajes, como se muestra en la Figura 215: [38]

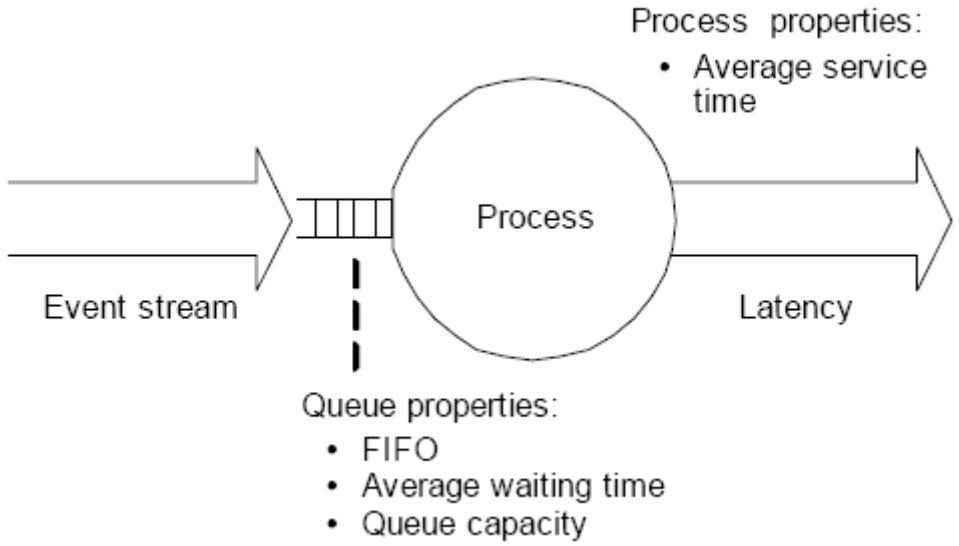


Figura 215. Proceso Único con una única Cola que sirve un único Flujo de Mensajes. [38]

En este caso, la latencia se compone de un tiempo de espera y un tiempo de servicio. El tiempo de espera se compone de dos partes: la cantidad de trabajo que está en la cola y el trabajo que queda para ser completado, para el evento que está siendo servido.

También se puede considerar el caso de múltiples flujos tratados como uno sólo servidos por un único proceso, tal y como se puede observar en la Figura 216: [38]

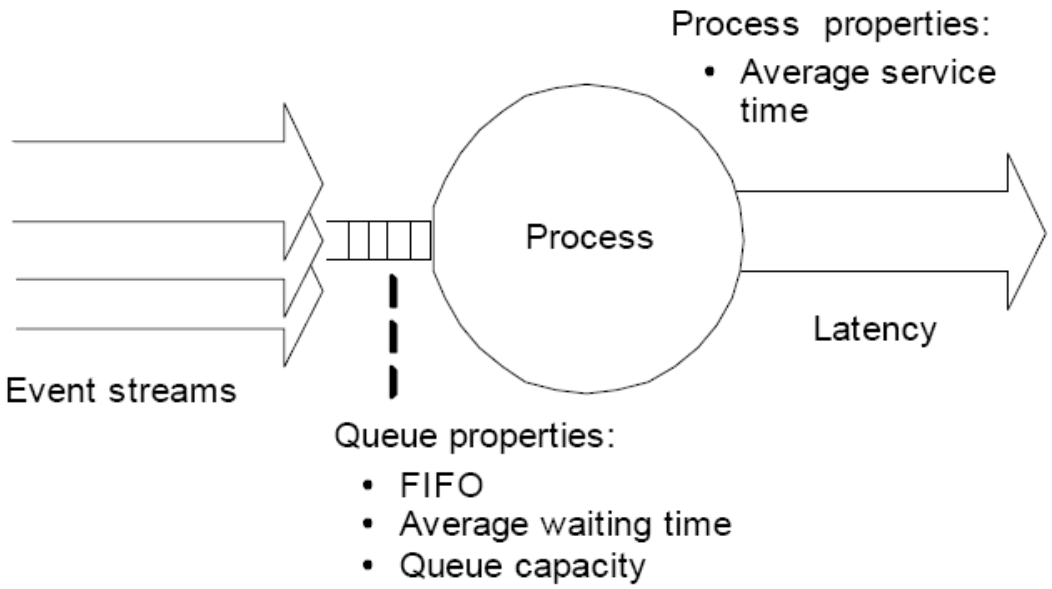


Figura 216. Flujos Múltiples que son tratados como un Único Flujo servido por un Único Proceso. [38]

- Arbitraje

Para el análisis del arbitraje, se tiene en cuenta la planificación por prioridades, y en los efectos que un flujo tiene sobre otro, en particular, considerando múltiples flujos de eventos, cada uno con unos tiempos de ejecución y entre llegadas limitados.

12.3.2.5 Implementación del Marco de Razonamiento de Rendimiento en ArchE

Según [40], el marco de razonamiento de rendimiento en ArchE está basado en el *Rate Monotonic Analysis*, adecuado para razonamiento sobre *deadlines* en tiempo real. Esta teoría usada en ArchE tiene las siguientes hipótesis:

- Único procesador
- La unidad computacional básica es una tarea
- Puede haber recursos compartidos entre tareas
- Las tareas son independientes excepto para dependencia explícita en recursos compartidos
- Sólo una tarea puede usar un recurso compartido en un punto dado en el tiempo.
- Las prioridades de planificación del procesador vienen dadas por el orden de la tarea (por ejemplo, la tarea 1 es la de mayor prioridad, la tarea 2 es la segunda más prioritaria, etc.)

El rendimiento de una arquitectura depende de la asignación de funcionalidad a las tareas. Luna de las típicas medidas de rendimiento es la latencia o tiempo que se tarda en llevar a cabo una tarea. Por lo tanto, la siguiente información debe estar disponible en ArchE:

- Escenarios de rendimiento: tienen un período asociado
- Asignación de escenarios a responsabilidades
- Propiedades de las responsabilidades: tiempo de ejecución.

El trabajo del arquitecto es asignar a cada responsabilidad un tiempo de ejecución. No hay manera de que ArchE pueda conocer los valores iniciales.

A partir de la descripción de la arquitectura, se construye un modelo de la manera siguiente:

- Cada escenario de rendimiento se convierte en una tarea.
- El período especificado para el escenario se convierte en el período de esta tarea.
- Cada responsabilidad tiene un tiempo de ejecución.
- Las responsabilidades asignadas a un escenario se convierten en responsabilidades asignadas a la tarea.
- Las responsabilidades no asignadas a un escenario de rendimiento se asignan a una tarea adicional de baja prioridad (tarea en segundo plano)
- Las responsabilidades compartidas se convierten en recursos compartidos
- Un recurso compartido tiene un tiempo de ejecución para cada tarea que usa este recurso.

En la Figura 217 se representa el modelo de rendimiento de ArchE:

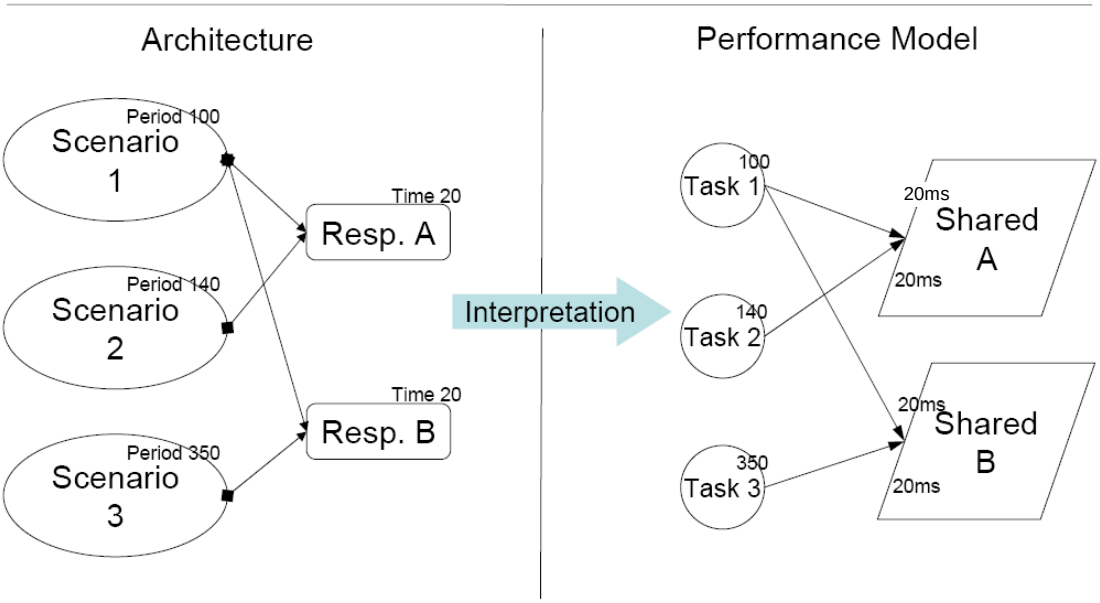


Figura 217. Marco de Razonamiento de Rendimiento: Relación Arquitectura-Modelo [40]

Respecto a las tácticas, hay que resaltar que, a día de hoy, ArchE no implementa tácticas en su marco de razonamiento de rendimiento.

Las únicas tácticas que se pueden utilizar son las acciones directas del arquitecto sobre los siguientes parámetros:

- *Tácticas en requisitos*:
  - Incrementar el período
  - Incrementar la deadline
- *Tácticas de tiempo de ejecución*: reducir el tiempo de ejecución de las responsabilidades.

12.3.2.6 Implementación en ArchE del Marco de Razonamiento a través de Lambda-WBA

Según [41], el marco de razonamiento Lambda-WBA (Worst-Case, Blocking and Asynchrony) predice la latencia en el peor caso como respuesta a un evento. El valor calculado es un límite superior de la latencia porque los efectos de tiempo de ejecución, bloqueo e interrupción del componente de peor caso se asume que ocurren al mismo tiempo.

La teoría subyacente a Lambda-WBA es la Generalized Rate Monotonic Analysis (GRMA), que es una técnica para analizar la planificabilidad de un conjunto de tareas con prioridades variables. Según esta teoría, cada tarea o respuesta está compuesta de una secuencia de subtareas que tienen asociadas un tiempo de ejecución y un nivel de prioridad. Esto hace posible analizar situaciones en las cuales la respuesta a un evento está compuesto de varios cálculos ejecutándose a diferentes prioridades. En este caso, la asignación de prioridades puede estar basada en *deadlines* o en la importancia semántica del componente.

Para evaluar la planificabilidad de tareas de prioridades variables, Lambda-WBA utiliza MAST, que es una herramienta de análisis del caso peor. MAST utiliza su propio lenguaje de entrada para modelos de rendimiento, por tanto los modelos generados por Lambda-WBA tienen que ser traducidos.

La latencia en el peor caso se calcula construyendo la peor alineación posible de efectos de interrupción y bloqueo para cada tarea. El caso peor resultante es un límite superior que puede no darse en la ejecución del sistema si no es probable que los peores efectos sucedan simultáneamente. En la Figura 218 se muestra la salida de MAST para un ejemplo. La latencia en el peor caso se muestra en fondo verde o rojo dependiendo de si la tarea cumple su *deadline* o no, respectivamente. En este caso, las respuestas a clock130 y clock450 no cumplen sus *deadlines*.

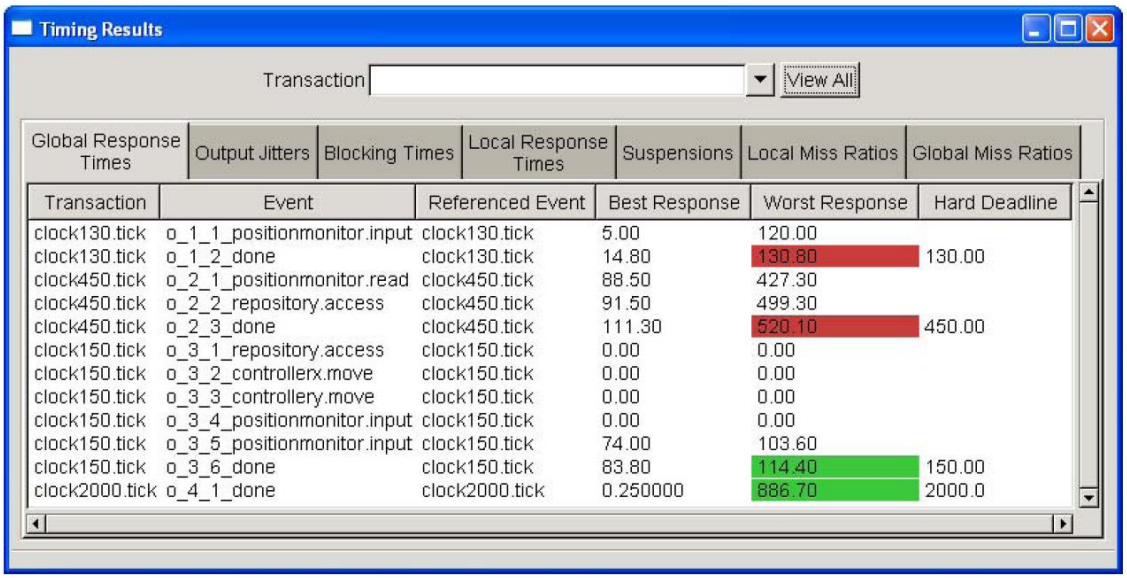


Figura 218. Resultados de la Evaluación Lambda-WBA con MAST. [41]

12.4 Desarrollar ArchE

ArchE, como se ha visto, aporta sólo dos marcos de razonamiento: modificabilidad y rendimiento. Para ampliarlo, simplemente se puede aprovechar las ventajas de la propia arquitectura de ArchE para añadir más marcos de razonamiento.

Según [40], hay dos formas de añadir nuevos marcos de razonamiento: de forma interna o externa: (Figura 219)

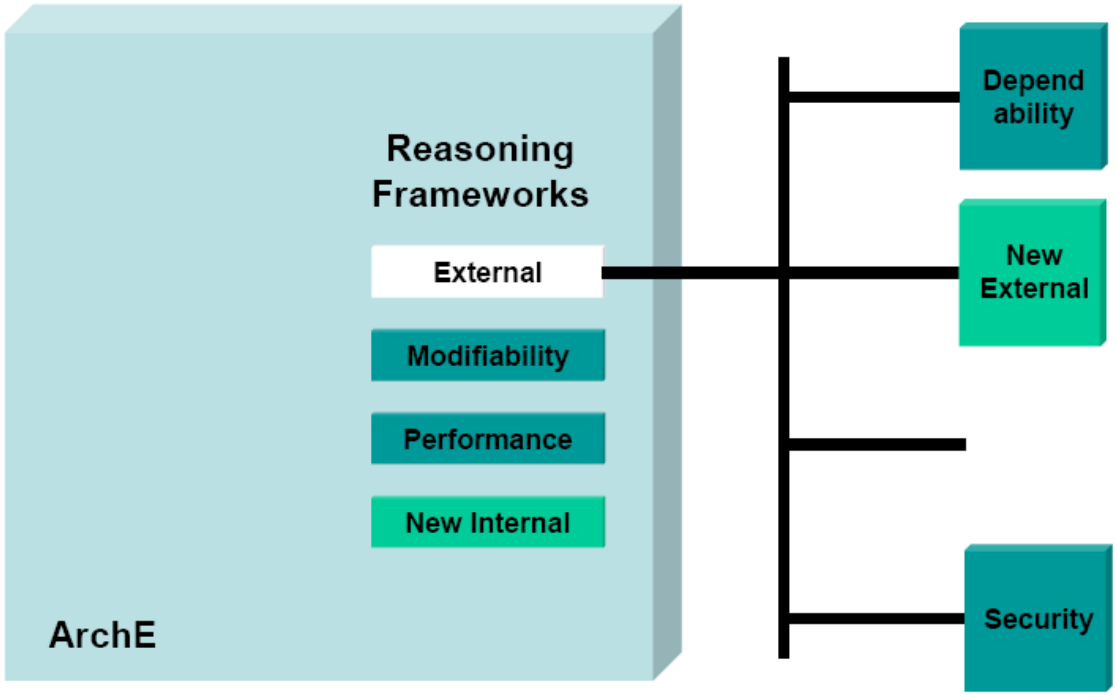


Figura 219. Formas de Añadir nuevos Marcos de Razonamiento en ArchE. [40]

Los marcos de razonamiento pueden ser implementados como *plug-ins* en la arquitectura interna de ArchE. En la Figura 220 se puede ver dicha implementación:

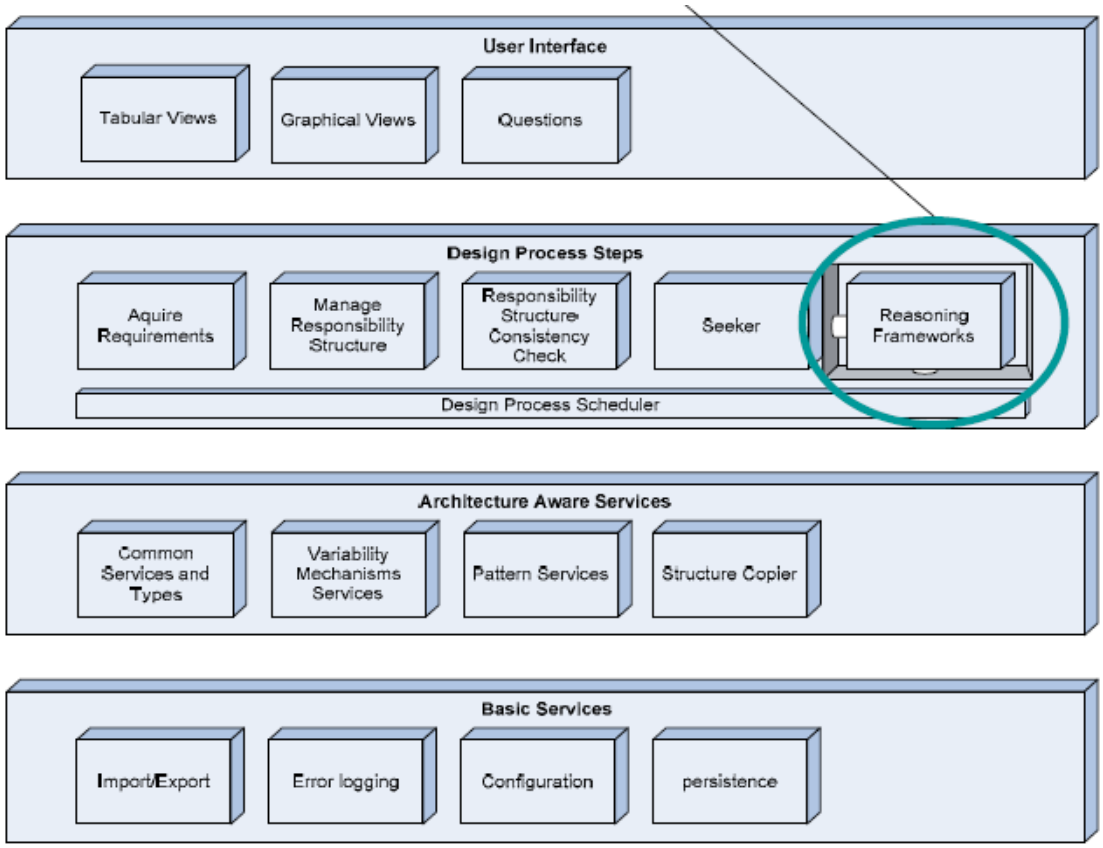


Figura 220. Implementación de *Plug-ins* como Marco de Razonamiento en ArchE. [41]

De hecho, ArchE está construido como un *plug-in* de Eclipse en lo alto de:

- JESS – un motor de reglas
- Java
- MySQL (base de datos)

Un marco de razonamiento se puede implementar en cualquier lenguaje y en cualquier tipo de sistema, y puede ser conectado a una instancia de ArchE via interface XML sobre la red (Figura 221); la interface ArchE-RF es una infraestructura colaborativa que permite desarrollar dichos marcos de razonamiento: [40], [44]

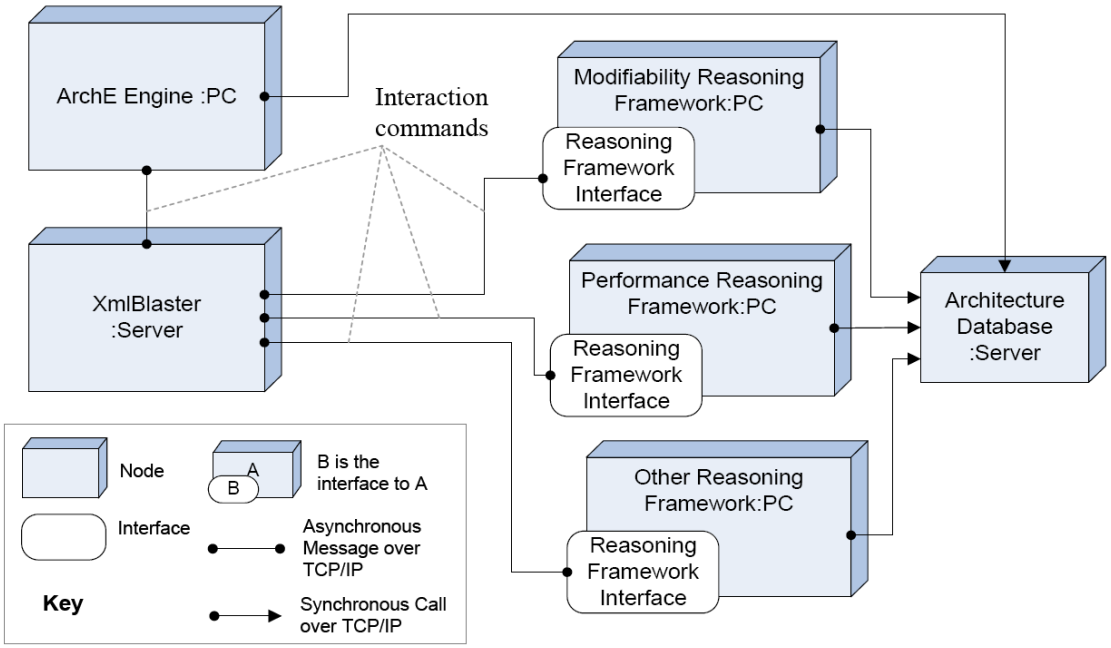


Figura 221. Implementación de Marco de Razonamiento via Servidor XMLBlaster. [44]

Según [44], esta interface Arche-RF es una infraestructura colaborativa que tiene los siguientes elementos constitutivos:

- ArchE Engine: motor Arche con poco conocimiento de técnicas de diseño de atributos de calidad o semánticas acerca del sistema que es diseñado. Sus responsabilidades son: procesamiento de entradas del usuario, actualización de los paneles GUI, analizador sintáctico del "manifestó" de los marcos de razonamiento, coordinación de los marcos de razonamiento, presentación de sus resultados y mostrar en pantalla las preguntas para el usuario.
- XmlBlaster: middleware orientado a mensajes donde pueden tener lugar invocaciones a mensajes implícitos entre diversos participantes sobre una red.
- *Reasoning Framework Interface*: interfaz con los marcos de razonamiento, que abstrae los detalles de trabajar con XmlBlaster, el protocolo de comunicaciones entre ArchE y el marco de razonamiento, y también los algoritmos que ejecutan los comandos de interacción.
- *Architecture Database*: repositorio que se usa para gestionar datos persistentes que necesitan ser compartidos por ArchE y todos los marcos de razonamiento

participantes. Por ejemplo, las arquitecturas originales y candidatas se almacenan aquí.

Según [42], la interface ArchE-RF proporciona técnicas y clases por defecto para desarrollar marcos de razonamiento sin tener que conocer los mecanismos internos de comunicación entre ArchE y los marcos de razonamiento. En la Figura 222 se muestran los diferentes pasos a seguir para desarrollar nuevos marcos de razonamiento: [42]

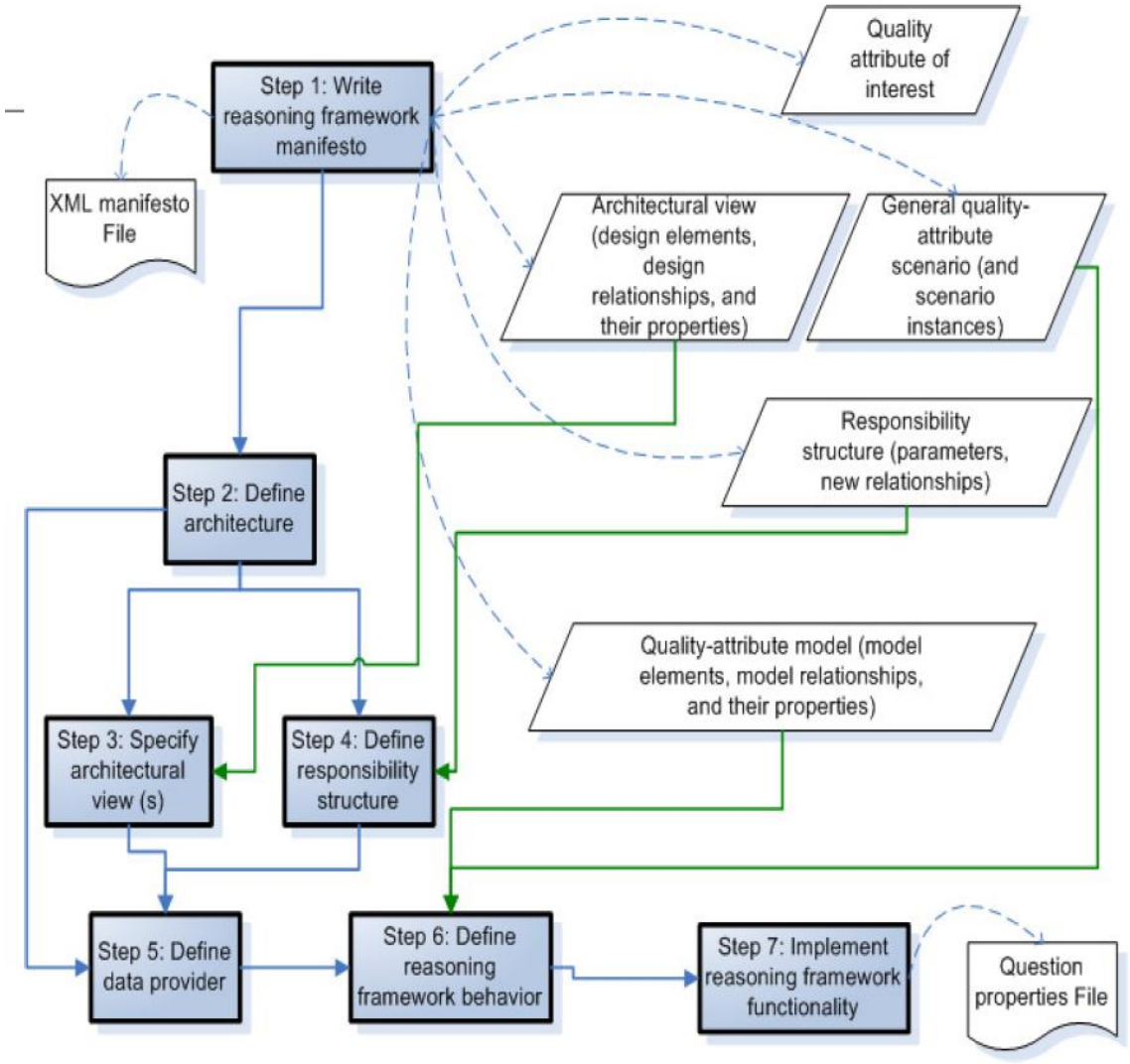


Figura 222. Pasos a seguir en el Desarrollo de un Marco de Razonamiento para ArchE. [42]

Según [44], un marco de razonamiento es una entidad modular que proporciona la capacidad de razonar sobre el comportamiento específico de un atributo de calidad de una arquitectura. Para razonar sobre una arquitectura, en marco de razonamiento necesita de un modelo de análisis, para dar soporte a las tres diferentes fases del análisis:

- Interpretación: el procedimiento de mapeo que convierte el modelo arquitectónico en el modelo de análisis

- Evaluación: el procedimiento usado para resolver el modelo de análisis y calcular las medidas de los atributos de calidad para los escenarios, para determinar si la arquitectura actual satisface dichos escenarios.
- Rediseño: en caso de que algunos escenarios no se cumplan, las tácticas permiten ajustar la estructura/comportamiento al modelo de arquitectura actual.

Para desarrollar dicho marco de razonamiento, se necesitan desarrollar ciertos conceptos arquitectónicos, recogidos en la Figura 223: [44]

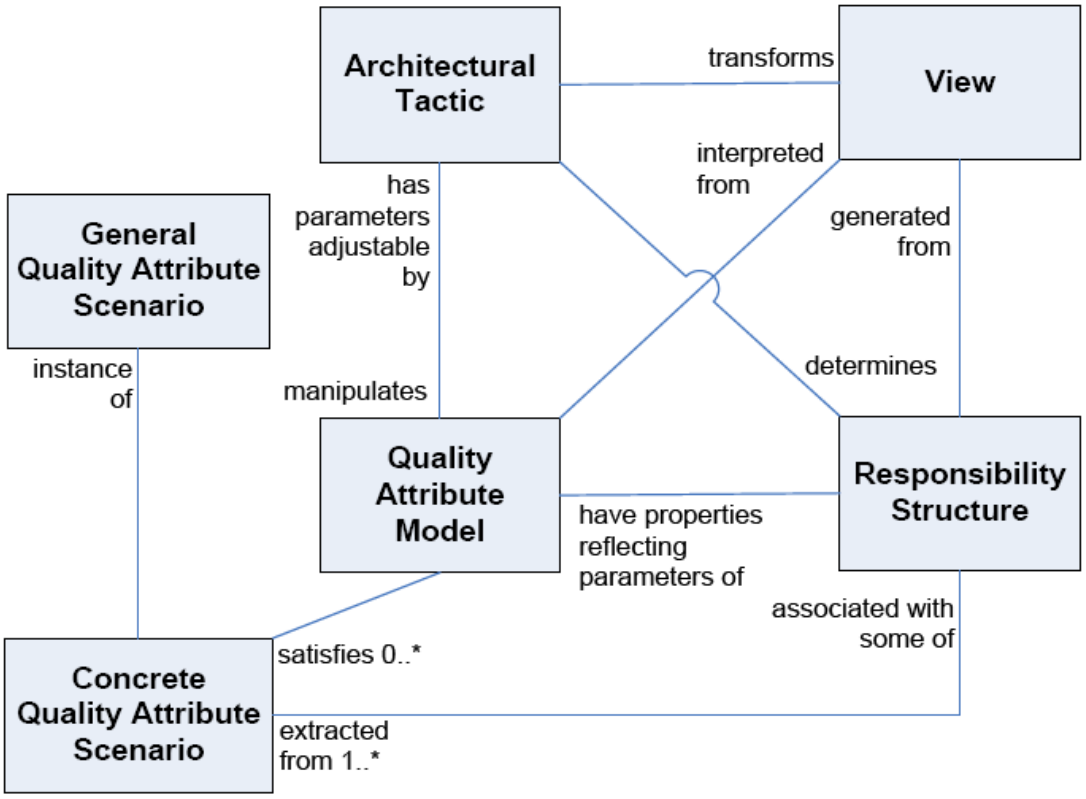


Figura 223. Arquitectura Conceptual Interna de un Marco de Razonamiento. [44]

- Escenario de Atributos de Calidad General: contiene una tabla independiente del sistema para los requisitos de atributos de calidad, basada en el esquema estimulo-fuente del estimulo-entorno-artefacto-respuesta-medida de la respuesta.
- Escenario Concreto de Atributos de Calidad: requisito específico del sistema que es una instancia de un escenario general.
- Modelo de atributos de calidad: el resultado de interpretar un diseño de arquitectura con una teoría analítica.
- Responsabilidades: actividades llevadas a cabo por el software que está siendo diseñado.
- Táctica arquitectónica: un medio para satisfacer una medida de respuesta a un atributo de calidad mediante la modificación de algunos aspectos del modelo de atributos de calidad, a través de decisiones de diseño.

- Vista arquitectónica: una estructura de diseño del sistema que puede ser vista desde determinado punto de vista, como un grafo compuesto por elementos de diseño arquitectónicos.

En la Figura 224 se muestran las diferentes interacciones entre el arquitecto, Arche, y los marcos de razonamiento. [42]

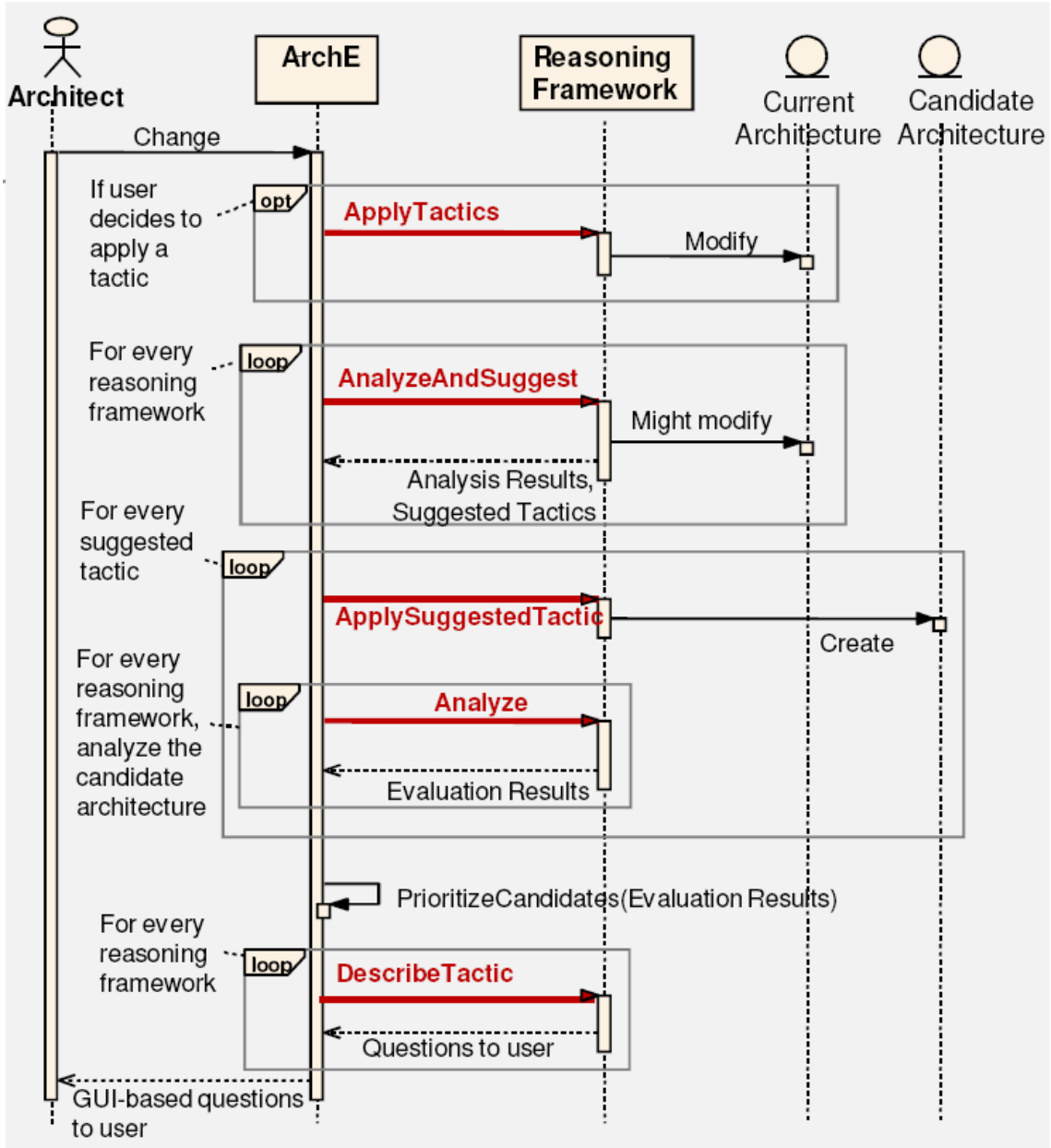


Figura 224. Interacciones entre ArchE y los Marcos de Razonamiento. [42]

Los cinco comandos se describen en [44] y son:

- *ApplyTactics*: solicita a un marco de razonamiento específico aplicar una táctica a la arquitectura actual para refinarla.
- *AnalyzeAndSuggest*: solicita a un marco de razonamiento analizar la arquitectura actual respecto a los escenarios de interés, y a sugerir nuevas tácticas si algunos escenarios no se cumplen.

- *ApplySuggestedTactic*: solicita a un marco de razonamiento aplicar una táctica a la arquitectura actual para crear una nueva arquitectura candidato.
- *Analyze*: solicita a un marco de razonamiento analizar una arquitectura candidato respecto a los escenarios de interés.
- *DescribeTactic*: solicita a un marco de razonamiento proporcionar a ArchE con las preguntas para los usuarios que describen las tácticas o cualquier otra recomendación.

Cuando se implementa la interface ArchE-RF, se supone que un marco de razonamiento va a implementar seis funcionalidades básicas, según [44], y son:

- Autodescripción (Manifesto)
- Crear diseño inicial
- Analizar (para los commandos *Analyze* y *AnalyzeAndSuggest*)
- Sugerir tácticas (para el comando *AnalyzeAndSuggest*)
- Aplicar tácticas (para los comandos *ApplyTactic* y *ApplySuggestedTactic*)
- Describir tácticas (para el comando *DescribeTactic*)

La ArchE-RF Interface API proporciona una guía para implementar internamente los marcos de razonamiento. Según [44], dicha API está estructurada en cuatro capas, cada una de las mismas proporciona servicios para las capas superiores:

- Capa de comunicaciones: la capa de más alto nivel, que incluye todas las clases e interfaces para interactuar con ArchE via XmlBlaster.
- Capa de ejecución: contiene un conjunto de algoritmos cada uno de los cuales procesa un comando de interacción reenviado desde la capa de comunicación.
- Capa de marco de razonamiento: proporciona la clase *ArchEReasoningFramework*, la cual ha de ser extendida por un desarrollador para implementar un marco de razonamiento específico.
- Capa de datos: es la capa de más bajo nivel, y proporciona a las capas más altas los conceptos de la Figura 224. Incluye las interfaces Java requeridas para gestionar los conceptos calves, los cuales deben ser mapeados a clases concretas y tablas de bases de datos.

13 ANEXO 2 – INSTALACIÓN Y CONFIGURACIÓN DE ARCHE

13.1 Introducción

En este anexo 2 se va a explicar cómo realizar la instalación de ArchE, así como todos sus complementos, necesarios para que la herramienta funcione correctamente.

ArchE fue instalado en una máquina virtual con el sistema operativo Windows XP usando Oracle VM VirtualBox v4.3.2.

Las aplicaciones instaladas fueron las siguientes:

1. Instalación de Java SDK → jdk-1_5_0_22-windows-i586-p.exe
2. Instalación de Eclipse → Eclipse-SDK-3.3-win32.zip
3. Instalación de MySQL → Mysql-essential-5.0.67-win32.msi
4. Instalación de GEF → GEF-runtime-3.3.zip
5. Instalación de JESS → Jess71p1.zip
6. Instalación de xmlBlaster → xmlBlaster_REL_1_6_1.zip
7. Instalación de ArchE → ArchE_3_0_0_Installer.exe

Esta guía de instalación de ArchE está basada en la guía ya realizada por un compañero del Máster en Investigación en Ingeniería Software y Sistemas Informáticos, D. Carlos Alberto Gaitán Peña, que realizó este mismo Trabajo Fin de Máster [48] en el curso 2012/2013. Esta guía viene a complementar el excelente trabajo realizado por este compañero, incluyendo algunos pasos más para que la instalación resulte aún si cabe más fácil de comprender.

13.2 Instalación del Java runtime 5.0

Se instala Java SDK 1.5.0, tal y como se ve en la Figura 225

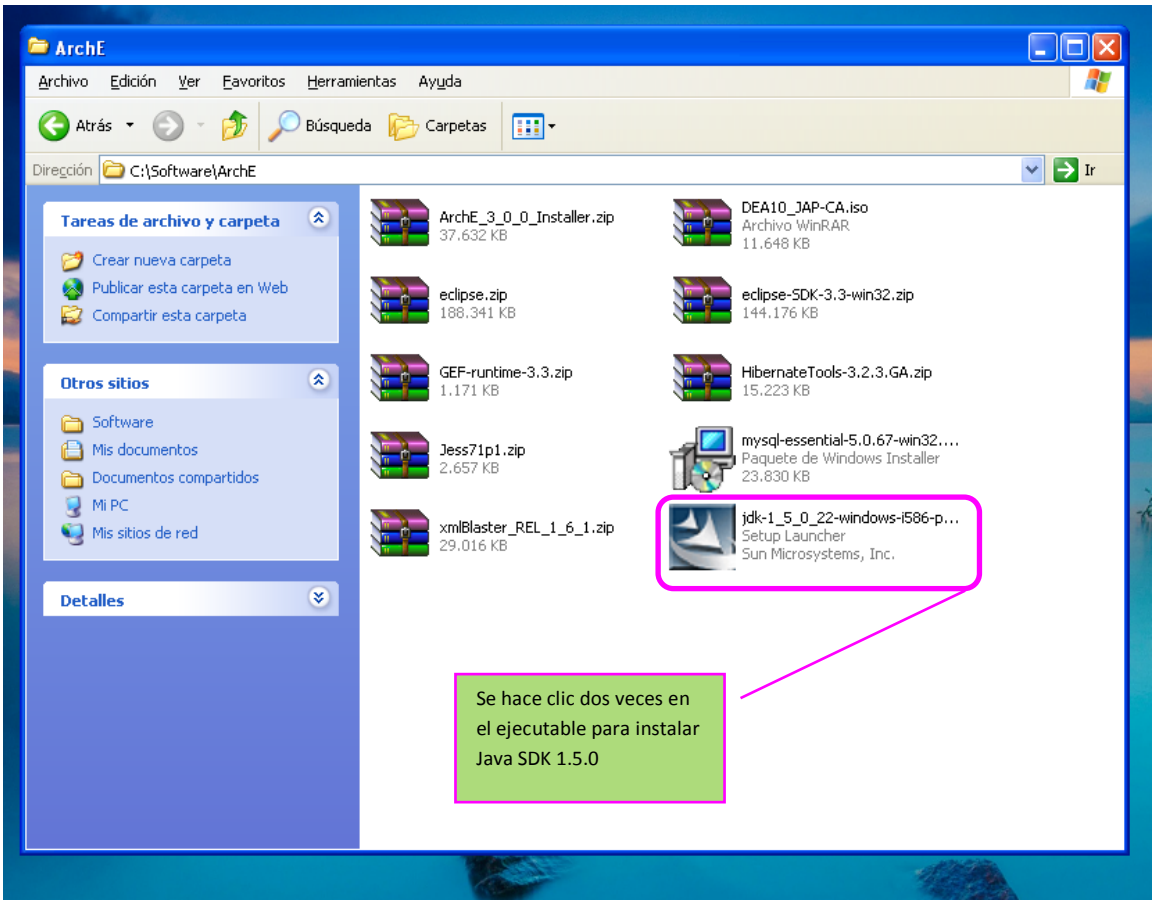


Figura 225. Instalación de Java SDK 1.5.0.

A continuación se muestran los paquetes a instalar. Se presione “Next >”: (Figura 226)

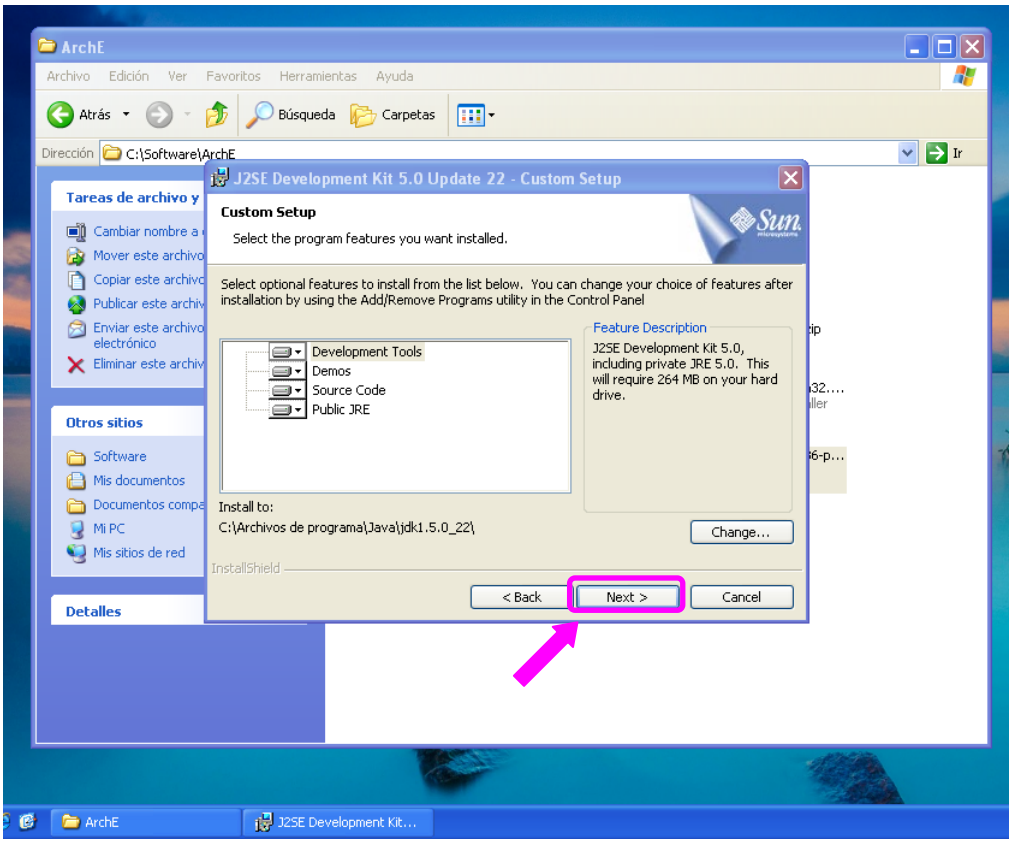


Figura 226. Opciones de Java SDK.

Se selecciona J2SE Runtime Environment y se presiona Next >: (Figura 227)

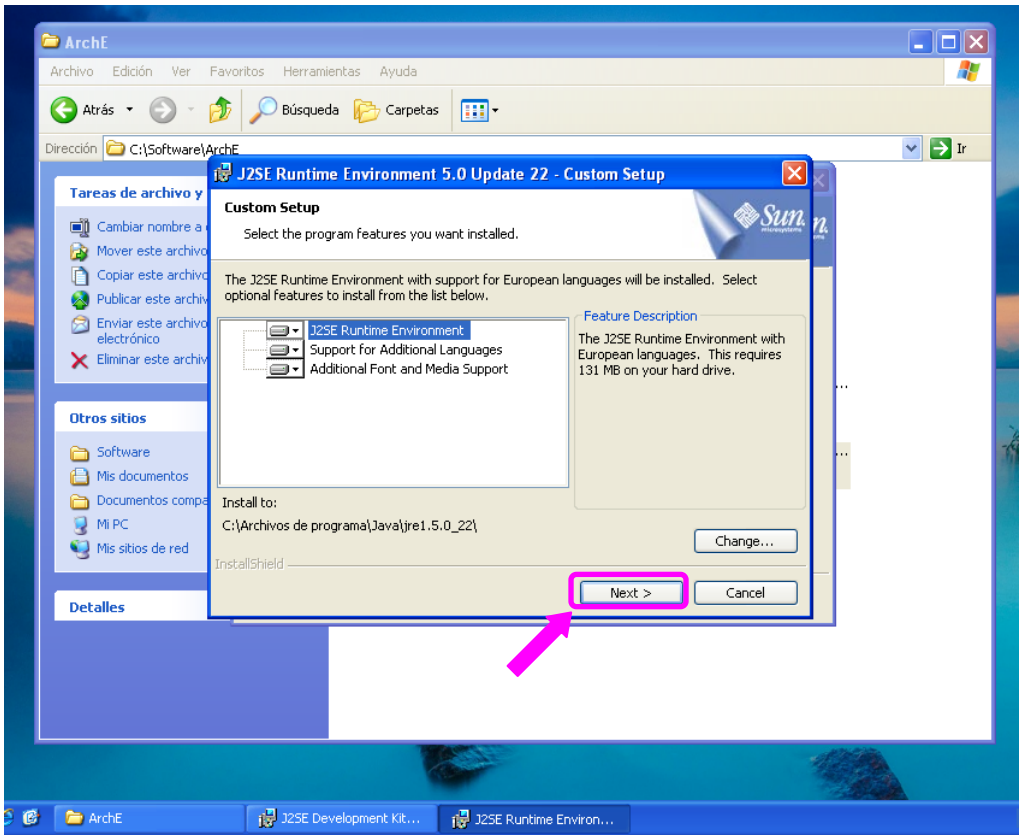


Figura 227. Instalación de J2SE.

Es importante destacar que no se deben de modificar los directorios de instalación de JDK ni de JRE. A continuación, se muestra una ventana cuando la instalación finaliza. Se presiona “Finish”: (Figura 228)

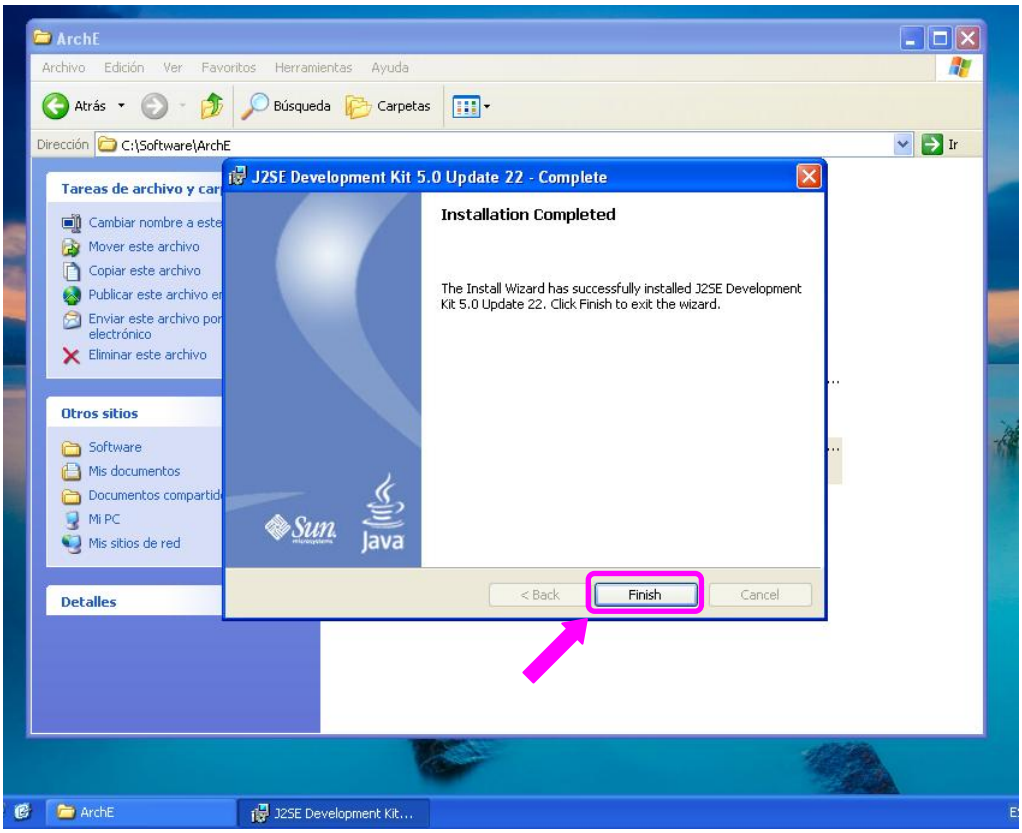


Figura 228. Instalación de J2SE Completada.

Una vez se ha instalado Java en el sistema, se han de configurar las variables del sistema, creando primero la variable JAVA_HOME con su valor: (Figura 229)

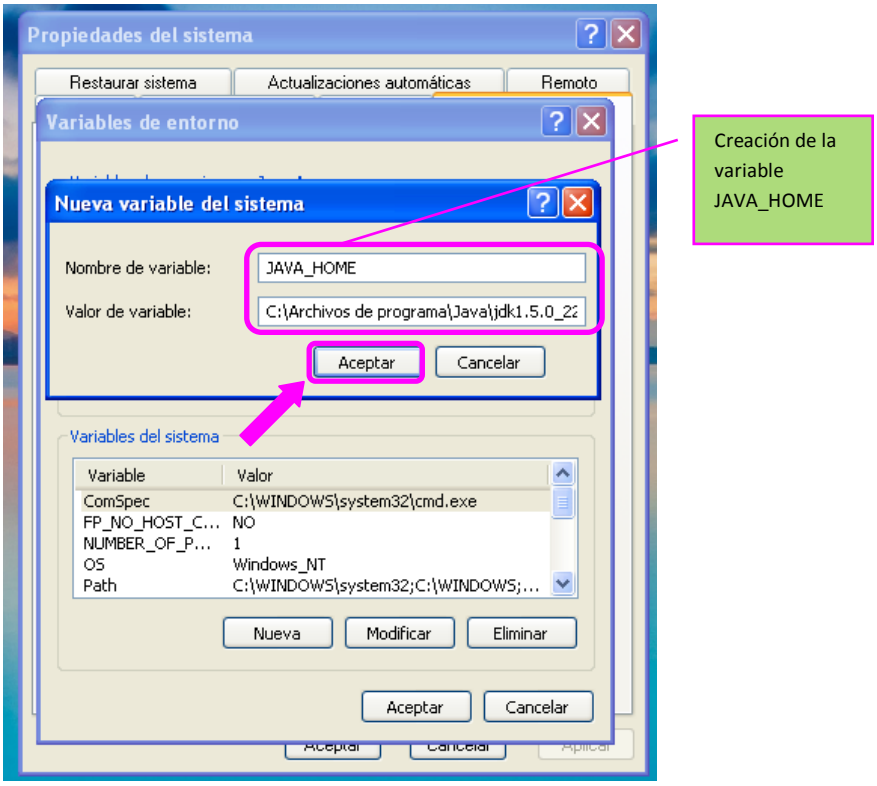


Figura 229. Creación de la Variable JAVA_HOME.

A continuación se modifica la variable de entorno Path, para permitir acceder a jdk desde cualquier directorio: (Figura 230 y Figura 231)

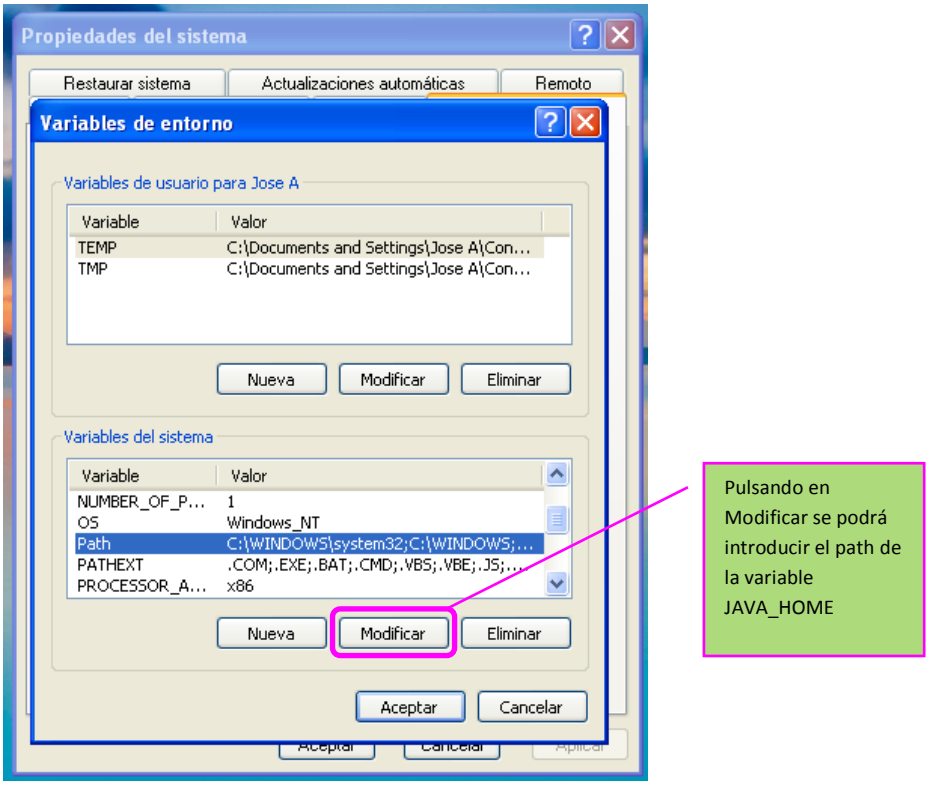


Figura 230. Modificación de la Variable Path para introducir JAVA_HOME.

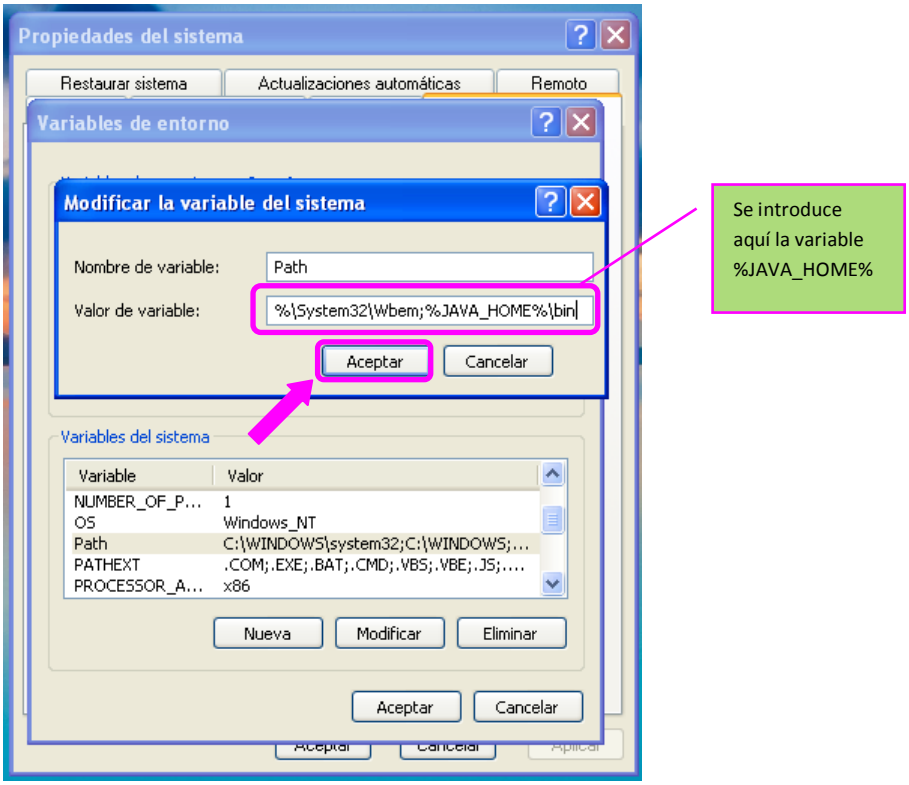


Figura 231. Introducción de la variable JAVA_HOME en Path.

Por último, se comprueba que se ha añadido bien la variable, ejecutando en la línea de comandos el comando “path”, que devolverá toda la ruta: (Figura 232)

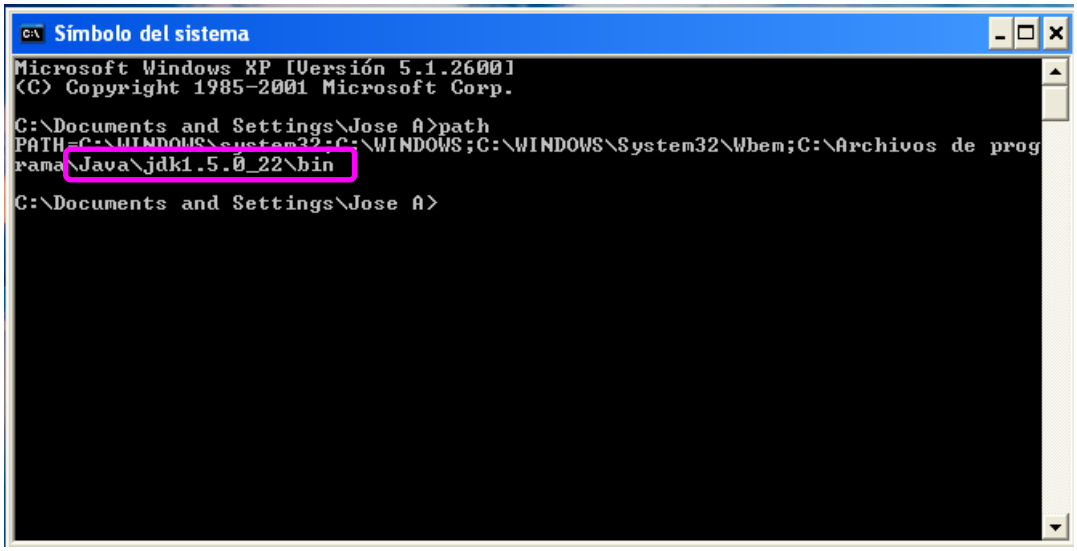


Figura 232. Comprobación en Línea de Comandos de la Variable JAVA_HOME.

13.3 Instalación de Eclipse

Se descomprime la carpeta eclipse en c:\ (Figura 233)

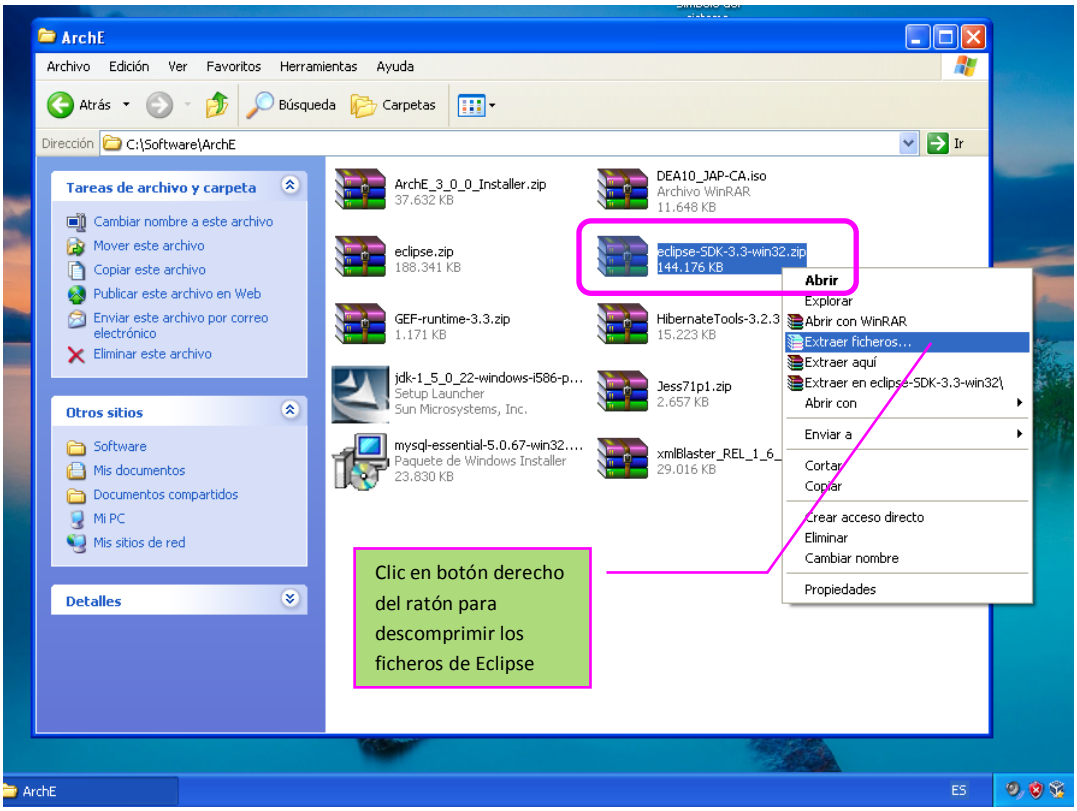


Figura 233. Instalar Eclipse Descomprimiendo los ficheros.

Una vez descomprimidos, se ejecuta el archivo eclipse.exe para iniciar la aplicación: (Figura 234 y Figura 235)

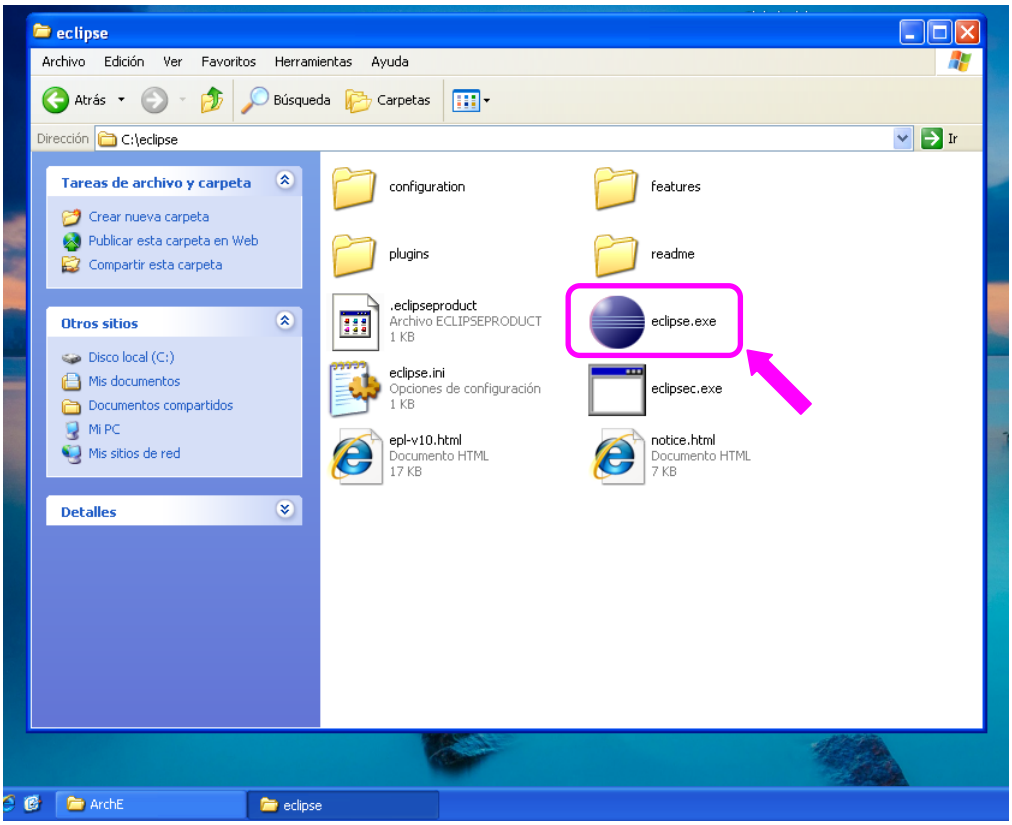


Figura 234. Arrancar Eclipse.

Figura 235. Eclipse se inicia con el entorno Java.

Una vez iniciado, el entorno Eclipse presenta este aspecto: (Figura 236)

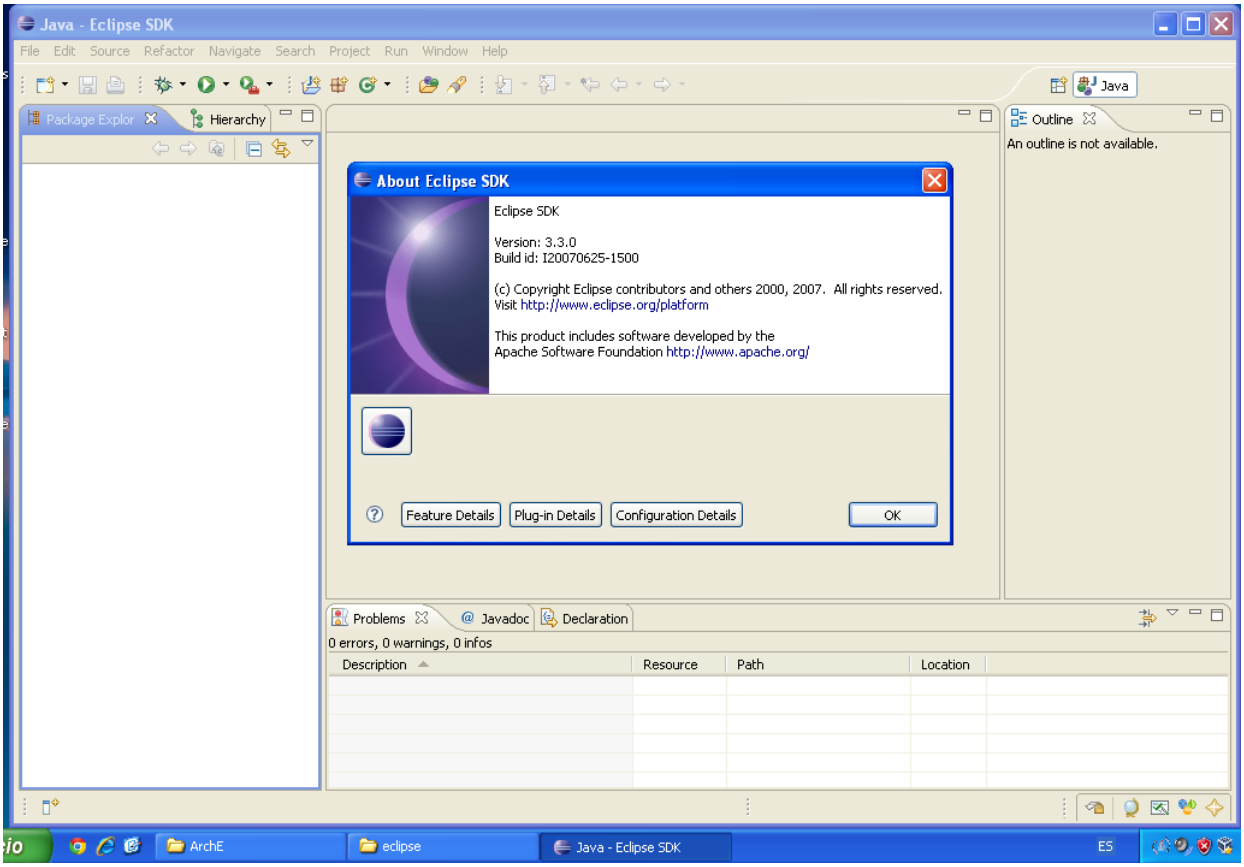


Figura 236. Entorno de Eclipse.

13.4 Instalación de MySQL

Para instalar MySQL, se ejecuta el archivo de instalación mysql-essential-5.0.67-win32.msi: (Figura 237)

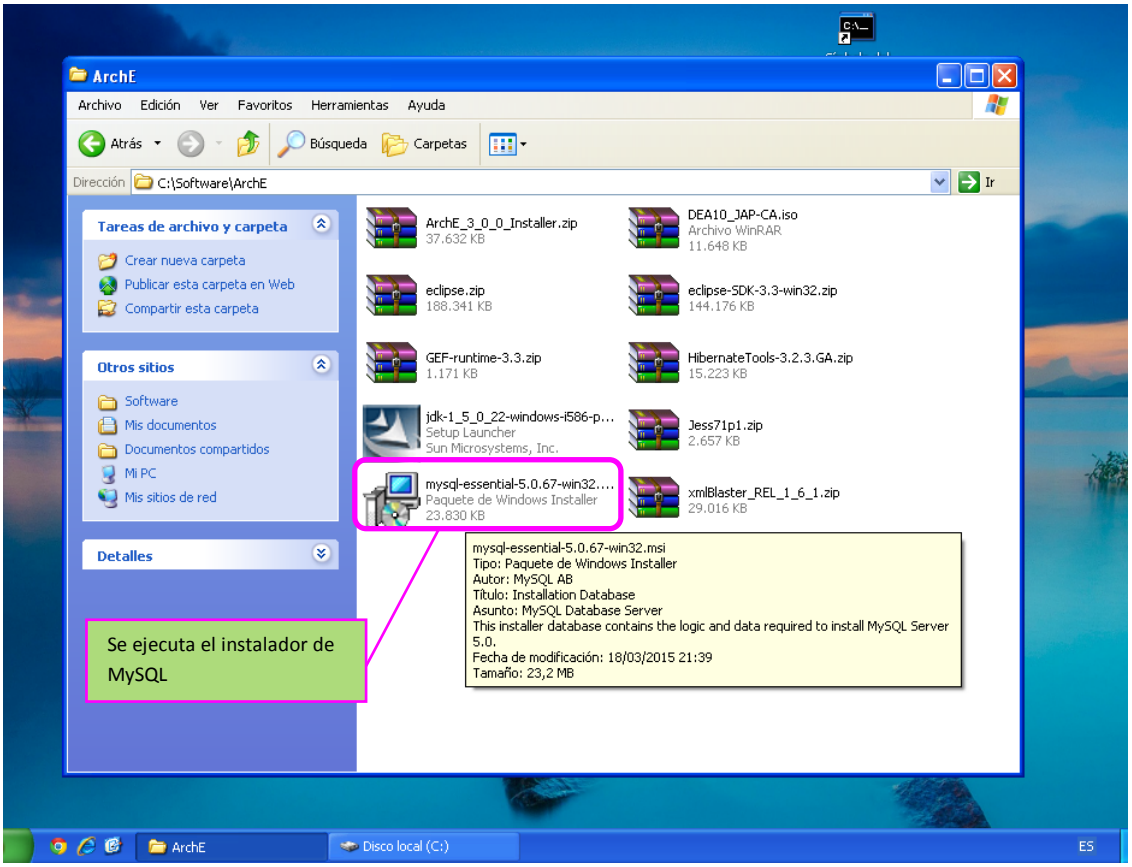


Figura 237. Instalador de MySQL.

Se abre la ventana del configurador de MySQL. Se presiona Next >: (Figura 238)

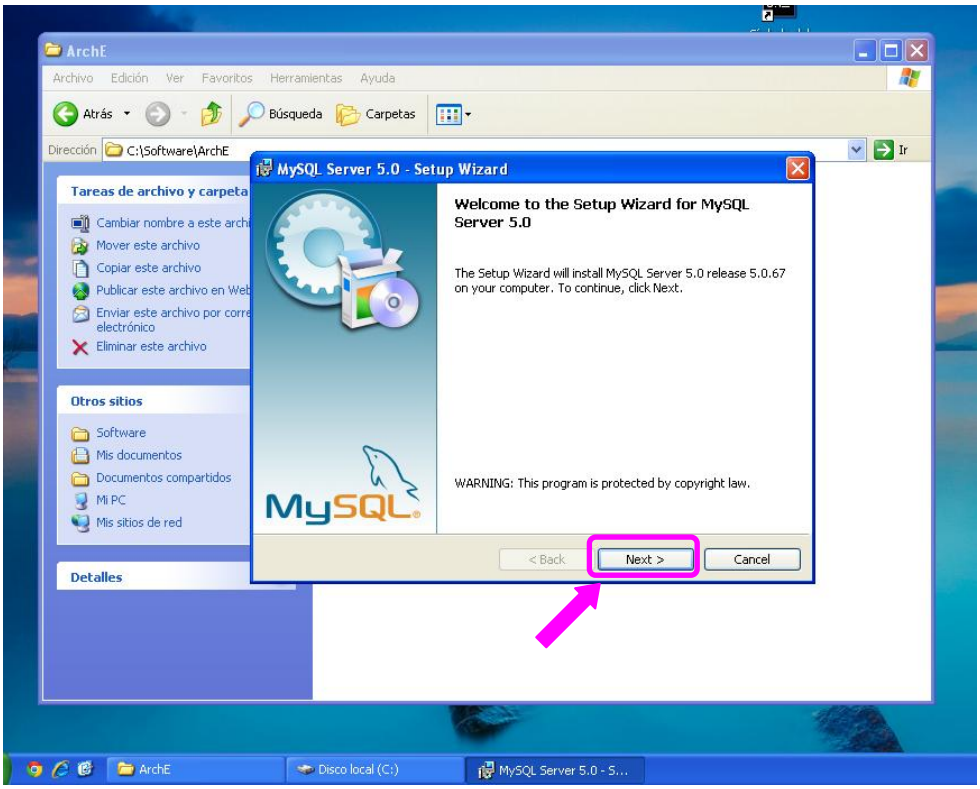


Figura 238. Ventana Inicial del Configurador de MySQL.

Se elige la opción de instalación completa y se pulsa Next >: (Figura 239)

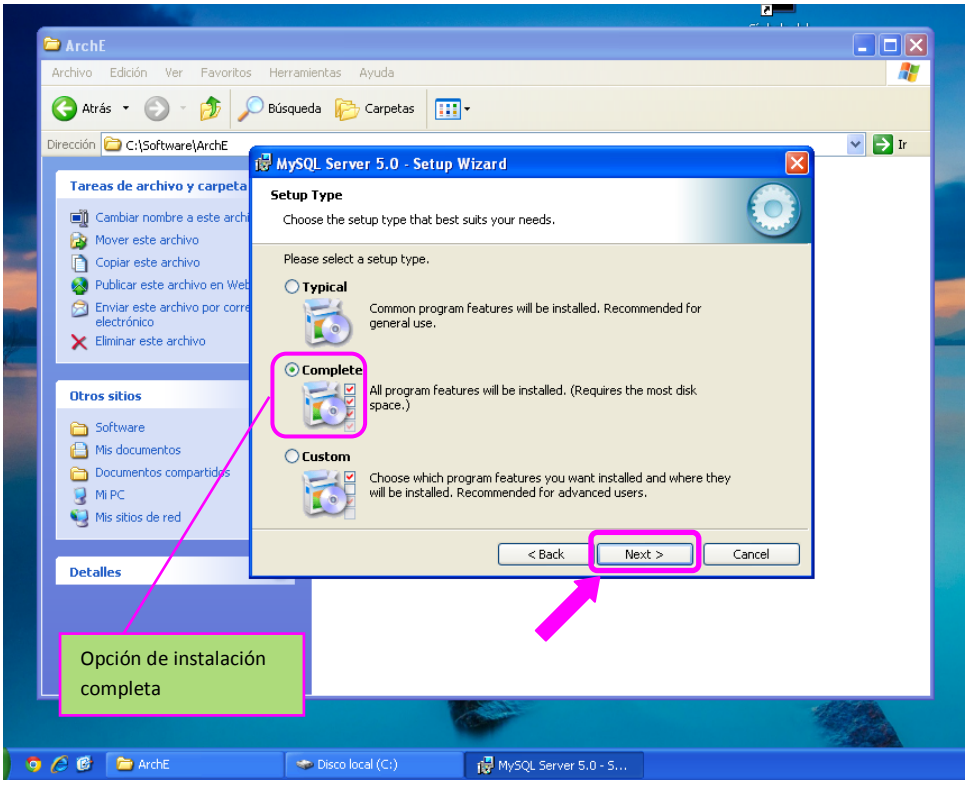


Figura 239. Opción de Instalación Completa para MySQL.

El configurador está listo para instalar MySQL. Se presiona Install: (Figura 240)

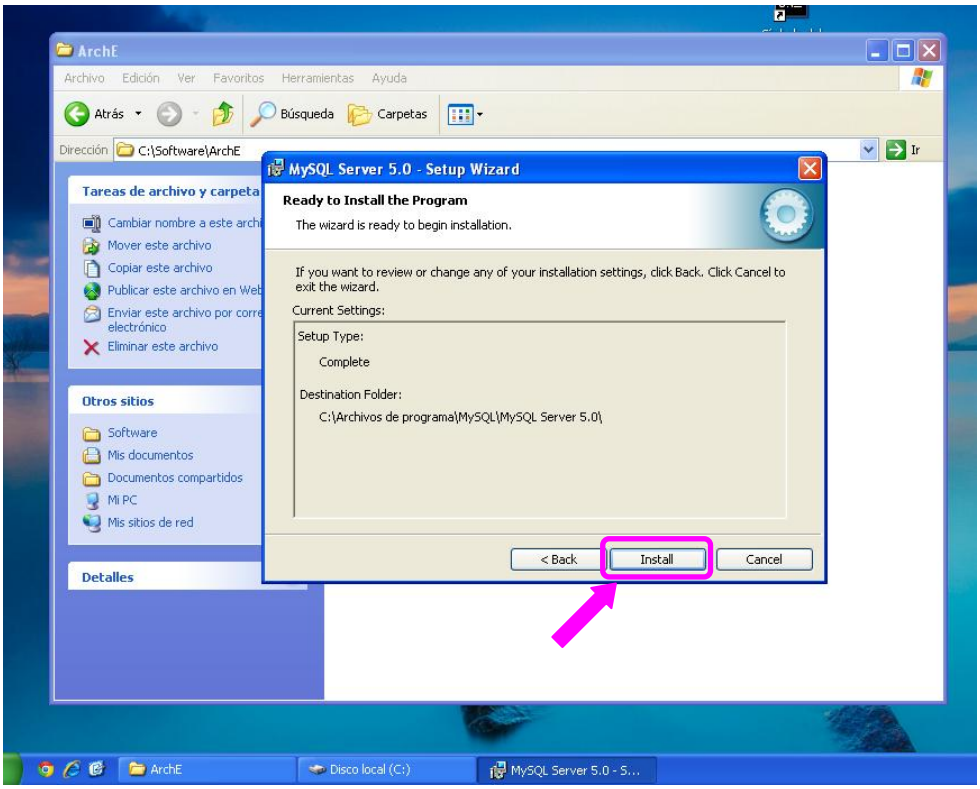


Figura 240. Preparada la Instalación de MySQL.

La instalación de MySQL está en progreso: (Figura 241)

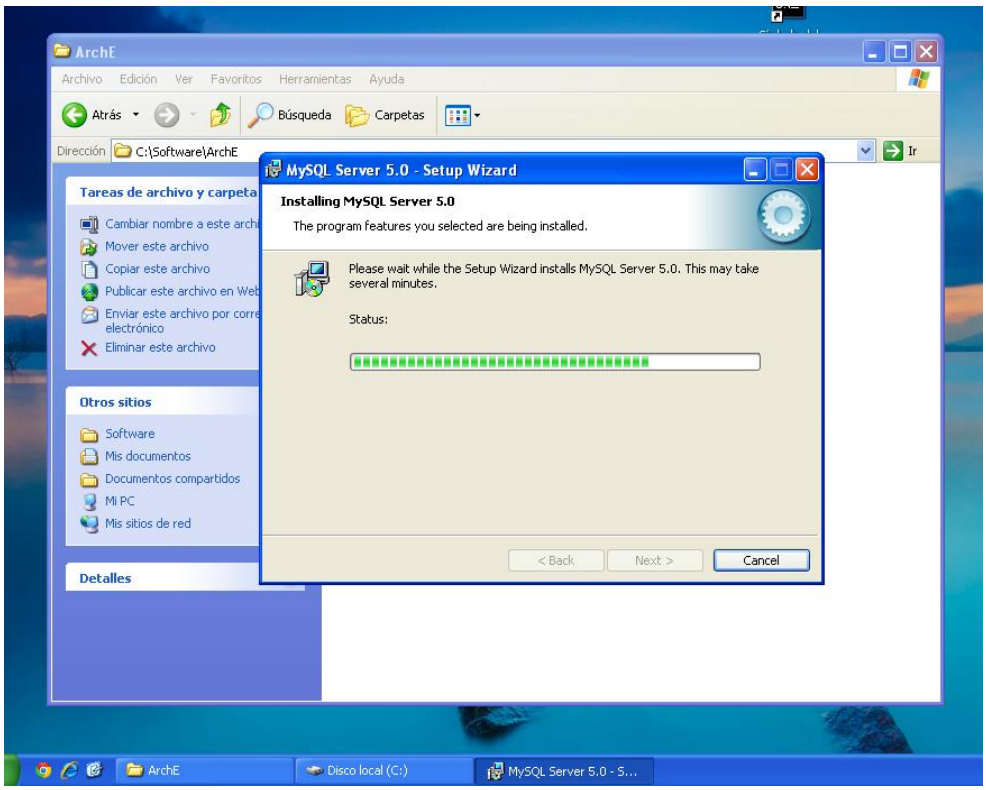


Figura 241. Instalación de MySQL en Progreso.

Una vez finalizada la instalación, se muestra la siguiente ventana: (Figura 242)

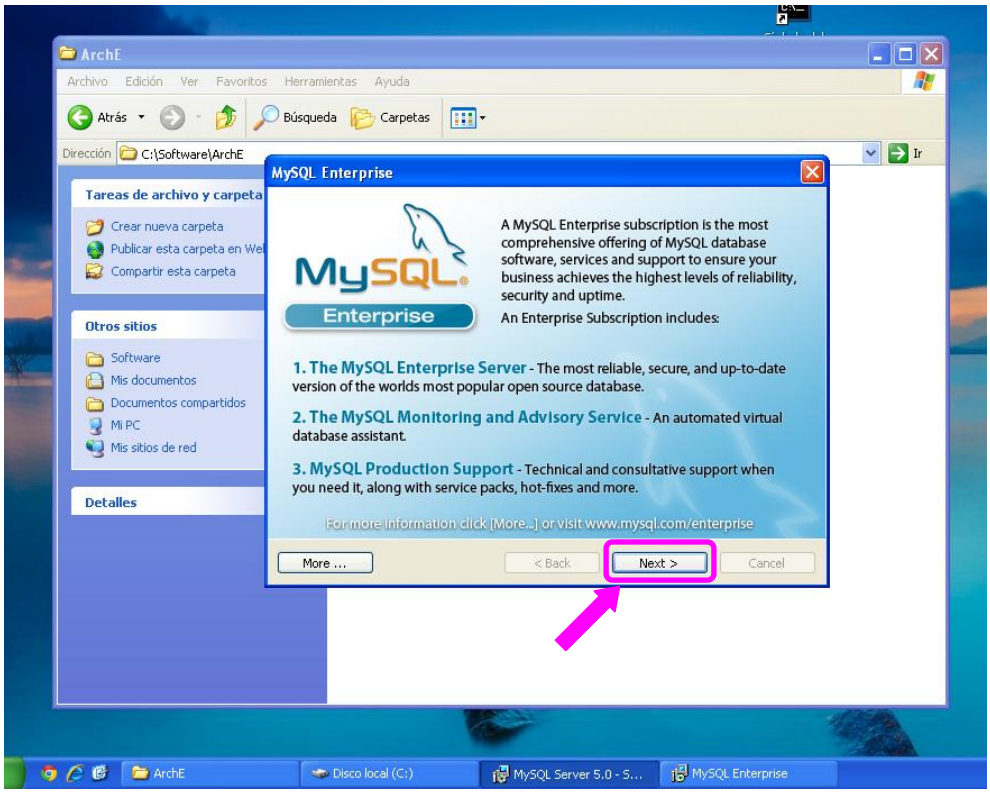


Figura 242. MySQL está instalado.

Se pulsa Next > y después Finish para salir del configurador del instalador: (Figura 243)

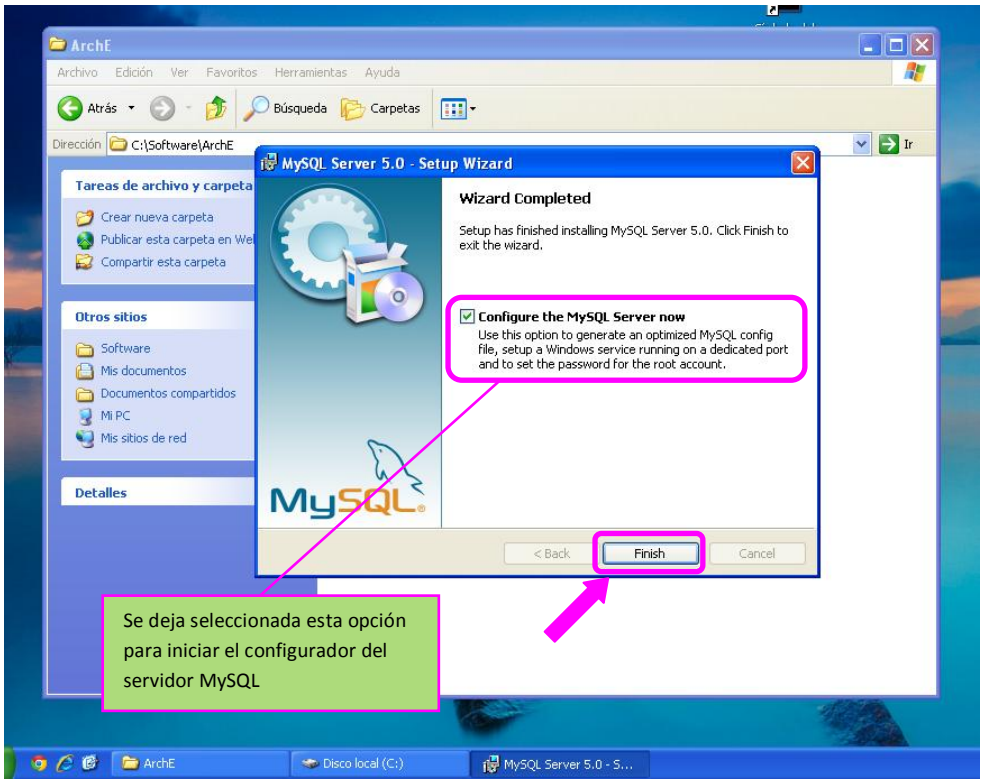


Figura 243. Finalizar Instalación de MySQL e Iniciar Configuración del Servidor.

Se muestra la ventana del configurador de la instancia de MySQL Server. Pulsar Next >: (Figura 244)

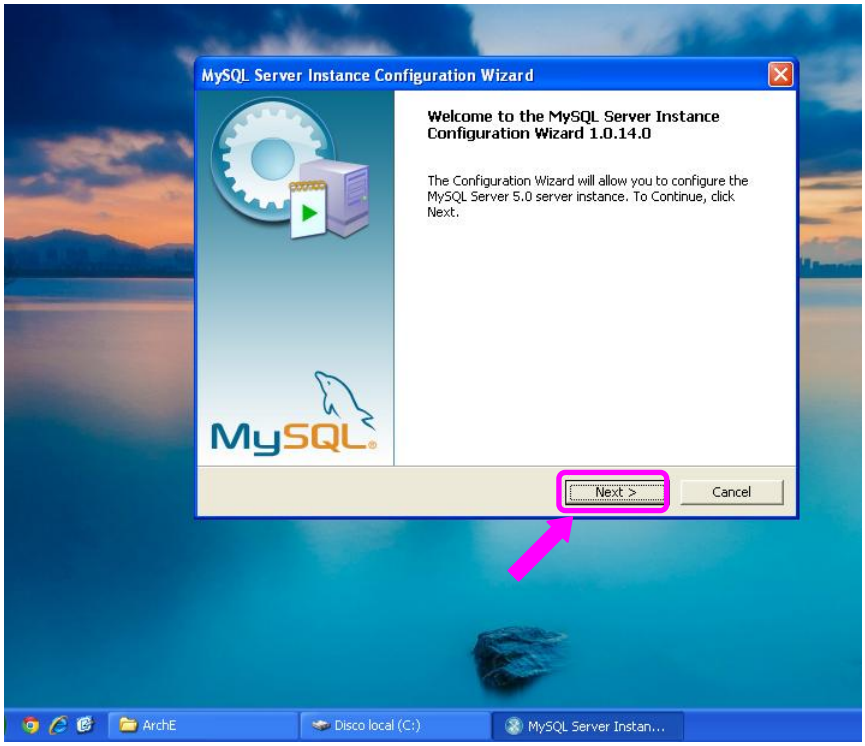


Figura 244. ventana del configurador de la instancia de MySQL Server

Seleccionar Configuración detallada y pulsar Next >: (Figura 245)

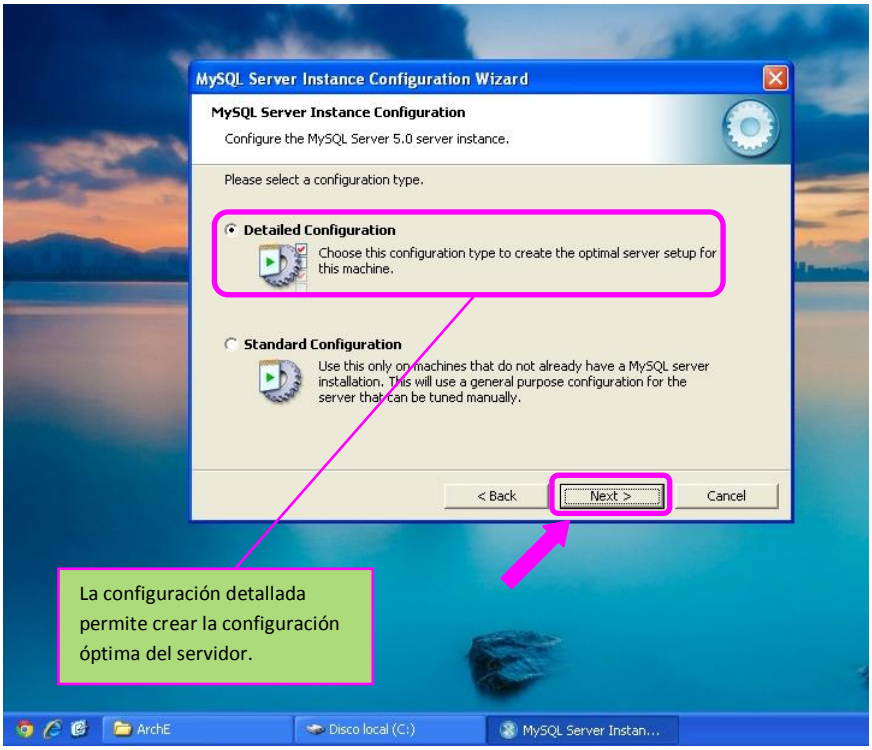


Figura 245. Selección de Configuración Detallada del Servidor MySQL.

En las siguientes ventanas, se seleccionan las opciones de Developer Machine y Multifunctional Database: (Figura 246 y Figura 247)

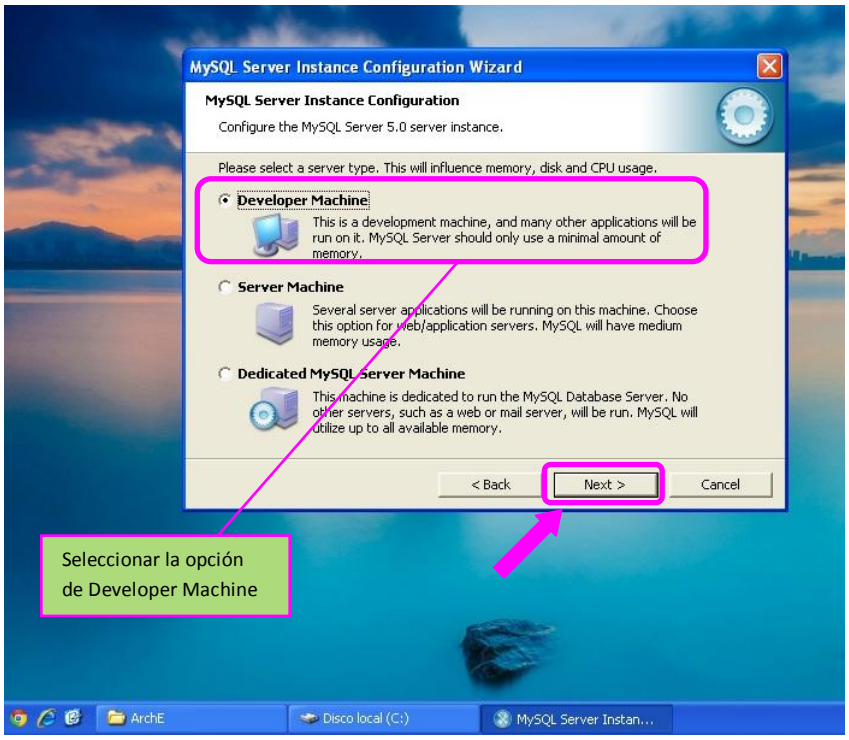


Figura 246. Selección de Developer Machine – MySQL.

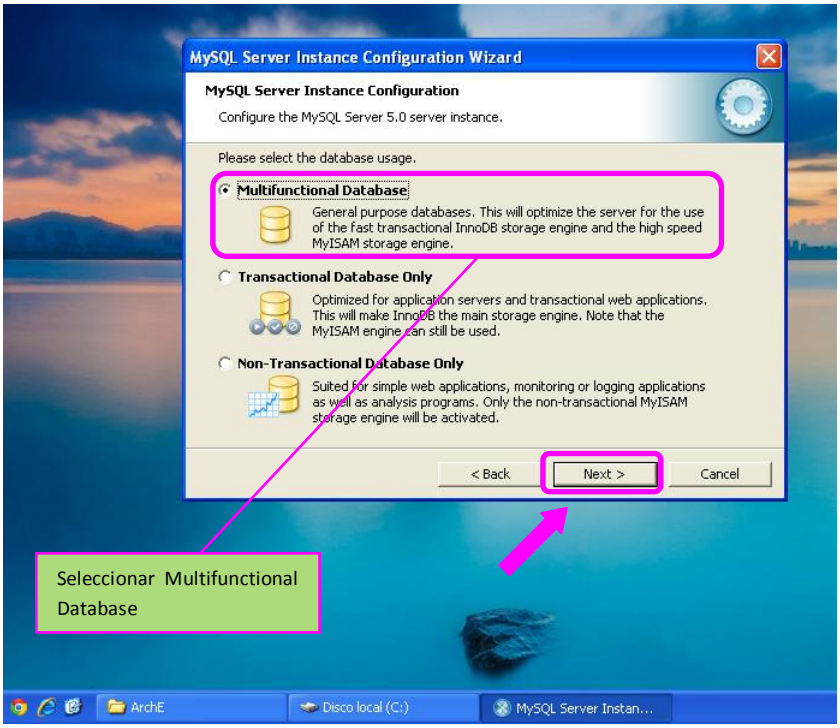


Figura 247. Selección de Multifunctional Database – MySQL

Se selecciona el disco u directorio para la base de datos y se pulsa Next >: (Figura 248)

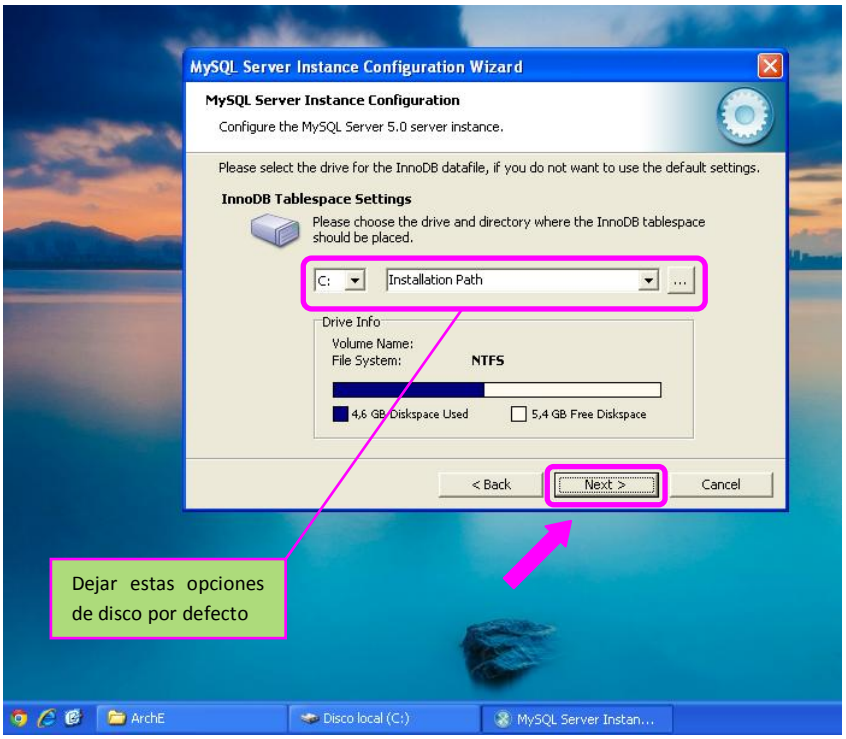


Figura 248. Selección de Disco para almacenar Base de Datos - MySQL.

Se seleccionan ahora el número de conexiones concurrentes al servidor: (Figura 249)

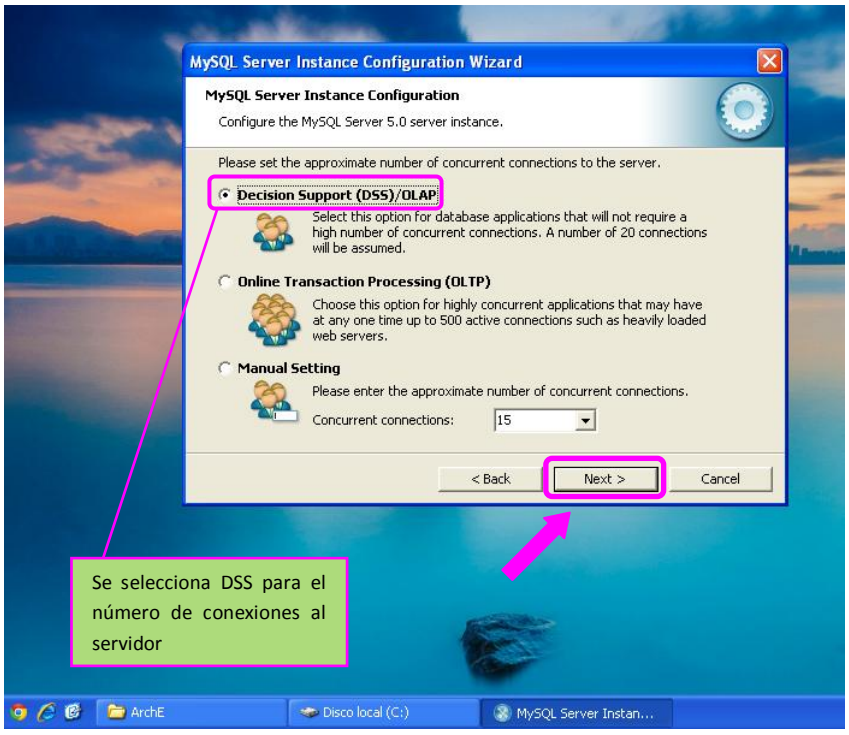


Figura 249. Selección de Número de conexiones Concurrentes al Servidor – MySQL.

Se seleccionan ahora las opciones de red y se pulsa Next >: (Figura 250)

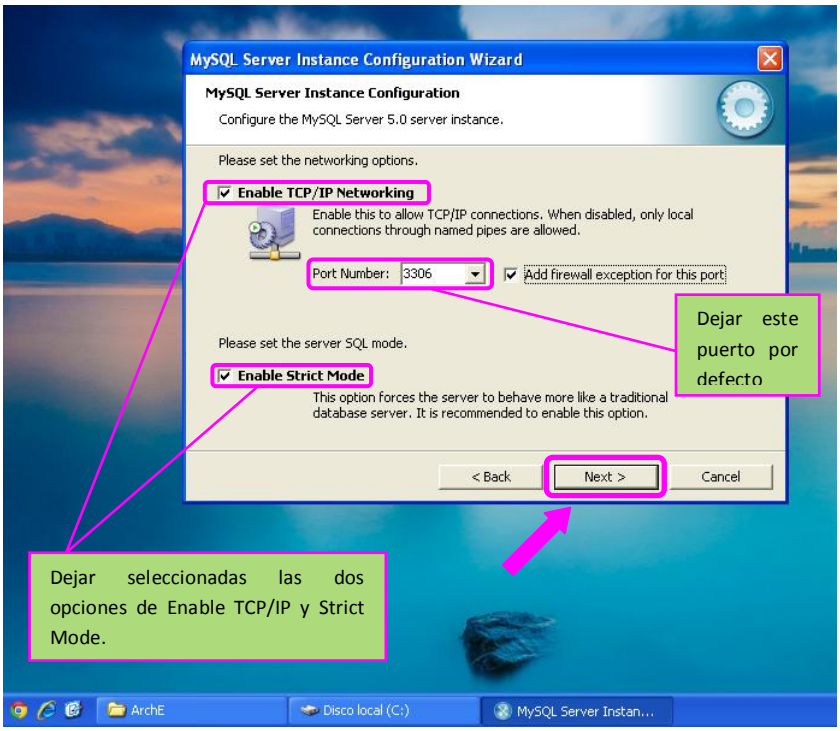


Figura 250. Selección de Opciones de Red – MySQL.

Se selecciona ahora el conjunto de caracteres por defecto y se pulsa Next >: (Figura 251)

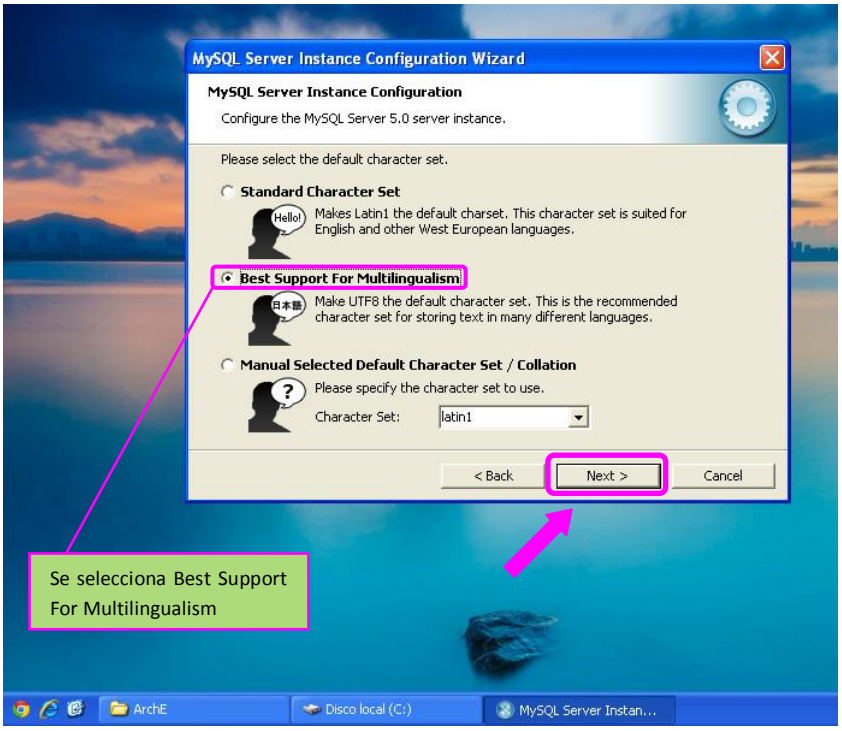


Figura 251. Selección de Conjunto de Caracteres por Defecto – MySQL.

Se seleccionan ahora las opciones para Windows y se pulsa Next >: (Figura 252)

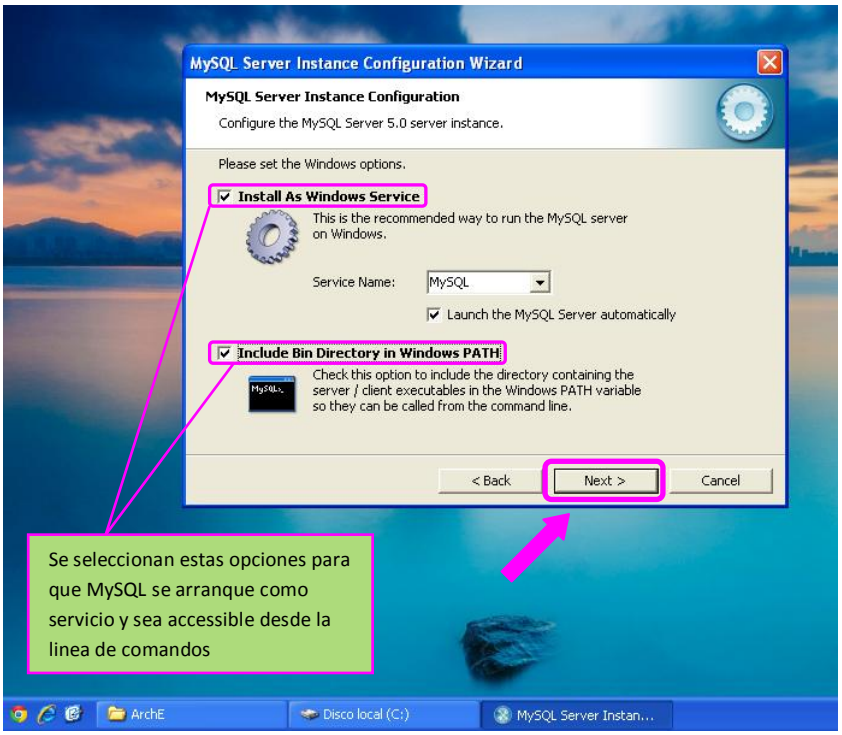


Figura 252. Selección de Opciones para Windows – MySQL.

Por último, se configuran las opciones de seguridad. En este paso, hay que introducir una contraseña que será la misma para configurar ArchE en el apartado 13.8; se pulsa Next >: (Figura 253)

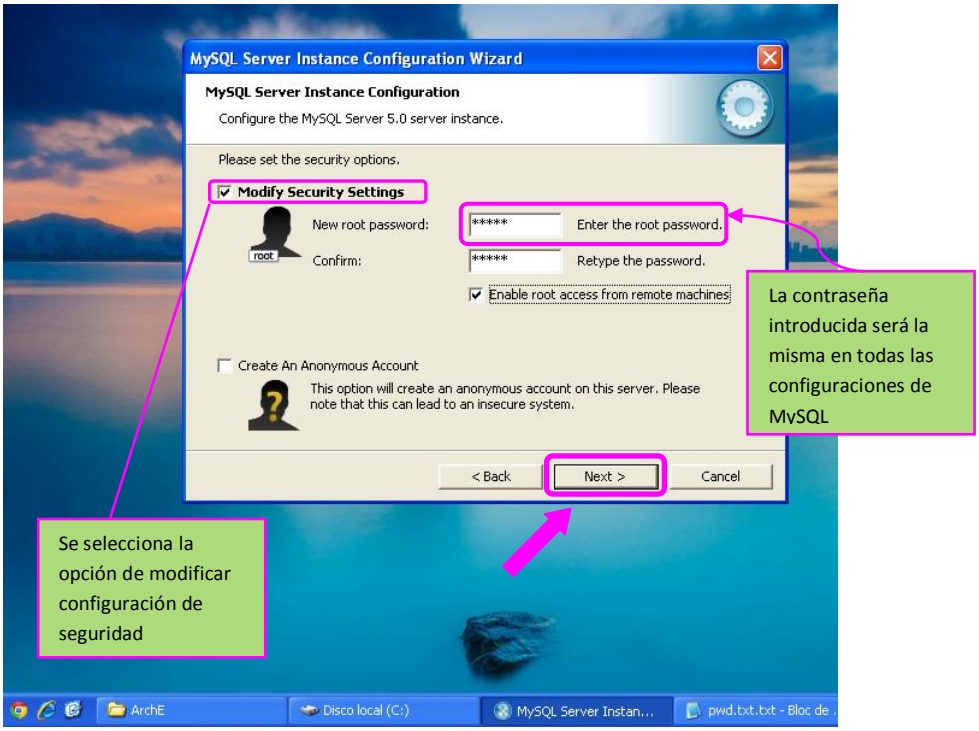


Figura 253. Configuración de Opciones de Seguridad – MySQL.

Una vez finalizado el proceso de configuración, se muestra la ventana para ejecutar los cambios; se pulsa Execute: (Figura 254)

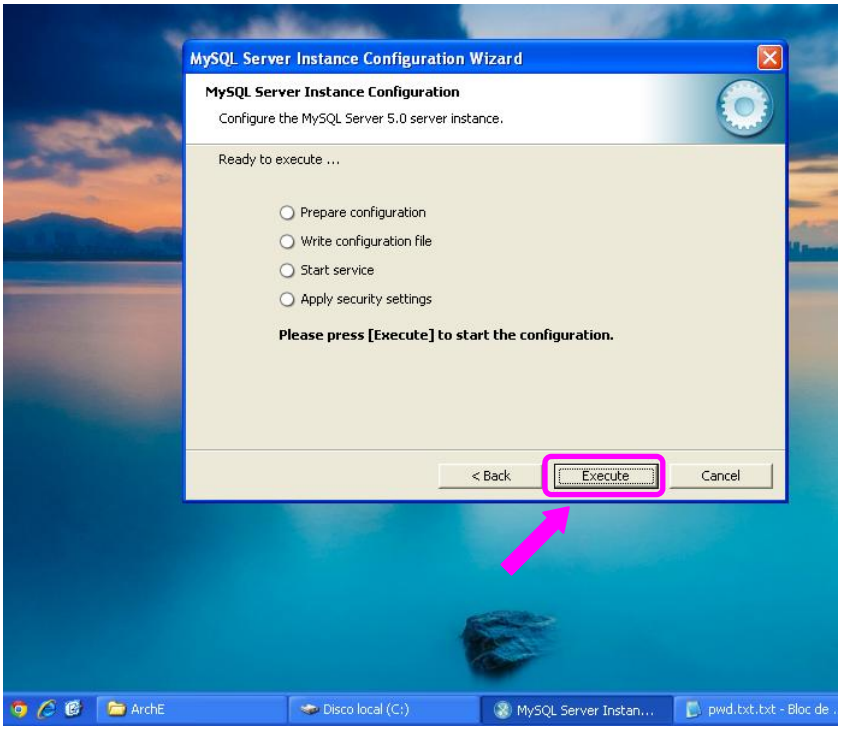


Figura 254. Ejecución de Cambios – MySQL.

Una vez finalizado el proceso, se muestra la ventana de resultados de la instalación; pulsar Finish: (Figura 255)

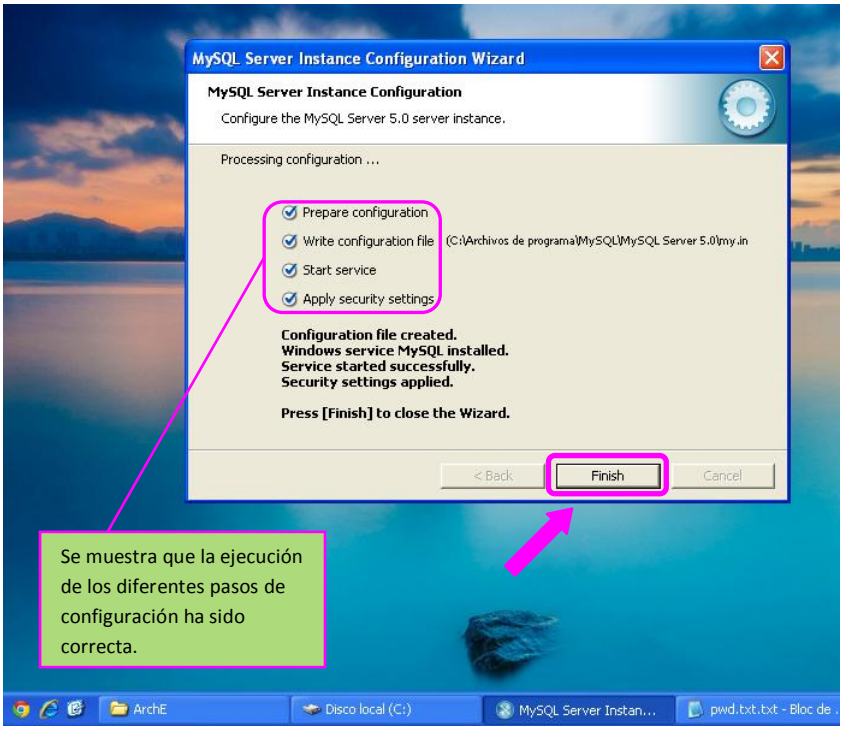


Figura 255. Ventana de Resultados del Proceso de Configuración – MySQL.

Si el proceso de configuración ha sido correcto, MySQL se mostrará como un servicio corriendo en Windows en la ventana del administrador de tareas: (Figura 256)

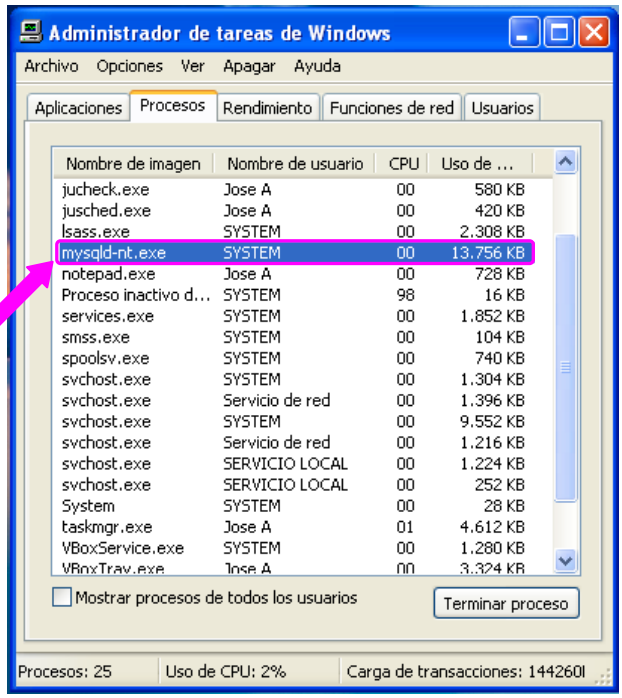


Figura 256. Proceso mysqld-nt.exe en el Administrador de Tareas de Windows.

Para comprobar que se tiene acceso a la aplicación desde la línea de comandos, se ejecuta lo siguiente (la contraseña por defecto seleccionada para la configuración es `arche`): (Figura 257)

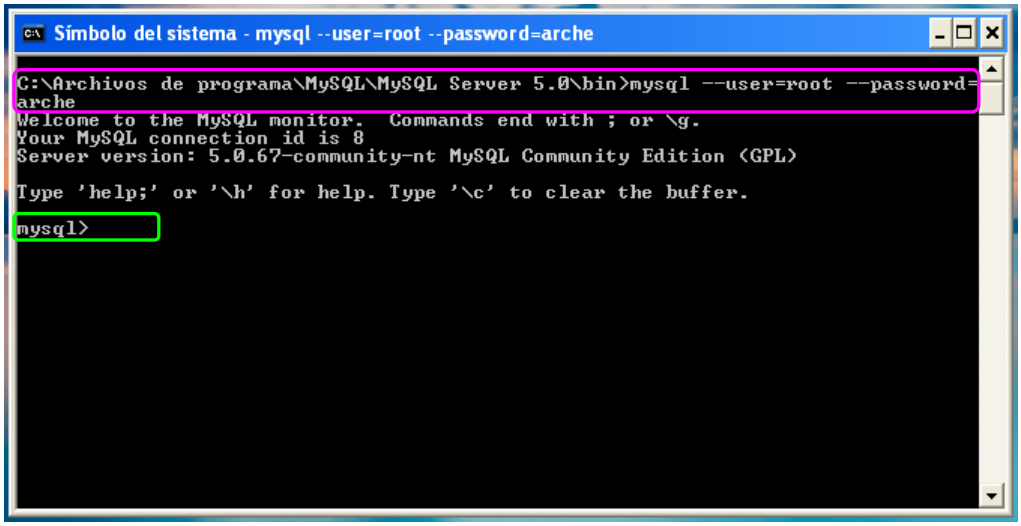


Figura 257. Acceso a MySQL desde la Línea de Comandos

Se comprueba por tanto que se tiene acceso como usuario root a la base de datos de MySQL.

13.5 Instalación de GEF (Graphical Editing Framework)

Para instalar el entorno gráfico GEF, se extraen los ficheros contenidos en el fichero GEF-runtime-3.3.zip en un directorio temporal: (Figura 258 y Figura 259)

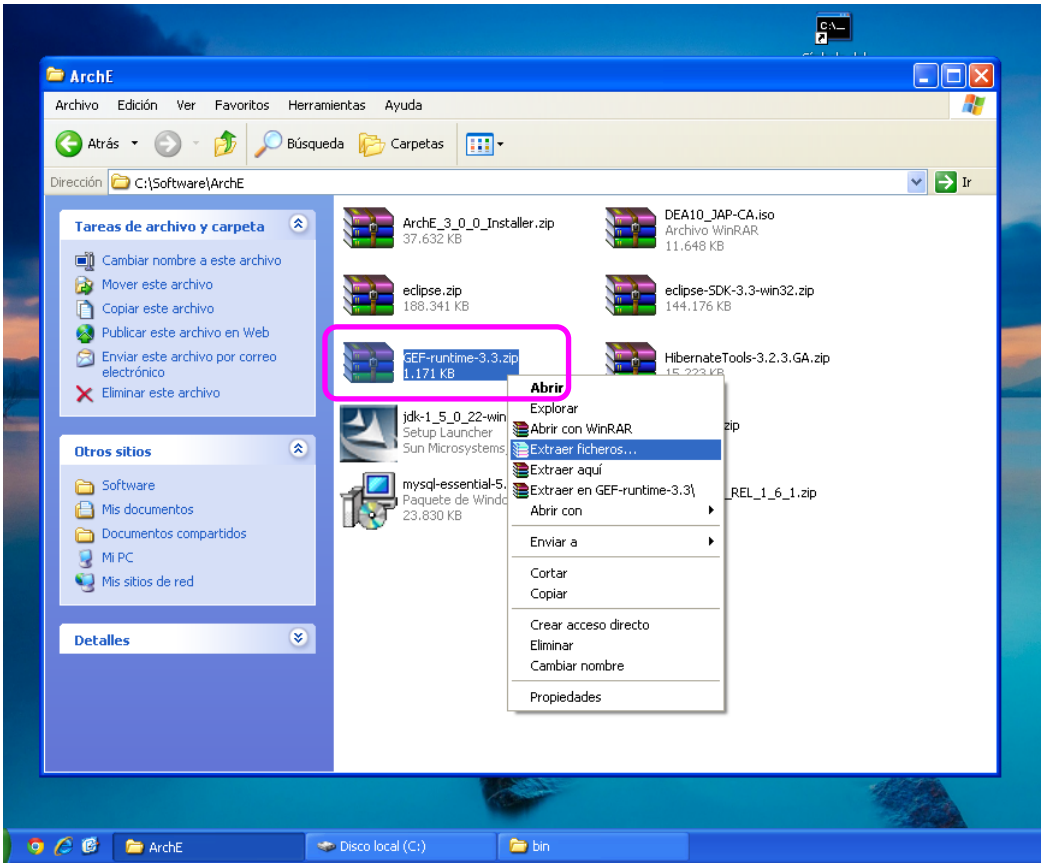


Figura 258. Extracción de los Ficheros del GEF.

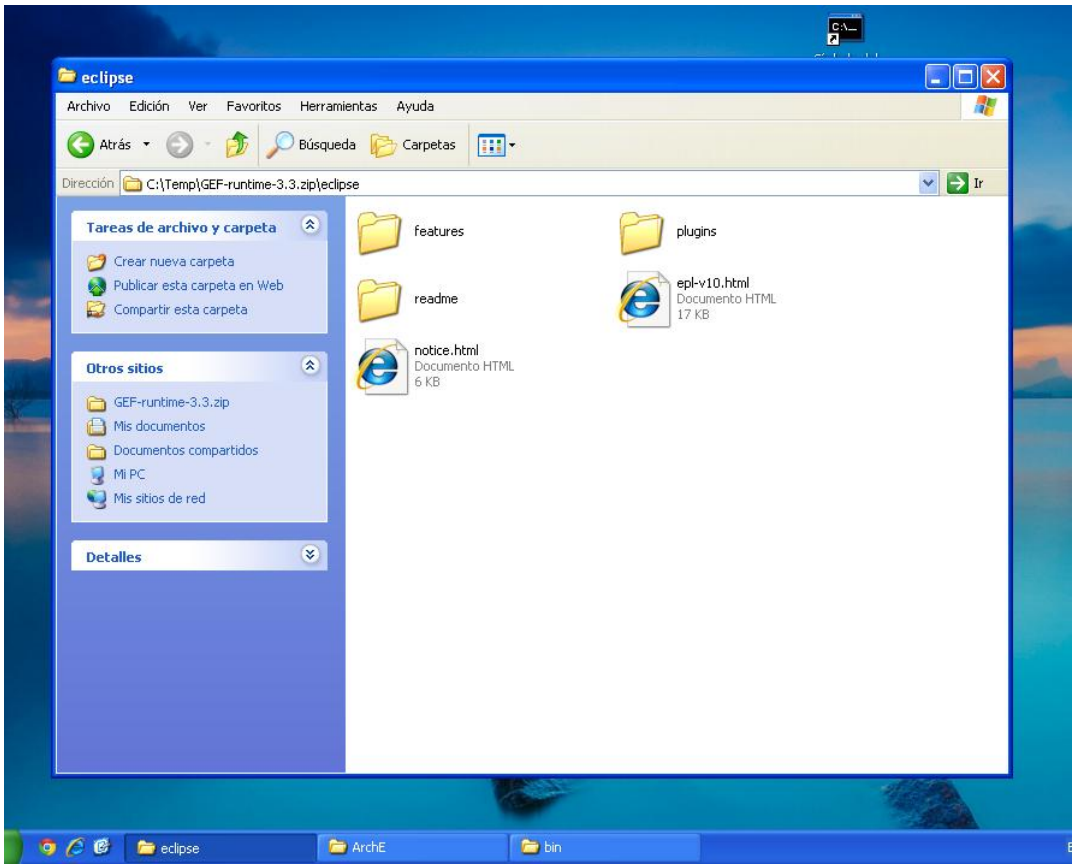


Figura 259. Ficheros de GEF en el directorio temporal.

A continuación, se copian y pegan los pluggins de GEF y la carpeta de features, del directorio temporal a la carpeta Eclipse correspondiente: (Figura 260 y Figura 261)

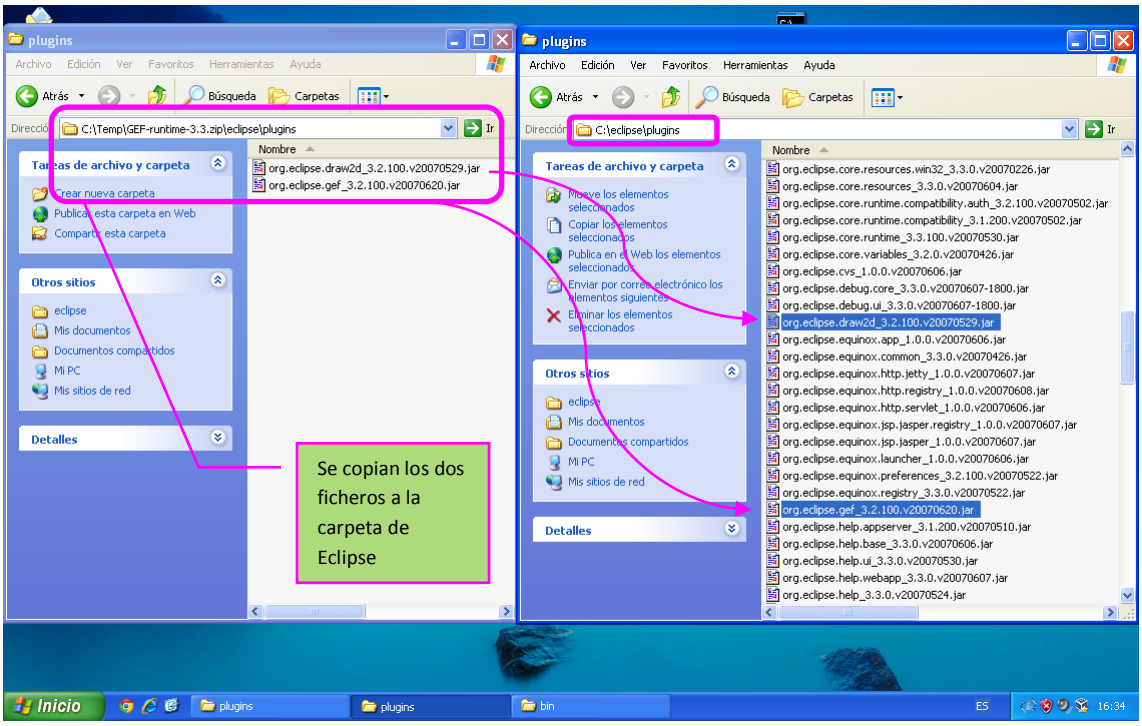


Figura 260. Copia de Pluggins de GEF en la Carpeta de Eclipse

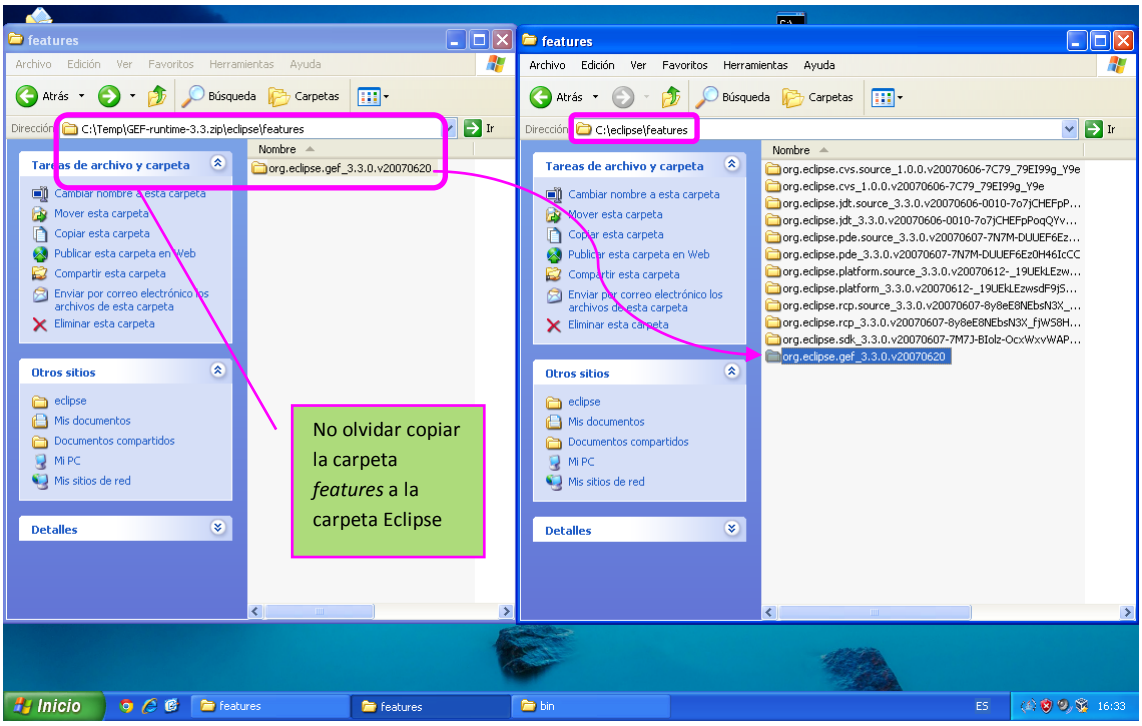


Figura 261. Copia de Carpeta de Features de GEF en la Carpeta de Eclipse

13.6 Instalación de JESS

Para instalar JESS, se extraen los ficheros del archivo Jess71p1.zip en un directorio temporal: (Figura 262)

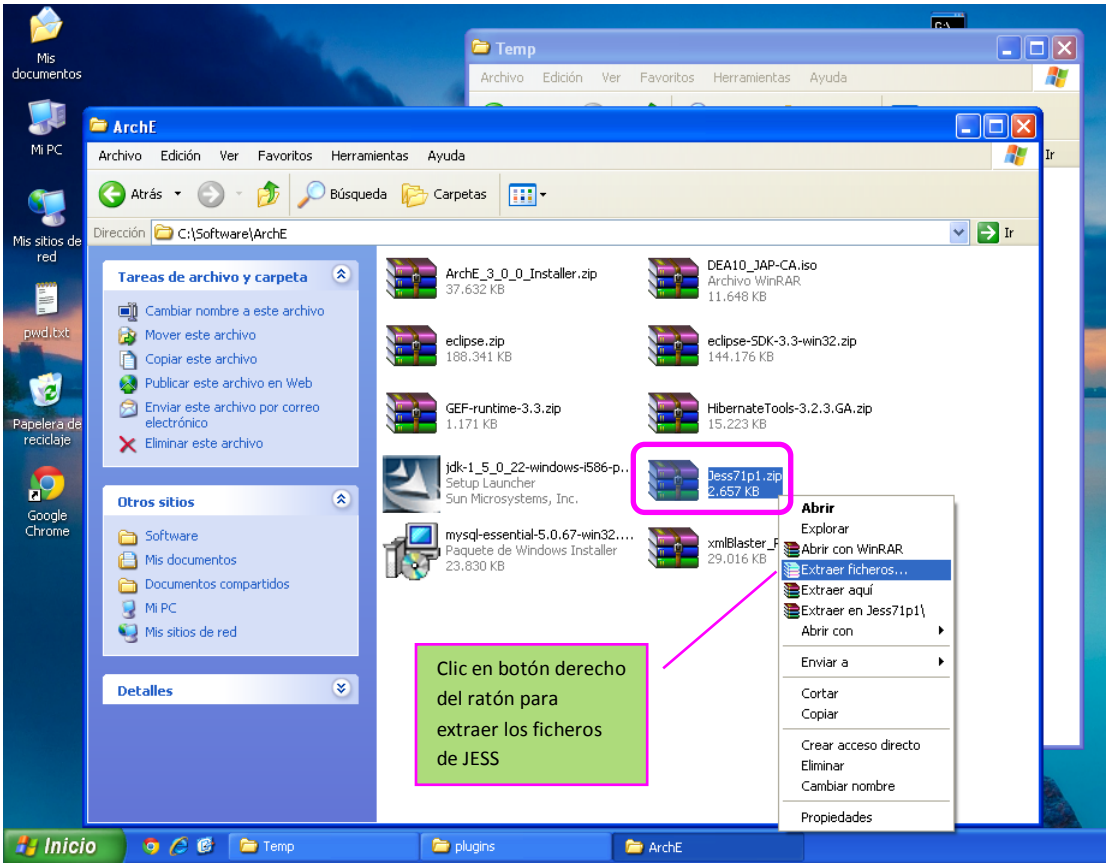


Figura 262. Extracción de Ficheros de JESS.

Una vez extraídos los ficheros, se extraen los nuevos .zip: (Figura 263)

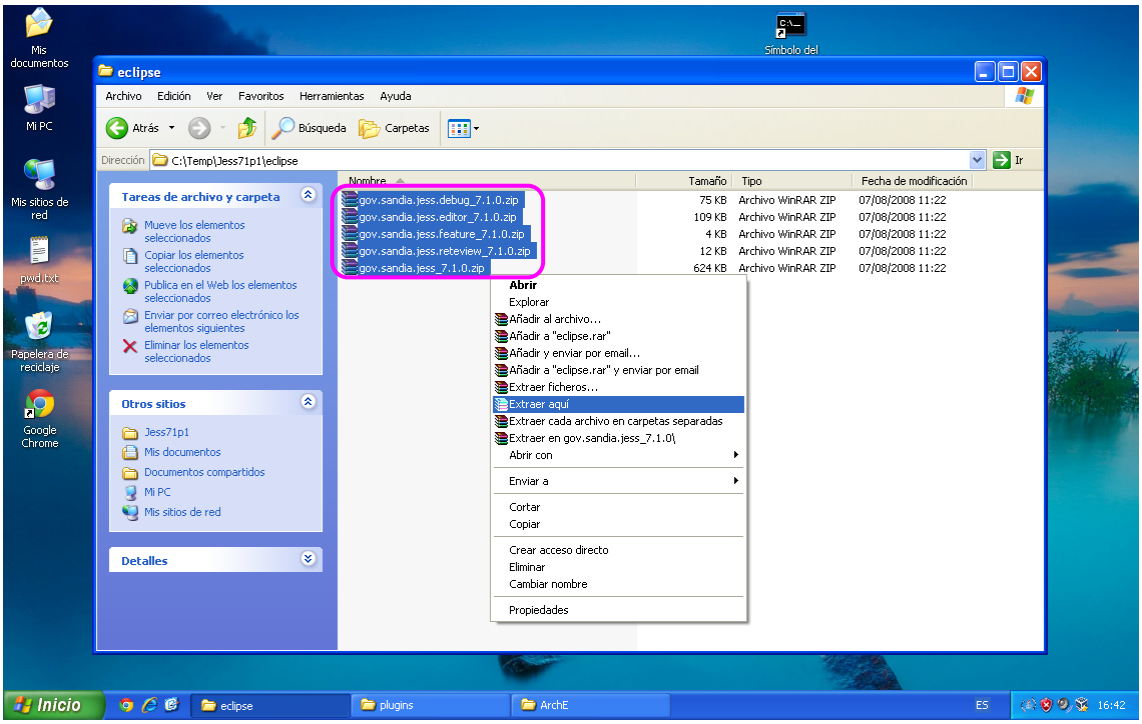


Figura 263. Extracción de los nuevos Ficheros zip de JESS.

Una vez extraídos, se ven dos carpetas nuevas, features y pluggins, igual que en el caso anterior de GEF: (Figura 264)

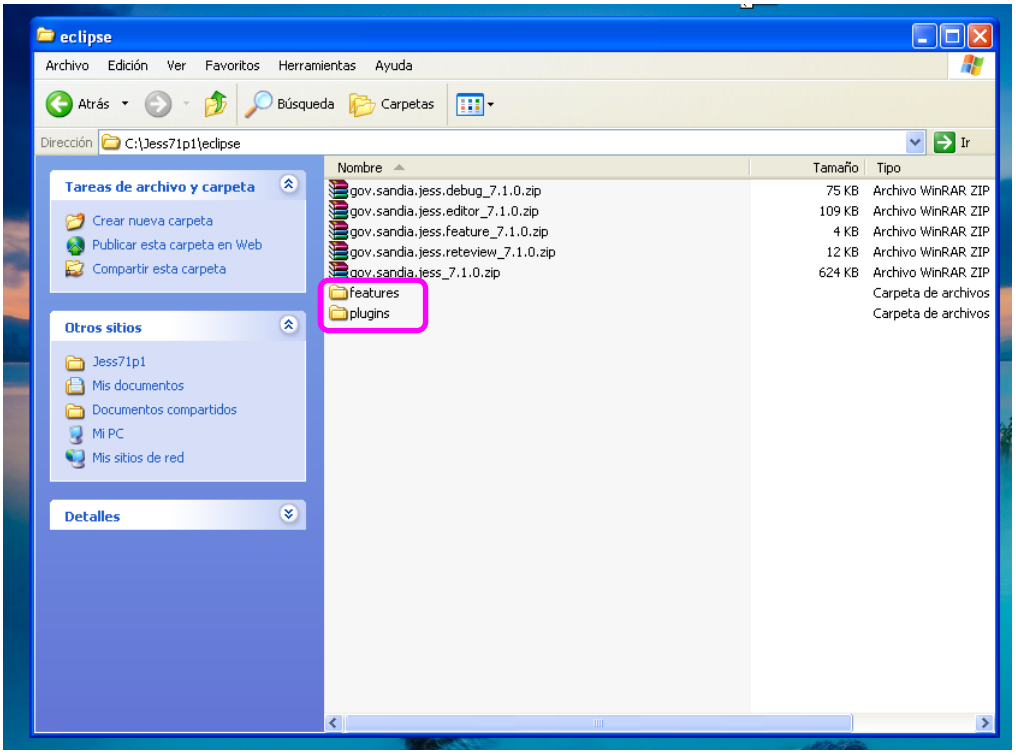


Figura 264. Carpetas de JESS para Eclipse.

A continuación, se copian los pluggins y las carpetas features a sus correspondientes de Eclipse: (Figura 265 y Figura 266)

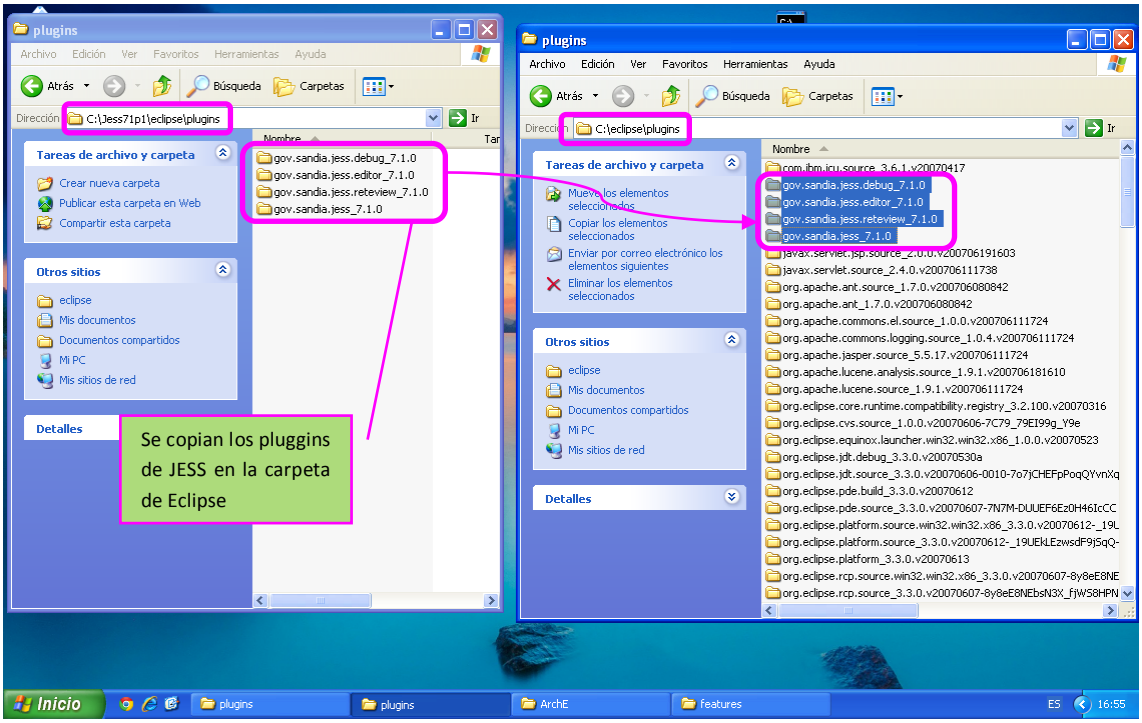


Figura 265. Copia de Pluggins de JESS en la Carpeta Eclipse.

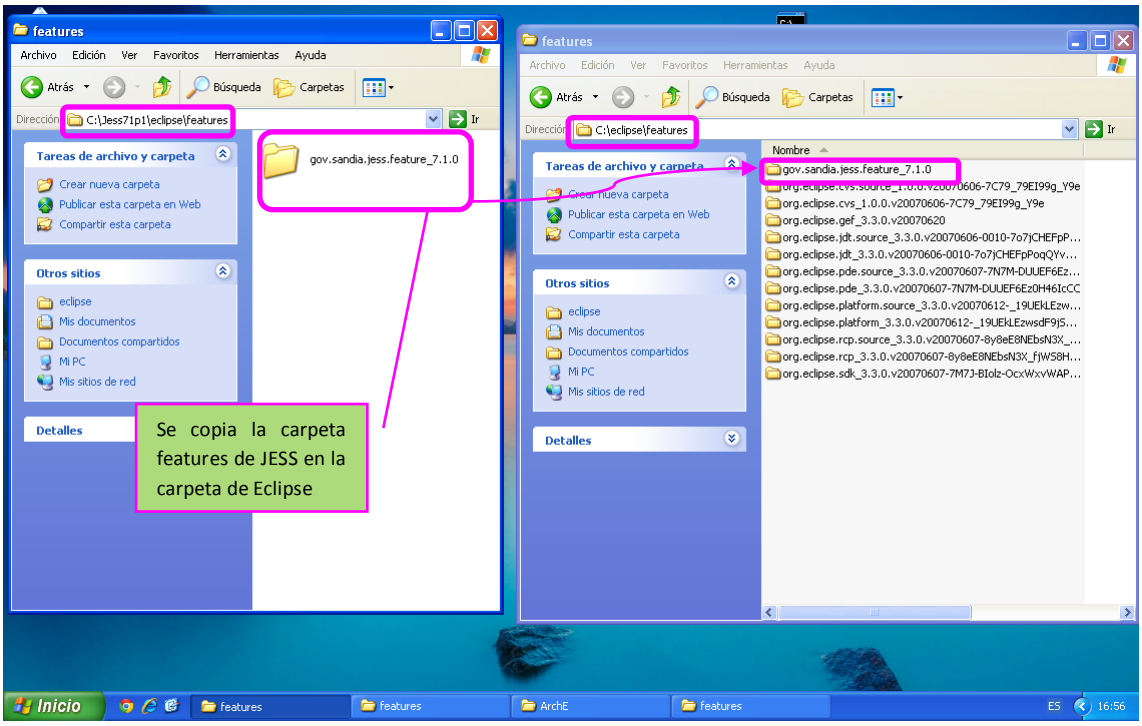


Figura 266. Copia de Features de JESS en la Carpeta Eclipse

Para terminar, se crea la variable del sistema JESS_HOME con la ruta de la aplicación en el disco C:\ (Figura 267)

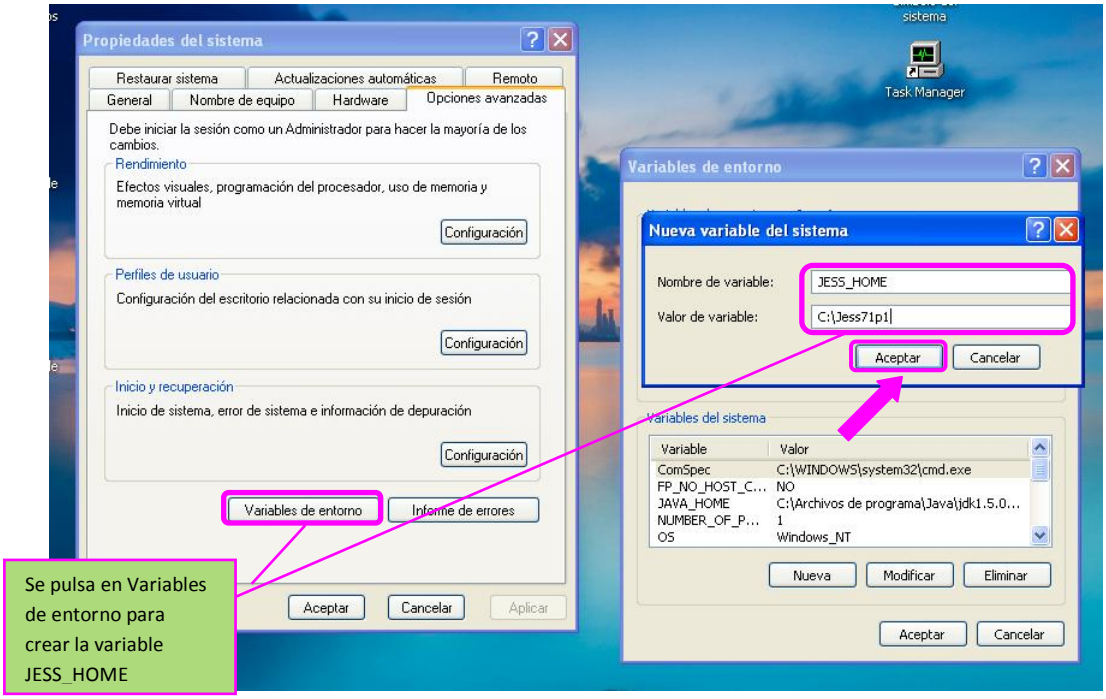


Figura 267. Creación de la Variable JESS_HOME.

Y se añade la variable JESS_HOME en la variable Path: (Figura 268)

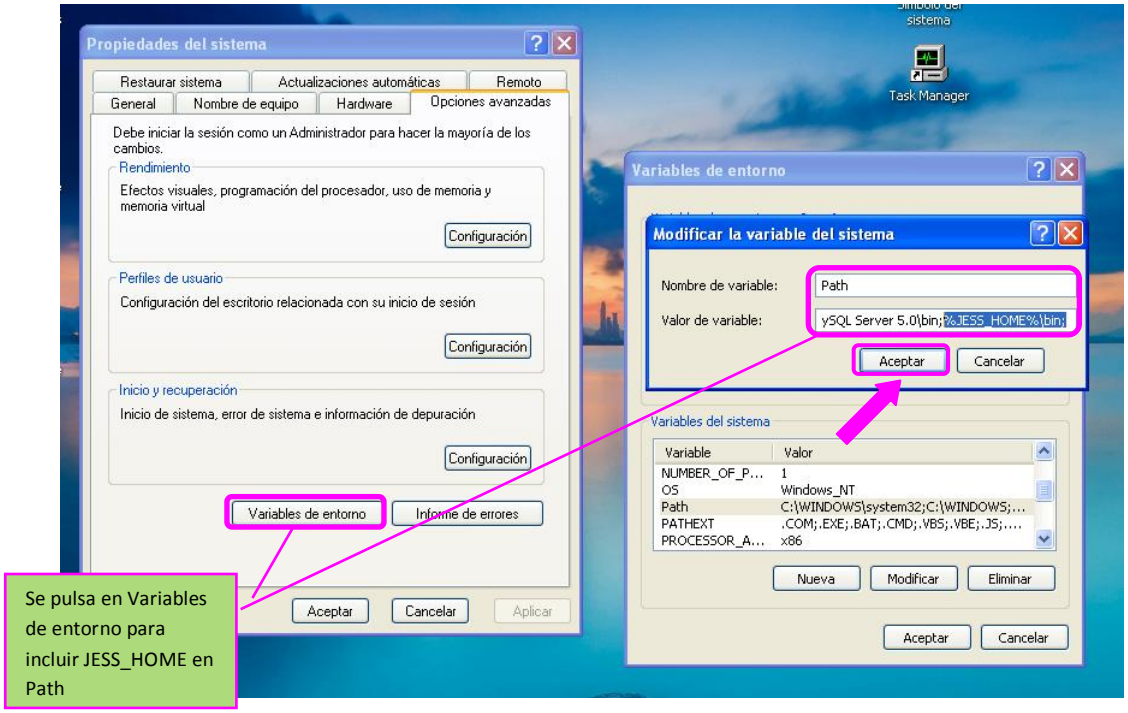


Figura 268. Inclusión de JESS_HOME en Path.

Se comprueba que se tiene acceso a la variable JESS_HOME mediante el comando Path de la línea de comandos: (Figura 269)

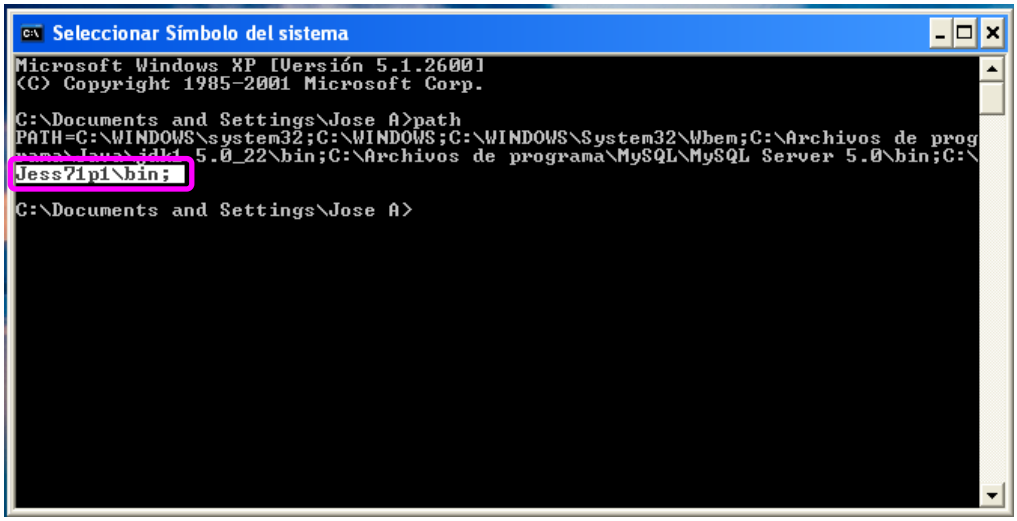


Figura 269. Acceso a Jess71p1 Mediante la Línea de Comandos.

Se inicia Eclipse pulsando en el icono eclipse.exe y se observa que los pluggins se han instalado correctamente: (Figura 270)

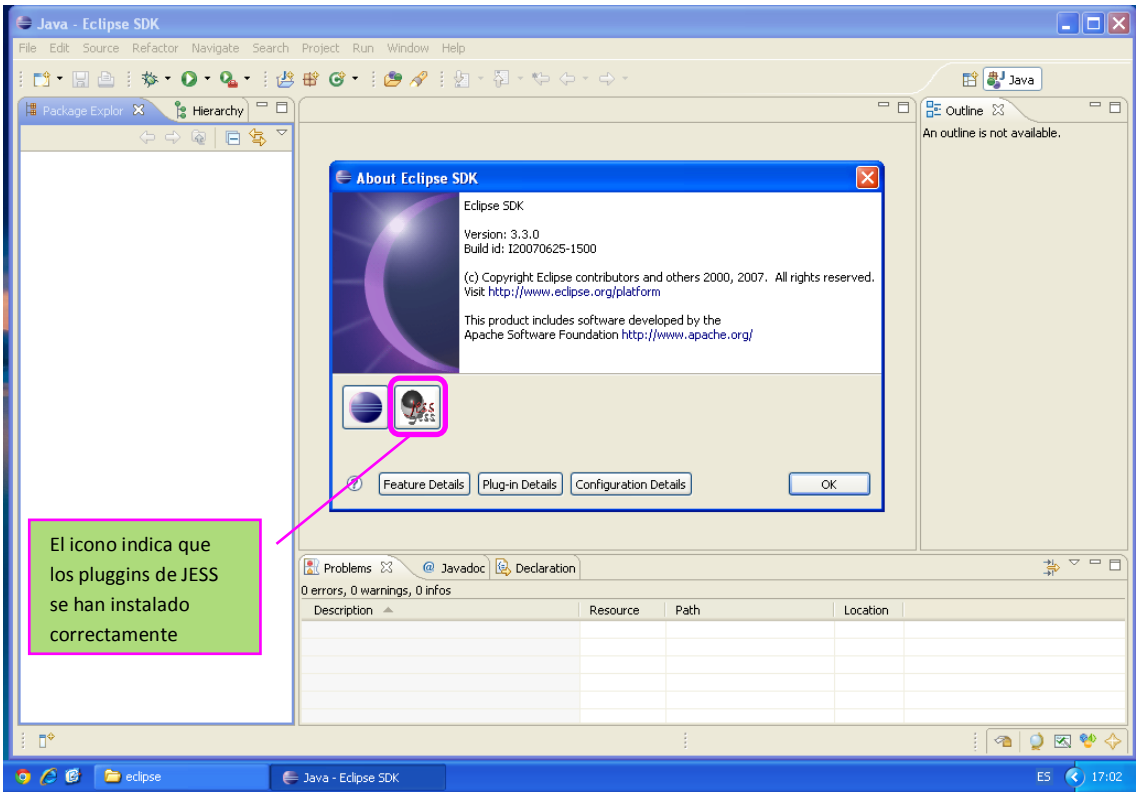


Figura 270. Pluggins de JESS instalados correctamente en Eclipse.

13.7 Instalación de xmlBlaster

Para instalar xmlBlaster, primero se descomprime el fichero xmlBlaster_REL_1_6_1.zip: (Figura 271 y Figura 272)

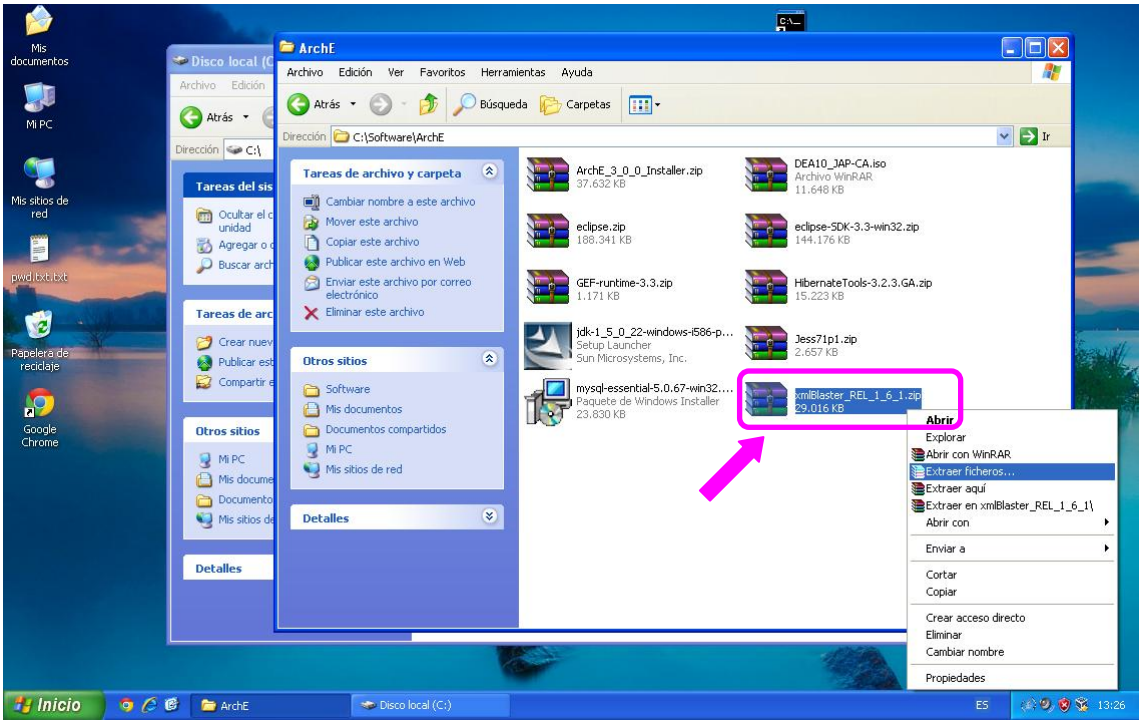


Figura 271. Descomprimir el fichero xmlBlaster_REL_1_6_1.zip.

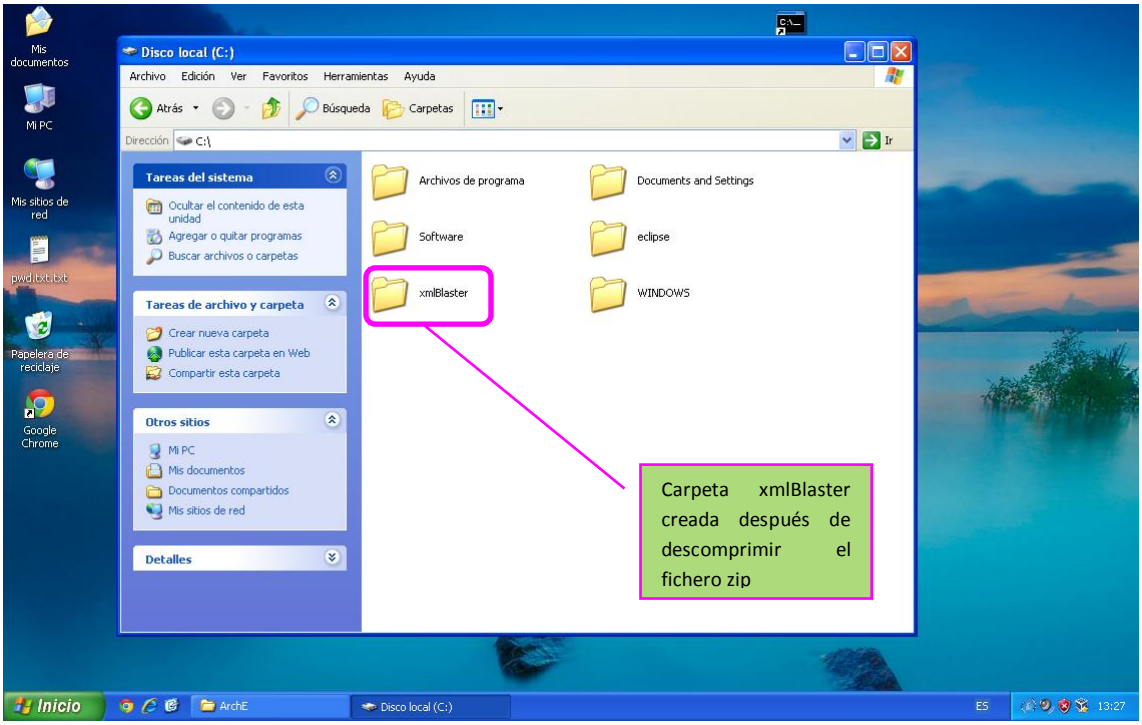


Figura 272. Carpeta xmlBlaster.

A continuación, hay que crear el archivo por lotes run_xmlblaster.bat. Se copia el siguiente texto en un fichero .txt y después se cambia la extensión a .bat: (Figura 273 y Figura 274)

```
@echo off
if "%OS%" == "Windows_NT" setlocal
rem Guess XMLBLASTER_HOME if not defined
if not "%XMLBLASTER_HOME%" == "" goto gotHome
set XMLBLASTER_HOME=.
if exist "%XMLBLASTER_HOME%\lib\xmlBlaster.jar" goto okHome
set XMLBLASTER_HOME=..
if exist "%XMLBLASTER_HOME%\lib\xmlBlaster.jar" goto okHome
set XMLBLASTER_HOME="%~dp0.."
if exist "%XMLBLASTER_HOME%\lib\xmlBlaster.jar" goto okHome

:gotHome
        if exist "%XMLBLASTER_HOME%\lib\xmlBlaster.jar" goto okHome
        echo "%XMLBLASTER_HOME%"
        echo The XMLBLASTER_HOME environment variable is not defined correctly,
        echo and this script can't find the file "xmlBlaster.jar" without it.
        echo Please set this environment variable to point to your xmlBlaster installation
        echo directory, then try again
        goto end

:okHome
        rem Make sure JAVA_HOME is set correctly
        if not "%JAVA_HOME%" == "" goto gotJavaHome
        echo Warning: the JAVA_HOME environment variable is not defined
        echo If Jess fails to start, set this environment variable to
        echo point to your JDK installation directory, then try again.
        set RUN_JAVA=java.exe
        goto start

:gotJavaHome
        if not exist "%JAVA_HOME%\bin\java.exe" goto noJavaHome
        goto okJavaHome

:noJavaHome
        echo The JAVA_HOME environment variable is defined incorrectly.
        echo If set, it must represent the path to a J2SDK installation.
        echo If it is unset, then java.exe must be on your path.
        goto end

:okJavaHome
        set RUN_JAVA="%JAVA_HOME%\bin\java.exe"
        goto start

:start
        %RUN_JAVA% -jar "%XMLBLASTER_HOME%\lib\xmlBlaster.jar"

:end
```

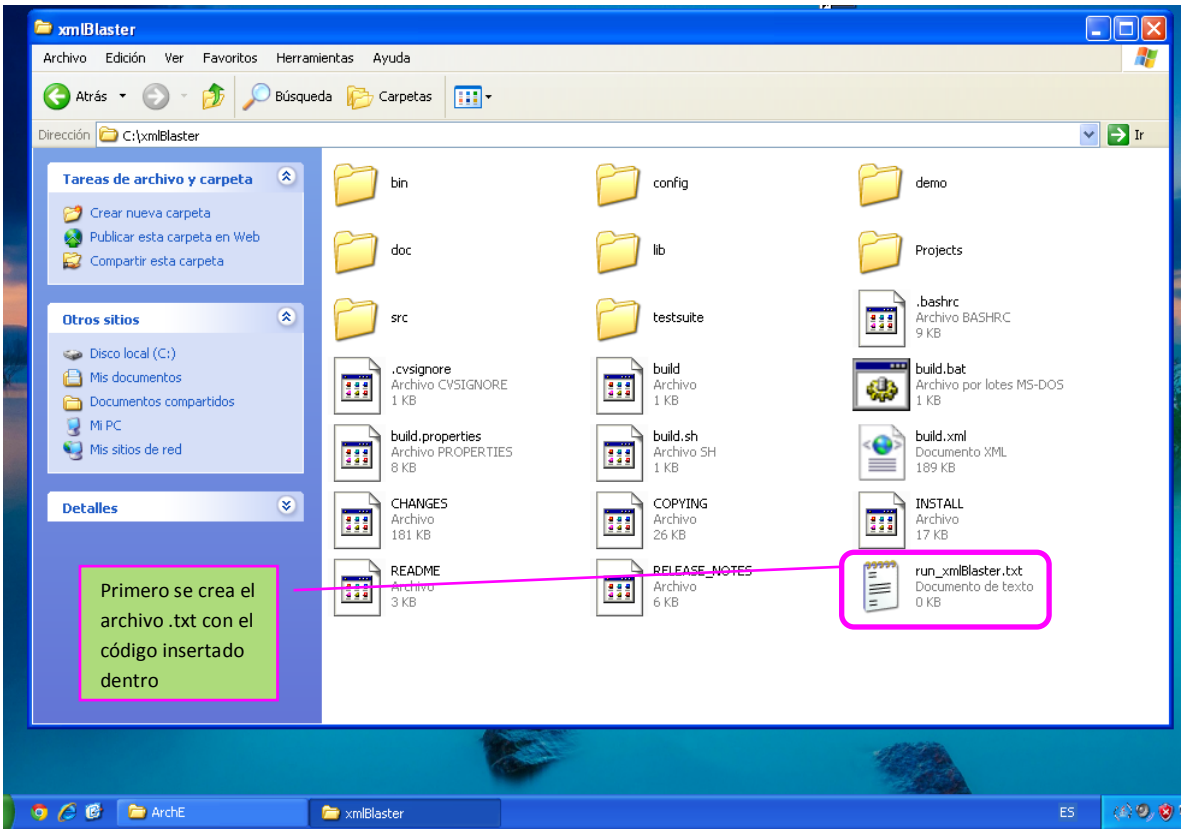


Figura 273. Fichero .txt con el código para iniciar xmlBlaster.

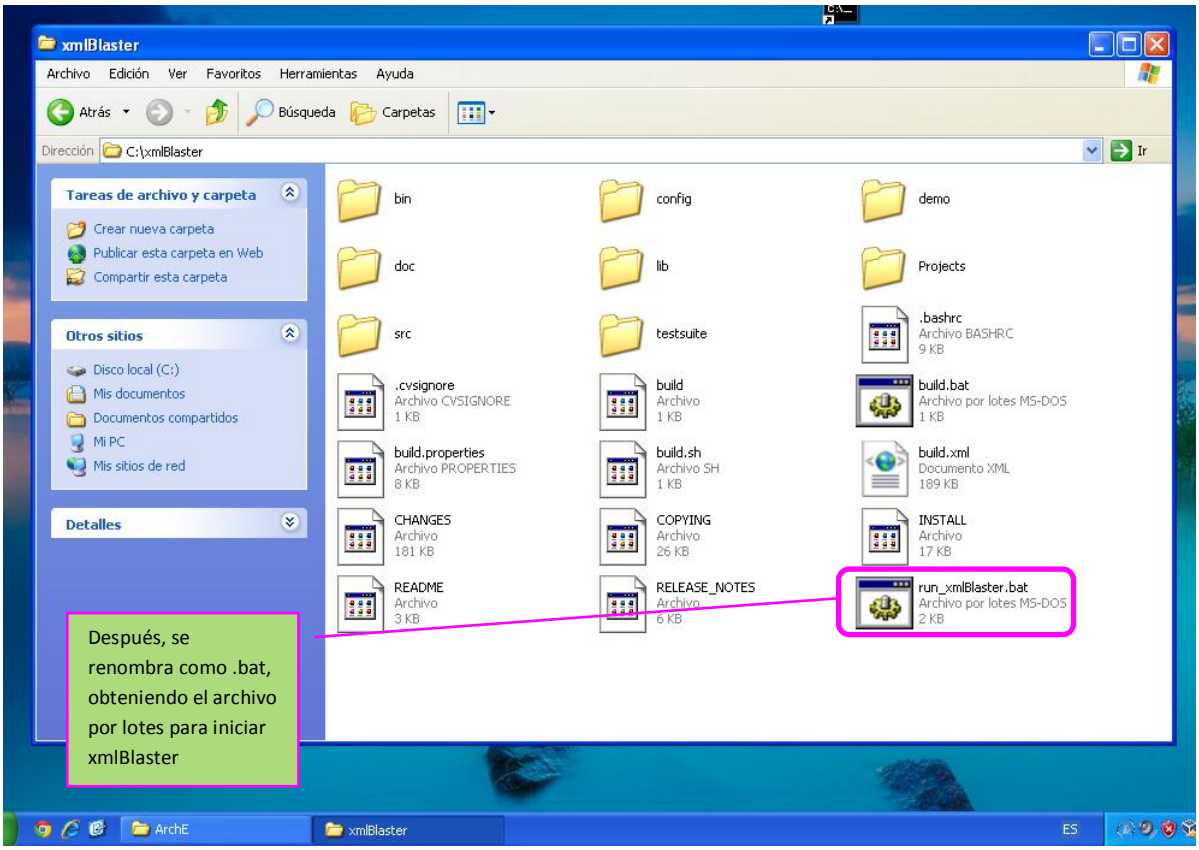


Figura 274. Fichero .bat con el código para iniciar xmlBlaster.

Se crea la variable del sistema XMLBLASTER_HOME indicando la ruta a la carpeta xmlBlaster desde C\: (Figura 275)

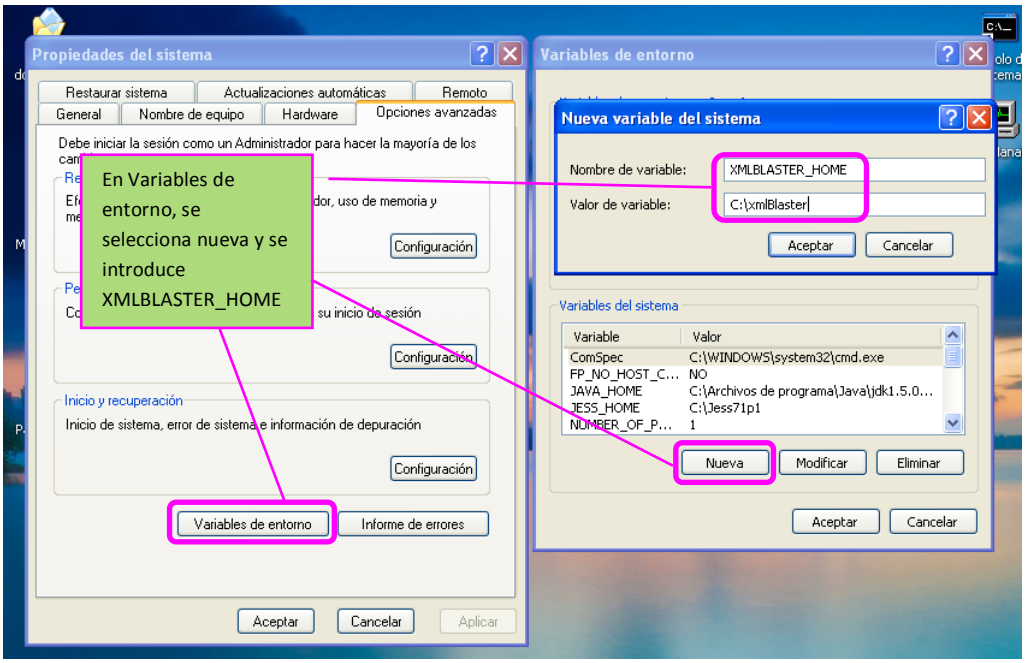


Figura 275. Creación de la Variable de Estado XMLBLASTER_HOME.

Y a continuación se modifica la variable del sistema path: (Figura 276)

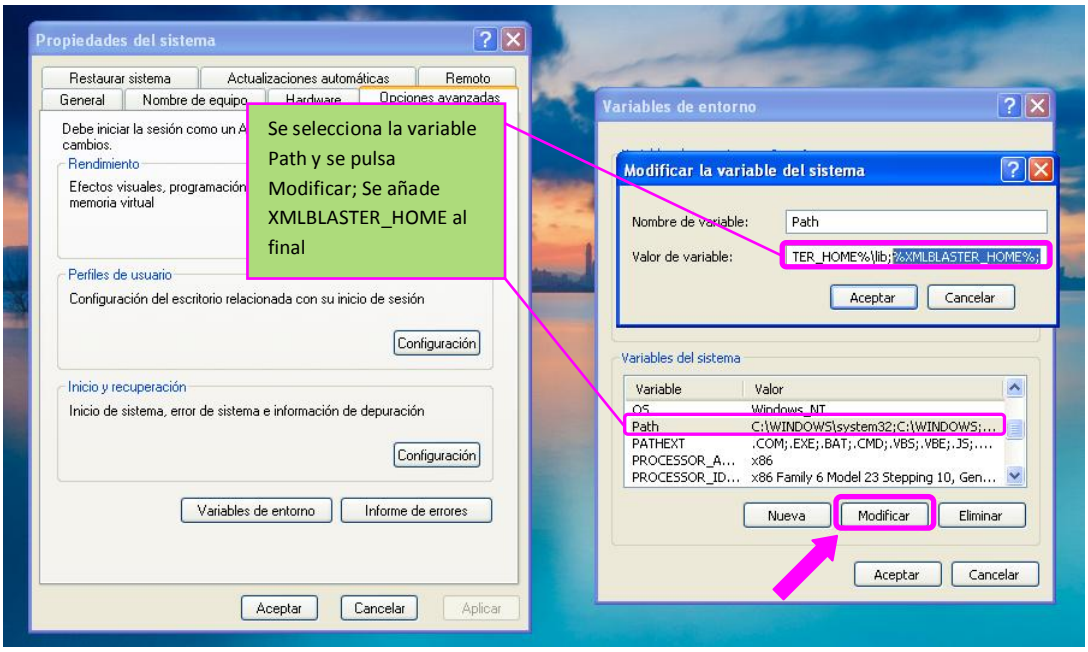


Figura 276. Modificación de Path añadiéndole XMLBLASTER_HOME.

A continuación se comprueba que se ha incluido correctamente, ejecutando el comando path en la línea de comandos: (Figura 277)

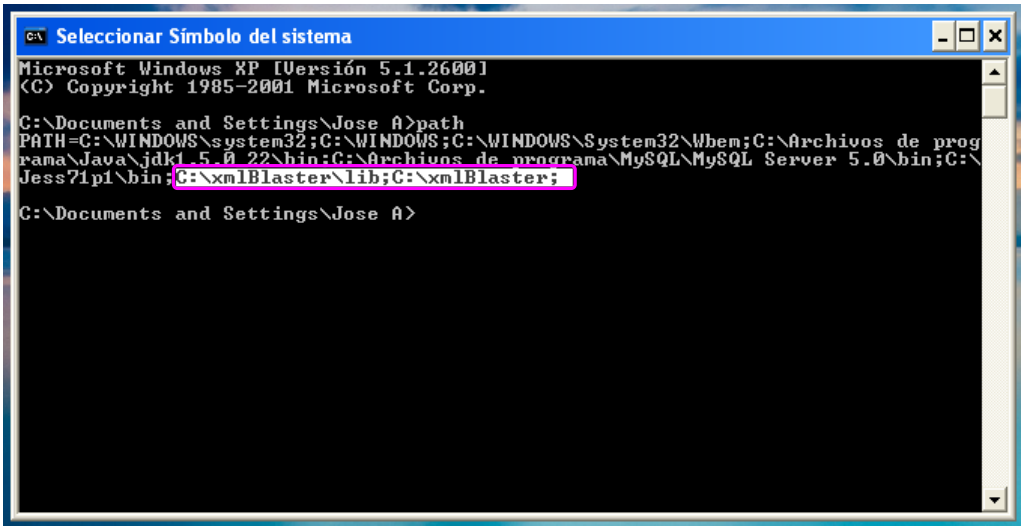


Figura 277. Comando Path en Línea de Comandos Muestra la ruta completa a xmlBlaster.

Comprobamos que funciona el programa, ejecutando el archivo por lotes run_xmlBlaster.bat: (Figura 278)

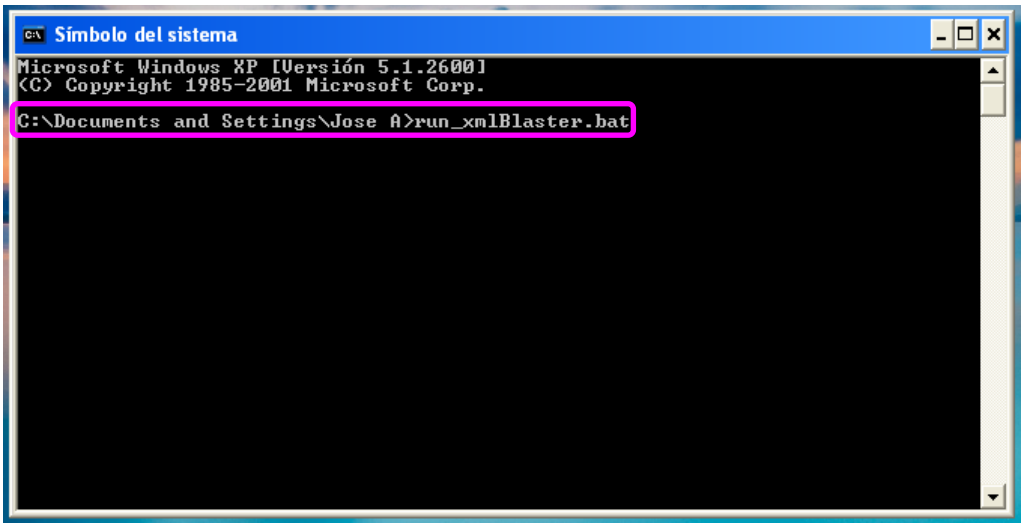


Figura 278. Ejecución de run_xmlBlaster.bat en línea de Comandos.

Como se puede ver, xmlblaster se ejecuta correctamente: (Figura 279)

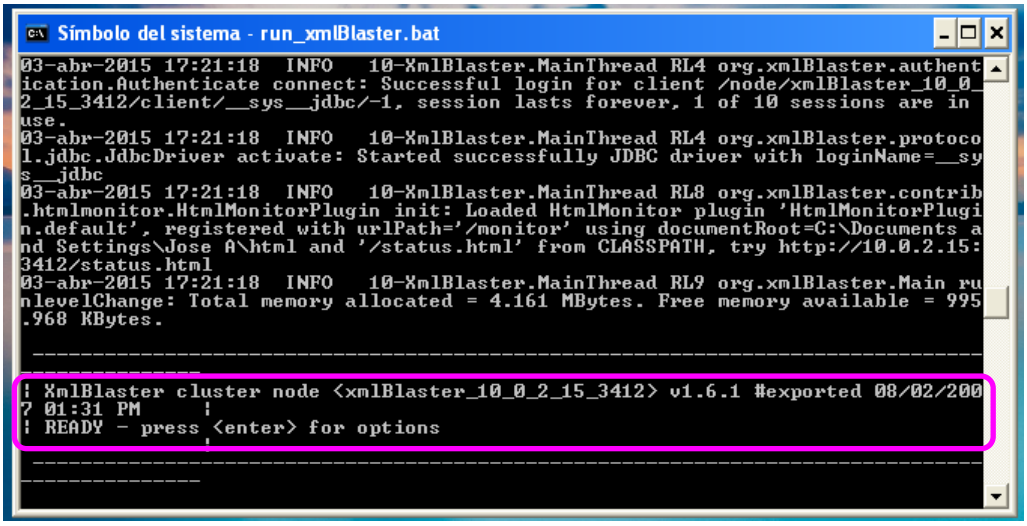


Figura 279. Ejecución de xmlblaster.

Por último, se presiona “r” cuando se ha terminado de ejecutar xmlblaster: (Figura 280)

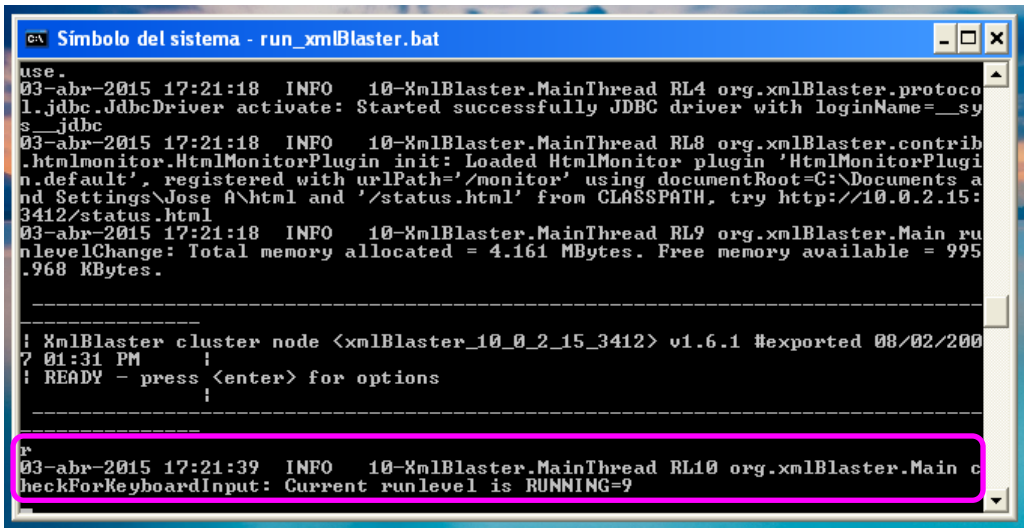


Figura 280. Pulsar "r" al finalizar el arranque de xmlBlaster.

Es importante destacar en este punto que lo primero que hay que hacer antes de ejecutar ArchE es arrancar xmlBlaster, pues es el responsable de la comunicación con los marcos de razonamiento; sin él, ArchE no funcionará correctamente.

Para comprobar adicionalmente que xmlBlaster funciona correctamente, se abre el navegador y se escribe la dirección:

http://10.0.2.15:3412/status.html

Se debe de abrir la siguiente ventana de estado: (Figura 281)

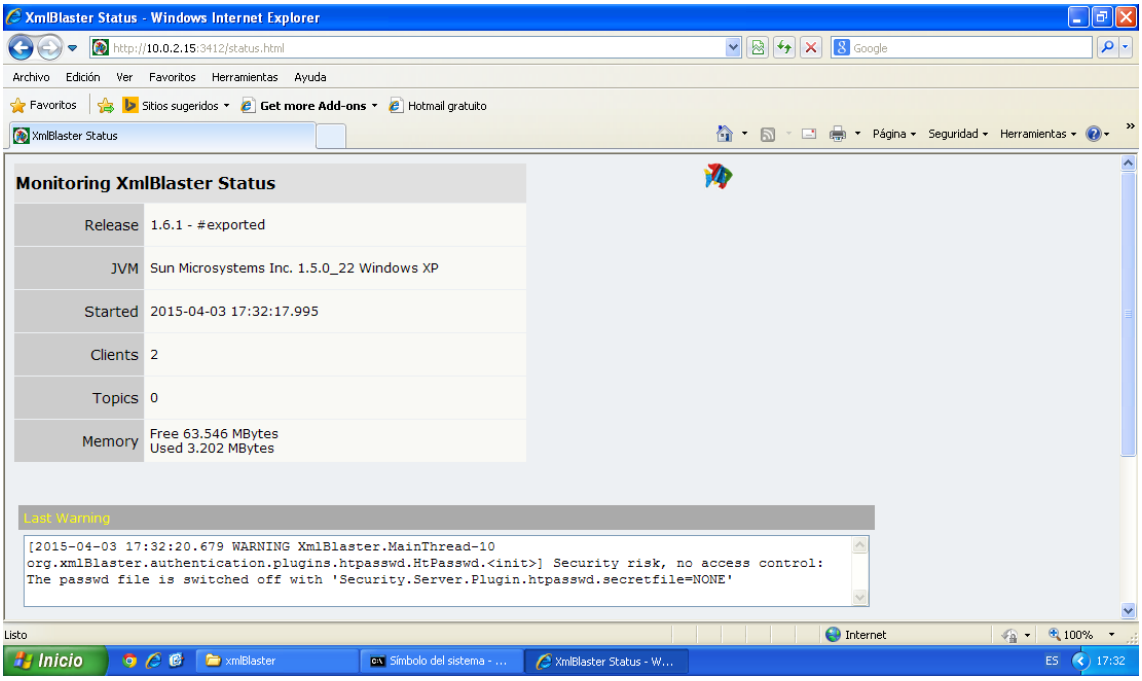


Figura 281. Ventana de Estado de xmlBlaster en el Navegador.

13.8 Instalación de ArchE

Para empezar la instalación de ArchE, hay que ejecutar el instalador: (Figura 282 y Figura 283)

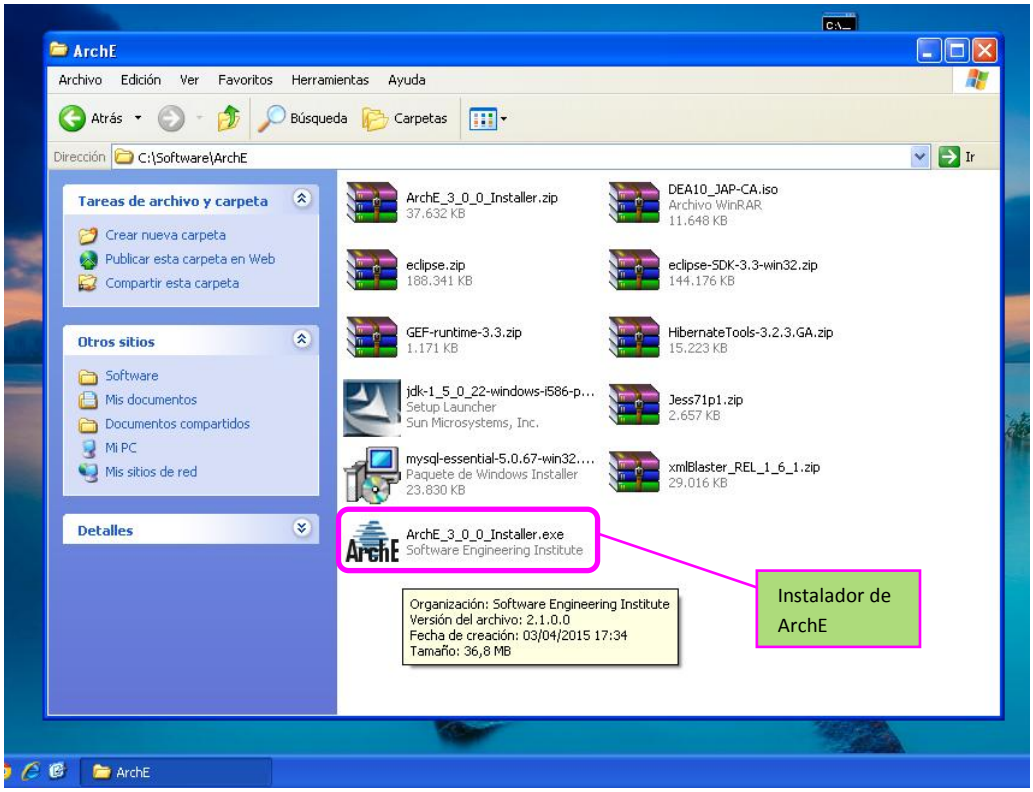


Figura 282. Ejecutar el Instalador de ArchE.

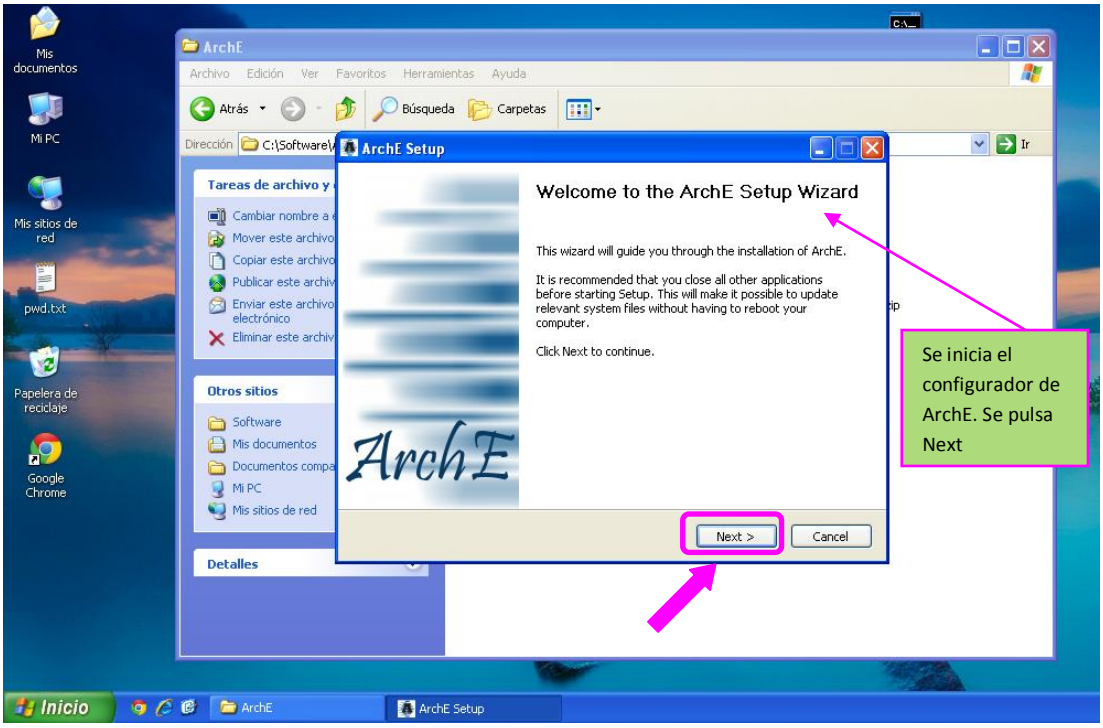


Figura 283. Configurador de ArchE.

Se selecciona el tipo de instalación completa y se pulsa Next >: (Figura 284)

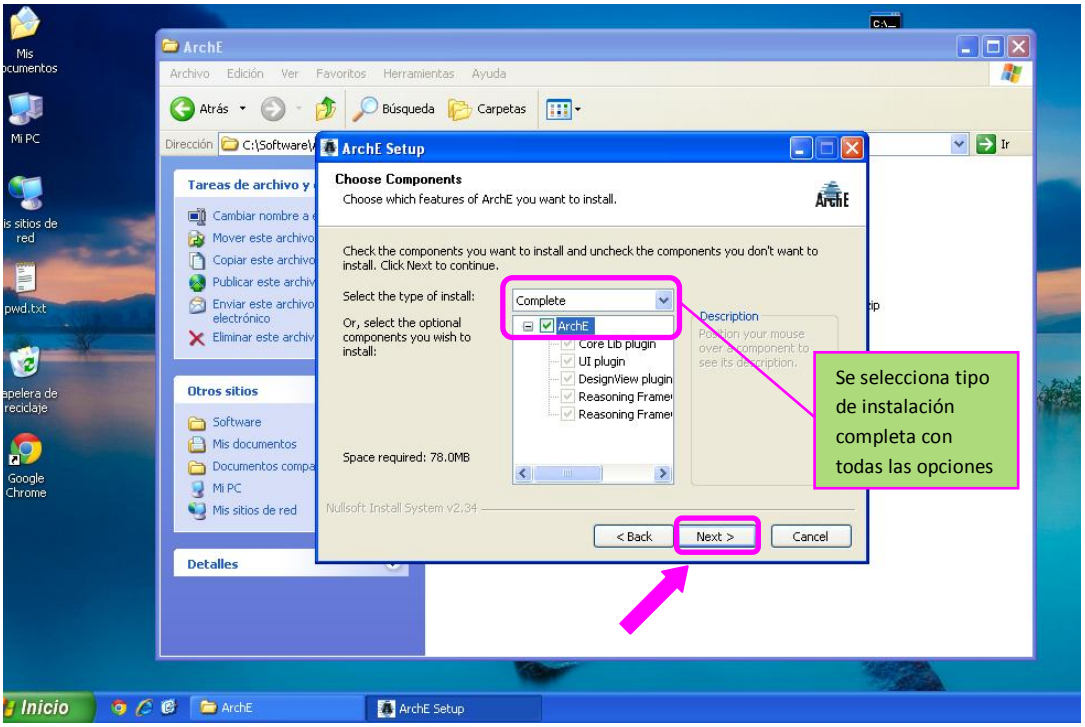


Figura 284. Selección de Instalación Completa de ArchE.

Se selecciona la carpeta de destino y se pulsa Next >: (Figura 285)

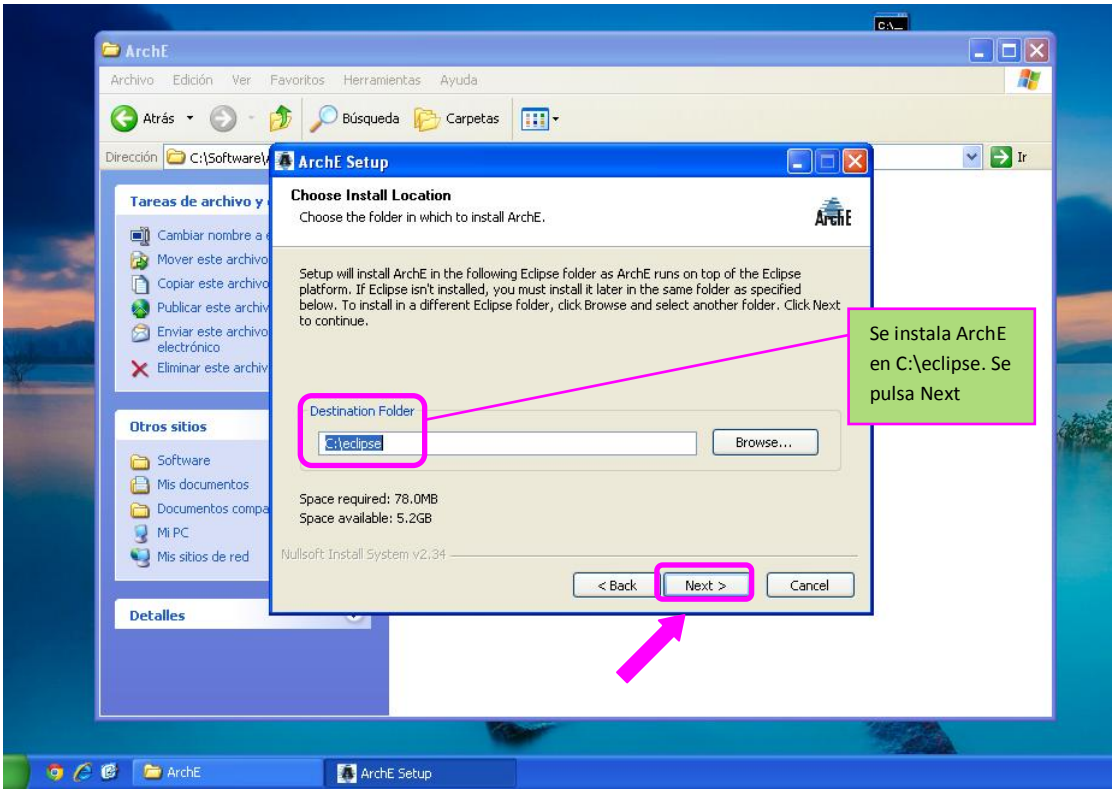


Figura 285. Selección de Carpeta de Destino de ArchE.

Se selecciona la carpeta del menú inicio y se pulsa Next >: (Figura 286)

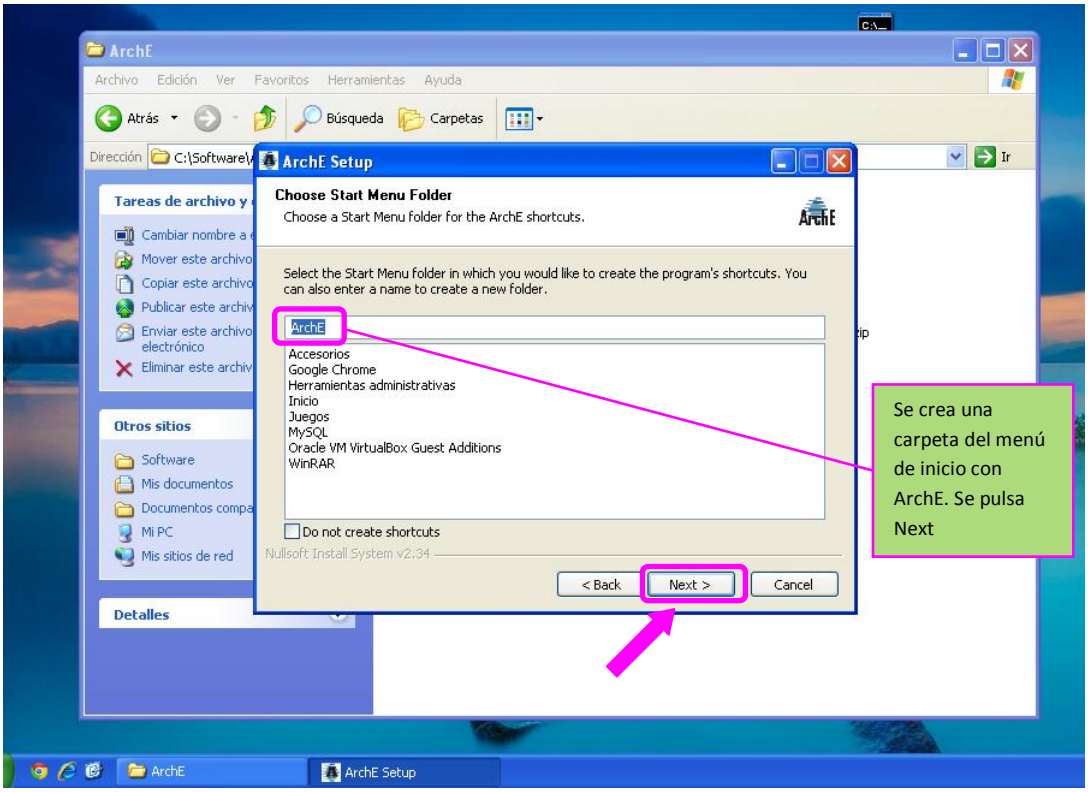


Figura 286. Carpeta del Menú Inicio de ArchE.

El configurador muestra la lista de componentes externos que se necesitan y los que faltan por instalar: (aunque en este caso, el configurador no reconoce a xmlBlaster, aunque esté ya instalado) (Figura 287)

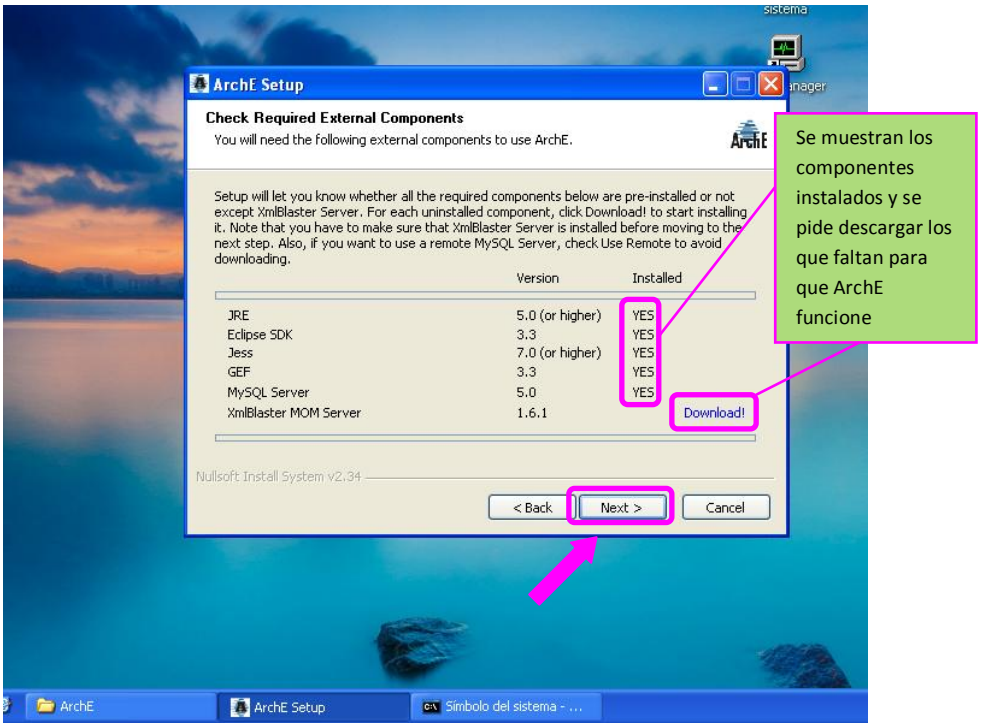


Figura 287. Componentes Externos Requeridos por ArchE.

Se muestra a continuación la pantalla de configuración de MySQL; la contraseña es la misma que se introdujo en la configuración de MySQL en el apartado 13.4, para hacer más sencilla la instalación: (Figura 288)

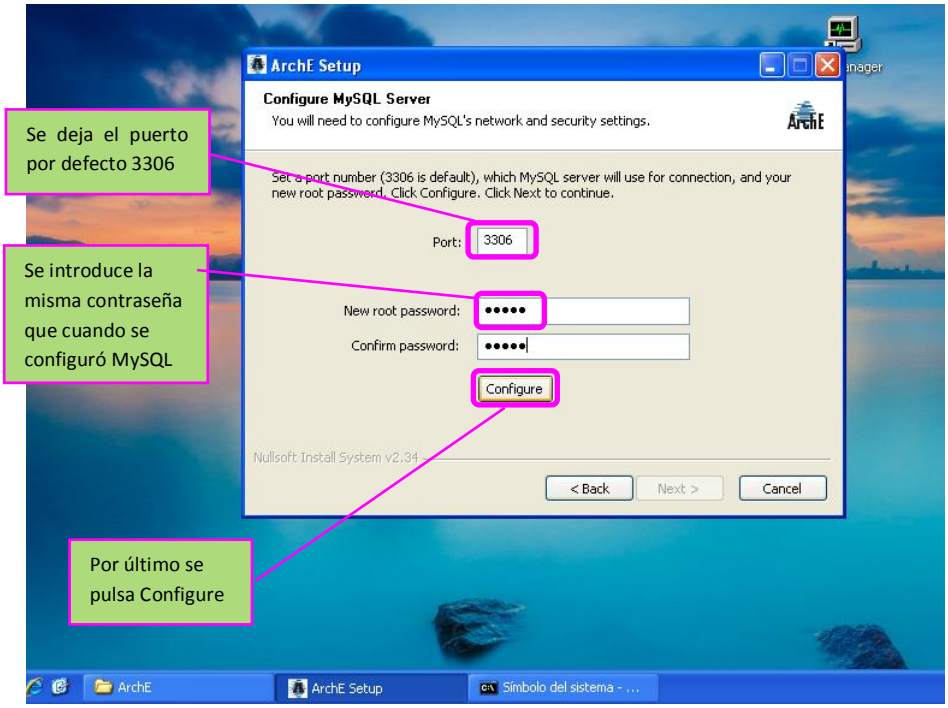


Figura 288. Configuración de MySQL con ArchE.

MySQL está siendo configurado por el asistente de ArchE: (Figura 289)

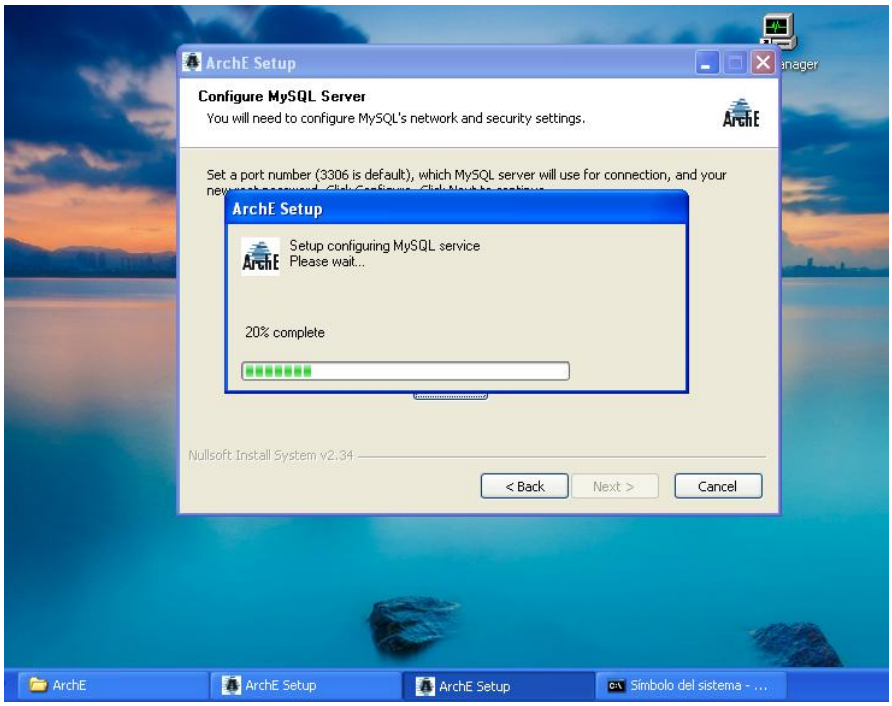


Figura 289. El Configurador de ArchE está configurando MySQL.

Se advierte de que el directorio de datos de MySQL existe y por tanto su contraseña también; se introducirá la misma contraseña y no se cambiará, para usar siempre la misma: (Figura 290 y Figura 291)

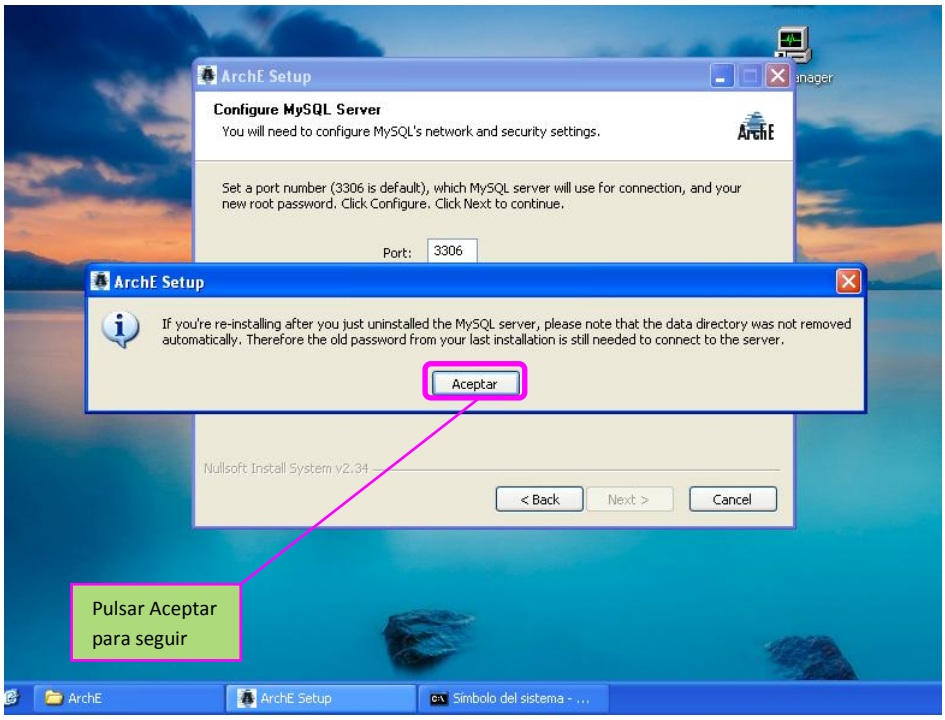


Figura 290. Siguiente Paso de Configuración de MySQL en ArchE.

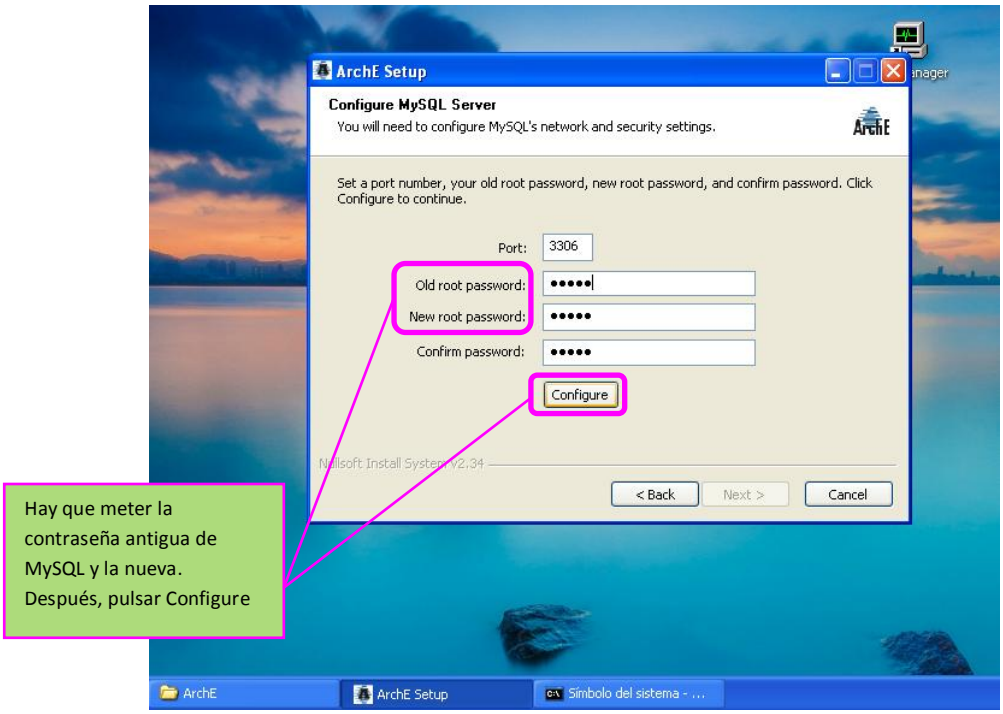


Figura 291. Introducir Antigua y Nueva Password de MySQL.

Una vez configurada la contraseña (que será siempre la misma usada en la anterior configuración de MySQL), se pulsa Next > para continuar: (Figura 292)

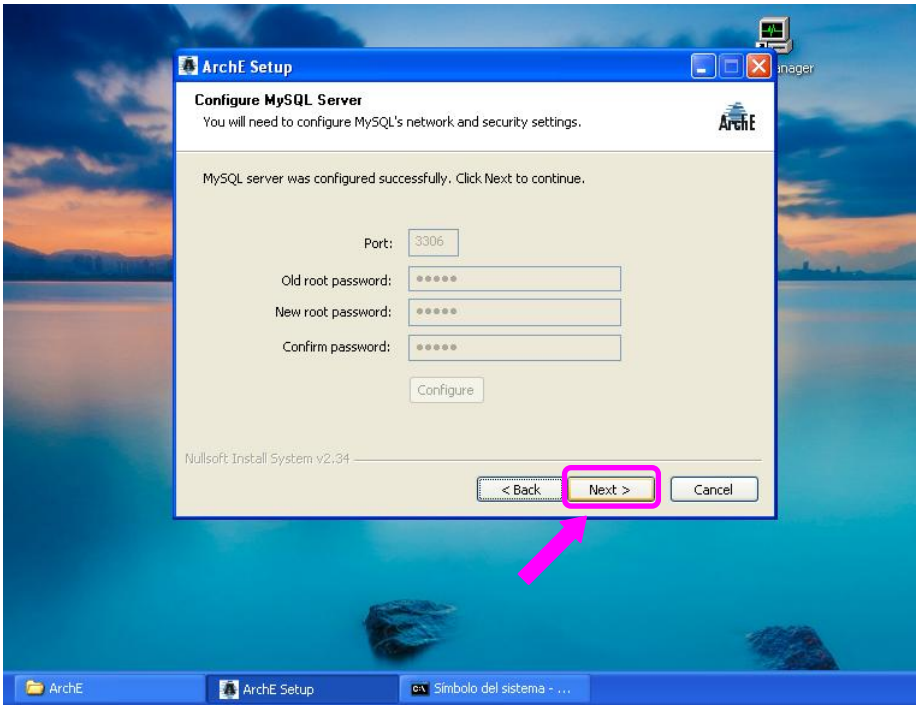


Figura 292. Siguiente Paso para configurar MySQL en Arche.

A continuación se creará la base de datos de ArchE y se pulsa Next >: (Figura 293 y Figura 294)

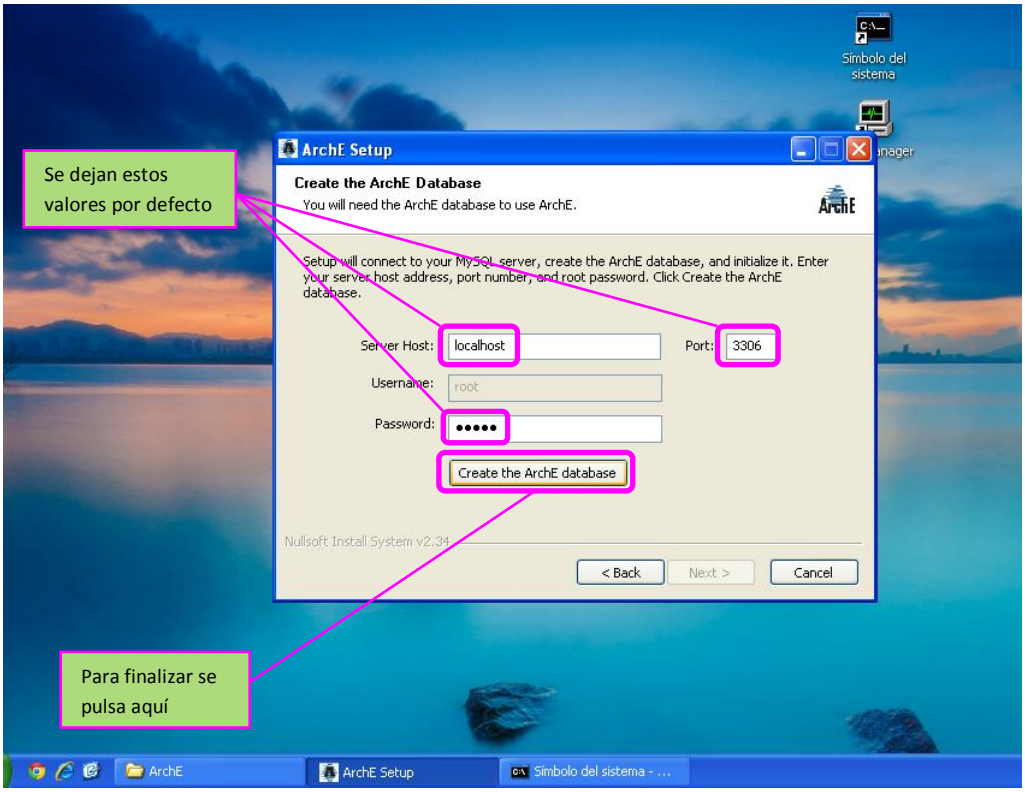


Figura 293. Creando la Base de Datos de ArchE.

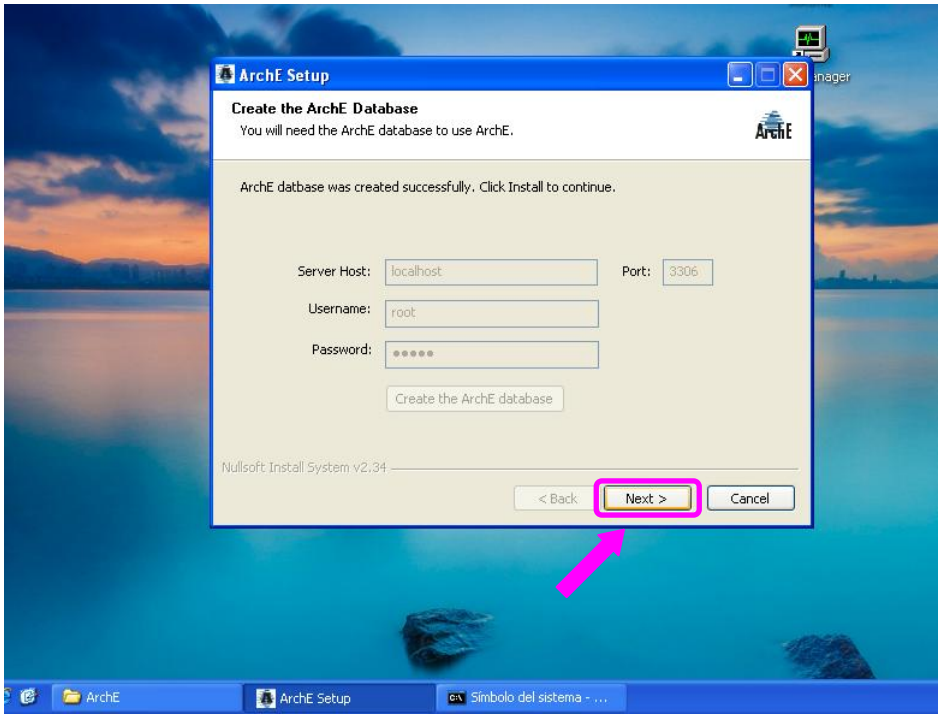


Figura 294. La Base de Datos de ArchE se creó correctamente.

Se configura la conexión con el servidor de xmlBlaster y se pulsa Install: (Figura 295 y Figura 296)

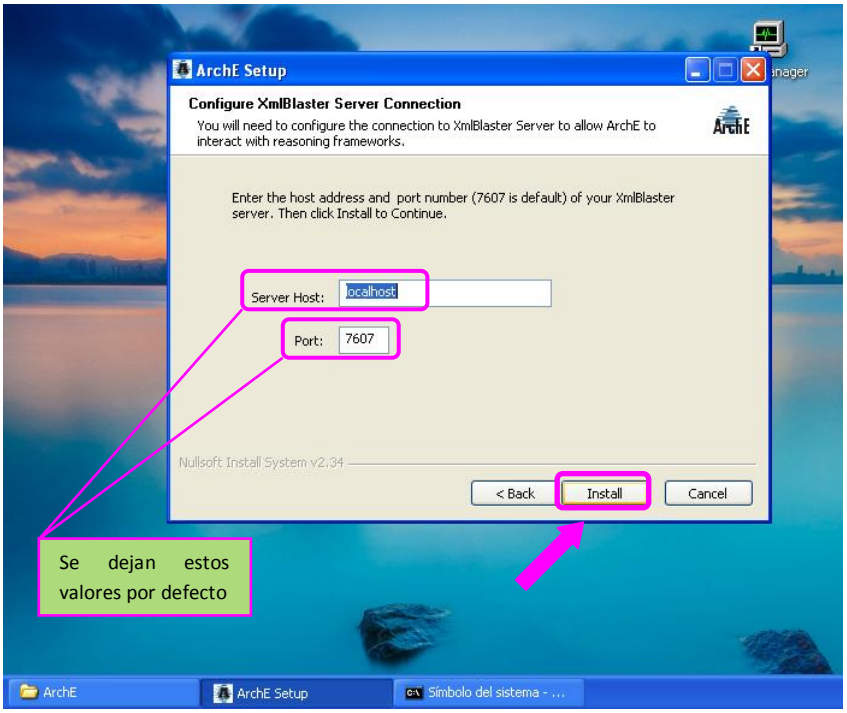


Figura 295. Configuración del host de xmlBlaster en ArchE.

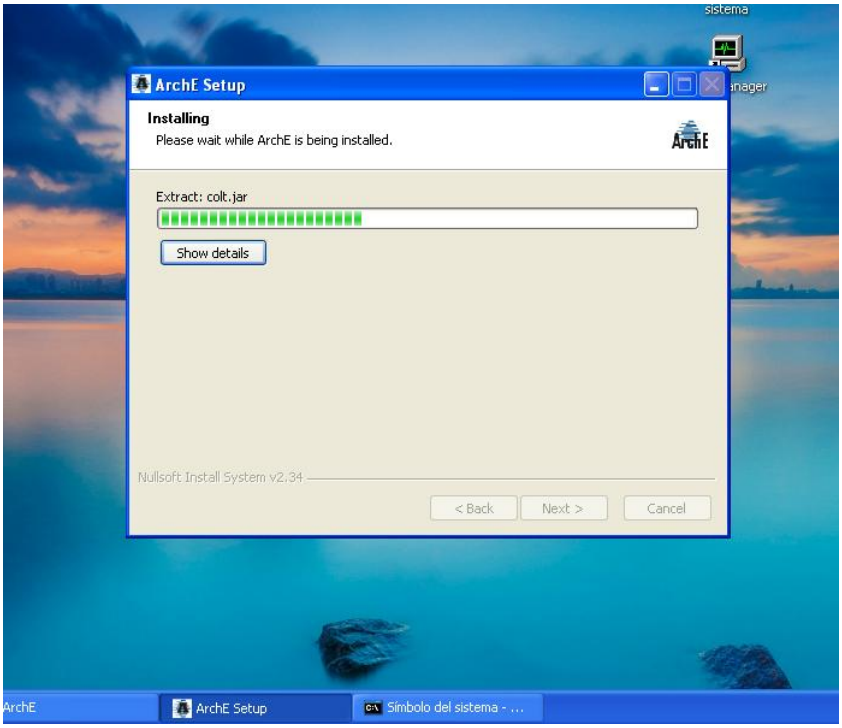


Figura 296. Finalizando la Instalación de ArchE.

Una vez completada la instalación, se pulsa Next >: (Figura 297)

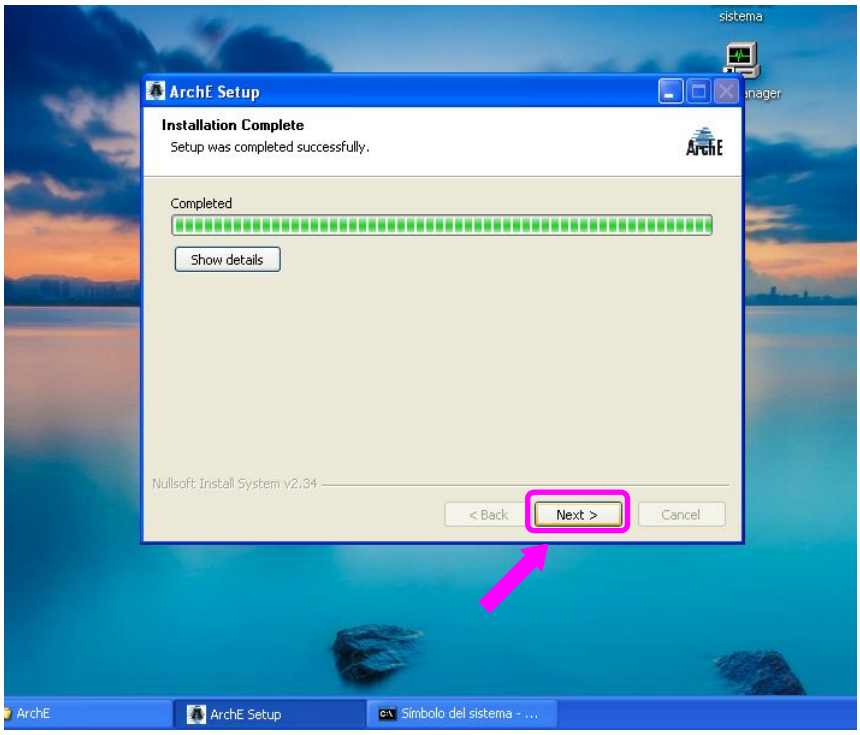


Figura 297. ArchE ha Finalizado su Instalación.

Una vez finalizado el proceso de instalación, se pulsa Finish para salir: (Figura 298)

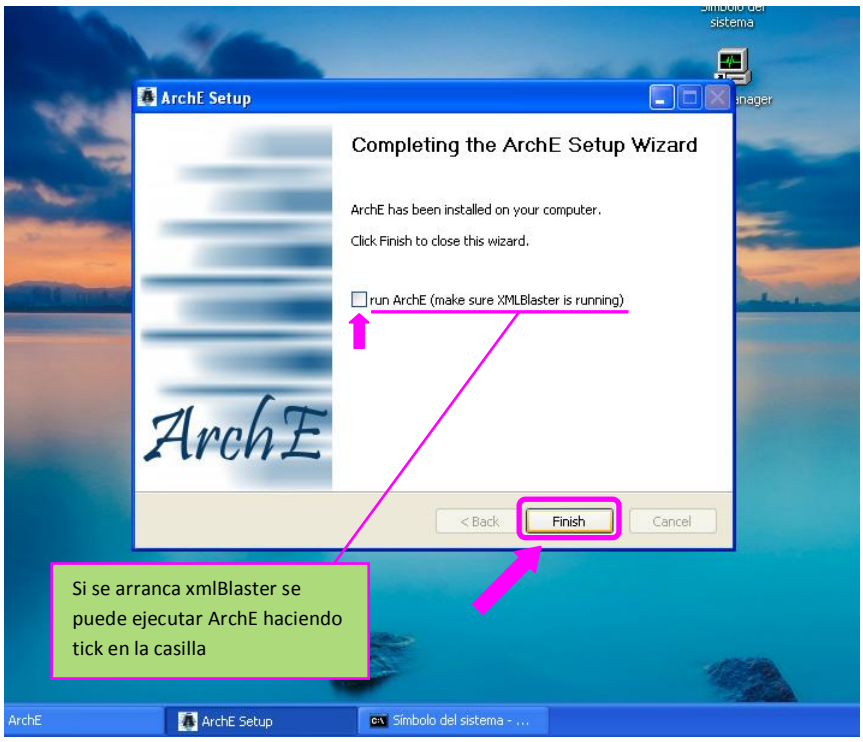


Figura 298. ArchE está instalado y se puede ejecutar.

Una vez iniciado, ArchE presenta esta pantalla de menús: (Figura 299)

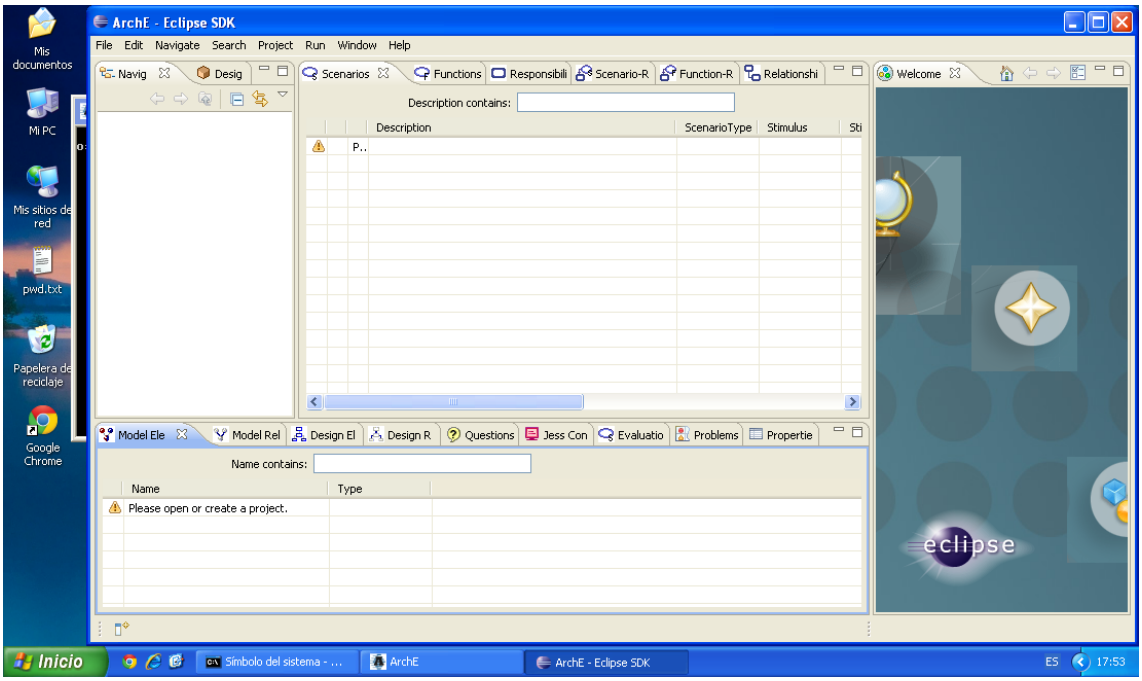


Figura 299. Pantalla de Menús de ArchE.

Un paso muy importante de configuración consiste en añadir las ventanas de los marcos de razonamiento: (Figura 300 y Figura 301)

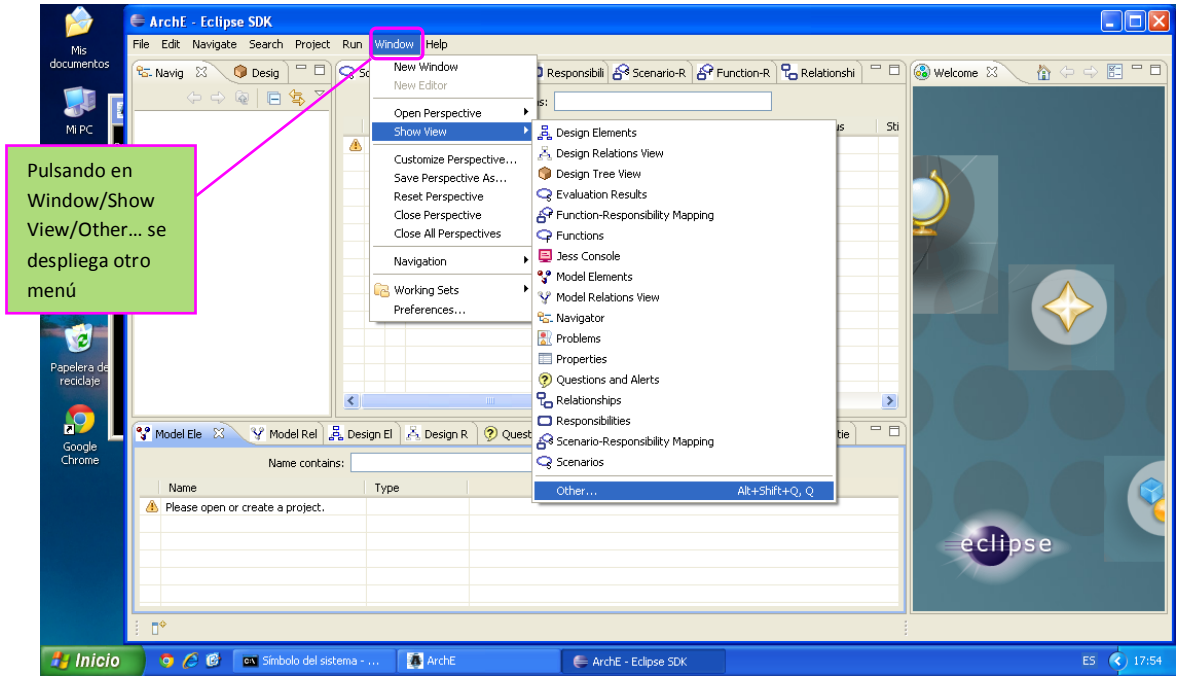


Figura 300. Configuración de Ventanas de Marcos de Razonamiento.

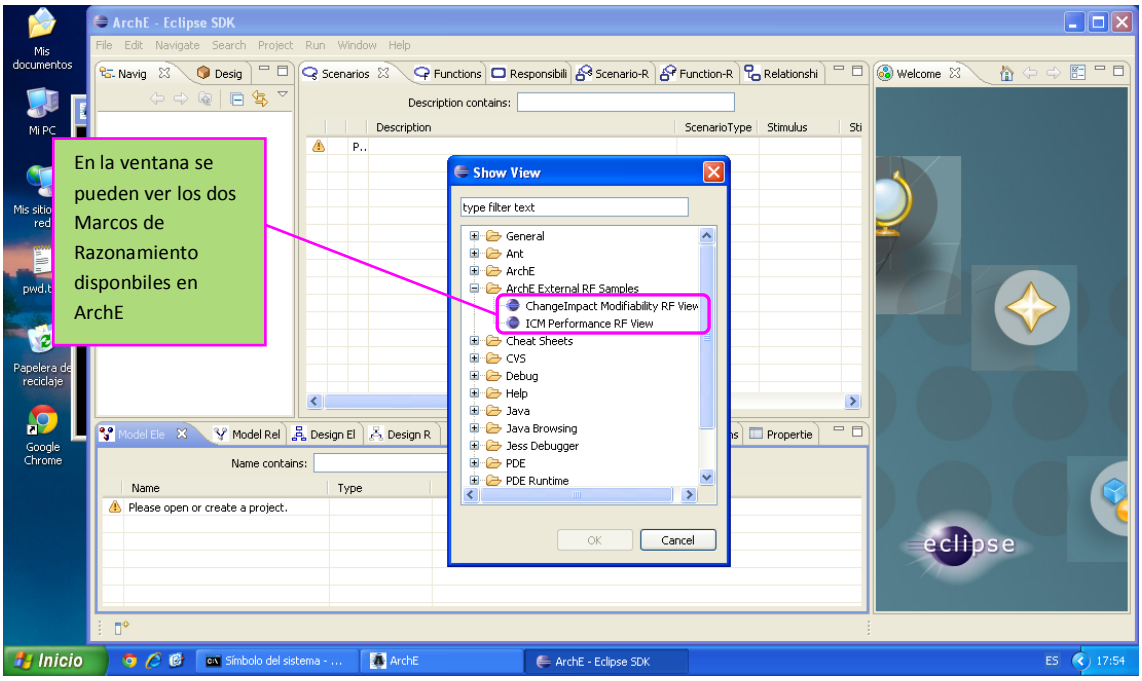


Figura 301. Selección de las Vistas de los Marcos de Razonamiento.

Seleccionando ambos, ChangeImpact Modifiability RF View y ICM Performance RF View, las ventanas de ambos marcos de razonamiento quedan instaladas en la vista de menús general; es importante arrancar ambos marcos para poder ejecutar ArchE apropiadamente: (Figura 302)

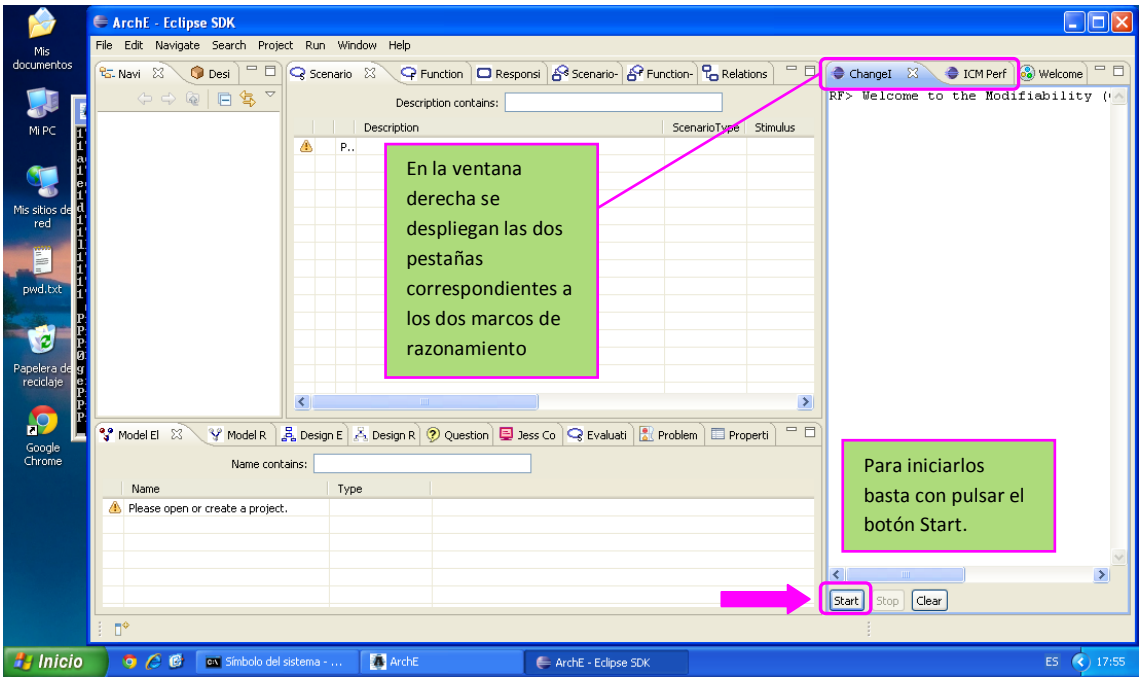


Figura 302. Vista Final de Arche con las Ventanas de los Marcos de Razonamiento.

13.9 Método Alternativo de Instalación de ArchE

Otro método alternativo de instalación de ArchE viene detallado en la página web:

https://github.com/Arch-E

que es un repositorio de software en el que se encuentran los paquetes necesarios para instalar las versiones más recientes de ArchE y sus marcos de razonamiento.

En la dirección web:

https://github.com/Arch-E/arche-core

Se descargan los componentes descritos en esta guía y los paquetes de ArchE, y además hay una guía paso a paso de instalación.

Y en la dirección web:

https://github.com/Arch-E/arche.modifChangeImpact

https://github.com/Arch-E/arche.performance

se pueden descargar las versiones más actualizadas de los marcos de razonamiento de modificabilidad y rendimiento de ArchE.

14 ANEXO 3 – GUÍA DE USUARIO DE ARCHE

14.1 Guía de Usuario de ArchE

En este apartado se dará una guía rápida para aprender a utilizar ArchE.

Los pasos correctos en la creación de un proyecto son los siguientes:

1.- Arrancar XmlBlaster.
2.- Iniciar ArchE.
3.- Crear un nuevo Proyecto
4.- Introducir Funciones y Responsabilidades
5.- Asignar relaciones entre responsabilidades
6.- Introducir escenarios
7.- Mapear escenarios a responsabilidades
8.- Analizar resultados

14.1.1 Arrancar xmlBlaster

xmlBlaster es necesario para que ArchE pueda comunicarse con los marcos de razonamiento; por tanto, lo primero que hay que hacer es iniciarlo, mediante la línea de comandos: (Figura 303)

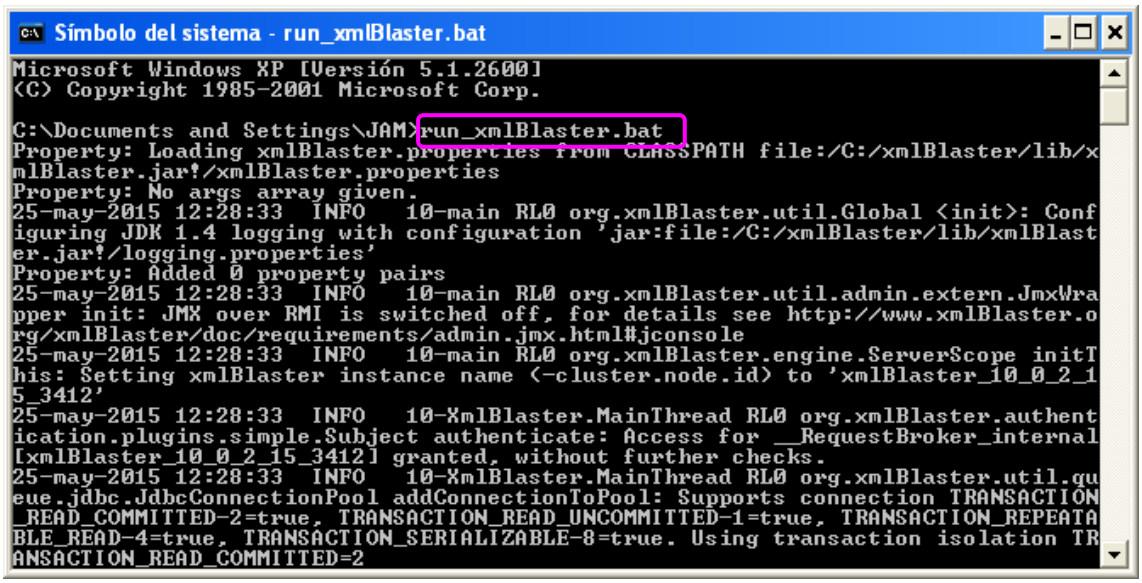


Figura 303. Arrancar XmlBlaster.

Una vez finalice el arranque del fichero run_xmlBlaster. bat, hay que presionar “r”.

14.1.2 Iniciar ArchE

Para iniciar ArchE, se hace doble clic en el icono de ArchE y a continuación el entorno Eclipse se cargará: (Figura 304)

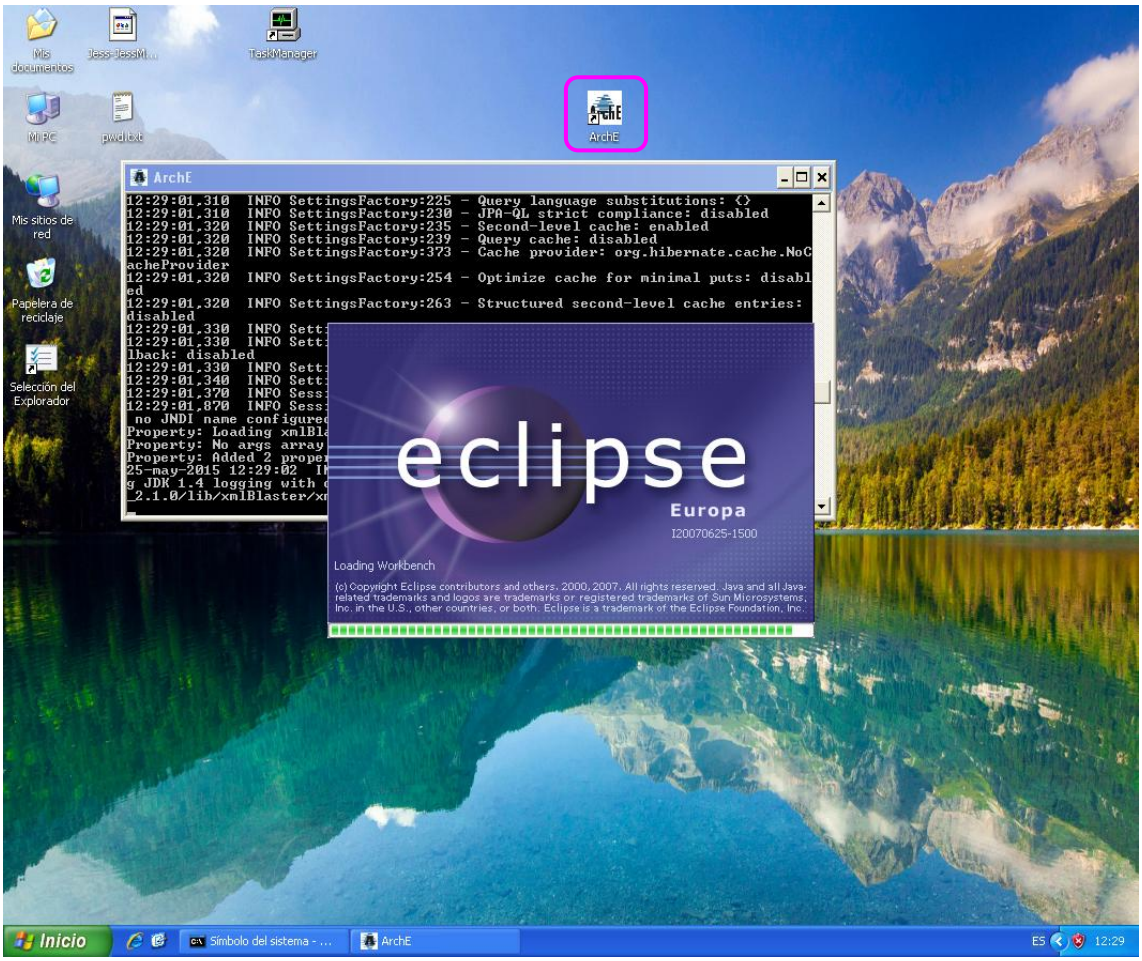


Figura 304. Aranque de ArchE.

El entorno ArchE se carga, como se puede ver en la Figura 305:

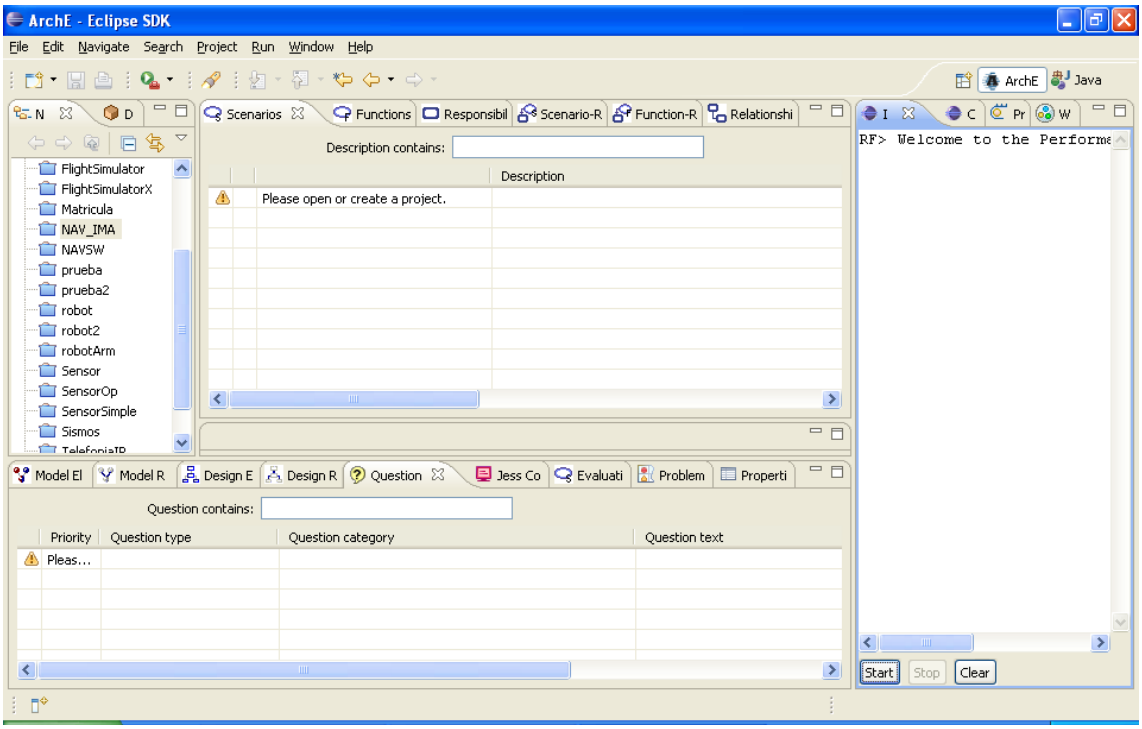


Figura 305. Entorno ArchE.

14.1.3 Creación de un Proyecto en ArchE

Para crear un proyecto, hay que ir a la opción File/New/Arche Project: (Figura 306)

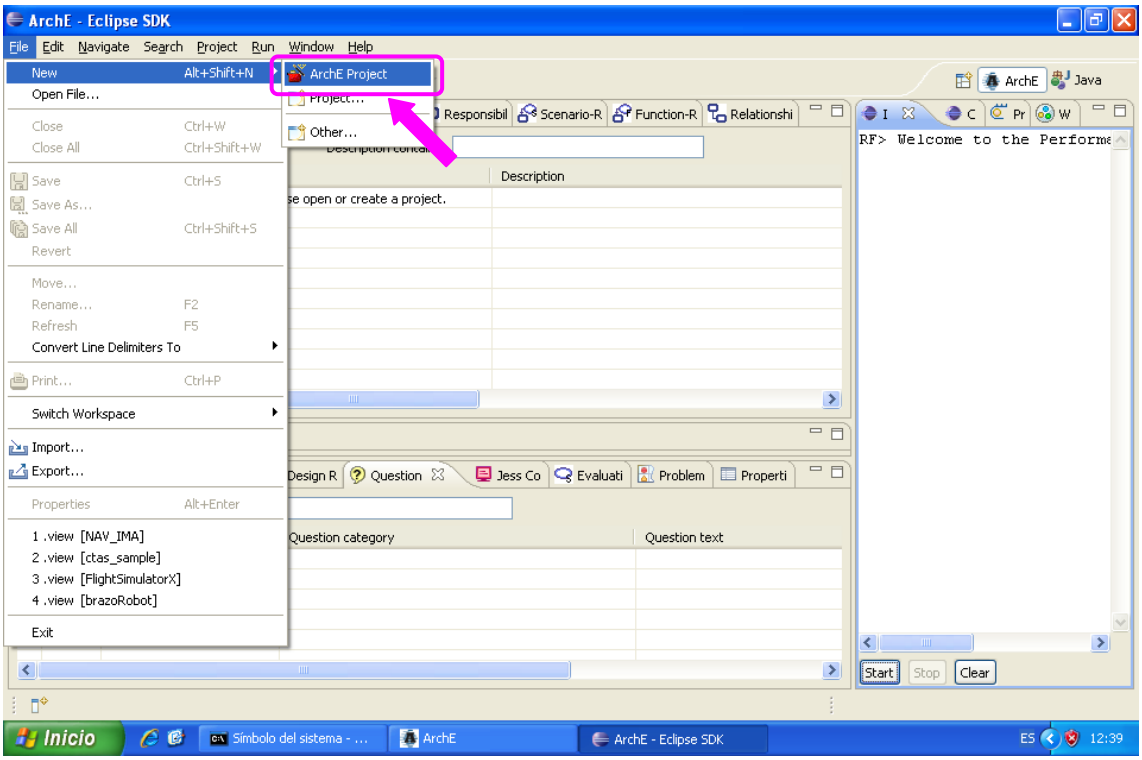


Figura 306. Creación de un Nuevo Proyecto en ArchE.

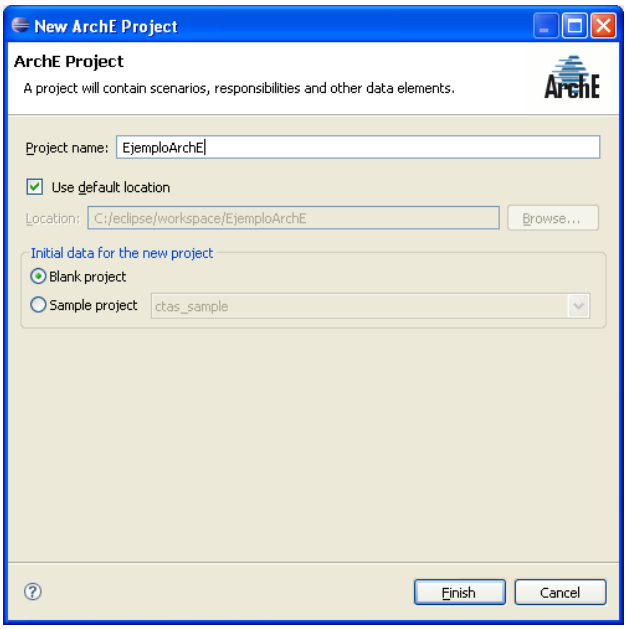


Figura 307. Nuevo Proyecto de ArchE.

Una vez abierto, ya se puede trabajar con el proyecto. Es importantísimo recalcar que todo cambio que se haga en el proyecto no se salvará si no se sigue el siguiente procedimiento: (Figura 308)

Project/Persist fact base

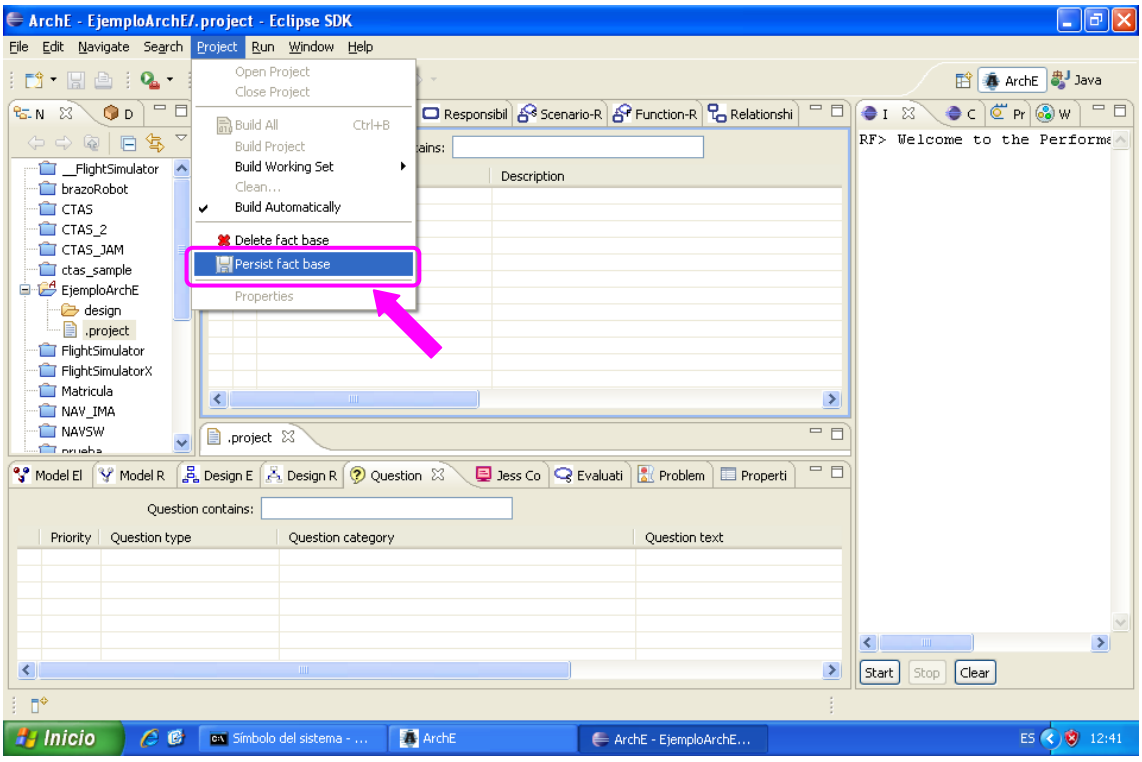


Figura 308. Guardar Cambios en Proyecto ARchE.

14.1.4 Introducir Responsabilidades

Una vez generado el proyecto, a continuación se introducirán sus funciones, haciendo clic en el botón derecho del ratón y seleccionando “New function”: (Figura 309, Figura 310 y Figura 311)

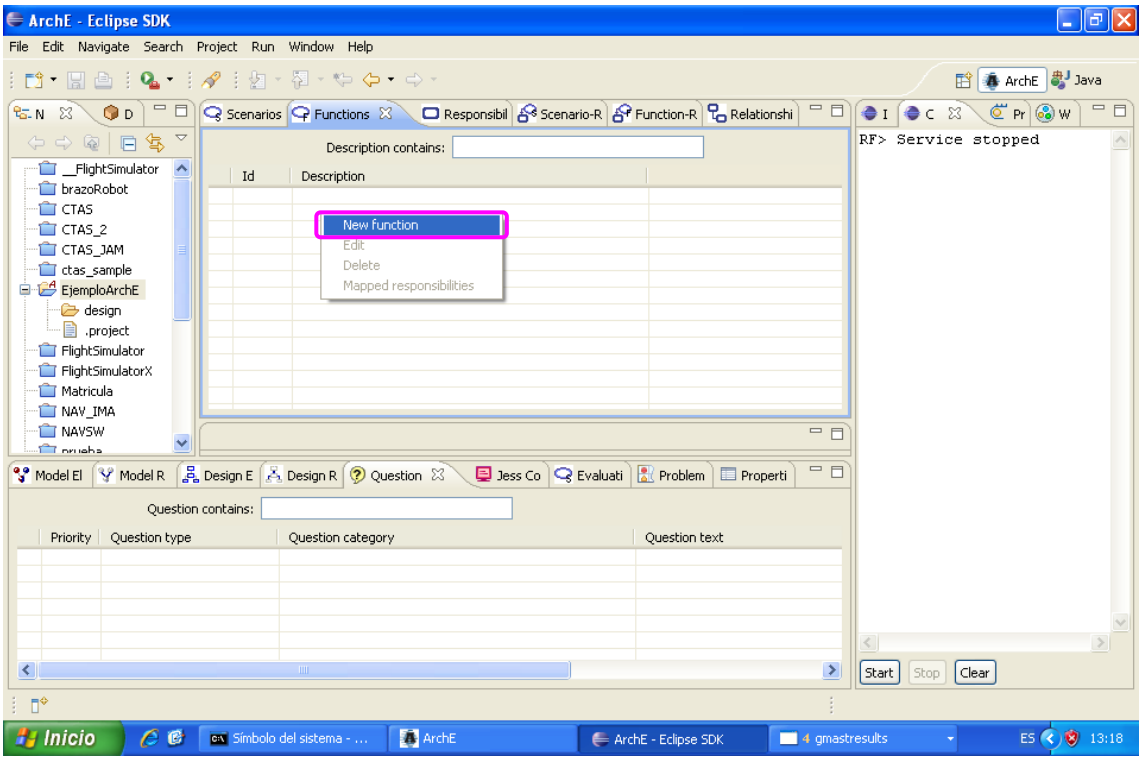


Figura 309. Introducir Funciones en Proyecto ArchE.

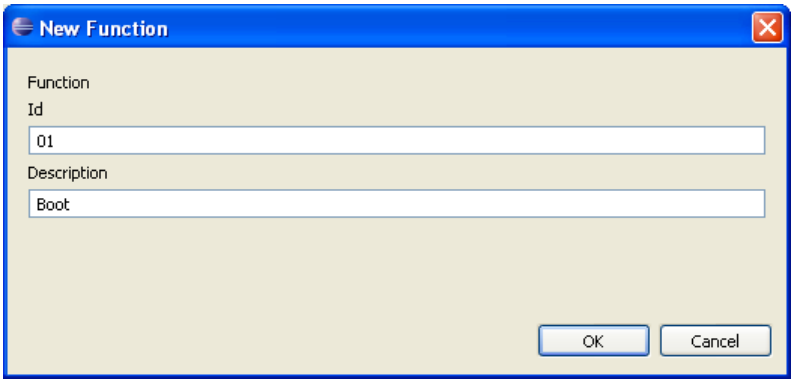


Figura 310. Nueva Función.

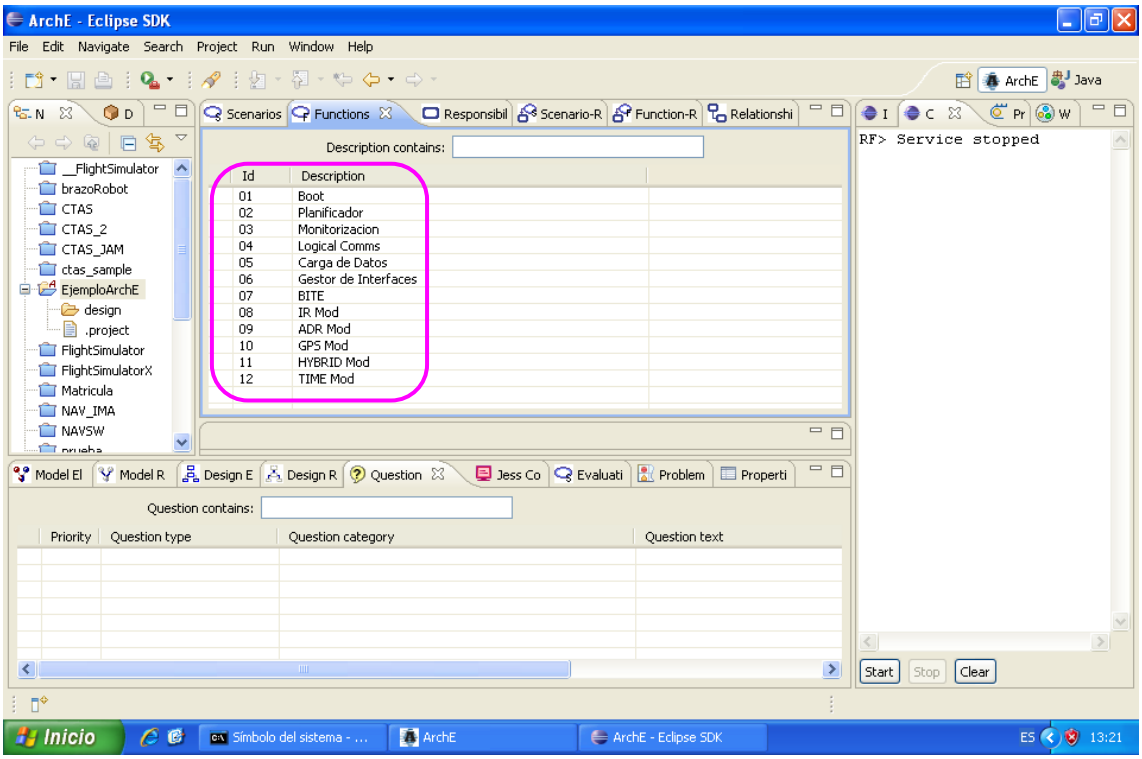


Figura 311. Las Funciones están introducidas en ArchE.

Las responsabilidades son asignadas de manera inmediata por ArchE a su respectiva funcionalidad. El arquitecto puede después modificar o asignar nuevas responsabilidades. (Figura 312 y Figura 313)

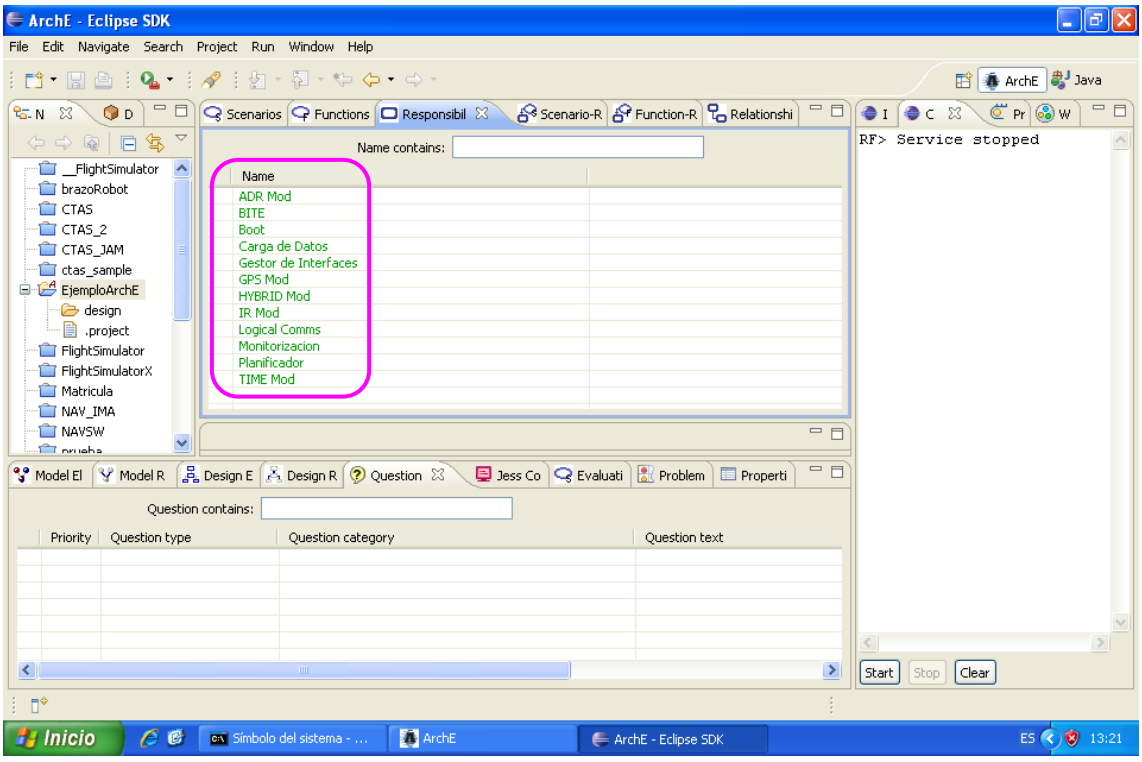


Figura 312. Introducción Automática de Responsabilidades por ArchE.

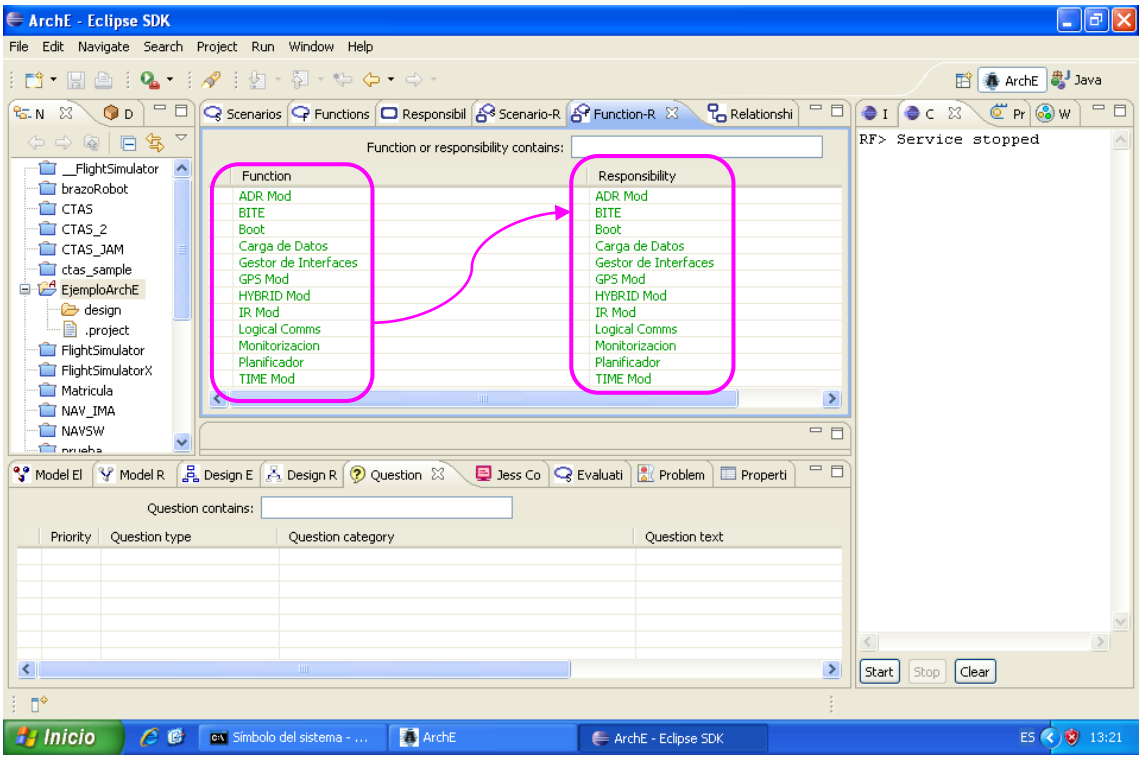


Figura 313. Mapeo Funciones-Responsabilidades en ArchE.

14.1.5 Asignar Relaciones entre Responsabilidades

En este paso, se establecerán las relaciones entre responsabilidades. Es muy importante arrancar ahora los dos marcos de razonamiento, modifiability y performance, puesto que si no, no se podrán establecer dichas relaciones: (Figura 314)

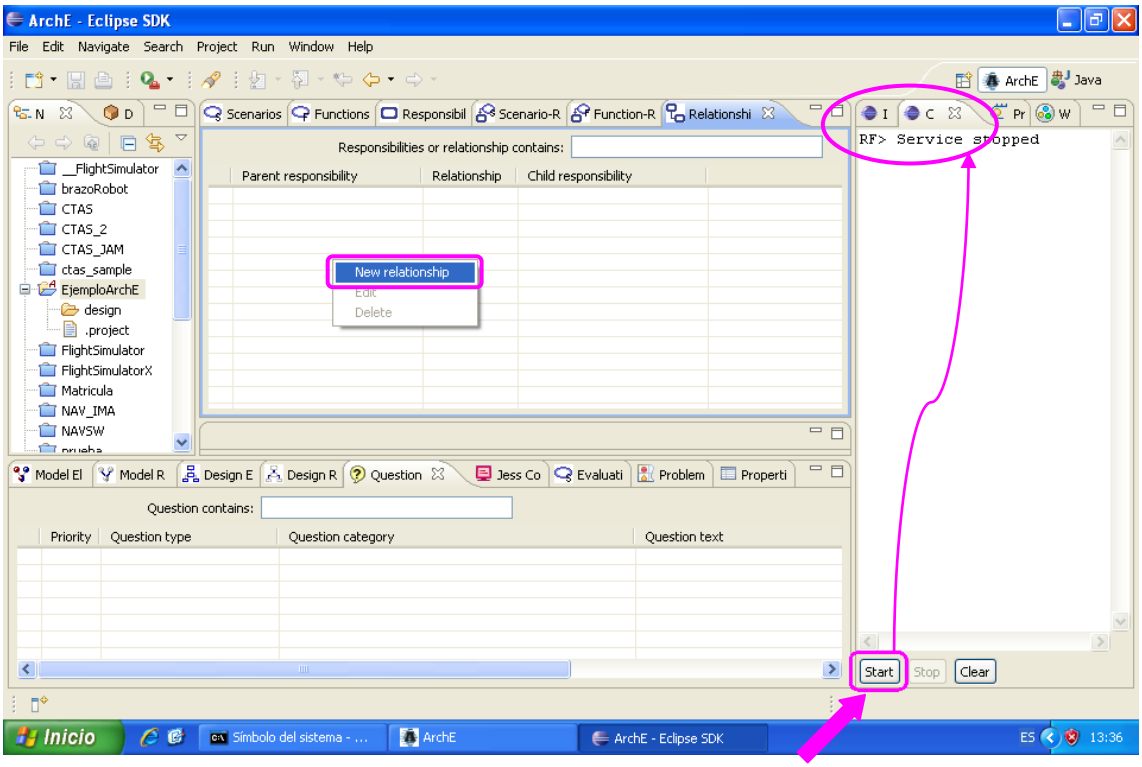


Figura 314. Establecer una nueva Relación entre Responsabilidades en ArchE.

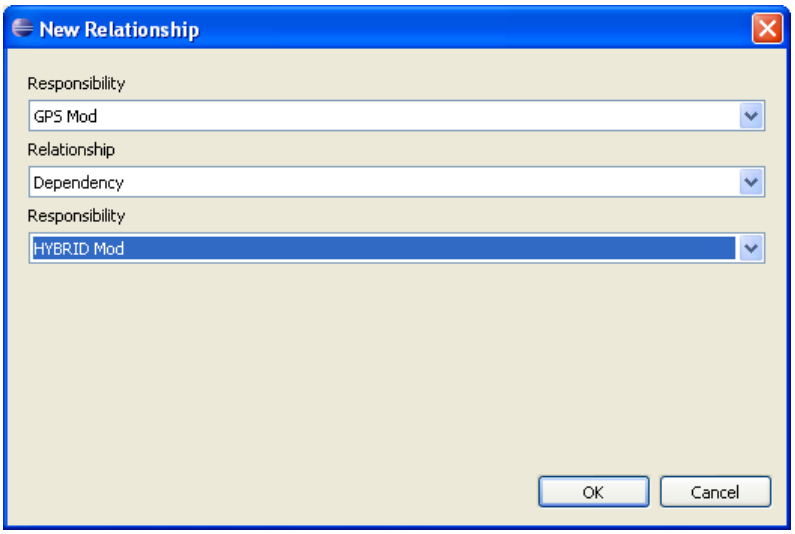


Figura 315. Selección de Responsabilidades y Tipo de Relación entre Ellas.

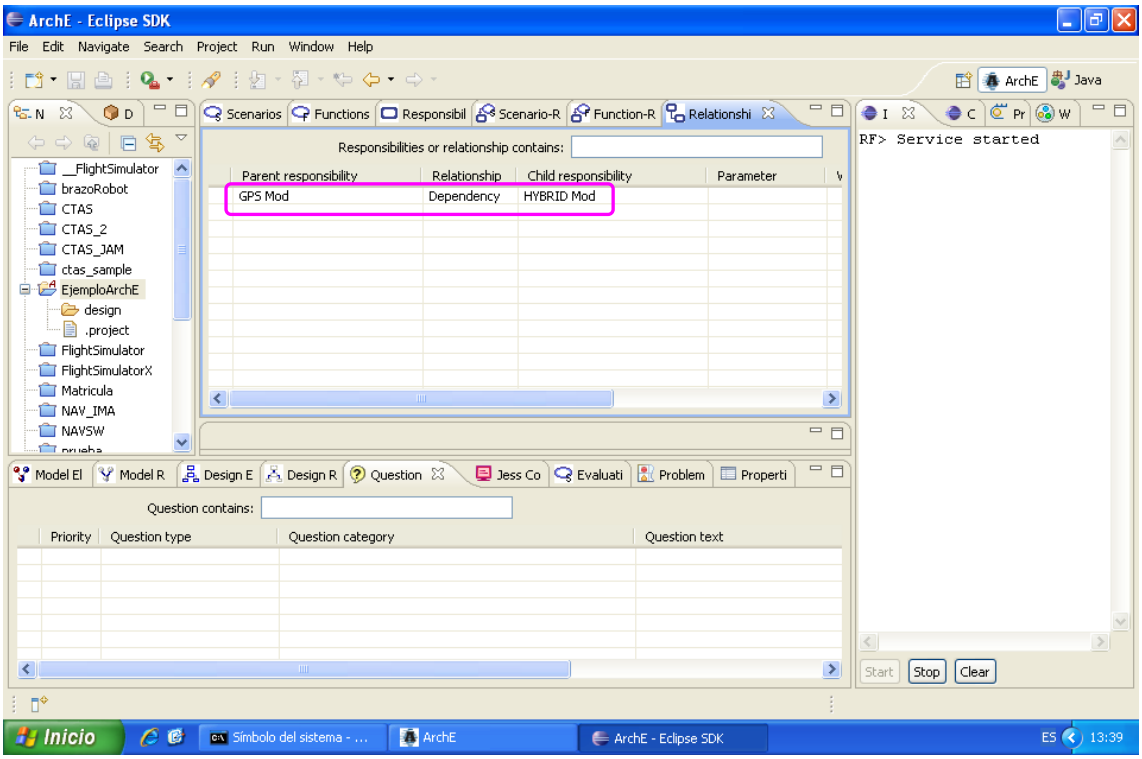


Figura 316. La Relación entre Responsabilidades queda creada.

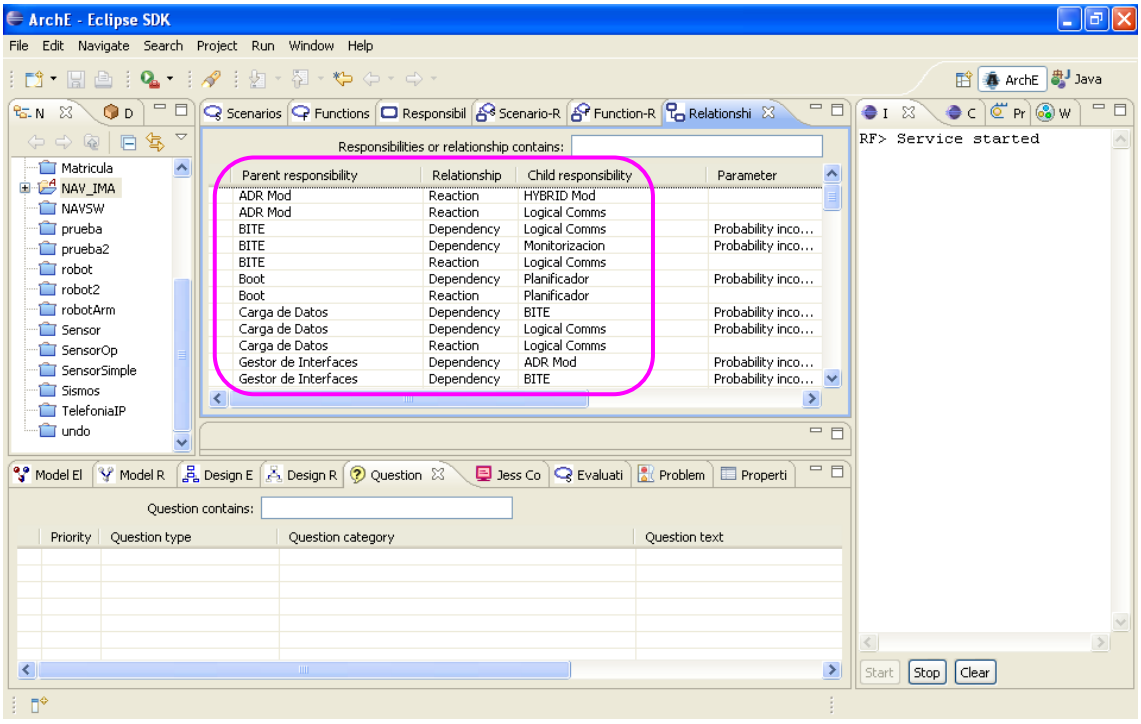


Figura 317. Conjunto de Relaciones en el Proyecto ArchE.

14.1.6 Introducir Escenarios

El paso siguiente es introducir los escenarios de atributos de calidad; para ello, los marcos de razonamiento han de seguir arrancados: (Figura 318)

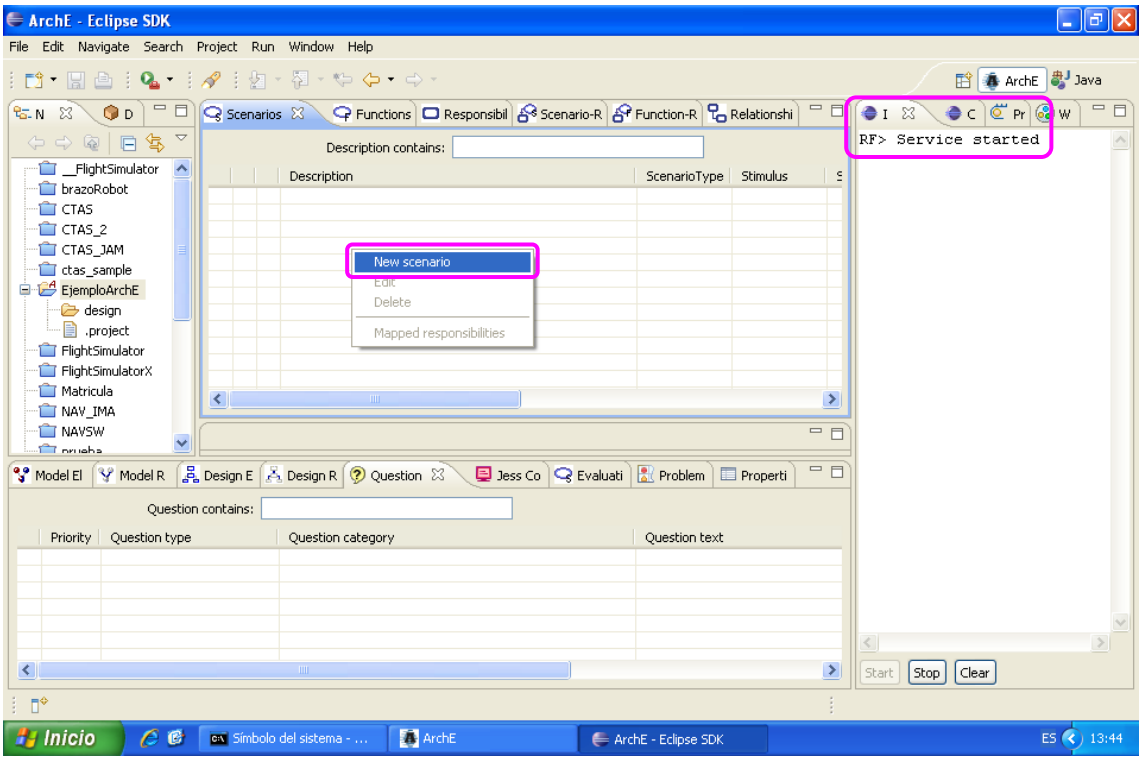


Figura 318. Introducir Nuevo Escenario en ARchE.

Se abre una ventana en la cual se introducen los parámetros del escenario, así como el tipo de marco de razonamiento: (Figura 319 y Figura 320)

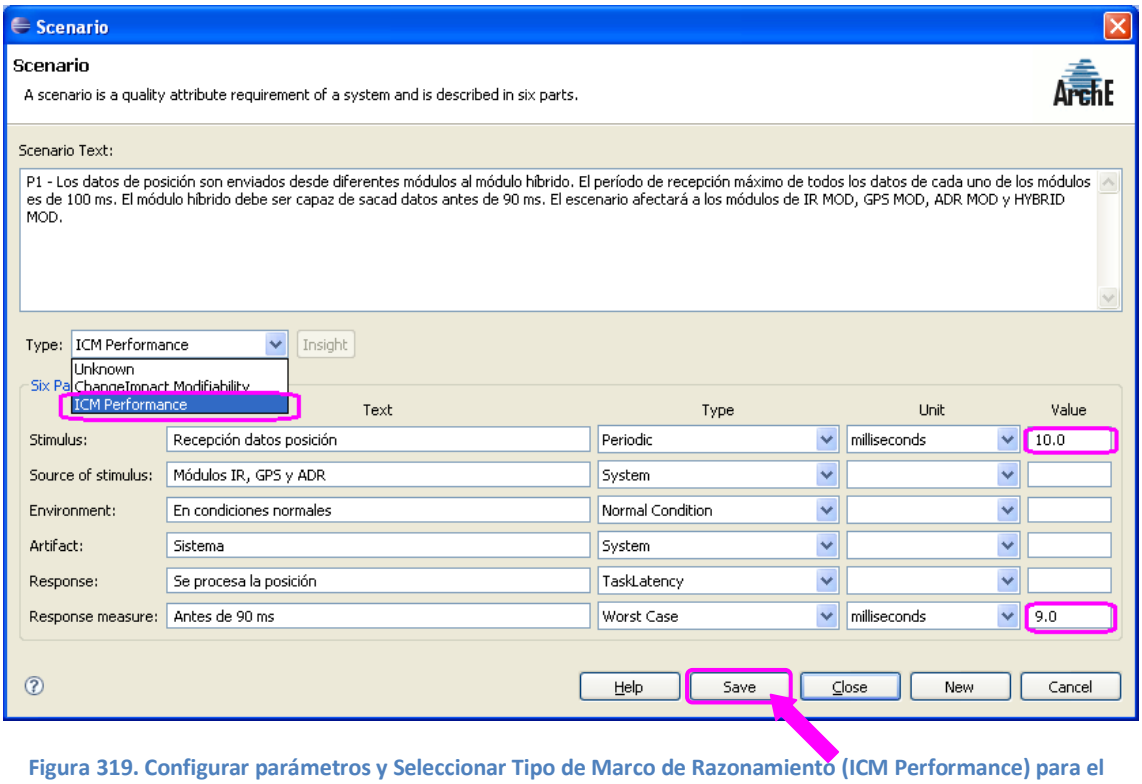


Figura 319. Configurar parámetros y Seleccionar Tipo de Marco de Razonamiento (ICM Performance) para el Escenario.

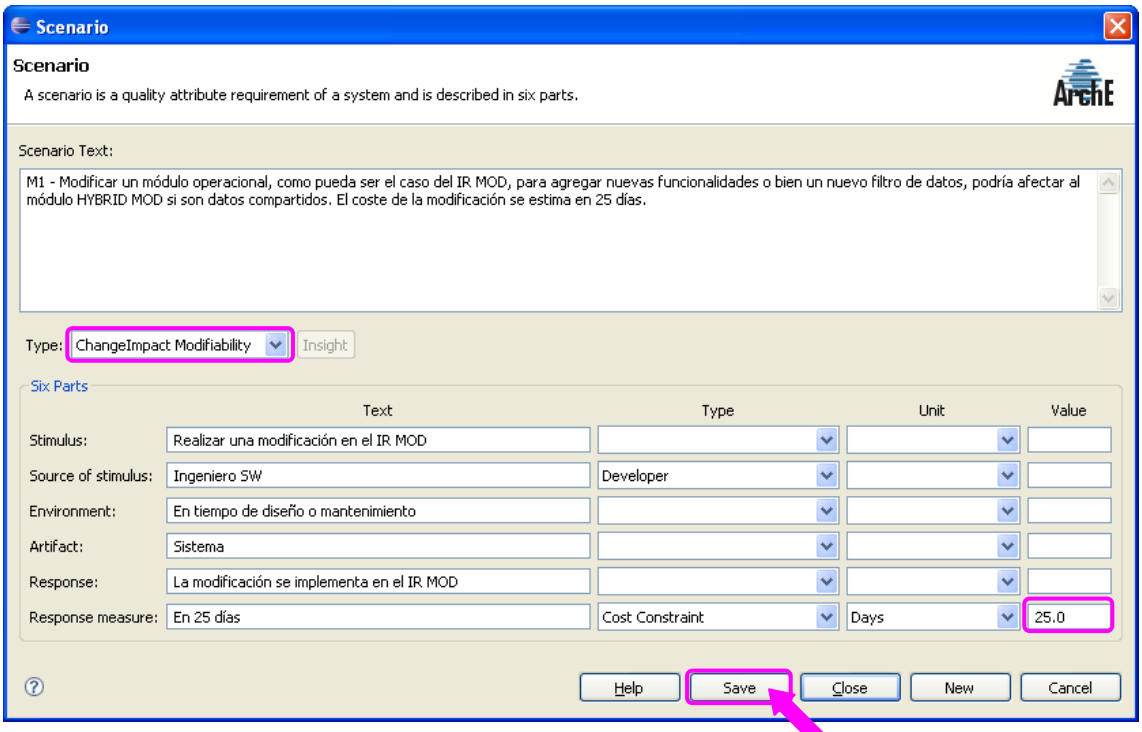


Figura 320. Configurar parámetros y Seleccionar Tipo de Marco de Razonamiento (ChangeImpact Modifiability) para el Escenario.

Una vez introducidos todos los escenarios, la vista de escenarios de ArchE quedaría de la siguiente manera: (Figura 321)

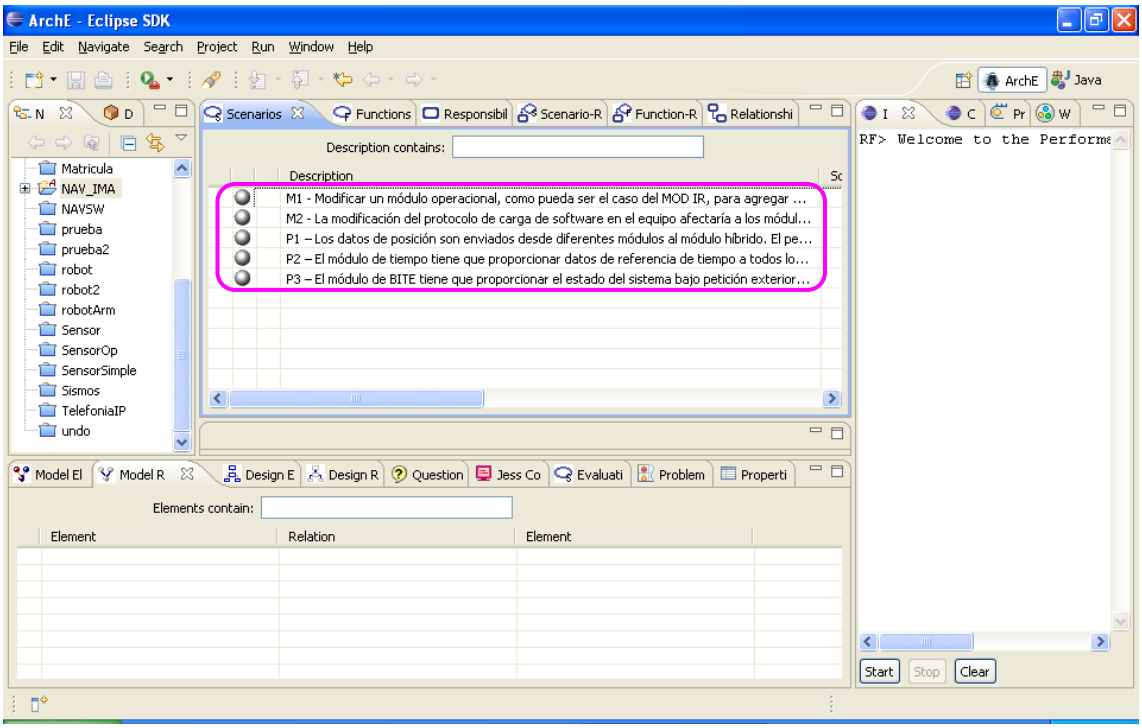


Figura 321. Escenarios de Atributos de Calidad en el Proyecto ArchE.

14.1.7 Mapeo Escenarios-Responsabilidades

Por último, se realiza el mapeo de escenarios a responsabilidades, haciendo como siempre clic derecho del ratón y seleccionando New Mapping en la pestaña de Scenario-Responsibility Mapping. Esto hará que los marcos de razonamiento (que permanecerán arrancados de pasos anteriores) empiecen a ejecutarse, lo que ralentizará el proceso, pues al final de dichas ejecuciones, ArchE presentará los resultados: (Figura 322)

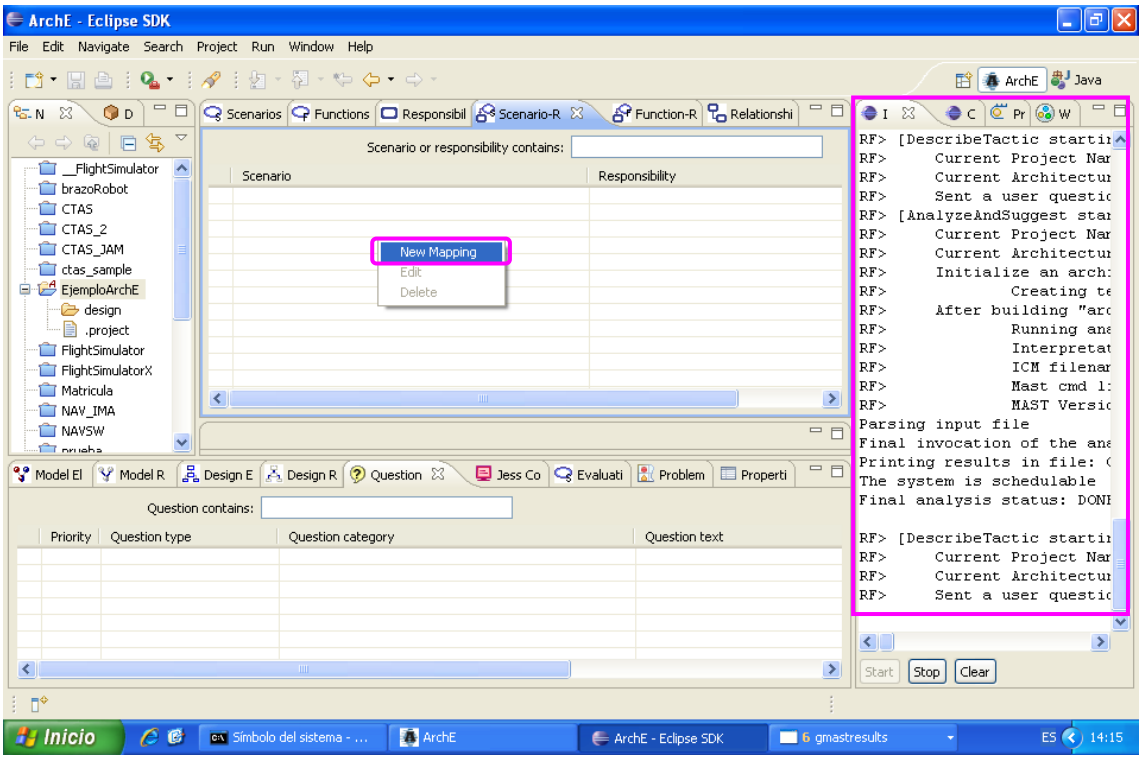


Figura 322. Mapeo Escenario-Responsabilidades.

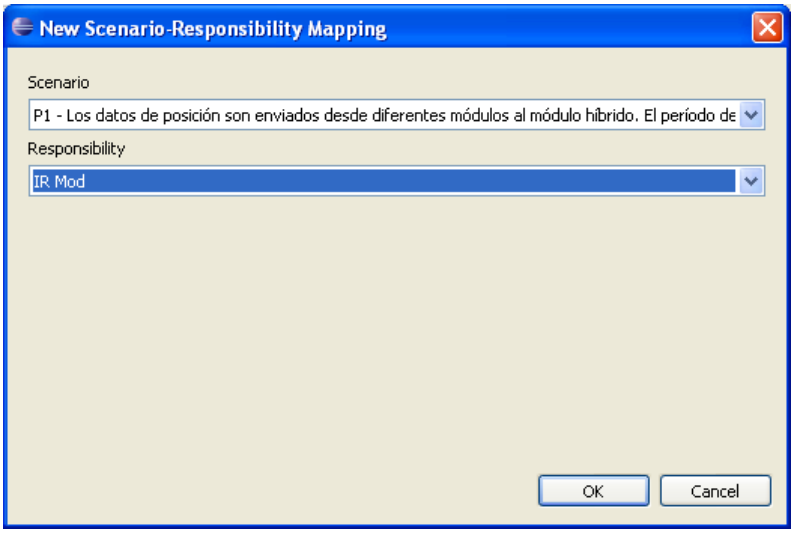


Figura 323. Se abre un Menú de Selección de Escenario y Responsabilidad Asociada.

El mapeo queda registrado en ArchE: (Figura 324)

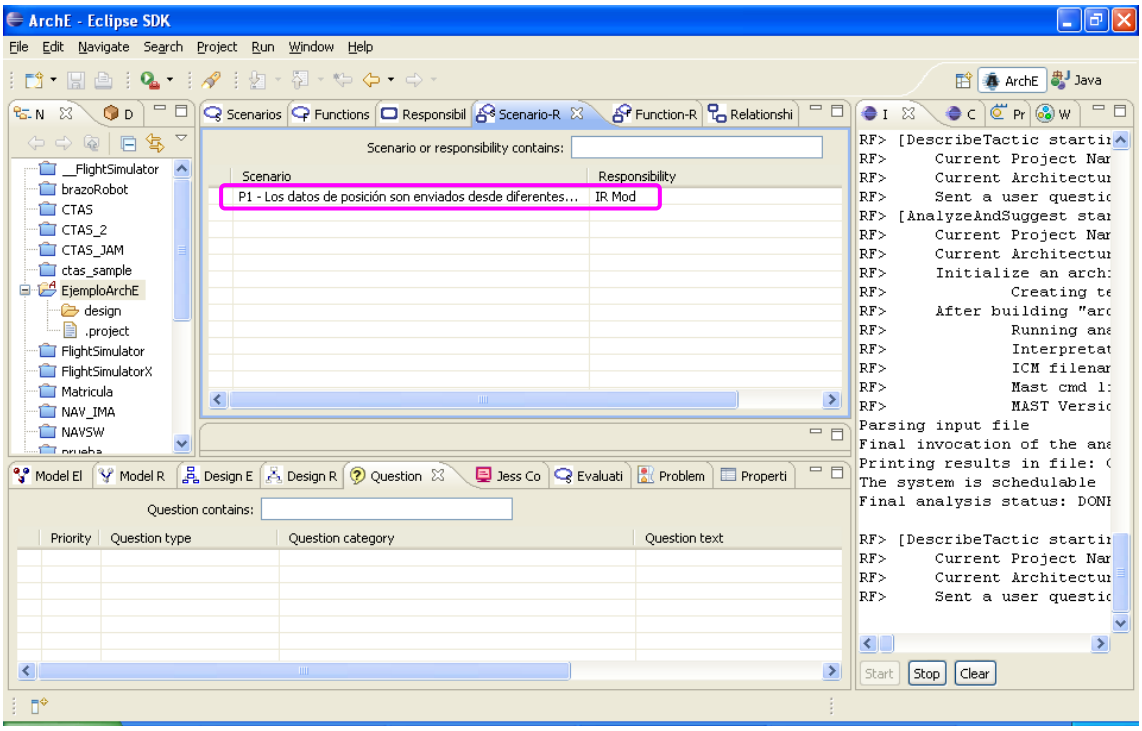


Figura 324. Mapeo Escenario-Responsabilidad ha sido creado.

Al final del proceso, todos los escenarios han de quedar mapeados a sus respectivas responsabilidades: (Figura 325)

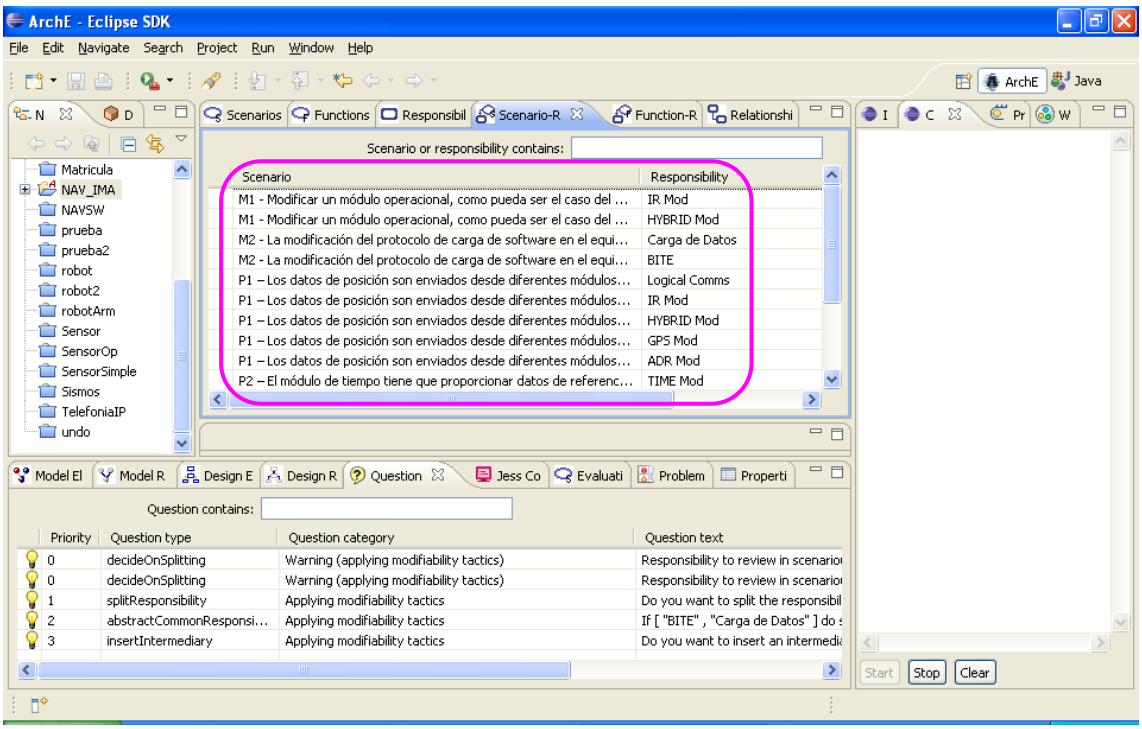


Figura 325. Mapeo Final Escenarios-Responsabilidades.

Para el resto de la guía, se utilizará el proyecto NAV_IMA desarrollado durante el trabajo.

14.1.8 Análisis de Resultados

En la pantalla inicial de escenarios pueden verse los resultados obtenidos en los escenarios, una vez los marcos de razonamiento han sido ejecutados: (Figura 326)

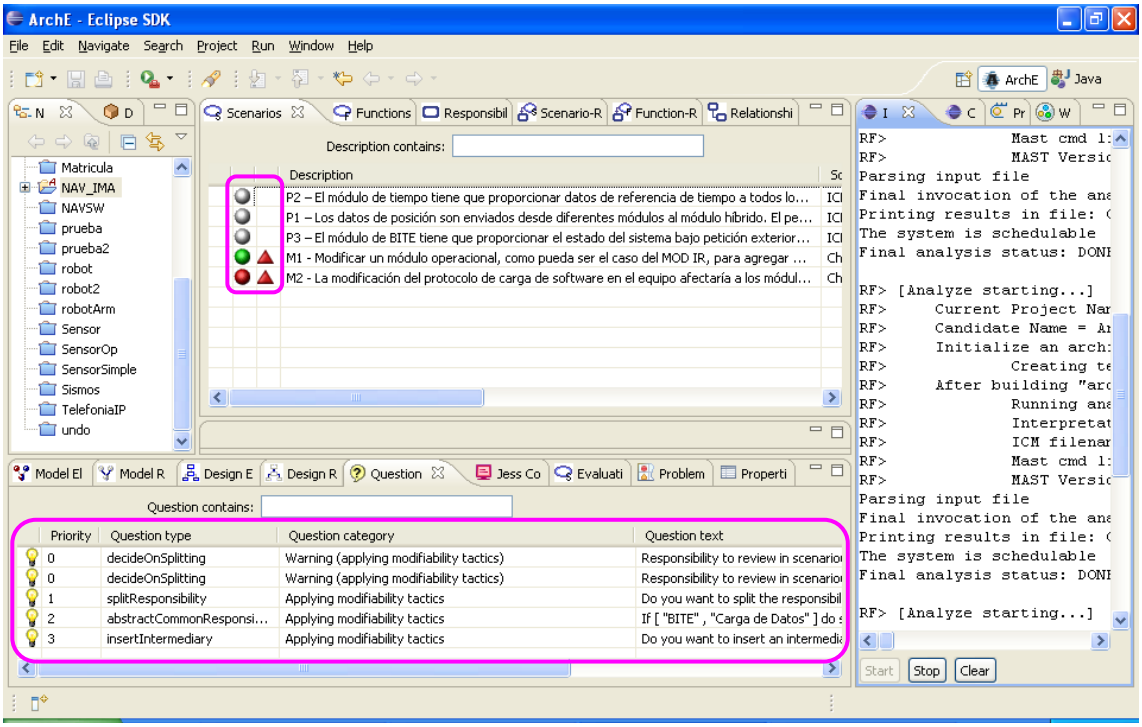


Figura 326. La Vista Escenarios muestra el Estado de Cumplimiento de los Escenarios con los Atributos de Calidad.

Los resultados se interpretan de la siguiente forma:

- Un círculo rojo significa que el escenario no se ha cumplido
- Un círculo verde significa que el escenario se cumple
- Un triángulo verde significa que la última modificación trajo consigo una mejora positiva del escenario.
- Un triángulo ámbar significa que la última modificación fue indiferente y no supuso ni mejora ni empeoramiento del escenario.
- Un triángulo verde significa que la última modificación trajo consigo un empeoramiento del escenario.

Si se han definido escenarios de modificabilidad y además se han asociado a responsabilidades, al pulsar sobre “Design Tree View” se obtiene una vista UML de la arquitectura propuesta por ARchE: (Figura 327)

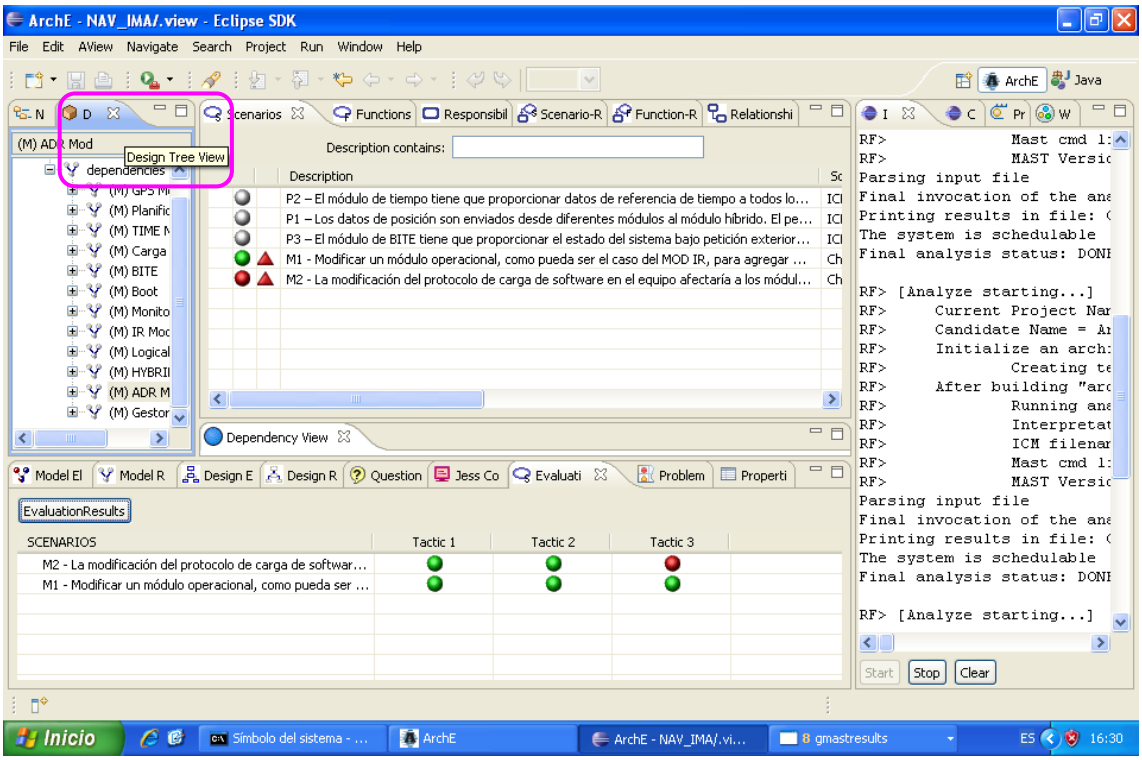


Figura 327. Selección de la Vista Arbol de Diseño en ArchE.

El diagrama UML de la arquitectura propuesto por ArchE e muestra: (Figura 328)

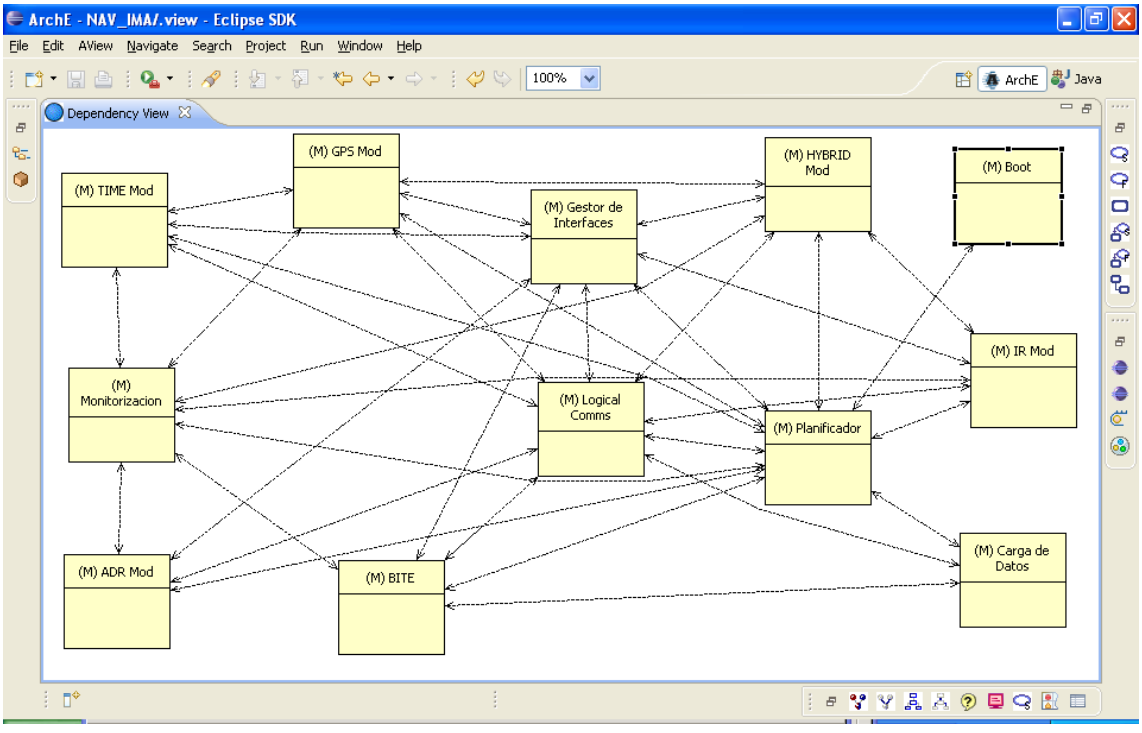


Figura 328. Arquitectura Propuesta por ArchE – Vista Design Tree.

En la pestaña “Responsibilities” pueden modificarse los parámetros de las responsabilidades, en este caso el coste del cambio y el tiempo de ejecución: (Figura 329)

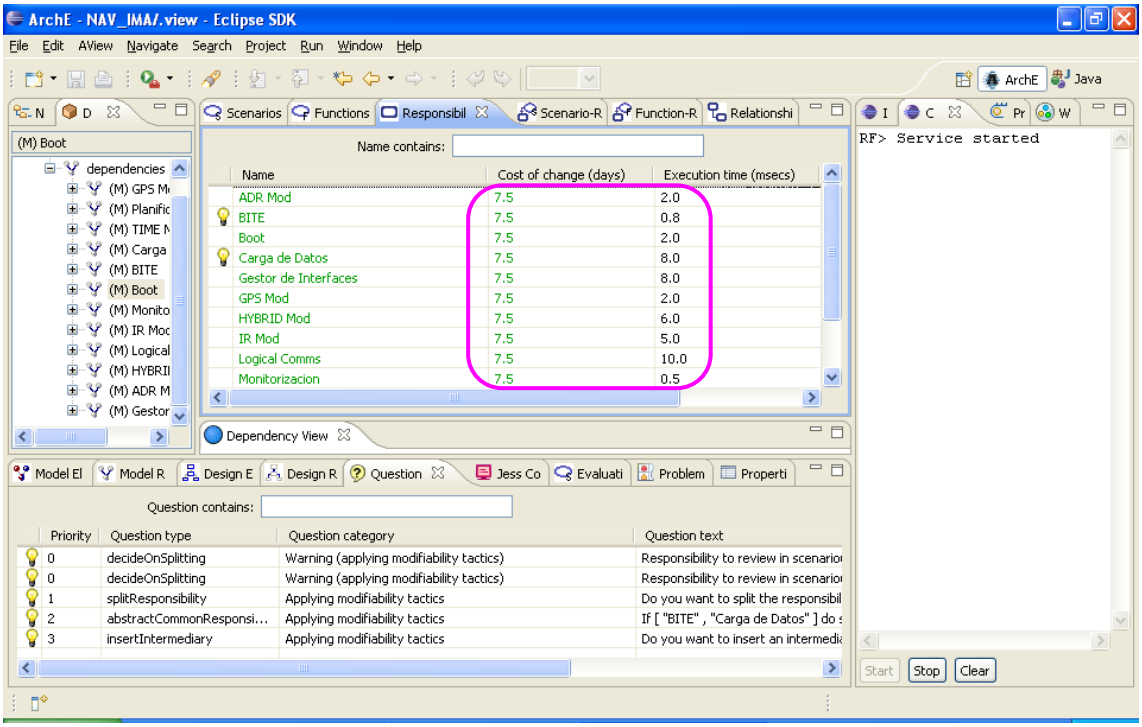


Figura 329. Vista Responsabilidades – Parámetros Coste del Cambio y Tiempo de Ejecución.

En la ventana “Relationships” pueden modificarse las probabilidades de propagación del cambio, tanto de entrada como de salida, de la responsabilidad correspondiente: (Figura 330)

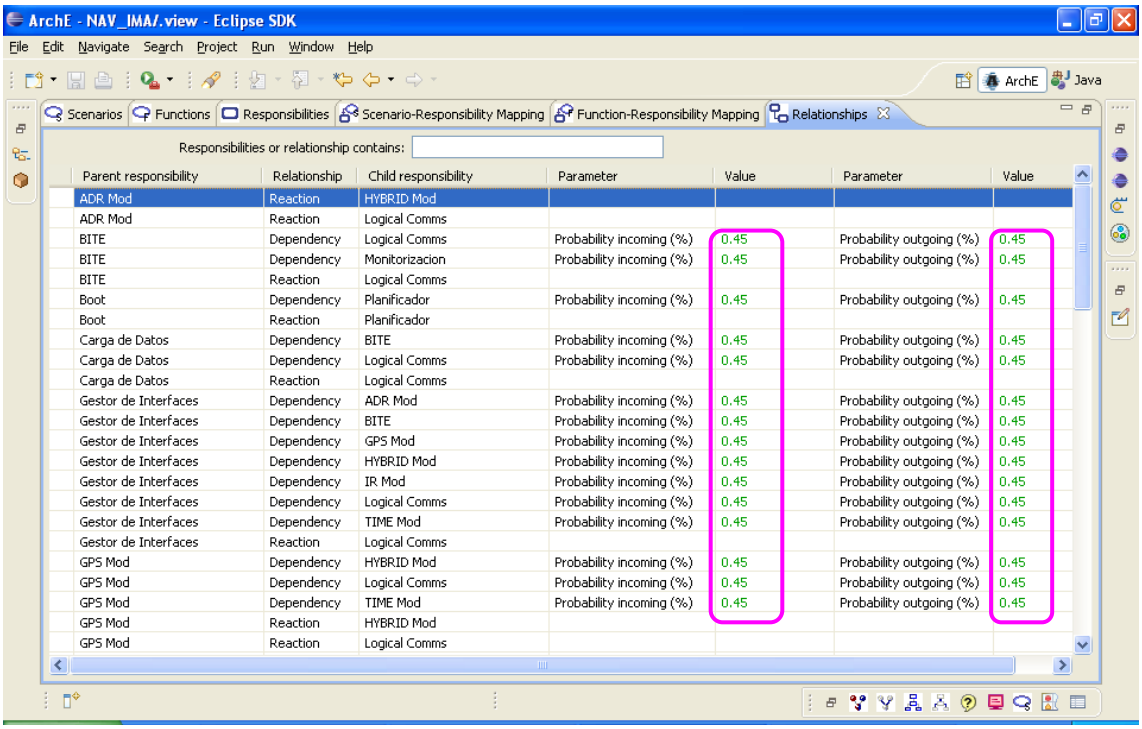


Figura 330. Vista Relaciones – la Probabilidad de Propagación del Cambio puede modificarse.

La ventana “Evaluation” propone las tácticas que pueden ser mejoras a la arquitectura; por ejemplo, en la Figura 331 las tácticas 1 y 2 mejorarían todos los escenarios, mientras que la táctica 3 mejoraría el escenario 1 y empeoraría el 2:

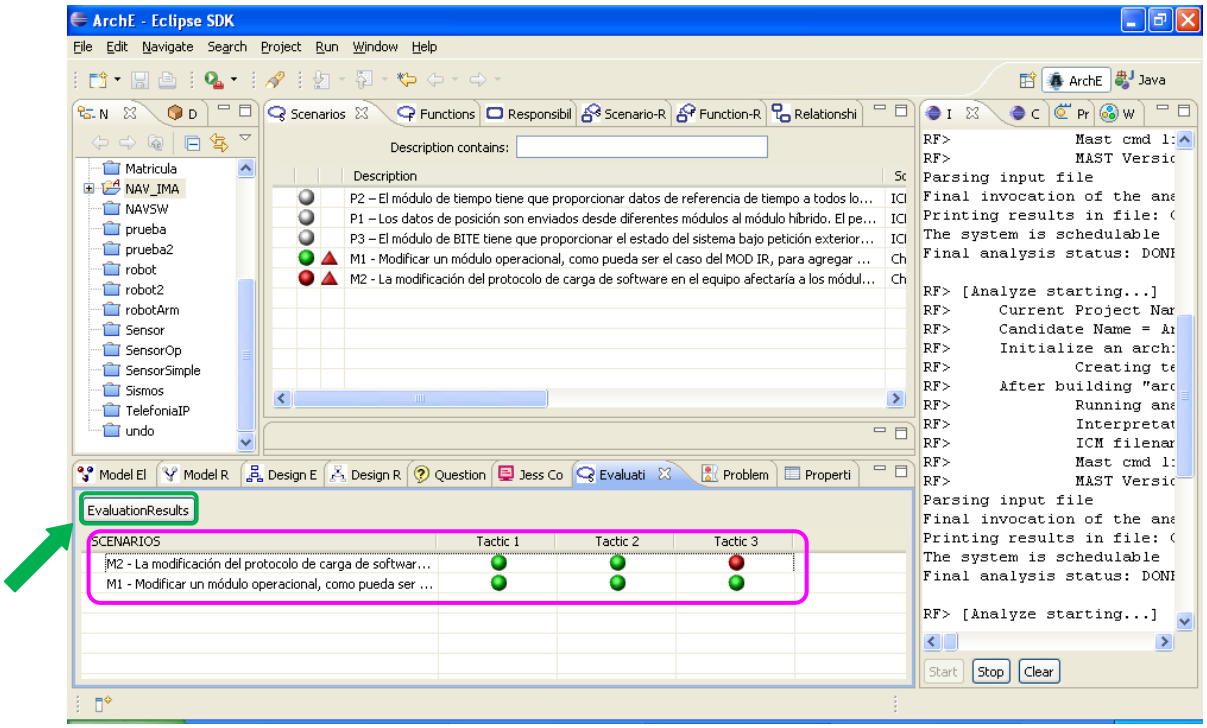


Figura 331. Vista Evaluations con las Posibles Tácticas a Aplicar y su Posible Resultado sobre el Escenario..

En general, el código de colores de los círculos tiene el siguiente significado:

- Un círculo verde ● significa que, aplicando la táctica correspondiente, el escenario se va a cumplir, debido a que la táctica mejorará las condiciones de dicho escenario.
- Un círculo ámbar ● significa que la aplicación de dicha táctica no afectará al cumplimiento del escenario (ni mejora ni degrada las condiciones del escenario).
- Un círculo rojo ● significa que, aplicando la táctica correspondiente, el escenario no se va a cumplir, debido a que la táctica degradará las condiciones de dicho escenario.

Pulsando el botón “EvaluationResults”, ArchE nos muestra otro formato de evaluación de la lista de escenarios, esta vez con triángulos cuyo código de colores tiene el siguiente significado:

- Un triángulo verde ▲ significa que la aplicación de la táctica produjo una mejora en el escenario particular, acercándolo a su umbral de cumplimiento.
- Un triángulo ámbar ▲ significa que la aplicación de la táctica no afectó al escenario particular.
- Un triángulo rojo ▲ significa que la aplicación de la táctica produjo una degradación en el escenario particular, alejándolo de su umbral de cumplimiento.

Si se pulsa en el botón EvaluationResults, se obtiene lo siguiente: (Figura 332)

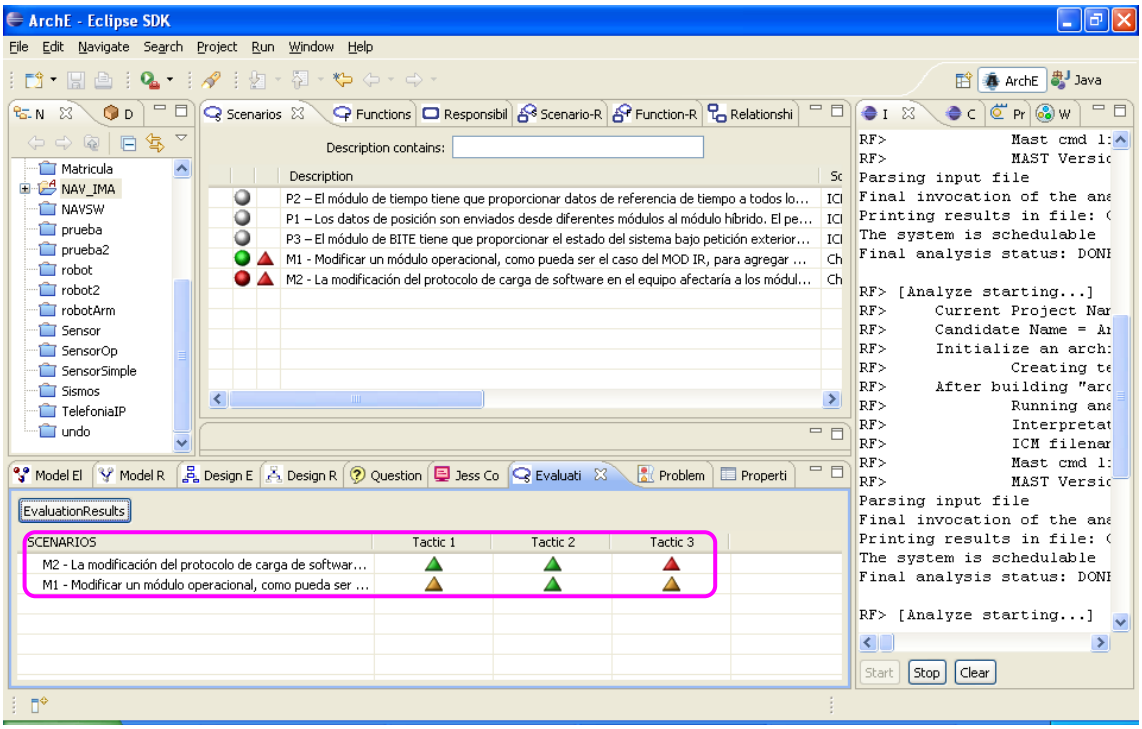


Figura 332. Vista Evaluations con las Posibles Tácticas a Aplicar y su Posible Mejora sobre el Escenario.

Se puede observar el resultado previsto de la aplicación de las tácticas sobre los escenarios, con el criterio descrito anteriormente.

En la ventana “Questions” se pueden ver las tácticas que propone ArchE. (Figura 333)

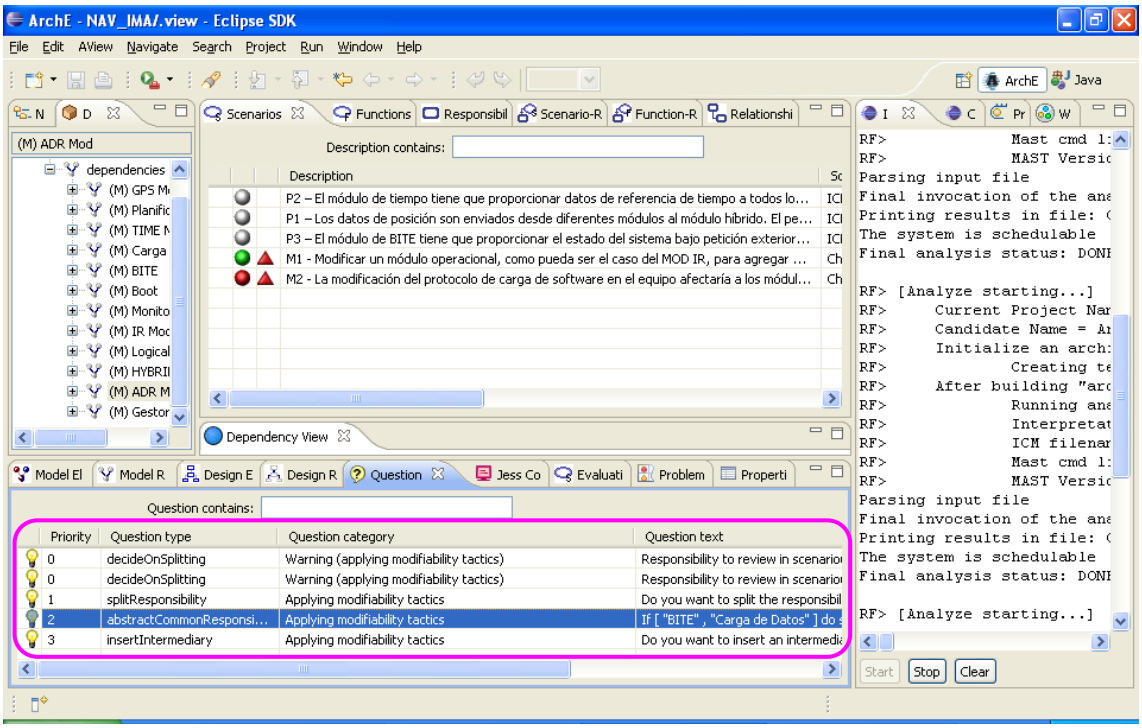


Figura 333. Ventana Questions con las Tácticas Sugeridas por ArchE.

Si se abre una de ellas, por ejemplo la 2: (Figura 334)

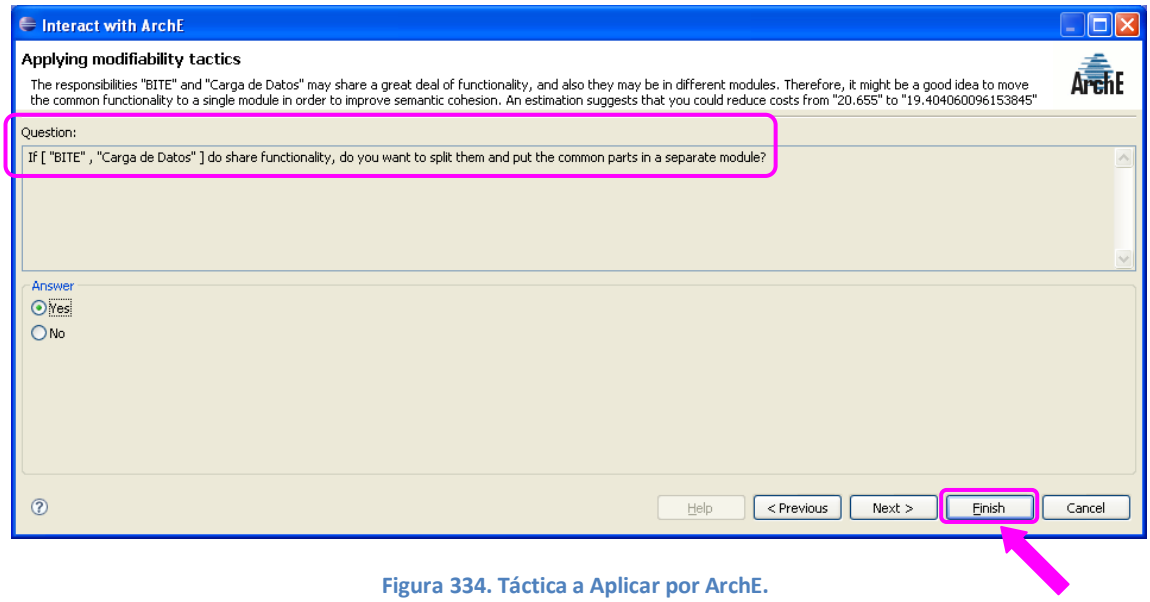


Figura 334. Táctica a Aplicar por ArchE.

Se pulsa sobre Finish y ArchE aplica cambios: (Figura 335)

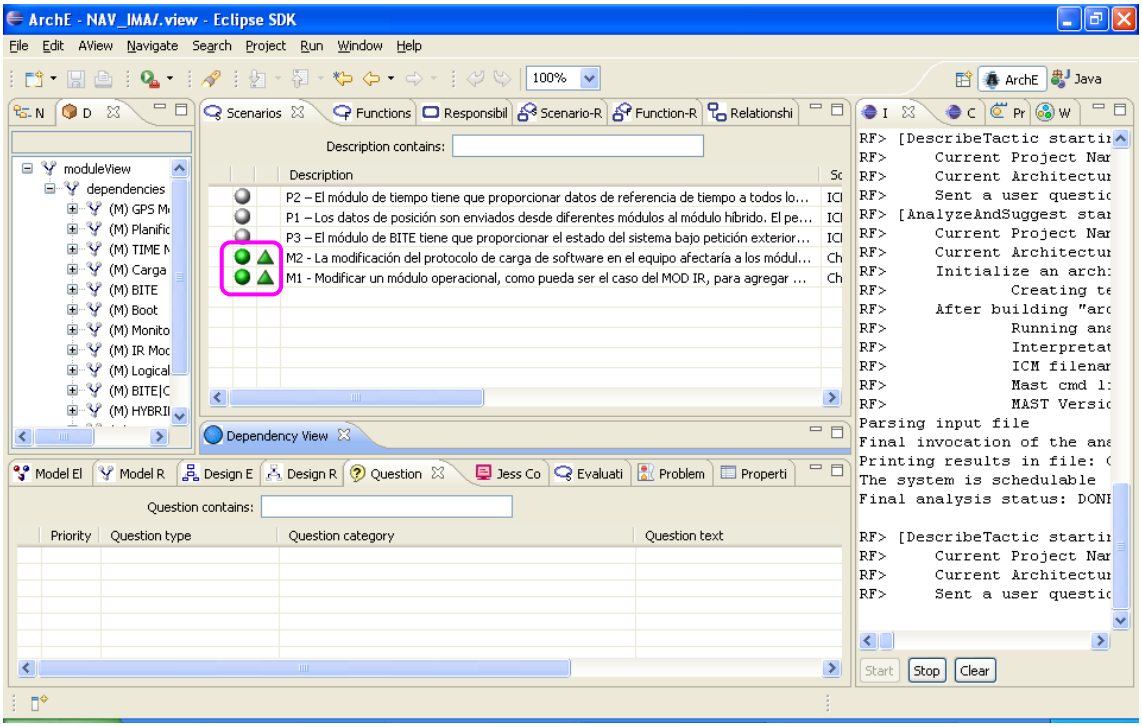


Figura 335. Resultados de Aplicar la Táctica de Arche sobre los Escenarios.

Se puede ver que ahora se cumplen los escenarios y por tanto han desaparecido las tácticas, puesto que ya no son necesarios.

La modificación introducida en el escenario de modificabilidad ha producido el siguiente cambio en la arquitectura: (Figura 336)

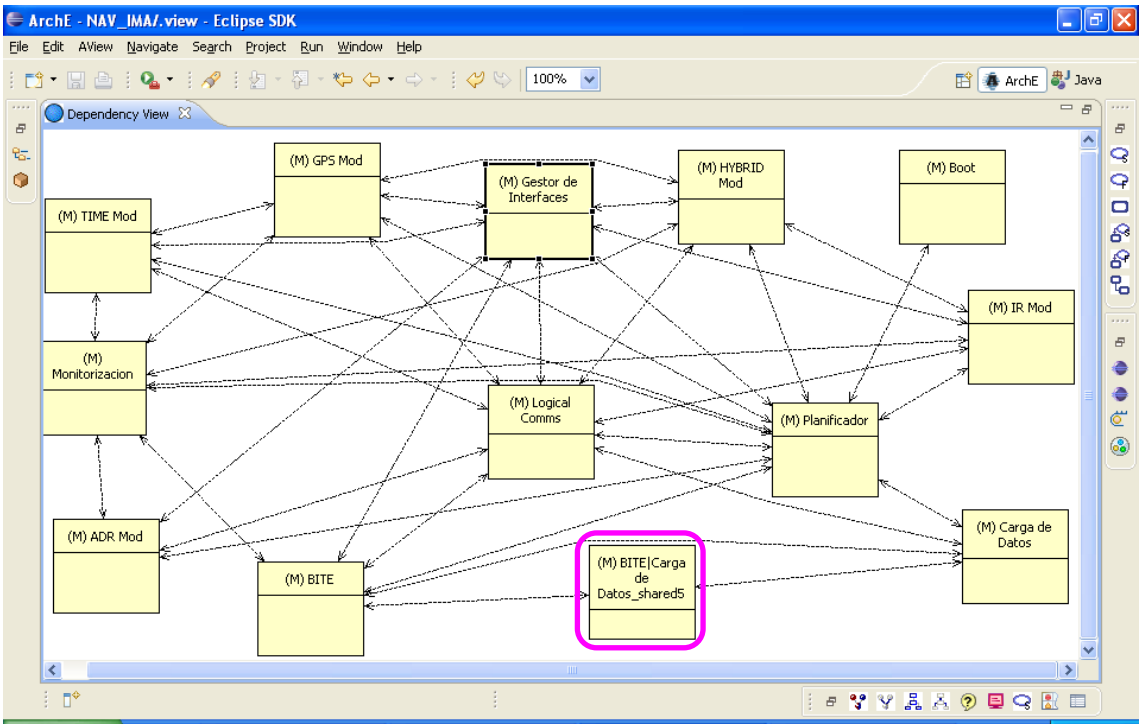


Figura 336. Nueva Vista de Árboles de Diseño con una nueva Responsabilidad creada por la Táctica.

Se observa como una nueva responsabilidad ha sido añadida, para eliminar desacoplamiento entre las antiguas responsabilidades.

En cuanto a rendimiento, se observa la ventana de MAST que indica que el sistema es planificable: (Figura 337)

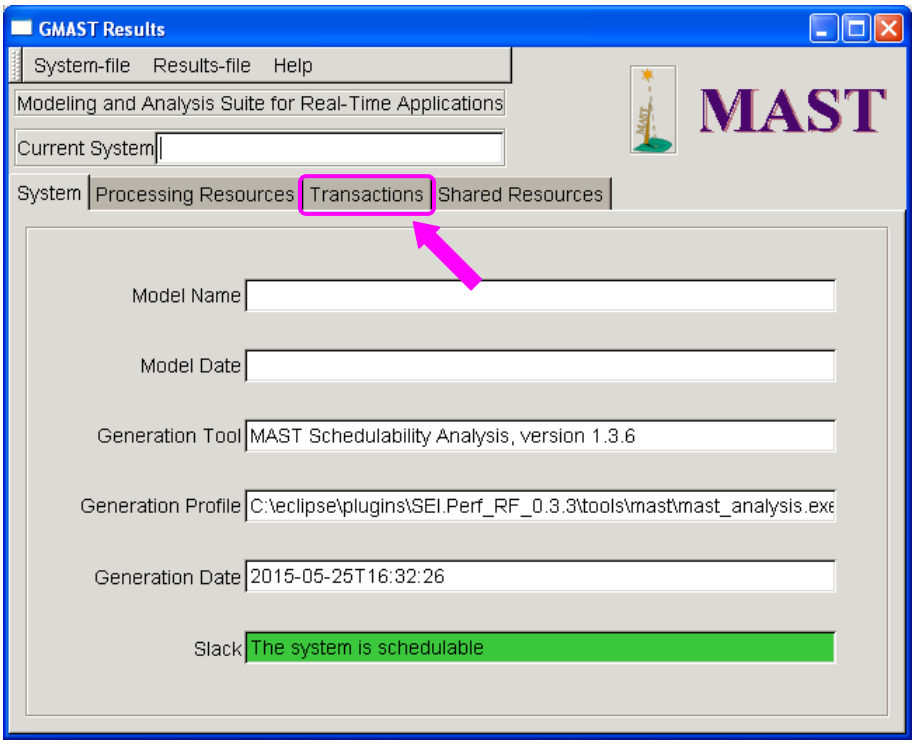


Figura 337. Ventana MAST con el Resultado de Evaluación del Rendimiento del Sistema

Pulsando sobre Transactions: (Figura 338)

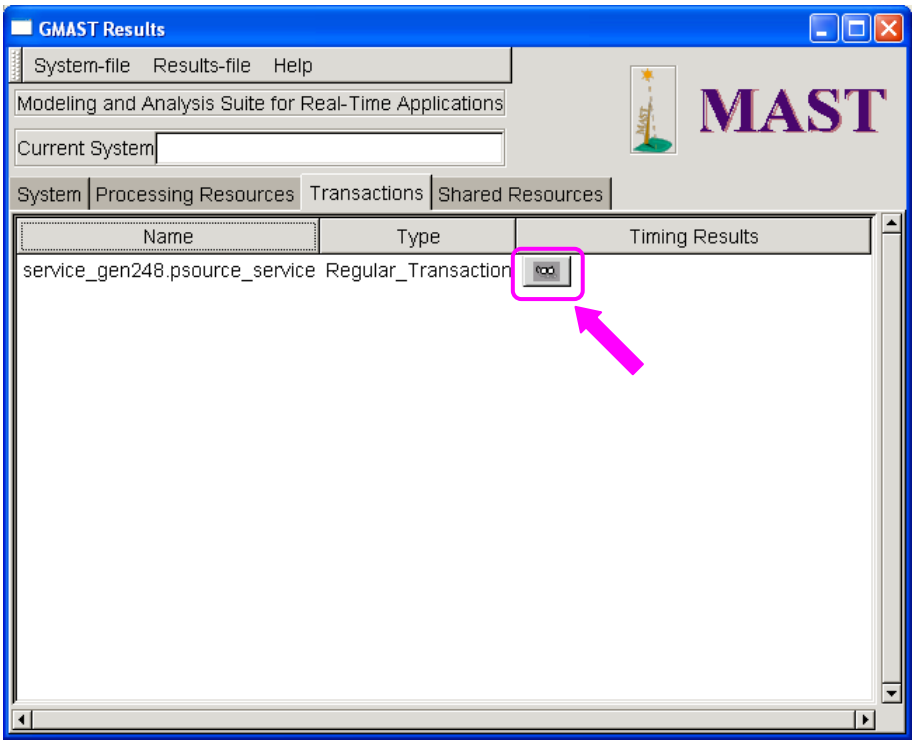


Figura 338. En la Pestaña Transactions se ofrece más Información.

Se pulsa sobre el icono de las gafas y a continuación pulsando sobre "View All" se observan todas las transacciones y que se cumplen las deadlines: (Figura 339)

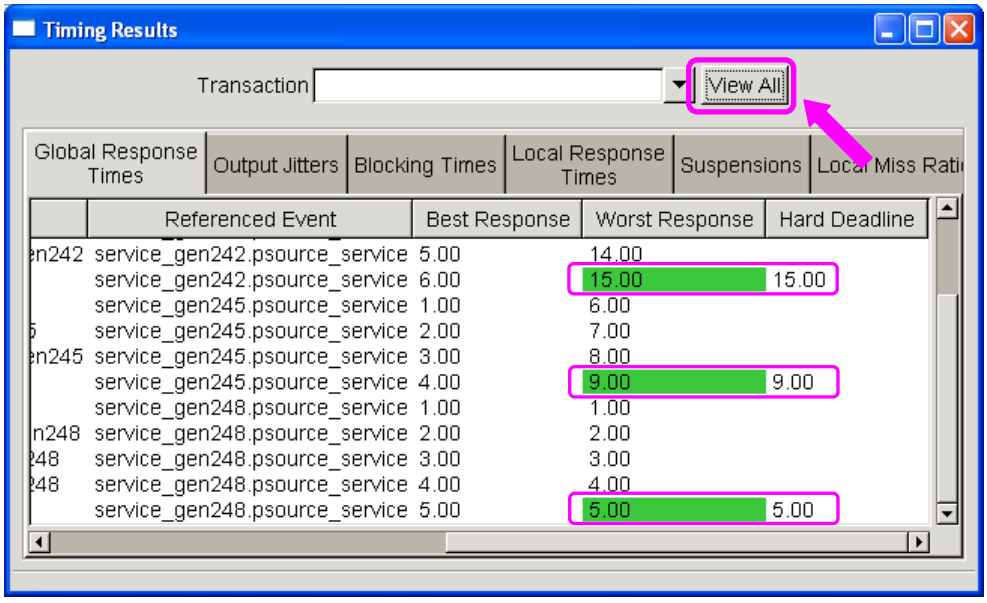


Figura 339. Vista MAST de los Escenarrios Cumpliendo sus Deadlines.

Si el sistema en su conjunto no fuera planificable, se obtendría el siguiente aviso: (Figura 340)

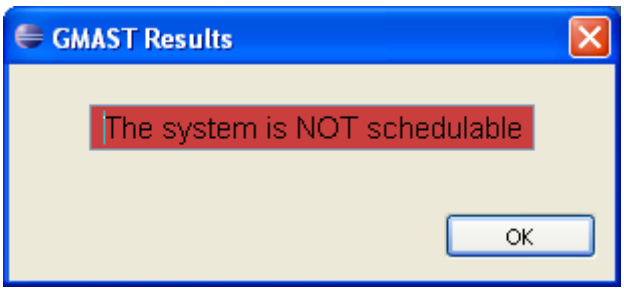


Figura 340. Aviso de Sistema No Planificable.

Si parte del sistema no fuera planificable en alguno de sus escenarios, se obtendría la siguiente ventana MAST: (Figura 341)

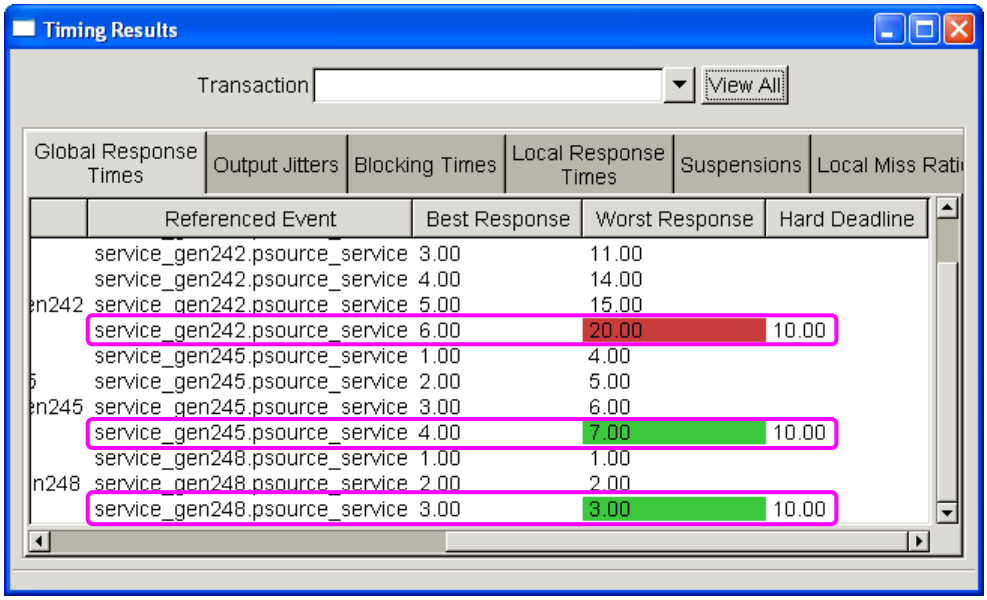


Figura 341. Ventana MAST con Escenarios que se Cumplen y otros que No.

En rojo el evento con latencia fuera de su deadline, y en verde los eventos con latencia cumpliendo su deadline.

Se pueden ver con detalle las ventanas de los marcos de razonamiento: (Figura 342 y Figura 343)

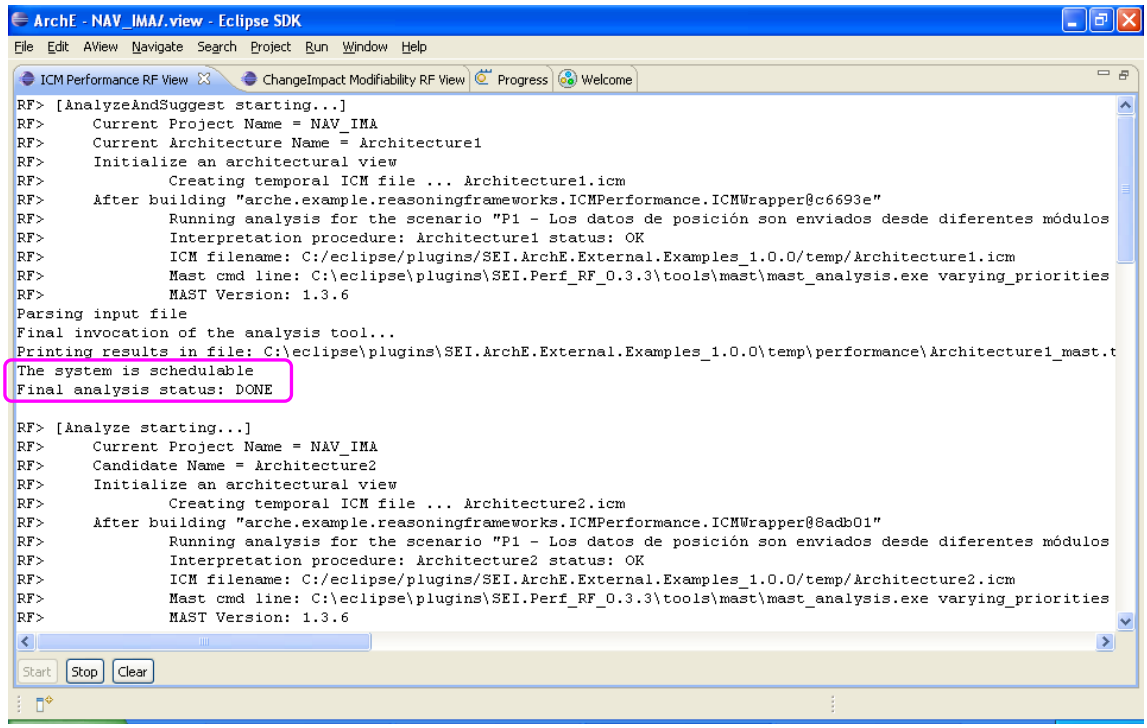


Figura 342. Ventana de Ejecución del Marco de Razonamiento ICM Performance.

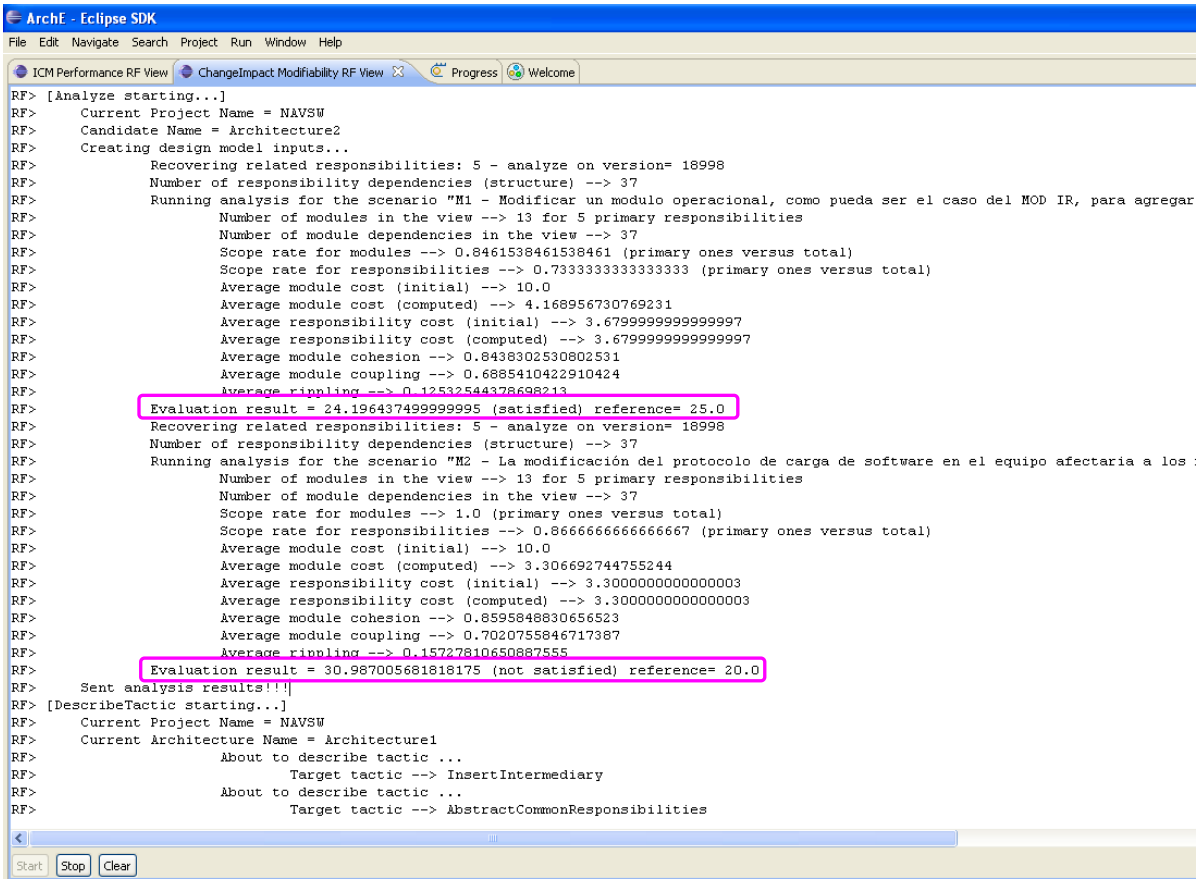


Figura 343. Ventana de Ejecución del Marco de Razonamiento ChangeImpact Modifiability.

14.2 Notas y Limitaciones de ArchE

Durante la realización de este trabajo, se han podido constatar las siguientes notas y limitaciones en el uso de la herramienta ArchE:

- Las responsabilidades tienen que tener un tiempo de ejecución entre 0 y 10 ms, lo cual obliga a escalar los posibles escenarios y las arquitecturas a esos órdenes de magnitud.
- Hasta que no hay creado al menos un escenario de modificabilidad y se aplique a al menos una responsabilidad, no se muestra el diagrama ULM de las responsabilidades.
- El diagrama UML muestra las relaciones de *dependency*, no las de *reaction*.
- Para que funcione el *ICM Performance*, se han de cumplir las siguientes condiciones:
  - Las responsabilidades deben tener un tiempo de ejecución
  - Los escenarios de *ICM Performance* han de estar definidos
  - Mapeo de escenario ICM Performance a responsabilidades establecido.
  - Relaciones tipo "*reaction*" entre responsabilidades asignadas a escenarios de ICM Performance
- Para que funcione el *ChangeImpact Modificability*:
  - Las responsabilidades deben de tener un coste del cambio
  - Los escenarios de *ChangeImpact Modificability* han de estar definidos
  - Mapeo de escenario *ChangeImpact Modificability* a responsabilidades
  - Relaciones tipo "*dependency*" entre responsabilidades asignadas a escenarios de *ChangeImpact Modificability*
- En *ICM Performance*, cuando hay varios escenarios de este tipo, y uno de ellos no se cumple, ArchE no realiza la comprobación para el resto.
- Cuando un escenario de performance no se cumple, se pone un círculo rojo. Pero cuando sí se cumple, no se pone verde, solo en gris: (Figura 344 y Figura 345)

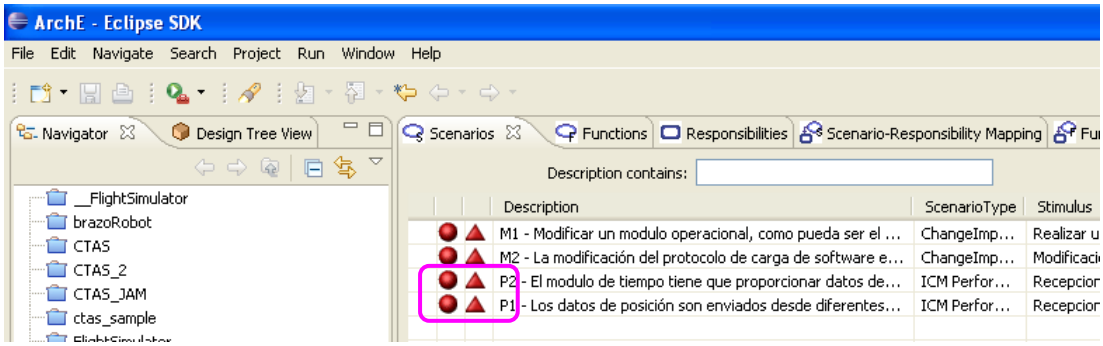


Figura 344. Resultado de ArchE – Caso de No Cumplimiento de los Escenarios de Rendimiento.

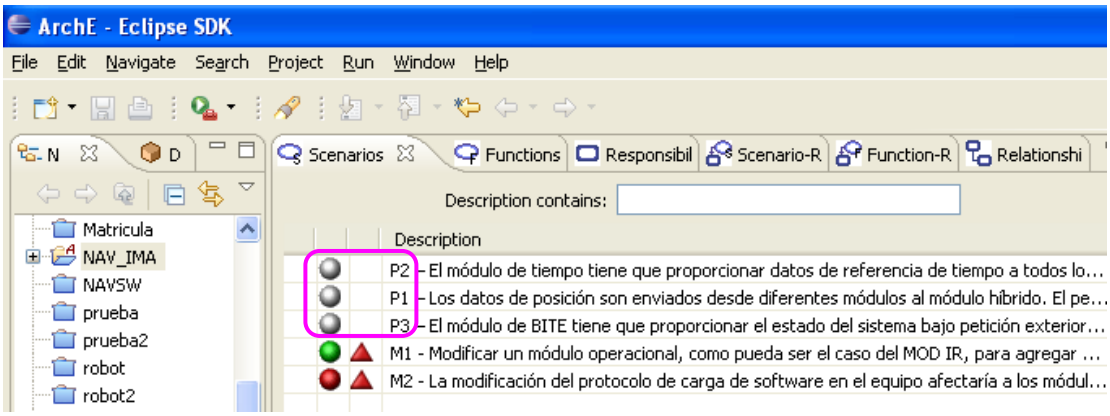


Figura 345. Resultado de ArchE – Caso de Cumplimiento de los Escenarios de Rendimiento.

- Para cumplir los escenarios de performance, hay que actuar sobre todas las responsabilidades afectadas por escenarios de *ICM Performance*, así como sobre las *deadlines* y los períodos de los escenarios de *ICM Performance*.
- No se permite establecer prioridades tanto a las responsabilidades como a las tareas (escenarios + responsabilidades) en *ICM Performance*
- La definición de responsabilidades es sensible a los caracteres no ingleses (por ejemplo, acentos). El programa no arranca MAST si se detecta algo así. Esto se ha visto sólo una vez durante una de las pruebas.
- Si los marcos de razonamiento no están cargados, no se pueden definir relaciones (de *dependency* o *reaction*) entre responsabilidades, ni elegir el tipo de escenario (*ICM Performance* o *ChangeImpact Modificability*) cuando se está definiendo.
- ArchE realiza las siguientes tareas en los atributos de calidad:
  - *ChangeImpact Modificability*: Análisis, tácticas
  - *ICM Performance*: Análisis; pero **ArchE no proporciona tácticas para *ICM Performance*.**
- El programa xmlBlaster ha de ser ejecutado (con la opción "r" al final de su ejecución) antes de ejecutar ArchE, puesto que xmlBlaster es el que gestiona las comunicaciones entre ArchE y los marcos de razonamiento, como se ha visto en el Anexo 1. Si no se ejecuta, los marcos de razonamiento no funcionarán apropiadamente.