UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

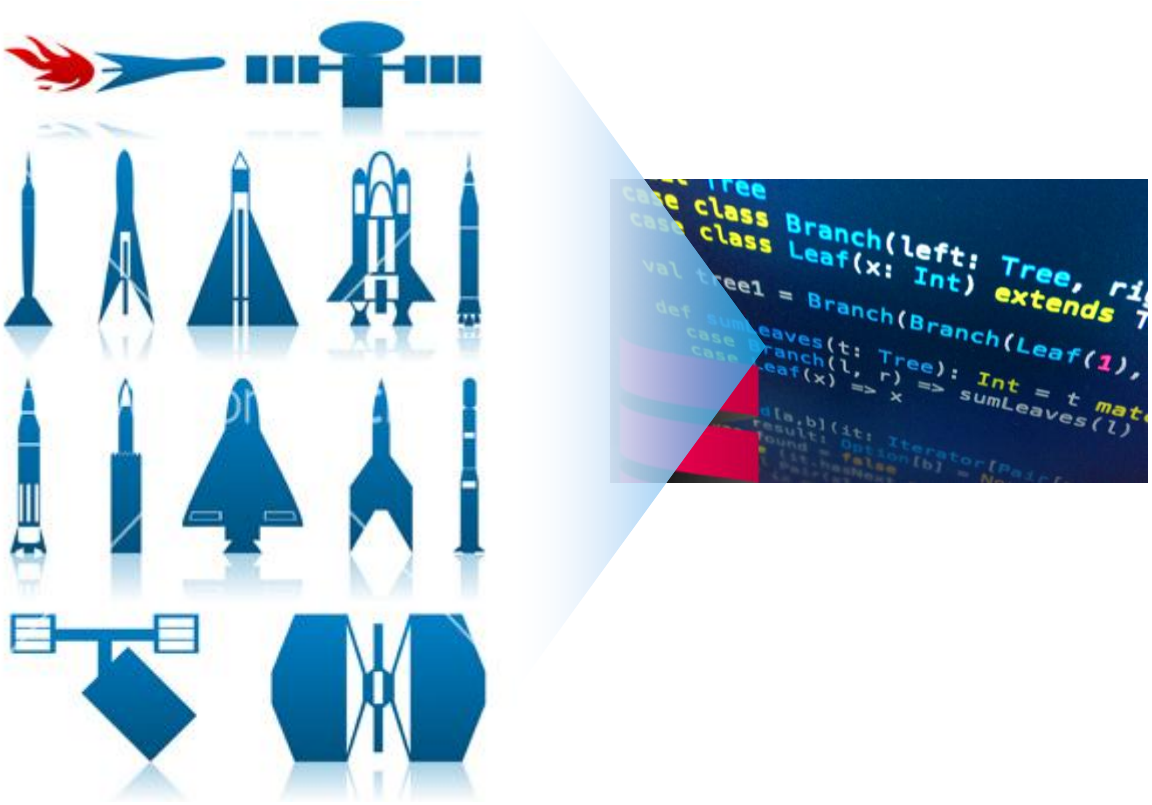
MÁSTER UNIVERSITARIO EN INVESTIGACIÓN EN INGENIERÍA DE SOFTWARE Y SISTEMAS INFORMÁTICOS

ITINERARIO DE INGENIERÍA DE SOFTWARE

CÓDIGO 31105128

TRABAJO FIN DE MASTER

- Anexo ArchE -



Estudiante: Jose Antonio Miranda Rodera

Profesor: José Félix Estívariz López

Curso: 2014/2015

Convocatoria: Junio - 2015

Índice de Contenido

12 ANEXO 1 - INTRODUCCIÓN A ARCHE

12.1 Conceptos Básicos de ArchE

ArchE (Architecture Expert), se define, tal y como viene recogido en [36], como un asistente que ayudará a los diseñadores software a generar una arquitectura que satisfaga los requisitos expresados como escenarios de atributos de calidad. Arche, tal y como dice [37], es una herramienta software para ayudar al diseñador a desarrollar arquitecturas que poseen niveles especificados de calidad requeridos.

ArchE tiene conocimiento de atributos de calidad pero no conoce ningún dominio del problema. Por consiguiente, ArchE puede ofrecer consejos sobre cómo satisfacer requisitos de atributos de calidad pero no conoce que significan estos consejos para el arquitecto con respecto al particular dominio del sistema.

ArchE es un sistema basado en reglas, en el cual los modelos de atributos de calidad son vistos como marcos. ArchE utiliza Jess (intérprete basado en reglas de Java). Además, ArchE está implementado como una aplicación Eclipse, el cual proporciona una familiarización inmediata con la interfaz de usuario y un concepto de operación para cualquiera que haya usado Eclipse.

En la Figura 208 se muestra el flujo general de ArchE:

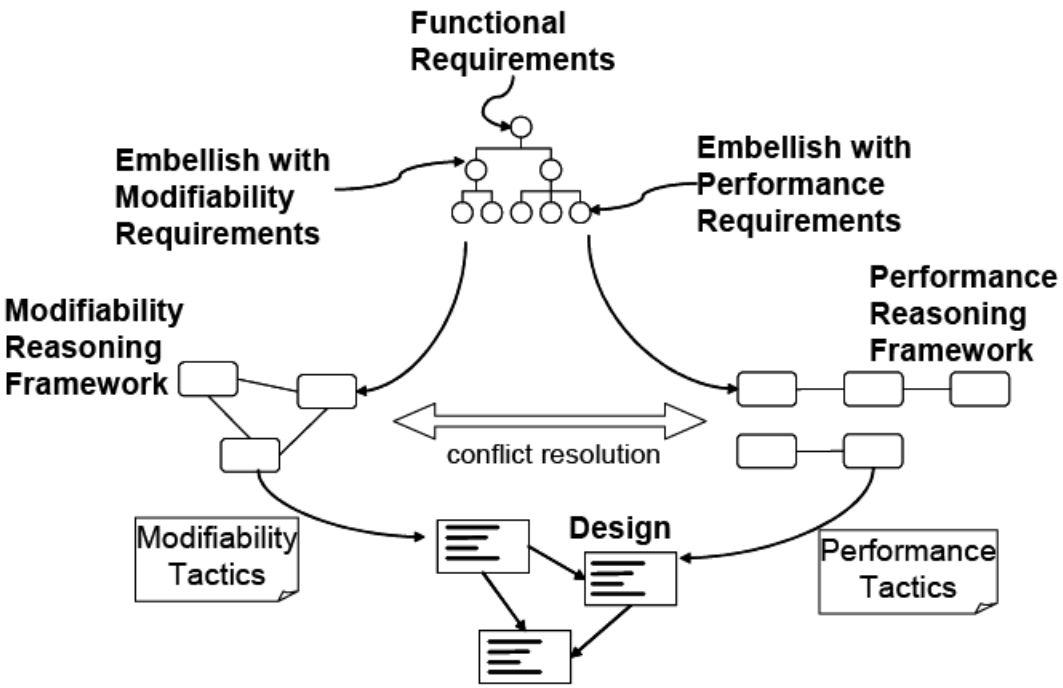


Figura 208. Flujo General de ArchE. [36]

Dentro de esta figura se recogen los conceptos más importantes que maneja ArchE, según [36]:

- *Escenarios de atributos de calidad:* tal y como se describieron en el capítulo 3, tienen una estructura que incluye seis conceptos: estímulo, fuente del estímulo, entorno, artefacto que es estimulado, respuesta y medida de respuesta.

- *Marcos de razonamiento:* es un cuerpo de conocimiento sobre un atributo de calidad particular, que incluye métodos para calcular la medida de la respuesta frente a un estímulo, dada una colección de parámetros independientes. Además, incluye tácticas arquitectónicas que permiten ajustar dichos parámetros independientes, para modificar y controlar el valor de los parámetros dependientes.
- *Responsabilidades:* son actividades asignadas al software que está siendo diseñado. Las responsabilidades son usadas por ArchE como medios para expresar requisitos funcionales, como una porción integral de los escenarios de atributos de calidad, y como medios de integrar los modelos producidos por diversos marcos de razonamientos de atributos de calidad.

Una vez definidos estos conceptos, Arche funciona de la siguiente manera: [36]

Dentro de ArchE los requisitos funcionales se presentan como grafos de responsabilidades y son complementados con requisitos de atributos de calidad, en la forma de escenarios de atributos de calidad.

Para cada atributo de calidad, existen marcos de razonamiento que convierten dichos escenarios de calidad en modelos específicos de atributos de calidad. Cada modelo representa un diseño que satisface los requisitos especificados para dicho atributo de calidad.

Los marcos de razonamiento resuelven conflictos entre diferentes modelos de atributos de calidad para crear un modelo global que satisfaga todos los requisitos de atributos de calidad. Este modelo global se convierte entonces en una representación arquitectónica del diseño.

En la Figura 209 [36] se muestra la estructura interna básica de ArchE, que es una estructura tipo *blackboard* donde el repositorio de datos es la *FactBase*, el motor de reglas es proporcionado por Jess, y cada conjunto de reglas es activado por los datos en el repositorio y sitúa nuevos datos en el conjunto de datos. Cada conjunto de datos se denomina módulo.

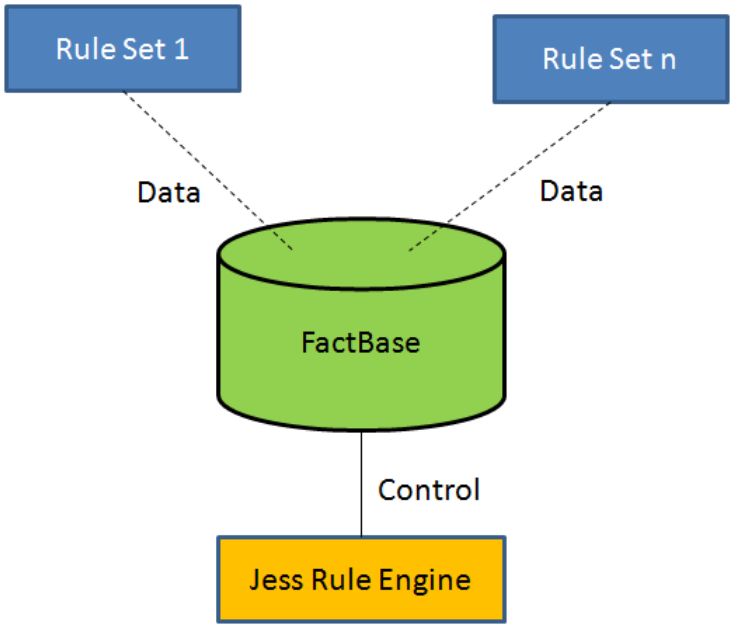


Figura 209. Arquitectura *Blackboard* de ArchE. [36]

En [40] describen los principios del diseño de arquitectura implementados en ArchE: (Figura 210)

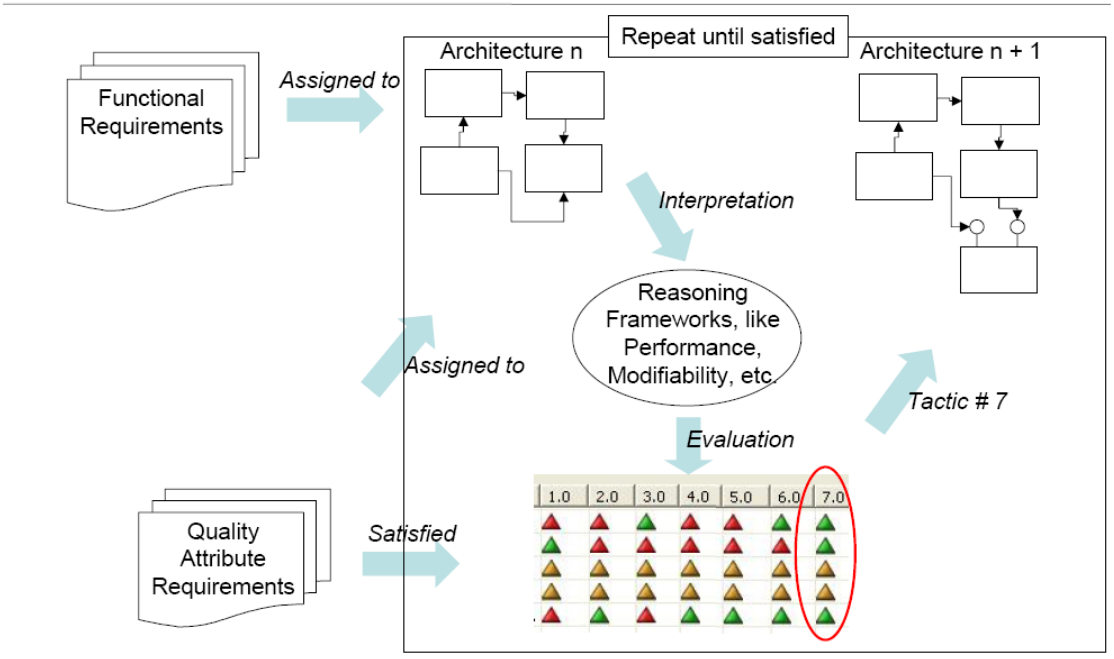


Figura 210. Principios de Diseño de Arquitectura en ArchE. [40]

Este principio de diseño se basa en los siguientes puntos:

- Establecer requisitos:
  - Requisitos funcionales
  - Dependencias entre requisitos
  - Requisitos de atributos de calidad
  - Diseño inicial conteniendo sólo el sistema
- Comprobar que el diseño es bueno, produciendo modelos de atributos de calidad que proporcionen información acerca de los atributos de calidad
  - Extraer información requerida del modelo desde el diseño (interpretación)
  - Correr el modelo para calcular los valores para los requisitos de atributos de calidad (evaluación)
- Intentar una mejora, mediante un conjunto de tácticas que mejoren la arquitectura
  - Interpretar el modelo para determinar posibles tácticas
  - Aplicar las tácticas al diseño mediante el cambio de elementos, relaciones y sus propiedades.

12.2 Operación de ArchE

12.2.1 Conceptos Claves de ArchE

En la Figura 211 se muestran los conceptos claves de ArchE: [36]

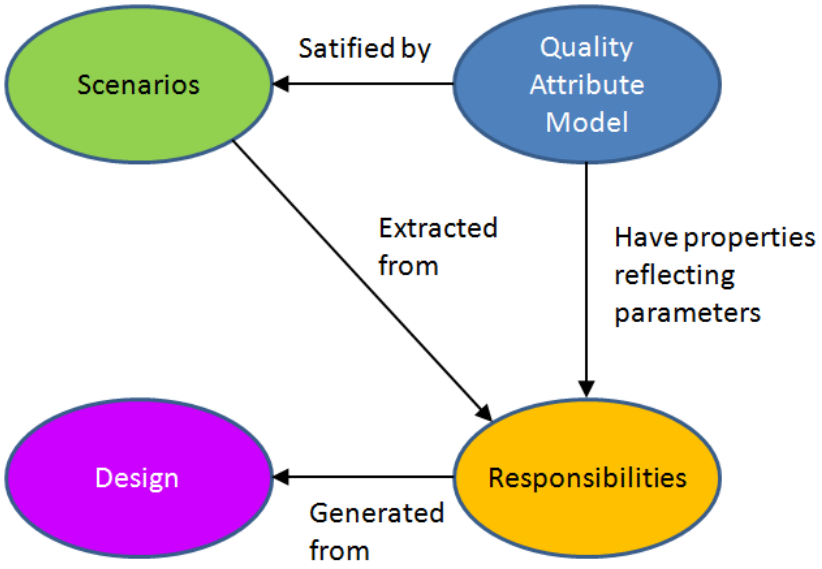


Figura 211. Conceptos Claves de Arche y sus Relaciones. [36]

- Escenarios: los requisitos de calidad para el sistema.
- Responsabilidades: definidas en la sección anterior, estas responsabilidades están ligadas a sus respectivas fuentes (por ejemplo, escenarios, requisitos o tácticas), incluyendo la relación entre los requisitos como se muestra en la Figura 211, y tienen parámetros que incluyen la asignación a los elementos arquitectónicos y propiedades que los diversos marcos de razonamiento necesitan, como tiempo de ejecución o coste de modificación.
- Modelo de Atributos de Calidad: es una instancia del marco de razonamiento completamente instanciada. Son los parámetros independientes para los marcos de razonamiento. Estos parámetros están ligados a sus fuentes (por ejemplo, los escenarios, especificaciones del diseñador, resultados de la aplicación de una táctica particular), así como a las responsabilidades a las cuales pertenecen.
- Diseño: una enumeración de elementos arquitectónicos, sus propiedades y sus relaciones.

12.2.2 Actividades Básicas de ArchE e Interacciones con el Usuario

Según [36], hay diferentes tipos de actividades que un usuario puede realizar en ArchE:

12.2.2.1 Interacciones Básicas

En cualquier punto de la preparación del diseño, el diseñador tiene la opción de salvar el estado actual del diseño, o restaurar el diseño de un estado previo salvado; especificar aspectos del diseño; e importar, exportar o modificar escenarios existentes o requisitos.

Dentro del apartado de especificar aspectos del diseño, hay múltiples tipos de actividades, que incluyen:

- Dar un nombre significativo al conjunto de los requisitos
- Especificar el valor de los parámetros, dentro de cualquier marco de razonamiento
- Especificar restricciones.

12.2.2.2 Adquirir Requisitos

El diseñador puede introducir requisitos, que ArchE transformará posteriormente en responsabilidades.

ArchE adquiere tanto los requisitos funcionales (que son traducidas a responsabilidades) como los requisitos de los escenarios de calidad.

12.2.2.3 Refinar Escenarios

ArchE refina los escenarios, identificando las seis partes en las que se descompone: estímulo, fuente del estímulo, entorno, artefacto que es estimulado, respuesta y medida de respuesta. Dichas partes son añadidas por el usuario en la interfaz de Eclipse.

12.2.2.4 Elegir Marco de Razonamiento

Arche debe determinar qué marco de razonamiento debe resolver un determinado escenario; la actual configuración de ArchE permite elegir entre dos opciones: *ICM Performance* y *ChangeImpact Modifiability*.

12.2.2.5 Construir Los Modelos de Atributos de Calidad.

Un modelo de atributo de calidad es una instancia completamente instanciada de un marco de razonamiento. Si ArchE está resolviendo múltiples escenarios simultáneamente, genera múltiples modelos de atributos de calidad, uno por cada escenario, que son consistentes.

Un modelo consiste en un conjunto de parámetros asociados con atributos de calidad.

Hay tres tipos de interacciones que pueden ocurrir entre ArchE y el diseñador durante la actividad de construir el modelo de atributos de calidad:

- Especificar parámetros: los parámetros dentro de un modelo de atributos de calidad son fijos o libres. Los parámetros pueden ser fijados por el estado actual del diseño o por la especificación del diseñador durante su actividad.
- Asistir a ArchE para elegir tácticas: el constructor del modelo de atributos de calidad explora posibles tácticas para generar una solución, y genera preguntas que deben ser respondidas por el diseñador.
- Reportar la incapacidad de ArchE para satisfacer escenarios particulares: si ArchE no es capaz de generar una solución para un escenario en particular, el diseñador es informado y las razones para esta incapacidad se presentan. El diseñador puede entonces modificar alguno de los escenarios y ArchE intentará generar el diseño de nuevo. El diseñador probablemente relajará la medida e respuesta pero puede también especificar alguno de los parámetros o cambiar el estímulo.

12.2.2.6 Construir el Diseño

Según se describe en [36], una vez que se crea un modelo, ArchE construye un diseño. Un diseño consiste en una serie de elementos arquitectónicos y sus propiedades.

La actividad de construcción del diseño es invocada por iniciativa de ArchE cuanto todos los escenarios se satisfacen o bien por iniciativa del diseñador. En cualquier caso, ArchE debe crear el diseño a partir del modelo existente.

La actividad de construcción del diseño puede fallar sólo cuando una restricción se especifica. Si no hay restricciones, el diseño debería ser alcanzable. Si un diseño falla, ArchE especifica una restricción en los valores de los parámetros y reintenta el diseño. El diseñador queda involucrado sólo cuando ArchE no puede determinar las restricciones.

Arche también propone, a parte del diseño, los requisitos de atributos de calidad no satisfechos por dicho diseño, y propone una serie de transformaciones arquitectónicas para mejorar el diseño con respecto a dichos requisitos de atributos de calidad, tal y como se define en [37].

El arquitecto selecciona una trasformación y proporciona información adicional para los nuevos elementos del diseño, tales como nombres significativos, tiempos de ejecución, o coste del cambio. Este proceso continua hasta que o todos los requisitos de calidad son satisfechos o ArchE no tenga más propuestas.

12.3 Marcos de Razonamiento en ArchE

Según [41], un marco de razonamiento o RF (*Reasoning Framework*) es un vehículo para encapsular el conocimiento de atributos de calidad y las herramientas necesarias para analizar el comportamiento de un sistema con respecto a algunos de esos atributos de calidad.

La razón para encapsular dicho conocimiento sobre atributos de calidad es para permitir el incorporar dicho conocimiento en ArchE sin requerir que los atributos de calidad sepan algo el uno del otro.

Según [40], un marco de razonamiento en Arche realiza las siguientes tareas:

- Traduce desde la descripción de la arquitectura al modelo de atributos de calidad (Interpretación).
- Evalúa los escenarios de atributos de calidad en términos del modelo (Evaluación).
- Propone tácticas para mejorar la arquitectura.

Se necesitan dos entradas para un marco de razonamiento en ArchE:

- La arquitectura actual.
- Los escenarios de atributos de calidad relevantes.

Y se proporcionan dos salidas:

- La evaluación de la arquitectura actual con respecto a los escenarios de atributos de calidad.
- La lista de potenciales tácticas para mejorar la arquitectura si al menos un escenario no se cumple.

Para hacer todo esto, los marcos de razonamiento de ArchE requieren una definición clara de los elementos arquitectónicos, relaciones y propiedades que pueden influir en un atributo de

calidad. La interpretación extrae esta información desde la arquitectura y crea un modelo a partir de ella.

Además, se requiere la existencia de una fórmula para realizar los cálculos con el modelo, para proporcionar información sobre el cumplimiento del atributo de calidad. Esta tarea es llevada a cabo por la evaluación.

Por último, ArchE requiere una clara definición de los posibles cambios a la arquitectura para conseguir un mejor cumplimiento de los atributos de calidad. Esta tarea se lleva a cabo por las tácticas.

Los marcos de razonamiento ArchE están basadas en las teorías explicadas en [38], y básicamente son dos: Modificabilidad y Rendimiento.

En [39] se proporcionan unos ejemplos acerca del funcionamiento práctico de los marcos de razonamiento de ArchE.

Según [41], los elementos de un marco de razonamiento se pueden observar en la Figura 212:

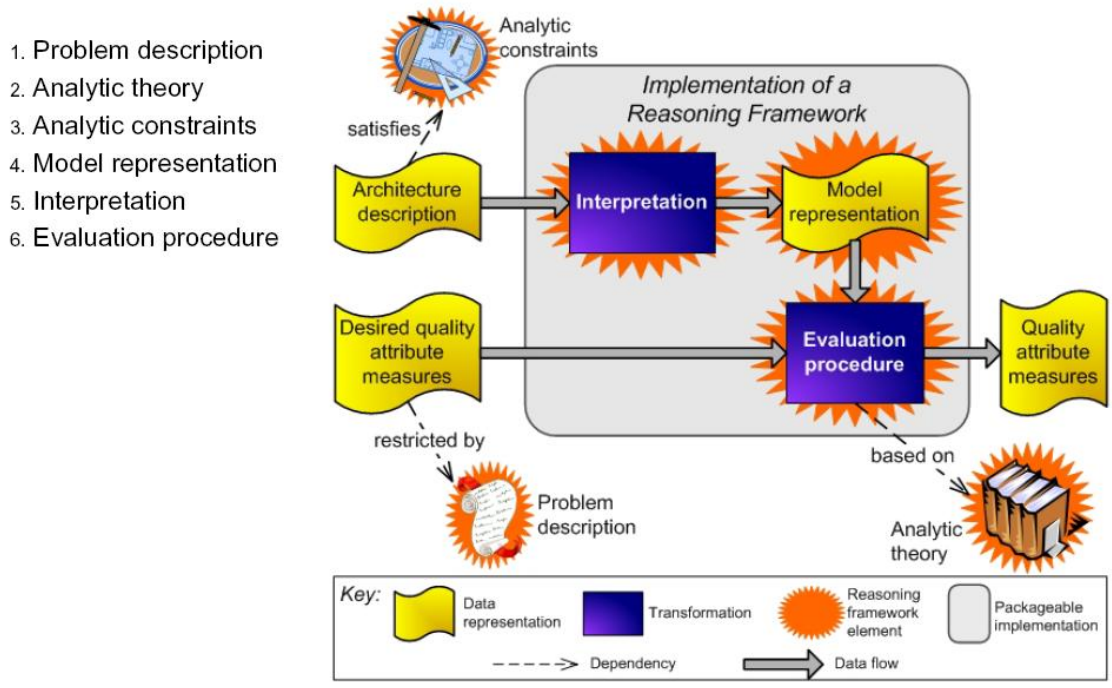


Figura 212. Elementos de un Marco de Razonamiento. [41]

Según [38], los principios fundamentales de dichos marcos de razonamiento son los siguientes:

12.3.1 Modificabilidad

Modificabilidad trata sobre el coste de hacer cambios. Intentar hacer una arquitectura modificable significa localizar cambios en una pequeña área de la misma. Lo peor que puede ocurrir para un arquitecto es que un pequeño cambio en una parte de la arquitectura afecte muchas otras partes.

Los cambios tienen tres aspectos:

- Qué se va a cambiar (interfaz de usuario, Sistema operativo, etc.)
- Quién debería hacer el cambio (desarrollador, administrador del sistema, etc.)
- Cuando se va a hacer el cambio (durante el desarrollo, instalación, cuando el sistema está corriendo, etc.)

12.3.1.1 Especificando Requisitos de Modificabilidad

Los cambios pueden clasificarse de de acuerdo a:

- Probabilidad: cómo de probable es que cierto cambio ocurra
- Frecuencia: cómo de a menudo cierto cambio ocurrirá.
- Dependencia: si el cambio está conectado a otro tipo diferente de cambio, es decir, si el cambio provoca a su vez otro cambio.

12.3.1.2 Elementos Primarios que Afectan a la Modificabilidad

Hay dos factores claves:

- El módulo por sí mismo. Un módulo es una unidad de implementación de software que proporciona una unidad coherente de funcionalidad, con su colección de interfaces y su conjunto de responsabilidades que definen las tareas a realizar. Otros módulos dependen de dichas tareas.
- De qué depende dicho módulo. Por ejemplo, el módulo B depende del A si un cambio en A requiere un cambio en el módulo B.

Otros factores que contribuyen a la modificabilidad son:

- Las responsabilidades se asignan a diferentes capas de una aplicación, para soportar la división entre la interfaz de usuario, la aplicación software en sí, y la abstracción hardware. Siempre y cuando se puedan ajustar en estas categorías, los cambios requeridos pueden ser hechos de manera más fácil.
- Cómo está definida la interfaz, es decir, qué es visible y qué está oculto tiene una gran influencia en la modificabilidad.
- La capa intermedia puede actuar como una intermediaria para prevenir cambios a las capas más bajas que se propaguen desde las capas más altas.

En la Figura 213 se muestra la división en capas de un módulo software: [38]

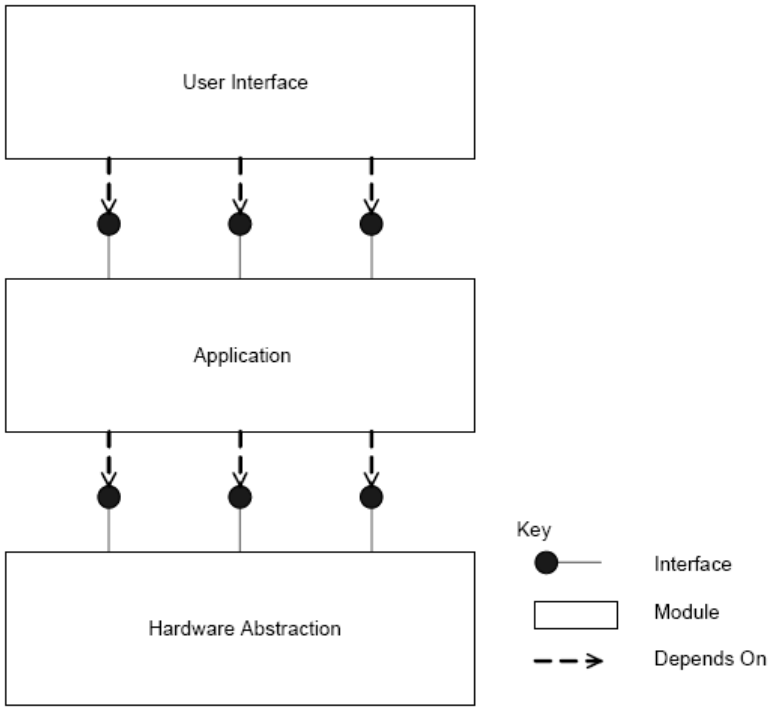


Figura 213. Ejemplo de Arquitectura en Capas. [38]

12.3.1.3 Tácticas de Modificabilidad

Se organizan en tres conjuntos:

- *Tácticas para localizar modificaciones esperadas.* Las responsabilidades que se asignan a los módulos tienen gran influencia en el coste de hacer cambios. Dependiendo de cómo se haga la asignación, un cambio específico puede afectar a un único módulo o a múltiples módulos. La meta de estas tácticas es afectar al menos número de módulos como sea posible por un cambio simple, presentando guías de cómo asignar responsabilidades.
- *Tácticas para restringir la visibilidad de las responsabilidades*. Si un módulo es afectado por un cambio, es importante saber si ese cambio será visible fuera de dicho módulo. Si es así, es posible que se requieran cambios en otros módulos.
- *Tácticas para impedir el efecto onda*. El efecto onda desde una modificación es la necesidad de hacer cambios a módulos que no están directamente afectados por dicha modificación. Esto ocurre debido a que la dependencia entre el módulo afectado por el cambio y otro que depende de él.

12.3.1.4 Análisis de Modificabilidad

El análisis se divide en tres partes:

- Determinar cómo de bien se han asignado las responsabilidades para localizar las modificaciones esperadas
- Determinar cómo de bien la información está oculta y si hay algún efecto onda
- Determinar las posibles dependencias entre módulos que pueden sufrir efectos del cambio en otros módulos.

12.3.1.5 Implementación del Marco de Razonamiento de Modificabilidad en ArchE

Según [40], la modificabilidad de una arquitectura depende de la asignación de funcionalidad a los módulos y de las dependencias entre módulos. La modificabilidad se mide en coste (esfuerzo) del cambio. Por lo tanto, la siguiente información debe estar disponible:

- Grafos de Responsabilidades: tienen dependencias, y pueden ser descompuestos
- Propiedades de responsabilidades: coste del cambio (asignado a cada responsabilidad)
- Propiedades de dependencia entre responsabilidades: aporta una idea sobre la fortaleza del acoplamiento
- Responsabilidades asignadas a los módulos.
- Escenarios de modificabilidad han de ser asignados a responsabilidades.

El trabajo del arquitecto es asignar costes del cambio a cada responsabilidad. No hay manera de que ArchE pueda conocer los valores iniciales. Si el arquitecto no asigna una medida de la fortaleza del acoplamiento para dependencias entre responsabilidades, entonces ArchE asume una probabilidad por defecto del 0.7. Esta probabilidad es una medida de cómo se va a propagar un cambio en una responsabilidad a otras responsabilidades en un módulo.

Es fácil construir el modelo a partir de la descripción de la arquitectura debido a la forma de la misma:

- Cada módulo tiene un coste del cambio que es la suma del coste del cambio de las responsabilidades asignadas.
- Cada módulo que no está descompuesto, se convierte en un nodo en el modelo.
- Cada nodo tiene un coste del cambio que es el coste del cambio del módulo
- Las dependencias entre responsabilidades determinan directamente las dependencias entre módulos.
- Las dependencias entre módulos se convierten en los arcos en el modelo que conecta los nodos.
- Cada dependencia de módulo tiene una fortaleza de acoplamiento, que se asigna a los arcos en el modelo.

En la Figura 214 se resume cómo ArchE crea un modelo de modificabilidad a partir de una arquitectura: [40]

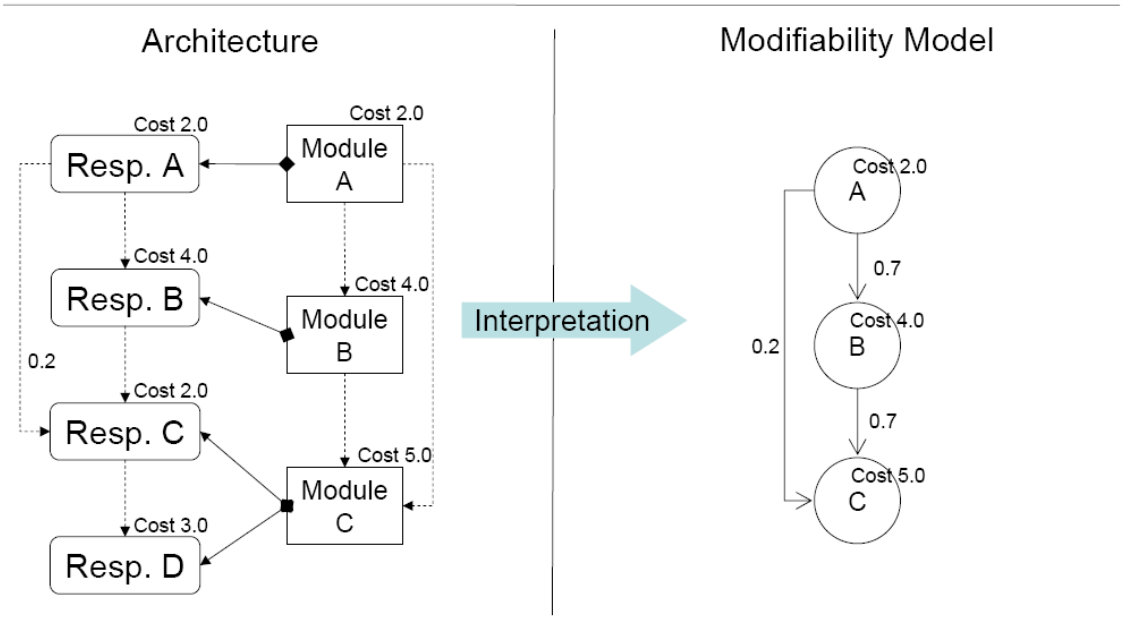


Figura 214. Marco de Razonamiento de Modificabilidad: Relación Arquitectura-Modelo. [40]

En [41] se describe cómo pueden estar acoplados los módulos:

- Altamente acoplados → alta probabilidad de propagación del cambio.
- Medianamente acoplados → probabilidad media de propagación del cambio.
- Bajo acoplamiento acoplados → baja probabilidad de propagación del cambio.

Respecto a las tácticas, [40] establece que las siguientes tácticas de modificabilidad están implementadas en ArchE:

- Encapsulación: reduce el acoplamiento
- Incrementar el nivel de abstracción: reduce el acoplamiento
- Insertar un intermediario: reduce el acoplamiento de algunas dependencias.
- Dividir responsabilidades: permite asignar nuevas responsabilidades a módulos y varía el coste del cambio.
- *Wrapper*: reduce todos los acoplamientos de salida.

12.3.2 Rendimiento

Según se detalla en [38], se detalla rendimiento como la habilidad de un sistema para asignar recursos a las peticiones de servicio de manera que:

- Se satisfagan los requisitos de tiempo
- Se proporcionan varios niveles de servicio

en presencia de:

- Peticiones en competencia
- Demanda variable; y
- Disponibilidad de recursos variable

12.3.2.1 Especificando Requisitos de Rendimiento

Los requisitos de rendimiento están especificados dentro de los escenarios de rendimiento que indican el estímulo al sistema y las respuestas requeridas del sistema a dichos estímulos. Los estímulos de rendimiento son flujos de eventos que el sistema tiene que procesar, y pueden ser clasificados como sigue:

- Periódicos: los intervalos entre llegadas son iguales.
- Estocásticos: los eventos llegan siguiendo alguna distribución probabilística.
- Esporádicos: los intervalos entre llegadas pueden no ser más pequeños que un mínimo especificado.

La respuesta a un escenario de rendimiento está usualmente especificado en términos de requisitos de latencia y tasa de procesamiento: aquellos para minimizar el *jitter*, asegurar que las relaciones de precedencia están adheridas; y proporcionar capacidad de reserva para crecimiento.

12.3.2.2 Elementos Primarios que Afectan al Rendimiento

La latencia es el tiempo que le lleva a un sistema responder a un estímulo incluyendo el tiempo gastado en esperar el tener acceso al recurso y usarlo. Los elementos que afectan a la latencia son:

- Recursos: unidades físicas con capacidad limitada que proporcionan servicios
- Demanda de recursos: generada por estímulos o eventos; un evento es una petición para realizar un cómputo, o lo que es lo mismo, para usar un recurso específico.
- Arbitraje: gestionar demandas competidoras por recursos, según ciertas reglas. Cuando múltiples eventos requieren atención al mismo tiempo, el sistema debe decidir cual atender primero, y como compartir el recurso.

Cuando se considera la latencia, es importante considerar el tiempo de ejecución y las fuentes del tiempo de espera, que incluyen:

- Un evento que está en cola detrás de otros eventos
- Un proceso que necesita acceder a datos compartidos, requiere acceso exclusivo mutuo, y está siendo accedido por otros procesos.
- Un recurso que requiere otros recursos para realizar una terea.
- Un proceso que está entregando el servicio requerido y q tiene que compartir el procesador con otros procesos.

La latencia total para que un recurso responda a un evento comprende el tiempo de ejecución, el overead (cualquier combinación de tiempo de computación, memoria, ancho de banda, o otros recursos que son requeridos para atender una tarea particular) *preemption time* (tiempo que una respuesta debe esperar mientras respuestas de prioridad mayor se están ejecutando), tiempo de bloqueo (tiempo que una respuesta de mayor prioridad debe esperar mientras que una respuesta de prioridad inferior hace uso de un recurso que no se puede liberar), tiempo de trasferencia y tiempo debido a la computación en serie.

Entre los factores que contribuyen a la latencia, están los siguientes:

- La atención en serie de peticiones lleva a paralizar recursos cuando la concurrencia física se puede aprovechar
- El overead utilizando servicios, los mensajes alineados y el transferir control introduce tiempo de ejecución extra.
- Los algoritmos, especialmente en sitios con alta demanda, tienen un impacto muy fuerte en el tiempo de ejecución
- El no tener en cuenta el criterio de arbitraje de recursos, tales como las *deadlines*, pueden llevar a un incremento de latencia para determinados eventos.

12.3.2.3 Tácticas de Rendimiento

En general, la latencia se ve afectada por el nivel y la naturaleza de la demanda de recursos. Esto incluye tasas de llegada de eventos, tiempo de ejecución resultante de la llegada de eventos, y el nivel de variabilidad en dicha llegada. También se ve afectada la latencia por las reglas para elegir entre demandas competidoras por un recurso. Así mismo, la incapacidad de usar un recurso también afecta a la latencia. Por tanto, las tácticas de rendimiento se agrupan en tres niveles:

- *Tácticas para gestionar la demanda de recursos*: controlan los tiempos de espera y de trasferencia mediante el control de las demandas de recursos
- *Tácticas para arbitraje entre demandas en conflicto*: estas tácticas controlan los tiempos de interrupción y espera cuando hay peticiones competidoras, y toman en consideración la importancia semántica o la urgencia (*deadlines*) cuando hay contención por recursos compartidos.
- *Tácticas para gestionar múltiples recursos*: estas tácticas habilitan que múltiples recursos sean usados de manera eficiente para asegurar que los recursos disponibles sean usados cuando se necesiten, por tanto controlando el tiempo de espera de los mismos.

12.3.2.4 Análisis de Rendimiento

Se consideran en el análisis del rendimiento dos posibilidades:

- Gestión de la demanda.

Para el análisis de la gestión de la demanda, se consideran colas FIFO, con un proceso simple, una cola simple y un flujo simple de mensajes, como se muestra en la Figura 215: [38]

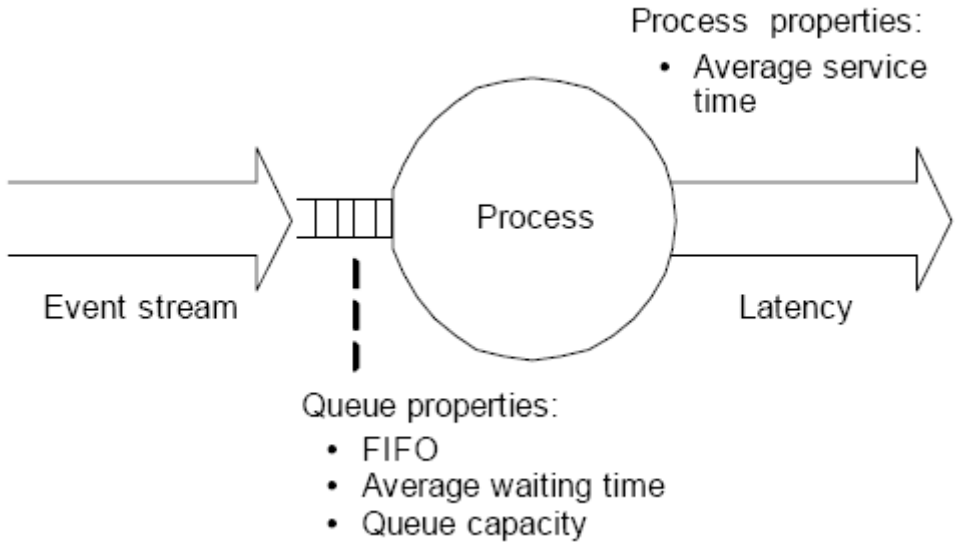


Figura 215. Proceso Único con una única Cola que sirve un único Flujo de Mensajes. [38]

En este caso, la latencia se compone de un tiempo de espera y un tiempo de servicio. El tiempo de espera se compone de dos partes: la cantidad de trabajo que está en la cola y el trabajo que queda para ser completado, para el evento que está siendo servido.

También se puede considerar el caso de múltiples flujos tratados como uno sólo servidos por un único proceso, tal y como se puede observar en la Figura 216: [38]

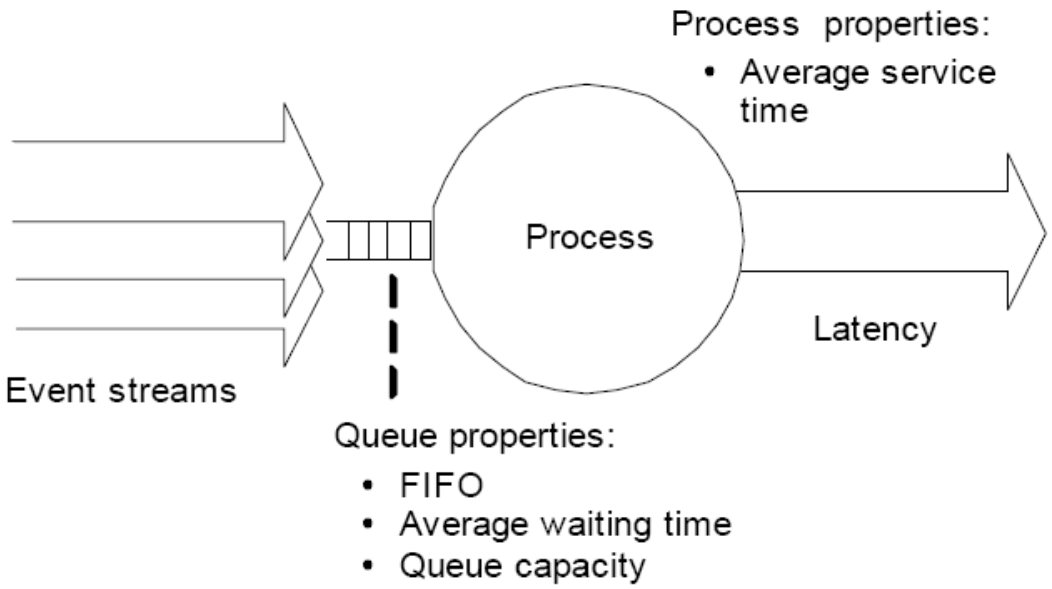


Figura 216. Flujos Múltiples que son tratados como un Único Flujo servido por un Único Proceso. [38]

- Arbitraje

Para el análisis del arbitraje, se tiene en cuenta la planificación por prioridades, y en los efectos que un flujo tiene sobre otro, en particular, considerando múltiples flujos de eventos, cada uno con unos tiempos de ejecución y entre llegadas limitados.

12.3.2.5 Implementación del Marco de Razonamiento de Rendimiento en ArchE

Según [40], el marco de razonamiento de rendimiento en ArchE está basado en el *Rate Monotonic Analysis*, adecuado para razonamiento sobre *deadlines* en tiempo real. Esta teoría usada en ArchE tiene las siguientes hipótesis:

- Único procesador
- La unidad computacional básica es una tarea
- Puede haber recursos compartidos entre tareas
- Las tareas son independientes excepto para dependencia explícita en recursos compartidos
- Sólo una tarea puede usar un recurso compartido en un punto dado en el tiempo.
- Las prioridades de planificación del procesador vienen dadas por el orden de la tarea (por ejemplo, la tarea 1 es la de mayor prioridad, la tarea 2 es la segunda más prioritaria, etc.)

El rendimiento de una arquitectura depende de la asignación de funcionalidad a las tareas. Luna de las típicas medidas de rendimiento es la latencia o tiempo que se tarda en llevar a cabo una tarea. Por lo tanto, la siguiente información debe estar disponible en ArchE:

- Escenarios de rendimiento: tienen un período asociado
- Asignación de escenarios a responsabilidades
- Propiedades de las responsabilidades: tiempo de ejecución.

El trabajo del arquitecto es asignar a cada responsabilidad un tiempo de ejecución. No hay manera de que ArchE pueda conocer los valores iniciales.

A partir de la descripción de la arquitectura, se construye un modelo de la manera siguiente:

- Cada escenario de rendimiento se convierte en una tarea.
- El período especificado para el escenario se convierte en el período de esta tarea.
- Cada responsabilidad tiene un tiempo de ejecución.
- Las responsabilidades asignadas a un escenario se convierten en responsabilidades asignadas a la tarea.
- Las responsabilidades no asignadas a un escenario de rendimiento se asignan a una tarea adicional de baja prioridad (tarea en segundo plano)
- Las responsabilidades compartidas se convierten en recursos compartidos
- Un recurso compartido tiene un tiempo de ejecución para cada tarea que usa este recurso.

En la Figura 217 se representa el modelo de rendimiento de ArchE:

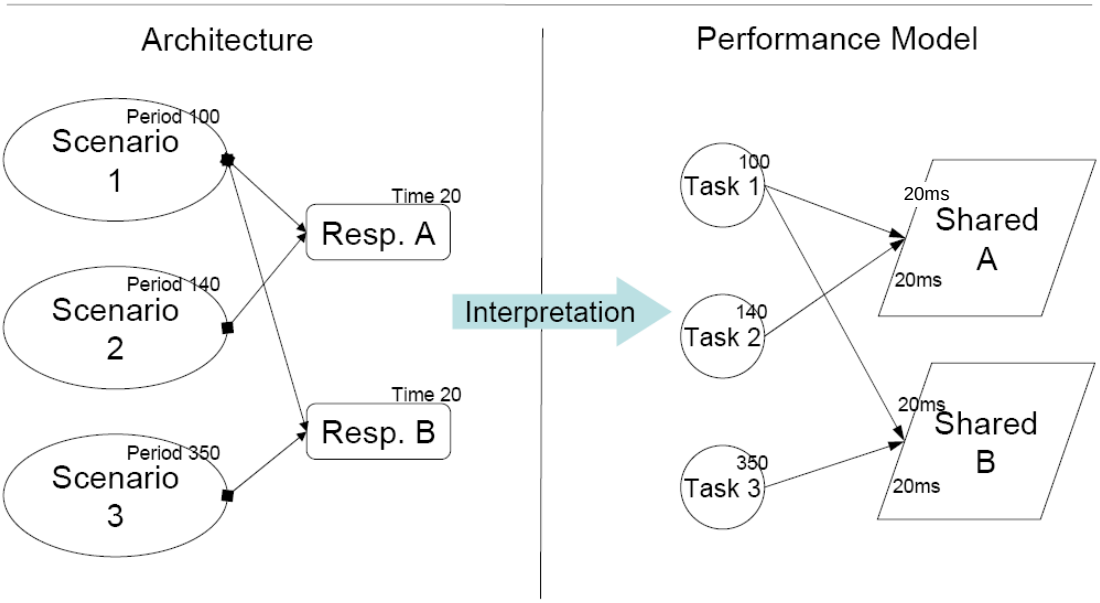


Figura 217. Marco de Razonamiento de Rendimiento: Relación Arquitectura-Modelo [40]

Respecto a las tácticas, hay que resaltar que, a día de hoy, ArchE no implementa tácticas en su marco de razonamiento de rendimiento.

Las únicas tácticas que se pueden utilizar son las acciones directas del arquitecto sobre los siguientes parámetros:

- *Tácticas en requisitos*:
  - Incrementar el período
  - Incrementar la deadline
- *Tácticas de tiempo de ejecución*: reducir el tiempo de ejecución de las responsabilidades.

12.3.2.6 Implementación en ArchE del Marco de Razonamiento a través de Lambda-WBA

Según [41], el marco de razonamiento Lambda-WBA (Worst-Case, Blocking and Asynchrony) predice la latencia en el peor caso como respuesta a un evento. El valor calculado es un límite superior de la latencia porque los efectos de tiempo de ejecución, bloqueo e interrupción del componente de peor caso se asume que ocurren al mismo tiempo.

La teoría subyacente a Lambda-WBA es la Generalized Rate Monotonic Analysis (GRMA), que es una técnica para analizar la planificabilidad de un conjunto de tareas con prioridades variables. Según esta teoría, cada tarea o respuesta está compuesta de una secuencia de subtareas que tienen asociadas un tiempo de ejecución y un nivel de prioridad. Esto hace posible analizar situaciones en las cuales la respuesta a un evento está compuesto de varios cálculos ejecutándose a diferentes prioridades. En este caso, la asignación de prioridades puede estar basada en *deadlines* o en la importancia semántica del componente.

Para evaluar la planificabilidad de tareas de prioridades variables, Lambda-WBA utiliza MAST, que es una herramienta de análisis del caso peor. MAST utiliza su propio lenguaje de entrada para modelos de rendimiento, por tanto los modelos generados por Lambda-WBA tienen que ser traducidos.

La latencia en el peor caso se calcula construyendo la peor alineación posible de efectos de interrupción y bloqueo para cada tarea. El caso peor resultante es un límite superior que puede no darse en la ejecución del sistema si no es probable que los peores efectos sucedan simultáneamente. En la Figura 218 se muestra la salida de MAST para un ejemplo. La latencia en el peor caso se muestra en fondo verde o rojo dependiendo de si la tarea cumple su *deadline* o no, respectivamente. En este caso, las respuestas a clock130 y clock450 no cumplen sus *deadlines*.

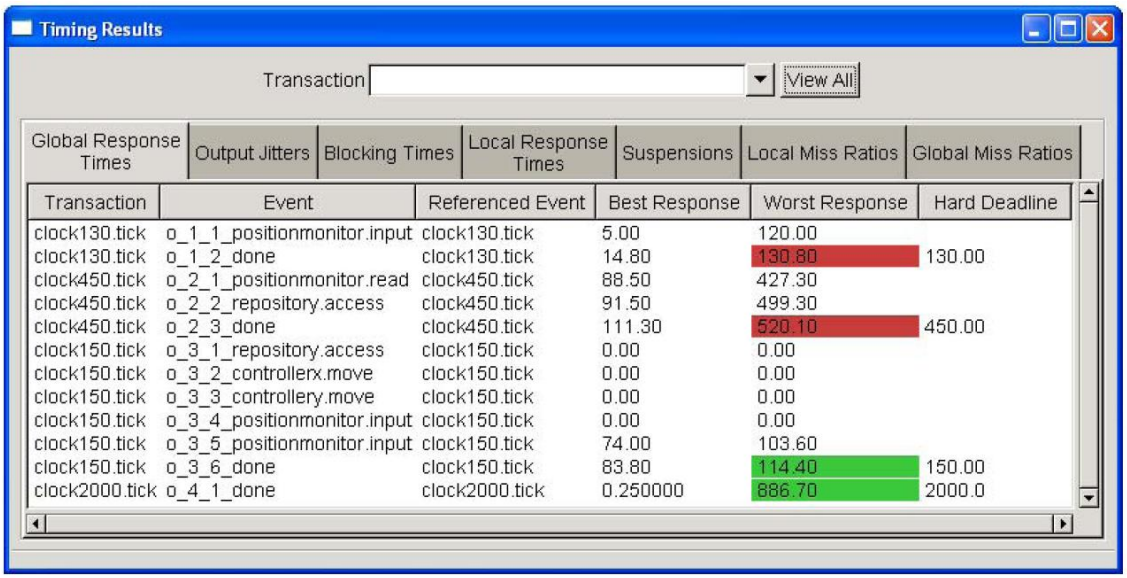


Figura 218. Resultados de la Evaluación Lambda-WBA con MAST. [41]

12.4 Desarrollar ArchE

ArchE, como se ha visto, aporta sólo dos marcos de razonamiento: modificabilidad y rendimiento. Para ampliarlo, simplemente se puede aprovechar las ventajas de la propia arquitectura de ArchE para añadir más marcos de razonamiento.

Según [40], hay dos formas de añadir nuevos marcos de razonamiento: de forma interna o externa: (Figura 219)

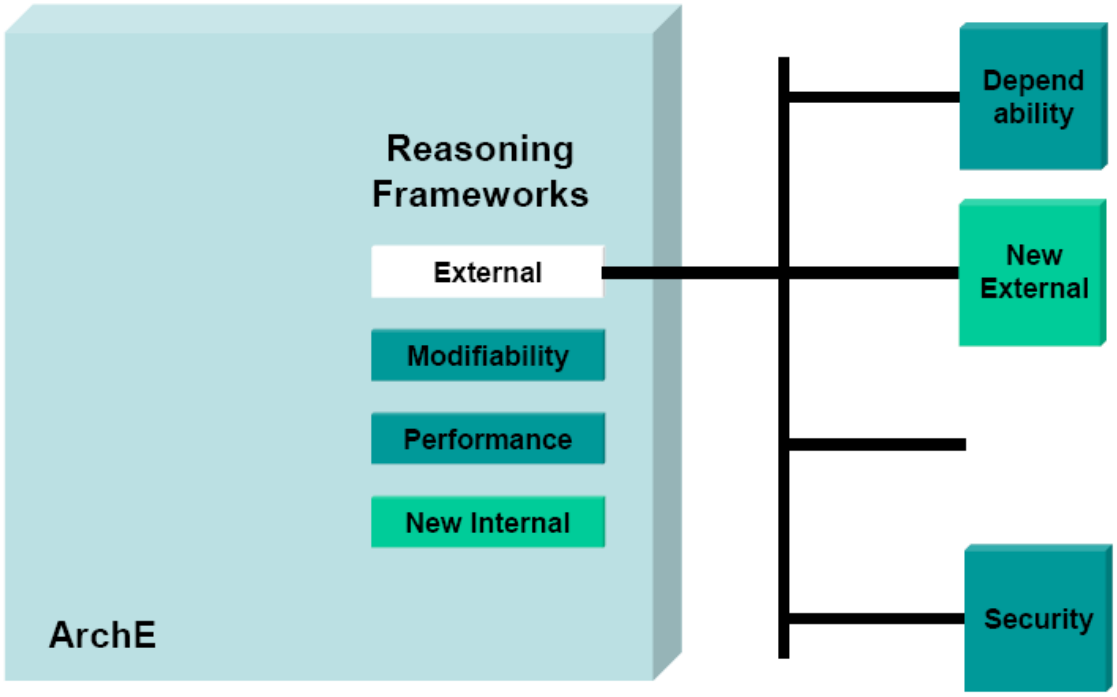


Figura 219. Formas de Añadir nuevos Marcos de Razonamiento en ArchE. [40]

Los marcos de razonamiento pueden ser implementados como *plug-ins* en la arquitectura interna de ArchE. En la Figura 220 se puede ver dicha implementación:

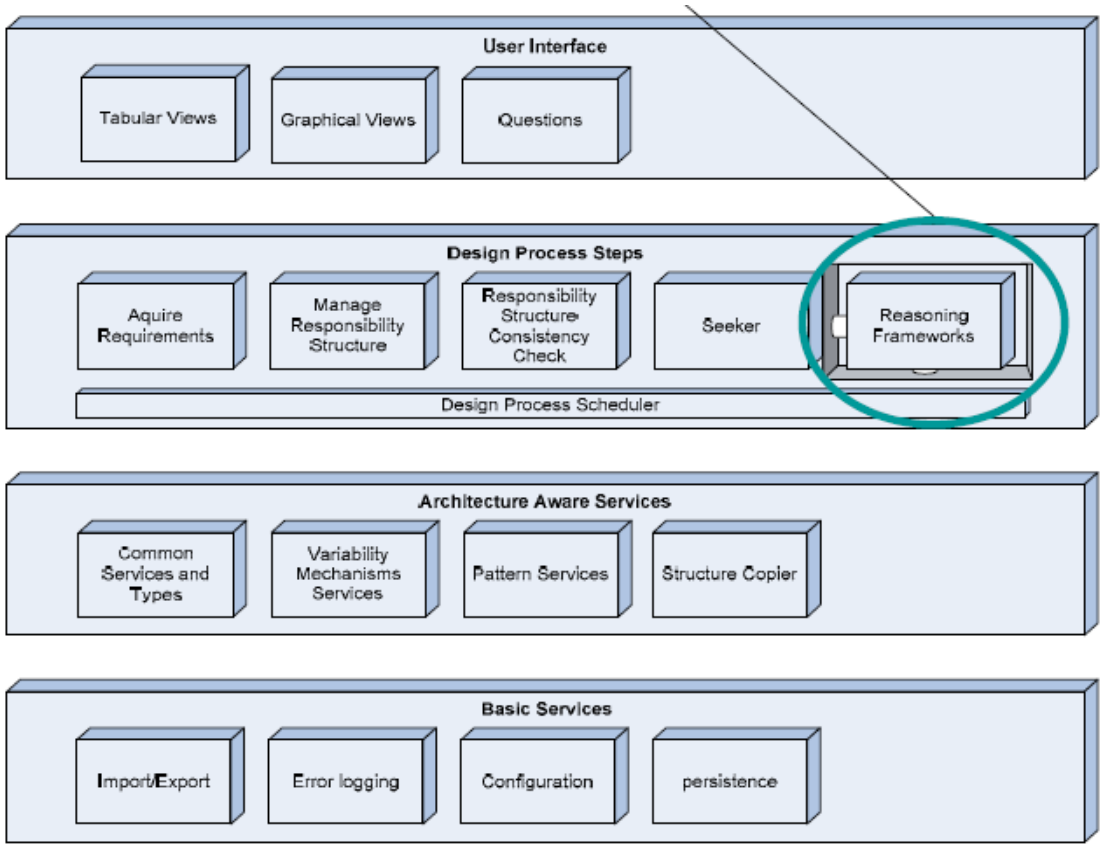


Figura 220. Implementación de *Plug-ins* como Marco de Razonamiento en ArchE. [41]

De hecho, ArchE está construido como un *plug-in* de Eclipse en lo alto de:

- JESS – un motor de reglas
- Java
- MySQL (base de datos)

Un marco de razonamiento se puede implementar en cualquier lenguaje y en cualquier tipo de sistema, y puede ser conectado a una instancia de ArchE via interface XML sobre la red (Figura 221); la interface ArchE-RF es una infraestructura colaborativa que permite desarrollar dichos marcos de razonamiento: [40], [44]

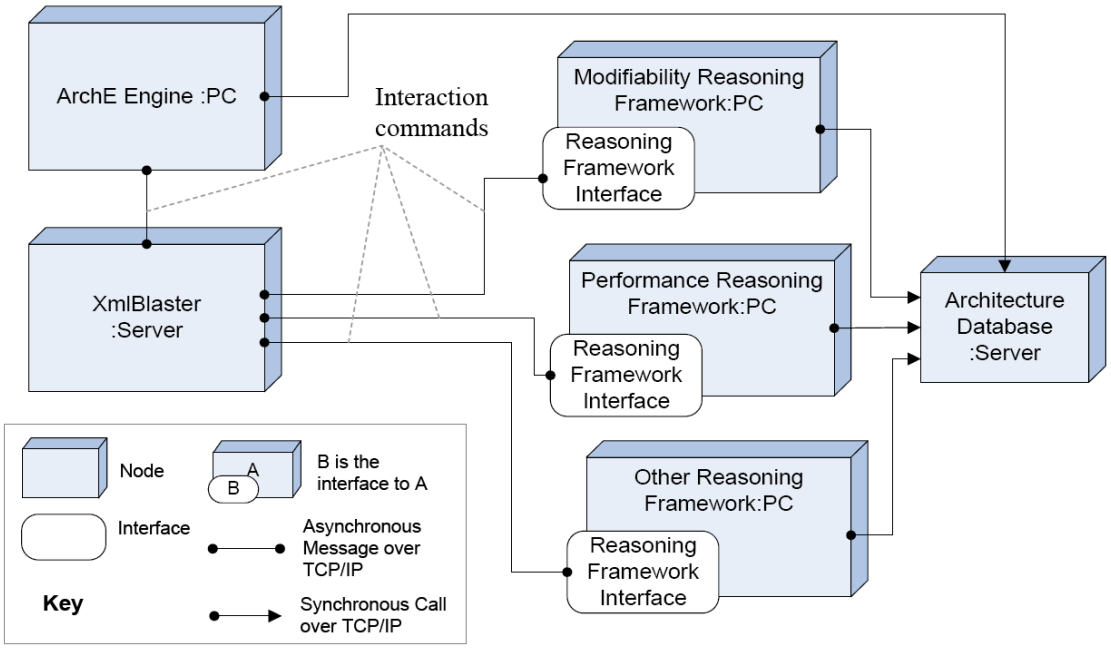


Figura 221. Implementación de Marco de Razonamiento via Servidor XMLBlaster. [44]

Según [44], esta interface Arche-RF es una infraestructura colaborativa que tiene los siguientes elementos constitutivos:

- ArchE Engine: motor Arche con poco conocimiento de técnicas de diseño de atributos de calidad o semánticas acerca del sistema que es diseñado. Sus responsabilidades son: procesamiento de entradas del usuario, actualización de los paneles GUI, analizador sintáctico del "manifestó" de los marcos de razonamiento, coordinación de los marcos de razonamiento, presentación de sus resultados y mostrar en pantalla las preguntas para el usuario.
- XmlBlaster: middleware orientado a mensajes donde pueden tener lugar invocaciones a mensajes implícitos entre diversos participantes sobre una red.
- *Reasoning Framework Interface*: interfaz con los marcos de razonamiento, que abstrae los detalles de trabajar con XmlBlaster, el protocolo de comunicaciones entre ArchE y el marco de razonamiento, y también los algoritmos que ejecutan los comandos de interacción.
- *Architecture Database*: repositorio que se usa para gestionar datos persistentes que necesitan ser compartidos por ArchE y todos los marcos de razonamiento

participantes. Por ejemplo, las arquitecturas originales y candidatas se almacenan aquí.

Según [42], la interface ArchE-RF proporciona técnicas y clases por defecto para desarrollar marcos de razonamiento sin tener que conocer los mecanismos internos de comunicación entre ArchE y los marcos de razonamiento. En la Figura 222 se muestran los diferentes pasos a seguir para desarrollar nuevos marcos de razonamiento: [42]

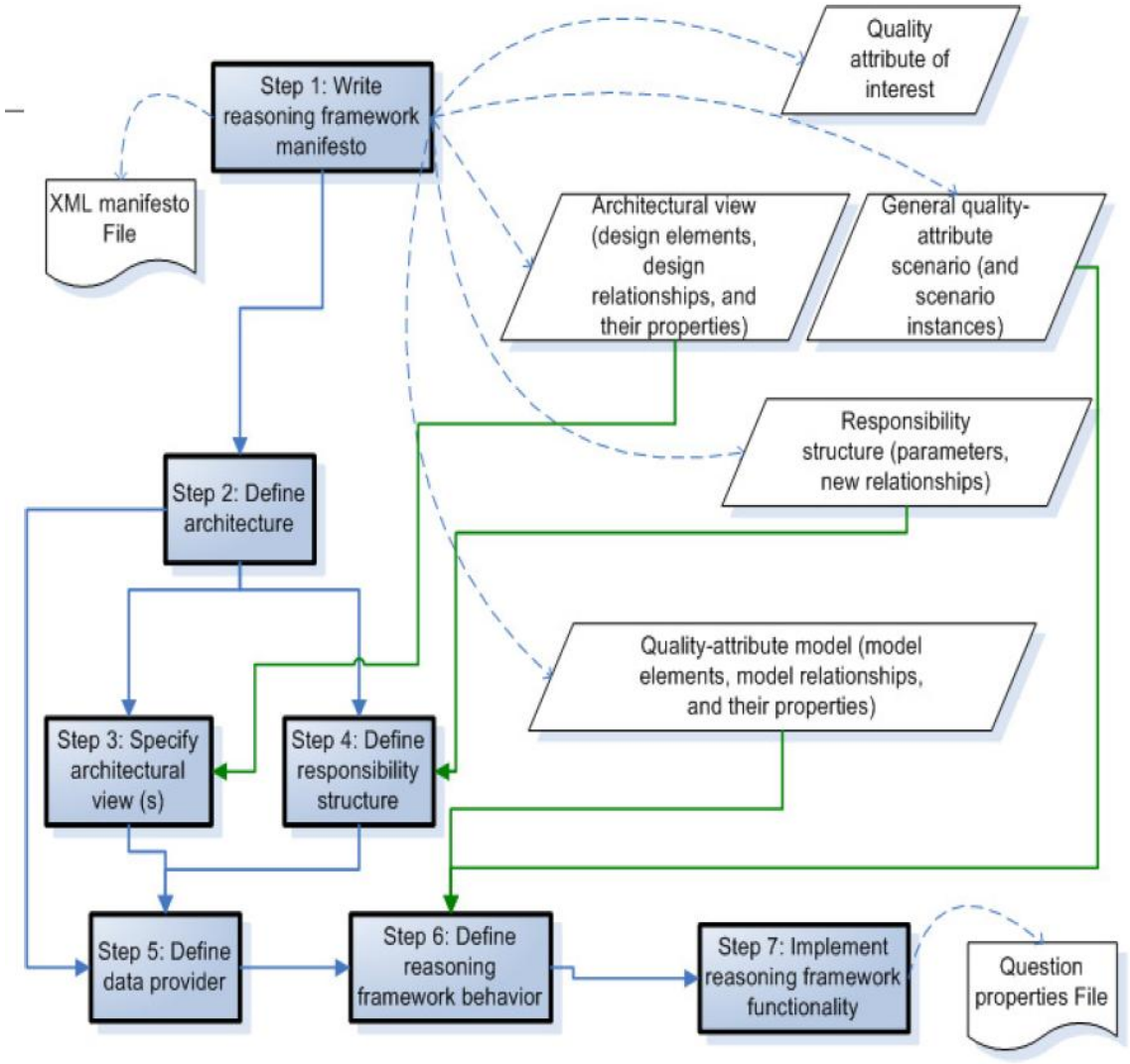


Figura 222. Pasos a seguir en el Desarrollo de un Marco de Razonamiento para ArchE. [42]

Según [44], un marco de razonamiento es una entidad modular que proporciona la capacidad de razonar sobre el comportamiento específico de un atributo de calidad de una arquitectura. Para razonar sobre una arquitectura, en marco de razonamiento necesita de un modelo de análisis, para dar soporte a las tres diferentes fases del análisis:

- Interpretación: el procedimiento de mapeo que convierte el modelo arquitectónico en el modelo de análisis

- Evaluación: el procedimiento usado para resolver el modelo de análisis y calcular las medidas de los atributos de calidad para los escenarios, para determinar si la arquitectura actual satisface dichos escenarios.
- Rediseño: en caso de que algunos escenarios no se cumplan, las tácticas permiten ajustar la estructura/comportamiento al modelo de arquitectura actual.

Para desarrollar dicho marco de razonamiento, se necesitan desarrollar ciertos conceptos arquitectónicos, recogidos en la Figura 223: [44]

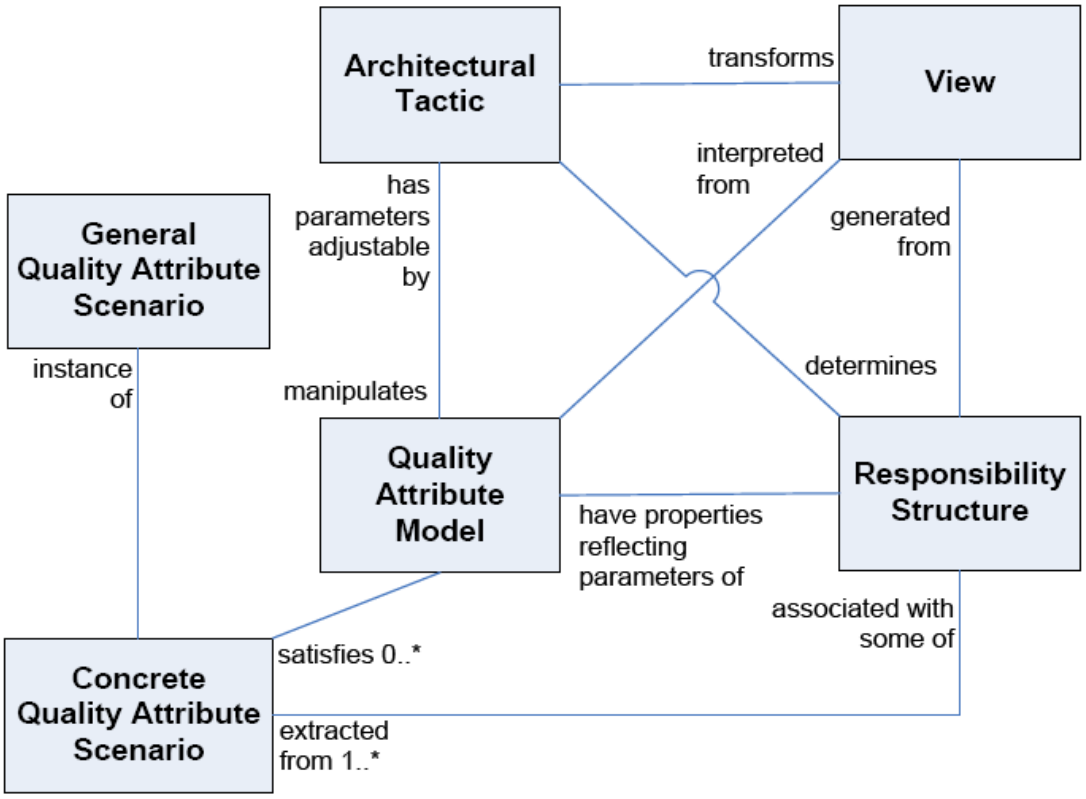


Figura 223. Arquitectura Conceptual Interna de un Marco de Razonamiento. [44]

- Escenario de Atributos de Calidad General: contiene una tabla independiente del sistema para los requisitos de atributos de calidad, basada en el esquema estimulo-fuente del estimulo-entorno-artefacto-respuesta-medida de la respuesta.
- Escenario Concreto de Atributos de Calidad: requisito específico del sistema que es una instancia de un escenario general.
- Modelo de atributos de calidad: el resultado de interpretar un diseño de arquitectura con una teoría analítica.
- Responsabilidades: actividades llevadas a cabo por el software que está siendo diseñado.
- Táctica arquitectónica: un medio para satisfacer una medida de respuesta a un atributo de calidad mediante la modificación de algunos aspectos del modelo de atributos de calidad, a través de decisiones de diseño.

- Vista arquitectónica: una estructura de diseño del sistema que puede ser vista desde determinado punto de vista, como un grafo compuesto por elementos de diseño arquitectónicos.

En la Figura 224 se muestran las diferentes interacciones entre el arquitecto, Arche, y los marcos de razonamiento. [42]

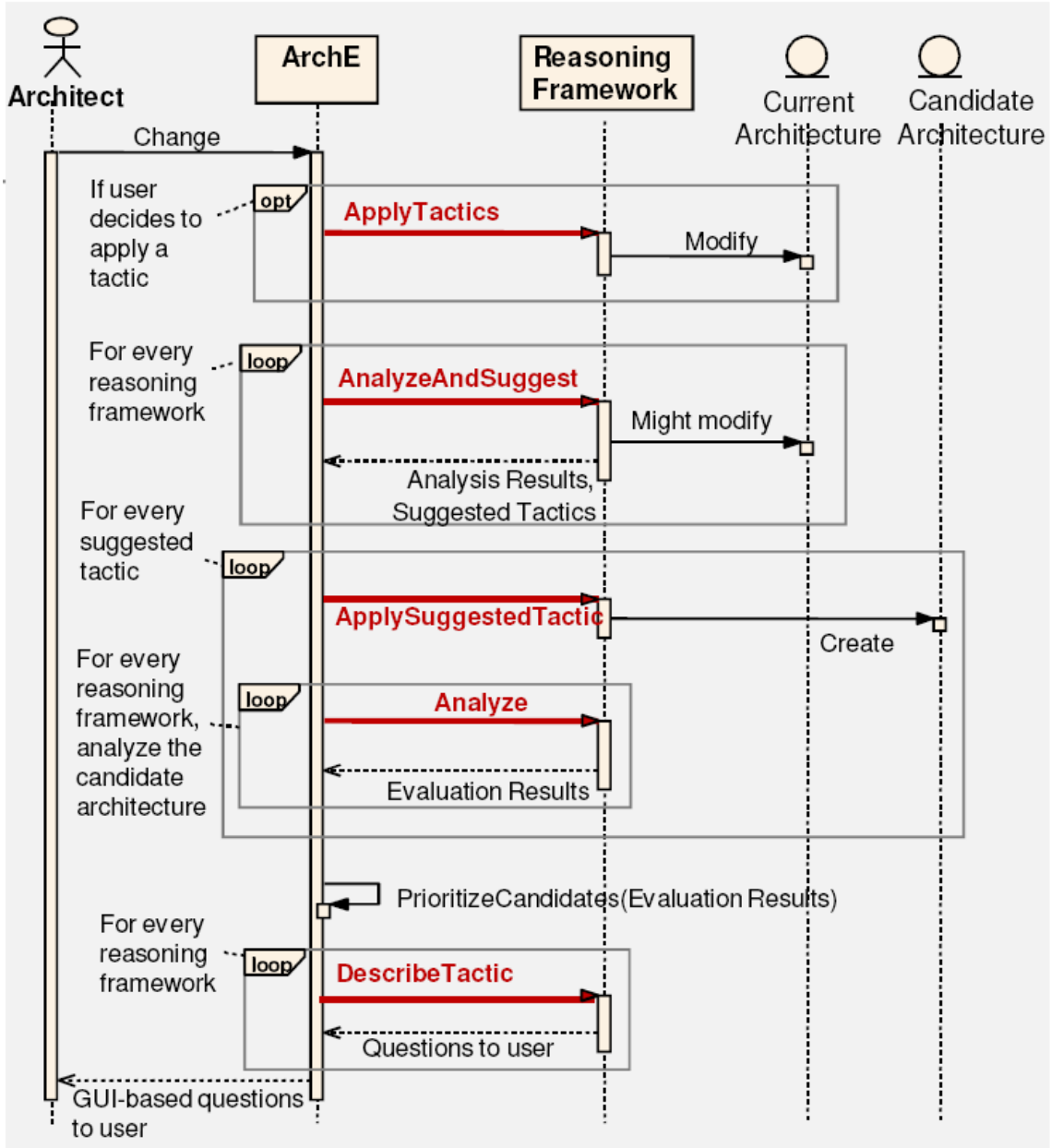


Figura 224. Interacciones entre ArchE y los Marcos de Razonamiento. [42]

Los cinco comandos se describen en [44] y son:

- *ApplyTactics*: solicita a un marco de razonamiento específico aplicar una táctica a la arquitectura actual para refinarla.
- *AnalyzeAndSuggest*: solicita a un marco de razonamiento analizar la arquitectura actual respecto a los escenarios de interés, y a sugerir nuevas tácticas si algunos escenarios no se cumplen.

- *ApplySuggestedTactic*: solicita a un marco de razonamiento aplicar una táctica a la arquitectura actual para crear una nueva arquitectura candidato.
- *Analyze*: solicita a un marco de razonamiento analizar una arquitectura candidato respecto a los escenarios de interés.
- *DescribeTactic*: solicita a un marco de razonamiento proporcionar a ArchE con las preguntas para los usuarios que describen las tácticas o cualquier otra recomendación.

Cuando se implementa la interface ArchE-RF, se supone que un marco de razonamiento va a implementar seis funcionalidades básicas, según [44], y son:

- Autodescripción (Manifesto)
- Crear diseño inicial
- Analizar (para los commandos *Analyze* y *AnalyzeAndSuggest*)
- Sugerir tácticas (para el comando *AnalyzeAndSuggest*)
- Aplicar tácticas (para los comandos *ApplyTactic* y *ApplySuggestedTactic*)
- Describir tácticas (para el comando *DescribeTactic*)

La ArchE-RF Interface API proporciona una guía para implementar internamente los marcos de razonamiento. Según [44], dicha API está estructurada en cuatro capas, cada una de las mismas proporciona servicios para las capas superiores:

- Capa de comunicaciones: la capa de más alto nivel, que incluye todas las clases e interfaces para interactuar con ArchE via XmlBlaster.
- Capa de ejecución: contiene un conjunto de algoritmos cada uno de los cuales procesa un comando de interacción reenviado desde la capa de comunicación.
- Capa de marco de razonamiento: proporciona la clase *ArchEReasoningFramework*, la cual ha de ser extendida por un desarrollador para implementar un marco de razonamiento específico.
- Capa de datos: es la capa de más bajo nivel, y proporciona a las capas más altas los conceptos de la Figura 224. Incluye las interfaces Java requeridas para gestionar los conceptos calves, los cuales deben ser mapeados a clases concretas y tablas de bases de datos.

10 BIBLIOGRAFÍA

[01] Vogel, Olivier; Arnold, Ingo; Chughtai, Arif; Kehrer, Timo. *Software Architecture - A Comprehensive Framework and Guide for Practitioners*. Heidelberg (Germany): Springer, 2009. ISBN: 978-3-642-19735-2.

[02] Shaw, Mary; Garlan, David. *Software architecture: Perspectives on an emerging discipline*. Pearson, 1996. ISBN-10: 0131829572.

[03] Schmidt, Richard F. *Software Engineering - Architecture-driven Software Development*. Elsevier, 2013. ISBN 978-0-12-407768-3.

[04] Gorton, Ian. *Essential Software Architecture*. Springer, 2011 [2ª Edición]. ISBN 978-3-642-19175-6.

[05] Bass, Len; Clements, Paul; Kazman, Rick. *Software Architecture in Practice*. Addison-Wesley, 2003. ISBN: 0-321-15495-9.

[06] Rozanski, Nick; Woods, Eoin. *Software Systems Architecture*. Addison-Wesley, 2005. ISBN 0-321-11229-6.

[07] Buschmann, Frank; Meunier, Regine; Rohnert, Hans; Sommerlad, Peter; Stal, Michael. *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*. Wiley, 1995. ISBN: 978-0-471-95869-7.

[08] Garlan, David; Shaw, Mary. *An Introduction to Software Architecture*. CMU-CS-94-166. 1994.

[09] *Introduction to Software Architecture Evaluation*. Universidad Politécnica de Valencia, de: http://users.dsic.upv.es/~jagonzalez/IST/files/IntroductionArchitectureEvaluation.pdf

[10] Dobrica, L.; Niemela, E. *A Survey on Software Architecture Analysis Methods*. 2002. En: Software Engineering, IEEE Transactions on Software Engineering (Volume:28 , Issue: 7). IEEE. ISSN: 0098-5589

[11] Koziolek, Heiko; Schlich, Bastian; Bilich, Carlos. *A Large-Scale Industrial Case Study on Architecture-based Software Reliability Analysis*. 2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE).

[12] Clements, Paul. *Current Best Practices in Software Architecture Session 4: Evaluating Software Architectures*. 13rd October 2005. Software Engineering Institute. Carnegie Mellon University.

[13] Wojcik, Rob; Bachmann, Felix; Bass, Len; Clements, Paul; Merson, Paulo; Nord, Robert; Wood, Bill. *Attribute-Driven Design (ADD), Version 2.0*. November 2006. Technical Report. CMU/SEI-2006-TR-023. ESC-TR-2006-023.

[14] RTCA DO-178C *Software Considerations in Airborne Systems and Equipment Certification*. 2011

[15] Rierson, Leanna K. *Using The Software Capability Maturity Model For Certification Projects*. Washington. 1998. FAA.

[16] Buter, Andrew; Stienstra, Curt; VanderLeest, Steven H. *Agile for Aerospace*. DornerWorks. GLSEC 2008.

[17] Casals, David. *Aeronautical Software Course: Real Time Software Architectures*. Mission Systems Department.Airbus Defense and Space. 2014

[18] *System Development Modular Approach Eases Avionics Certification Challenges* - COTS Journal online, de: http://www.cotsjournalonline.com/articles/view/101451

[19] ARINC Report 651-1. *Design Guidance For Integrated Modular Avionics*. 1997

[20] DOT/FAA/AR-03/77. *Commercial Off-The-Shelf Real-Time Operating System and Architectural Considerations*. 2004. FAA.

[21] ARINC SPECIFICATION 653-1. *Avionics Application Software Standard Interface*. 2003.

[22] Rufino, José; Craveiro, Joao; *Robust Partitioning and Composability in ARINC 653 Conformant Real-Time Operating Systems*.

[23] Wind River VxWorks 653. 2010, de:

http://www.windriver.com/products/product-overviews/PO_VxWorks653_Platform_0210.pdf

[24] LynxOS-178 RTOS for DO-178B Software Certification, de:

http://www.lynx.com/products/real-time-operating-systems/lynxos-178-rtos-for-do-178b-software-certification/

[25] Green Hills Software - Safety Critical Products: INTEGRITY®-178B RTOS, de:

http://www.ghs.com/products/safety_critical/integrity-do-178b.html

[26] Mejia Alvarez, Pedro; Cova Suazo Nancy, Noemí; Pérez Reséndiz Marisol; *Arquitectura de Software*. CINVESTAV – IPN. Sección de Computación. De:

http://slideplayer.es/slide/1057374/

[27] XPort1020 Freescale MPC8270 Processor-Based Multi-Protocol Twelve-Port Serial 6U cPCI Module, de:

http://www.xes-inc.com/products/view/xport1020/

[27] http://www.xes-inc.com/products/view/xport1020/

[28] http://comps.canstockphoto.es/can-stock-photo_csp26053611.jpg

[29] http://www.futureplatone.com/img/simulador-vuelo-valencia.png

[30] http://www.fancyicons.com/free-icons/108/occupations/png/256/pilot_female_light_256.png

[31] http://comps.canstockphoto.es/can-stock-photo_csp18061284.jpg // http://st2.depositphotos.com/1429923/5516/v/950/depositphotos_55164017-Flat-illustration-of-expert-with-control-panel.-Analytics-and-management.jpg

[32] http://st2.depositphotos.com/1000244/5869/v/450/depositphotos_58693619-sound-speaker-icon.jpg

[33] http://ubuntuarte.com/wordpress/wp-content/uploads/2008/04/sun_server_familyubuntu.jpg

[34] http://www.sdm.es/Web/wp-content/uploads/2014/09/servidores.png

[35] http://previews.123rf.com/images/scanrail/scanrail1205/scanrail120500028/13877506-Hard-disk-and-database-icon-isolated-on-white-background-Stock-Photo.jpg

[36] Bachmann, Felix; Bass, Len; Klein, Mark; *Preliminary Design of ArchE: A Software Architecture Design Assistant*. September 2003. Technical Report. CMU/SEI-2003-TR.

[37] Bachmann, Felix; Bass, Len; Bianco, Philip; Klein, Mark. *Using ArchE in the Classroom: One Experience.* September 2007. Technical Note. CMU/SEI-2007-TN-001.

[38] Bachmann, Felix; Bass, Len; Klein, Mark. *Illuminating the Fundamental Contributors to Software Architecture Quality*. August 2002. Technical Report. CMU/SEI-2002-TR-025. ESC-TR-2002-025.

[39] Bachmann, Felix; Klein, Mark. *Methodical Design of Software Architecture Using an Architecture Design Assistant (ArchE)*. 2005 by Software Engineering Institute - Carnegie Mellon University. Pittsburgh.

[40] Bachmann, Felix; Bass, Len; Bianco, Phil. *Software Architecture Design with ArchE*. Marzo de 2007. Software Engineering Institute. Carnegie Mellon University. Pittsburgh.

[41] Bass, Len. *ArchE – An Architecture Design Assistant*. August 2, 2007. Software Engineering Institute. Carnegie Mellon University. Pittsburgh.

[42] Bianco, Phil; Diaz-Pace, Andres. *Current SEI SAT Initiative Technology Investigations*. May 1st, 2008. Software Engineering Institute. Carnegie Mellon University. Pittsburgh.

[43] Moreno, Gabriel A.; Hansen, *Jeffrey. Overview of the Lambda-* Performance Reasoning Frameworks*. February 2009. Technical Report. CMU/SEI-2008-TR-020. ESC-TR-2008-020.

[44] Diaz-Pace, Andres ; Kim, Hyunwoo; Bass, Len; Bianco, Phil; Bachmann, Felix. *Integrating Quality-attribute Reasoning Frameworks in the ArchE Design Assistant*. Software Engineering Institute. Carnegie Mellon University. Pittsburgh.

[45] Champagne, Roger; Gagné, Sébastien. *Towards automation of architectural tactics application – an example with ArchE*. 2011. Dept. of Software and IT Engineering. ÉTS (University of Québec). Montréal, Canada.

[46] Lee, Jinhee; Bass, Len. *Elements of a Usability Reasoning Framework*. September 2005. Software Architecture Technology Initiative. Technical Note. CMU/SEI-2005-TN-030.

[47] Clements, Paul; Bass, Len. Relating Business Goals to Architecturally Significant Requirements for Software Systems. May 2010. Technical Note. CMU/SEI-2010-TN-018.

[48] Gaitán Peña, Carlos Alberto. *Arquitecturas Software: Gestión de los atributos de calidad y su incorporación en ArchE para la mejora del análisis, evaluación y soporte en la toma de decisiones durante el desarrollo arquitectónico*. Enero de 2014. Trabajo Fin de Master del Máster Universitario En Investigación En Ingeniería De Software Y Sistemas Informáticos – UNED. Curso 2012/2013.

[49] Clements, Paul; Bachmann, Felix; Bass, Len; Garlan, David; Ivers, James; Little, Reed; Merson, Paulo; Nord, Robert; Stafford, Judith. *Documenting Software Architectures. Views and Beyond*. Pearson, 2011 [2ª Edición]. ISBN-10: 0321552687 / ISBN-13: 9780321552686.

[50] Malveau, Raphael; Mowbray, Thomas J. *Software Architect Bootcamp*. Prentice Hall, 2000. ISBN: 0-13-027407-0