UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

MÁSTER UNIVERSITARIO DE INVESTIGACIÓN EN INGENIERÍA DEL SOFTWARE Y SISTEMAS INFORMÁTICOS

ITINERARIO DE INGENIERÍA DE SOFTWARE
CÓDIGO: 31105128(2C)

TRABAJO DE INVESTIGACIÓN

EVALUACIÓN Y MEJORA DE ARQUITECTURAS SOFTWARE A TRAVÉS DE LA VALORACIÓN DE LOS ATRIBUTOS DE CALIDAD MEDIANTE SU INCORPORACIÓN Y EVALUACIÓN EN LA HERRAMIENTA ARCHE

ALUMNO: DAVID BELTRÁN RUIZ
DIRECTOR: DR. JOSÉ FÉLIX ESTÍVARIS LÓPEZ
CURSO ACADÉMICO: 2014-2015
CONVOCATORIA: SEPTIEMBRE 2015

MÁSTER UNIVERSITARIO DE INVESTIGACIÓN EN INGENIERÍA DEL SOFTWARE Y SISTEMAS INFORMÁTICOS

ITINERARIO DE INGENIERÍA DE SOFTWARE
CÓDIGO: 31105128(2C)

EVALUACIÓN Y MEJORA DE ARQUITECTURAS SOFTWARE A TRAVÉS DE LA VALORACIÓN DE LOS ATRIBUTOS DE CALIDAD MEDIANTE SU INCORPORACIÓN Y EVALUACIÓN EN LA HERRAMIENTA ARCHE

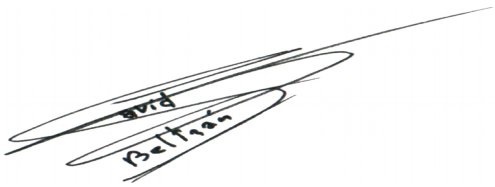
TIPO DE TRABAJO: A
ALUMNO: DAVID BELTRÁN RUIZ
DIRECTOR: DR. JOSÉ FÉLIX ESTÍVARIS LÓPEZ

CALIFICACIÓN

AUTORIZACIÓN DE PUBLICACIÓN Y DIFUSIÓN DEL TRABAJO DE FIN DE MÁSTER PARA FINES ACADÉMICOS

AUTORIZACIÓN

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



Firma: _____

David Beltrán Ruiz

RESUMEN

En el presente documento se presenta la propuesta de investigación “Evaluación y mejora de arquitecturas software a través de la valoración de los atributos de calidad mediante su incorporación y evolución en la herramienta ArchE” para el Máster de Investigación en Ingeniería Software de la UNED.

En dicha propuesta se obtiene una mejora notable en la arquitectura de la capa M-Plane de una BTS del ámbito de las redes de telecomunicaciones. Con el fin de identificar dicha mejora se utilizan los conocimientos adquiridos en el libro que se ha utilizado como punto de partida [Bass 13] a través de la aplicación de un enfoque orientado a los atributos de calidad y sus escenarios asociados.

Tras identificar los aspectos más débiles, se propone una arquitectura mejorada partiendo nuevamente del análisis de los atributos de calidad.

Por último, mediante el uso de la herramienta ArchE del SEI y atendiendo exclusivamente a su *plugin* de rendimiento, se analizan ambas arquitecturas arrojando un incremento del 30% en el rendimiento del sistema.

Palabras clave

ArchE, Arquitectura Software, Atributos de Calidad, Attribute Driven Design, Telecomunicaciones, BTS, Patrones Arquitectónicos, Tácticas.

ABSTRACT

The current document includes the research proposal “Evaluation and Enhancement of Software Architectures through Quality Attributes by their Evaluation in the ArchE Tool” for the Master in Research in Software Engineering and Computer Systems of the UNED (Universidad Nacional de Educación a Distancia).

In the above-mentioned proposal a significant improvement is obtained for the architecture of the M-Plane layer of a BTS inside the field of telecommunication networks. In order to identify that enhancement the knowledge acquired in the book [Bass 13] is used through the application of a quality attributes driven approach.

After the identification of the flaw points and starting from the quality attributes approach a new architecture is proposed.

Finally, by using the ArchE tool of the SEI and considering only its performance plugin, both architectures are evaluated and show an increase of 30% in the performance of the new architecture.

Keywords

ArchE, Software Architecture, Quality Attributes, Attribute Driven Design, Telecommunications, BTS, Architectural Patterns, Tactics.

AGRADECIMIENTOS

Dicen que es de bien nacido ser agradecido. Por ello no quiero dejar pasar la ocasión de agradecer a aquellas personas e instituciones que me han sido de especial ayuda.

A Dios, mi Señor y dador de vida. Podía pasarme la vida dándole gracias y nunca le haría justicia.

A mi mujer, Aleksandra, por su apoyo y por ser el motivo de tanta felicidad.

A nuestro hijo, José María, que aunque está por llegar ya nos ha dado tantas alegrías.

A mis padres, que me han dado la vida y han creado una tierra fértil en la que Dios se hizo fuerte.

Al director de la tesis, Don José Félix Estívaris López, por su apoyo y guía durante la elaboración de este trabajo.

A Cecilia y Juan Carlos, por estar ahí, siempre que los hemos necesitado.

A la UNED, por haberme dado la oportunidad de estudiar, a pesar de mi carga laboral o de vivir tan lejos de España.

A todos, muchísimas gracias.

Índice

LISTADO DE FIGURAS

LISTADO DE TABLAS

INTRODUCCIÓN

OBJETIVOS

El presente trabajo de fin de Máster tiene el propósito de emplear los conocimientos adquiridos durante el Máster de Investigación en Ingeniería Software y en particular los obtenidos en la asignatura de Arquitectura de Sistemas Software para obtener una mejora notable en un problema surgido en el ámbito profesional del autor mediante el uso de la herramienta ArchE.

Durante la búsqueda de la mejora se utilizarán los conocimientos adquiridos en el libro que se ha utilizado como punto de partida [Bass 13] a través de la aplicación de un enfoque orientado a los atributos de calidad y sus escenarios asociados. Para ello se presentará un problema del cual se obtendrá su arquitectura actual y se evaluará esta con respecto a sus atributos de calidad. Posteriormente se propondrá una mejora en la arquitectura que se evaluará en términos de rendimiento.

Dicha evaluación se realizará con la herramienta ArchE atendiendo exclusivamente a su *plugin* de rendimiento.

ESTRUCTURA DEL TRABAJO

El presente trabajo se divide en los siguientes capítulos que se recogen a continuación.

Capítulo 1: La arquitectura software y su importancia. Este capítulo pretende acercar al lector al concepto de arquitectura software, tratando de subrayar su importancia para el éxito de los sistemas software modernos. Además, se realizan un acercamiento a diversas técnicas empleadas en el ámbito de esta disciplina y en particular a aquellas basadas en los atributos de calidad.

Capítulo 2: Atributos de calidad, escenarios y tácticas. Este capítulo avanza en los conceptos relacionados con los atributos de calidad y su especificación en forma de escenarios, así como de las tácticas empleadas para mejorarlos. Tras un primer acercamiento, se recorre cada uno de los principales atributos de calidad identificados en [Bass 13] y que se identifican las posibles tácticas que pueden emplearse para mejorar el diseño arquitectónico con respecto a estos atributos.

Capítulo 3: ArchE: introducción y manual de instalación. Aquí se recoge una visión general de la herramienta basada en un artículo publicado por los autores de las herramientas [Bass 07], y posteriormente se incluye una guía de instalación para el usuario. Además se añade un apartado con las limitaciones encontradas en el *plugin* de rendimiento, muchas de las cuales han sido corregidas a fin de poder desarrollar el presente trabajo.

Capítulo 4: Presentación del problema. Este capítulo introduce un problema del ámbito de las telecomunicaciones cercano al autor mediante la presentación de su arquitectura, sus problemas y ventajas, todo ello a partir de un enfoque desde los atributos de calidad asociados. Por último, a manera de ejemplo se introduce un problema real relacionado con el diseño de la arquitectura.

Capítulo 5: Propuesta de diseño. Nuevamente, a partir de los atributos de calidad, a lo largo de este capítulo se realiza una propuesta de mejora que consiste en una nueva arquitectura centrada en

subsanar los puntos débiles de la arquitectura original. Al final se vuelve al problema de diseño presentado en el capítulo anterior y se demuestra que realmente dicho problema no pertenece al ámbito de la arquitectura sino del diseño, a pesar de que la arquitectura pueda ayudar a ponerlo de relieve.

Capítulo 6: Análisis y evaluación en ArchE. Este capítulo realiza un análisis detallado partiendo de la descripción del sistema y sus casos de uso para extraer la información necesaria a fin de evaluar ambas arquitecturas en ArchE.

Capítulo 7: Conclusiones. Finalmente se extraen las conclusiones del trabajo y se realizan varias propuestas de cara a futuros trabajos.

Anexo I: Mejoras del marco de rendimiento de ArchE. Este anexo recoge algunos de los problemas encontrados en la herramienta durante el desarrollo de este trabajo, además de su causa y por último su solución mediante la aportación del código fuente corregido.

1. LA ARQUITECTURA SOFTWARE Y SU IMPORTANCIA

A medida que los sistemas informáticos han ido desarrollándose y aumentando en complejidad, el concepto de arquitectura software ha ido emergiendo y consolidándose a lo largo de la literatura relacionada con el ámbito profesional del software.

En la actualidad no puede concebirse ningún sistema complejo sin la aplicación del diseño arquitectónico [Garlan 95]. De hecho, la elección de la arquitectura correcta puede predestinar el éxito o fracaso de un proyecto software.

En las grandes empresas existen puestos de trabajo especializados cuya única labor es la de definir la arquitectura del sistema software y estudiar cómo esta satisface los requisitos actuales y cómo es capaz de adaptarse para cumplir los requisitos futuros.

Además, la arquitectura del software no se limita sólo a las fases tempranas del desarrollo de un proyecto, como la toma de requisitos o el diseño de alto nivel, ya que es capaz de marcar otras etapas posteriores como el diseño detallado, la implementación y la fase de pruebas. Como ejemplo, un ingeniero de software no podrá tomar las decisiones adecuadas acerca de la implementación sin un conocimiento profundo de las consecuencias derivadas de la arquitectura en el ámbito que está implementando. De ello se desprende la necesidad de que las arquitecturas no sólo sean correctas y cumplan los requisitos sino que además sean fácilmente accesibles. Por otro lado, una buena arquitectura deberá tener en cuenta la posibilidad no sólo de facilitar información básica, como por ejemplo si la funcionalidad es correcta o no, sino de ofrecer información adicional cuando esta se requiera con el fin de poder subsanar los posibles errores.

Tras esta breve introducción se hace indispensable la definición del concepto de arquitectura software.

1.1 DEFINICIÓN DE ARQUITECTURA SOFTWARE

En [Bass 13] se define la arquitectura software como:

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

De esta definición se desprenden varias consecuencias, entre las que destacan dos:

- **La arquitectura es un conjunto de estructuras software**. Una estructura es a su vez un conjunto de elementos unidos por una relación. Los sistemas software están compuestos por estructuras y ninguno de ellos puede afirmar ser la arquitectura en su totalidad. En general pueden identificarse tres tipos de estructuras:
  - Las particiones del sistema o módulos del mismo. Un conjunto de módulos pueden formar módulos de más alto nivel a veces denominados capas.
  - La naturaleza dinámica del sistema, definida como la forma en que los elementos interactúan entre sí en tiempo de ejecución para resolver la funcionalidad del sistema.
  - La forma en que los elementos del sistema se asocian con los niveles de organización, desarrollo, instalación y ejecución del sistema.

- **La arquitectura es una abstracción del sistema**. La arquitectura del sistema expone ciertos detalles de un sistema e intencionadamente omite otros, con el fin de ofrecer exclusivamente la información que permita razonar sobre la naturaleza del mismo. De esta forma se evita un nivel de detalle innecesario que no sólo no ayudaría al razonamiento sino que sería perjudicial al añadir confusión.

Una vez definida a continuación se explica el porqué de la importancia de la arquitectura.

1.2 IMPORTANCIA DE LA ARQUITECTURA SOFTWARE

En [Garlan 94] se proporcionan cuatro razones fundamentales por las que no puede concebirse la ingeniería del software sin el diseño arquitectónico:

- La necesidad de reconocer paradigmas comunes de forma que las relaciones de alto nivel entre sistemas puedan ser fácilmente comprensibles y por lo tanto, nuevos sistemas puedan construirse como variaciones de otros existentes.
- La capacidad de elegir la arquitectura correcta, lo cual resulta crucial para el éxito del diseño de un sistema.
- La posibilidad de ofrecer a los ingenieros un conocimiento detallado de las arquitecturas software de forma que puedan tomarse las decisiones correctas en cuanto a las alternativas de diseño existentes.
- La necesidad de representar la arquitectura de un sistema de forma que puedan analizarse y describirse las propiedades de alto de nivel de un sistema complejo.

[Garlan 95] añade un quinto argumento, con el objetivo de ofrecer una primera prueba a los responsables de la toma de decisiones de que un proyecto es viable y que será capaz de satisfacer los requisitos establecidos.

A medida que aumenta la complejidad de un sistema se vuelve más difícil satisfacer los problemas expuestos sin el uso del diseño arquitectónico.

1.3 METODOLOGÍAS DE DISEÑO ARQUITECTÓNICO

Existen varias metodologías de diseño arquitectónico. Se incluyen aquí algunas de ellas, sobre todo aquellas relacionadas con los atributos de calidad por ser de particular relevancia para este trabajo como se verá más adelante:

- **ABD**. Tal y como se refleja en [Bachman 00], el diseño basado en la arquitectura (*Architecture Based Design* o sus siglas ABD en inglés) se basa en tres pilares:
  - la descomposición funcional del sistema a través del uso de técnicas conocidas para la búsqueda del desacoplamiento y la cohesión;
  - la realización de los atributos de calidad[1] y de los requisitos de negocio mediante la elección de los estilos arquitectónicos correctos y
  - el uso de plantillas software[2], que definen qué responsabilidades y características deben

1 Este concepto se explicará ampliamente, más adelante, en el capítulo 2.

2 Este concepto recuerda el de patrón arquitectónico. En la 3ª edición de *Software Architecture in Practice* ya no

poseer los elementos del sistema por el hecho de pertenecer al sistema. Puede consultarse más acerca de este concepto en [Chastek 96].

- **ADD**. Tal y como se expone en [Bass 01], el diseño dirigido por atributos (*Attribute Driven Design* o sus siglas ADD en inglés) es una aproximación para definir la arquitectura software basando el proceso de diseño en hacer que el software cumpla los atributos de calidad identificados en los requisitos[3]. Este es un proceso de descomposición recursivo en el que en cada fase de la descomposición se eligen las primitivas arquitectónicas[4] con el fin de satisfacer un conjunto de escenarios de calidad. Seguidamente se instancian los componentes y los conectores definidos en dichas primitivas con el fin de asignar a dichas instancias la funcionalidad del sistema.
  A grandes rasgos, ADD toma como entrada los requisitos funcionales y de calidad y produce la arquitectura conceptual. Por lo tanto, ADD no devuelve un diseño arquitectónico completo, sino un primer esqueleto a partir del cual podrá desarrollarse aquel.

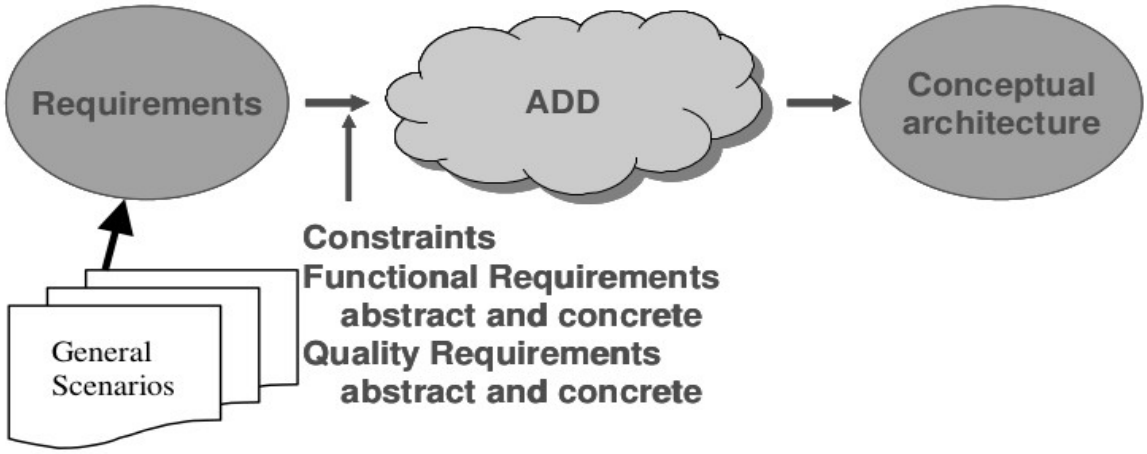


Ilustración 1: ADD en el ciclo de vida (Fuente: SEI [Bass 01]).

- **ATAM**. El método de análisis de arquitectura basado en *trade-off*[5] (*Architecture Tradeoff Analysis Method* o su siglas ATAM en inglés), es un método para la evaluación de arquitecturas basado en escenarios donde las personas decisivas y los dueños del producto articulan una lista precisa de atributos de calidad en forma de escenarios, además de intentar tomar las decisiones arquitectónicas que satisfagan los escenarios con mayor prioridad. Las decisiones pueden entenderse como la identificación de riesgos o la falta de estos a fin de señalar los posibles puntos problemáticos de la arquitectura. Puede obtenerse más información en [4] o en [Clements 01].

aparece definido, mientras que el de patrones arquitectónicos tiene dedicado un capítulo completo, lo que podría confirmar esta teoría. Sin embargo, examinando [Chastek 96] el concepto recuerda más bien a la generación de código a través de los datos del dominio y plantillas de generación tal y como se hace en la aproximación MDA [8].

3 Este método de diseño contrasta con [Bosch 00], donde también se da especial importancia a los requisitos de calidad pero aún desempeñan un papel secundario con respecto a los funcionales.

4 No debe confundirse este concepto con el de patrones arquitectónicos. Las primitivas arquitectónicas deben entenderse como las tácticas para satisfacer los atributos de calidad, y presentan un nivel de abstracción más bajo que los patrones arquitectónicos, los cuales pueden utilizar varias tácticas, tal y como se expone en [Bass 13].

5 Se usa el término en inglés, al no encontrarse una traducción adecuada. Se refiere a la identificación de las relaciones entre los atributos de calidad y de cómo el beneficio en uno puede perjudicar a otro.

Ilustración 2: Flujo conceptual del método ATAM (Fuente: SEI [4]).

Además de los métodos anteriores, según las distintas etapas del ciclo del software y los atributos de calidad a los que otorguen mayor relevancia, se mencionan aquí otros, sin el propósito de presentar una lista exhaustiva, como:

- *Quality Attribute Workshops* (**QAW**) [Barbacci 03], creado para la identificación de los atributos de calidad antes de pasar a la creación de la arquitectura;
- *Cost Benefit Analysis Method* (**CBAM**) [3], centrado en el análisis de la implicación de las elecciones arquitectónicas en los costes, beneficios y planificabilidad de los sistemas informáticos;
- *Active Review for Intermediate Design* (**ARID**) [Clements 00], utilizado para la revisión de diseño de alto nivel;
- *Family-Architecture Analysis Method* (**FAAM**) [Dolan 02], centrado en la interoperabilidad y extensibilidad de las familias de sistemas de información;
- *Architecture Level Modifiability* (**ALMA**) [Bengtsson 04], centrado en la modificabilidad de los sistemas de información de negocio.

Una visión general de los métodos basados en escenarios puede obtenerse en [Ionita 02].

1.4 PATRONES ARQUITECTÓNICOS

Como se desprende de la primera de las necesidades identificadas anteriormente en [Garlan 94][6], con el avance de la industria del software se han ido identificando problemas y soluciones comunes que pueden aplicarse sin tener en cuenta el dominio de la aplicación a resolver. Dichas soluciones han dado lugar a los patrones arquitectónicos, cuya finalidad es la de resolver un problema recurrente con una solución de la cual se derivan una serie de ventajas e inconvenientes. Corresponderá al arquitecto evaluar dichas ventajas e inconvenientes a fin de seleccionar el patrón o patrones adecuados para su arquitectura.

Entre otros, algunos patrones de gran popularidad son:

6 La de identificar patrones comunes en los sistemas software.

- Tuberías y filtros.
- Cliente – servidor.
- Abstracciones de datos y organización orientada a objetos.
- Sistemas basados en eventos o invocación implícita.
- Sistemas basados en capas.
- Repositorios.

Dichos patrones y otros más pueden consultarse en detalle en [Garlan 94] o en autores de otras posiciones, dentro de la amplia literatura sobre patrones existente, tales como [Gamma 94].

La existencia de patrones no sólo facilita la resolución de problemas de diseño, sino que permite facilitar el razonamiento del sistema ya que, al aumentar la popularidad de los mismos, aumenta también la familiaridad de los ingenieros con los mismos.

Por último, por ser de especial relevancia para el problema arquitectónico propuesto en este trabajo, se incluyen aquí los detalles del patrón *blackboard*.

1.4.1 EL PATRÓN *BLACKBOARD*

Tal y como se detalla en [Garlan 94] el patrón *blackboard* es una derivación de los patrones de tipo repositorio. Estos últimos se caracterizan por tener dos tipos de componentes:

- una estructura de datos central que representa el estado del sistema, y
- una colección de componentes interdependientes que operan sobre el almacén central.

Atendiendo a los mecanismos de control del sistema, los repositorios se dividen en dos grandes familias, dependiendo de si la transacción determina explícitamente qué componentes deben ser notificados o si el estado del sistema invoca implícitamente los componentes que deben ser notificados. En el primer caso los repositorios pueden considerarse como una base de datos tradicional, mientras que en el segundo dan lugar a los patrones de tipo *blackboard* el cual se detalla a continuación.

Una arquitectura de tipo *blackboard* se divide principalmente en las tres partes siguientes:

1. **Los orígenes de conocimiento**. Estos son segmentos independientes que representan parcelas separadas del conocimiento del sistema. La interacción entre orígenes tiene lugar exclusivamente a través de la pizarra[7].
2. **La estructura central de datos o pizarra**. Contiene el estado de la solución cuya representación dependerá del sistema en cuestión. Los orígenes de conocimiento interactúan con la pizarra de forma colaborativa hasta llegar a la solución del problema.
3. **Mecanismo de control**. Se deriva íntegramente del estado de la pizarra, de forma que los orígenes de conocimiento responden oportunamente cuando el estado de los datos se lo permite.

El diagrama siguiente ilustra el diseño del patrón. Aunque el control del sistema no quede reflejado en la ilustración, este consiste en la activación de los orígenes de conocimiento a medida que se

7 Se utiliza aquí el término traducido al español con el objetivo de explicar el funcionamiento del patrón, aunque posteriormente se volverá al término en inglés para referirse al nombre del patrón.

modifica el estado de la pizarra. El estado, por tanto, deberá evolucionar de forma que siempre que la solución esté incompleta, uno o más orígenes deberán estar bien en ejecución o bien pendientes de ser activados por el estado actual. De lo contrario, la ejecución se interrumpiría debido a que ninguno de los componentes se encuentra en estado de contribuir a la solución y esta no se ha alcanzado aún, lo que se conoce como *deadlock*.

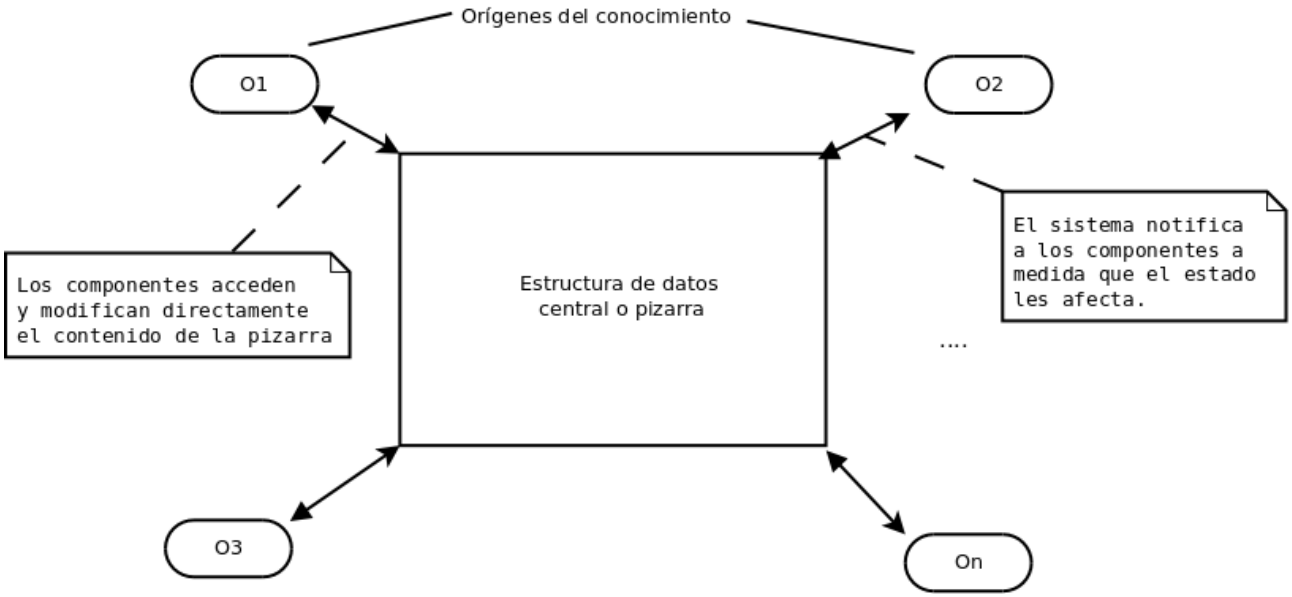


Ilustración 3: Diseño del patrón blackboard (Fuente: elaboración propia a partir de [Garlan 94]).

2. ATRIBUTOS DE CALIDAD, ESCENARIOS Y TÁCTICAS

2.1 LOS ATRIBUTOS DE CALIDAD

Según la definición dada en [Bass 13], los atributos de calidad son propiedades medibles de un sistema que se usan para indicar hasta qué nivel el sistema satisface las necesidades del cliente[8]. Se puede entender un atributo de calidad como una medida de la "bondad"[9] de un producto a lo largo de las distintas dimensiones de interés para un cliente.

En lo que se refiere a la relación de los atributos de calidad, un requisito siempre puede dividirse entre tres categorías, una de la cuales corresponde a los propios atributos de calidad:

- **Requisitos funcionales**. Establecen lo que el sistema debe hacer y cómo reaccionará ante un estímulo.
- **Atributos de calidad**. Determinan en qué medida se cumplen los requisitos funcionales, por ejemplo, con qué rendimiento, tasa de disponibilidad, fallo, etc.
- **Restricciones**. Representan decisiones de diseño en las que no se admite flexibilidad y que han sido tomadas por el cliente. Por ejemplo el uso de un lenguaje de programación o una plataforma software particular.

[Bass 13] identifica siete atributos de calidad, que son los más importantes debido a la frecuencia con la que aparecen identificados en las arquitecturas software modernas. No obstante, los propios autores reconocen que existen muchos otros cuyo número también dependerá de la taxonomía de clasificación empleada. Los siete atributos mencionados son:

- **disponibilidad**,
- **interoperabilidad**,
- **modificabilidad**,
- **rendimiento**,
- **seguridad**,
- **testeabilidad** y
- **usabilidad**.

Más adelante se describirá cada uno de ellos por separado junto con sus tácticas asociadas.

2.2 LOS ESCENARIOS

Un atributo de calidad en sí es ambiguo e incapaz de ser medido o probado. La forma de especificar un atributo de calidad consiste en capturarlo de manera formal en lo que se denomina un escenario, el cual se compone de las siguientes partes:

1. **Origen del estímulo**. Es la entidad (un humano, ordenador, componente, etc.) que genera el estímulo[10].

8 La versión original usa el término *stakeholder*, que no tiene porque ser un cliente propiamente dicho. Podría haberse optado por otra traducción, pero se entiende que en general se refiere al beneficiario y promotor del sistema.

9 Se refiere a cómo de bien el sistema hace lo que debe o, dicho de otra forma, cómo de "bueno" es en lo que debe hacer.

10 Como opinión personal es de utilidad entender el origen del estímulo como el papel que desempeña un actor en un

2. **Estímulo**. Es la condición que requiere una respuesta del sistema cuando este la recibe.
3. **Entorno**. Son las condiciones en las que se halla el sistema cuando ocurre un determinado estímulo. Por ejemplo, el estímulo puede referirse a un nivel de carga particular del sistema dejando de ser válido en otras.
4. **Artefacto**. Es el elemento del sistema que resulta estimulado. Puede ser el sistema en su totalidad, una parte, etc.
5. **Respuesta**. Es la actividad generada como resultado de la llegada del estímulo.
6. **Medida de la respuesta**. Es la magnitud en la que se puede medir la respuesta de forma que el escenario pueda ser comprobado.

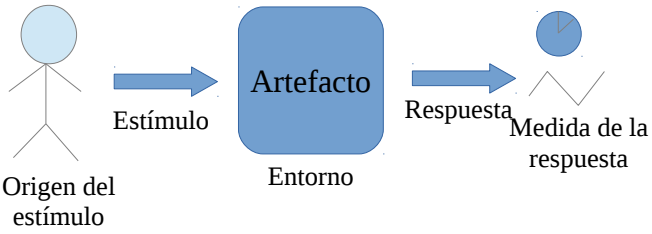


Ilustración 4: Las partes del escenario de un atributo de calidad (Fuente: elaboración propia).

2.3 LAS TÁCTICAS

Nuevamente en [Bass 13] se define una táctica como la decisión de diseño que influencia la consecución de la respuesta a un atributo de calidad.

Cada táctica se centra en un único atributo de calidad y no existe, por tanto, necesidad de equilibrar el coste-beneficio con otros atributos. Esto constituye la principal diferencia entre un patrón arquitectónico y una táctica, ya que el patrón suele tener una serie de ventajas e inconvenientes[11].

Cada atributo de calidad conlleva una serie de tácticas cuyo objetivo es ayudar al sistema a cumplir dicho atributo. En los apartados siguientes se describen los siete atributos más importantes junto con sus tácticas asociadas.

2.4 DISPONIBILIDAD

La disponibilidad se refiere a la propiedad del software de estar en el lugar esperado y listo para realizar la tarea para la que está diseñado en el momento en que sea requerido.
Existen múltiples motivos por los que un sistema puede no estar disponible, como pueden ser las faltas[12], tanto de software como de hardware, la falta de capacidad debido a problemas de

caso de uso.

11 También debido a que un patrón es más complejo que una táctica al incluir un conjunto de ellas, como ya se ha mencionado previamente.

12 Se define una falta como la causa de un fallo, y este último como la desviación de un sistema de su especificación

rendimiento, situaciones de sobrecarga por actuaciones malintencionadas, etc.

Se puede calcular la disponibilidad del sistema como la probabilidad de que este proveerá los servicios dentro los límites establecidos durante un intervalo de tiempo específico.

Las tácticas para aumentar la disponibilidad de un sistema se dividen en tres grandes familias:

- **Detección de faltas**. Antes de que un sistema pueda tomar ninguna medida frente a una falta, a fin de encubrirla, esta debe ser previamente detectada. Las tácticas dentro de esta categoría incluyen el uso de *ping*, la monitorización, los mensajes de salud[13], las marcas de tiempo, los chequeos de integridad, la monitorización de condiciones, la votación mediante el uso de redundancia, la detección de excepciones y los auto tests.
- **Recuperación de faltas**. Una vez detectada la falta será necesario recuperarse de ella. Se divide a su vez en dos:
  - Preparación y reparación. Consiste en incluir elementos de redundancia o en reintentar la operación. Dentro de esta rama pueden incluirse tácticas como, la redundancia activa, la redundancia pasiva, el repuesto, la gestión de excepciones, la vuelta atrás, la actualización del software, el reintento, la supresión del comportamiento erróneo, la degradación y la reconfiguración.
  - Reintrodución. Consiste en la sustitución o recuperación del componente erróneo. Incluye tácticas como el modo en sombra[14], la sincronización de estados, el reinicio escalado, y el *non-stop fowarding* (NFS)[15].
- **Prevención de faltas**. Consiste, como su propio nombre indica, en diseñar el sistema de forma que dichas faltas no lleguen a ocurrir. Esta familia de tácticas incluye la puesta en fuera de servicio, las transacciones, los modelos predictivos, la prevención de excepciones y el aumento de competencias.

2.5 INTEROPERABILIDAD

La interoperabilidad es el grado en que uno o más sistemas pueden de forma útil y significativa intercambiar información a través de sus interfaces en un contexto particular.

La interoperabilidad se ve afectada por la forma de codificar el conocimiento de un sistema sobre el otro, pudiendo este haber sido codificado en la lógica del programa, o quedar fuera de ella mediante mecanismos de detección y acoplamiento en tiempo real.

Las tácticas de este atributo se dividen en dos categorías:

- **Localización**. Se refiere a la capacidad del sistema de localizar los sistemas con los que se comunica. Incluye la táctica de servicios de descubrimiento.
- **Gestión de interfaces**. Se refiere a la forma en que se utilizan las interfaces. Incluye dos tácticas: la orquestación y las interfaces a medida.

siempre que dicha desviación sea visible.

13 Habitualmente conocidos como *heartbeat* en la literatura técnica.

14 Se traduce *shadow* literalmente por “sombra”. En la literatura técnica se suele entender *shadowing* como la realización de una actividad de forma paralela a la que pretende sustituir hasta que está lista para dicha sustitución. Por ejemplo, la copia de un archivo con un nombre temporal hasta que esté listo para sustituir al archivo que debe reemplazar.

15 Se usa el término en inglés para no añadir confusión, al ser esta una propiedad común de los routers.

2.6 MODIFICABILIDAD

La modificabilidad es la propiedad referida al riesgo y al coste de los cambios para un sistema.

En lo referente a esta propiedad el arquitecto debe tratar de identificar qué es lo que puede cambiar en el sistema, la probabilidad de que dichos cambios se realicen, cuándo se harían dichos cambios y por quién y cuál sería su coste.

En general, cuanta menos necesidad de cambiar tenga un sistema, o menos esfuerzo requieran los cambios, menos coste supondrán dichos cambios[16].

Las tácticas de este atributo se dividen en cuatro categorías:

- **Reducción del tamaño de los módulos**. Incluye la táctica de la división de módulos.
- **Incremento de la cohesión**. Se consigue a través de la táctica de incrementar la coherencia semántica.
- **Reducción del acoplamiento**. Se consigue mediante las tácticas de: encapsulación, uso de un intermediario, restricción de dependencias, refactorización y abstracción de servicios comunes.
- **Acoplamiento retardado**. Consiste en acoplar los componentes lo más tarde posible siempre que el coste de hacerlo compense los costes de la modificabilidad obtenida. Incluye múltiples tácticas según el tiempo de acoplamiento[17], ya sea en tiempo de compilación, despliegue, arranque o ejecución.

2.7 RENDIMIENTO

El rendimiento se concibe como la propiedad de un sistema software para cumplir los requisitos de tiempo.

En general, cada sistema tiene requisitos de rendimiento incluso si no han sido explícitamente detallados. Por ejemplo, es posible encontrar para cada sistema un tiempo de retardo a partir del cual su uso se hace prácticamente desaconsejable[18].

Las tácticas de esta propiedad se dividen en dos categorías:

- **El control de la demanda de recursos**. Consiste en mantener los niveles de demanda de manera que sean aceptables para el sistema. Incluye tácticas como: la gestión de las tasas de muestra, la limitación de los tiempos de respuesta, la priorización de eventos, la reducción

16 Esta propiedad es de capital importancia, aunque a menudo se pase por alto por no estar directamente relacionada con los requisitos funcionales del sistema. Como ya se sabe, la mayor parte de los costes del software se deben al mantenimiento. Una manera de reducir dichos costes es mediante la inversión en este atributo de calidad. Por suerte, la experiencia personal del autor es que, los responsables de las decisiones son cada vez más conscientes de la importancia de esta propiedad del software junto con la calidad en general, y tienden a invertir y a exigir ciertos niveles de la misma. De hecho, suele ser común el uso de métricas para identificar un bajo nivel en la modificabilidad del proyecto (aunque sea mencionada como calidad del software en general). Véanse proyectos como Sonarqube [5].

17 La lista es larga y puede consultarse en [Bass 13]. No se incluye aquí por motivos de brevedad.

18 Más aún, teniendo en cuenta que esta magnitud tiende a infinito así que siempre será posible encontrar una magnitud de tiempo superior a lo que ningún usuario es capaz de esperar.

de la sobrecarga, el establecimiento de límites de tiempos de ejecución y el aumento de la eficiencia de los recursos.

- **La gestión de recursos**. Consiste en operar sobre los tiempos de respuesta para hacerlos más eficientes en la gestión de la demanda. Incluye tácticas como: aumentar los recursos, introducir concurrencia, mantener múltiples copias de cálculo, limitar el tamaño de las colas y planificar los recursos.

2.8 SEGURIDAD

La seguridad se refiere a la habilidad de un sistema para proteger los datos y la información, para denegar los accesos no autorizados y, al mismo tiempo, para permitir el acceso a las personas que sí cuentan con autorización.

Una forma común de caracterizar la seguridad es mediante los atributos de:

- confidencialidad, que es la propiedad de los servicios de datos de protegerse de accesos no autorizados;
- integridad, que es la propiedad de los servicios de datos de no ser manipulados de forma no autorizada, y
- disponibilidad, que es la propiedad de un sistema de estar disponible para su uso legítimo.

Las tácticas de seguridad se dividen en cuatro categorías:

- **Detección de ataques**. Incluye tácticas como: detección de intrusos, detección de denegación de servicio, verificación de la integridad de los mensajes y detección de retardos en los mensajes.
- **Resistencia a ataques**. Incluye tácticas como: identificar a los actores, autenticar a los actores, autorizar a los actores, limitar el acceso, limitar la exposición, encriptar los datos, separar las entidades y cambiar la configuración por defecto.
- **Reacción ante ataques**. Incluye las tácticas de: revocación de acceso, bloqueo del ordenador y notificación a los actores.
- **Recuperación de los ataques**. Incluye las tácticas de auditoría de acciones y recuperación, dentro de las cuales se incluyen aquellas tácticas ya vistas en la disponibilidad para la recuperación de servicios.

2.9 TESTEABILIDAD

La testeabilidad[19] o capacidad de prueba se refiere a la facilidad con la que las pruebas son capaces de detectar las faltas existentes en un sistema.

Para que un sistema sea testeable, como mínimo, se debe posibilitar el control de las entradas y la observación de las salidas de cada uno de sus componentes. Además, si es posible, puede ser deseable la capacidad de observar los estados internos de sus componentes.

Las tácticas de este atributo se dividen en dos categorías:

- **Control y observación de los estados del sistema**. Incluye tácticas tales como: la

19 Se usa el término derivado del inglés por ser de uso amplio en la comunidad de habla española de desarrollo de software.

especialización de interfaces, la grabación y reproducción, la localización del estado de los datos, las abstracciones de los orígenes de datos, los *sandbox* o cajas de arena y las aserciones ejecutables.

- **Limitar la complejidad**. A medida que un sistema es más complejo se hace más difícil de probar. Esta categoría intenta disminuir la complejidad del mismo. Dentro de ella se utilizan las tácticas de: limitar la complejidad estructural, limitar el no determinismo.

2.10 USABILIDAD

La usabilidad se refiere a la facilidad con la que el usuario es capaz de completar la tarea deseada y el tipo de soporte que el sistema provee al usuario.

La usabilidad comprende áreas como: el aprendizaje de las características del sistema, el uso eficiente del sistema, la minimización del impacto de los errores, la adaptabilidad del sistema a las necesidades del usuario y el incremento de confianza y satisfacción.

Dentro de este atributo se incluyen dos categorías:

- **El soporte a la iniciativa del usuario**. Las tácticas comprendidas tienden a informar al usuario de lo que el sistema está haciendo y darle la posibilidad de que realice las acciones adecuadas. Incluye las tácticas de: cancelar, deshacer, pausar/reanudar y la composición de operaciones.
- **El soporte a la iniciativa del sistema**. Estas tácticas se refieren al uso de modelos para predecir el comportamiento del sistema o del usuario. Se incluyen tácticas como: mantener un modelo de tarea, mantener un modelo de usuario y mantener un modelo de sistema.

3. ARCHE: INTRODUCCIÓN Y MANUAL DE INSTALACIÓN

3.1 ARCHE: VISIÓN GENERAL

Tal y como se indica en [Bass 07][20], el sistema experto en diseño de arquitecturas (ArchE) es una herramienta experimental creada para asistir al arquitecto en el diseño de arquitecturas, cuya licencia es de uso no comercial y que ha sido desarrollada por el SEI.

En general el propósito de la herramienta es tratar de responder a la pregunta de cómo codificar el conocimiento de los atributos de calidad para ayudar al arquitecto durante el proceso de diseño.

Para ello, la herramienta se basa en cuatro conceptos diferentes:

1. **Escenarios** de atributos de calidad. ArchE requiere que cada escenario para un atributo de calidad sea especificado a través de una estructura formal compuesta por seis elementos: estímulo, origen del estímulo, entorno, artefacto activado por el estímulo, respuesta al estímulo, y forma de medir dicha respuesta.
2. **Responsabilidades**. Las actividades que se pretenden diseñar dentro del sistema se representan mediante responsabilidades. A medida que el diseño avanza, se añadirán nuevas responsabilidades y/o se dividirán o modificarán las existentes, con el fin de cumplir los requisitos de un determinado atributo de calidad.
3. **Marcos de razonamiento**. El conocimiento sobre un atributo de calidad se inserta dentro de un marco de razonamiento. Un marco de razonamiento incluirá el conocimiento abstracto relevante para ese atributo, y las posibles tácticas que pueden aplicarse como mecanismos de mejora.
4. **Tácticas arquitectónicas**. Una táctica arquitectónica posee la finalidad de satisfacer un atributo de calidad mediante la manipulación de un determinado aspecto de la arquitectura.

La aproximación de funcionamiento de ArchE consiste en que la herramienta contiene el conocimiento de los atributos de calidad y de cómo se relacionan con los requisitos de calidad de la arquitectura, pero no posee conocimiento semántico alguno de la arquitectura que se está diseñando. La función del arquitecto será proveer a la herramienta del conocimiento semántico del sistema.

El proceso de diseño a través de la herramienta consiste en el siguiente diálogo entre el arquitecto y la herramienta:

- El arquitecto introduce:
  - Las funcionalidades que necesitan ser resueltas por el sistema diseñado.
  - Los requisitos para los atributos de calidad especificados como escenarios.
- ArchE solicitará la información necesaria para determinar el comportamiento de los atributos de calidad, tales como el tiempo de ejecución o el coste de cambio de varias funcionalidades.
- ArchE
  - propone un diseño;
  - identifica los escenarios de los atributos de calidad que no se satisfacen con el diseño actual;

20 Aunque el artículo citado se refiere a la versión 2.1 de la herramienta, la funcionalidad aquí descrita puede aplicarse sin reparos a la versión 3.0 utilizada en el presente trabajo.

- presenta una lista de tácticas que podrían mejorar el diseño, como por ejemplo:
  - ❒ Insertar un intermediario para mejorar la modificabilidad de una cierta área.
  - ❒ Introducir concurrencia para mejorar el rendimiento de una cierta área[21].
- El arquitecto elige una o más tácticas, y ArchE las aplica para producir una nueva propuesta de diseño.
- El arquitecto podría necesitar introducir información adicional sobre el nuevo diseño tal y como nombres con sentido para los componentes introducidos.
- Este proceso se repetirá hasta que:
  - se cumplan los requisitos de calidad;
  - el arquitecto se considere satisfecho con el diseño actual, o
  - ArchE no sea capaz de realizar nuevas propuestas.

3.2 MANUAL DE INSTALACIÓN

Puesto que la herramienta ArchE se basa en Eclipse y este se basa, a su vez, en la plataforma Java, el primer paso será la instalación de la misma.

3.2.1 INSTALACIÓN DE LA JRE

En particular se instalará la última versión de la JRE 1.5, que puede obtenerse de la página oficial. A fecha de redacción de este trabajo esta versión es la 1.5.0.22. Para la instalación simplemente se ejecuta el instalador y se siguen los pasos indicados en el mismo.

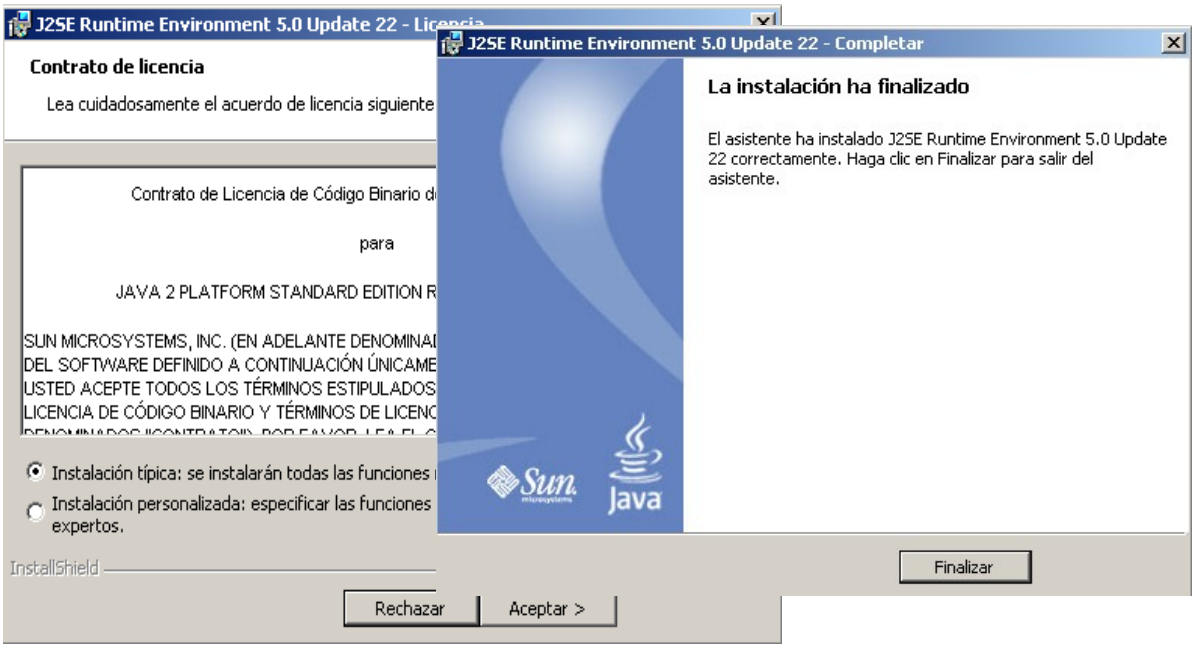


Ilustración 5: Asistente de instalación de Java J2SE 5.0.

21 Realmente el marco de razonamiento de rendimiento en su versión actual no propone ninguna táctica, al no estar completa su implementación, y se limita a comprobar si se satisfacen los requisitos de rendimiento. Esto último tampoco es cierto en determinados escenarios, como se verá más adelante, debido a un error en la implementación actual a la hora de contemplar el coste de las responsabilidades encadenadas y bifurcadas.

3.3 INSTALACIÓN DE ECLIPSE

A continuación se instalará Eclipse [10] en su versión 3.3. Para ello se obtiene el archivo zip, **Eclipse-SDK-3.3-win32.zip**, y se copia en la raíz del sistema[22]. En el caso de Eclipse la instalación se limita a descomprimir el archivo en la misma raíz sin crear ninguna carpeta intermedia (opción "Extraer aquí", en la mayoría de programas de compresión). En este caso se ha utilizado el propio asistente de Windows.

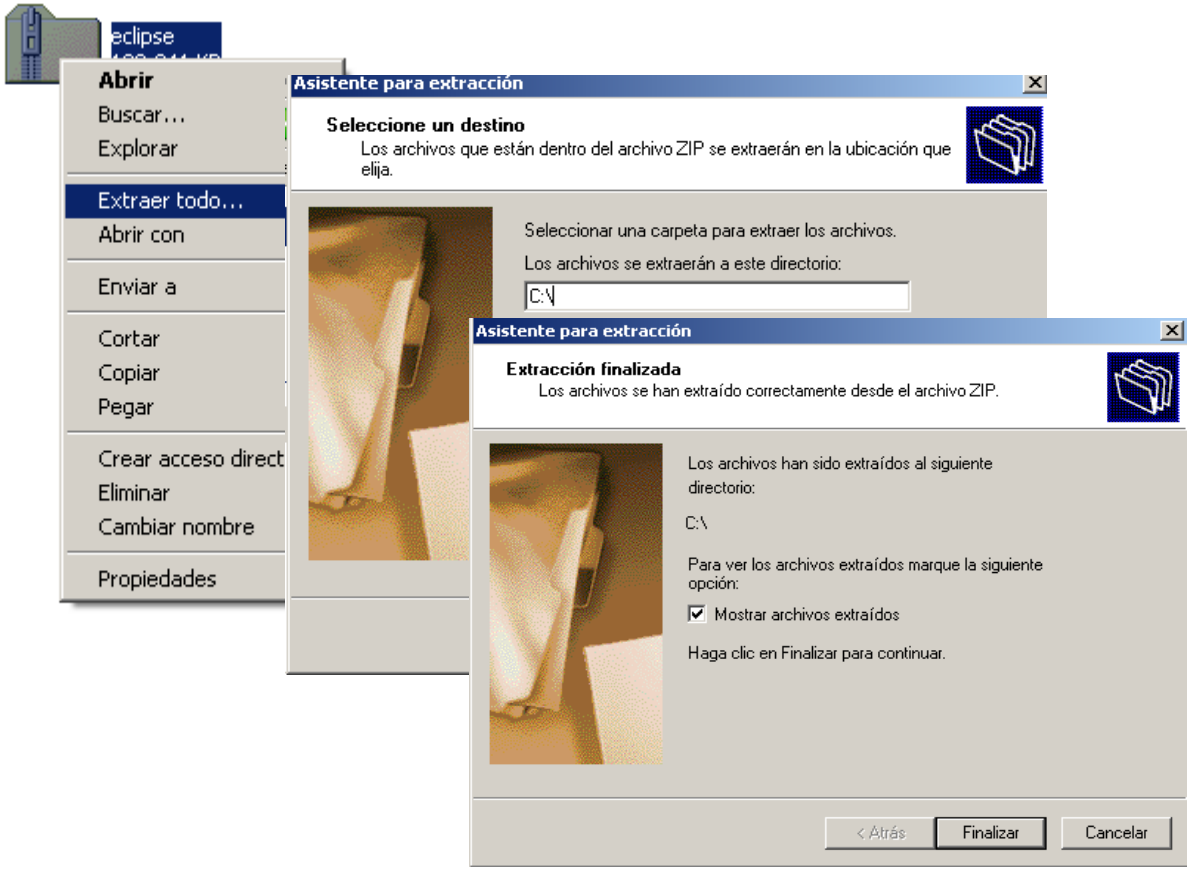


Ilustración 6: Descompresión de Eclipse en la raíz del sistema.

3.4 INSTALACIÓN DE MYSQL

El siguiente paso será la instalación de la base de datos MySQL [11] requerida por ArchE para la persistencia de los marcos de razonamiento y de los proyectos del usuario entre otros datos. Para ello se ha obtenido la versión 5.0.67[23].

El proceso consiste simplemente en ejecutar el instalador y seguir los pasos del mismo. Al llegar a la pantalla final es necesario desmarcar la casilla de configuración de la base de datos para que

22 Con el objetivo de simplificar se elige la raíz del sistema, C:\, para crear la estructura de directorios de la instalación. No se ha probado otra ruta de instalación, pero en principio no debería existir dificultad alguna para seleccionar una ruta diferente.

23 Tal y como ocurre con otras herramientas, se selecciona esta versión debido a que su compatibilidad con ArchE ha quedado plenamente probada en trabajos anteriores.

dicha configuración la realice el instalador de la herramienta ArchE.

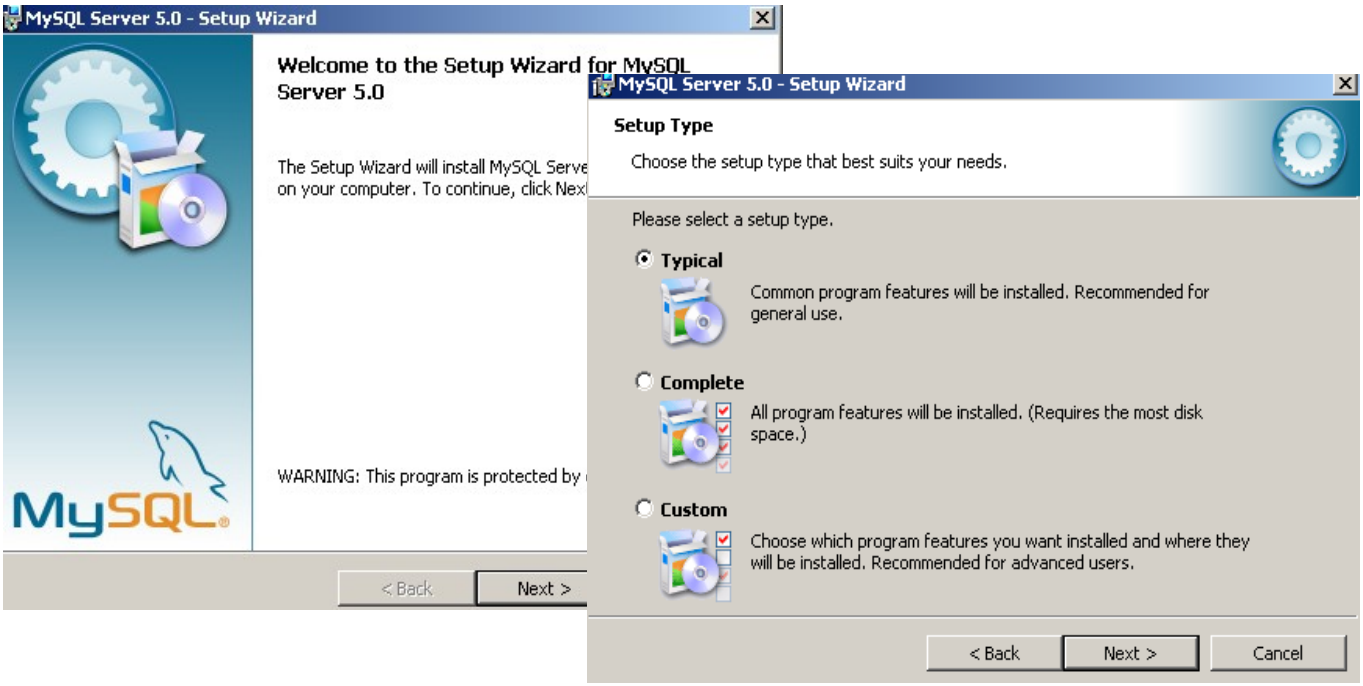


Ilustración 7: Asistente de instalación de MySql Server 5.0.

Ilustración 8: Finalización de la instalación de MySql. Se omite la configuración del servidor.

3.5 INSTALACIÓN DE GEF

La herramienta ArchE se basa en el *framework* de edición gráfico para Eclipse, GEF [9]. Para ello se obtiene el zip de instalación para su versión 3.3, **GEF-runtime-3.3.zip,** compatible con la misma versión de Eclipse y se descomprime nuevamente en la carpeta raíz del sistema (C:\).

Durante la descompresión del mismo será necesario reemplazar algunos archivos de la instalación de Eclipse. Para ello se pulsa la opción "Sí a todo" cuando el programa de compresión realiza la pregunta sobre el reemplazo de los mismos.

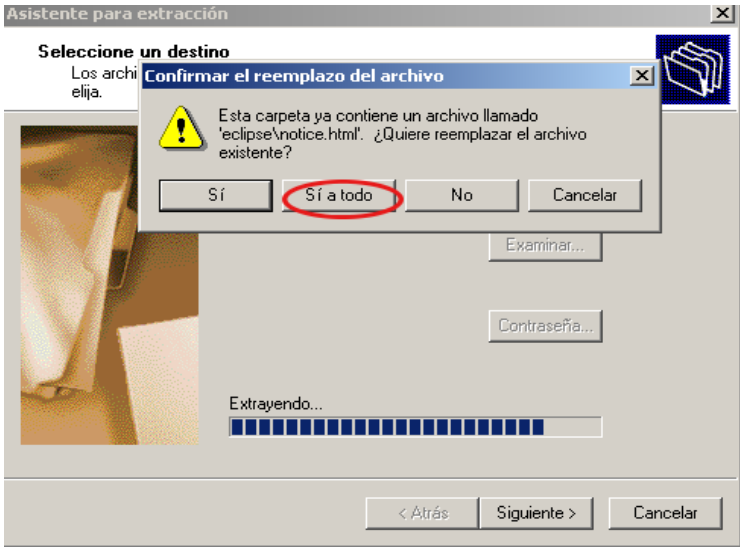


Ilustración 9: Descompresión de GEF en la raíz del sistema. Se reemplazan los archivos existentes.

3.6 INSTALACIÓN DE JESS

Jess [1] es un motor de reglas y entorno de *scripts* escrito completamente en Java en el cual se basa la herramienta ArchE. Mediante el uso de Jess, se puede desarrollar software Java dotado con la capacidad de "razonar" utilizando el conocimiento provisto en la forma de reglas declarativas. Según los autores, Jess es un motor de reglas pequeño y ligero y uno de los más rápidos que existen.

Para la instalación se descomprime el archivo **Jess71p1.zip**[24] en la raíz del sistema (nuevamente C:\). Tras la descompresión se habrá obtenido la estructura de directorios mostrada en la siguiente ilustración.

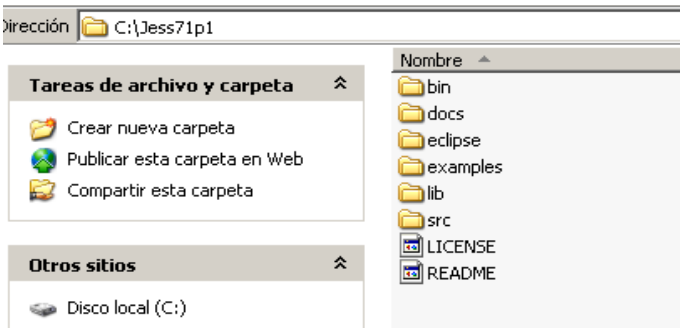


Ilustración 10: Estructura de directorios tras descomprimir Jess.

Dentro de dicha estructura se observa una carpeta con nombre "eclipse" que contiene a su vez unos

24 Se utiliza directamente el archivo obtenido de la UNED para evitar problemas con la licencia ya que esta herramienta no es gratuita.

archivos comprimidos que será necesario descomprimir. Tras la descompresión se obtienen dos nuevas carpetas, *features* y *plugins*, que será necesario copiar dentro de la carpeta de la instalación de Eclipse (C:\eclipse en el caso actual).

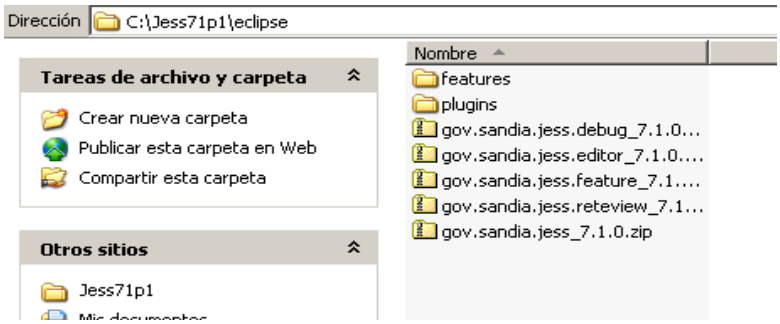


Ilustración 11: Descompresión de los plugins *y* features *para Eclipse.*

Una vez copiados, la instalación de Eclipse contendrá la *feature* y los *plugins* de Jess.

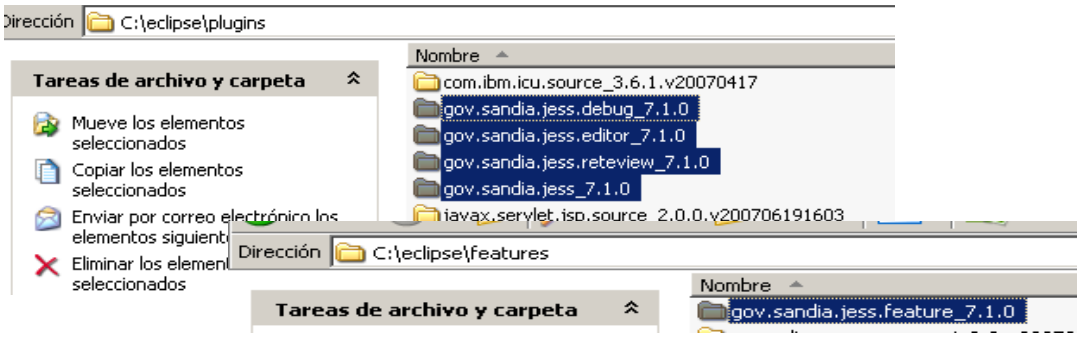


Ilustración 12: Plugins *y* feature *de Jees tras su instalación.*

3.7 INSTALACIÓN DE XMLBLASTER

XmlBlaster [2] es una plataforma de mensajería basada en XML que es necesaria para el correcto funcionamiento de la herramienta ArchE. En el caso actual se utilizará la versión 1.6.1, que se puede obtener directamente de la página oficial. Para su instalación se descomprime el archivo **xmlBlaster_REL_1_6_1.zip** nuevamente en la raíz del sistema. A fin de arrancar la plataforma se utiliza el *script* arrancar_xmlblaster.bat[25]. Este *script* deberá arrancarse siempre antes de utilizar la herramienta ArchE.

Para comprobar si la plataforma está funcionando correctamente se puede comprobar la dirección: http://iplocal:3412/status.html, donde iplocal será la IP local de la máquina[26].

25 Obtenido nuevamente de la UNED.

26 Esta puede obtenerse fácilmente utilizando el comando ipconfig. También puede utilizarse netstat -an para ver en qué dirección se está escuchando en el puerto 3412.

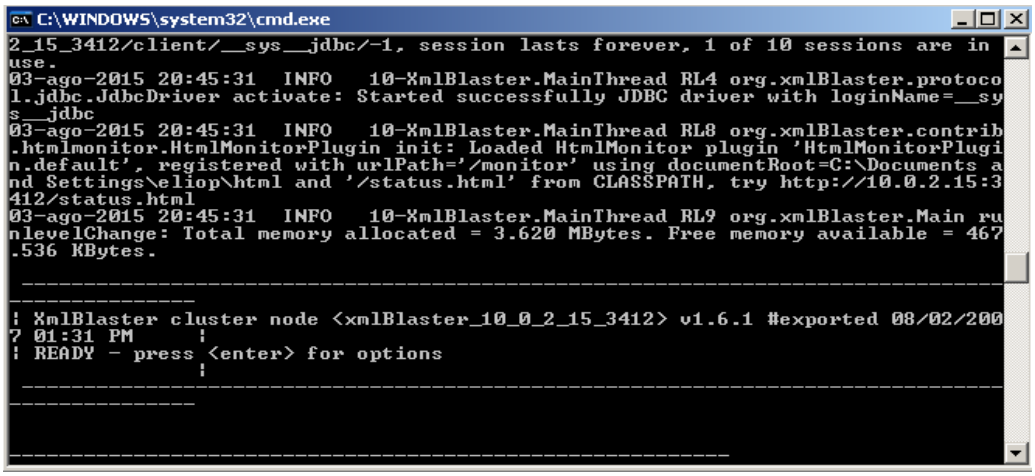


Ilustración 13: Plataforma XmlBlaster arrancada.

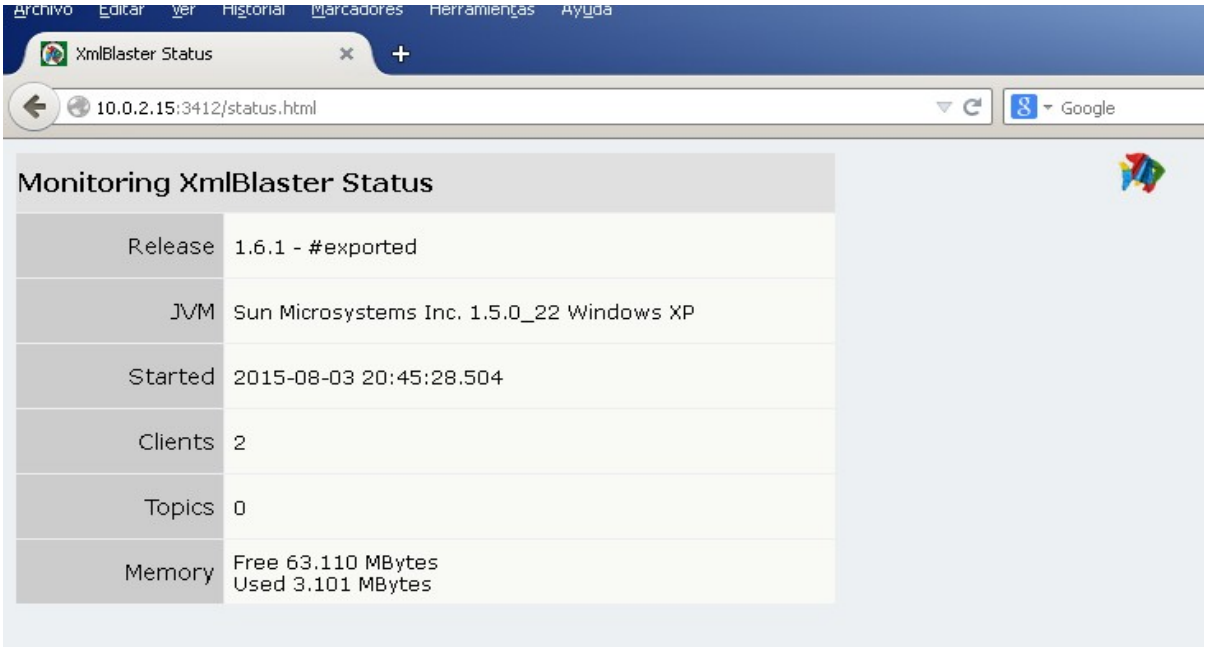


Ilustración 14: Página de estado de la herramienta XmlBlaster.

3.8 INSTALACIÓN DE ARCHE

Finalmente, una vez instaladas las dependencias de la herramienta se procede a la instalación de la propia herramienta ArchE. Para ello se descomprime el archivo **ArchE_3_0_0_Installer.zip**[27] y se ejecuta el instalador contenido en el mismo. Una vez ejecutado se siguen los pasos del asistente.

Durante la instalación la herramienta detectará la ruta de instalación de Eclipse (**C:\eclipse** en el caso actual). Además, también detectará que todas las dependencias han sido correctamente instaladas a excepción de XmlBlaster, el cual no es capaz de detectar. Simplemente se ignora dicha advertencia y se hace clic en siguiente.

27 Se utiliza nuevamente el archivo de la UNED, pues el SEI ya no mantiene la página de la herramienta, previamente http://www.sei.cmu.edu/architecture/tools/define/arche.cfm.

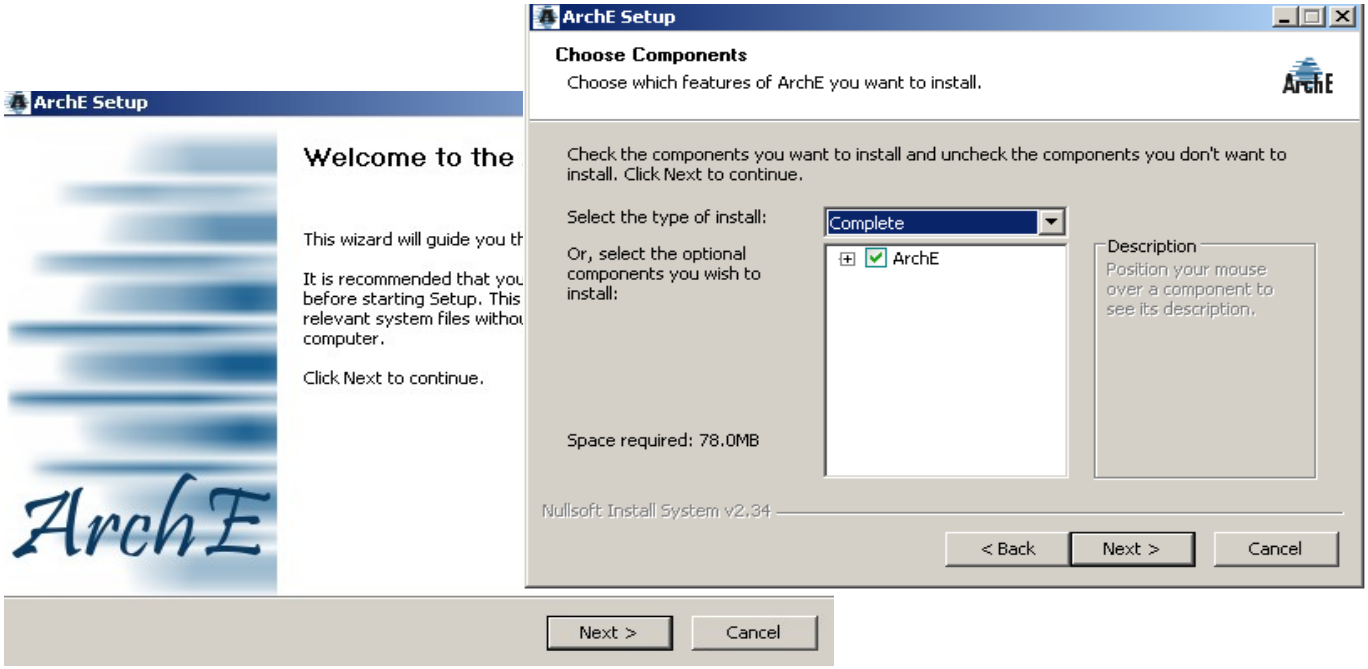


Ilustración 15: Asistente de instalación de ArchE.

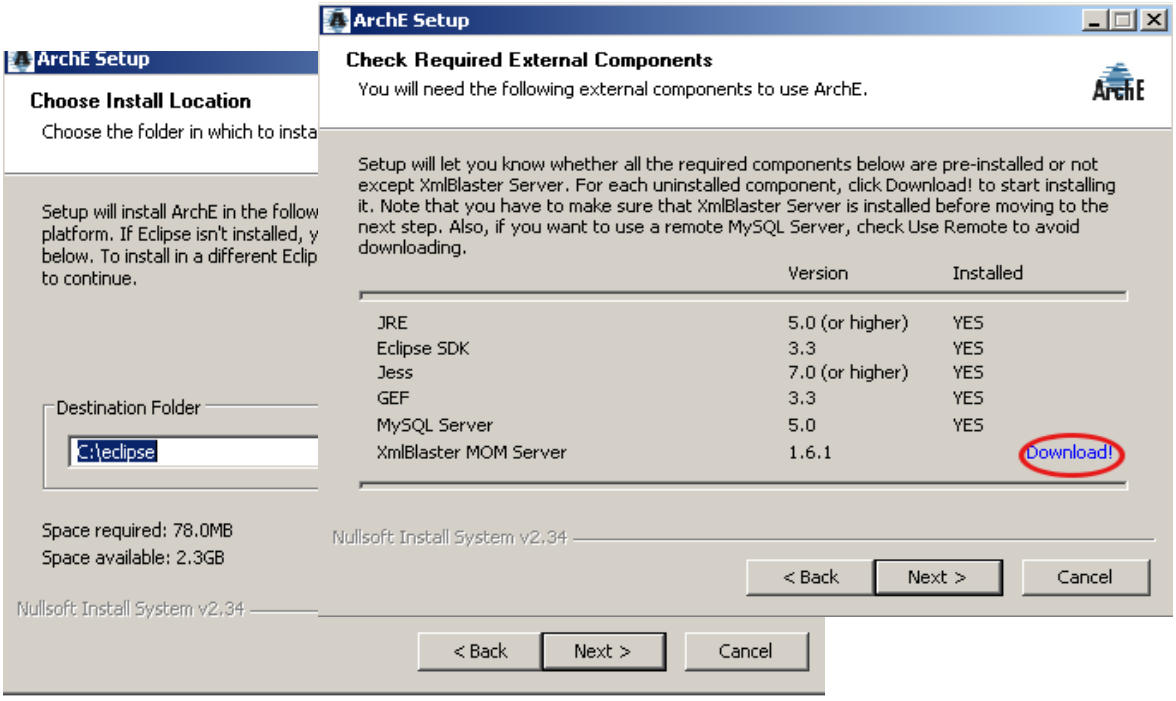


Ilustración 16: Indicación de la ruta de Eclipse y detección de dependencias. Se hace caso omiso a la falta de XmlBlaster.

A continuación el instalador procederá a configurar la base de datos MySql. Para ello solicitará la creación de la clave de *root*. Tras un primer intento de configuración el instalador avisará de que la clave anterior de *root* es necesaria; puesto que esta no se había creado se deja vacía y se reintenta la creación.

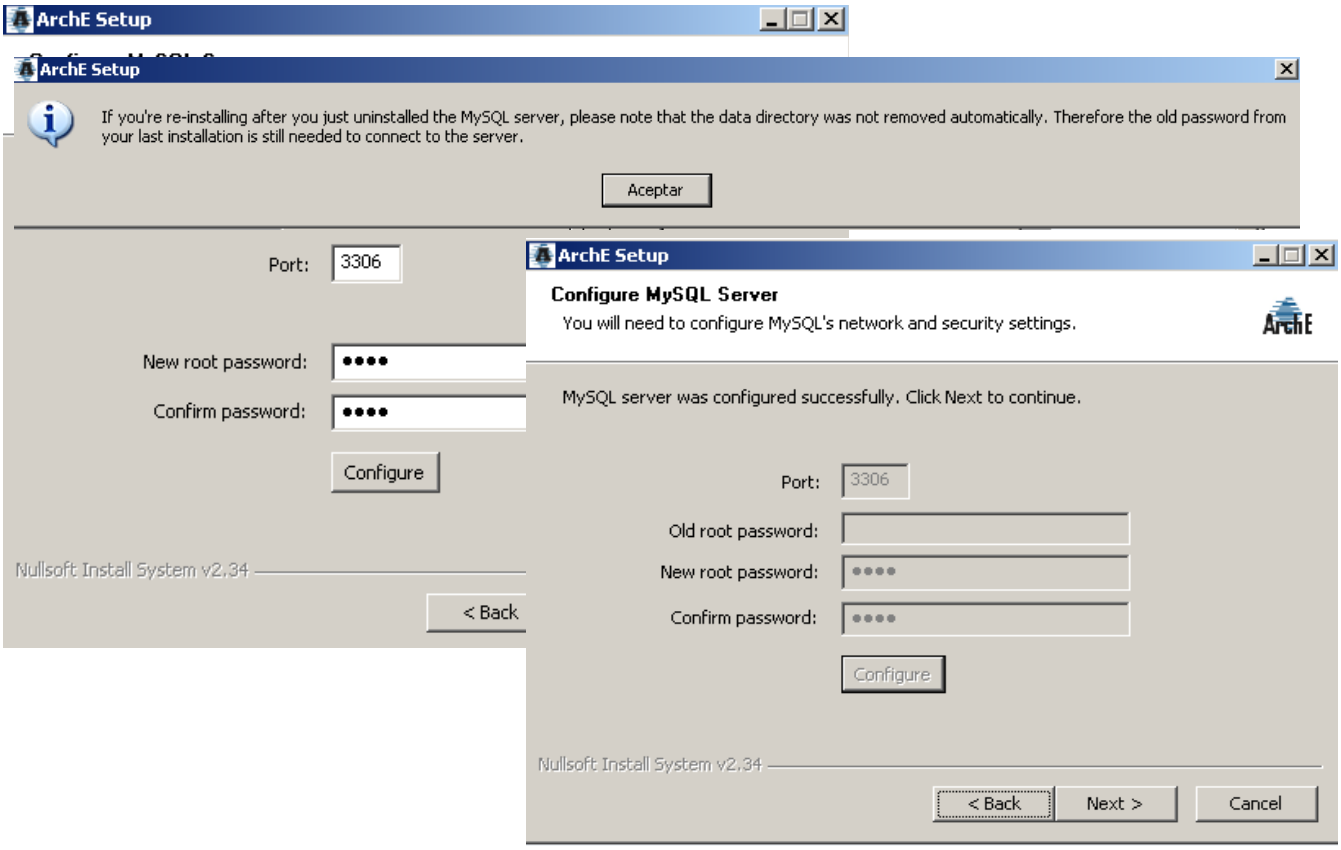


Ilustración 17: Configuración del usuario root *de la base de datos.*

Seguidamente, el instalador procederá a la creación de la base de datos de ArchE. Para ello solicitará de nuevo la clave del usuario *root*, además de la confirmación de los parámetros de conexión a la base de datos tales como el *host* y el puerto donde estará escuchando el servicio de MySQL.

Por último el instalador solicitará los parámetros de conexión con la plataforma de mensajería XmlBlaster, en particular, el *host* y el puerto de dicho servicio. En el caso actual serán localhost y 7607 respectivamente.

Llegados a este punto, el asistente procederá a completar la instalación y posteriormente indicará que esta ha finalizado.

Si todo ha ido bien, se podrá ejecutar la herramienta directamente desde el menú de inicio.

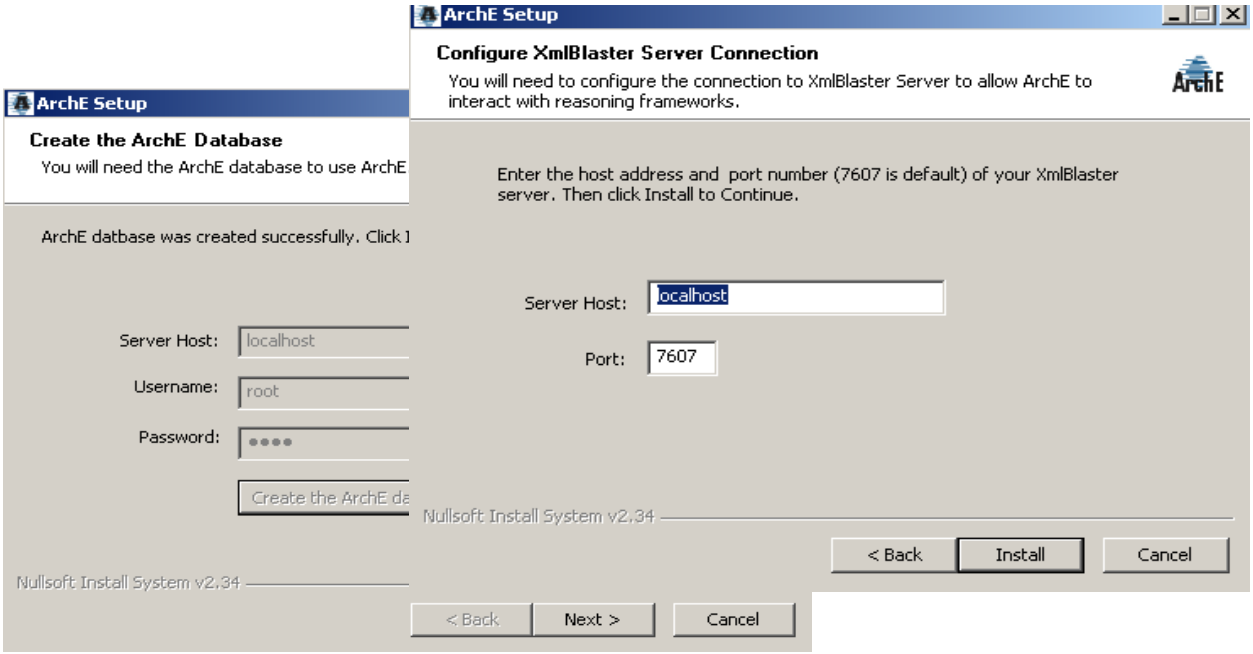


Ilustración 18: Creación de la base de datos ArchE y configuración de XmlBlaster.

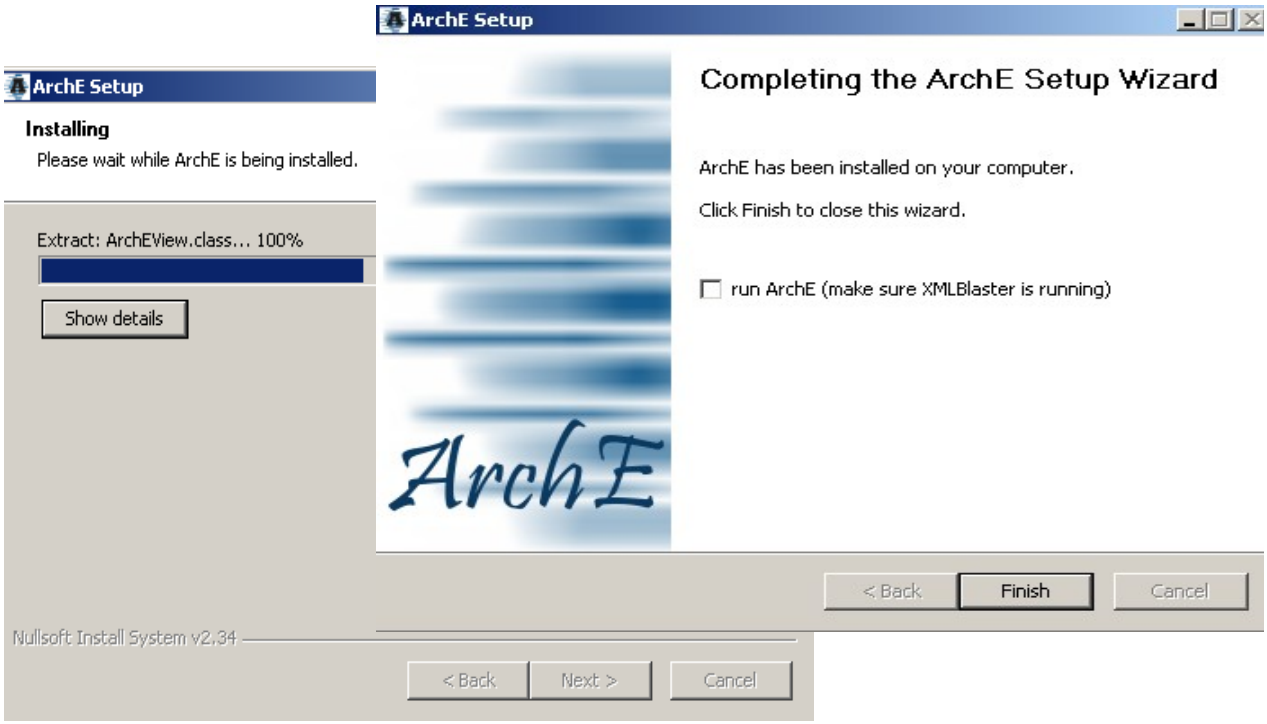


Ilustración 19: Final de la instalación de ArchE.

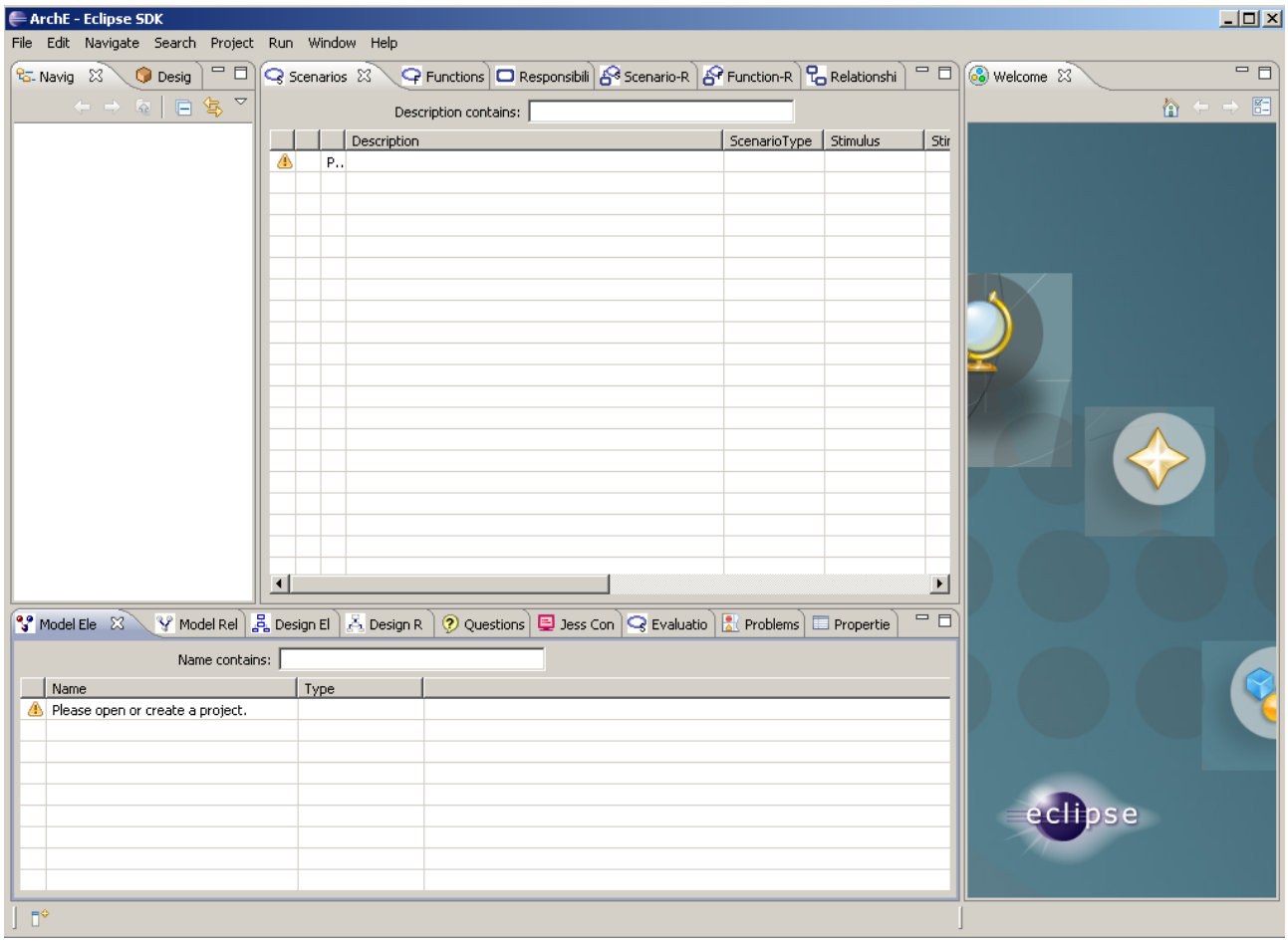


Ilustración 20: Herramienta ArchE en ejecución.

3.9 LIMITACIONES DE LA HERRAMIENTA

Puesto que el trabajo actual se centra en el marco de razonamiento de rendimiento se incluyen aquí exclusivamente las limitaciones relacionadas con él:

- El coste de las responsabilidades encadenadas a la inicial estimulada por el escenario no se tiene en cuenta a la hora de calcular si se cumplen los requisitos de rendimiento del escenario[28].
- El coste de las responsabilidades que realizan una bifurcación (la llamada de una responsabilidad a al menos otras dos distintas) no está soportado[29].
- El coste de las responsabilidades se limita a 10 milisegundos[30].
- Al analizar los escenarios, el coste se tiene en cuenta siempre en ms, independientemente de la magnitud seleccionada[31].
- Los escenarios cuya periodicidad no se indique, se interpretarán siempre como periódicos con una periodicidad de 10 ms. Esto quiere decir que los escenarios con periodicidad estocástica o esporádica no están soportados por la herramienta[32].
- El marco de razonamiento de rendimiento no propone ninguna táctica y se limita a calcular si se cumplen los requisitos indicados[33].
- La visualización de errores en la pestaña de marcadores no funciona y en caso de que el modelo generado no sea válido la aplicación no deja ningún *log*[34].
- En algunos casos con múltiples escenarios de rendimiento la herramienta MAST no es capaz de evaluar los datos de entrada debido a la duplicidad en los identificadores de los servicios[35].

28 Esto se debe a un error en la implementación del marco de razonamiento de rendimiento. Los detalles de dicho error y su solución pueden verse en el Anexo I. Con el objetivo de realizar este trabajo dicha limitación ha sido subsanada por el autor del mismo.

29 Este fallo surge a la vista tras subsanar el primero.

30 Dicha limitación ha sido también subsanada por el autor del trabajo. Los detalles pueden verse en el Anexo I.

31 Este problema también ha sido subsanado.

32 Para el presente trabajo se definirán exclusivamente escenarios de tipo periódico y se incluirá siempre la magnitud de la periodicidad de los mismos.

33 Debido a ello, en el presente trabajo se propondrán las tácticas fuera de la herramienta y se utilizará esta simplemente para comprobar que se cumplen los escenarios de rendimiento.

34 En el Anexo I se indica cómo se ha mejorado este problema.

35 Se debe a que se usa el campo Id de la clase Scenario el cual no siempre es único. Se ha corregido utilizando en cambio el campo FactId. Podría haberse usado también el campo Uuid.

4. PRESENTACIÓN DEL PROBLEMA

En los siguientes apartados se introduce el problema de la arquitectura correspondiente a una estación base de las redes de telefonía móvil. Por ser este un elemento extremadamente complejo tanto a nivel software como hardware[36], el trabajo se centrará en la arquitectura de la capa de operación y mantenimiento de dicho elemento.

4.1 DISEÑO GENERAL DE LA BTS

La estación base de transmisión (*Base transceiver station o* BTS, en inglés) es un elemento de la red de telefonía móvil. Dicho elemento utiliza las antenas de telefonía móvil para comunicarse con los dispositivos móviles[37]. Para ello se instala cerca de las antenas con frecuencia en condiciones atmosféricas poco amigables. Al mismo tiempo, una estación debe comunicarse con las estaciones vecinas para transferir la gestión de algunos de sus dispositivos móviles cuando estos salen fuera del alcance de sus antenas. Además, la estación se comunica con la red central del operador, lo cual hace posible la comunicación de los dispositivos móviles con otros dispositivos del mundo.

Cada estación base puede atender varias celdas las cuales representan un área dentro de la cual un UE puede comunicarse con la estación sin necesidad de realizar ninguna transferencia a otra celda. Normalmente dicha celda suele estar delimitada por el rango de alcance de las antenas de telefonía empleadas en la misma.

A lo largo del tiempo las redes de telefonía han ido evolucionando, dando lugar a varias generaciones según los elementos y tecnologías que utilicen. Las generaciones actuales se encuentran recogidas dentro del estándar 3GPP, con el cual deben ser compatibles los suministradores de redes.

Actualmente se encuentra ampliamente difundida la generación 4G, también conocida como LTE, que basa su funcionamiento en técnicas de modulación más eficientes y un uso general del direccionamiento IP para identificar cada uno de los elementos de la red. Esta y otras mejoras hacen posible una mejor experiencia tanto para el usuario, con mayores velocidades para el contenido multimedia, como para el operador, con un mayor aprovechamiento de las bandas de frecuencia que le han sido asignadas.

Dentro de una red LTE, el elemento correspondiente a la estación base con las responsabilidades expuestas anteriormente recibe el nombre de *eNodeB*. A continuación se incluyen los componentes de una red LTE.

36 Una estación base suele tener varios módulos hardware cada uno con múltiples procesadores y con un software de varios millones de líneas de código.

37 En el ámbito de telecomunicaciones se suele utilizar el término *user equipment* (UE) para referirse a cualquier dispositivo móvil ya sea un teléfono, un portátil, un tablet, etc.

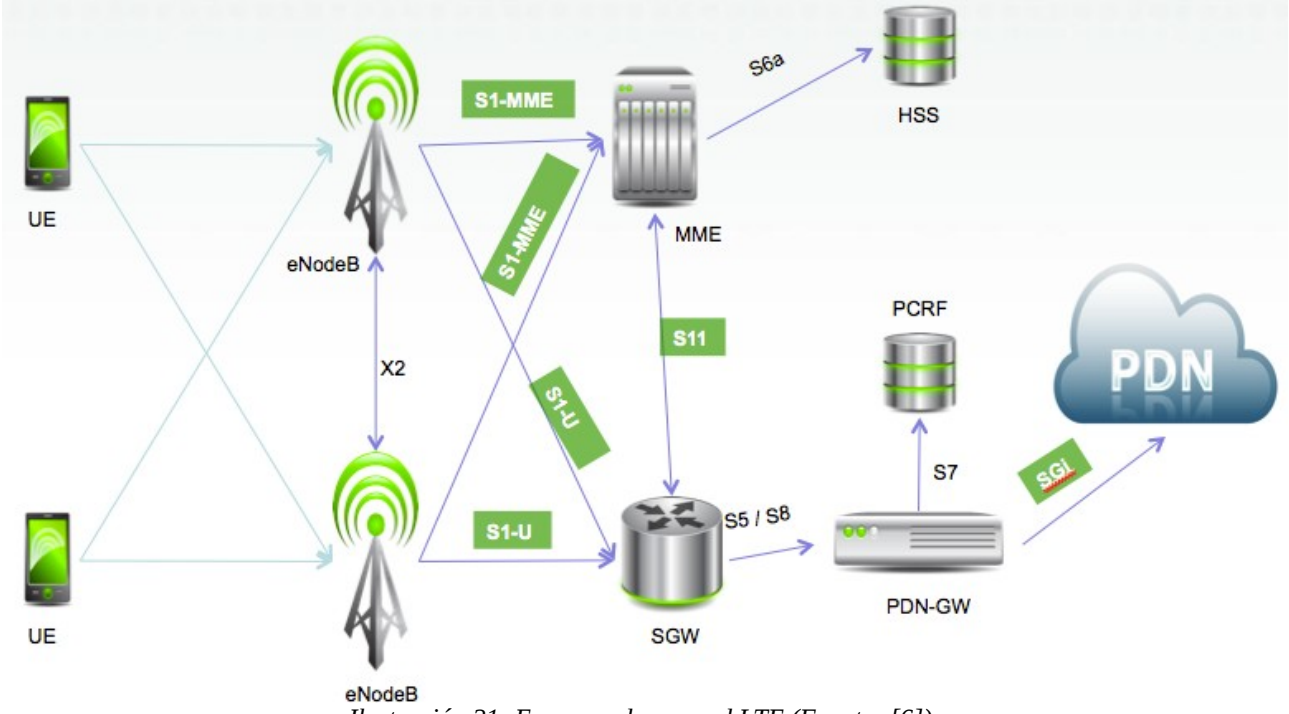


Ilustración 21: Esquema de una red LTE (Fuente: [6]).

En el diagrama anterior se aprecia cómo la conexión entre estaciones vecinas se realiza a través del enlace x2, mientras que los datos de control y de usuario se envían por las conexiones S1-MME y S1-U respectivamente[38].

Dentro del software de la BTS la arquitectura puede dividirse en tres capas principales según los datos que estas gestionen:

- **U-PLANE**. Esta capa gestiona los datos de usuario, tales como el contenido de los datos que un equipo envía a internet, los datos de voz de una conversación telefónica, etc.
- **C-PLANE**. Esta capa gestiona los datos de control, como el establecimiento de la conexión con el UE, su autenticación y autorización, o la transferencia del mismo a una estación vecina.
- **M-PLANE**. Esta capa se encarga de gestionar el hardware, encendiéndolo o apagándolo, así como de permitir al operador la gestión de la estación, la gestión de la configuración, el envío de alarmas, etc.

4.2 ARQUITECTURA DE LA CAPA M-PLANE

A continuación se muestra la arquitectura de la capa M-PLANE dentro de una BTS en la que se centrará el presente trabajo[39].

38 En el diagrama se observan otros elementos como el MME, HSS, SGW, PDN-GW. Estos elementos se omiten intencionadamente al no ser de interés para el problema que se trata aquí de resolver. No obstante, sirva como nota que estos elementos se encargan de conectar el *eNodeB* con la infraestructura del operador, para los datos de facturación, etc, y con la red mundial de telefonía.

39 La arquitectura aquí ilustrada es, como cabe entender, una arquitectura simplificada a partir de la real utilizada en el software empresarial. Muchos nombres, por tanto, se han cambiado, para adaptarse al nuevo esquema simplificado.

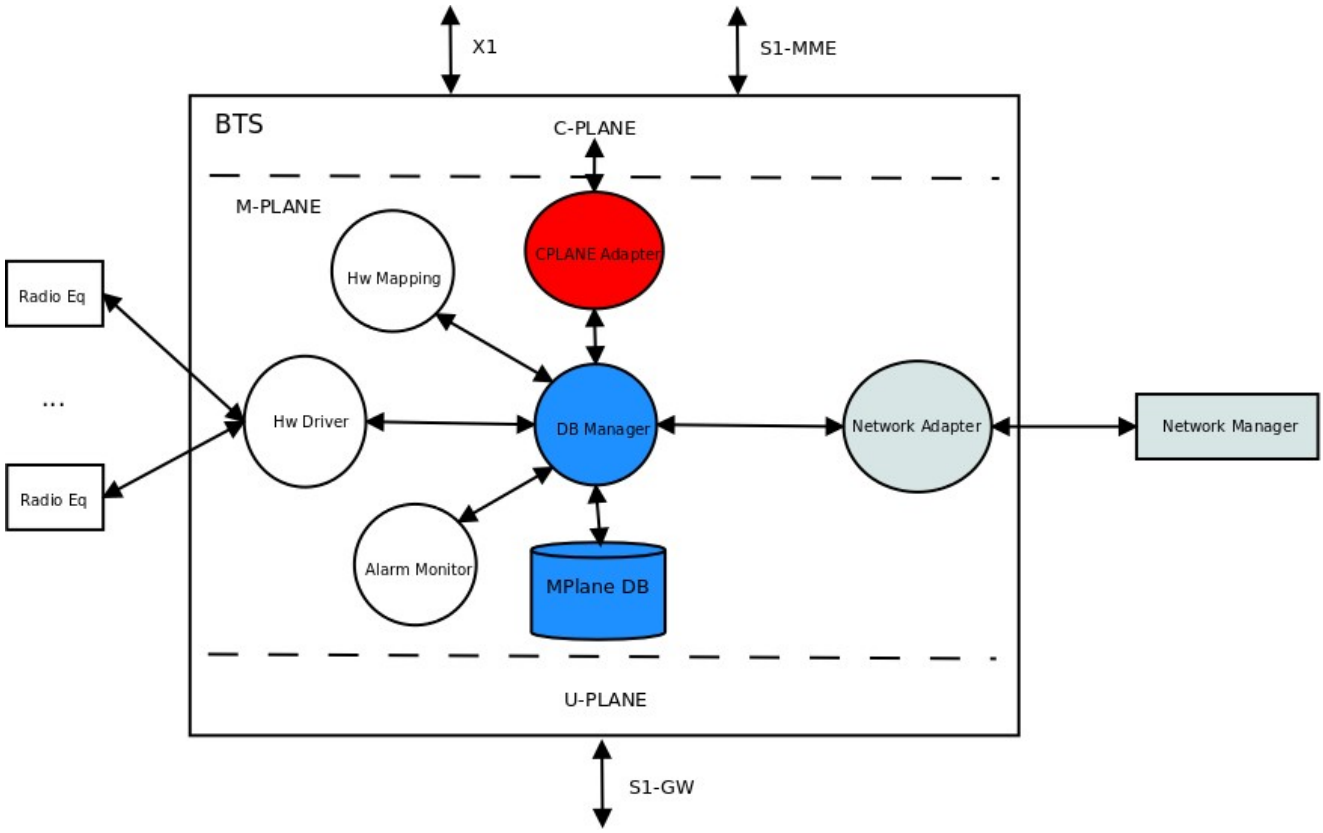


Ilustración 22: Arquitectura de la capa M-PLANE de la BTS (Fuente: elaboración propia).

El diagrama anterior muestra una serie de procesos, ilustrados mediante círculos que corren en el mismo procesador y que se comunican entre sí mediante el paso de mensajes en memoria. Además se identifica una base de datos M-PlaneDB la cual se basa en una estructura en forma de árbol almacenada en la memoria RAM, y por tanto volátil. Además, fuera de la BTS se identifican varios sistemas software, tales como el NetworkManager y los equipos de radio, que se comunican mediante paso de mensajes a través de protocolos como el TCP.

Centrándose en la capa M-Plane, la arquitectura expuesta en el diagrama presenta las siguientes características:

- La arquitectura es una modificación del patrón *blackboard*, donde todos los procesos leen y escriben en la base de datos de la capa M-Plane, a través del proceso intermediario DB Manager mediante solicitudes de lectura y escritura.
- Al igual que en el patrón *blackboard*, la única forma de comunicación entre los componentes de la capa M-Plane es a través de la base de datos[40].
- Cada proceso es dueño de un subconjunto de los elementos de la base de datos y sólo él puede modificarlo. Esta propiedad es más un acuerdo de diseño que una restricción del sistema. El resto de procesos sólo pueden leerlos y la única forma de forzar un cambio en un elemento del que no se es propietario es solicitarlo al proceso propietario. Estas peticiones pueden realizarse mediante la ejecución de llamadas RPC a través del propio protocolo de la base de datos. Para ello algunos procesos declaran servicios que definen las operaciones que

40 Con la excepción de las llamadas RPC, tal y como se explica seguidamente aunque estas también están integradas en el funcionamiento de la base de datos.

pueden ser invocadas por otros procesos clientes[41].

- Cada componente mantiene una copia local en memoria de la base de datos la cual obtiene del proceso intermediario mediante una operación de solicitud de integridad al conectarse al mismo. Cuando el proceso intermediario recibe dicha solicitud, envía el contenido íntegro de la base de datos con su estado actual. Posteriormente, cada vez que un proceso modifica algún elemento, mediante la correspondiente solicitud de escritura al intermediario, este notifica dicho cambio al resto de componentes suscritos a la base de datos.

En lo que se refiere a las responsabilidades de cada uno de los componentes ilustrados en el diagrama, son las siguientes:

- **Network Adapter**. Este proceso se encarga de comunicar la BTS con las herramientas de gestión de la red del operador mediante el envío de mensajes a la aplicación NetworkManager. Para ello recibe la configuración de la BTS, la parsea[42] y la introduce en la base de datos. Dicha configuración posee información de la propia BTS como el hardware conectado a la misma, su nombre lógico dentro de la red, cuántas celdas debe operar y con qué recursos, etc. Además, envía el estado del hardware interconectado al Network Manager cada vez que este cambia, así como las alarmas del sistema y los cambios detectados en las estaciones vecinas.
- **Network Manager**. Tal y como se ha explicado anteriormente se comunica con la BTS y le envía el fichero de configuración de la misma. Además, recibe de esta el estado del hardware, sus alarmas, y la información de las estaciones vecinas.
- **Hw[43] Driver**. Se encarga de la comunicación de bajo nivel con los equipos de radio y otros elementos hardware. Para ello los autodetecta, configura y gestiona con operaciones de parada, arranque, etc.
- **Radio Eq[44]**. Se encarga de modular la señal física para comunicar con los UE. Para ello recibe los datos de la BTS y una vez modulados los envía a las antenas de telefonía. Igualmente recibe los datos del UE y realiza la conversión inversa para decodificarlos y posteriormente enviarlos a la BTS.
- **Hw Mapping**. Se encarga de analizar el hardware declarado en la configuración y compararlo con el hardware detectado por el Hw Driver. Como resultado de dicha comparación, establece el estado correspondiente para los elementos hardware. Dichos estados pueden ser: detectado-no configurado, detectado-configurado y no detectado-configurado[45].
- **Alarm Monitor**. Se encarga de monitorizar los procesos del sistema y los elementos hardware con el fin de generar o borrar las alarmas correspondientes según sus estados.
- **C-Plane Adapter**. Se encarga de comunicar con la capa C-Plane mediante el paso de mensajes, para poner en operación las celdas y para obtener información de las estaciones vecinas.
- **DB Manager**. Se encarga de recibir las operaciones de lectura y escritura para los elementos de la base de datos, realizarlas en la misma y notificarlas a los componentes suscritos. Además, envía la información de integridad a los componentes con el estado de la base de

41 Este mecanismo es muy parecido al de las invocaciones RPC del mundo Java, ampliamente utilizadas en los servidores CORBA.
42 Se refiere a la acción de leer la estructura sintáctica del fichero y extraer su información semántica.
43 En lo referente a los nombre de los procesos y responsabilidades se usará Hw como abreviatura de hardware.
44 Eq es abreviatura de *equipment* (“equipo” en español).
45 También puede estar no detectado-no configurado, pero este estado no es de la incumbencia de la BTS.

datos una vez se suscriben a la misma. Por último ejecuta las llamadas RPC mediante su redirección al proceso destino y el posterior reenvío del resultado al proceso llamante.

- **M-Plane DB**. Base de datos de la capa M-Plane. Es una base de datos en forma de árbol donde cada elemento es un objeto, identificado por un nombre jerárquico y que posee una serie de atributos. Esta base de datos está almacenada en memoria y su contenido no se preserva entre rearranques de la BTS.

Aunque en el diagrama aparece la capa U-Plane, esta no se tiene en cuenta en adelante, ya que toda comunicación con la misma se realiza a través de la capa C-Plane y por tanto no es necesario su análisis en la arquitectura de la capa M-Plane.

4.3 ANÁLISIS DE LA ARQUITECTURA DE LA CAPA M-PLANE

La arquitectura expuesta anteriormente presenta las siguientes características con respecto a los atributos de calidad:

Atributo	*Evaluación*
Disponibilidad	La arquitectura actual tiene una sola instancia para cada componente, por lo que basta con que deje de funcionar para que se interrumpa el procesamiento de dicho componente. Dependiendo del componente que falle, el sistema puede quedar en un estado degradado o, como en el caso del componente DBManager, provocar la inutilidad del sistema[46].
Interoperabilidad	Al comunicarse todos los componentes mediante una única interfaz, el de la propia base de datos, no es difícil añadir nuevos componentes, o extraer los datos de la base de datos.
Modificabilidad	La arquitectura actual, por su herencia del patrón *blackboard*, posee un bajo acoplamiento ya que los elementos realizan invocaciones de otros elementos de forma implícita. Esto significa que la modificación de los flujos se consigue simplemente con modificar los estados a los que los componentes reaccionan y no es necesario modificar aquellos que los producen. Para el caso de las llamadas RPC el acoplamiento tampoco se ve afectado, ya que los servicios se realizan mediante la creación de objetos servidores en la base de datos[47]. Por tanto, un proceso cliente invoca el objeto servidor y no el proceso que lo gestiona, pudiendo cambiar este en el futuro.
	Del mismo modo, los dominios de conocimiento favorecen la cohesión, ya que cada objeto de la base de datos pertenece a un único proceso.
	El hecho de que los componentes de la arquitectura sean distribuidos y de que cada uno posea su propia copia de los datos significa que cada uno

46 Esta restricción es aceptada y la forma de paliarla es introduciendo redundancia a nivel de red mediante múltiples BTS. De esta forma, si una BTS falla, un UE podrá comunicarse a través de otra.

47 Además, hay que tener en cuenta que las llamadas RPC pueden ser modeladas como la creación de un objeto petición al que el servidor está subscrito. Por otro lado, en la arquitectura evaluada su uso no tiene relevancia, por lo que no se tendrá en cuenta en capítulos posteriores.

	tendrá su propia vista de la base de datos, cuyo contenido sólo se puede determinar a través de implicaciones semánticas[48]. Esto dificulta el razonamiento sobre los flujos de control y de datos entre los componentes.
Rendimiento	El hecho de que cada componente tenga su propia vista de la base de datos favorece que el rendimiento de las operaciones de lectura sea óptimo ya que se hacen directamente en la propia memoria del proceso.
	Las copias de la base de datos crean una redundancia innecesaria que disminuye la memoria disponible en el sistema.
	Las operaciones de escritura tienen un retraso desde que son realizadas hasta que están disponibles para los componentes interesados, ya que la comunicación con el proceso intermediario y la posterior replicación conlleva un retraso.
Seguridad	Aunque los aspectos de seguridad de la BTS se gestionan por una entidad externa, el patrón *blackboard* mejora la implementación de mecanismos de seguridad al disminuir los puntos de entrada y las relaciones entre componentes.
Testeabilidad	Puesto que todos los flujos de control se realizan a través de cambios en la base de datos, para observar estos, basta con suscribirse a la base de datos como un cliente más. Igualmente, puede sustituirse fácilmente un componente y realizar operaciones en la base de datos para su simulación a fin de probar otros componentes.
Usabilidad	Los aspectos de usabilidad no aplican a la arquitectura descrita, aunque, no es difícil implementar modelos del usuario o el sistema, ya que todos los estados de los componentes están disponibles en la base de datos y son fácilmente extraíbles de esta.

Tabla 1: Evaluación de la arquitectura de M-Plane con respecto a los atributos de calidad.

Como se desprende de la tabla anterior, los puntos mejorables de la arquitectura actual son:

- La redundancia de la base de datos, debido a las copias de la base de datos en cada uno de los procesos[49].
- Los retardos en el procesamiento, debido a la comunicación con el intermediario para la modificación y posterior replicación.
- La dificultad para razonar a la hora de diseñar, debido a la existencia de múltiples vistas de la base de datos a lo largo de los distintos procesos.

Como ejemplo del último punto véase el caso que se incluye en el subapartado siguiente.

48 Por ejemplo se puede deducir que un componente A ha visto un cambio *c* porque ha cambiado al estado *s* el cual tiene como prerrequisito a *c*.

49 En realidad esta desventaja no es tan grave, ya que no todos los componentes están interesados en todos los objetos de la base de datos, de forma que cada uno de ellos contiene un subconjunto de la misma. No obstante, la redundancia sigue existiendo. En el presente trabajo, por simplificar, se asumirá que todos los componentes poseen una copia de todos los objetos.

4.4 ILUSTRACIÓN DE LOS PROBLEMAS DE RAZONAMIENTO PARA EL DISEÑO CON LA ARQUITECTURA ACTUAL

A continuación se incluye un diagrama de secuencia que ilustra un problema de diseño con la arquitectura actual. El caso expuesto muestra la autodetección de dos equipos de radio que han sido previamente configurados por el operador, por lo que deberían mostrarse en el componente Network Manager como detectado-configurado.

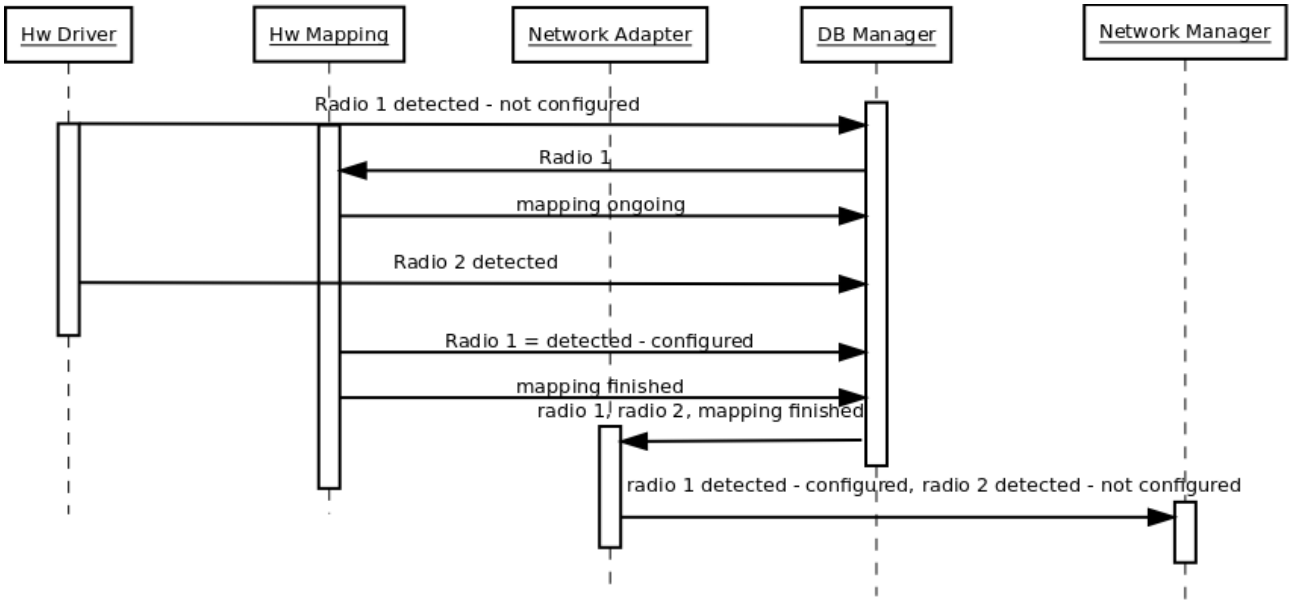


Ilustración 23: Ilustración de una condición de carrera en la detección de equipos de radio[50] [51] (Fuente: elaboración propia).

La condición de carrera ocurre por la siguiente secuencia de eventos:

- Network Adapter monitoriza el estado del objeto *mapping* para saber si Hw Mapping ha terminado de relacionar el hardware detectado con la configuración,
- Hw Mapping advierte la nueva radio detectada y comienza su tarea de análisis del hardware y la configuración existente. Para indicar que está trabajando cambia el estado de su objeto *mapping* a *ongoing*.
- Antes de que Hw Mapping termine su tarea, Hw Driver detecta una nueva radio.
- Posteriormente Hw Mapping indica que ha finalizado mediante el cambio de estado en su objeto *mapping*.
- Network Adapter recive las dos radios, una como configurada y la otra no, además de la notificación de que el Hw Mapping ha terminado.
- Network Adapter notifica a Network Manager que hay dos radios conectadas, una configurada y la otra no.
- Network Manager muestra tres radios al operador, ya que la configurada y no detectada se

50 Este problema ocurre en el software real y se debe a que el componente que realiza el trabajo de Network Adapter no sabe cuál es el estado del hardware según el componente Hw Mapping.

51 La modificación de la Radio 1 por parte de Hw Mapping viola la arquitectura ya que esta también es modificada por Hw Driver. En la realidad, esta operación se realiza mediante una notificación de Hw Mapping a Hw Driver para que modifique el valor, salvaguardando así la arquitectura. En el diagrama se han omitido estos pasos para ganar en simplicidad.

considera una radio adicional, dando así lugar a la situación errónea[52].

52 Nótese que el problema es temporal, ya que Hw Mapping acabará procesando la segunda radio, y Network Adapter actualizará el estado al componente externo. No obstante no deja de ser una situación errónea que debe ser subsanada.

5. PROPUESTA DE DISEÑO

Partiendo de los puntos débiles identificados en el capítulo anterior se propondrá una arquitectura mejorada que tratará de eliminarlos. Las características diferenciadoras de esta nueva arquitectura son:

- La base de datos es ahora un archivo de memoria mapeada accesible por todos los componentes de la arquitectura, de forma que ya no existen más copias de la base de datos.
- Debido al punto anterior se elimina el problema de las distintas vistas de la base de datos de cada uno de los procesos.
- Para seguir manteniendo una única interfaz de comunicación con la base de datos se introduce una API para acceder a la misma, DB Client. Esta API consiste en una librería dinámica que realizará las operaciones de bajo nivel sobre los objetos de la memoria compartida. Así mismo incluirá mecanismos de sincronización para garantizar la integridad de las transacciones como por ejemplo mediante el uso de semáforos. Por último, las llamadas RPC se realizan mediante invocaciones a los procesos que gestionan el objeto servidor directamente desde el proceso llamante. Es decir, el proceso es similar pero se elimina el paso por el intermediario.
- Con el fin de evitar que los componentes tengan que hacer *polling* continuamente sobre la base, lo que antiguamente se conseguía escuchando únicamente los cambios recibidos desde el DB Manager, ahora se introduce una cola de cambios y un proceso notificador de cambios para los componentes suscritos. La API DB Client será la encargada de introducir los cambios en la cola de cambios, y el proceso DB Notifier será el encargado de replicarlos a los clientes suscritos.

El diagrama siguiente ilustra la arquitectura propuesta.

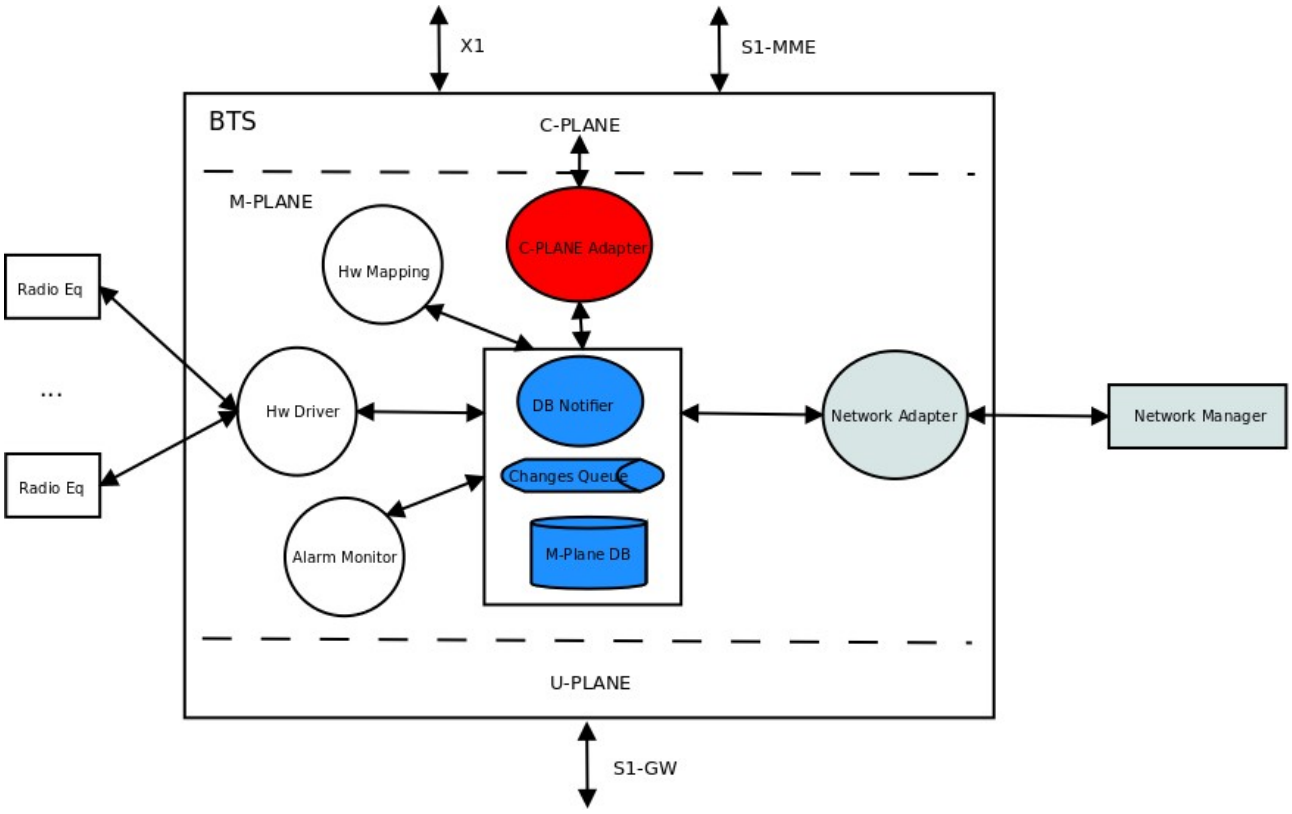


Ilustración 24: Arquitectura mejorada de la capa M-Plane (Fuente: elaboración propia).

A continuación se reevalúa la nueva arquitectura con respecto a los atributos de calidad:

Atributo	*Evaluación*
Disponibilidad	La nueva arquitectura no introduce cambios con respecto a este atributo.
Interoperabilidad	La base de datos sigue siendo el único punto de comunicación con lo que los beneficios explicados para las arquitecturas anteriores siguen siendo actuales.
	Al dejar de ser la base de datos un servicio basado en mensajes, y pasar a accederse a través de una API, si se quieren añadir nuevos componentes, estos deberán ser locales, o bien habrá que utilizar algún tipo de gateway que se comunique localmente con la base de datos y se comunique mediante mensajes con el nuevo componente externo a la BTS.
Modificabilidad	El flujo de control sigue siendo gobernado por invocaciones implícitas, con los beneficios vistos anteriormente.
	Los dominios de conocimiento siguen favoreciendo la cohesión como se afirmó anteriormente para la arquitectura original.
	El uso de transacciones es ahora indispensable ya que, si un proceso es ejecutado por un cambio notificado, podrá leer cambios posteriores de la base de datos (los cuales todavía no han sido procesados y permanecen en su cola de notificaciones) durante el tratamiento de uno previo.

Rendimiento	Puesto que la base de datos es ahora una memoria compartida las lecturas pueden verse afectadas por la sincronización de las transacciones.
	Las operaciones de escritura son rápidas, ya que se realizan directamente sobre la base de datos y sólo se requiere el coste adicional de introducirlas en la cola de cambios.
Seguridad	Los aspectos de seguridad no cambian con respecto a la arquitectura anterior.
Testabilidad	Los aspectos de testeabilidad no cambian en general, con la excepción de que si la herramienta de pruebas está fuera de la BTS necesitará un *gateway* para realizar cambios sobre la base de datos ya que esta no es ahora accesible directamente.
Usabilidad	Los aspectos de usabilidad no cambian.

Tabla 2: Evaluación de la arquitectura mejorada con respecto a los atributos de calidad.

En resumen:

- Se ha mejorado el rendimiento mediante la eliminación de colas y de la comunicación basada en mensajes.
- La modificabilidad ha mejorado en ciertos aspectos al desaparecer las copias, pero ahora aparece con más fuerza el problema de las lecturas sucias[53], con lo que se hace indispensable el uso de transacciones.
- Han empeorado la interoperabilidad y la testeabilidad en algunos aspectos, ya que para accesos fuera de la BTS se hace indispensable la creación de un *gateway* dentro de la misma que permita el acceso remoto.

5.1 EL PROBLEMA DE LA DETECCIÓN DE RADIOS REVISADO

El diagrama de secuencias incluido a continuación demuestra que el problema de la detección de radios no es un problema de arquitectura sino de diseño. Tal y como se aprecia, el problema persiste tras la eliminación de las múltiples vistas, mediante la introducción de la nueva arquitectura.

53 Puede obtenerse más información del problema de las lecturas sucias (d*irty read*) y del aislamiento en general mediante la consulta del estándar SQL 92 [7].

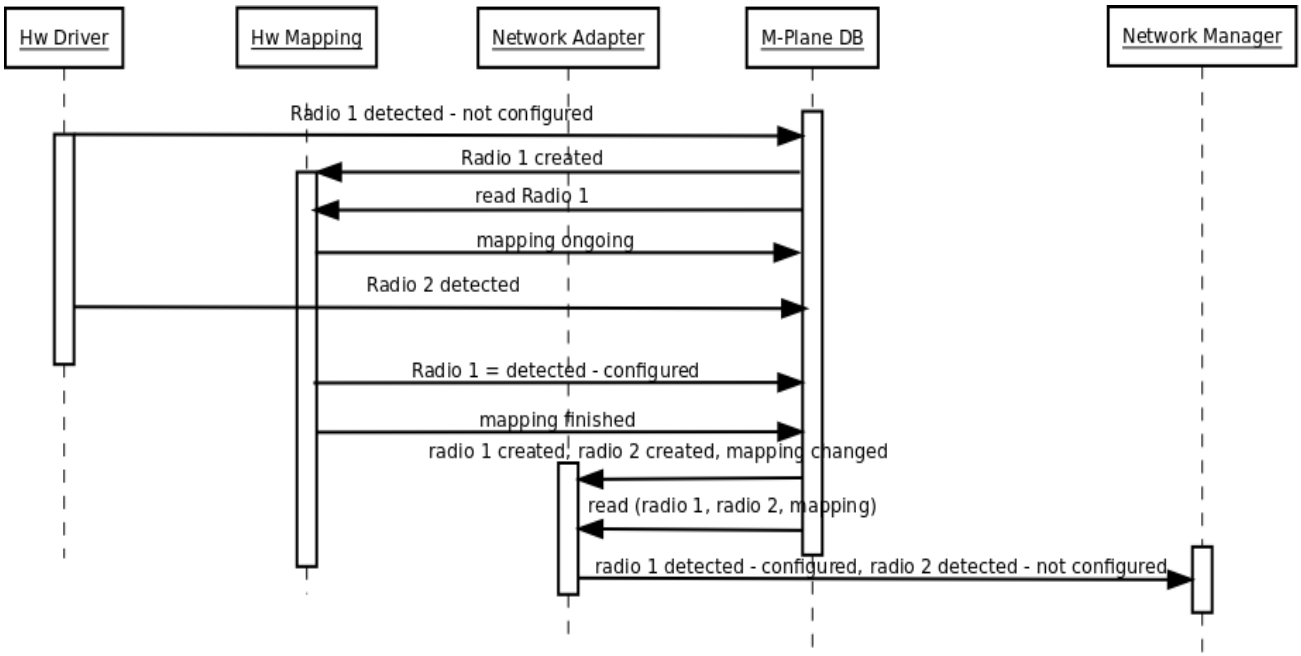


Ilustración 25: Revisión del problema de la detección de radios (Fuente: elaboración propia).

La secuencia ahora es la siguiente:

- Hw Driver detecta la radio 1 y la introduce en la base de datos.
- DB Notifier notifica a Hw Mapping que la radio 1 ha sido creada.
- Este lee el objeto de la base de datos, cambia su estado y se dispone a procesarla.
- Hw Driver detecta la radio 2 y la introduce en la base de datos.
- Hw Mapping cambia el estado de la radio 2 e indica que ha terminado.
- Network Adapter recibe la notificación de que la radio 1 y la radio 2 han sido creadas y de que el objeto mapping ha cambiado.
- Network Adapter lee los tres objetos de la base de datos y envía la notificación errónea a Network Manager.

La explicación de que la condición de carrera persista es que sigue sin cambiar la clave del problema. Es decir, continúa el problema de que que Network Adapter no sepa que elementos han sido procesados por Hw Mapping durante una determinada transacción.

Una posible solución a este problema de diseño sería el añadir un número de versión global del hardware a todos los elementos hardware. Por ejemplo, cuando la radio 1 se detecta, se le asigna la versión de hardware 1. Cuando se detecta la segunda radio se le asigna la versión 2. Posteriormente Hw Mapping indicará cuando termine de mapear que la última versión de hardware que ha visto es 1. De esta forma Network Adapter sabrá que puede notificar la primera radio 1 y que deberá esperar a que HwMapping pase a la versión 2 para enviar la segunda.

6. ANÁLISIS Y EVALUACIÓN EN ARCHE

A lo largo de esta sección se realiza un análisis del problema partiendo del modelo de negocio y sus casos de uso, a fin de obtener los datos necesarios para la posterior evaluación de ambas arquitecturas en la herramienta ArchE.

6.1 DESCRIPCIÓN DEL SISTEMA

A continuación se incluye la descripción del sistema que se va a evaluar en ArchE.

La BTS es un elemento hardware de una red de operación de telefonía móvil que se ocupa de comunicarse directamente con los UE y conectarlos al resto de la red. A nivel lógico, cada BTS atiende varias celdas, cada una con unos recursos hardware asignados, número de antenas, radios, etc., o parámetros de configuración, número de UE soportados, etc.

Cada BTS, tal y como se obtiene de fábrica, posee una configuración básica que le informa de su configuración hardware y de ciertos parámetros estáticos como los parámetros de comunicación de los diversos modelos de radios soportados. Sin embargo, la BTS no posee información de qué elementos hardware van a conectarse a ella y qué función lógica realizarán dentro de cada celda. Tras arrancar la BTS está se encontrará en estado no-configurado. Este estado implica que se puede comunicar con la BTS mediante una herramienta de gestión, pero esta es incapaz de utilizarse dentro de la red para establecer celdas y comunicarse con los UE.

Para salir del estado anterior, el operador envía el fichero de configuración basado en XML a la BTS con la información de los elementos que planea conectar y con la configuración lógica de la red. Tras enviar la configuración, la BTS pasará al estado configurado. En dicho estado la BTS tiene la información suficiente para comunicarse con la red de telefonía pero sigue sin poder comunicar con los UE porque no tiene ningún equipo radio conectado. Dichos equipos se notificarán a partir de este momento a la herramienta de gestión con el estado configurado-no detectado.

Seguidamente el operador conecta el equipo de radio a la BTS y esta lo autodetecta. Si el equipo se identifica como aquel que había sido configurado previamente, su configuración lógica estará disponible y la BTS podrá establecer la celda o celdas asociadas a dicho equipo. Una vez establecidas las celdas, la BTS ya podrá comunicarse con los UE. En este momento se dice que la BTS está en estado “en línea”[54].

Además, tras la conexión de los equipos de radio, la BTS actualizará su estado a la herramienta de gestión a través de la aplicación NetworkManager y le informará del nuevo hardware que se encontrará en estado configurado-detectado.

Periódicamente, el hardware conectado a la BTS informará de sus cambios de estado y esta a su vez lo comunicará en tiempo real a las herramientas de gestión.

Además, la BTS monitorizará continuamente el estado del hardware y de sus propios elementos para detectar cualquier fallo que pudiera ocurrir. En caso de fallo se generará una alarma que se almacenará en la BTS y se comunicará a la herramienta de gestión.

54 En realidad el término técnico es *onAir* pero se utiliza esta traducción por ser más clara en español.

Posteriormente, si el elemento que estaba en fallo se recupera, la BTS borrará la alarma correspondiente de su base de datos y notificará el borrado a la herramienta de gestión.

Por último, la BTS se comunicará continuamente con sus estaciones vecinas y obtendrá información de la configuración de estas, replicándola posteriormente a las herramientas de gestión a través de la aplicación Network Manager.

6.2 DIAGRAMA DE CASOS DE USO

En la descripción anterior se identifican seis casos de uso especialmente relevantes para la capa M-Plane de la BTS que se evaluarán posteriormente. Dichos casos son:

- **Enviar la configuración a la BTS**. Incluye el arranque de la BTS, el establecimiento de la comunicación con esta a través de la herramienta de gestión y el envío de la configuración lógica a la misma.
- **Conectar el hardware**. Una vez configurada, se conecta a la BTS el hardware, que es autodetectado y puesto en servicio. Además, la BTS comunica la detección a la herramienta de gestión.
- **Actualizar el hardware**. El hardware conectado notifica periódicamente el cambio de sus estados en tiempo real; dichos cambios son replicados por la BTS a la herramienta de gestión.
- **Notificar las alarmas**. En caso de fallo del hardware, la BTS lo detecta y genera la alarma correspondiente, que es replicada a la herramienta de gestión.
- **Borrar las alarmas**. Una vez recuperado el hardware, la BTS borra la alarma previamente generada y replica dicho borrado a la herramienta de gestión.
- **Actualizar la configuración de red**. Cada vez que se detecta un cambio en la configuración de una BTS vecina se almacena dicho cambio y se replica a la herramienta de gestión.

A continuación se incluye el diagrama para los casos de uso expuestos anteriormente. Dentro del mismo, con el objeto de aumentar la legibilidad, se han utilizado los siguientes colores, concordando con los diagramas de arquitectura expuestos en los capítulos anteriores:

- **Amarillo**, para los actores y el primer nivel de descomposición de las funcionalidades.
- **Azul oscuro**, para las funcionalidades relacionadas con la M-Plane DB.
- **Azul claro**, para las funcionalidades relacionadas con la adaptación entre la BTS y el Network Adapter.
- **Rojo**, para las funcionalidades relacionadas con la configuración de los elementos de red vecinos de la BTS.

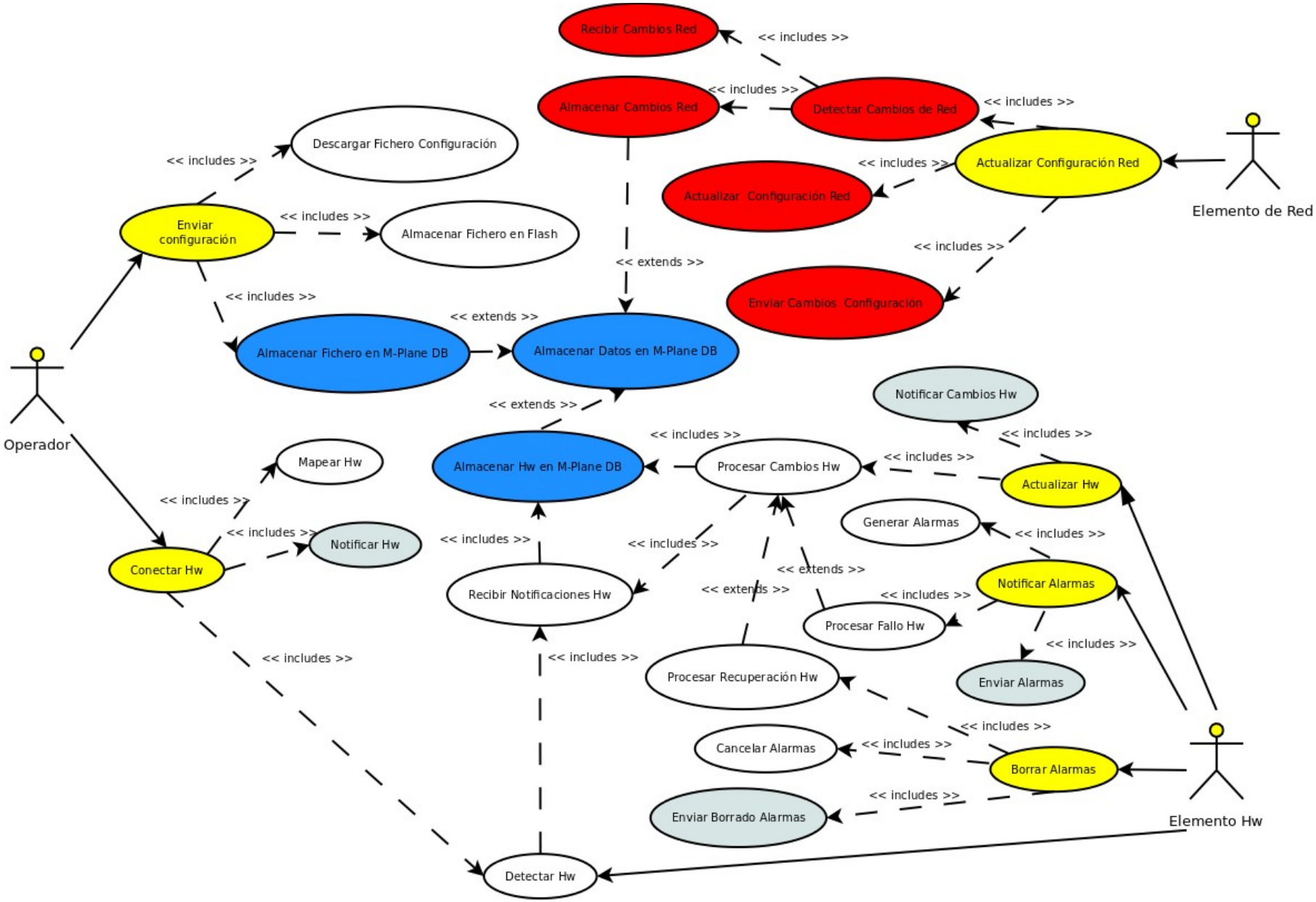


Ilustración 26: Diagrama de casos de uso de la capa M-PLANE (Fuente: elaboración propia).

6.3 IDENTIFICACIÓN DE FUNCIONALIDADES

Identificador	*Nombre*	*Descripción*
Oper_EnviarConf	**Enviar Configuración**	Permite enviar una configuración a la BTS
Oper_EnviarConf_DescFich	**Descargar Fichero Configuración**	Descarga el fichero desde el Network Manager a través de una conexión FTP
Oper_EnviarConf_AlmFichFlash	**Almacenar Fichero en Flash**	Almacena el fichero descargado en la memoria flash
Oper_EnviarConf_AlmFichDB	**Almacenar Fichero en M-Plane DB**	Parsea el fichero y lo almacena en la base de datos M-Plane
Comun_AlmFichDB	**Almacenar Datos en M-Plane DB**	Almacena datos en la base de datos M-Plane
Oper_ConecHw	**Conectar Hw**	Permite conectar un nuevo hardware a la BTS y visualizarlo en Network Manager como configurado o no dependiendo de si se había enviado la configuración del mismo a la BTS
Oper_ConecHw_DetHw	**Detectar Hw**	Detecta el hardware recién conectado y lo almacena en la M-Plane DB
Comun_RecNotHw	**Recibir Notificaciones Hw**	Recibe información de los elementos hardware conectados a la BTS.
Comun_AlmacHwDB	**Almacenar Hw en M-Plane DB**	Almacena los datos de hardware en la M-Plane DB
Oper_ConecHw_MapHw	**Mapear Hw**	Mapea el hardware detectado con el definido en la configuración existente
Oper_ConecHw_NotHw	**Notificar Hw**	Notifica al Network Manager el hardware detectado
Hw_ActHw	**Actualizar Hw**	Envía actualizaciones con el estado del hardware al Network Manager
Hw_ActHw_ProcCambHw	**Procesar Cambios Hw**	Recibe las notificaciones con la información del hardware y las almacena en la M-Plane DB
Hw_ActHw_NotCambHw	**Notificar Cambios Hw**	Notifica los cambios con el estado del hardware al Network Manager
Hw_NotAlarm	**Notificar Alarmas**	Notifica las alarmas detectadas en la BTS al Network Manager

Hw_NotAlarm_ProcFallo	**Procesar Fallo Hw**	Recibe las notificaciones con la información de los fallos hardware y las almacena en la M-Plane DB
Hw_NotAlarm_GenAlarm	**Generar Alarmas**	Analiza el estado del hardware y genera la alarma correspondiente en caso de fallo.
Hw_NotAlarm_EnvAlarm	**Enviar Alarmas**	Envía las alarmas generadas al Network Manager
Hw_BorrAlarm	**Borrar Alarmas**	Notifica las cancelaciones de alarmas al Network Manager
Hw_BorrAlarm_ProcRecHw	**Procesar Recuperación Hw**	Recibe las notificaciones con la información de la recuperación del hardware y las almacena en la M-Plane DB
Hw_BorrAlarm_CancAlarm	**Cancelar Alarmas**	Detecta la recuperación del hardware y cancela las alarmas asociadas en el sistema
Hw_BorrAlarm_EnvBorrAlarm	**Enviar Borrado Alarmas**	Notifica el borrado de las alarmas al Network Manager
ElemRed_ActConf	**Actualizar Configuración Red**	Detecta los cambios de configuración en los elementos de red vecinos a la BTS, actualiza la configuración en la memoria flash de la BTS y notifica dichos cambios al Network Manager
ElemRed_ActConf_DetCamb	**Detectar Cambios de Red**	Detecta los cambios de configuración en los elementos de red y los almacena en la M-Plane DB
ElemRed_ActConf_DetCamb_RecCamb	**Recibir Cambios Red**	Recibe los cambios de configuración de los elementos de red vecinos a la BTS a través de la capa C-Plane
ElemRed_ActConf_DetCamb_AlmCamb	**Almacenar Cambios Red**	Almacena los cambios de configuración de los elementos de red en la M-Plane DB
ElemRed_ActConf_AlmConfFlash	**Actualizar Configuración**	Actualiza el fichero de configuración de la memoria flash
ElemRed_ActConf_EnvConf	**Enviar Cambios Configuración**	Envía los cambios de configuración en forma de fichero delta[55] al Network Manager.

Tabla 3: Listado de funcionalidades a evaluar para la capa M-Plane.

55 Este fichero tiene el mismo formato que el fichero de configuración, pero contiene sólo la información de los elementos que han cambiado con respecto a la última actualización por lo que recibe el nombre de “delta”.

6.4 CASOS DE USO

Una vez identificadas las funcionalidades se analizan por separado cada uno de los casos de uso.

6.4.1 CASO DE USO: ENVIAR CONFIGURACIÓN

Caso de uso	*Enviar Configuración*
Actor	Operador
Pre-condiciones	La BTS está arrancada y comunicando con el Network Manager.
Post-condiciones	La BTS se ha configurado correctamente, el fichero de configuración ha sido almacenado en la memoria flash, y la configuración está disponible en la M-Plane DB en un tiempo no superior a 10 segundos.
Descripción y flujos de información	
Paso 1: El operador solicita el envío de la configuración de la BTS, que está disponible para que esta se lo descargue a través de un servidor FTP. La solicitud incluye los parámetros de conexión al servidor y el fichero a descargar. Paso 2: La BTS descarga y almacena el fichero de configuración en la memoria flash. Paso 3: La BTS parsea el fichero de configuración y lo almacena en la M-Plane DB.	

Tabla 4: Refinamiento del caso de uso: Enviar Configuración.

6.4.2 CASO DE USO: CONECTAR HW

Caso de uso	*Conectar Hw*
Actor	Operador
Pre-condiciones	La BTS está arrancada y comunicando con el Network Manager. La BTS se ha configurado correctamente.
Post-condiciones	El hardware conectado es visible en Network Manager en un tiempo no superior a 2 segundos.
Descripción y flujos de información	
Paso 1: El operador conecta un nuevo equipo de radio a la BTS. Paso 2: La BTS detecta el hardware e introduce su información en la M-Plane DB. Paso 3: La BTS comprueba si el hardware ha sido previamente configurado o no y almacena dicha información en la M-Plane DB. Paso 4: La BTS notifica el nuevo hardware al Network Manager.	

Tabla 5: Refinamiento del caso de uso: Conectar Hw.

6.4.3 CASO DE USO: ACTUALIZAR HW

Caso de uso	*Actualizar Hw*
Actor	Elemento hardware
Pre-condiciones	La BTS está arrancada y comunicando con el Network Manager. La BTS se ha configurado correctamente. Al menos un elemento hardware ha sido detectado y se encuentra comunicando con la BTS.
Post-condiciones	Los cambios en el elemento hardware son visibles en el Network Manager en un tiempo no superior a 1 segundo.
Descripción y flujos de información	
Paso 1: El elemento hardware envía una notificación a la BTS informando de un cambio en su estado. Paso 2: La BTS almacena los datos de estado en la M-Plane DB. Paso 3: La BTS notifica los cambios de estado al Network Manager. Paso 4: La BTS notifica el nuevo hardware al Network Manager.	

Tabla 6: Refinamiento del caso de uso: Actualizar Hw

6.4.4 CASO DE USO: NOTIFICAR ALARMAS

Caso de uso	*Notificar Alarmas*
Actor	Elemento hardware
Pre-condiciones	La BTS está arrancada y comunicando con el Network Manager. La BTS se ha configurado correctamente. Se ha detectado al menos un elemento hardware que se encontraba comunicando con la BTS cuando sufre un fallo.
Post-condiciones	Se ha generado una alarma de fallo del elemento hardware. La alarma ha sido almacenada en la M-Plane DB y notificada al Network Manager en un tiempo no superior a 2 segundos desde que se produjo el fallo en el elemento.
Descripción y flujos de información	
Paso 1: Un elemento hardware envía una notificación a la BTS informando de un fallo en su funcionamiento, o bien la BTS detecta que no es posible la comunicación con un elemento hardware. Paso 2: La BTS genera una alarma y la almacena en la M-Plane DB. Paso 3: La BTS notifica la alarma al Network Manager.	

Tabla 7: Refinamiento del caso de uso: Notificar Alarmas.

6.4.5 CASO DE USO: BORRAR ALARMAS

Caso de uso	*Borrar Alarmas*
Actor	Elemento hardware
Pre-condiciones	La BTS está arrancada y comunicando con el Network Manager. La BTS se ha configurado correctamente. Se ha detectado al menos un elemento hardware. El elemento hardware se encontraba en estado de fallo y se ha generado la alarma correspondiente. El elemento se recupera.
Post-condiciones	La alarma del elemento hardware ha sido borrada de M-Plane DB y notificada al Network Manager en un tiempo no superior a 2 segundos desde que se recuperó el elemento.
Descripción y flujos de información	
Paso 1: Un elemento hardware previamente en fallo envía una notificación a la BTS informando de que su funcionamiento es correcto, o bien la BTS detecta que la comunicación con un elemento hardware es correcta, al igual que su estado. Paso 2: La BTS borra la alarma previamente generada de la M-Plane DB. Paso 3: La BTS notifica el borrado de la alarma al Network Manager.	

Tabla 8: Refinamiento del caso de uso: Borrar Alarmas.

6.4.6 CASO DE USO: ACTUALIZAR CONFIGURACIÓN RED

Caso de uso	*Actualizar Configuración Red*
Actor	Elemento de Red
Pre-condiciones	La BTS está arrancada y comunicando con el Network Manager. La BTS se ha configurado correctamente. La conexión X1 está funcionando correctamente. Un elemento de red vecino a la BTS informa de un cambio en su configuración.
Post-condiciones	El cambio de configuración del elemento de red es almacenado en la M-Plane DB. El fichero de configuración ha sido actualizado con el cambio. Network Manager ha sido informado del cambio mediante un fichero delta con los cambios de configuración en un tiempo no superior a 2 segundos.
Descripción y flujos de información	
Paso 1: La BTS recibe información acerca de un cambio de configuración en un elemento de red vecino a ella. Paso 2: La BTS actualiza la información de configuración en la M-Plane DB. Paso 3: La BTS actualiza el fichero de configuración en la memoria flash.	

Paso 4: La BTS envía un fichero delta con los cambios de configuración al Network Manager.

Tabla 9: Refinamiento del caso de uso: Actualizar Configuración de Red.

6.5 DEFINICIÓN DE ESCENARIOS

A partir de los casos de uso se refinan los escenarios correspondientes. Todos los escenarios se refieren al atributo de calidad de rendimiento.

6.5.1 ESCENARIO S1

El operador envía la configuración de la BTS. La BTS descarga y almacena el fichero de configuración en la memoria flash. Posteriormente parsea el fichero de configuración y lo almacena en la base de datos. La configuración debe ser almacenada en condiciones normales en un tiempo no superior a 10 segundos.

Escenario	*Parámetros*
Origen del estímulo	Operador
Estímulo	El operador envía una configuración a la BTS
Artefacto	BTS
Entorno	En condiciones normales
Respuesta	La configuración es almacenada en la BTS tanto en la memoria flash como en la M-Plane DB
Medida de la respuesta	No superior a los 10 segundos

Tabla 10: Parámetros del escenario S1.

6.5.2 ESCENARIO S2

El operador conecta un nuevo equipo de radio a la BTS. La BTS detecta el hardware, comprueba si es un hardware previamente configurado o no y notifica su estado al Network Manager en un tiempo no superior a 2 segundos.

Escenario	*Parámetros*
Origen del estímulo	Operador
Estímulo	El operador conecta un nuevo equipo de radio a la BTS
Artefacto	BTS
Entorno	En condiciones normales
Respuesta	La BTS actualiza la M-Plane DB y notifica al Network Manager el nuevo equipo conectado
Medida de la	No superior a los 2 segundos

respuesta	

Tabla 11: Parámetros del escenario S2.

6.5.3 ESCENARIO S3

Un elemento hardware envía información sobre sus variables de tiempo real, como la temperatura, potencia, etc. La BTS almacena los datos y los notifica al Network Manager en un tiempo no superior a 1 segundo.

Escenario	*Parámetros*
Origen del Estímulo	Elemento hardware
Estímulo	Un elemento hardware envía información sobre sus variables de tiempo real
Artefacto	BTS
Entorno	En condiciones normales
Respuesta	La BTS actualiza la M-Plane DB y notifica al Network Manager las variables recibidas
Medida de la respuesta	No superior a 1 segundo

Tabla 12: Parámetros del escenario S3.

6.5.4 ESCENARIO S4

Se produce un fallo en un elemento hardware. El fallo es detectado por la BTS, que genera la alarma correspondiente. La alarma es notificada en un tiempo no superior a 2 segundos desde que ocurrió el fallo del equipo.

Escenario	*Parámetros*
Origen del estímulo	Elemento hardware
Estímulo	Se produce un fallo en un elemento hardware
Artefacto	BTS
Entorno	En condiciones normales
Respuesta	La BTS genera una alarma y la comunica al Network Manager
Medida de la respuesta	No superior a los 2 segundos

Tabla 13: Parámetros del escenario S4.

6.5.5 ESCENARIO S5

El elemento hardware previamente en fallo se recupera. La recuperación es detectada por la BTS, y

esta cancela la alarma correspondiente. La cancelación de la alarma es notificada en un tiempo no superior a 2 segundos desde que se produjo la recuperación del elemento.

Escenario	*Parámetros*
Origen del estímulo	Elemento hardware
Estímulo	Se recupera un elemento hardware previamente en estado de fallo
Artefacto	BTS
Entorno	En condiciones normales
Respuesta	La BTS cancela la alarma previamente generada y la comunica al Network Manager
Medida de la respuesta	No superior a los 2 segundos

Tabla 14: Parámetros del escenario S5.

6.5.6 ESCENARIO S6

Se detecta un cambio de configuración en un elemento de la red vecino a la BTS. La BTS recibe el cambio a través del enlace X1 y lo almacena en el fichero de configuración de la memoria flash. Seguidamente almacena los cambios en la memoria y genera un fichero con el cambio de configuración que es enviado al Network Manager. El cambio debe ser visible en Network Manager en un tiempo no superior a 2 segundos.

Escenario	*Parámetros*
Origen del estímulo	Elemento de red
Estímulo	Se produce un cambio de configuración en un elemento de red vecino
Artefacto	BTS
Entorno	En condiciones normales
Respuesta	La BTS actualiza su configuración tanto en la memoria flash como en la M-Plane DB y envía el cambio de configuración al Network Manager
Medida de la respuesta	No superior a los 2 segundos

Tabla 15: Parámetros del escenario S6.

6.6 EVALUACIÓN DE LA ARQUITECTURA ORIGINAL EN ARCHE

Hasta este punto no se ha mencionado ningún detalle acerca de la arquitectura que se va a utilizar, ya que los escenarios son parte de la especificación de requisitos y, como tales, es la arquitectura la que guiará la forma de cumplirlos. Es decir, los requisitos guían la arquitectura y no al revés[56].

56 En realidad los requisitos deben adaptarse a las posibilidades de una arquitectura viable, por motivos obvios como las limitaciones del hardware, la complejidad de los algoritmos, etc.

La introducción de la arquitectura se realizará ahora mediante el mapeo entre funcionalidades y responsabilidades, y durante el cálculo de sus costes asociados. Además será necesario añadir aquellas responsabilidades que sean específicas de la arquitectura.

6.7 MAPEO FUNCIONALIDAD - RESPONSABILIDAD

Funcionalidad	*Id Resp*	*Responsabilidad*	*Coste*
Oper_EnviarConf_DescFich	**R1.1**	Descargar Fichero de Configuración	160 ms[57]
Oper_EnviarConf_AlmFichFlash	**R1.2**	Almacenar Fichero en Flash	200 ms[58]
Comun_AlmFichDB	**R1.3.1**	Almacenar Datos configuración en M-Plane DB	60 ms[59]
-	**R1.3.2**	Replicar Datos Configuración	300 ms[60]
Comun_RecNotHw	**R2.1.1**	Recibir Notificaciones Hw	10 ms
Comun_AlmacHwDB	**R2.1.2**	Almacenar Datos Hw en M-Plane DB	1 ms[61]
-	**R2.1.3**	Replicar Datos Hw	5 ms
Oper_ConecHw_MapHw	**R2.2**	Mapear Hw	1 ms
-	**R2.2.1**	Replicar Mapeo Hw	5 ms
Oper_ConecHw_NotHw	**R2.3**	Notificar Hw	1 ms
Hw_ActHw_ProcCambHw	**R3.1**	Procesar Cambios Hw	1 ms
Hw_ActHw_NotCambHw	**R3.2**	Notificar Cambios Hw	1 ms
Hw_NotAlarm_GenAlarm	**R4.2**	Generar Alarmas	10 ms
-	**R4.2.1**	Replicar Datos Alarmas	5 ms
Hw_NotAlarm_EnvAlarm	**R4.3**	Enviar Alarmas	1 ms
Hw_BorrAlarm_Canc	**R5.2**	Cancelar alarmas	10 ms

57 Se supone un fichero de 2 MB descargado a través de una red de 100 Mb/s.
58 Se supone la escritura del fichero a una tasa de 10 MB/s.
59 Se supone la escritura de 100 000 objetos con un tiempo de acceso de 60 ns. Para simplificar se considera la BD como una tabla *hash* por lo que el acceso a un objeto es directo a través del calculo de la *hash* del identificador del mismo.
60 Se multiplica el tiempo de almacenamiento por los 5 componentes a replicar. Se desestima el coste del envío de los mensajes, al poder ser en su mínima expresión una copia en memoria.
61 Posiblemente menor al milisegundo pero se toma este como magnitud mínima.

Alarm			
Hw_BorrAlarm_EnvBorrAlarm	**R5.3**	Enviar Borrado Alarmas	1 ms
ElemRed_ActConf_DetCamb_RecCamb	**R6.1.1**	Recibir Cambios Red	80 ms[62]
ElemRed_ActConf_DetCamb_AlmCamb	**R6.1.2**	Almacenar Cambios Red	6 ms
-	**R6.1.3**	Replicar Cambios Red	30 ms
ElemRed_ActConf_AlmConfFlash	**R6.2**	Actualizar Configuración	200 ms[63]
ElemRed_ActConf_EnvConf	**R6.3**	Enviar Cambios Configuración	10 ms

Tabla 16: Mapeo entre funcionalidades y responsabilidades para la arquitectura original.

Tal y como se aprecia en la tabla anterior, fruto del mapeo de la arquitectura original, han aparecido las responsabilidades de replicar los datos en los componentes del sistema. Además, se han omitido las funcionalidades que son una ligera variación de otras, como *Procesar Fallos Hw* y *Procesar Cambios Hw*.

6.8 MAPEO ENTRE ESCENARIOS Y RESPONSABILIDADES

La siguiente tabla indica qué responsabilidades se ven implicadas en un determinado escenario.

Id Resp	*Nombre*	*S1: Enviar Configuración*	*S2: Conectar Hw*	*S3:Actualizar Hw*	*S4: Notificar Alarmas*	*S5: Borrar Alarmas*	*S6: Actualizar Configuración de Red*
R1.1	Descargar Fichero de Configuración	**X**					
R1.2	Almacenar Fichero en Flash	**X**					
R1.3.1	Almacenar Datos configuración en M-Plane DB	**X**					

62 Se supone un 1 MB de datos a través de una red 100 Mb/s.
63 Actualizar la configuración es tan costoso como reescribir el fichero de nuevo.

R1.3.2	Replicar Datos Configuración	**X**					
R2.1.1	Recibir Notificaciones Hw.		**X**				
R2.1.2	Almacenar Datos Hw en M-Plane DB		**X**	**X**	**X**	**X**	
R2.1.3	Replicar Datos Hw		**X**	**X**	**X**	**X**	
R2.2	Mapear Hw		**X**				
R2.2.1	Replicar Mapeo Hw		**X**				
R2.3	Notificar Hw		**X**				
R3.1	Procesar Cambios Hw			**X**	**X**	**X**	
R3.2	Notificar Cambios Hw			**X**			
R4.2	Generar Alarmas				**X**		
R4.2.1	Replicar Datos Alarmas				**X**	**X**	
R4.3	Enviar Alarmas				**X**		
R5.2	Cancelar alarmas					**X**	
R5.3	Enviar Borrado Alarmas					**X**	
R6.1.1	Recibir Cambios Red						**X**
R6.1.2	Almacenar Cambios Red						**X**
R6.1.3	Replicar Cambios Red						**X**
R6.2	Actualizar Configuración						**X**
R6.3	Enviar Cambios Configuración						**X**

Tabla 17: Mapeo entre escenarios y responsabilidades.

6.9 RELACIONES ENTRE RESPONSABILIDADES

El último paso de refinamiento consiste en determinar las relaciones de dependencia entre las responsabilidades, lo cual se realiza a continuación.

Id Resp	*Responsabilidad padre*	*Id Resp*	*Responsabilidad hijo*
R1.1	Descargar Fichero de Configuración	**R1.2**	Almacenar Fichero en Flash
R1.1	Descargar Fichero de Configuración	**R1.3.1**	Almacenar Datos configuración en M-Plane DB
R1.3.1	Almacenar Datos configuración en M-Plane DB	**R1.3.2**	Replicar Datos Configuración
R2.1.1	Recibir Notificaciones Hw	**R2.1.2**	Almacenar Datos Hw en M-Plane DB
R2.1.2	Almacenar Datos Hw en M-Plane DB	**R2.1.3**	Replicar Datos Hw

R2.1.3	Replicar Datos Hw	**R2.2**	Mapear Hw
R2.2	Mapear Hw	**R2.2.1**	Replicar Mapeo Hw
R2.2.1	Replicar Mapeo Hw	**R2.3**	Notificar Hw
R3.1	Procesar Cambios Hw	**R2.1.2**	Almacenar Datos Hw en M-Plane DB
R2.1.3	Replicar Datos Hw	**R3.2**	Notificar Cambios Hw
R2.1.3	Replicar Datos Hw	**R4.2**	Generar Alarmas
R4.2	Generar Alarmas	**R4.2.1**	Replicar Datos Alarmas
R4.2.1	Replicar Datos Alarmas	**R4.3**	Enviar Alarmas
R2.1.3	Replicar Datos Hw	**R5.2**	Cancelar alarmas
R5.2	Cancelar alarmas	**R4.2.1**	Replicar Datos Alarmas
R4.2.1	Replicar Datos Alarmas	**R5.3**	Enviar Borrado Alarmas
R6.1.1	Recibir Cambios Red	**R6.1.2**	Almacenar Cambios Red
R6.1.2	Almacenar Cambios Red	**R6.1.3**	Replicar Cambios Red
R6.1.3	Replicar Cambios Red	**R6.2**	Actualizar Configuración
R6.1.3	Replicar Cambios Red	**R6.3**	Enviar Cambios Configuración

Tabla 18: Relaciones entre responsabilidades.

6.10 INTRODUCCIÓN DE DATOS Y EVALUACIÓN EN ARCHE

Una vez refinados todos los datos el último paso es introducirlos en la herramienta ArchE. A continuación se incluyen las ilustraciones del resultado de dicha operación.

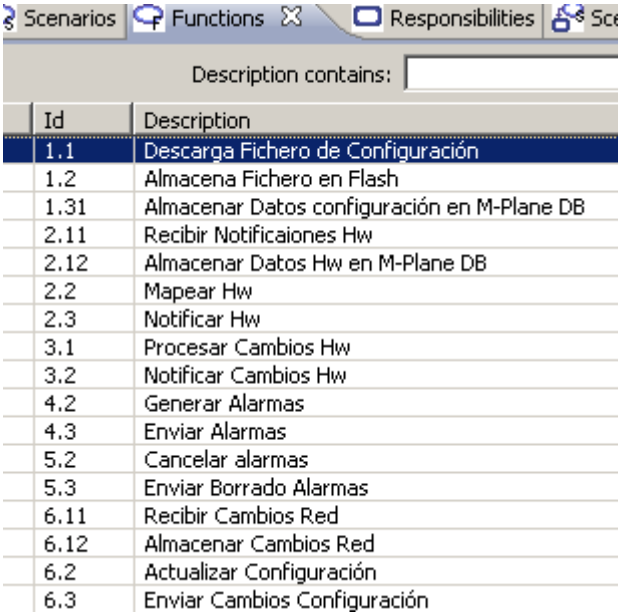
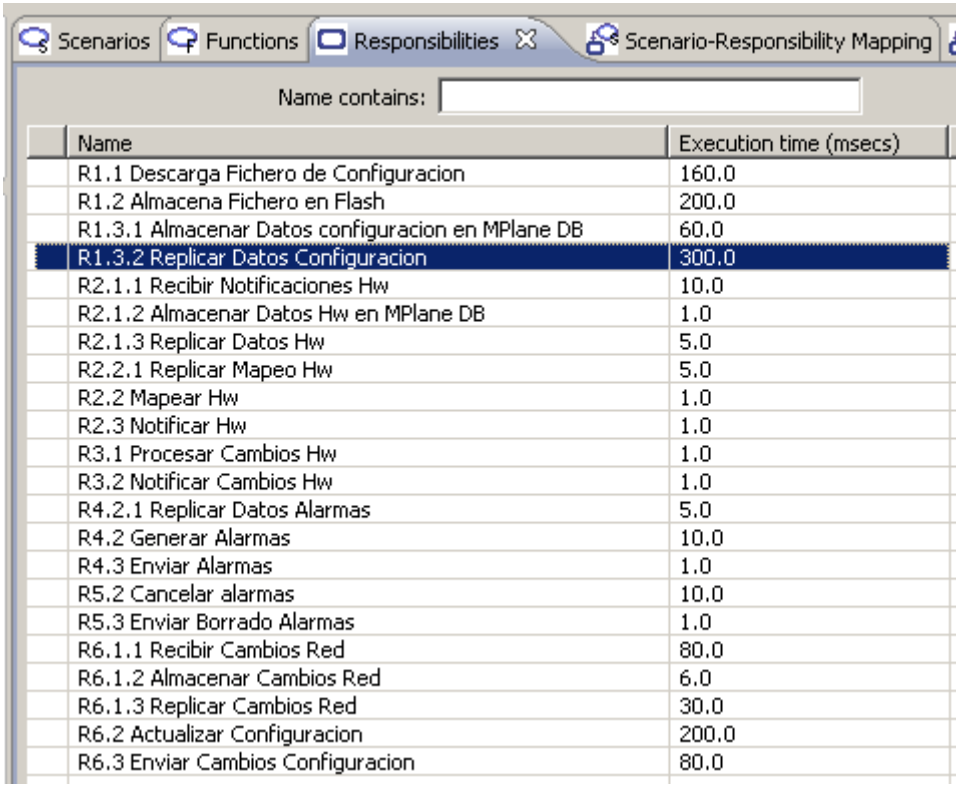


Ilustración 27: Inserción de funcionalidades.

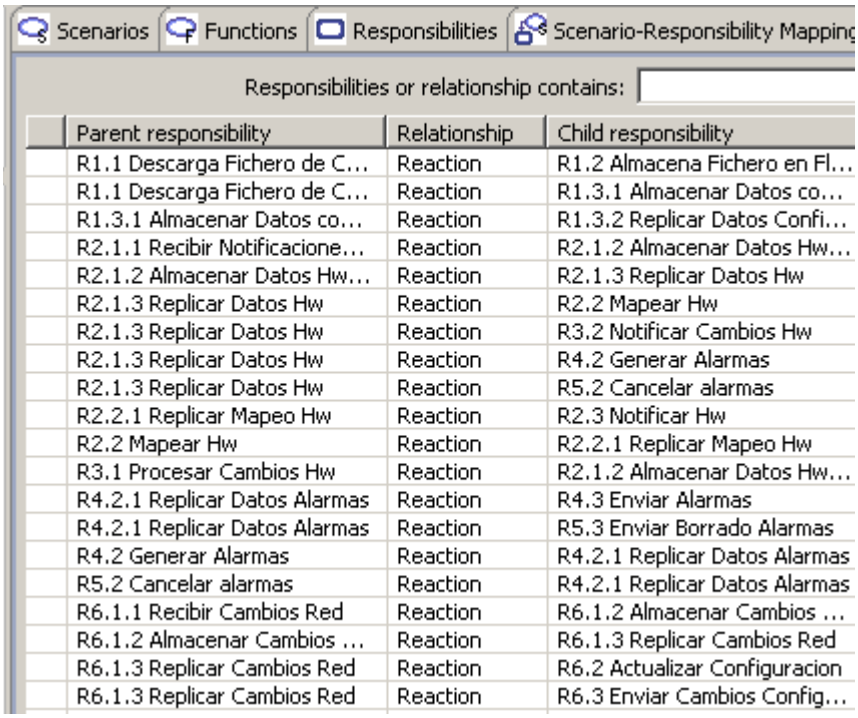
Function	Responsibility
Descarga Fichero de Configuracion	R1.1 Descarga Fichero de Configuracion
Almacena Fichero en Flash	R1.2 Almacena Fichero en Flash
Almacenar Datos configuracion en MPlane DB	R1.3.1 Almacenar Datos configuracion en MPlane DB
Recibir Notificaciones Hw	R2.1.1 Recibir Notificaciones Hw
Almacenar Datos Hw en MPlane DB	R2.1.2 Almacenar Datos Hw en MPlane DB
Mapear Hw	R2.2 Mapear Hw
Notificar Hw	R2.3 Notificar Hw
Procesar Cambios Hw	R3.1 Procesar Cambios Hw
Notificar Cambios Hw	R3.2 Notificar Cambios Hw
Generar Alarmas	R4.2 Generar Alarmas
Enviar Alarmas	R4.3 Enviar Alarmas
Cancelar alarmas	R5.2 Cancelar alarmas
Enviar Borrado Alarmas	R5.3 Enviar Borrado Alarmas
Recibir Cambios Red	R6.1.1 Recibir Cambios Red
Almacenar Cambios Red	R6.1.2 Almacenar Cambios Red
Actualizar Configuracion	R6.2 Actualizar Configuracion
Enviar Cambios Configuracion	R6.3 Enviar Cambios Configuracion

Ilustración 28: Mapeo entre funcionalidades y responsabilidades.



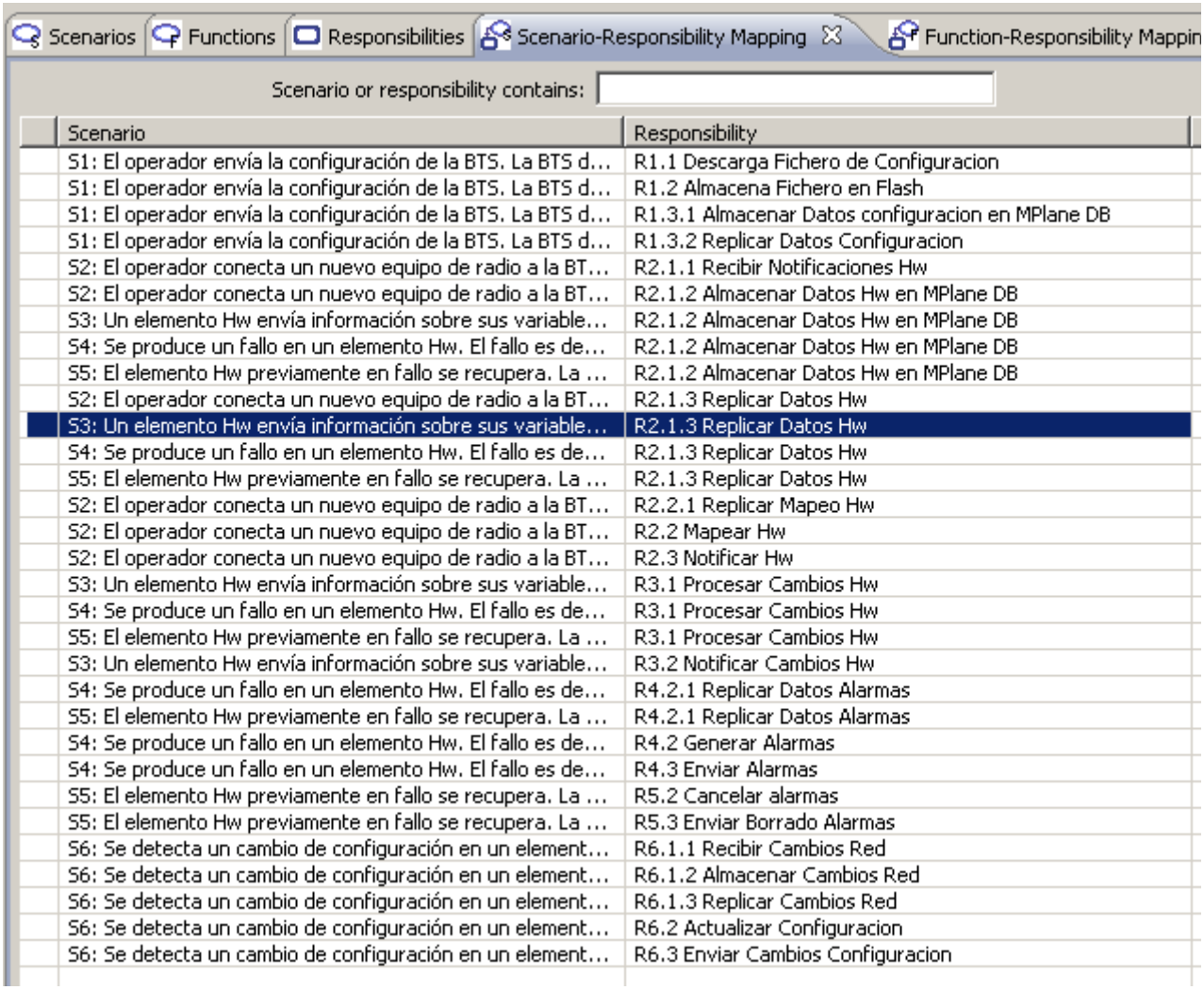
Name	Execution time (msecs)
R1.1 Descarga Fichero de Configuracion	160.0
R1.2 Almacena Fichero en Flash	200.0
R1.3.1 Almacenar Datos configuracion en MPlane DB	60.0
R1.3.2 Replicar Datos Configuracion	300.0
R2.1.1 Recibir Notificaciones Hw	10.0
R2.1.2 Almacenar Datos Hw en MPlane DB	1.0
R2.1.3 Replicar Datos Hw	5.0
R2.2.1 Replicar Mapeo Hw	5.0
R2.2 Mapear Hw	1.0
R2.3 Notificar Hw	1.0
R3.1 Procesar Cambios Hw	1.0
R3.2 Notificar Cambios Hw	1.0
R4.2.1 Replicar Datos Alarmas	5.0
R4.2 Generar Alarmas	10.0
R4.3 Enviar Alarmas	1.0
R5.2 Cancelar alarmas	10.0
R5.3 Enviar Borrado Alarmas	1.0
R6.1.1 Recibir Cambios Red	80.0
R6.1.2 Almacenar Cambios Red	6.0
R6.1.3 Replicar Cambios Red	30.0
R6.2 Actualizar Configuracion	200.0
R6.3 Enviar Cambios Configuracion	80.0

Ilustración 29: Introducción de responsabilidades adicionales y asignación de costes.



Parent responsibility	Relationship	Child responsibility
R1.1 Descarga Fichero de C...	Reaction	R1.2 Almacena Fichero en Fl...
R1.1 Descarga Fichero de C...	Reaction	R1.3.1 Almacenar Datos co...
R1.3.1 Almacenar Datos co...	Reaction	R1.3.2 Replicar Datos Confi...
R2.1.1 Recibir Notificacione...	Reaction	R2.1.2 Almacenar Datos Hw...
R2.1.2 Almacenar Datos Hw...	Reaction	R2.1.3 Replicar Datos Hw
R2.1.3 Replicar Datos Hw	Reaction	R2.2 Mapear Hw
R2.1.3 Replicar Datos Hw	Reaction	R3.2 Notificar Cambios Hw
R2.1.3 Replicar Datos Hw	Reaction	R4.2 Generar Alarmas
R2.1.3 Replicar Datos Hw	Reaction	R5.2 Cancelar alarmas
R2.2.1 Replicar Mapeo Hw	Reaction	R2.3 Notificar Hw
R2.2 Mapear Hw	Reaction	R2.2.1 Replicar Mapeo Hw
R3.1 Procesar Cambios Hw	Reaction	R2.1.2 Almacenar Datos Hw...
R4.2.1 Replicar Datos Alarmas	Reaction	R4.3 Enviar Alarmas
R4.2.1 Replicar Datos Alarmas	Reaction	R5.3 Enviar Borrado Alarmas
R4.2 Generar Alarmas	Reaction	R4.2.1 Replicar Datos Alarmas
R5.2 Cancelar alarmas	Reaction	R4.2.1 Replicar Datos Alarmas
R6.1.1 Recibir Cambios Red	Reaction	R6.1.2 Almacenar Cambios ...
R6.1.2 Almacenar Cambios ...	Reaction	R6.1.3 Replicar Cambios Red
R6.1.3 Replicar Cambios Red	Reaction	R6.2 Actualizar Configuracion
R6.1.3 Replicar Cambios Red	Reaction	R6.3 Enviar Cambios Config...

Ilustración 30: Relaciones entre responsabilidades.



Scenario	Responsibility
S1: El operador envía la configuración de la BTS. La BTS d...	R1.1 Descarga Fichero de Configuracion
S1: El operador envía la configuración de la BTS. La BTS d...	R1.2 Almacena Fichero en Flash
S1: El operador envía la configuración de la BTS. La BTS d...	R1.3.1 Almacenar Datos configuracion en MPlane DB
S1: El operador envía la configuración de la BTS. La BTS d...	R1.3.2 Replicar Datos Configuracion
S2: El operador conecta un nuevo equipo de radio a la BT...	R2.1.1 Recibir Notificaciones Hw
S2: El operador conecta un nuevo equipo de radio a la BT...	R2.1.2 Almacenar Datos Hw en MPlane DB
S3: Un elemento Hw envía información sobre sus variable...	R2.1.2 Almacenar Datos Hw en MPlane DB
S4: Se produce un fallo en un elemento Hw. El fallo es de...	R2.1.2 Almacenar Datos Hw en MPlane DB
S5: El elemento Hw previamente en fallo se recupera. La ...	R2.1.2 Almacenar Datos Hw en MPlane DB
S2: El operador conecta un nuevo equipo de radio a la BT...	R2.1.3 Replicar Datos Hw
S3: Un elemento Hw envía información sobre sus variable...	R2.1.3 Replicar Datos Hw
S4: Se produce un fallo en un elemento Hw. El fallo es de...	R2.1.3 Replicar Datos Hw
S5: El elemento Hw previamente en fallo se recupera. La ...	R2.1.3 Replicar Datos Hw
S2: El operador conecta un nuevo equipo de radio a la BT...	R2.2.1 Replicar Mapeo Hw
S2: El operador conecta un nuevo equipo de radio a la BT...	R2.2 Mapear Hw
S2: El operador conecta un nuevo equipo de radio a la BT...	R2.3 Notificar Hw
S3: Un elemento Hw envía información sobre sus variable...	R3.1 Procesar Cambios Hw
S4: Se produce un fallo en un elemento Hw. El fallo es de...	R3.1 Procesar Cambios Hw
S5: El elemento Hw previamente en fallo se recupera. La ...	R3.1 Procesar Cambios Hw
S3: Un elemento Hw envía información sobre sus variable...	R3.2 Notificar Cambios Hw
S4: Se produce un fallo en un elemento Hw. El fallo es de...	R4.2.1 Replicar Datos Alarmas
S5: El elemento Hw previamente en fallo se recupera. La ...	R4.2.1 Replicar Datos Alarmas
S4: Se produce un fallo en un elemento Hw. El fallo es de...	R4.2 Generar Alarmas
S4: Se produce un fallo en un elemento Hw. El fallo es de...	R4.3 Enviar Alarmas
S5: El elemento Hw previamente en fallo se recupera. La ...	R5.2 Cancelar alarmas
S5: El elemento Hw previamente en fallo se recupera. La ...	R5.3 Enviar Borrado Alarmas
S6: Se detecta un cambio de configuración en un element...	R6.1.1 Recibir Cambios Red
S6: Se detecta un cambio de configuración en un element...	R6.1.2 Almacenar Cambios Red
S6: Se detecta un cambio de configuración en un element...	R6.1.3 Replicar Cambios Red
S6: Se detecta un cambio de configuración en un element...	R6.2 Actualizar Configuracion
S6: Se detecta un cambio de configuración en un element...	R6.3 Enviar Cambios Configuracion

Ilustración 31: Mapeo entre escenarios y responsabilidades.

Una vez introducidos todos los datos se procede a la evaluación. Tal y como se aprecia en las ilustraciones siguientes, el sistema no es viable. El motivo es que el escenario 3, el envío de información de estado del hardware, no se satisface por ser el más restrictivo en cuanto al tiempo de ejecución. El tiempo total, en el peor de los casos, incluyendo el posible retraso por la ejecución del resto de escenarios, es de 1204 ms lo cual supera el segundo previsto.

La última ilustración muestra la tabla de tiempos de respuesta desglosado por tareas y escenarios, para el mejor y peor caso. El escenario fallido tiene un coste propio en su peor caso de 8 ms [64] y los 1196 ms restantes vienen dados por la eventual ejecución del resto de escenarios.

64 El coste en el mejor de los casos es una estimación del sistema basada en una distribución simple. De esta forma el coste medio es la mitad del proporcionado y el mejor es siempre 1. Debido a esto, dicho campo no debería tenerse en cuenta a la hora de analizar el rendimiento, ya que dicho valor no tiene ninguna relación con la realidad.

		Description	ScenarioType	Stimulus	StimulusType	Source	Artifact	Environment	Response	Measure	Value
●	▲	S6: Se detecta un cambio de configuración en un element...	ICM Perfor...	Se produce...	Periodic	Elemento d...	BTS	En condicio...	La BTS actu...	En segundos	2.0
●	▲	S5: El elemento Hw previamente en fallo se recupera. La ...	ICM Perfor...	Se recuper...	Periodic	Elemento Hw	BTS	En condicio...	La BTS can...	En segundos	2.0
●	▲	S4: Se produce un fallo en un elemento Hw. El fallo es de...	ICM Perfor...	Se produce...	Periodic	Elemento Hw	BTS	En condicio...	La BTS gen...	En segundos	2.0
●	▲	S2: El operador conecta un nuevo equipo de radio a la BT...	ICM Perfor...	El operador...	Periodic	Operador	BTS	En condicio...	La BTS actu...	En segundos	2.0
●	▲	S1: El operador envía la configuración de la BTS. La BTS d...	ICM Perfor...	El operador...	Periodic	Operador	BTS	En condicio...	La configur...	En segundos	10.0
●	▲	S3: Un elemento Hw envía información sobre sus variable...	ICM Perfor...	Un element...	Periodic	Elemento Hw	BTS	En condicio...	La BTS actu...	En segundos	1.0

The scenario is not satisfied.
value = 1204.0

Ilustración 32: Resultado de la ejecución de los escenarios.

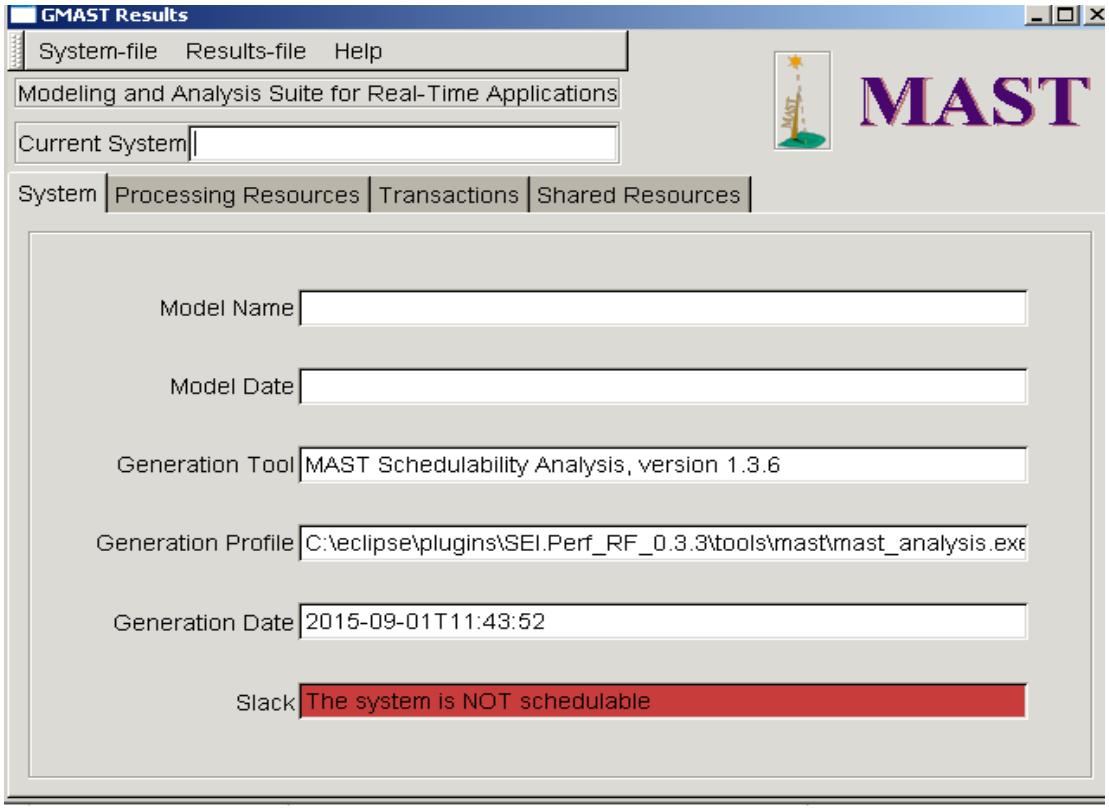


Ilustración 33: MAST informa de que el sistema no es viable[65].

65 La traducción exacta sería “planificable”.

o_1_5_done	service_fact_7.psource_service	5.00	1204.0
o_2_1_icmc_r2.1.1_recibir_notificaciones_hw.psink_service_fact_8	service_fact_8.psource_service	0.00	0.00
o_2_2_icmc_r2.1.2_almacenar_datos_hw_en_mplane_db.psink_r2.1.2_almacenar_datos_hw_en_mplane_db	service_fact_8.psource_service	2.00	1186.0
o_2_3_icmc_r2.1.3_replicar_datos_hw.psink_r2.1.3_replicar_datos_hw	service_fact_8.psource_service	0.00	0.00
o_2_4_icmc_r2.2_mapear_hw.psink_r2.2_mapear_hw	service_fact_8.psource_service	4.00	1197.0
o_2_5_icmc_r2.2.1_replicar_mapeo_hw.psink_r2.2.1_replicar_mapeo_hw	service_fact_8.psource_service	5.00	1198.0
o_2_6_icmc_r2.3_notificar_hw.psink_r2.3_notificar_hw	service_fact_8.psource_service	6.00	1203.0
o_2_7_done	service_fact_8.psource_service	7.00	1204.0
o_3_1_icmc_r3.1_procesar_cambios_hw.psink_service_fact_9	service_fact_9.psource_service	0.00	0.00
o_3_2_icmc_r2.1.2_almacenar_datos_hw_en_mplane_db.psink_r2.1.2_almacenar_datos_hw_en_mplane_db	service_fact_9.psource_service	2.00	1192.0
o_3_3_icmc_r2.1.3_replicar_datos_hw.psink_r2.1.3_replicar_datos_hw	service_fact_9.psource_service	0.00	0.00
o_3_4_icmc_r3.2_notificar_cambios_hw.psink_r3.2_notificar_cambios_hw	service_fact_9.psource_service	4.00	1203.0
o_3_5_done	service_fact_9.psource_service	5.00	1204.0
o_4_1_icmc_r3.1_procesar_cambios_hw.psink_service_fact_10	service_fact_10.psource_service	0.00	0.00
o_4_2_icmc_r2.1.2_almacenar_datos_hw_en_mplane_db.psink_r2.1.2_almacenar_datos_hw_en_mplane_db	service_fact_10.psource_service	2.00	1177.0
o_4_3_icmc_r2.1.3_replicar_datos_hw.psink_r2.1.3_replicar_datos_hw	service_fact_10.psource_service	0.00	0.00
o_4_4_icmc_r4.2_generar_alarmas.psink_r4.2_generar_alarmas	service_fact_10.psource_service	4.00	1188.0
o_4_5_icmc_r4.2.1_replicar_datos_alarmas.psink_r4.2.1_replicar_datos_alarmas	service_fact_10.psource_service	5.00	1198.0
o_4_6_icmc_r4.3_enviar_alarmas.psink_r4.3_enviar_alarmas	service_fact_10.psource_service	0.00	0.00
o_4_7_done	service_fact_10.psource_service	7.00	1204.0
o_5_1_icmc_r3.1_procesar_cambios_hw.psink_service_fact_11	service_fact_11.psource_service	0.00	0.00
o_5_2_icmc_r2.1.2_almacenar_datos_hw_en_mplane_db.psink_r2.1.2_almacenar_datos_hw_en_mplane_db	service_fact_11.psource_service	2.00	1177.0
o_5_3_icmc_r2.1.3_replicar_datos_hw.psink_r2.1.3_replicar_datos_hw	service_fact_11.psource_service	0.00	0.00
o_5_4_icmc_r5.2_cancelar_alarmas.psink_r5.2_cancelar_alarmas	service_fact_11.psource_service	4.00	1188.0
o_5_5_icmc_r4.2.1_replicar_datos_alarmas.psink_r4.2.1_replicar_datos_alarmas	service_fact_11.psource_service	5.00	1198.0
o_5_6_icmc_r5.3_enviar_borrado_alarmas.psink_r5.3_enviar_borrado_alarmas	service_fact_11.psource_service	6.00	1203.0
o_5_7_done	service_fact_11.psource_service	7.00	1204.0
o_6_1_icmc_r6.1.1_recibir_cambios_red.psink_service_fact_353	service_fact_353.psource_service	0.00	0.00
o_6_2_icmc_r6.1.2_almacenar_cambios_red.psink_r6.1.2_almacenar_cambios_red	service_fact_353.psource_service	2.00	883.00
o_6_3_icmc_r6.1.3_replicar_cambios_red.psink_r6.1.3_replicar_cambios_red	service_fact_353.psource_service	3.00	894.00
o_6_4_icmc_r6.2_actualizar_configuracion.psink_r6.2_actualizar_configuracion	service_fact_353.psource_service	4.00	924.00
o_6_5_icmc_r6.3_enviar_cambios_configuracion.psink_r6.3_enviar_cambios_configuracion	service_fact_353.psource_service	5.00	1124.0
o_6_6_done	service_fact_353.psource_service	6.00	1204.0

Ilustración 34: Tabla de tiempos de respuesta.

6.11 EVALUACIÓN DE LA ARQUITECTURA MODIFICADA EN ARCHE

La principal diferencia entre la arquitectura modificada y la original se basa en la reducción de los costes de replicación, ya que al modificar la M-Plane DB sólo es necesario enviar los identificadores de los objetos que han cambiado y además sólo aquellos procesos que se encuentren suscritos. Debido esto se estimará que el coste es mínimo.

Sólo se incluye en esta sección los nuevos costes de las responsabilidades ya que el resto de tablas no cambian.

6.12 MAPEO FUNCIONALIDAD - RESPONSABILIDAD

La siguiente tabla muestra los nuevos costes de la replicación. Tal y como se aprecia son los únicos que disminuyen[66].

Funcionalidad	*Id Resp*	*Responsabilidad*	*Coste*
Oper_EnviarConf_DescFich	**R1.1**	Descargar Fichero de Configuración	160 ms
Oper_EnviarConf_AlmFichFlash	**R1.2**	Almacenar Fichero en Flash	200 ms
Comun_AlmFichDB	**R1.3.1**	Almacenar Datos configuración en M-Plane DB	60 ms
-	**R1.3.2**	Replicar Datos Configuración	1 ms
Comun_RecNotHw	**R2.1.1**	Recibir Notificaciones Hw	10 ms
Comun_AlmacHwDB	**R2.1.2**	Almacenar Datos Hw en M-Plane DB	1 ms
-	**R2.1.3**	Replicar Datos Hw	1 ms
Oper_ConecHw_MapHw	**R2.2**	Mapear Hw	1 ms
-	**R2.2.1**	Replicar Mapeo Hw	1 ms
Oper_ConecHw_NotHw	**R2.3**	Notificar Hw	1 ms
Hw_ActHw_ProcCambHw	**R3.1**	Procesar Cambios Hw	1 ms
Hw_ActHw_NotCambHw	**R3.2**	Notificar Cambios Hw	1 ms
Hw_NotAlarm_GenAlarm	**R4.2**	Generar Alarmas	10 ms
-	**R4.2.1**	Replicar Datos Alarmas	1 ms
Hw_NotAlarm_EnvAlarm	**R4.3**	Enviar Alarmas	1 ms
Hw_BorrAlarm_CancAlarm	**R5.2**	Cancelar alarmas	10 ms
Hw_BorrAlarm_EnvBorrAlarm	**R5.3**	Enviar Borrado Alarmas	1 ms

66 Aunque sería más adecuado el término “notificar”, se mantiene el término “replicar” en el nombre de las responsabilidades para evitar confusiones, debido a que el término “notificar” se ha utilizado para la acción de enviar información al exterior de la BTS.

ElemRed_ActConf_DetCamb_RecCamb	**R6.1.1**	Recibir Cambios Red	80 mls
ElemRed_ActConf_DetCamb_AlmCamb	**R6.1.2**	Almacenar Cambios Red	6 mls
-	**R6.1.3**	Replicar Cambios Red	1 mls
ElemRed_ActConf_AlmConf Flash	**R6.2**	Actualizar Configuración	200 mls
ElemRed_ActConf_EnvConf	**R6.3**	Enviar Cambios Configuración	10 mls

Tabla 19: Mapeo entre funcionalidades y responsabilidades para la arquitectura mejorada.

6.13 INTRODUCCIÓN DE DATOS Y EVALUACIÓN EN ARCHE

Tras el cálculo de los nuevos costes se introducen estos en ArchE y se evalúa de nuevo la arquitectura.

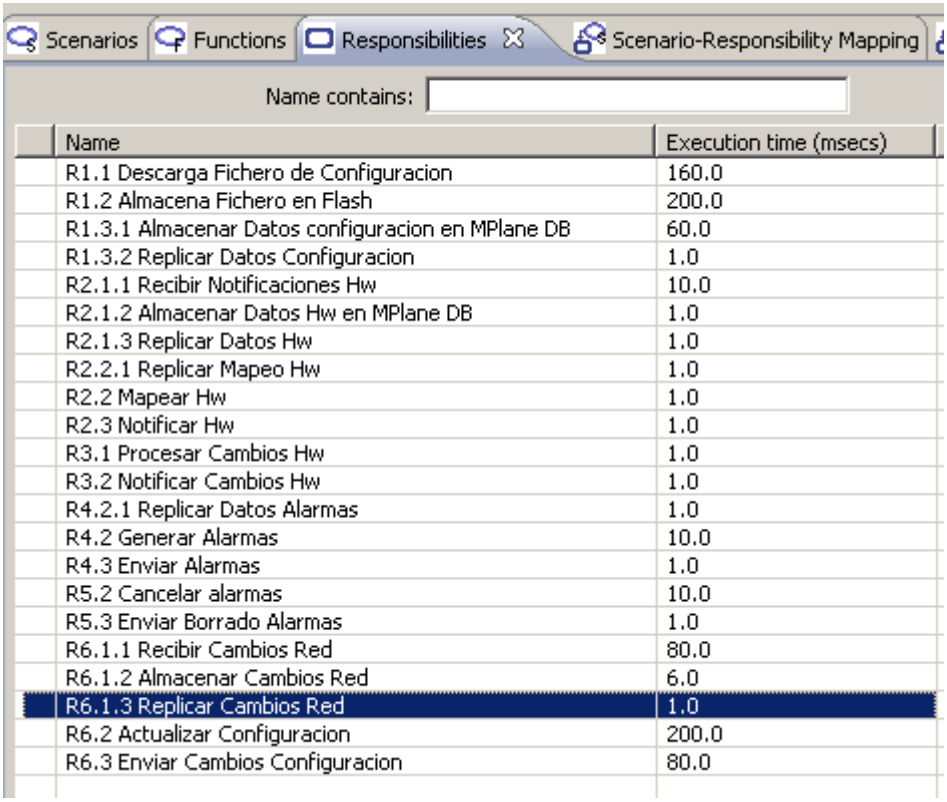


Ilustración 35: Responsabilidades y sus costes tras la mejora de la arquitectura.

Las siguientes ilustraciones muestran el resultado de los cambios en la arquitectura. Como se aprecia, tras su aplicación todos los escenarios son viables y el nuevo tiempo de respuesta en el peor de los casos es de 848 ms, es decir 356 ms menos. Esto supone una mejora, a priori[67], de casi el 30% de rendimiento.

67 La evaluación no ha tenido en cuenta el coste de la sincronización entre procesos y otros muchos detalles. Por tanto debe tomarse como lo que es, una estimación grosso modo.

De cara al futuro, se aprecia que los mayores tiempos vienen dados por la escritura flash, por lo tanto, deberá considerarse la necesidad de introducir mejoras, tales como, reducir el número de escrituras y el tamaño de las mismas[68] (control de la demanda), o utilizar otro medio de almacenamiento más rápido (control de los recursos).

68 De hecho esta aproximación se ha realizado en el producto real, ya que el hardware no puede cambiarse, o de lo contrario podría perderse información en caso de que ocurra un fallo en la alimentación. En dicho caso se perderán todos los cambios desde la última vez que se almacenó el fichero.

		Description	ScenarioType	Stimulus	StimulusType	Source	Artifact	Environment	Response	Measure	Value
		S1: El operador envía la configuración de la BTS. La BTS d...	ICM Perfor...	El operador...	Periodic	Operador	BTS	En condicio...	La configur...	En segundos	10.0
		S2: El operador conecta un nuevo equipo de radio a la BT...	ICM Perfor...	El operador...	Periodic	Operador	BTS	En condicio...	La BTS actu...	En segundos	2.0
		S3: Un elemento Hw envía información sobre sus variable...	ICM Perfor...	Un element...	Periodic	Elemento Hw	BTS	En condicio...	La BTS actu...	En segundos	1.0
		S4: Se produce un fallo en un elemento Hw. El fallo es de...	ICM Perfor...	Se produce...	Periodic	Elemento Hw	BTS	En condicio...	La BTS gen...	En segundos	2.0
		S5: El elemento Hw previamente en fallo se recupera. La ...	ICM Perfor...	Se recuper...	Periodic	Elemento Hw	BTS	En condicio...	La BTS can...	En segundos	2.0
		S6: Se detecta un cambio de configuración en un element...	ICM Perfor...	Se produce...	Periodic	Elemento d...	BTS	En condicio...	La BTS actu...	En segundos	2.0

Ilustración 36: Todos los escenarios son viables con la nueva arquitectura.

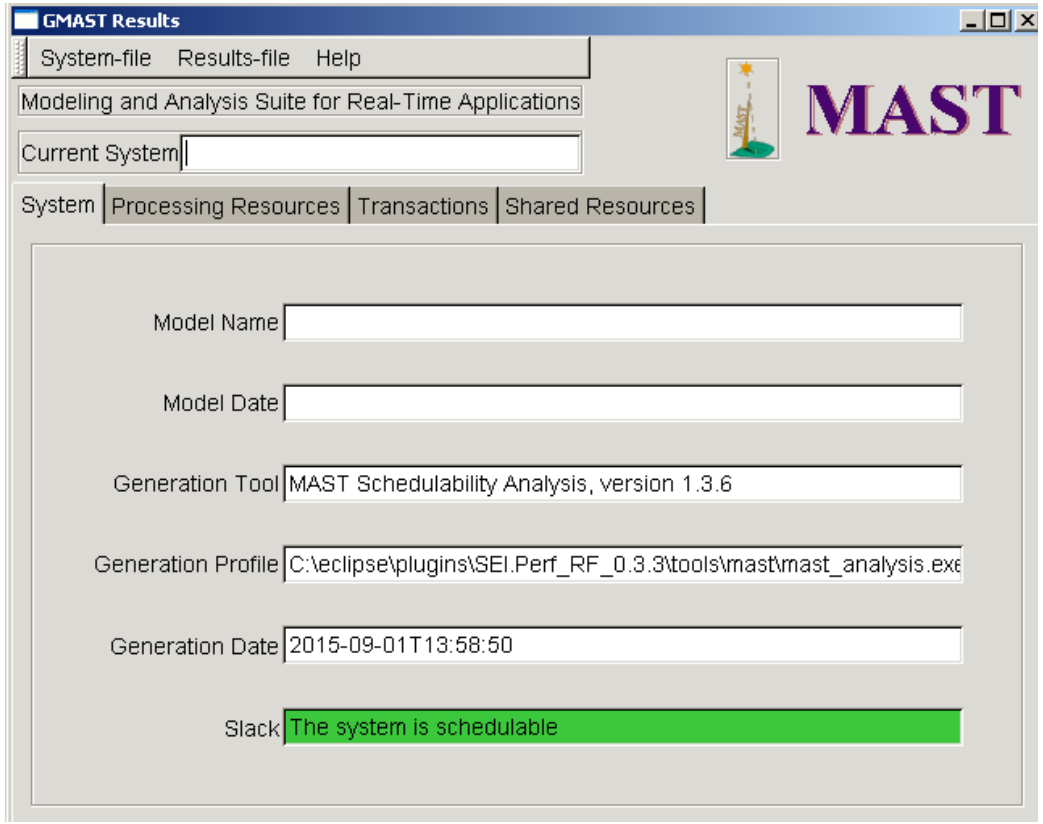


Ilustración 37: MAST informa acerca de la viabilidad del sistema.

o_1_5_done	service_fact_7.psource_service	5.00	848.00
o_2_1_icmc_r2.1.1_recibir_notificaciones_hw.psink_service_fact_8	service_fact_8.psource_service	0.00	0.00
o_2_2_icmc_r2.1.2_almacenar_datos_hw_en_mplane_db.psink_r2.1.2_almacenar_datos_hw_en_mplane_db	service_fact_8.psource_service	2.00	838.00
o_2_3_icmc_r2.1.3_replicar_datos_hw.psink_r2.1.3_replicar_datos_hw	service_fact_8.psource_service	0.00	0.00
o_2_4_icmc_r2.2_mapear_hw.psink_r2.2_mapear_hw	service_fact_8.psource_service	4.00	845.00
o_2_5_icmc_r2.2.1_replicar_mapeo_hw.psink_r2.2.1_replicar_mapeo_hw	service_fact_8.psource_service	5.00	846.00
o_2_6_icmc_r2.3_notificar_hw.psink_r2.3_notificar_hw	service_fact_8.psource_service	6.00	847.00
o_2_7_done	service_fact_8.psource_service	7.00	848.00
o_3_1_icmc_r3.1_procesar_cambios_hw.psink_service_fact_9	service_fact_9.psource_service	0.00	0.00
o_3_2_icmc_r2.1.2_almacenar_datos_hw_en_mplane_db.psink_r2.1.2_almacenar_datos_hw_en_mplane_db	service_fact_9.psource_service	2.00	840.00
o_3_3_icmc_r2.1.3_replicar_datos_hw.psink_r2.1.3_replicar_datos_hw	service_fact_9.psource_service	0.00	0.00
o_3_4_icmc_r3.2_notificar_cambios_hw.psink_r3.2_notificar_cambios_hw	service_fact_9.psource_service	4.00	847.00
o_3_5_done	service_fact_9.psource_service	5.00	848.00
o_4_1_icmc_r3.1_procesar_cambios_hw.psink_service_fact_10	service_fact_10.psource_service	0.00	0.00
o_4_2_icmc_r2.1.2_almacenar_datos_hw_en_mplane_db.psink_r2.1.2_almacenar_datos_hw_en_mplane_db	service_fact_10.psource_service	2.00	829.00
o_4_3_icmc_r2.1.3_replicar_datos_hw.psink_r2.1.3_replicar_datos_hw	service_fact_10.psource_service	0.00	0.00
o_4_4_icmc_r4.2_generar_alarmas.psink_r4.2_generar_alarmas	service_fact_10.psource_service	4.00	836.00
o_4_5_icmc_r4.2.1_replicar_datos_alarmas.psink_r4.2.1_replicar_datos_alarmas	service_fact_10.psource_service	5.00	846.00
o_4_6_icmc_r4.3_enviar_alarmas.psink_r4.3_enviar_alarmas	service_fact_10.psource_service	0.00	0.00
o_4_7_done	service_fact_10.psource_service	7.00	848.00
o_5_1_icmc_r3.1_procesar_cambios_hw.psink_service_fact_11	service_fact_11.psource_service	0.00	0.00
o_5_2_icmc_r2.1.2_almacenar_datos_hw_en_mplane_db.psink_r2.1.2_almacenar_datos_hw_en_mplane_db	service_fact_11.psource_service	2.00	829.00
o_5_3_icmc_r2.1.3_replicar_datos_hw.psink_r2.1.3_replicar_datos_hw	service_fact_11.psource_service	0.00	0.00
o_5_4_icmc_r5.2_cancelar_alarmas.psink_r5.2_cancelar_alarmas	service_fact_11.psource_service	4.00	836.00
o_5_5_icmc_r4.2.1_replicar_datos_alarmas.psink_r4.2.1_replicar_datos_alarmas	service_fact_11.psource_service	5.00	846.00
o_5_6_icmc_r5.3_enviar_borrado_alarmas.psink_r5.3_enviar_borrado_alarmas	service_fact_11.psource_service	6.00	847.00
o_5_7_done	service_fact_11.psource_service	7.00	848.00
o_6_1_icmc_r6.1.1_recibir_cambios_red.psink_service_fact_353	service_fact_353.psource_service	0.00	0.00
o_6_2_icmc_r6.1.2_almacenar_cambios_red.psink_r6.1.2_almacenar_cambios_red	service_fact_353.psource_service	2.00	556.00
o_6_3_icmc_r6.1.3_replicar_cambios_red.psink_r6.1.3_replicar_cambios_red	service_fact_353.psource_service	3.00	567.00
o_6_4_icmc_r6.2_actualizar_configuracion.psink_r6.2_actualizar_configuracion	service_fact_353.psource_service	4.00	568.00
o_6_5_icmc_r6.3_enviar_cambios_configuracion.psink_r6.3_enviar_cambios_configuracion	service_fact_353.psource_service	5.00	768.00
o_6_6_done	service_fact_353.psource_service	6.00	848.00

Ilustración 38: Tabla de tiempos de respuesta con la nueva arquitectura.

7. CONCLUSIONES Y TRABAJOS FUTUROS

7.1 CONCLUSIONES

En el presente trabajo se ha empleado la herramienta ArchE con éxito para evaluar una arquitectura cercana a la realidad del ámbito de las telecomunicaciones. En particular se ha analizado la arquitectura de una BTS y en concreto de su capa M-Plane, la cual emplea el patrón *blackboard* como modelo de colaboración entre los componentes. Dichos componentes están formados por una serie de procesos que se comunican entre sí mediante la lectura y escritura de una base de datos distribuida.

Tras el análisis de la arquitectura mencionada, y partiendo desde los atributos de calidad, se ha llegado a la conclusión de que una base de datos distribuida complica el diseño y empeora el rendimiento. En concreto, en la arquitectura original concurren los siguientes problemas:

- Existe una redundancia de la base de datos debido a las copias de la base de datos en cada uno de los procesos.
- La arquitectura introduce retardos, debido a la comunicación con el intermediario para la modificación y posterior replicación de los datos.
- Existe cierta dificultad para razonar a la hora de diseñar debido a la existencia de múltiples vistas de la base datos a lo largo de los distintos procesos. Esto implica que no se puede suponer un estado particular para otro proceso a no ser que el estado actual lo implique a nivel semántico.
- El coste de la adición de futuros componentes es lineal con respecto al rendimiento y a la memoria.

Como mejora para dicha arquitectura se ha propuesto la centralización de la base de datos mediante el uso de un fichero en memoria compartida. Dicha arquitectura modificada presenta las siguientes ventajas:

- La nueva arquitectura requiere una menor cantidad de memoria debido a que se eliminan las copias redundantes.
- Se eliminan los retardos innecesarios ya que no es necesario comunicar con un intermediario.
- El coste de añadir componentes a la nueva arquitectura es constante con respecto a la memoria y cercano a este con respecto al rendimiento.

La nueva arquitectura en ArchE arrojó una mejora en el rendimiento de al menos el 30% además de ayudar a vislumbrar los futuros cuellos de botella a resolver.

7.2 TRABAJOS FUTUROS

La evaluación de la nueva arquitectura en ArchE ha puesto de relieve que los tiempos de escritura en flash constituyen tiempos de respuesta que están cerca de los límites aceptables en los escenarios. De cara a futuras mejoras, o posibles extensiones de los requisitos de la BTS, sería bueno explorar alternativas para mejorar el rendimiento, como la limitación de la escritura, ya sea escribiendo con menos frecuencia o escribiendo en una estructura distinta que no implique actualizar el fichero completo. También cabe mejorar el hardware para conseguir tasas de escritura

más rápidas, pero esto no es aplicable a las versiones de hardware actualmente en explotación por el cliente que todavía están en condiciones de actualizarse a futuras versiones.

Por otro lado, como se ha indicado, caben muchas líneas de mejora para la herramienta ArchE y en particular a su marco de rendimiento, como las que se describen a continuación:

- Dotar a dicho marco de la capacidad de proponer tácticas. Sería necesario estudiar de qué forma afectaría y en qué aspectos.
- Dotar a las responsabilidades de mayor semántica, que permita expresar relaciones más complejas en las responsabilidades, como el funcionamiento en paralelo, la sincronización, etc.
- Mejorar la usabilidad de la herramienta, permitiendo la introducción de datos relacionados con el marco de rendimiento de forma *offline* y evitando así ejecuciones innecesarias para datos intermedios.

8. BIBLIOGRAFÍA

8.1 OBRAS Y ARTÍCULOS

[Bachman 00] Bachman, F., Bass, L., Chastek, G., Donohoe, P. y Peruzzi, F., "The Architecture Based Design Method", SEI, 2000.

[Barbacci 03] Barbacci, M., Ellison, R., Lattanze, A., Stafford, J., Weinstock, C. y Wood, W., *Quality Attribute Workshops (QAWs)*, 3ª edición, SEI, agosto 2003.

[Bass 01] Bass, L., Klein, M. y Bachman, F., "Quality Attribute Design Primitives and the Attribute Driven Design Method", *SEI*, 2001.

[Bass 07] Bass, L., "ArchE the Architecture Expert", SEI, 2007.

[Bass 13] Bass, L., Clements, P. y Kazman, R., *Software Architecture in Practice*, Addison-Wesley, 3ª edición, 2013.

[Bengtsson 04] Bengtsson, P., Lassing, N., Bosch, J., y Van Vliet, H., "Architecture-level modifiability analysis (ALMA)", *Journal of Systems and Software*, 69(1), 2004, págs. 129-147.

[Bosch 00] Bosch, J., *Design & Use of Software Architectures*, Addison-Wesley, 2000.

[Chastek 96] Chastek, G.y Brownsword, L., "A Case Study in Structural Modeling", Carnegie-Mellon University y SEI, diciembre 1996.

[Clements 00] Clements, P., "Active Reviews for Intermediate Designs", Carnegie-Mellon University y SEI, agosto 2000.

[Clements 01] Clements, P., Kazman, R. y Klein, M., *Evaluating software architectures*, Addison-Wesley Professional, 2001.

[Dolan 02] Dolan, T., "Architecture Assessment of Information-System Families", Eindhoven University of Technology, febrero 2002.

[Gamma 94] Gamma, E., Helm, R., Johnson, R. y Vlissides, J., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, octubre 1994.

[Garlan 94] Garlan, D. y Shaw, M., "An introduction to Software Architecture", *School of Computer Science, Carnegie Mellon University, Pittsburgh*, enero 1994.

[Garlan 95] Garlan, D. y Perry, D., "Introduction to the special issue on software architecture", *IEEE Transactions on Software Engineering*, vol. 21(4), 1995, págs. 269-274.

[Ionita 02] Ionita, M. T., Hammer, D. K. y Obbink, H., "Scenario-based software architecture evaluation methods: An overview", ICSE/SARA, 2002.

8.2 RECURSOS

[1] Página oficial de la plataforma Jess: http://herzberg.ca.sandia.gov/.

[2] Página oficial de la plataforma XmlBlaster: http://www.xmlblaster.org/.

[3] Descripción del método Cost Benefit Analysis Method (CBAM) en la página del SEI: http://www.sei.cmu.edu/architecture/tools/evaluate/cbam.cfm.

[4] Descripción del método Architecture Tradeoff Analysis Method en la página del SEI: http://www.sei.cmu.edu/architecture/tools/evaluate/atam.cfm.

[5] Página oficial del proyecto Sonarqube: http://www.sonarqube.org/.

[6] Blog de investigación sobre seguridad: https://penturalabs.wordpress.com/2013/12/07/what-is-2g-3g-4g/

[7] ANSI SQL 92 estándar, ISO/IEC 9075:1992.

[8] Página oficial del MDA en la página del Object Management Group: http://www.omg.org/mda/.

[9] Página oficial del *plugin* GEF de Eclipse: http://www.eclipse.org/gef/.

[10] Página oficial de Eclipse: http://www.eclipse.org/.

[11] Página oficial de MySQL: https://www.mysql.com/.

9. DEFINICIÓN DE SIGLAS, ATRIBUTOS Y ACRÓNIMOS

Término	*Definición*
BTS	Una estación de transmisión básica (*base transceiver station* o BTS en inglés) es un equipo que provee la comunicación inalámbrica entre un UE y una red.
dirty read	Problema de aislamiento de la base de datos en el que una lectura puede obtener datos temporales no confirmados por otra transacción.
eNodeB	E-UTRAN Node B, también conocido como *Evolved Node B*, (“Nodo B evolucionado” en español), es el elemento hardware de la red LTE conectado a la red de telefonía móvil que comunica directamente con el UE tal y como lo hacía la BTS en una red GSM.
feature	El Instituto de ingenieros eléctricos y electrónicos (IEEE en inglés) define el término *feature* en el estándar IEEE 829 como “Una característica distinguida de un elemento software".
FTP	FTP (siglas en inglés de *File Transfer Protocol*, “Protocolo de Transferencia de Archivos”): en informática es un protocolo de red para la transferencia de archivos entre sistemas conectados a una red TCP (*Transmission Control Protocol*), basado en la arquitectura cliente-servidor.
GEF	Framework de edición gráfica (*Graphical Editing Framework* o GEF en inglés): provee la tecnología necesaria para crear editores y vistas de gráficos enriquecidos.
IP	*Internet Protocol* (en español “protocolo de internet”) o IP es un protocolo de comunicación de datos digitales clasificado funcionalmente en la capa de red según el modelo internacional OSI.
JDK	*Java Development Kit* o JDK es un software que provee herramientas de desarrollo para la creación de programas en Java.
JRE	*Java Runtime Environment* o JRE es un conjunto de utilidades que permite la ejecución de programas Java. En su forma más simple, el entorno en tiempo de ejecución de Java está conformado por una Máquina Virtual de Java o JVM, un conjunto de bibliotecas Java y otros componentes necesarios para que una aplicación escrita en lenguaje Java pueda ser ejecutada. El JRE actúa como un “intermediario” entre el sistema operativo y Java.
J2SE	Hasta la versión 5.0 la plataforma Java SE se conocía como *Java Platform* 2, *Standard Edition* o J2SE. En la actualidad se denomina sin la versión como: *Java Platform, Standard Edition* o Java SE.
LTE	*Long Term Evolution* (LTE, “Evolución a Largo Plazo”) es un estándar de la norma 3GPP definida por unos como una evolución de la norma 3GPP UMTS (3G), y por otros como un nuevo concepto de arquitectura evolutiva (4G).
MAST	MAST es una *suite* de herramientas de código abierto para realizar análisis de

	sistemas distribuidos condicionados por una amplia variedad de requisitos temporales.
MDA	La arquitectura dirigida por modelos (*Model-Driven Architecture* o MDA) es un acercamiento al diseño de software, propuesto y patrocinado por el Object Management Group (OMG). MDA se ha concebido para dar soporte a la ingeniería dirigida a modelos de los sistemas de software.
plugin	Un complemento (*plugin* en inglés) es una aplicación que se relaciona con otra para aportarle una función nueva y generalmente muy específica. Esta aplicación adicional es ejecutada por la aplicación principal y ambas interactúan por medio de la API.
RPC	La Llamada a Procedimiento Remoto (del inglés *Remote Procedure Call*, RPC) es un protocolo de red que permite a un programa de computadora ejecutar código en otra máquina remota sin tener que preocuparse por las comunicaciones entre ambas.
SEI	El Software Engineering Institute (SEI) es un instituto federal estadounidense de investigación y desarrollo, fundado por Congreso de los Estados Unidos en 1984 para desarrollar modelos de evaluación y mejora en el desarrollo de software, que dieran respuesta a los problemas que generaba al ejército estadounidense la programación e integración de los subsistemas de software en la construcción de complejos sistemas militares.
script	Un *script* archivo de órdenes, archivo de procesamiento por lotes o guión es un programa usualmente simple, que por lo regular se almacena en un archivo de texto plano. Los *scripts* suelen ser casi siempre interpretados.
UE	En una red de telefonía móvil el *User Equipment* (“equipo del usuario” en inglés) o UE es el equipo utilizado directamente por el usuario para comunicarse. Puede ser un teléfono móvil, un portátil, etc.
3GPP	El 3GPP (*3rd Generation Partnership Project*: Proyecto Asociación de Tercera Generación) es una colaboración de grupos de asociaciones de telecomunicaciones, conocidos como miembros organizativos. La estandarización 3GPP abarca radio, redes de núcleo y arquitectura de servicio.

ANEXO I: MEJORAS DEL MARCO DE RENDIMIENTO DE ARCHE

A lo largo de este apartado se describen las correcciones destinadas a solucionar algunas de las limitaciones mencionadas previamente en el marco de rendimiento de la herramienta ArchE.

PRERREQUISITOS PARA LA COMPILACIÓN DE LOS MÓDULOS DE ARCHE

Las limitaciones indicadas previamente se encuentran situadas en la implementación del módulo: SEI.ArchE.External.Examples_1.0.0 que se instala dentro de los *plugins* de Eclipse. Las fuentes de dicho módulo se encuentran dentro de la propia carpeta del *plugin* en el archivo ExternalRFExamplessrc.zip. Además, para compilar dicho proyecto se necesitarán resolver varias dependencias:

- JDK 5.0. Puesto que se está usando la JRE 1.5 para ejecutar ArchE, el *plugin* deberá compilarse con la JDK 5.0 ya que una superior no será reconocida por el entorno de ejecución.
- Las librerías de Eclipse. Para simplificar se añaden todas las librerías de la carpeta de *plugins* de Eclipse cuyo nombre empiece por el prefijo org.eclipse[69].
- Las librerías dentro del propio *plugin*. Estas se encuentran dentro de la carpeta del *plugin* en la carpeta lib.
- La librería SEI.ArchE.External.0.1.0. Esta librería define las interfaces para implementar marcos de razonamiento así como varias funcionalidades comunes.
- La librería Perf_RF. Esta contiene la implementación base para realizar análisis de rendimiento basándose en simuladores externos tales como MAST, utilizado por el marco de razonamiento de ejemplo.

LIMITACIÓN DE COSTES DE FUNCIONALIDADES A 10 MS

Esta limitación se analiza en la clase *ICMPerformanceReasoningFramework* dentro del método *checkRFDependencies*.

```
                    if ((value > MAX_EXECUTION_TIME) || (value < MIN_EXECUTION_TIME)) {

                            // Error reporting
                            String message = "Parameter "+PARAMETER_EXECUTION_TIME+" must be in
range ["
                                    + MIN_EXECUTION_TIME +" - "
                                    + MAX_EXECUTION_TIME + "] (value= "+ value + " !?)";
                    }
```

De dicho código se desprende que para eliminar la limitación basta con cambiar el valor de variable `MAX_EXECUTION_TIME`. Para el presente trabajo se ha subido el valor a 10 segundos[70].

```
private static final double MAX_EXECUTION_TIME            = 10000.0;
```

69 Una aproximación más adecuada sería identificar estrictamente aquellas dependencias que son necesarias. No obstante, puesto que el número de dependencias no es pequeño, se ha tomado esta opción con el fin de ahorrar tiempo.

70 Se escribe 10000 puesto que el valor debe expresarse en ms.

CORRECCIÓN PARA EL CÁLCULO DE COSTES EN TAREAS ENCADENADAS Y BIFURCADAS

El problema del cálculo de costes se debe a una mala conexión de las tareas implicadas en un escenario. Para dicho cálculo el marco de razonamiento debe convertir el modelo basado en escenarios, responsabilidades y relaciones, a un modelo basado en servicios, componentes y conexiones[71].

En el modelo final, las conexiones estarán conectadas entre sí y además estarán asociadas a los componentes que interconectan. Cada conexión puede ser de dos tipos: origen y destino[72]. Además cada conexión almacena información de los elementos a los que está conectada en ambas direcciones de la cadena. Por tanto, el pin de origen almacena los pines de destino que lo invocan y los pines de destino que él invoca. Igualmente, el pin de destino, almacena los pines de origen que lo invocan y los pines de origen que él invoca.

La funcionalidad expuesta anteriormente se realiza en la clase *ICMWrapper* en su método *createICM*.

En particular, la implementación original ejecutaba los siguientes pasos:

- Para cada responsabilidad se crea un componente.
- Para cada relación se crea un pin de origen y otro de destino y se asocian a sus respectivos componentes asociados a las responsabilidades de la relación. Cada pin de destino tendrá el coste de tiempo de su componente (responsabilidad) asociado. Además se conecta el pin de origen al de destino y viceversa.
- Para cada escenario se crea un servicio y un pin de origen[73] del servicio. Además, para cada componente que es inicial, es decir, que no es invocado por nadie en ninguna relación, se crea un pin de destino con su coste asociado al componente y se asocia al pin de origen del escenario.

El diagrama siguiente ilustra el modelo creado y su conectividad para un caso a modo de ejemplo donde un escenario S utiliza las responsabilidades A, B y C, y donde existen dos relaciones: A-B y B-C. Como se puede observar, la clase ha creado un modelo con un servicio cuyo pin de origen se ha conectado a un pin de destino correspondiente al componente A. Los pines correspondientes a las relaciones están conectados entre sí pero no han sido conectados ni al servicio ni entre las propias relaciones. Cada componente está conectado a sus pines asociados, pero esta información no es utilizada a la hora de generar el modelo que se va a evaluar. Como resultado, el escenario recibido por la herramienta MAST contendrá sólo los elementos dentro del marco rojo, es decir, el correspondiente al componente A. Por tanto, sólo el coste de su responsabilidad asociada será tenido en cuenta a la hora de evaluar el escenario, lo cual es incorrecto.

71 ICMService, ICMComponent y PinInstance en el código.
72 SourcePinInstance y SinkPinInstance en el código.
73 ServiceSourcePinIcm en el código, siendo aquel un subtipo de SourcePinInstance.

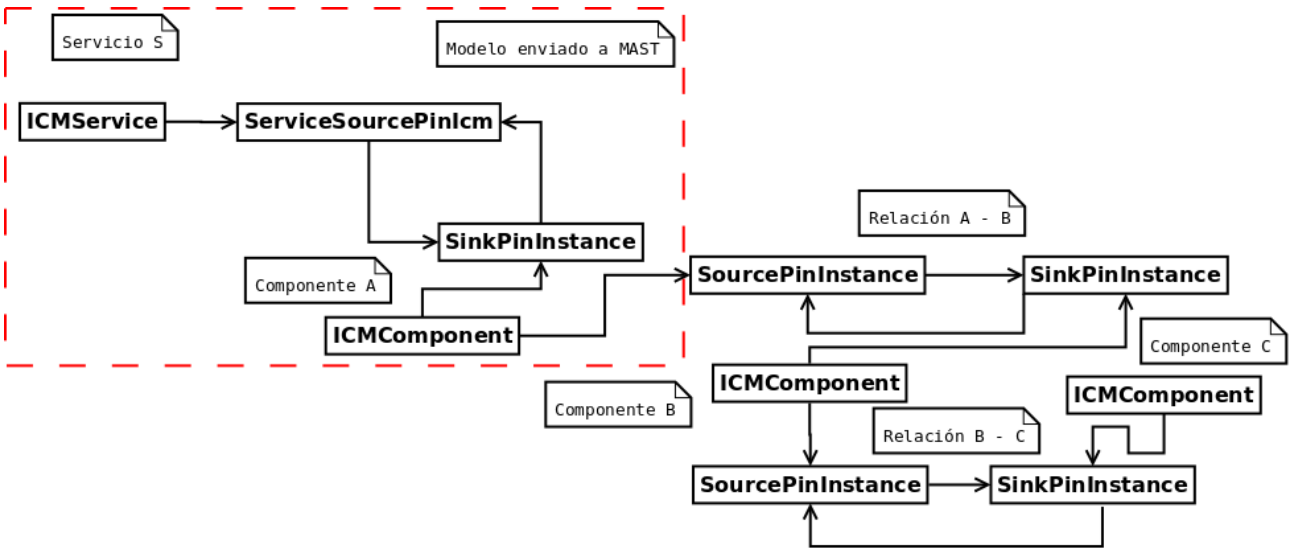


Ilustración 39: Modelo generado por la clase ICMWrapper original (Fuente: elaboración propia).

Para una correcta generación del modelo se modifica el algoritmo de forma que se realicen los pasos explicados a continuación.

Paso 1

Para cada responsabilidad se crea un componente asociado. Este paso se mantiene como estaba, ya que no es necesaria ninguna modificación.

Paso 2

Para cada escenario se crean pines asociados sólo a las relaciones cuyas responsabilidades forman parte del escenario. Además se crea una base de datos con la información de los pines de origen y de destino de las relaciones de cada escenario.

Paso 2.1

Para cada relación se crea un componente de destino por cada responsabilidad hija en la relación eliminando duplicados. Es decir, si existen dos relaciones A-B y C-B, sólo se creará un pin de destino para el componente B.

Paso 2.2

Por cada relación se crea, si no existe ya, un pin de origen para la responsabilidad padre. Además se conecta el pin de origen con el de destino que él invoca dentro de dicha relación. Por último se añade dicho pin de origen al pin de destino que le hace reaccionar. Esto enlaza dos relaciones cuando el elemento hijo de la primera y el padre de la segunda coinciden. Por ejemplo, para el caso A-B y B-C, el pin de origen correspondiente a B se añade a la lista de pines que A invoca. Además se crean las conexiones de forma que el pin de origen A invoca al de destino B y el de origen B al de destino C[74].

74 Esto último ya lo hacía correctamente el código original.

A continuación se incluye el código completo del paso 2.

```
    // Rule 2: For each reaction relationship create corresponding source and sink pins
    // in the components associated to the responsibilities involved in the relationship
    IcmComponent component = null;
    SinkPinInstance pinSinkInstance = null;
    SourcePinInstance pinSourceInstance = null;
    ArchEResponsibilityReactionRelationVO reaction = null;
    List<String> connectedComponents = new ArrayList<String>();
     Map<ArchEScenarioVO, Map<String, SinkPinInstance>> reactSinksdb = new HashMap<ArchEScenarioVO,
Map<String, SinkPinInstance>>();
             Map<ArchEScenarioVO,   Map<String,   SourcePinInstance>>   initialSourcesdb   =   new
HashMap<ArchEScenarioVO, Map<String, SourcePinInstance>>();

    double avgExecutionTime = 0;
    Distribution uniform = null;

    for (Iterator<ArchEScenario> itScenarios = scenarios.iterator(); itScenarios.hasNext();) {
      ArchEScenarioVO scItem = (ArchEScenarioVO) (itScenarios.next());
      Map<String, SinkPinInstance> reactSinks = new HashMap<String, SinkPinInstance>();
      Map<String, SourcePinInstance> initialSources = new HashMap<String, SourcePinInstance>();

      reactSinksdb.put(scItem, reactSinks);
      initialSourcesdb.put(scItem, initialSources);

      Set<String> respNames = getMappedResponsibilitiesNames(scItem, translations, reactions);

      System.out.println("DEBUG: scenario " + scItem + "respNames: " + respNames);

    // For connecting the pins within a component
      // Iterave over reactions to create sink pins
      for (Iterator<ArchERelation> it1 = reactions.iterator(); it1.hasNext();) {
        reaction = (ArchEResponsibilityReactionRelationVO) (it1.next());

        nameWithoutSpaces = removeSpaces(reaction.getChild().getName());
        component = allComponents.get(nameWithoutSpaces);

        if (!respNames.contains(nameWithoutSpaces)) {
             System.out.println("DEBUG: Child doesn't apply for current scenario. Child Name:" +
nameWithoutSpaces);
          continue;
        }

      // This creates a sink pin for the child responsibility,
        // assuming that parent --> child maps to source_parent --> sink_child
        pinSinkInstance = factory.createSinkPinInstance();
        pinSinkInstance.setMode(DEFAULT_SINKPIN_MODE);
        pinSinkInstance.setPriority(getTaskPriority());
        pinSinkInstance.setName("psink_" + nameWithoutSpaces);

        try {
                                                                          avgExecutionTime       =
reaction.getChild().getDoubleParameter(ICMPerformanceReasoningFramework.PARAMETER_EXECUTION_TIME);
          uniform = getUniformDistribution(factory, 1.0, avgExecutionTime);
          pinSinkInstance.setExecTimeDistribution(uniform);
        } catch (ArchEException e) {
          // TODO Auto-generated catch block
          e.printStackTrace();
        }

        pinSinkInstance.setElementInstance(component);

      //Save the sink in the reactMap to latter connect it to the
        //reacting sources
        reactSinks.put(nameWithoutSpaces, pinSinkInstance);
```

```
      // This saves the names of those components that have at least one sink pin
        // connected, for later.
        if (!connectedComponents.contains(nameWithoutSpaces)) {
          connectedComponents.add(nameWithoutSpaces);
        }
      }

      // Iterate over reactions to process fhe source pins
      for (Iterator<ArchERelation> it1 = reactions.iterator(); it1.hasNext();) {
        reaction = (ArchEResponsibilityReactionRelationVO) (it1.next());

        nameWithoutSpaces = removeSpaces(reaction.getParent().getName());

        if (!respNames.contains(nameWithoutSpaces)) {
            System.out.println("DEBUG: Parent doesn't apply for current scenario. Parent Name: " +
nameWithoutSpaces);
          continue;
        }

        pinSourceInstance = initialSources.get(nameWithoutSpaces);

        // If the sourcePin doesn't already exits
        if (pinSourceInstance == null) {
          // This creates a source pin for the child responsibility
          pinSourceInstance = factory.createSourcePinInstance();
          pinSourceInstance.setMode(DEFAULT_SOURCEPIN_MODE);

          pinSourceInstance.setName("psource_" + nameWithoutSpaces);
          component = allComponents.get(nameWithoutSpaces);
          pinSourceInstance.setElementInstance(component);
        }

        String sinkName = removeSpaces(reaction.getChild().getName());

        // This connects the source for the sink to react to
        pinSinkInstance = reactSinks.get(sinkName);
        if (pinSinkInstance != null) {
          pinSourceInstance.getSinks().add(pinSinkInstance);
        }

      //Save the source pink for later bind to the service sink pin
        //in case this component is initial
        initialSources.put(nameWithoutSpaces, pinSourceInstance);

        //Try to find the sink that could invoke this source
        pinSinkInstance = reactSinks.get(nameWithoutSpaces);
        if (pinSinkInstance != null) {
          pinSinkInstance.getReactSources().add(pinSourceInstance);
        }
      }
    }
```

Paso 3

Para cada escenario se crea un servicio y un pin de origen y de destino asociados. Además, para cada componente inicial, que no es invocado por ninguna relación[75], se crea un pin de destino, se busca el pin de origen asociado a dicho componente y se añade a las listas de pines que el pin creado invoca. De esta forma se establece la relación servicio-primer componente de la cadena de pines.

A continuación se incluye el código modificado correspondiente a la parte de búsqueda y enlace del

75 Es decir, no aparece como responsabilidad hija en ninguna relación.

pin inicial del servicio:

```
   // Create a sink pin for every component that is "initial" for the scenario
        for (Iterator<IcmComponent> itComponents = list.iterator(); itComponents.hasNext();) {
          initialComponent = itComponents.next();
        ...
        SourcePinInstance componentSource =
               initialSourcesdb.get(scItem).get(initialComponent.getName());

          if (componentSource != null) {
            System.out.print("DEBUG! bound source for " + initialComponent.getName());
            serviceSinkPin.getReactSources().add(componentSource);
          } else {
                          System.out.print("ERROR!  couldnt  find  sourcePin  for  component  "  +
initialComponent.getName());
          }
         }
        ...
       }
```

Por último, se incluye el modelo resultante con el código corregido con un caso de ejemplo más complejo. En dicho caso existe un escenario S, asociado a las responsabilidades A, B, C, D, E y a sus relaciones A-B, B-C, B-D. Para simplificar se han omitido los componentes del diagrama, pero se asume que por cada responsabilidad se ha creado un componente el cual contiene sus pines de origen y de destino asociados.

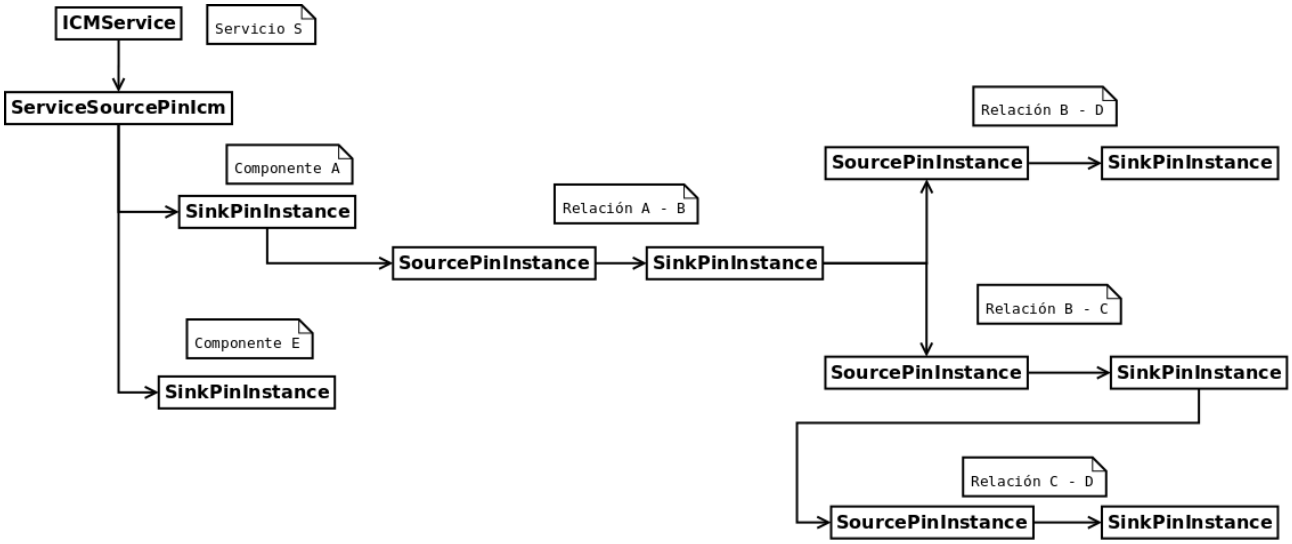


Ilustración 40: Modelo generado por la clase ICMWrapper corregida (Fuente: elaboración propia).

Tal y como se observa, la cadena ahora está correctamente enlazada y soporta las bifurcaciones como ocurre en las relaciones B-C y B-D. Además se han eliminado las conexiones innecesarias de los destinos a los orígenes que los invocan[76].

CORRECCIÓN DE LA DUPLICACIÓN DE LOS NOMBRES DE SERVICIO

Esta limitación se debe al hecho de utilizar para la creación del nombre de los servicios el campo *Id*

76 Aunque el *framework* permite la definición bidireccional de las relaciones, el *framework* de rendimiento de la librería Perf_RF sólo utiliza las relaciones en dirección de avance.

del escenario, que no es único. Para corregirlo se utiliza el campo *FactId*, que sí lo es.

A continuación se incluye el fragmento de código corregido en el método *createICM* de la clase *ICMWrapper*.

```
       nameWithoutSpaces = removeSpaces(scItem.getFactId());
       service.setName("Service_" + nameWithoutSpaces);
```

CORRECCIÓN DEL CÁLCULO DE TIEMPOS

Como ya se indicó, el *framework* no tiene en cuenta la unidad a la hora del cálculo de tiempos, y siempre la interpreta en milisegundos. Esta limitación se debe a que la implementación actual del *plugin* para el marco de razonamiento sencillamente hace caso omiso de la unidad seleccionada en la ventana de edición de la información de escenarios. Para subsanarlo se añade dicha lógica en diversas partes del código donde esa unidad es necesaria.

En la clase *ICMPerformanceReasoningFramework* se crea la siguiente función de conversión que se encarga de obtener el valor de la medida para el escenario en milisegundos, una vez tenida en cuenta la unidad seleccionada.

```
 private double getMeasureValueForScenario(ArchEScenario currentScenario) {
   double measureVal = 0.0; // The minimum worst case execution time for performance

   if (currentScenario.getMeasureValue() != null) {
     measureVal = currentScenario.getMeasureValue();

     //If the measure is in second, convert it to mls
     if (currentScenario.getMeasureUnit().startsWith("sec")) {
       measureVal *= 1000;
     }
   }

   return measureVal;
 }
```

Posteriormente, en el método *analyze* de la misma clase se utiliza la función de conversión expuesta más arriba.

```
public ArchEEvaluationResult analyze(ArchEArchitecture architecture,
         ArchEScenario currentScenario) throws ArchEException {

   ...
   double response = INVALID_RESPONSE;//Double.NaN;
   double measureVal = getMeasureValueForScenario(currentScenario);
```

La misma modificación es necesaria en el método *analyzeAndSuggest* para que la medida sea correctamente interpretada a la hora de calcular si el escenario se cumple o no.

```
@Override
 public ArchEAnalysisResult analyzeAndSuggest(
         ArchEArchitecture architecture, ArchEScenario currentScenario,
         List<ArchETryTacticResult> outTactics) throws ArchEException {

   ….
   if (this.getAnalysisStatus() == RF_WARNING) {
```

```
      this.setAnalysisStatus(RF_ERROR); // Because it's the analysis of the current architecture
    }
    double measureVal = getMeasureValueForScenario(currentScenario);
```

Además en la clase *ICMWrapper*, en concreto en su método *createICM*, es necesario modificar la lógica para obtener la periodicidad del escenario, que quedará de la siguiente forma:

```
      list = getInitialComponentsFor1(scItem, translations, reactions, allComponents);
      if (list.size() > 0) { // The scenario has at least one responsibility
        interarrival = scItem.getStimulusValue();
        if (interarrival == null) {
          interarrival = DEFAULT_EVENT_INTERARRIVAL_TIME;
        }
        else
        {
          //The unit is seconds. Convert it to mls
          if (scItem.getStimulusUnit().equalsIgnoreCase("seconds"))
          {
            interarrival*=1000;
          }
        }
```

Finalmente en el mismo método se realiza una modificación similar a la hora de calcular el valor del tiempo máximo de ejecución del escenario:

```
        // The deadline comes from the response of the scenario
        double deadline = scItem.getMeasureValue();

        //Convert the deadline to mls if it is given in seconds
        if(scItem.getMeasureUnit().startsWith("sec"))
           deadline*=1000;
```

CORRECCIÓN DE LA INFORMACIÓN DE ERROR DURANTE EL ANÁLISIS DEL MODELO

Puesto que la implementación original del *plugin* genera excepciones a la hora de crear marcas de errores en la herramienta Eclipse durante el análisis del modelo generado para su evaluación por MAST, es necesario añadir otra vía que muestre las causas de los errores en caso de que los hubiera. Para ello se añade el siguiente código en el método *analyze* de la clase *ICMPerformanceReasoningFramework*[77].

```
      printLog(2, Level.INFO, "Interpretation procedure: " + assembly.getName() + " status: " +
statusInterpretation.getMessage());
      PerformanceModel tasks = rfperf.getPerformanceModel();
      if (rfperf.getNumInterpretationErrors() > 0) {
        //this.setAnalysisStatus(RF_ERROR);
         this.setAnalysisStatus(RF_WARNING); // Because it is just an error on the alternative and
not in the original architecture
                  //throw new ArchEException("Errors in the interpretation of ICM model
"+wrapper.getICMModel());
          printLog(2, Level.INFO, "Found " + rfperf.getNumInterpretationErrors() + " errors during
interpretation!");

        for (edu.cmu.sei.pacc.perf.util.Error e : rfperf.getInterpretationErrors()) {
          printLog(2, Level.INFO, e.toString());
        }
```

77 Por ejemplo, sin esta funcionalidad habría sido imposible, durante la realización de este trabajo, eliminar problemas como el de la generación de nombres de servicios duplicados, o la incapacidad de la herramienta de soportar determinados caracteres en los identificadores de las responsabilidades tales como el signo menos.

```
    }
```

CÓDIGO FUENTE CORREGIDO

A continuación se incluye el código completo de las dos clases modificadas en su versión final tras incorporar todas las correcciones.

Archivo ICMWrapper.java

```
/**
 * ******************************************************************************
 * Copyright (c) 2007, Software Engineering Institute. All rights reserved. This
 * program and the accompanying materials are made available under the terms of
 * the Common Public License v1.0 which accompanies this distribution, and is
 * available at http://www.eclipse.org/legal/cpl-v10.html
 *
 ******************************************************************************
 */
/**
 * An implementation of an ICM-based architectural view (used by performance
 * reasoning framework). This class is actually a wrapper for the EMF-based
 * classes defined by PACC for ICM.
 * <p>
 * Note: Instead of loading the ICM model from a file, the ICM model is
 * constructed out of ArchE responsibilities and user-specific relations for
 * them (this modification allows us to 'share' changes in the responsibility
 * structure between different reasoning frameworks)
 *
 * @author Andres Diaz-Pace
 */
package arche.example.reasoningframeworks.ICMPerformance;

import java.io.File;
import java.io.IOException;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Enumeration;
import java.util.HashMap;
import java.util.Hashtable;
import java.util.Iterator;
import java.util.List;

import org.eclipse.core.resources.IContainer;
import org.eclipse.core.resources.IFile;
import org.eclipse.core.resources.IResource;
import org.eclipse.core.resources.IWorkspaceRoot;
import org.eclipse.core.resources.ResourcesPlugin;
import org.eclipse.core.runtime.FileLocator;
import org.eclipse.core.runtime.IPath;
import org.eclipse.core.runtime.Path;
import org.eclipse.core.runtime.Platform;
import org.eclipse.emf.common.util.URI;
import org.eclipse.emf.ecore.resource.Resource;
import org.eclipse.emf.ecore.resource.ResourceSet;
import org.eclipse.emf.ecore.resource.impl.ResourceSetImpl;
import org.eclipse.emf.ecore.xmi.impl.XMIResourceFactoryImpl;

import arche.example.hibernate.ArchECoreResponsibilityStructure;
import arche.example.hibernate.ArchECoreView;
import arche.example.hibernate.vo.ArchEResponsibilityReactionRelationVO;
import arche.example.hibernate.vo.ArchEResponsibilityVO;
import arche.example.hibernate.vo.ArchEScenarioVO;
```

```
import arche.example.hibernate.vo.ArchETranslationRelationVO;
import arche.example.hibernate.vo.ArchEVersionVO;
import edu.cmu.sei.arche.ArchEException;
import edu.cmu.sei.arche.external.data.ArchEArchitecture;
import edu.cmu.sei.arche.external.data.ArchERelation;
import edu.cmu.sei.arche.external.data.ArchEResponsibility;
import edu.cmu.sei.arche.external.data.ArchEResponsibilityStructure;
import edu.cmu.sei.arche.external.data.ArchEScenario;
import edu.cmu.sei.arche.external.data.ArchEView;
import edu.cmu.sei.pacc.perf.ReasoningFramework;
import edu.cmu.sei.pacc.perf.icm.AssemblyInstance;
import edu.cmu.sei.pacc.perf.icm.Constant;
import edu.cmu.sei.pacc.perf.icm.Distribution;
import edu.cmu.sei.pacc.perf.icm.ElementInstance;
import edu.cmu.sei.pacc.perf.icm.IcmComponent;
import edu.cmu.sei.pacc.perf.icm.IcmFactory;
import edu.cmu.sei.pacc.perf.icm.IcmPackage;
import edu.cmu.sei.pacc.perf.icm.IcmService;
import edu.cmu.sei.pacc.perf.icm.Scenario;
import edu.cmu.sei.pacc.perf.icm.ServiceSourcePinIcm;
import edu.cmu.sei.pacc.perf.icm.SinkPinInstance;
import edu.cmu.sei.pacc.perf.icm.SinkPinMode;
import edu.cmu.sei.pacc.perf.icm.SourcePinInstance;
import edu.cmu.sei.pacc.perf.icm.SourcePinMode;
import edu.cmu.sei.pacc.perf.icm.Uniform;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;

public class ICMWrapper extends ArchECoreView implements ArchEView {

  private static final SourcePinMode DEFAULT_SOURCEPIN_MODE = SourcePinMode.SYNCH_LITERAL;
  private static final SinkPinMode DEFAULT_SINKPIN_MODE = SinkPinMode.MUTEX_LITERAL;

  private static final int DEFAULT_PRIORITY = 1;
  private static final double DEFAULT_CONNECTION_OVERHEAD = 0.0;
  private static final double DEFAULT_EVENT_INTERARRIVAL_TIME = 10;

  private AssemblyInstance icmModel = null;

  // This method is no longer used, because it is replaced by the method
  // below which already loads the AssemblyInstance by delegating in
  // the facade ReasoningFramework
  private static AssemblyInstance loadICM(String filename) {

    // Paste here the invocation to ICM EMF model
    ResourceSet resourceSet = new ResourceSetImpl();

    // Register the appropriate resource factory to handle all file extensions.
    resourceSet.getResourceFactoryRegistry().getExtensionToFactoryMap().put(Resource.Factory.Regist
ry.DEFAULT_EXTENSION,
            new XMIResourceFactoryImpl());

    // Register the package to ensure it is available during loading.
    resourceSet.getPackageRegistry().put(IcmPackage.eNS_URI, IcmPackage.eINSTANCE);

    URI fileURI = URI.createFileURI(new File(filename).getAbsolutePath());
    Resource resource = resourceSet.getResource(fileURI, true);
    AssemblyInstance assembly = (AssemblyInstance) resource.getContents().get(0);

    return (assembly);
  }

  protected static AssemblyInstance loadnewICM(String filename) {
    // There may be other ways of loading the file
    IWorkspaceRoot root = ResourcesPlugin.getWorkspace().getRoot();
    IResource resource = root.findMember(new Path(filename));
    IContainer container = (IContainer) resource;
```

```
    IFile ifile = container.getFile(new Path(filename));
    // This is the method provided by the ReasoningFramework API
    AssemblyInstance assembly = ReasoningFramework.loadConstructiveAssembly(ifile);
    return (assembly);
  }

  private static String removeSpaces(String withSpaces) {
    java.util.StringTokenizer t = new java.util.StringTokenizer(withSpaces, " ");
    StringBuffer result = new StringBuffer("");
    while (t.hasMoreTokens()) {
      result.append(t.nextToken());
      if (t.hasMoreTokens()) {
        result.append("_");
      }
    }
    return result.toString().replace("<", "").replace(">", "").replace("-", "_");
  }

  private static Distribution getUniformDistribution(IcmFactory icmFactory, double min, double max)
{
    Uniform uniform = icmFactory.createUniform();
    uniform.setMin(min);
    uniform.setMax(max);
    return (uniform);
  }

  private static Distribution getConstantDistribution(IcmFactory icmFactory, double c) {
    Constant constant = icmFactory.createConstant();
    constant.add(c);
    return (constant);
  }

  private static int _PRIORITY = 0;

  private static int getTaskPriority() {
    _PRIORITY++;
    return (_PRIORITY);
  }

  private static void resetTaskPriority() {
    _PRIORITY = 0;
  }

  private static AssemblyInstance createICM(String name, List<ArchEScenario> scenarios,
          List<ArchEResponsibility> responsibilities, List<ArchERelation> reactions,
          List<ArchERelation> translations) {

    //System.out.println("@@@@@CREATION OF ICM WITH REACTIONS: "+reactions.size());
    IcmFactory factory = IcmFactory.eINSTANCE;
    AssemblyInstance assembly = factory.createAssemblyInstance();
    assembly.setName(name);
    assembly.setConnectionOverhead(DEFAULT_CONNECTION_OVERHEAD);
    resetTaskPriority();

    String nameWithoutSpaces = null;

    // Rule 1: Create an ICM component for each responsibility
    IcmComponent comp = null;
    ArchEResponsibilityVO resp = null;
    Hashtable<String, IcmComponent> allComponents = new Hashtable<String, IcmComponent>();
          for  (Iterator<ArchEResponsibility>  itResponsibilities  =  responsibilities.iterator();
itResponsibilities.hasNext();) {
      resp = (ArchEResponsibilityVO) (itResponsibilities.next());
      comp = factory.createIcmComponent();
      nameWithoutSpaces = removeSpaces(resp.getName());
      comp.setName(nameWithoutSpaces);
      allComponents.put(nameWithoutSpaces, comp);
      assembly.getElements().add(comp);
```

```
    }

    // Rule 2: For each reaction relationship create corresponding source and sink pins
    // in the components associated to the responsibilities involved in the relationship
    IcmComponent component = null;
    SinkPinInstance pinSinkInstance = null;
    SourcePinInstance pinSourceInstance = null;
    ArchEResponsibilityReactionRelationVO reaction = null;
    List<String> connectedComponents = new ArrayList<String>();
     Map<ArchEScenarioVO, Map<String, SinkPinInstance>> reactSinksdb = new HashMap<ArchEScenarioVO,
Map<String, SinkPinInstance>>();
             Map<ArchEScenarioVO,   Map<String,   SourcePinInstance>>   initialSourcesdb   =   new
HashMap<ArchEScenarioVO, Map<String, SourcePinInstance>>();

    double avgExecutionTime = 0;
    Distribution uniform = null;

    for (Iterator<ArchEScenario> itScenarios = scenarios.iterator(); itScenarios.hasNext();;) {
      ArchEScenarioVO scItem = (ArchEScenarioVO) (itScenarios.next());
      Map<String, SinkPinInstance> reactSinks = new HashMap<String, SinkPinInstance>();
      Map<String, SourcePinInstance> initialSources = new HashMap<String, SourcePinInstance>();

      reactSinksdb.put(scItem, reactSinks);
      initialSourcesdb.put(scItem, initialSources);

      Set<String> respNames = getMappedResponsibilitiesNames(scItem, translations, reactions);

      System.out.println("DEBUG: scenario " + scItem + "respNames: " + respNames);

    // For connecting the pins within a component
      // Iterave over reactions to create sink pins
      for (Iterator<ArchERelation> it1 = reactions.iterator(); it1.hasNext();;) {
        reaction = (ArchEResponsibilityReactionRelationVO) (it1.next());

        nameWithoutSpaces = removeSpaces(reaction.getChild().getName());
        component = allComponents.get(nameWithoutSpaces);

        if (!respNames.contains(nameWithoutSpaces)) {
             System.out.println("DEBUG: Child doesn't apply for current scenario. Child Name:" +
nameWithoutSpaces);
          continue;
        }

      // This creates a sink pin for the child responsibility,
        // assuming that parent --> child maps to source_parent --> sink_child
        pinSinkInstance = factory.createSinkPinInstance();
        pinSinkInstance.setMode(DEFAULT_SINKPIN_MODE);
        pinSinkInstance.setPriority(getTaskPriority());
        pinSinkInstance.setName("psink_" + nameWithoutSpaces);

        try {
                                                                          avgExecutionTime       =
reaction.getChild().getDoubleParameter(ICMPerformanceReasoningFramework.PARAMETER_EXECUTION_TIME);
          uniform = getUniformDistribution(factory, 1.0, avgExecutionTime);
          pinSinkInstance.setExecTimeDistribution(uniform);
        } catch (ArchEException e) {
          // TODO Auto-generated catch block
          e.printStackTrace();
        }

        pinSinkInstance.setElementInstance(component);

      //Save the sink in the reactMap to latter connect it to the
        //reacting sources
        reactSinks.put(nameWithoutSpaces, pinSinkInstance);

      // This saves the names of those components that have at least one sink pin
        // connected, for later.
```

```
        if (!connectedComponents.contains(nameWithoutSpaces)) {
          connectedComponents.add(nameWithoutSpaces);
        }
      }

      // Iterate over reactions to process fhe source pins
      for (Iterator<ArchERelation> it1 = reactions.iterator(); it1.hasNext();) {
        reaction = (ArchEResponsibilityReactionRelationVO) (it1.next());

        nameWithoutSpaces = removeSpaces(reaction.getParent().getName());

        if (!respNames.contains(nameWithoutSpaces)) {
           System.out.println("DEBUG: Parent doesn't apply for current scenario. Parent Name: " +
nameWithoutSpaces);
          continue;
        }

        pinSourceInstance = initialSources.get(nameWithoutSpaces);

        // If the sourcePin doesn't already exits
        if (pinSourceInstance == null) {
          // This creates a source pin for the child responsibility
          pinSourceInstance = factory.createSourcePinInstance();
          pinSourceInstance.setMode(DEFAULT_SOURCEPIN_MODE);

          pinSourceInstance.setName("psource_" + nameWithoutSpaces);
          component = allComponents.get(nameWithoutSpaces);
          pinSourceInstance.setElementInstance(component);
        }

        String sinkName = removeSpaces(reaction.getChild().getName());

        // This connects the source for the sink to react to
        pinSinkInstance = reactSinks.get(sinkName);
        if (pinSinkInstance != null) {
          pinSourceInstance.getSinks().add(pinSinkInstance);
        }

      //Save the source pink for later bind to the service sink pin
        //in case this component is initial
        initialSources.put(nameWithoutSpaces, pinSourceInstance);

        //Try to find the sink that could invoke this source
        pinSinkInstance = reactSinks.get(nameWithoutSpaces);
        if (pinSinkInstance != null) {
          pinSinkInstance.getReactSources().add(pinSourceInstance);
        }
      }
    }

    // Rule 3: Create services for each of the scenarios, and then link them
    // to responsibilities (components) that have no incoming reactions (within the same scenario)
    ServiceSourcePinIcm serviceSourcePin = null;
    IcmService service = null;
    IcmComponent initialComponent = null;
    SinkPinInstance serviceSinkPin = null;
    ArchEScenarioVO scItem = null;
    List<IcmComponent> list = null;
    Scenario scenarioMode = null;
    int index = 1;
    Double interarrival = null;
    Distribution constant = null;

    for (Iterator<ArchEScenario> itScenarios = scenarios.iterator(); itScenarios.hasNext();) {
      scItem = (ArchEScenarioVO) (itScenarios.next());

      // Rule 4: Create Icm scenarios (modes) for the assembly
      scenarioMode = factory.createScenario();
```

```
      scenarioMode.setNumber(index);
      index++;
      nameWithoutSpaces = scItem.getId();
         scenarioMode.setName(nameWithoutSpaces); // Check if the number depends on the services
(Gabriel)!!
      assembly.getScenarios().add(scenarioMode);

      //list = getInitialComponentsFor(scItem,translations,markedComponents,allComponents);
      list = getInitialComponentsFor1(scItem, translations, reactions, allComponents);
      if (list.size() > 0) { // The scenario has at least one responsibility
        interarrival = scItem.getStimulusValue();
        if (interarrival == null) {
          interarrival = DEFAULT_EVENT_INTERARRIVAL_TIME;
        } else {
          //The unit is seconds. Convert it to mls
          if (scItem.getStimulusUnit().equalsIgnoreCase("seconds")) {
            interarrival *= 1000;
          }
        }

        constant = getConstantDistribution(factory, interarrival);

        // Create a service for the scenario
        service = factory.createIcmService();
        service.setPriority(getTaskPriority());
        nameWithoutSpaces = removeSpaces(scItem.getFactId());
        service.setName("Service_" + nameWithoutSpaces);
        assembly.getElements().add(service);

        // Create a source pin for the service
        serviceSourcePin = factory.createServiceSourcePinIcm();

        // The deadline comes from the response of the scenario
        double deadline = scItem.getMeasureValue();

        //Convert the deadline to mls if it is given in seconds
        if (scItem.getMeasureUnit().startsWith("sec")) {
          deadline *= 1000;
        }

        serviceSourcePin.setDeadline(deadline);
        serviceSourcePin.setMode(DEFAULT_SOURCEPIN_MODE);
        avgExecutionTime = ICMPerformanceReasoningFramework.DEFAULT_EXECUTION_TIME;
        serviceSourcePin.setEventDistribution(constant);
        uniform = getUniformDistribution(factory, 1.0, avgExecutionTime);
        serviceSourcePin.setExecTimeDistribution(uniform);
        serviceSourcePin.setName("psource_service");
        serviceSourcePin.setElementInstance(service);

        // Create a sink pin for every component that is "initial" for the scenario
        for (Iterator<IcmComponent> itComponents = list.iterator(); itComponents.hasNext();;) {
          initialComponent = itComponents.next();

          // This saves the names of those components that have at least one sink pin
          // connected, for later.
          if (!connectedComponents.contains(initialComponent.getName()));
          connectedComponents.add(initialComponent.getName());

          serviceSinkPin = factory.createSinkPinInstance();
          serviceSinkPin.setMode(DEFAULT_SINKPIN_MODE);
          serviceSinkPin.setPriority(1);
          //avgExecutionTime = ICMPerformanceReasoningFramework.DEFAULT_EXECUTION_TIME;
          avgExecutionTime = retrieveExecutionTimeFor(initialComponent, responsibilities);
          uniform = getUniformDistribution(factory, 1.0, avgExecutionTime);
          serviceSinkPin.setExecTimeDistribution(uniform);
          serviceSinkPin.setName("psink_" + service.getName());
          serviceSinkPin.setElementInstance(initialComponent);
```

```
          // This connects the source for the sink to react to
          serviceSourcePin.getSinks().add(serviceSinkPin);

                                                        SourcePinInstance     componentSource     =
initialSourcesdb.get(scItem).get(initialComponent.getName());

          if (componentSource != null) {
            System.out.print("DEBUG! bound source for " + initialComponent.getName());
            serviceSinkPin.getReactSources().add(componentSource);
          } else {
                          System.out.print("ERROR!  couldnt  find  sourcePin  for  component  "  +
initialComponent.getName());
          }
        }

        serviceSourcePin.getScenarios().add(scenarioMode);
      } // End if list.size() > 0
    }

    // Rule 5: Create a default scenario for all the remaining "unconnected" components
//                    if (allComponents.size() > connectedComponents.size()) {
//                            scenarioMode = factory.createScenario();
//                            scenarioMode.setNumber(index);
//                            index++;
//                            nameWithoutSpaces = "background";
//                            scenarioMode.setName(nameWithoutSpaces);  //  Check  if  the  number
depends on the services (Gabriel)!!
//                            assembly.getScenarios().add(scenarioMode);
//
//                            interarrival = DEFAULT_EVENT_INTERARRIVAL_TIME;
//                            constant = getConstantDistribution(factory,interarrival);
//
//                            // Create a service for the default scenario
//                            service = factory.createIcmService();

//                            service.setPriority(getTaskPriority());
//                            service.setName("Service_background");
//                            assembly.getElements().add(service);
//
//                            // Create a source pin for the service
//                            serviceSourcePin = factory.createServiceSourcePinIcm();
//                            // The deadline comes from the response of the scenario
//                            serviceSourcePin.setDeadline(100.0); // Some default long value
//                            serviceSourcePin.setMode(DEFAULT_SOURCEPIN_MODE);
//                            avgExecutionTime                                                    =
ICMPerformanceReasoningFramework.DEFAULT_EXECUTION_TIME;
//                            serviceSourcePin.setEventDistribution(constant);
//                            uniform = getUniformDistribution(factory,1.0, avgExecutionTime);
//                            serviceSourcePin.setExecTimeDistribution(uniform);
//                            serviceSourcePin.setName("psource_service");
//                            serviceSourcePin.setElementInstance(service);
//
//                            for(Enumeration<IcmComponent> itComponents = allComponents.elements();
itComponents.hasMoreElements();;) {
//                                    initialComponent = itComponents.nextElement();
//                                    // This means the component is isolated
//                                    if  (!connectedComponents.contains(initialComponent.getName()))
{
//                                            serviceSinkPin = factory.createSinkPinInstance();
//                                            serviceSinkPin.setMode(DEFAULT_SINKPIN_MODE);
//                                            serviceSinkPin.setPriority(getTaskPriority());
//                                            //avgExecutionTime                                  =
ICMPerformanceReasoningFramework.DEFAULT_EXECUTION_TIME;
//                                            avgExecutionTime                                  =
retrieveExecutionTimeFor(initialComponent, responsibilities);
//                                            uniform      =      getUniformDistribution(factory,1.0,
avgExecutionTime);
//                                            serviceSinkPin.setExecTimeDistribution(uniform);
```

```
//                                          serviceSinkPin.setName("psink_" + service.getName());
//                                          serviceSinkPin.setElementInstance(initialComponent);
//
//                                          // This connects the source for the sink to react to
//                                          serviceSourcePin.getSinks().add(serviceSinkPin);
//                                          serviceSinkPin.getLinkSources().add(serviceSourcePin);

//                                    }
//                              }
//
//                    }
                for    (Enumeration<IcmComponent>    itComponents    =    allComponents.elements();
itComponents.hasMoreElements();;) {
      initialComponent = itComponents.nextElement();
      initialComponent.setName("IcmC_" + initialComponent.getName());
    }

    // This prints the assembly just created, for testing purposes
//                    System.out.println("ASSEMBLY ==> "+assembly.getName());
//                    ElementInstance componenti = null;
//                    PinInstance pini = null;
//                    for (Iterator it3 = assembly.getElements().iterator(); it3.hasNext();;) {
//                           componenti = (ElementInstance)(it3.next());
//                           System.out.println("    Component: "+ componenti.getName());
//                           for (Iterator it4 = componenti.getPins().iterator(); it4.hasNext();;) {
//                                  pini = (PinInstance)(it4.next());
//                                  System.out.println("        Pin: "+ pini.getName());

//                           }
//                    }
    return (assembly);
  }

  private static double retrieveExecutionTimeFor(
          IcmComponent initialComponent, List<ArchEResponsibility> resps) {
    ArchEResponsibilityVO respItem = null;
               for    (Iterator<ArchEResponsibility>    itResponsibilities    =    resps.iterator();
itResponsibilities.hasNext();;) {
      respItem = (ArchEResponsibilityVO) (itResponsibilities.next());
      if (removeSpaces(respItem.getName()).equals(initialComponent.getName())) {
        try { // This is the executionTime associated to the original responsibility
                                                                                          return
(respItem.getDoubleParameter(ICMPerformanceReasoningFramework.PARAMETER_EXECUTION_TIME));
        } catch (ArchEException e) {
          // TODO Auto-generated catch block
          e.printStackTrace();
          return (0);
        }
      }
    } // Otherwise, return the default value
    return (ICMPerformanceReasoningFramework.DEFAULT_EXECUTION_TIME);

  }

  private static List<IcmComponent> getInitialComponentsFor(ArchEScenarioVO scenario,
          List<ArchERelation> translations, List<String> listOfComponents,
          Hashtable<String, IcmComponent> allComponents) {
    List<IcmComponent> result = new ArrayList<IcmComponent>();
    ArchETranslationRelationVO trItem = null;
    String componentKey = null;
    for (Iterator<ArchERelation> itRelations = translations.iterator(); itRelations.hasNext();;) {
      trItem = (ArchETranslationRelationVO) (itRelations.next());
      ArchEScenarioVO parent = trItem.getParent();
      if ((parent != null)
              && parent.equals(scenario)) {
        componentKey = removeSpaces(trItem.getChild().getName());
        if (!listOfComponents.contains(componentKey)) {
          result.add(allComponents.get(componentKey));
```

```
        }
      }
    }
    return (result);
  }

  private static Set<String> getMappedResponsibilitiesNames(ArchEScenarioVO scenario,
          List<ArchERelation> translations, List<ArchERelation> reactions) {
    Set<String> mappedResponsibilities = new HashSet<String>();

    ArchETranslationRelationVO trItem = null;
    for (Iterator<ArchERelation> itRelations = translations.iterator(); itRelations.hasNext();) {
      trItem = (ArchETranslationRelationVO) (itRelations.next());
      ArchEScenarioVO parent = trItem.getParent();
      if ((parent != null) && parent.equals(scenario)) {
        mappedResponsibilities.add(removeSpaces(trItem.getChild().getName()));
      }
    }
    return mappedResponsibilities;
  }

  private static List<ArchEResponsibilityVO> getInitialResponsibilities(ArchEScenarioVO scenario,
          List<ArchERelation> translations, List<ArchERelation> reactions) {

    List<ArchEResponsibilityVO> result = new ArrayList<ArchEResponsibilityVO>();

    List<ArchEResponsibilityVO> mappedResponsibilities = new ArrayList<ArchEResponsibilityVO>();

    ArchETranslationRelationVO trItem = null;
    for (Iterator<ArchERelation> itRelations = translations.iterator(); itRelations.hasNext();) {
      trItem = (ArchETranslationRelationVO) (itRelations.next());
      ArchEScenarioVO parent = trItem.getParent();
      if ((parent != null) && parent.equals(scenario)) {
        mappedResponsibilities.add(trItem.getChild());
      }
    }

    ArchEResponsibilityVO respItem = null;
    ArchEResponsibilityReactionRelationVO reactionItem = null;
    boolean isInitial = true;
          for   (Iterator<ArchEResponsibilityVO>   itResps   =   mappedResponsibilities.iterator();
itResps.hasNext();) {
      respItem = itResps.next();
      isInitial = true;
      for (Iterator<ArchERelation> itReactions = reactions.iterator(); itReactions.hasNext();) {
        reactionItem = (ArchEResponsibilityReactionRelationVO) (itReactions.next());
        if (reactionItem.getChild().equals(respItem)
                && mappedResponsibilities.contains(reactionItem.getParent())) {
          isInitial = false;
        }
      }
      if (isInitial) {
        result.add(respItem);
      }
    }

    return (result);
  }

  private static List<IcmComponent> getInitialComponentsFor1(ArchEScenarioVO scenario,
          List<ArchERelation> translations, List<ArchERelation> reactions,
          Hashtable<String, IcmComponent> allComponents) {

    List<IcmComponent> result = new ArrayList<IcmComponent>();
    String componentKey = null;

       List<ArchEResponsibilityVO>  initialResponsibilities  =  getInitialResponsibilities(scenario,
translations, reactions);
```

```
    ArchEResponsibilityVO respItem = null;
          for   (Iterator<ArchEResponsibilityVO>   itResps   =   initialResponsibilities.iterator();
itResps.hasNext();;) {
      respItem = itResps.next();
      componentKey = removeSpaces(respItem.getName());
      result.add(allComponents.get(componentKey));
    }

    return (result);
  }

  public ICMWrapper(ArchEArchitecture architecture) {
    super(architecture);
    icmModel = null;
  }

  public void setInputInformation(String name, List<ArchEScenario> scenarios,
          ArchEResponsibilityStructure responsibilityStructure) {

    List<ArchEResponsibility> responsibilities = responsibilityStructure.getResponsibilities();
          ArchECoreResponsibilityStructure   coreStructure   =   ((ArchECoreResponsibilityStructure)
responsibilityStructure);
    String reactionTypeVO = ArchEResponsibilityReactionRelationVO.class.getName();
    List<ArchERelation> reactions = coreStructure.getRelations(reactionTypeVO);
    String translationTypeVO = ArchETranslationRelationVO.class.getName();
    List<ArchERelation> translations = coreStructure.getRelations(translationTypeVO);

    icmModel = createICM(name, scenarios, responsibilities, reactions, translations);
    // We assume always the first scenario in the ICM file is the one to analyze
    icmModel.setSourceFile(this.saveICMAssembly(icmModel));
  }

  // TODO: This filename should be configurable from the outside
  // private static final String PATH_ICMFILE = "C://eclipse33/eclipse/workspace/ArchE External RF
Examples/";
  private String saveICMAssembly(AssemblyInstance assembly) {

    // configure this RF with the configuration file
    try {
            URL  url  =  FileLocator.find(Platform.getBundle("SEI.ArchE.External.Examples"),  new
Path("/temp"), null);
      String tempPathName = FileLocator.resolve(url).getPath();

      ResourceSet resourceSet = new ResourceSetImpl();

      IWorkspaceRoot workspaceRoot = ResourcesPlugin.getWorkspace().getRoot();
           //IPath  path  =  new  Path(workspaceRoot.getFullPath().toString()+"/"+assembly.getName()
+".icm");
      IPath path = new Path(tempPathName + assembly.getName() + ".icm");

      IFile xmlFile = workspaceRoot.getFile(path);

      // Get the URI of the model file
      URI fileURI = URI.createFileURI(path.toString());

      // Create a resource for this file.
      Resource resource = resourceSet.createResource(fileURI);

      // Add the assembly to the resource contents
      resource.getContents().add(assembly);

      // Save the contents of the resource to the file system.
      String xmlAssemblyFile = null;
      try {
        //resource.save(null);
        resource.save(Collections.EMPTY_MAP);
      } catch (IOException e) {
      }
```

```
      xmlAssemblyFile = fileURI.toFileString();
                            //System.out.println("  ICM FILE GENERATED ___>>> "+xmlAssemblyFile);

      //return (xmlAssemblyFile);
      return (path.toPortableString());

    } catch (MalformedURLException e1) {
      e1.printStackTrace();
    } catch (IOException e1) {
      e1.printStackTrace();
    }

    return null; // for error
  }

  public boolean clearICMAssembly() {
    File icmFile = new Path(icmModel.getSourceFile()).toFile();
    boolean ok = icmFile.delete();
    return (ok);
  }

  public AssemblyInstance getICMModel() {
    return (icmModel);
  }

  public String getServicePinNameFor(ArchEScenario scenario) {

    ElementInstance elem = null;
    String nameWithoutSpaces = null;

            for   (Iterator<ElementInstance>   itComponents   =   icmModel.getElements().iterator();
itComponents.hasNext();) {
      elem = itComponents.next();

      if (elem instanceof IcmService) {
        nameWithoutSpaces = "Service_" + removeSpaces(((ArchEScenarioVO) scenario).getFactId());
        nameWithoutSpaces = nameWithoutSpaces + ".psource_service";
        return (nameWithoutSpaces.toLowerCase());
      }
    }
    return (null);
  }

  // This method is necessary to know the evaluation procedure to apply on the ICM model
  public HashMap<Integer, Boolean> getScenarioMapFor(ArchEScenario scenario) {
    HashMap<Integer, Boolean> map = new HashMap<Integer, Boolean>();

    //String scenarioDescriptor = null;
    if (scenario == null) { // It means to activate all the scenario modes
                for   (Iterator<Scenario>   itScenarioModes   =   icmModel.getScenarios().iterator();
itScenarioModes.hasNext();) {
        map.put(itScenarioModes.next().getNumber(), new Boolean(true));
      }
      return (map);
    }

    // Otherwise, a specific scenario gets activated
    Scenario scenarioMode = null;
    String archeScenarioName = ((ArchEScenarioVO) scenario).getId();
              for   (Iterator<Scenario>   itScenarioModes   =   icmModel.getScenarios().iterator();
itScenarioModes.hasNext();) {
      scenarioMode = itScenarioModes.next();
      //scenarioDescriptor = scenarioMode.getName();
      if (scenarioMode.getName().equals(archeScenarioName))// The scenario to be analyzed
      {
        map.put(scenarioMode.getNumber(), new Boolean(true));
      } else {
```

```
        map.put(scenarioMode.getNumber(), new Boolean(false));
      }

    }
    return (map);
  }

  //---- The remaining methods are unimplemented, since this view doesn't need to be saved/restored
  // from the database but rather it is always (re-)generated from the responsibility structure
  @Override
  public void delete() throws ArchEException {
    // TODO Auto-generated method stub

  }

  @Override
  public void restore() throws ArchEException {
    // TODO Auto-generated method stub

  }

  @Override
  public void save() throws ArchEException {
    // TODO Auto-generated method stub

  }

  @Override
  public void saveAs(ArchEVersionVO newVersion) throws ArchEException {
    // TODO Auto-generated method stub

  }

}
```

Archivo ICMPerformanceReasoningFramework.java

```
/**
 * ********************************************************************************
 * Copyright (c) 2007, Software Engineering Institute. All rights reserved. This
 * program and the accompanying materials are made available under the terms of
 * the Common Public License v1.0 which accompanies this distribution, and is
 * available at http://www.eclipse.org/legal/cpl-v10.html
 *
 * ********************************************************************************
 */
package arche.example.reasoningframeworks.ICMPerformance;

/**
 * An integration of the schedulability analysis for ICM models used by PACC.
 * This is an 'analysis-only' reasoning framework, because it just analyzes the
 * architecture for performance but it doesn't support tactics.
 * <p>
 * Note: This example is still under experimentation.
 *
 * @author Andres Diaz-Pace
 */
import java.io.IOException;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.ArrayList;
import java.util.Enumeration;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
```

```
import java.util.logging.Level;

import org.eclipse.core.runtime.FileLocator;
import org.eclipse.core.runtime.IPath;
import org.eclipse.core.runtime.IStatus;
import org.eclipse.core.runtime.Path;
import org.eclipse.core.runtime.Platform;
//import org.eclipse.swt.widgets.Display;
//import org.eclipse.ui.console.MessageConsoleStream;

import arche.example.hibernate.ArchECoreResponsibilityStructure;
import arche.example.hibernate.vo.ArchEResponsibilityReactionRelationVO;

import edu.cmu.sei.arche.ArchEException;
import edu.cmu.sei.arche.external.data.ArchEArchitecture;
import edu.cmu.sei.arche.external.data.ArchERelation;
import edu.cmu.sei.arche.external.data.ArchERequirementModel;
import edu.cmu.sei.arche.external.data.ArchEResponsibility;
import edu.cmu.sei.arche.external.data.ArchEResponsibilityStructure;
import edu.cmu.sei.arche.external.data.ArchEScenario;
import edu.cmu.sei.arche.external.data.ArchEView;
import edu.cmu.sei.arche.external.reasoningframework.ArchEAnalysisProblem;
import edu.cmu.sei.arche.external.reasoningframework.ArchEAnalysisResult;
import edu.cmu.sei.arche.external.reasoningframework.ArchEEvaluationResult;
import edu.cmu.sei.arche.external.reasoningframework.ArchEReasoningFramework;
import edu.cmu.sei.arche.external.reasoningframework.ArchETryTacticResult;
import edu.cmu.sei.arche.external.reasoningframework.ArchEUserQuestion;

import edu.cmu.sei.pacc.perf.ConsolePrinter;
import edu.cmu.sei.pacc.perf.PerformanceRFPlugin;
import edu.cmu.sei.pacc.perf.ReasoningFramework;
import edu.cmu.sei.pacc.perf.eval.EvaluationProcedure;
import edu.cmu.sei.pacc.perf.eval.mast.MastOptions;
import edu.cmu.sei.pacc.perf.icm.AssemblyInstance;
import edu.cmu.sei.pacc.perf.model.PerformanceModel;
import java.io.File;

public class ICMPerformanceReasoningFramework extends ArchEReasoningFramework {

  //---- Names of parameters used by this reasoning framework
  public static final String PARAMETER_EXECUTION_TIME = "P_ExecutionTime";

  //---- Default values of parameters
  public static final double DEFAULT_EXECUTION_TIME = 1.0;
  private static final double MIN_EXECUTION_TIME = 0.0;
  private static final double MAX_EXECUTION_TIME = 10000.0;

  protected static final double INVALID_RESPONSE = Double.MAX_VALUE / 2; // An arbitrary high value

  //---- Questions and warning defined in the question file by this reasoning framework
  public static final String FIX_ANALYSIS_PARAMETERS_WARNING = "fixAnalysisParameters";

  //---- Performance-specific types of errors
  private static final int EXECUTION_TIME_OUT_OF_LIMITS_ERROR = 1;

  private boolean enableMASTViewer = false;

  public ICMPerformanceReasoningFramework() {
    super();

    // set data provider
    setDataProvider(new ICMPerformanceRFDataProvider());

    // configure this RF with the configuration file
    try {
            URL  url  =  FileLocator.find(Platform.getBundle("SEI.ArchE.External.Examples"),  new
Path("/config"), null);
      String installPathName = FileLocator.resolve(url).getPath();
```

```
      String rfconfigFullPath = installPathName + "perficm-rfconfig.xml";
      configure(rfconfigFullPath);
    } catch (MalformedURLException e1) {
      e1.printStackTrace();
    } catch (IOException e1) {
      e1.printStackTrace();
    }
  }

  /**
   * Note that any transformations for analysis must be done within this method
   * (e.g., ArchE to ICM, ICM to ICMPerf Performance Model, etc.)
   *
   * @param architecture current architecture model (assumed consistent)
   * @param currentScenario scenario to be analyzed on the architecture
   * @throws ArchEException
   */
  @Override
  public ArchEEvaluationResult analyze(ArchEArchitecture architecture,
          ArchEScenario currentScenario) throws ArchEException {

    ICMWrapper wrapper = (ICMWrapper) (architecture.getView());

                printLog(2,    Level.INFO,    "Running    analysis    for    the    scenario    \""    +
currentScenario.getDescription() + "\"");

    double response = INVALID_RESPONSE;//Double.NaN;
    double measureVal = getMeasureValueForScenario(currentScenario);

    if (this.isResponsibilityStructureValid()) {

      // This code calls the API for the ICMPerformance ReasoningFramework developed by PACC
      ReasoningFramework rfperf = new ReasoningFramework();

      // Transformation of the ICM to tasks
      AssemblyInstance assembly = wrapper.getICMModel();
      File f = new File(assembly.getSourceFile());
      if (!f.exists()) {
        System.out.println("File: " + f.getAbsolutePath() + "doesn't exits");
      }
      HashMap<Integer, Boolean> map = wrapper.getScenarioMapFor(null); // Null means to analyze all
the scenarios at once!!
      IStatus statusInterpretation = rfperf.doInterpretation(assembly, map);
       printLog(2, Level.INFO, "Interpretation procedure: " + assembly.getName() + " status: " +
statusInterpretation.getMessage());
      PerformanceModel tasks = rfperf.getPerformanceModel();
      if (rfperf.getNumInterpretationErrors() > 0) {
        //this.setAnalysisStatus(RF_ERROR);
         this.setAnalysisStatus(RF_WARNING); // Because it is just an error on the alternative and
not in the original architecture
                  //throw   new   ArchEException("Errors   in   the   interpretation   of   ICM   model
"+wrapper.getICMModel());
         printLog(2, Level.INFO, "Found " + rfperf.getNumInterpretationErrors() + " errors during
interpretation!");

        for (edu.cmu.sei.pacc.perf.util.Error e : rfperf.getInterpretationErrors()) {
          printLog(2, Level.INFO, e.toString());
        }
      }

      Double worstCaseAvg = null;
      if (statusInterpretation.isOK()) {

        // Evaluation of the task model
                                                List<EvaluationProcedure>      procedures      =
PerformanceRFPlugin.getDefault().getEvaluationProcedures();
                            // I simply select any of the available procedures (hopefully, it
works)
```

```
        // at 0: Simulation (Extend)
        // at 1: Closed form (SS latency)
        // at 2: MAST v1.3.6
        // at 3: SIM-MAST

        // Application of MAST
        EvaluationProcedure evalproc = (EvaluationProcedure) (procedures.get(2));
        MastOptions options = new MastOptions();
        options.useMastResultsViewer = enableMASTViewer;
        options.templateArgs.setModel(tasks);

        // Note that if the filename for the ICM assembly is not set, the evaluation doesn't work
        String icmFilename = assembly.getSourceFile();
        printLog(2, Level.INFO, "ICM filename: " + icmFilename);
        if (icmFilename == null) {
          printLog(2, Level.INFO, "ICM filename: null");
        }
              IStatus statusEvaluation = rfperf.doEvaluation(evalproc, tasks, icmFilename, new
MyDefaultConsolePrinter(), options);
           printLog(2, Level.INFO, "Evaluation procedure: " + evalproc.getName() + " status: " +
statusEvaluation.getMessage());
                            //printLog(2,  Level.INFO,  "Deleting  temporal  ICM  file:  "  +
icmFilename+" ... "+wrapper.clearICMAssembly());

                            // If the system is NOT schedulable, the status is still OK but the
        // the MAST file shouldn't be read (because it's empty)
        if (statusEvaluation.isOK()) {
          // Access the MAST results from a .csv file
          IPath modelPath = new Path(icmFilename);
          String baseFilename = modelPath.removeFileExtension().lastSegment();
                                                                    IPath       mastPath       =
modelPath.removeLastSegments(1).append("performance").append(baseFilename);
          printLog(3, Level.INFO, "Generated MAST filename: " + mastPath.toString() + "_mast.csv");
                  //worstCaseAvg  =  CSVMastFileReader.getWorstCaseAvgParameter(mastPath.toString()
+"_mast.csv");
          String k = wrapper.getServicePinNameFor(currentScenario);
           worstCaseAvg = CSVMastFileReader.getWorstCaseParameter(mastPath.toString() + "_mast.csv",
k);
          printLog(3, Level.INFO, "MAST result = " + worstCaseAvg + "  ( " + k + " )");
        }
        if (worstCaseAvg == null) { // Something failed with the analysis
          response = INVALID_RESPONSE; // This is invalid number for computations
            printLog(2, Level.INFO, "Evaluation result = " + response + " (no improvements)" + "
reference= " + measureVal);
          //this.setAnalysisStatus(RF_ERROR);
          this.setAnalysisStatus(RF_WARNING); // Because it is just an error on the alternative and
not in the original architecture
        } else {
          response = worstCaseAvg;
          this.setAnalysisStatus(RF_OK);
          if (worstCaseAvg < measureVal) {
                printLog(2, Level.INFO, "Evaluation result = " + response + " (improved)" + "
reference= " + measureVal);
          }
        }

      } else { // the responsibility structure is OK, but the interpretation/evaluation had errors
        printLog(3, Level.INFO, "Interpretation: Error(s) ocurred when interpreting ICM Model ... "
+ wrapper.getICMModel());
        printLog(3, Level.INFO, "Evaluation result = " + response);
        //this.setAnalysisStatus(RF_ERROR);
         this.setAnalysisStatus(RF_WARNING); // Because it is just an error on the alternative and
not in the original architecture
      }
    } else { // The responsibility structure had problems, so analysis couldn't be performed
       printLog(3, Level.INFO, "Pre-Analysis: Error(s) ocurred when checking the responsibility
structure ...");
      printLog(3, Level.INFO, "Analysis result = " + response);
```

```
      //this.setAnalysisStatus(RF_ERROR);
      this.setAnalysisStatus(RF_WARNING); // Because it is just an error on the alternative and not
in the original architecture
    }

    ArchEEvaluationResult evaluationResult = new ArchEEvaluationResult();
    evaluationResult.setScenario(currentScenario.getFactId());

    // Set the difference with the value from the previous round of analysis (if any)
                                      ArchEAnalysisResult         previousAnalysisResult         =
this.restoreAnalysisResult(currentScenario.getFactId());
    double diff = 0.0;
    if (previousAnalysisResult != null) {
      diff = response - previousAnalysisResult.getValue();
    }
    evaluationResult.setChange(diff);

    if (response <= measureVal) // This estimation of utility is just a possible example
    {
      evaluationResult.setUtility(1.0);
    } else {
      evaluationResult.setUtility(0.0);
    }

    evaluationResult.setResult(response);
    return (evaluationResult);
  }

  /**
   * It simply invokes the same analysis than before, returning no tactics
   *
   * @param architecture current architecture model (assumed consistent)
   * @param currentScenario scenario to be analyzed on the architecture
   * @param outTactics the list of suggested tactics (output parameter)
   * @throws ArchEException
   */
  @Override
  public ArchEAnalysisResult analyzeAndSuggest(
          ArchEArchitecture architecture, ArchEScenario currentScenario,
          List<ArchETryTacticResult> outTactics) throws ArchEException {

    // It invokes the analyze() method above and returns no candidate tactics
    enableMASTViewer = true;
    ArchEEvaluationResult evaluationResult = this.analyze(architecture, currentScenario);
    enableMASTViewer = false;
    double worstCaseAvg = evaluationResult.getResult();

    if (this.getAnalysisStatus() == RF_WARNING) {
      this.setAnalysisStatus(RF_ERROR); // Because it's the analysis of the current architecture
    }
    double measureVal = getMeasureValueForScenario(currentScenario);

    ArchEAnalysisResult analysisResult = new ArchEAnalysisResult();
    analysisResult.setValue(worstCaseAvg);
    analysisResult.setOwner(currentScenario.getFactId());
    analysisResult.setReasoningFramework(this.getID());
    if (this.isAnalysisValid() && (worstCaseAvg <= measureVal)) {
      analysisResult.setSatisfied(true);
    } else {
      analysisResult.setSatisfied(false);
    }
    analysisResult.setQuality(this.getQuality());

    // Set the value from the previous round of analysis (if any)
                                      ArchEAnalysisResult         previousAnalysisResult         =
this.restoreAnalysisResult(currentScenario.getFactId());
    if (previousAnalysisResult != null) {
      analysisResult.setOldValue(previousAnalysisResult.getValue());
```

```
    } else {
      analysisResult.setOldValue(0.0);
    }

    return (analysisResult);
  }

  private double getMeasureValueForScenario(ArchEScenario currentScenario) {
    double measureVal = 0.0; // The minimum worst case execution time for performance

    if (currentScenario.getMeasureValue() != null) {
      measureVal = currentScenario.getMeasureValue();

      //If the measure is in second, convert it to mls
      if (currentScenario.getMeasureUnit().startsWith("sec")) {
        measureVal *= 1000;
      }
    }

    return measureVal;
  }

  @Override
  public boolean applySuggestedTactic(ArchEArchitecture architecture,
          ArchETryTacticResult suggestedTactic)
          throws ArchEException {
    // TODO Auto-generated method stub
    return false;
  }

  @Override
  public boolean applyTacticByUserQuestion(
          ArchEArchitecture architecture, ArchEUserQuestion userQuestion)
          throws ArchEException {
    // TODO Auto-generated method stub
    return false;
  }

  /**
   * Every time it's invoked, we create new assemblies for the ICM view out of
   * the responsibilities and dependencies among them
   *
   * @param view an empty ICM view
   * @param responsibilityStructure the current responsibility structure
   * @param requirementModel the current requirement model (not used here)
   * @throws ArchEException
   */
  @Override
  public boolean initializeView(ArchEView view,
          ArchEResponsibilityStructure responsibilityStructure,
          ArchERequirementModel requirementModel)
          throws ArchEException {

    boolean changed = false;

    ICMWrapper myICMView = (ICMWrapper) view;
    printLog(1, Level.INFO, "Initialize an architectural view");

    List<ArchEScenario> scenarios = requirementModel.getScenariosByReasoningFramework(this);

              // Create an ICM assembly instance, based on the responsibilities, and the sets of
    // reactions and interactions among them
    String icmName = myICMView.getParent().getCurrentVersion().getVersionName();
    myICMView.setInputInformation(icmName, scenarios, responsibilityStructure);
    changed = true;

    printLog(2, Level.INFO, "Creating temporal ICM file ... " + icmName + ".icm");
    printLog(1, Level.INFO, "After building \"" + myICMView + "\"");
```

```
    return (changed);

  }

  @Override
  public ArchEUserQuestion describeTactic(
          ArchETryTacticResult tactic)
          throws ArchEException {
    // TODO Auto-generated method stub
    return null;
  }

  /**
   * Every time it's invoked, we check that the parameters of responsibilities
   * and relationships are appropriately configured
   *
   * @param requirementModel the current requirement model
   * @param responsibilityStructure the current responsibility structure
   * @throws ArchEException
   */
  @Override
  public boolean checkRFDependencies(ArchERequirementModel requirementModel,
          ArchEResponsibilityStructure responsibilityStructure)
          throws ArchEException {

    boolean changed = false;
           ArchECoreResponsibilityStructure   coreStructure   =   ((ArchECoreResponsibilityStructure)
responsibilityStructure);

    // Check for missing parameters in the responsibilities
    double executionTime = DEFAULT_EXECUTION_TIME;
    ArchEResponsibility itemResp = null;
      for (Iterator<ArchEResponsibility> itResps = coreStructure.getResponsibilities().iterator();
itResps.hasNext();) {
      itemResp = itResps.next();

      if (!itemResp.hasParameter(PARAMETER_EXECUTION_TIME)) {
                              // In case no executionTime parameter exists, the responsibility
        // is modified to include this parameter
        itemResp.defineParameter(PARAMETER_EXECUTION_TIME, executionTime);
        changed = true;
      }

      // The value executionTime must be within the range [min .. max]
      double value = itemResp.getDoubleParameter(PARAMETER_EXECUTION_TIME);
      if ((value > MAX_EXECUTION_TIME) || (value < MIN_EXECUTION_TIME)) {

        // Error reporting
        String message = "Parameter " + PARAMETER_EXECUTION_TIME + " must be in range ["
                + MIN_EXECUTION_TIME + " - "
                + MAX_EXECUTION_TIME + "] (value= " + value + " !?)";
        ArchEAnalysisProblem problem = new ArchEAnalysisProblem(message, itemResp);
                this.setResponsibilityStructureStatus(RF_ERROR,  EXECUTION_TIME_OUT_OF_LIMITS_ERROR,
problem);

        this.printLog(1, Level.INFO, message);
        //throw new ArchEException(message);
      }

    } // End iteration over responsibilities

    // Check and delete orphan reactions among responsibilities
    String reactionTypeVO = ArchEResponsibilityReactionRelationVO.class.getName();
    ArchEResponsibilityReactionRelationVO respReactionVO = null;

                                for        (Iterator<ArchERelation>        itDependencies        =
coreStructure.getRelations(reactionTypeVO).iterator(); itDependencies.hasNext();) {
```

```
      respReactionVO = (ArchEResponsibilityReactionRelationVO) (itDependencies.next());
      if (!coreStructure.containResponsibility(respReactionVO.getParent())
              || !coreStructure.containResponsibility(respReactionVO.getChild())) {
        // This is an orphan dependency that must be deleted from the responsibility structure
        coreStructure.deleteRelation(respReactionVO);
        changed = true;
      }
    }

              // So far, we don't handle relation properties, so this consistency checking does
nothing
    return (changed);
  }

//     public boolean mergeResponsibilityRelationships(
//                   ArchEResponsibilityStructure newResponsibilityStructure,
//                   ArchEResponsibilityStructure parentResponsibilityStructure) {
//
//             boolean changed = true;
//             ICMPerformanceResponsibilityStructure                 newStructure                =
((ICMPerformanceResponsibilityStructure)newResponsibilityStructure);
//             ICMPerformanceResponsibilityStructure               previousStructure               =
((ICMPerformanceResponsibilityStructure)parentResponsibilityStructure);
//
//             String relationTypeVO = ArchEResponsibilityReactionRelationVO.class.getName();
//             System.out.println("NEW                                                          STRUCTURE:
"+newStructure.getRelations(relationTypeVO).size());
//             System.out.println("PARENT                                                       STRUCTURE:
"+previousStructure.getRelations(relationTypeVO).size());
//
//             return (changed);
//     }
  @Override
  public List<ArchEUserQuestion> describeOtherThanTactic(
          ArchERequirementModel requirementModel,
          ArchEArchitecture architecture) throws ArchEException {

    List<ArchEUserQuestion> warningQuestions = new ArrayList<ArchEUserQuestion>();
    ArchEUserQuestion question = null;

    if (!this.isAnalysisValid()) {

      Integer key = null;
      ArchEAnalysisProblem problem = null;
      String text = null;

                        for    (Enumeration<Integer>    enumErrors    =    reportedProblems.keys();
enumErrors.hasMoreElements();) {
        key = enumErrors.nextElement();
        problem = reportedProblems.get(key);
        text = problem.getMesssage() + " ( error= " + key + " )";

        question = new ArchEUserQuestion();
        question.setQuestionID(FIX_ANALYSIS_PARAMETERS_WARNING);
        question.setParent(problem.getSource());
        question.setAffectedFacts(null);

        List parameters = new ArrayList();
        parameters.add(text);
        question.setParameters(parameters);

        warningQuestions.add(question);
      }

      // This code is now called by ReasoningFrameworkExecutor after finishing its cycle
//                   this.clearAnalysisStatus();
//                   this.clearResponsibilityStructureStatus();
    }
```

```
    return (warningQuestions);

  }

  @Override
  public void beginLearning() throws ArchEException {
              // TODO Auto-generated method stub

  }

  @Override
  public List<ArchEUserQuestion> describeWhatLearned()
          throws ArchEException {
    // TODO Auto-generated method stub
    return null;
  }

  @Override
  public void learnBy(int typeOfLearning, Object[] args)
          throws ArchEException {
              // TODO Auto-generated method stub

  }

  /**
   * Implementation of ConsolePrinter that allows printing messages from the
   * evaluation procedures directly to the console.
   *
   */
  class MyDefaultConsolePrinter implements ConsolePrinter {

    /**
     * @see
     * edu.cmu.sei.pacc.perf.ConsolePrinter#printToConsole(java.lang.String)
     */
//        public void printToConsole(String msg) {
//            final String msgToPrint = msg == null ? "" : msg;
//            Display.getDefault().asyncExec(new Runnable() {
//
//     public void run() {
//             MessageConsoleStream                                out                                =
ArchEUIPlugin.getDefault().getConsole().newMessageStream();
//             assert out != null : "ERROR - MessageConsoleStream is null when printing:\n" +
msgToPrint;
//              System.out.println("msgToPrint: " + msgToPrint);
//              out.println(msgToPrint);
//              try {
//                    out.flush();
//                out.close();
//              } catch (IOException e) {
//                    // nothing to do
//              }
//         }
//         });
//         }
    public void printToConsole(String msg) {
      printLog(2, Level.INFO, msg);
    }

  } // End MyDefaultConsolePrinter
}
```