UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA
MÁSTER UNIVERSITARIO EN INVESTIGACIÓN EN INGENIERÍA DE SOFTWARE Y SISTEMAS INFORMÁTICOS

ITINERARIO INGENIERÍA DE SOFTWARE
COD: 31105128

TRABAJO DE INVESTIGACIÓN

"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"

Estudiante: Carlos Alberto Gaitán Peña
Dpto: Ingeniería de Software y Sistemas Informáticos
Prof.: Dr. D. José Félix Estívariz López
Curso (2012/2013)
Enero de 2014

UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA
MÁSTER UNIVERSITARIO EN INVESTIGACIÓN EN INGENIERÍA DE SOFTWARE Y SISTEMAS INFORMÁTICOS

ITINERARIO INGENIERÍA DE SOFTWARE
COD: 31105128

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

Carlos Alberto Gaitán Peña
TRABAJO DE INVESTIGACIÓN
Dpto: Ingeniería de Software y Sistemas Informáticos
Prof.: Dr. D. José Félix Estívariz López

Arquitecturas Software: Gestión de los atributos de calidad y su incorporación en ArchE para la mejora del análisis, evaluación y soporte en la toma de decisiones durante el desarrollo arquitectónico

Trabajo de Investigación para el Máster en Investigación en Ingeniería de Software y Sistemas Informáticos modalidad oferta general curso (2012/2013)

Realizado por: _____ (firma) _____

Dirigido por: _____ (firma) _____

Supervisado por:_____ (firma) _____

Tribunal / Órgano Calificador:

Presidente: D./Da _____ (firma) _____

Secretario: D./Da _____ (firma) _____

Vocal: D./Da _____ (firma) _____

Fecha de lectura y defensa: _____

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: 

Carlos Alberto Gaitán Peña

Juan del Rosal, 16
28040, Madrid
Tel: 91 398 89 10
Fax: 91 398 89 09

www.issi.uned.es

RESUMEN

En este documento se presenta la propuesta de investigación "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" para el Máster en Investigación en Ingeniería de Software y Sistemas Informáticos de la UNED. La propuesta consiste en la evaluación de arquitecturas a partir de un conjunto de tácticas las cuales permiten mejorar u optimizar los atributos de calidad para el soporte y diseño de sistemas altamente reutilizables y escalables.

A lo largo del proceso de desarrollo de software el arquitecto utiliza gran variedad de técnicas para validar sus diseños, siendo una de ellas la que responde a la valoración de los atributos de calidad en función de los requerimientos del sistema. Estos atributos y su definición a través de los escenarios de calidad proporcionan un mecanismo eficiente para la evaluación de cualquier arquitectura a partir de los seis (6) atributos reconocidos por (Bass, Len; Clements, Paul; Kazman, Rick, 2003) en su libro "*Software Architecture in Practice*". Una de las características fundamentales de dichos atributos consiste en que a partir de su definición se puedan tomar decisiones "tempranas" sobre una determinada arquitectura. Las decisiones o "Estrategias" no son más que la suma de un conjunto de reglas que permiten ajustar los diseños a un determinado modelo de negocio con base en las necesidades y los planes estratégicos de la organización.

Un diseño arquitectónico es el resultado de la toma de decisiones y la aplicación de un conjunto de técnicas y tácticas para la mejora en los procesos de negocio, donde indiscutiblemente los requerimientos juegan un papel importante ya que a partir de su concepción y definición, el arquitecto modela su experiencia de acuerdo al criterio del medio "entorno" en el cual se está desempeñando la arquitectura. Es evidente que un mejor diseño siempre conllevará a una independencia y flexibilidad de los diferentes componentes de un sistema software, lo que hace escalable cualquier arquitectura; por ello el análisis y la planificación de los requerimientos debe ser una tarea fundamental para el arquitecto, la cual debe ir acompañada siempre de procesos de mejora continua para asegurar la calidad y la vida útil de cualquier sistema informático lo que se traduce en un ROI más eficiente.

La propuesta contempla la utilización de la herramienta ArchE del SEI, un instrumento que sirve como base para el diseño de arquitecturas que soportadas por Marcos de Razonamiento para la evaluación de atributos de calidad como la Modificabilidad y el Rendimiento, la cual se convierte en un soporte para el acercamiento de una arquitectura "ideal" para sistemas basados en Requerimientos.

Palabras Claves: Arquitectura de Software, Análisis y Diseño Basado en Arquitecturas, Atributos de Calidad, Diseño Dirigido por Atributos de Calidad, ArchE, Estrategias, Tácticas para Arquitecturas de Software, Estilos Arquitectónicos

ABSTRACT

This document introduce to "Quality Attributes in Software Architecture and Integration with ArchE for the Quality Improvement, Analysis, Evaluation and Decision Making in Architectonic Design Software" for the Masters in Investigation in Software Engineering and Computer Systems of the UNED. This thesis proposes the evaluation of software architectures from a set of Tactics for Quality Attributes to improve the design, reliability, reusability and performance for escalable systems.

In the process of software development, the architect uses a great variety of techniques to validate his designs, being the most important the Quality Attributes to improve the design of software architectures and their last evaluation form of the system requirements. These attributes and their definition through the Quality Scenaries provide an efficient mechanism for the evaluation of any architecture from the six (6) attributes mentioned by (Bass, Len; Clements, Paul; Kazman, Rick, 2003) in their book "*Software Architecture in Practice*". The principal features of these attributes consists of the capacity that they have to make "early" decisions over an specific architecture. These decisions or "Strategies" correspond to a set of rules to improve the designs and to fit their behavior to the own needs of business and the strategic plans of the organization.

Architectural Design is the result of the decision making and the application of a set of techniques and tactics for the improvement in the business processes, where unquestionably the requirements play an important role since from their conception and their definition, the architect models his experience according to the criterion of means "environment" in which the architecture is evolving. It´s evident that a better design will always entail to an independence and flexibility over the different components from the architecture, turning any into highly scalable; for that reason the analysis and the planning of the requirements must be a fundamental task for all software architects, who must accompany their analysis with estrategical process to improve and to insurance the quality and the best life cycle over the system software for a better ROI.

This work evaluate the use of the Architecture Expert (ArchE) of SEI-CMU, as an instument to drive the process to design systems software and different kind of architectures based on Reasoning Framework as Modifiability and Performance for the evaluation of Quality Attributes. This tool help to software architects and designers to explore ef-fective alternatives in architectural models and designs for a particular set of requirements for an "ideal" architectural model.

Keywords: Software Architecture, Architectural Styles, Scenario-based Analysis of Software Architecture, Quality Attributes, Attibute Driven Design, ArchE, Strategies, Tactics

AGRADECIMIENTOS

El Autor quiere expresar sus agradecimientos a aquellas personas e instituciones que de forma directa e indirecta influyeron en la realización y conclusión del presente trabajo de investigación.

A Dios, señor y dador de vida por ser El quien motiva mi vida y mis deseos de superación y para quien siempre dedicaré mis logros y confiaré mis pensamientos.

A mi hijo Sergio Alejandro, el ser más maravilloso y la mayor bendición de mi vida; para Él una dedicatoria especial, por ser ese motor que mueve mi corazón y mi orgullo como padre, ya que sin su amor, sacrificio y dedicación no hubiese sido posible la realización de este trabajo.

A mi Madre, por ser esa persona excepcional que comparte mis triunfos y fracasos. A ella mis agradecimientos infinitos por que sin su apoyo y dedicación no hubiese sido posible la realización de muchas metas como individuo y como ser social. A ella debo la disciplina y consagración en mis estudios así como la vida misma.

A mi Padre… que sería de mi vida sin el ejemplo de este hombre ejemplar, para El también va dedicado este triunfo, ya que sin su apoyo y consejos no hubiese podido llevar a buen término mi trabajo de grado.

A mi hermano, y demás familiares, gracias por su apoyo incondicional en este y en tantos proyectos de mi vida.

A mi director de Tesis, Dr. D. José Félix Estívariz López, por haber depositado su confianza en mí y por haberme acompañado en todo el proceso permitiendo gestionar mi conocimiento y contribuyendo a la ampliación de mis saberes en esta y otras asignaturas del Máster, a El mis agradecimientos y gratitud por su labor.

A la alma Máter, UNED, al equipo docente del Máster de la ETSI y demás miembros que integran este prestigioso claustro universitario, mis más sentidos agradecimientos por su apoyo y por haberme acogido como uno de sus estudiantes, al brindarme una oportunidad de aprendizaje y crecimiento personal.

A todos vosotros, gracias.

TABLA DE CONTENIDO

Pág.

LISTA DE FIGURAS

Pág.

LISTA DE TABLAS

Pág.

COTENIDO| DE LA TESIS

El contenido de cada uno de los capítulos es el siguiente:

Capítulo 1: **Planteamiento y Objetivos de la Tesis**

La elaboración del informe de investigación surge a partir de una serie de propuestas planteadas en cada una de las líneas de investigación del Máster en Investigación en Ingeniería de Software y Sistemas Informáticos de la UNED. La propuesta orientada a la descripción y evaluación de arquitecturas software y la respectiva toma de decisiones, toma como base a ArchE, un sistema experto para el análisis de los atributos de calidad y define cómo ésta arquitectura permiten mejorar los diseños o modelos a partir de un conjunto de tácticas y estilos de arquitectura.

Capítulo 2: **Introducción a la Arquitectura de Software**

Este capítulo se ha destinado específicamente al estado del arte de la Arquitectura de Software. Se revisan algunos términos, definiciones y estándares que gobiernan a la disciplina como tal. En este apartado se toman definiciones como patrón de diseño, filosofía de orientación a objetos, desarrollo dirigido por modelos y una breve introducción a las SPLE "Ingeniería para Líneas de Producto Software", como una de las vertientes más recientes para la descripción de arquitecturas y diseño de software. En resumen, este capítulo hará que el lector se involucre de primera mano con los conceptos que motivan a esta hermosa disciplina denominada "Ingeniería del Software".

Capítulo 3: **Calidad del Producto Software**

La mejora continua de los procesos se convierte en la razón de ser en todo proceso de calidad, ya que de ello dependen el éxito y consecución de las metas en la organización gestionando de manera eficiente los recursos y mejorando el rendimiento de los procesos los cuales a su vez repercuten en la disminución de los costos y tiempo de implementación. La Calidad es un proceso inherente al desarrollo del software y para ello se han destinado un sinnúmero de estándares y atributos que permiten mejorar sus capacidades. Este capítulo describe en esencia los estándares y marcos de desarrollo que han motivado a la industria del software para la producción de herramientas con calidad.

Capítulo 4: **Arquitectura de Software**

En este capítulo se presentan algunos conceptos básicos como Estilos Arquitectónicos, Requerimientos y Escenarios los cuales están relacionados estrechamente con los parámetros de calidad del software. El capítulo describe las metodologías y lenguajes de descripción arquitectónica bajo los cuales un arquitecto puede modelar un sistema determinado, así como los diferentes Frameworks de desarrollo que le permitirán modelar los distintos escenarios de calidad para un determinado producto software. ABD y ATAM, junto con otros lenguajes son la base de análisis de la Arquitectura Experta (ArchE), de ahí su importancia y definición en este capítulo.

Capítulo 5: **Atributos de Calidad**

Este capítulo se destinará exclusivamente a la definición de los Atributos de Calidad propuestos por (Bass, Len; Clements, Paul; Kazman, Rick, 2003) en su libro "*Software Architecture in Practice*" y su posterior uso a través los **Escenarios** de Calidad en una arquitectura software. Una de las directrices del informe de investigación consiste en la definición de los (6) atributos descritos por los autores, y cómo éstos atributos afectan la mejora del producto software, permitiendo la evaluación de cualquier arquitectura a través de un conjunto de estrategias para adaptar, modificar, corregir o validar un conjunto de estímulos provistos en los distintos escenarios de la arquitectura.

Capítulo 6: **Estrategias Arquitectónicas**

Este capítulo describe en detalle a las Tácticas (Estrategias Arquitecturales). Una **Táctica** consiste en la aplicación de una determinada estrategia que permite optimizar y organizar de forma adecuada uno o más componentes de un sistema o modelo. A estos componentes se les asignan características específicas para controlar sus comportamientos y salidas (posibles respuestas a eventos) para así soportar uno varios atributos de calidad específicos. En este apartado se describen las estrategias más empleadas en los 6 atributos descritos en el capítulo anterior. También se definen algunos patrones arquitectónicos, los cuales representan en sí a una serie de decisiones de diseño potenciales para mejorar los atributos de calidad de un sistema.

Capítulo 7: **Architecture Expert Design Assistant .:.ArchE.:.**

En este capítulo se describe en detalle a la Arquitectura Experta (ArchE), un Sistema Experto basado en atributos de calidad para herramientas software desarrollado por el SEI de la universidad Cernegie Mellon. Esta herramienta "basada inicialmente en Frameworks de razonamiento para los atributos de Performance y Modifiability", es una herramienta que ayuda a explorar

diversas alternativas de diseño a través de un conjunto de tácticas y escenarios para mejorar la calidad del producto. En este capítulo se explorarán sus componentes, funcionalidades, estructura y gestión de conocimiento a través de su motor de inferencia y reglas de producción.

Capítulo 8: **Configuración de la Arquitectura .:.ArchE.:.**

Antes de iniciar con el proceso de implementación de la Arquitectura Experta (ArchE) en un caso práctico, este capítulo describe en forma detallada y secuencial, los pasos para la instalación de la arquitectura, así como el conjunto de pre-requisitos y componentes que deben ser instalados para un correcto funcionamiento de la arquitectura. ArchE está basado en el lenguaje de programación para sistemas expertos JESS. También se basa en el Framework para mapeo de Objetos Relacionales en bases de datos MySQL denominado Hibernate y el Middleware XmlBlaster. Todas estas herramientas en conjunto hacen parte de la suite de ArchE el cual tiene como centro de desarrollo al IDE de Java Eclipse.

Capítulo 9: **Utilización de ArchE en un caso práctico – CTAS**

A partir de los atributos caracterizados en el capítulo 5, específicamente la Modificabilidad y Rendimiento, marcos descritos por el análisis de la Arquitectura Experta ArchE, se realiza una descripción e implementación del caso CTAS (Clemson Traveler Assistant System), un modelo basado en PDA para itinerarios de viajero. Sobre este caso se realizará la respectiva trazabilidad en cuanto a procesos de tácticas y estrategias para la mejora de su diseño y se plantearán las conclusiones en el proceso y las dificultades y demás consideraciones encontradas al momento de implementar la arquitectura.

Capítulo 10: **Otros Casos propuestos y su implementación en ArchE**

En este capítulo se ilustran dos ejemplos con sus respectivos componentes de análisis, asociando los mismos a las funcionalidades de la Arquitectura Experta (ArchE). Cada ejemplo hace parte de la experiencia y pruebas sobre sistemas reales los cuales referencia el autor para una mejor comprensión del sistema experto y cómo éste ayuda al arquitecto en la toma de decisiones para la mejora de los atributos de calidad en un determinado diseño.

Capítulo 11: **Conclusiones**

Se exponen los resultados obtenidos en este trabajo de tesis.

1

OBJETO PRELIMINAR DE LA INVESTIGACIÓN

"Nunca consideres el estudio como un deber, sino como una oportunidad para penetrar en el maravilloso mundo del saber" - Albert Einstein

1.1. OBJETIVO GENERAL

El objetivo del presente trabajo de investigación consiste en analizar y evaluar la herramienta ArchE y su utilidad en el diseño de arquitecturas para sistemas software a partir del uso de los atributos de Modificabilidad y Rendimiento y traducirlos en escenarios de calidad para la evaluación de modelos o arquitecturas a partir de un conjunto de Tácticas para la mejora en torno a las decisiones de diseño en concordancia con los objetivos y visión del negocio.

1.2. OBJETIVOS ESPECÍFICOS

Los objetivos específicos del presente trabajo de investigación se orientan con base en las siguientes actividades:

- Analizar y describir en detalle las propuestas realizadas por (Bass, Len; Clements, Paul; Kazman, Rick, 2003) en su libro "*Software Architecture in Practice*" y cómo éstas estrategias pueden ser implementadas a un modelo determinado a partir de la definición de escenarios de calidad que permitan al arquitecto definir entre diversas alternativas cuál es la mejor con base en costos asociados y la toma de decisiones para la mejora en la calidad del producto software.

- Instalar y configurar la Arquitectura Experta (ArchE).

- A partir de la documentación e instalación de la Arquitectura ArchE, se debe implementar el ejemplo CTAS para verificar hasta qué punto la arquitectura cumple con los principios básicos para arquitecturas software descritos en el libro

"*Software Architecture in Practice*" de (Bass, Len; Clements, Paul; Kazman, Rick, 2003) y como se pueden llevar dichos postulados a un buen diseño a partir de los atributos de calidad, para la toma de decisiones.

- Una vez desarrollado e implementado el ejemplo CTAS, el objetivo subsecuente consistirá en utilizar la Arquitectura Experta (ArchE) en un modelo o sistema desarrollado, midiendo la mejora y concluyendo resultados. Aunque el tema podría incluir el diseño de un nuevo Framework de Razonamiento, este documento se ha enfocado en la aplicación de los postulados de Bass, Clements y Kazman, y cómo éstos mecanismos pueden ser implementados a cualquier arquitectura o modelo de desarrollo.

2

INTRODUCCIÓN A LA ARQUITECTURA DE SOFTWARE

"Todo debe hacerse tan simple como sea posible, pero no más simple" – Albert Einstein

2.1. ESTADO DEL ARTE

La Arquitectura de Software remonta sus inicios a los años 60's. Si bien es cierto que para ese entonces no era consideraba como una disciplina, los inicios de la misma se fundaron en lo que se conoció inicialmente como la Ingeniería del Software. El término de Arquitectura empezó a tomar eco a partir de la denominada etapa de "Crisis del Software". Fue allí donde estudioso de la disciplina de Ingeniería del Software de todo el mundo empezaron a contemplar la posibilidad de separar estas dos vertientes; Arquitectura e Ingeniería.

La disciplina de la Arquitectura de Software tuvo sus inicios a mediados de 1968 en las mentes de 3 ingenieros; Edsger Dijkstra, David Parnas y Fred Brooks quienes acuñaron por primera vez el término. A pesar de su entusiasmo, el mundo de la Ingeniería de Desarrollo se enfocaba en otros aspectos y no fue sino hasta hace unas dos décadas (Inicios de los 90's), cuando Dewayne Perry, Ingeniero de Laboratorios Bell AT&T y Alexander Wolf de la Universidad de Colorado fundaron la disciplina de la Arquitectura del Software.

A raíz del surgimiento de la Arquitectura de Software, y motivados por la mejora de procesos y la calidad de los productos software, en el año de 1969 Ingenieros del SEI "Software Engineering Institute" de la universidad de Carnegie Mellon de los Estados Unidos, aunaron esfuerzos y decidieron aportar significativamente al proceso de desarrollo de software. Nombres como David Garlan, Mary Shaw, Paul Clements y Robert Allen empezaron a trabajar en el contexto de la Arquitectura de Software. Años más tarde y tras un proceso de madurez comprensible, ingenieros del SEI han seguido trabajando en la actualidad para mejorar la perspectiva de la disciplina con ideas innovadoras que permiten mejorar día a día los procesos software y la construcción de modelos. De esta vertiente se destacan Rick Kazman, Mark Klein, Len Bass, Felix Bachmann, entre otros.

Para 1971, Niklaus Wirth en su artículo "*Program development by stepwise refinement*" (Wirth, 1971), publicado por la ACM da crédito a los conceptos planteados inicialmente por Dijsktra quien diferenció y separó por primera vez el término Arquitectura e Ingeniería de Software. Para 1976 DeRemer y Kron respaldarían los mismos planteamientos en el artículo "*Programming-in-the-large versus programming-in-the-small*" publicado por la IEEE. (DeRemer & Kron, 1976).

En 1975, Fred Brooks quien fuera el diseñador del sistema operativo OS/360 hace sus propios aportes a la nueva disciplina. Brooks consideraba que la Arquitectura se orienta fundamentalmente a los aspectos de interfaces de usuario y que los términos Arquitectura e Implementación se debían formalizar en función del ***Qué Hacer*** y el ***Cómo se Hacen*** respectivamente. Para Brooks las estructuras de alto nivel, tenían un papel fundamental en el desarrollo de software y por ello se debía pensar en el Software y el Arquitecto como Agentes. El término *agente* actualmente se describe como una **ABSTRACCIÓN** en términos computacionales, una idea, o un concepto, (este concepto es ampliamente utilizado por el paradigma de orientación a objetos POO a través de la descripción de métodos, funciones, y objetos.

La década de los 70's se caracterizó por ser un periodo donde la disciplina del software evolucionó de manera notoria. Con la aparición del diseño estructurado y los primeros modelos para ciclos de vida del software, se empezaron a notar grandes diferencias entre Arquitectura e Ingeniería. Fue a partir de esta evolución donde se separaron los aspectos de diseño (el Qué) de los aspectos de implementación (el Cómo) dando origen a un sinnúmero de técnicas, herramientas, lenguajes de programación y se sentaron las bases para los lenguajes de modelado.

En 1972 David Parnas introdujo el término "Módulo" a la disciplina de la Arquitectura del Software. (Parnas, 1972), en su artículo "*On the Criteria for Decomposing Systems into Modules*", describe un aspecto esencial en el desarrollo de software "***La Modularidad***"; De hecho sus apreciaciones se consideran hoy por hoy, el insumo más importante en la Arquitectura de Software. Parnas describió a los sistemas como un conjunto de estructuras modulares dando origen al concepto de **Ocultación** u Ocultamiento de la Información, un aspecto fundamental en la Programación Orientada a Objetos o POO.

Parnas así mismo consideró que la capacidad de modularidad en el diseño de una Arquitectura era fundamental para mejorar la calidad, la flexibilización y la capacidad del software, disminuyendo los tiempos de desarrollo. Para 1974, Parnas introduce el término "Estructuras de Software" y en 1976 introduce el concepto de "Familias de Programas", lo que dio origen a lo que hoy se conoce como ***Estilos Arquitectónicos***.

En su artículo "*On the Design and Development of Program Families*", (Parnas, David, 1976) describe a la familia de programas en base a los siguientes postulados:

- Una familia de programas es un conjunto de elementos que tienen la capacidad de ejecutarse en diversas arquitecturas tanto de hardware como de

software. (por ejemplo, juegos de video, navegadores, sistemas operativos, etc.).

- Sin importar la Arquitectura para la que fueran diseñada, una familia de programas ejecuta las mismas operaciones tanto en una como en otra arquitectura, solo que sus entradas y salidas se modifican solo en su formato y en su forma de presentación.
- Las estructuras de datos difieren de los algoritmos debido a la disponibilidad de recursos y al tamaño de las entradas.
- Se considera importante el uso del concepto Módulo y Sub-rutina, para referirse al conjunto de instrucciones o capacidades que determina la funcionalidad de un determinado objeto.
- Una familia de programas debe ser concebida bajo el esquema de consumo. Para ello se deben evaluar las funcionalidades, y entregar a cada quien las respectivas responsabilidades de acuerdo a la demanda.
- Una familia de programas debe ser considerada como un árbol de decisiones. Las hojas del árbol representan sistemas ejecutables y sus derivaciones siempre partirán como consecuencia de un modelo ya existente. (estructuras Top-Down).

Con una clara visión del concepto de calidad aplicada al software, Parnas hacía énfasis en la aplicación de "decisiones tempranas", como un mecanismo para mejorar la calidad del producto final. Es así como a partir de estas alternativas de decisión, en 1996, Paul Clements y Linda Northrop del SEI, denominarían a dichas alternativas "***Decisiones Arquitectónicas***". Según (Clements & Nortrhop, "Software architecture: An executive overview, 1996), *"…los detalles estructurales y una elección temprana o correcta de una Arquitectura, es un factor crítico para el éxito y desarrollo de una solución de software…*". Hoy por hoy las Decisiones Arquitectónicas son la razón de ser de la disciplina de Arquitectura de Software.

En la actualidad, David Parnas es considerado como uno de los verdaderos precursores de la Arquitectura de Software, pues fue éste quien realmente sentó las bases de la Arquitectura de Software y enfatizó en la importancia de los procesos de desarrollo a partir de modelos de calidad.

En la década de los 80's surge un nuevo paradigma de diseño. Tras casi 12 años de continua evolución y aplicación de la teoría estructurada nace el concepto Orientación a Objetos. Aunque Parnas nunca acuñó su término "Ocultación" en función de un paradigma de objetos, éste fue fundamental para la concepción de conceptos que fueran determinantes para la implementación de este nuevo paradigma, ellos son la ***Encapsulación*** y la ***Abstracción***, temas en los cuales trabajara en profundidad la investigadora del SEI Mary Shaw en 1989.

Sin duda alguna el término Arquitectura de Software en la mayor parte de la literatura, le es atribuida a Dewayne Perry y Alexander Wolf, quienes en 1992 sentaron las bases para la disciplina como tal. (Dewayne & Wolf, 1992). Para ese entonces ya era claro que la visión de Parnas en cuanto a la separación de los aspectos arquitectónicos de los de

implementación eran correctos, y que para poder aplicar de forma adecuada estos parámetros, se requeriría de lenguajes y modelos específicos. El resultado de esta evolución dio como resultado la aparición de las primeras herramientas de Diseño Asistido por Ordenador, comúnmente conocidas como CASE[1].

En la década de los 90's se consolidó la disciplina de la Arquitectura de Software, y sin duda alguna el SEI de la universidad Carnegie Mellon, fue la organización que más aportó a la consolidación de la disciplina. A partir de los elementos de juicio proporcionados por Perry y Wolf, la disciplina se convirtió en un marco de referencia para satisfacer requerimientos, para verificar atributos de calidad y estimación de costos, análisis de dependencias, estilos arquitectónicos y niveles de consistencia en cualquier sistema de información. A partir de este conjunto de conceptos y vertientes Paul Clements y Linda Northrop en 1996 crean un modelo paralelo de arquitectura a la cual denominaron "Programación por Componentes", de la cual Clements es el exponente más sobresaliente en este tema.

Otro gran avance en la disciplina, giró en torno a la aparición del término **PATRÓN**, acuñado por primera vez en 1977 por Christopher Alexander, un Arquitecto estructural quien definió al Patrón como "…*una regla con tres partes que expresa la relación entre un determinado contexto, un problema y una solución…*" Este concepto fue ampliamente difundido en su libro "*The Timeles Way of Bulding*" en el cual soportó el uso de patrones para construir edificios de mejor calidad. Christopher en su libro "*A Pattern Language*" promueve el uso de patrones en la arquitectura pero insiste que ésta idea es aplicables a cualquier otra disciplina de la Ingeniería, incluyendo a la Ingeniería del Software.

Retomando el concepto de patrón de Chisrtopher, en 1987, Ward Cunningham y Kent Beck desarrollan por primera vez un lenguaje basado en el uso de los 5 patrones descritos por Christopher al cual denominaron Smalltalk. Estos resultados fueron publicados en el congreso internacional OOPSLA[2] en 1987, una conferencia anual de la ACM.

En congresos posteriores se dedicaron reuniones especiales para debatir el tema de patrones y el asunto fue madurando hasta la publicación 1995 del libro de Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides, conocidos habitualmente como la banda de los cuatro (**Gang of Four**) o simplemente GoF. A partir de este momento se empieza a trabajar en base a patrones de diseño y se estandariza el uso de los mismos a través de un marco común estructurado en 23 patrones.

Un **patrón** de diseño corresponde a una generalización de una metodología de diseño aplica a una línea de producto software. Estos patrones se aplican con frecuencia en las etapas de diseño y no en la implementación misma. La idea principal de un patrón consiste en minimizar los errores derivados de un mal diseño y permitiendo a través de

[1] ***CASE***, "**C**omputer **A**ided **S**oftware **E**ngineering" o Ingeniería de Software Asistida por Computadora.
[2] ***OOPSLA*** – "**O**bject-**O**riented **P**rogramming, **S**ystems, **L**anguages & **A**pplications"
[3] ***SPLE***, Software Product Lines Engineering, Ingeniería para líneas de productos software.

estas técnicas, mejorar los procesos de factoría de software y componentes. Los patrones de diseño en este sentido no corresponden a implementaciones complejas o específicas sino que define a partir de ciertos elementos y sus propiedades, reglas que permiten que el desarrollador compruebe las estructuras que serán implementadas posteriormente, analizando los comportamientos de los objetos resolviendo los problemas de diseño y aplicando un tipo de plantilla para asegurar la calidad del producto software. Un patrón se caracteriza por tener:

- **Nombre del Patrón**: hace referencia al nombre en sí del patrón a implementar.

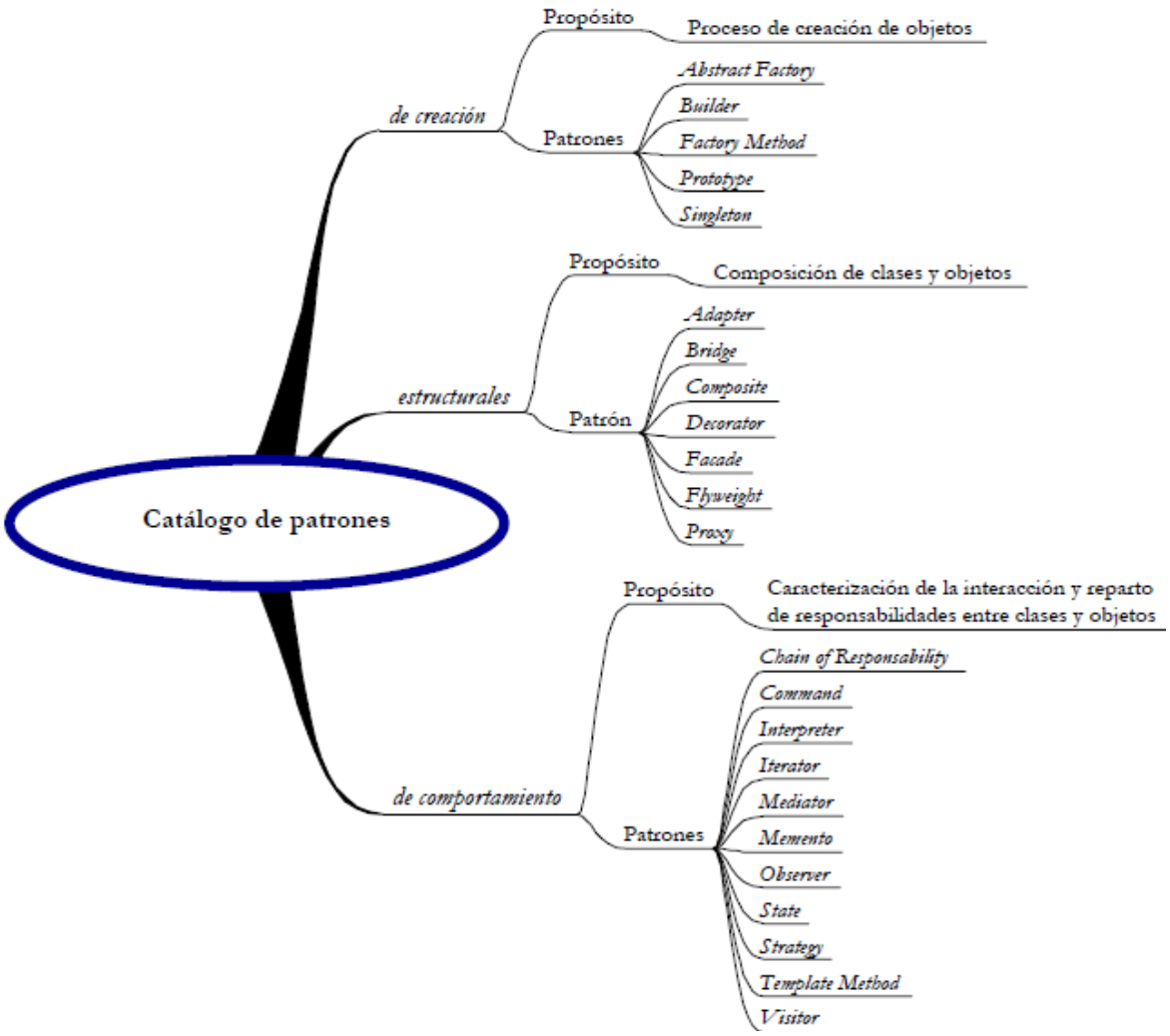
- **Función Objetivo**: define la forma, la descripción y el origen del problema a resolver. Esta característica define cuando y de qué forma debe aplicarse un patrón determinado.

- **Implementación**: corresponde a la solución. Describe los elementos de diseño, sus interacciones y propiedades fundamentadas en el patrón a implementar.

- **Resultados**: en este sentido no siempre los patrones ofrecen una visión global del proceso de desarrollo de software. Aunque los patrones se definieron para mejorar la abstracción de los procesos de desarrollo en el paradigma orientado a objetos, también son susceptibles de ser aplicados desde cualquier metodología de desarrollo. Las bondades o desventajas dependerán en buena medida de la definición de aspectos como la flexibilidad, eficiencia, mantenibilidad, reusabilidad y la definición del mismo patrón en sí.

A partir de los patrones descritos por (Gamma, Helm, & Vlissides, 1994), se han implementado en todo el mundo los conceptos de ***REUSABILIDAD*** y ***PATRÓN DE DISEÑO***, a cualquier tipo de desarrollo arquitectónico y de detalle aplicados más al diseño que a la misma implementación. De la tesis doctoral (Gamma, E., 1991) la cual dio origen a los patrones que actualmente se aplican en la ingeniería de software, surgieron 23 patrones organizados en 3 grupos de acuerdo a su funcionalidad; ellos corresponden a patrones de ***creación***, ***estructurales*** y de ***comportamiento***.

Figura 1. Patrones de Diseño



Fuente: Metodología de desarrollo de software basada en el paradigmas generativos. (Gil, 2007).

A partir del uso general de patrones y la idea predominante de reutilización, la Arquitectura de Software comienza a dar sus primeros frutos al homogeneizar procesos de desarrollo, estandarizar lenguajes (Lenguajes de Descripción de Arquitecturas o ADL's) y al implementar el uso de un verdadero estilo arquitectónico. También hace parte de estos frutos el uso de las Vistas Arquitectónicas implementados por Frameworks como 4+1 basado en SOA, TOGAF, RM/ODP e IEEE y más recientemente el framework propuesto por Roy Thomas Fielding denominado REST (Fielding, 2000). REST propone orientar toda la capacidad de la Arquitectura de Software al desarrollo de soluciones en el marco de las nuevas tecnologías de internet y los modelos orientados a servicios o SAS y recursos distribuidos.

En la actualidad, la Arquitectura de Software se orienta a las LSP o Líneas de Producto Software. El Instituto de Ingeniería de Software de la Universidad Carnegie Mellon (SEI, 2012), define a una SPL o LSP como: "*…un conjunto de sistemas de software*

que comparten un conjunto común y gestionado de características que satisfacen las necesidades específicas de un segmento de mercado particular o misión, y que son desarrolladas de forma prescrita a partir de un conjunto común de elementos clave…". En este contexto, una SPL permite acotar el tiempo de producción de componentes software, con altos niveles de calidad y una alta capacidad de reutilización.

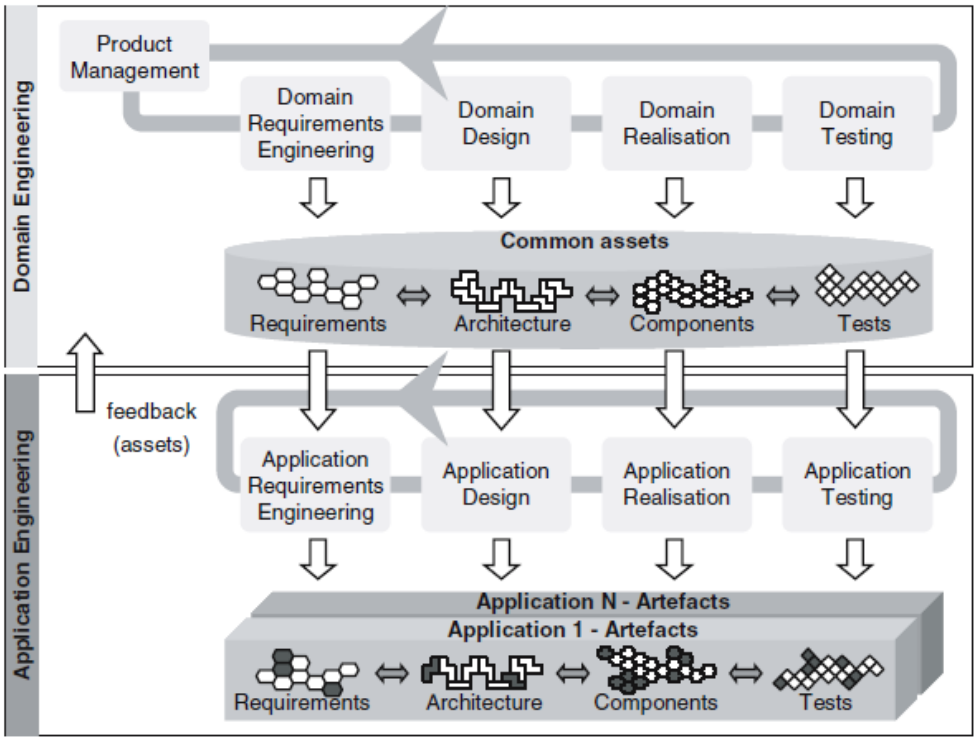
Como enfoque de desarrollo, la SPL utiliza la reutilización como un componente fundamental para el desarrollo de herramientas software flexibles. La reutilización valida los objetos y sus características, realiza abstracciones y a partir de estas validaciones genera nuevos modelos o derivaciones. A partir de este concepto, una SPL puede concebirse a partir de un enfoque guiado por modelos MDD a través de una serie de transformaciones (Arboleda, Casallas, & Royer, 2007).

MDD es una aproximación para solucionar el problema asociado a la complejidad de cada plataforma tecnológica y la inhabilidad que experimentan los lenguajes de propósito general en aliviar esta complejidad (Miller & Mukerji, 2003).

Un enfoque de Desarrollo Dirigido por Modelos (***Model Driven Development***), establece que para el desarrollo de un componente software, solo habrá de diseñarse un modelo específico que corresponda a la abstracción de la realidad que se desea representar y a partir de este modelo se generarán los componentes, especificaciones y código para diversas plataformas separando los detalles arquitectónicos y de implementación, permitiendo la generación de uno o varios modelos a partir de un mismo PIM (***Platform Independent Model***). Con la implementación de este enfoque se adiciona mayor flexibilidad, interoperabilidad, flexibilidad y reutilización de componentes en una línea de producto software, minimizando el tiempo de desarrollo y ahorrando de manera sustancial la producción de líneas de código. La implementación más conocida de MDD se denomina Model Driven Architecture (MDA).

Según la (ISO 25000, 2005), antigua norma ISO/IEC9126, define la usabilidad como "*la capacidad que tiene un software para ser comprendido, aprendido, usado y ser atractivo para el usuario, en condiciones específicas de uso*". En el proceso de ingeniería para desarrollo de líneas de producto software, la usabilidad se convierte en un factor fundamental que flexibiliza la factorización de componentes software gracias a su capacidad para mantener componentes comunes entre sistemas a través del uso eficiente de modelos y sus transformaciones. La variabilidad y combinación de modelos y la adaptabilidad de este concepto a cualquier arquitectura, supone que la usabilidad permite la reutilización de bases comunes tanto en la capa de ingeniería de dominio como en la ingeniería de la aplicación. (Fig. 2).

Figura 2. Etapas y Subprocesos en un Ciclo de Vida para SPLE[3]



Fuente: Software Product Lines in Action (Van Der Linden, Schmid, & Rommes, 2007).

La SPLE consta de dos etapas: Ingeniería de Dominio e Ingeniería de Aplicación. Ambas etapas siguen el esquema de un ciclo tradicional para diseño y desarrollo de software involucrando en sus procesos las fases de: análisis de requerimientos, diseño arquitectónico, implementación y pruebas.

La ingeniería de dominio es la encargada de diseñar los componentes de software que de acuerdo a su trazabilidad, hacen parte del dominio común y que son sujetos a reutilización. En esta etapa se definen las similitudes y variantes del modelo de dominio en cada una de las fases. Como producto final en esta etapa se considera un componente altamente reusable y flexible que permita a partir de su definición, la consecución de nuevos modelos y artefactos.

La Ingeniería de Aplicación se encarga de articular los artefactos definidos en la etapa de dominio a partir de una plataforma específica. El objetivo de esta etapa consiste en lograr la integración de artefactos bajo uno o varios sistemas, logrando con ello un alto grado de reusabilidad de sus componentes a partir de las especificaciones, la definición y desarrollo, documentación e interoperabilidad entre componentes.

[3]***SPLE***; Software Product Lines Engineering.Ingeniería para líneas de productos software.

Para finalizar, se definirán y diferenciarán las dos disciplinas de las que trata este referente histórico.

- La **Arquitectura de Software** según el estándar IEEE 1471-2000 se refiere a la misma como la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución.

- Según Paul Clements, la **Arquitectura de Software** es, a grandes rasgos, "*una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se la percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema. La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones*". (Clements & Nortrhop, "Software architecture: An executive overview, 1996).

- La **Ingeniería de Software** es según el estándar IEEE 610.12.1990, la aplicación de una estrategia sistemática, disciplinada y cuantificable al desarrollo, aplicación y mantenimiento del software; esto es, la aplicación de la ingeniería al software.

- De acuerdo con (Sommerville, 2005), La **Ingeniería de Software** es una disciplina de la Ingeniería que concierne a todos los aspectos de la producción de software. En esta disciplina, los Ingenieros de Software adoptan un enfoque sistemático para llevar a cabo su trabajo y utilizan las herramientas y técnicas necesarias para resolver el problema planteado, de acuerdo a las restricciones de desarrollo y recursos disponibles.

3

CALIDAD DEL PRODUCTO SOFTWARE

"La mejor forma de predecir el futuro es implementarlo" - David Heinemeier Hansson

3.1. EL PROCESO DEL SOFTWARE

Antes de que existiera la Ingeniería del Software como disciplina, La mayoría del software se desarrollaba y era utilizado en forma limitada. La misma persona lo escribía, lo ejecutaba y, si fallaba, lo depuraba. Debido a este entorno personalizado del software, el diseño era un proceso implícito, costoso y generaba dependencias marcadas entre el diseñador y la empresa donde se aplicaba el software.

A raíz de la crisis del software en los 60's dada por la inoperancia en los métodos de desarrollo con pocos o escasos enfoques a la calidad del producto de software, nace la Ingeniería del software como una nueva área de la Ingeniería, enfocada al establecimiento de normas y métodos ingenieriles para obtener **software de calidad** (Pressman, 2002). La Ingeniería del software es una disciplina o área de la Informática o Ciencias de la Computación, que ofrece procesos, métodos y técnicas para desarrollar y mantener software de calidad que resuelven problemas de todo tipo.

El proceso software es el medio por el cual las personas, procedimientos, métodos y herramientas se integran para producir los resultados deseados (Medina D., 2010). Asimismo, el proceso software también puede ser definido como el conjunto de actividades, métodos, prácticas y transformaciones que las personas utilizan para desarrollar y mantener el software y los productos asociados (planes de proyecto, documentación de diseño, código, casos de prueba, manuales de usuario)(SEI, 2006).

La Mejora de Procesos se define como una estrategia que permite a las organizaciones trabajar de forma más eficiente, mejorando los procesos de desarrollo y calidad en los productos software, incrementando sustancialmente la productividad y obteniendo mayores beneficios al momento de su implementación. Alrededor del mundo esta tendencia ha sido incluida como componente fundamental de la organizacional, debido a su alto impacto que involucra el uso de métodos y técnicas para el desarrollo sostenible y buenas prácticas de ingeniería de software en un mercado cada vez más competitivo.

La mejora continua de los procesos se convierte en la razón de ser en todo proceso de calidad, ya que de ello dependen el éxito y consecución de las metas en la organización gestionando de manera eficiente los recursos y mejorando el rendimiento de los procesos los cuales a su vez repercuten en la disminución de los costos y tiempo de implementación.

La mejora de procesos es una actividad cíclica que interviene en cada una de las líneas o fases de un proceso y tienen lugar en la provisión de bienes o servicios. La mejora interviene a lo largo de toda la cadena del producto, desde que inicia hasta que llega al cliente final o a la satisfacción de un objetivo propuesto. La mejora puede darse en aspectos tan importantes que no solo incluyen precisamente cadenas de producción a nivel de desarrollo de software, también tiene que ver con la imagen que se transmite al entorno, la resolución e innovación de productos, bienes o servicios y la forma en cómo nuestros productos son consumidos por el cliente o cómo interactúan los proveedores con la cadena de suministro.

Todas estas capacidades son susceptibles de mejora diaria mediante reorganización de procesos, implantación de nuevas tecnologías, conectividad, sistemas de información y el análisis propio de los datos "***Business Intelligence***". La finalidad de todo este ciclo es mantener el producto en altos grados de calidad, lo que se traduce en consumo, efectividad y competitividad.

Figura 3. Proceso de Mejora Continua

En la figura anterior, la ***responsabilidad empresarial*** se orienta hacia la consecución de las metas propuestas por la organización y el compromiso constante con la mejora. ***El análisis de la mejora*** corresponde a aquellos factores críticos que se presentan en un proceso y que limitan la ejecución y optimización del flujo de información entre procesos. En este punto se identifican las debilidades, fortalezas e interacciones y se identifican potenciales cuellos de botella. ***La gestión de recursos*** juega un papel importante en la gestión de la calidad ya que en ella se encuentra inmerso el factor de cambio que optimiza los recursos existentes y es aquí donde se itera de alguna manera cada uno de los procesos, haciendo reingeniería e identificando posibles fallos en el proceso.

3.1.1. Concepto de Calidad del Software

Según la norma ISO (ISO 8402, 1994), **Calidad** se define como "*un conjunto de características de una entidad que le confiere la aptitud para satisfacer las necesidades establecidas y las implícitas*". ISO en su norma para la Gestión de la Calidad (ISO 9000, 2000) la define como el "*Grado en el que un conjunto de características (rasgos diferenciadores) inherentes cumplen con unos requisitos (Necesidad o expectativa establecida, generalmente implícita u obligatoria)"*. La calidad surge en el momento que se desea producir un bien o servicio conforme a las necesidades de un mercado abierto y globalizado que se encuentra en constante evolución.

En la actualidad, la **Calidad** es un factor de competitividad y una característica determinante para posicionar el negocio a nivel global. Es un sello que garantiza las buenas prácticas y el desarrollo de productos con altos estándares y como objetivo central propende por la satisfacción del cliente, lo que redunda en beneficios tanto para quien lo provee como para quien lo consume.

Ajustando la definición de Calidad a la disciplina de Ingeniería del Software se puede decir que la calidad del *software* es el conjunto de cualidades y características que determinan la utilidad o la existencia de un producto de software y que se enfocan a la resolución de problemas en una forma eficiente. La calidad es sinónimo de eficiencia, flexibilidad, corrección, confiabilidad, mantenibilidad, portabilidad, usabilidad, seguridad e integridad (Fernández C. & García L., 2013). Según (Pressman, 2002), la calidad del software es "la concordancia con los requerimientos funcionales y de rendimiento explícitamente establecidos, con los estándares de desarrollo explícitamente documentados y con las características implícitas que se espera de todo software desarrollado profesionalmente".

3.1.2. Factores de Calidad

Desde un punto de vista sistémico, se puede definir el factor de calidad en el software en 4 elementos: Eficiencia del Producto, Efectividad del Producto, Eficiencia del Proceso y Efectividad del Proceso, considerando las dimensiones del Cliente y del Usuario (Callaos & Callaos, 1996).

Desde este enfoque, la perspectiva de calidad en el software concibe el sistema en conjunto como un producto sinérgico donde:

- ***La Eficiencia del Producto*** se consigue a través de la aplicación de buenas prácticas en lo que corresponde al diseño físico y al desarrollo de algoritmos eficientes,

- ***la Efectividad del Producto*** se relaciona directamente con el factor ergonómico que busca la optimización de las interfaces (humano-máquina-entorno), a través del análisis e identificación de requerimientos funcionales y no funcionales,

- ***la Eficiencia del Proceso*** contribuye a la productividad, al cumplimiento de metas y a la disminución de recursos, optimizando los tiempos de entrega y gestionando de manera eficiente los cronogramas de actividades y planes para el desarrollo de módulos o aplicaciones,

- por último, ***la Efectividad del Proceso*** se orienta hacia la gestión de recursos (humanos, técnicos, tecnológicos, organizacionales y estratégicos) que sumados en su conjunto contribuyen al desarrollo de sistemas informáticos mejor pensados con un alto nivel operativo.

La calidad del software no es algo que depende de una sola característica o factor en particular, sino que obedece a la sincronización y colaboración de las partes involucradas (Eficacia, Eficiencia, Efectividad) (Callaos & Callaos, 1996). La calidad de un producto de software está influenciada por la calidad en el proceso de desarrollo, las cuales se determinan a través de una serie de características. Estas definiciones se encuentran detalladas en el estándar internacional ISO/IEC 9126(ISO, 1991).

3.2. MEDIDA EN LA CALIDAD DEL SOFTWARE

La medición siempre ha sido una herramienta fundamental para el desarrollo de las ciencias y el conocimiento. Medir implica conocer, y ese conocimiento es la base de cualquier procedimiento y técnica que se desee implementar. En el ámbito científico el medir contribuye a la generación de teorías y nuevas perspectivas que abren un amplio margen de conceptos. Las mediciones se realizan a través de métodos cuantitativos que involucran en su mayoría conceptos matemáticos para la resolución y verificación de los procesos.

A mediados de la década de los 60’s se dio la denominada “Crisis del Software”. En este contexto, el desarrollo de soluciones software solo se enfocaba hacia la resolución de problemas específicos. Estos programas de ordenador poseían pobres diseños además de contar con serios fallos a nivel estructural, que eran por lo general difíciles de resolver. Lo anterior sumado a la pobre calidad en los sistemas, plazos de entrega tardíos y los

elevados costes de mantenimiento y desarrollo, llevaron a la OTAN[4] en 1968 a organizar una conferencia en Garmisch-Partenkirchen (Alemania); donde se reunieron investigadores de todo el mundo y especialistas en la disciplina de la informática, para debatir y proponer soluciones para mejorar las técnicas y procedimientos para la elaboración de software. De esta cita surgió la disciplina de la ingeniería del software,

La ***Ingeniería de software*** es la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento de software, y el estudio de estos enfoques, es decir, la aplicación de la ingeniería al software (IEEE Computer Society, 2004). Otra definición muy asertiva corresponde a la descrita por la ACM[5], la cual define a la Ingeniería de software como la aplicación de las ciencias y la ingeniería al desarrollo del software, ya que integra matemáticas, ciencias de la computación, ciencias exactas y prácticas cuyos orígenes se encuentran en la ingeniería (ACM, 2006).

En 1986, Norman Fenton en su artículo "***Software Metrics, a Rigorous Approach***", concluye que pasados 20 años desde que se acuñara el término Ingeniería de software en la reunión de Garmisch-Partenkirchen, el software seguía teniendo serios inconvenientes de desarrollo debido a que no se contaba con una férrea disciplina para la medición de los factores que intervienen en el desarrollo de un software con estándares de calidad.

Teniendo en cuenta lo anterior, Fenton afirmó que una de las razones principales del incremento masivo en el interés en la medición software ha sido la percepción de que las métricas son necesarias para la mejora de la calidad del proceso (Fenton & Pflegger, 2001), ya que el uso de las métricas ayudan a comprender lo que sucede durante el proceso de desarrollo, permite controlar la evolución e implementación de los proyectos y ayuda a mejorar los flujos de procesos tanto en el nivel de factoría como la comprensión y la especificidad de las herramientas software.

3.3. MÉTRICAS DE SOFTWARE

Las métricas se utilizan a lo largo del ciclo de vida del software. La medición de procesos en la ingeniería del software tiene como objetivo fundamental satisfacer necesidades de información partiendo de las entidades (objetos) y sus propiedades (atributos) y que son susceptibles de medición. A través de las métricas, se persigue la eficacia del proyecto, la optimización del producto y la calidad del mismo. La medición en el software es necesaria porque:

- Permite entender en sí el proceso de producción y desarrollo de una herramienta de software,
- permite conocer y determinar cuáles herramientas deben ser utilizadas. Estas deben contribuir al mejoramiento y optimización de los procesos de desarrollo a lo largo de todo el ciclo de vida del software,

[4] OTAN, NATO, "***Organización para el Tratado del Atlántico Norte***"
[5] **ACM**; Association for Computing Machinery "*Advancing Computing as a Science & Profession*". Sitio web: http://www.acm.org

- permite establecer estrategias efectivas para la gestión y planificación de procesos, datos, recursos y talento humano,
- con la medición es posible determinar cuál es el grado de cumplimiento y nivel de productividad alcanzado en cada una de las fases de desarrollo,
- es evidente que la finalidad que persigue la medición es ***Mejorar*** el producto de software, incrementando sus factores de calidad (qué hace?, cómo lo hace?, dónde se aplica?, etc; lo que redunda en reducción y optimización de costos, procesos y aumento en la calidad y la competitividad del software.

A la luz del método científico, el proceso de medición, se puede resumir en 5 etapas o actividades:

- ***Formulación***: Corresponde a la cuantificación y obtención de medidas y métricas del software que permitan la representación de los componentes de software, sus variables, dependencias y relaciones.

- ***Recolección***: Consiste en un mecanismo que permite recolectar la información necesaria para realizar las mediciones o cuantificación de las variables del software, y que permitan validar las métricas formuladas.

- ***Análisis***: Corresponde al cálculo de las métricas y la aplicación de herramientas matemáticas, estadísticas y fácticas para el análisis de la información recolectada.

- ***Interpretación***: Permite la evaluación de resultados partiendo de las métricas empleadas que permitan gestionar un componente con calidad.

- ***Retroalimentación***: Son todas aquellas recomendaciones obtenidas de la interpretación de las métricas y las técnicas que serán transmitidas al equipo de desarrollo de software, para la mejora del producto.

3.3.1. Clasificación de las Métricas

Las métricas de software se pueden catalogar en 2 grupos; Métricas que se orientan al ***Producto*** y Métricas que se orientan al ***Proceso***. En la actualidad existen innumerables métricas que permiten determinar o describir la conducta y funcionalidad del software. Estás métricas se implementan con la visión de medir aspectos como: la calidad, el desempeño, la complejidad, la productividad, las técnicas de mantenimiento, líneas de código, errores del producto, cronograma de metas, etc; que contribuyen al establecimiento de reglas y procedimientos que permitan de forma sistemática y sinérgica, obtener una visión real del producto y su mejora.

A continuación, en la Tabla 1 se resumen algunas características y tipos de métricas utilizadas en software. (Eijogu, 1988):

Tabla 1. Clasificación de Métricas empleadas en el desarrollo de Software

MÉTRICA	DESCRIPCIÓN
Métricas de Calidad	Se orientan a la medición de: exactitud, funcionamiento de las estructuras de datos y componentes, pruebas de unidad, mantenimiento, reusabilidad, entre otras. Estas métricas permiten evaluar la calidad en la codificación de algoritmos, las pruebas y el mantenimiento.
Métricas de Desempeño	Se orientan a la medición de: comportamiento de módulos y el sistema de software y su interoperabilidad con los lenguajes de programación, sistemas operativos y componentes del hardware. Estas métricas evalúan el desempeño del sistema en los sistemas de memoria y procesamiento, almacenamiento en dispositivos y la eficiencia de algoritmos implementados.
Métricas de Refinamiento	Se orientan a la medición de: estilos de codificación, reglas nemotécnicas y uso de variables en entornos de programación, y limitaciones del sistema.
Métricas de Complejidad	Se orientan a la medición de: la complejidad (volumen, tamaño, anidaciones, estimación de costos y configuración). Estas métricas se encargan de evaluar y analizar la viabilidad del diseño.
Métricas de Competencia	Se orientan a la medición de: rapidez, eficiencia y competencia del equipo de desarrolladores. Esta medición es importante ya que de ella deriva en parte el éxito del proyecto, habilidades, aptitudes, trabajo en equipo e innovación, también son factores que determinan la competencia y conocimientos del equipo de trabajo.
Otras métricas	Se orientan a la medición de: portabilidad, disponibilidad, localización, y otros factores (por lo general externos), que afectan el desarrollo del producto.

3.3.2. Métricas usadas en desarrollo de Software

En la Tabla 2 se describen algunos elementos que han sido tomados de los modelos de calidad (ISO-9126, 1982), (McCall & Richards, 1977), (Boehm, Brown, Kaspar, & Lipow, 1978) e (IEEE Std729, 1983), como marco para el desarrollo de software de calidad. Se debe hacer énfasis que un modelo de calidad básicamente obedece a un enfoque, y su aplicación depende de las variables que se deseen medir y cuantificar.

Tabla 2. Resumen de Métricas en Software

MEDIDA	DESCRIPCIÓN
Confiabilidad	Es la capacidad que tiene un programa de computadora para operar de forma ininterrumpida en un entorno determinado y en un tiempo específico sin que éste presente falla alguna.
Comprensibilidad	Esta característica es una cualidad interna del software y permite al usuario interactuar de manera eficaz con los componentes o funcionalidades en una forma sencilla y amigable. ***Si es fácil de usar, es fácil de comprender***. No obstante, no todos los sistemas se hacen con esta métrica, pues habrá casos donde la computación exige de métodos y modelos matemáticos más complejos que dificultan la amigabilidad con el usuario.
Corrección	Esta propiedad establece la relación directa que existe entre los algoritmos, los cálculos y la precisión del software para responder a las funcionalidades para las cuales fue diseñado cada componente o conjunto. ***¿Hace lo que se le pide con exactitud y eficiencia?***
Eficiencia	Es la capacidad que tiene un programa de computadora para hacer buen uso de los

MEDIDA	DESCRIPCIÓN
	recursos informáticos (software, Hardware y Sistemas Operativos) en donde se ejecuta. ***¿Qué recursos hardware y software utiliza y cómo es su tiempo de respuesta?***
Escalabilidad	La escalabilidad describe la facilidad con la que pueden agregar o quitar componentes de un sistema a la vez que se mantiene la confiabilidad del mismo y su calidad.
Mantenibilidad	Esta característica tiene que ver con la capacidad que tiene un aplicativo de software para permitir de manera eficiente correcciones a errores o bugs posteriores a su implementación. Existen 3 tipos de pruebas a este nivel y corresponden a: mantenimientos ***correctivos, adaptativos y perfectivos***. El primero corresponde a la eliminación de errores presentes en el producto al ser implementado o a errores introducidos en su etapa de mantenimiento. El segundo permite realizar ajustes a la aplicación conforme a cambios realizados en el entorno (cambios a nivel de bases de datos, acceso a medios, cambios de hardware o nuevas versiones y funcionalidades en sistemas operativos). Por último, el mantenimiento perfectivo hace referencia a nuevas funcionalidades que una vez introducidas en el sistema informático, dan mayor performance a la aplicación facilitando su utilización e incrementando la calidad del producto software.
Facilidad de prueba	Las pruebas de software involucran a aquellas actividades que permiten verificar la operatividad del sistema bajo condiciones controladas y evaluando los resultados. El objetivo fundamental de esta métrica pretende la introducción intencional de errores para forzar a la aplicación a generar fallos, para determinar si los errores ocurren cuando no tendrían que ocurrir o cuando los errores no son reportados cuando se presentan.
Fiabilidad	La fiabilidad se define como la probabilidad que tiene una herramienta de software para realizar sus tareas y objetivos de manera eficiente y satisfactoria (sin la presencia de fallos), en un determinado periodo de tiempo y en un entorno definido.
Facilidad de Aprendizaje	Está relacionada con la predictibilidad, la amigabilidad, la sintetización, la familiaridad, la generalización de los conocimientos previos y la consistencia de la aplicación. Esta cualidad facilita que nuevos usuarios desarrollen una interacción más efectiva con el sistema o producto, sin necesidad de recurrir a extensas capacitaciones o a la elaboración de manuales técnicos bastante complejos.
Flexibilidad	Esta cualidad determina la capacidad que tiene la herramienta software para intercambiar información con el usuario en forma variada. Abarca la posibilidad de diálogos y mensajes informativos, la multiplicidad de vías para realizar las tareas y la optimización del flujo de información entre el usuario y el sistema.
Integridad	Corresponde a la capacidad que tiene un sistema software para proteger sus componentes (programas, datos, etc.), contra modificaciones y accesos no autorizados.
Interoperabilidad	Es la capacidad que tiene un sistema software para interactuar o comunicarse con otros sistemas por medio de determinados protocolos, interfaces de aplicación y otros mecanismos indirectos.
Portabilidad	Se define como la característica que posee un software para ejecutarse en diferentes plataformas. En este caso, el código fuente del software es capaz de reutilizarse en vez de crearse un nuevo código cuando el software pasa de una plataforma a otra. A mayor portabilidad menor es la dependencia del software con respecto a la plataforma.
Reusabilidad	Reusar significa reciclar, y en este sentido se aplica a la capacidad que tiene uno o varios componentes de software para generar otros componentes o aplicaciones diferentes derivados de un código propio. Al reutilizar se escribe menos código y en consecuencia se puede invertir mayor tiempo en otros factores para mejorar la calidad del producto.
Robustez	La robustez es una acción complementaria de la Corrección. Está relacionada con la capacidad de recuperación de información y de ajuste de la tarea al usuario. La

MEDIDA	DESCRIPCIÓN
	robustez asegura que no se causen eventos de índole grave o catastrófica al momento de ejecutarse una aplicación. De presentarse errores imprevistos en el software, la robustez debe hacer un despliegue de mensajes de error apropiados, en donde se pueda terminar la aplicación de manera limpia y segura para los datos.
Usabilidad	Se refiere a la capacidad de un software de ser comprendido, aprendido, usado y ser atractivo para el usuario, en condiciones específicas de uso.
Funcionalidad	La funcionalidad hace referencia a la capacidad que tiene un elemento software para satisfacer las necesidades de uso específico y a nivel operativo y de gestión.
Adecuación	Corresponde a la capacidad que tiene un sistema para definir un conjunto de reglas y funciones apropiadas para una tarea específica o usuarios específicos.
Seguridad	Es una medida de calidad que permite a los sistemas software proteger la información y datos de tal manera que procesos no autorizados o usuarios no certificados, substraigan, usurpen, roben o procesen información sin los debidos niveles de acceso.
Rendimiento	Es una medida orientada a la evaluación de conformidades de un sistema software o sus componentes con base en un conjunto específico de requerimientos. Para realizar las optimizaciones, a menudo se requerirá de herramientas automatizadas para simular un gran número de usuarios, carga y volumen de información para monitorear el comportamiento del software y el hardware asociado. El rendimiento en sí evalúa el comportamiento ante situaciones extremas y saturación de procesos.
Disponibilidad	La disponibilidad es la probabilidad de que un sistema sea capaz de operar cuando le sea requerido, bajo circunstancias especificadas. La disponibilidad de un sistema tiene que ver con las fallas de este y las consecuencias asociadas. Una falla de sistema ocurre cuando el sistema no proporciona un servicio en el tiempo especificado, una falla como tal es observada por los usuarios del sistema.
Modificabilidad	La Modificabilidad determina cual será el costo promedio de esas nuevas implementaciones sin que éstas modifiquen otros componentes asociados al modulo sobre el cual se está efectuando la modificación o reestructuración.

4

ARQUITECTURA DE SOFTWARE

"Tenemos que cambiar la tradicional actitud ante la construcción de software. En vez de pensar que nuestra principal tarea es indicar a un ordenador qué hacer, concentrémonos en explicar a las personas lo que queremos que el ordenador haga"
-- Donald E. Knuth

Para (Booch, Rambaugh, & Jacobson, 1999) el término **Arquitectura** hace referencia al conjunto de decisiones significativas acerca de la organización de un sistema, la selección de sus elementos estructurales y las interfaces que componen el sistema, junto con la colaboración entre los elementos estructurales y la descomposición en subsistemas según el funcionamiento de estos elementos. Por tanto, la arquitectura se compone de elementos que forman el sistema, sus interfaces, la colaboración entre ellos y su composición.

Para (Shaw & Garlan, 1996) el término **Arquitectura**, asocia las capacidades del sistema especificadas en el requerimiento con los componentes del sistema que habrán de implementarla. La descripción arquitectónica incluye componentes y conectores (en términos de estilos) y la definición de operadores que crean sistemas a partir de subsistemas o, en otros términos, componen estilos complejos a partir de estilos simples.

Para (Boehm & Port, 1999), una **Arquitectura** comprende un conjunto de componentes del sistema, conexiones y restricciones, un conjunto de requisitos del sistema y unos marcos de razonamiento que demuestran que los componentes, sus conexiones y restricciones que definen el sistema, una vez implementados, satisfacen el conjunto de requisitos del sistema.

Según (Jazajery, Ran, & Linden, 2000), la **Arquitectura** de un sistema es un conjunto de conceptos y decisiones de diseño sobre la estructura del sistema, que deben ser tomadas en cuenta antes de su realización para satisfacer de una manera efectiva los requisitos de funcionamiento y calidad implícitos en el sistema.

La definición de **Arquitectura de Software** aceptada mundialmente corresponde a la publicada por la IEEE en su estándar 1471:2000. La Arquitectura de Software según (Bass, Len; Clements, Paul; Kazman, Rick, 2003), se define como:

> "*…la estructura del sistema, que comprende elementos software, las propiedades de esos elementos visibles externamente y las relaciones entre ellos. La arquitectura software conforma el esqueleto de cualquier sistema software, y es la principal responsable de los atributos de calidad del sistema. Una arquitectura adecuada, correctamente diseñada, documentada y evaluada, constituye la base para que un proyecto finalice con éxito…*".

El estándar IEEE1471:200, establece que una Arquitectura de un sistema debe responder a los siguiente elementos u objetivos:

1) Especificación de Actores del sistema.

2) Identificación y descripción de objetivos del sistema en términos de métricas de calidad tales como:

- ✓ ***Funcionalidad***, hace referencia a la capacidad que tiene un elemento software para satisfacer las necesidades de uso específico y a nivel operativo

y de gestión. La funcionalidad plantea el siguiente interrogante ¿Qué necesita hacer el sistema?

- ✓ ***Rendimiento***, que deben ir orientadas a la evaluación de conformidades de un sistema software o sus componentes con base en un conjunto específico de requerimientos. Para realizar las optimizaciones, a menudo se requerirá de herramientas automatizadas para simular un gran número de usuarios, carga y volumen de información para monitorear el comportamiento del software y el hardware asociado. El rendimiento en sí evalúa el comportamiento ante situaciones extremas y saturación de procesos.

- ✓ ***Seguridad***, una medida de calidad que permite a los sistemas software proteger la información y datos de tal manera que procesos no autorizados o usuarios no certificados, substraigan, usurpen, roben o procesen información sin los debidos niveles de acceso.

- ✓ ***Viabilidad***, planteándose si realmente se puede implementar o no el sistema.

3) Organización modular y control de dominios a través de la especificación de una o más vistas de la misma arquitectura.

4) Planificación de Objetivos en base a decisiones arquitectónicas que ofrezcan una posibilidad de aplicación de tácticas, alternativas o estrategias para la toma de decisiones mejorando los modelos y la calidad del producto software.

4.1. CONCEPTOS FUNDAMENTALES EN LA ARQUITECTURA DE SOFTWARE

La Arquitectura de Software se articula en base a una serie de conceptos y características; ellos son, los estilos arquitectónicos, los lenguajes de descripción arquitectónica ADL, los Frameworks y Vistas, la Abstracción, las metodologías y los escenarios.

4.1.1. Estilos Arquitectónicos

Según (Klein & Kazman, 1999) un **Estilo** Arquitectónico se define como la abstracción de distintas arquitecturas software. Perry y Wolf la definen como un concepto descriptivo que define una forma de articulación u organización arquitectónica. Los estilos arquitectónicos están compuestos por componentes, conectores, restricciones (*constraints*) y configuraciones. Un estilo arquitectónico define una familia de sistemas (cierto tipo de sistemas) en términos de patrones estructurales, de control, de comunicación, etc. (Sommerville, 2005).

Un estilo arquitectónico describe:

- Un conjunto de componentes (con sus responsabilidades);
- Un conjunto de conectores entre componentes (comunicación, coordinación, cooperación, etc.);
- Restricciones que definen cómo se integran los componentes para formar el sistema y
- Una serie de Modelos que permiten comprender las propiedades de un sistema general en función de las propiedades conocidas de las partes que lo integran.

Dicho en otras palabras, los estilos arquitectónicos sirven para:

- Sintetizar estructuras de soluciones;
- Es un concepto abstracto que permite encapsular de manera eficiente un conjunto de configuraciones y características concretas;
- Definir los posibles comportamientos o patrones de diseño de las aplicaciones y
- Evaluar las posibles soluciones a través de alternativas de diseño a partir de un conjunto de requerimientos.

A continuación se relaciona el conjunto de estilos arquitectónicos más empleados en el ámbito de la Arquitectura de Software.

Tabla 3. Estilos Arquitectónicos

ESTILO ARQUITECTÓNICO	ESTRATEGIA
ESTILOS DE FLUJO DE DATOS	Tubería y filtros
ESTILOS CENTRADOS EN DATOS	Arquitecturas de Pizarra o Repositorios
ESTILOS DE LLAMADA Y RETORNO	Model-View-Controller (MVC) Arquitecturas en Capas Arquitecturas Orientadas a Objetos Arquitecturas Basadas en Componentes
ESTILOS DERIVADOS	C2 GenVoca REST
ESTILOS DE CÓDIGO MÓVIL	Arquitectura de Máquinas Virtuales
ESTILOS HETEROGÉNEOS	Sistemas de control de procesos Arquitecturas Basadas en Atributos
ESTILOS PEER-TO-PEER	Arquitecturas Basadas en Eventos Arquitecturas Orientadas a Servicios (SOA) Arquitecturas Basadas en Recursos

4.1.2. Lenguajes de Descripción Arquitectónica

Un ADL (Lenguaje de Descripción de Arquitectura) es un lenguaje que se utiliza para describir de forma abstracta la arquitectura de una herramienta software. Los Lenguajes ADL proporcionan mecanismos para modelar la arquitectura conceptual, centrándose específicamente en la estructura de los componentes y no en la implementación de los módulos en sí. Los ADL permiten modelar una arquitectura sin programar códigos, a partir de un conjunto de atributos para determinar puntos críticos y simular comportamientos del software mucho antes de su implementación.

La definición de estos lenguajes más que ser un estándar en la industria del software, corresponden a propuestas netamente académicas de universidades e investigadores de la ingeniería del software alrededor del mundo. Los ADL surgieron en paralelo con el Lenguaje Unificado de Modelado UML en la década de los 90's.

Los lenguajes ADL están compuestos por 3 elementos arquitectónicos a saber. Ellos son: los ***componentes***, los ***conectores*** y las ***configuraciones***.

- **Componentes.** Elementos arquitectónicos que representan y describen las funcionalidades específicas de un modelo o sistema.

- **Conectores.** Elementos arquitectónicos que representan las relaciones entre los distintos componentes que componen la arquitectura. También se encarga de proveer los mecanismos para representar la comunicación entre los elementos representados en el modelo.

- **Configuraciones.** Es un mecanismo que permite fusionar los componentes y los conectores en una única capa. Estas dirigen el comportamiento del sistema de software.

A continuación se detallan algunas de las implementaciones de ADL más conocidas en la actualidad.

4.1.2.1. Architecture Based Design Method (ABD)

Las entradas para el método ABD responden a una lista de requerimientos funcionales (abstractos y concretos), de calidad y de negocio (abstractos y concretos) con soporte en restricciones. Los requerimientos de tipo abstracto son empleados para generar el diseño mientras que los concretos se usan para validar las decisiones producto de los requerimientos abstractos. El método ABD procesa de forma recursiva las entradas (modelo inicial) y realiza sobre éste un proceso de descomposición. Cada proceso de descomposición se conoce como **Refinamiento**.

Tras cada descomposición "Splitting" se agregan nuevos elementos y a cada uno de ellos se les asigna una nueva responsabilidad. Los niveles de descomposición en ABD son

examinados desde la perspectiva de las vistas a nivel lógico, de concurrencia y de desarrollo. (Bachmann, Bass, Chastek, Donohue, & Peruzzi, 2000).

4.1.2.2. Active Design Reviews (ADR)

ADR es un método utilizado para la evaluación de diseños detallados a nivel de unidades de software o componentes. ADR asegura la participación activa de los distintos Stakeholders "involucrados" en el proceso de desarrollo a través del uso de cuestionarios enfocados en el área específica. (Parnas, David; Weiss, David, 1985). Estos cuestionarios se aplican en base al grado de completitud del proceso, la calidad del módulo, la suficiencia, la documentación que soporta el artefacto software o diseño y la conveniencia de los servicios que provee el diseño propuesto.

4.1.2.3. Active Reviews for Intermediated Designs (ARID)

ARID incorpora fuertes cualidades del método ADR y ATAM. Este método evalúa la idoneidad de los diseños arquitectónicos en las primeras fases de definición. Permite reunir las partes interesadas y los diseñadores en las primeras fases de desarrollo del software. Los involucrados generan los escenarios para analizar la capacidad de la arquitectura, mientras que el uso de escenarios genera una lluvia de ideas por parte de los diseñadores, sobre nuevos diseños y evaluaciones.

Luego de terminado el proceso de evaluación, se pasa a realizar las adecuaciones pertinentes en el diseño de la arquitectura. Este método tiene como principal desventaja que sólo analiza los componentes de la arquitectura, sin tener en cuenta las conexiones que se establecen entre ellos. Unido a esto, está el inconveniente de que no contempla ningún atributo de calidad, sino que sólo evalúa la conveniencia del diseño arquitectónico. (Clements, Paul, 2000).

4.1.2.4. Architecture Tradeoff Analysis Method (ATAM)

Corresponde a una evolución del método SAAM (Software Architecture Analysis Method), el cual se basa en la descripción de **Escenarios**. ATAM no solo basa su análisis a partir de atributos de calidad como la Modificabilidad "**Modifiability**" o el Rendimiento "**Performance**", sino que también hace uso de esos mismos atributos y los aplica entre sí para ver que tanto afecta el uno al otro en torno a una variable o atributo del modelo. El proceso de análisis de la arquitectura incluye la identificación de:

- **Riesgos**: decisiones aplazadas o decisiones cuyo efecto no se alcanza a valorar.
- **Puntos sensibles**: partes de la arquitectura que pueden tener mucha influencia en algún atributo de calidad.

- **Puntos de compromiso**: partes de la arquitectura cuya modificación significa mejorar algún atributo de calidad a costa de empeorar otro. Es lo que sucede, en algunos casos, con la modificabilidad y el rendimiento.

El objetivo de ATAM es poder tomar decisiones razonadas acerca del diseño para, en sucesivos análisis, dedicar más esfuerzo a completar esas partes de la arquitectura. (Kazman, Klein, & Clements, CMU/SEI-2000-TR-004 - ESC-TR-2000-004, 2000).

4.1.2.5. Attribute-Based Architectural Styles (ABAS)

Esta arquitectura se basa en la implementación de estilos arquitectónicos. ABAS utiliza los atributos de calidad para definir un marco de aplicación de un estilo arquitectónico concreto, proporcionando un razonamiento, cuantitativo o cualitativo, que justifique la utilización de un determinado estilo arquitectónico en un diseño o proceso de modelado. ABAS construye una arquitectura a partir de **Marcos de Razonamiento** o Reasoning Frameworks basados en atributos de calidad como el Rendimiento "Performance" y la Fiabilidad "Reliability". (Klein & Kazman, 1999).

4.1.2.6. Attribute-Driven Design (ADD)

Es un lenguaje que permite la descripción de un modelo o diseño arquitectónico a partir de atributos de calidad. Este modelo se basa en modelos de comportamiento más que en las propias funcionalidades del sistema. Las entradas para un sistema basado en ADD se modelan a través de **Escenarios** mediante los cuales se formulan los requisitos de calidad. La salida del modelo se centra en una arquitectura conceptual basada en atributos de calidad deseados, proporcionando las alternativas y estructuras necesarias para proveer al sistema de las funcionalidades requeridas para mejorar su capacidad. (Wojcik, y otros, 2006).

4.1.2.7. Cost-Benefit Analysis Method (CBAM)

Es un método para evaluar los beneficios, costes y riesgos de las diferentes decisiones que se toman para diseñar la arquitectura software del sistema. Se basa en ATAM para modelar costos y beneficios de una decisión arquitectural. CBAM provee un marco de evaluación para decisiones arquitecturales de tipo técnico y económico. CBAM se basa en los artefactos producidos por ATAM (Objetivos de negocio, Vistas arquitecturales, Árbol de utilidad, Riesgos identificados y Puntos de Tradeoff) y utiliza dichos atributos para ayudar en el descubrimiento de costos y beneficios generados por la arquitectura.

CBAM al ser una arquitectura basada en la Teoría Costo/Beneficio, permite la evaluación de un modelo en base a **estrategias/tácticas** arquitectónicas para proveer mayor utilidad o rendimiento en el proceso de diseño, a partir del ROI de cada uno de los módulos para tomar una decisión adecuada y conforme a los requerimientos del sistema. CBAM

utiliza un conjunto de escenarios para describir la arquitectura con base en los siguientes elementos: Estímulo, Ambiente, Respuesta y Métricas de respuesta (Variaciones). El objetivo es obtener la curva de utilidad/respuesta en base a dichos elementos y que sean de utilidad para los diferentes stakeholders. Para la curva se utilizan los siguientes parámetros: Best-case (valor 100), Worst Case (valor 0), Current Level y Desired Level.

Al igual que QAW, también está pensado para su integración con ATAM, ya que los resultados producidos por la evaluación de la arquitectura, especialmente las estrategias arquitectónicas a seguir, se utilizan como entrada en CBAM para tomar decisiones basadas en criterios económicos. (Bass, Len; Clements, Paul; Kazman, Rick, 2003).

4.1.2.8. Quality-Attribute-Driven Software Architecture Reconstruction (QADSAR)

QADSAR es un Framework que vincula diversos atributos de calidad para reconstruir una vista arquitectónica. (Stoemer, O'Brien, & Verhoef, 2003). QADSAR concentra su capacidad en (5) cinco pasos: Identificación y Alcance "Scope Identification", extracción del modelo fuente "Source Model", abstracción del modelo fuente "source model abstraction", elementos y propiedades de instanciación "elements and properties instantiation" y la evaluación de atributos de calidad "quality attribute evaluation". QARSAR es considerado como un método eficiente para el diseño de arquitecturas ya que permite la inclusión sistemática de atributos de calidad desde la perspectiva de la reconstrucción de la arquitectura de software. Sin embargo; QADSAR está limitado en su capacidad de reconstrucción debido a que el Framework no hace uso de <vistas hipotéticas> para la reconstrucción de arquitecturas sofisticadas.

4.1.2.9. Quality Attribute Workshops (QAW)

Es un método creado para complementar ATAM. Su utilidad reside en la identificación de los atributos de calidad que dirigen el proceso de diseño de la arquitectura, por lo que su aplicabilidad es previa. (Barbacci, y otros, 2003).

QAW es un método que compromete en forma temprana a los stakeholders ciñendo su capacidad al ciclo de vida del software, para descubrir y dirigir los modelos de desarrollo en base a atributos de calidad "performance, availability, security, y modifiability" para sistemas intensivos en software. Este método se desarrollado para complementar al método ATAM "Architecture Tradeoff Analysis Method" y provee un conjunto de elementos para identificar los atributos de calidad más relevantes del modelo a implementar. También posee una característica que hace que éste método sea eficaz para identificar los requerimientos antes de que la arquitectura de software sea diseñada como tal.

Lo que pretende QAW es definir un conjunto de atributos de calidad con base en el contexto del sistemas y la forma en cómo estos atributos pueden ser priorizados para mejorar el diseño. El resultado de este proceso es una lista de factores que van a dirigir el diseño de la arquitectura y una lista de escenarios que servirán para evaluar los atributos de calidad.

4.1.2.10. Software Architecture Analysis Method (SAAM)

SAAM es una arquitectura orientada fundamentalmente a la aplicación del atributo de Modificabilidad "Modifiability" en una arquitectura de software. Según (Kazman, Bass, Abowd, & Webb, 2007) SAAM permite la evaluación y comparación de arquitecturas en el contexto del negocio y las necesidades propias del modelo. Es así como SAAM se concentra en las actividades claves del negocio y articula aquellos atributos con mayor demanda o de mayor relevancia en el modelo de negocio. SAAM requiere de la definición y selección de tareas específicas donde habrán de probarse e implementarse tales atributos.

4.1.3. Frameworks y Vistas

Un framework, es una abstracción en la que cierto código común provee una funcionalidad genérica que puede ser sobrescrita o especializada de forma selectiva por medio de código con funcionalidad específica provisto por los clientes del framework (desarrolladores de software / programadores).

Un framework facilita el desarrollo de software permitiendo a los diseñadores y programadores dedicar su tiempo a lograr los requerimientos de software en lugar de lidiar con los detalles de bajo nivel necesarios para obtener un sistema funcional, reduciendo así el tiempo de desarrollo.

De acuerdo con la Enciclopedia Social Wikipedia[6], En el ámbito del desarrollo de *software*, un ***framework***, "es una estructura conceptual y tecnológica de soporte definido, normalmente con artefactos o módulos de *software* concretos, que puede servir de base para la organización y desarrollo de *software*. Típicamente, puede incluir soporte de programas, bibliotecas, y un lenguaje interpretado, entre otras herramientas, para así ayudar a desarrollar y unir los diferentes componentes de un proyecto. Un Framework representa una arquitectura de *software* que modela las relaciones generales de las entidades del dominio, y provee una estructura y una especial metodología de trabajo, la cual extiende o utiliza las aplicaciones del dominio".

Un **componente** es un artefacto auto-contenidos, claramente identificable que describe y/o ejecutan funciones específicas y tienen interfaces claras, una documentación apropiada y un estado de reuso definido (Sametinger, 1997). Un componente en otras palabras corresponde a un módulo de bajo acoplamiento y alta cohesión que denota una abstracción simple(Booch, Rambaugh, & Jacobson, 1999). Un componente es entonces cualquier elemento o recurso (por ejemplo, módulos, códigos, diseños, requerimientos, modelos de negocio, documentación, etc.), que puede ser reutilizado para posteriores desarrollos.

A continuación se describen algunos de los Frameworks más utilizados en el ámbito de la Arquitectura de Software.

[6] Definición de Framework, En: http://es.wikipedia.org/wiki/Framework

4.1.3.1. Framework para Arquitecturas Empresariales de Zachman

Contiene 36 Vistas y 6 Niveles. Es un modelo considerado como rígido y difícil de implementar. Muchos autores coinciden en que no es un marco aplicable a la Arquitectura de Software pero destacan la importancia en la definición de 3 vistas (Conceptual, Lógica y Física) de las 36, bajo las cuales se conciben actualmente los Frameworks de desarrollo. Su estructura se representa por: *Alcance, Empresa, Sistema Lógico, Tecnología, Representación y Funcionalidad*.

4.1.3.2. Pattern Oriented Software Architecure "POSA"

Los patrones de arquitectura expresan el esquema fundamental de organización para sistemas de software. Proveen un conjunto de subsistemas predefinidos; especifican sus responsabilidades e incluyen reglas y guías para organizar las relaciones entre ellos (Microsoft, 2013). POSA como arquitectura orientada a las vistas está compuesta de 4 elementos: Lógica, Procesos, Física y Desarrollo.

- La vista **Lógica** de la cual hace parte los objetos de diseño y el diagrama relacional;
- La vista de **Proceso** que se encarga de manejar los mecanismos de concurrencia y sincronización de procesos;
- La vista **Física** que relaciona al software con el hardware y otros soportes para artefactos y modelos en entornos distribuidos; y
- La vista de **Desarrollo** que hace referencia a la producción de módulos o componentes en un entorno de desarrollo.

Este modelo coincide con el de Kruchten, pero no hace énfasis en el 5 elemento denominado Escenarios.

4.1.3.3. C4ISR Architecture Framework (Architecture Working Groups)

Es un Framework desarrollado por el Departamento de Defensa de los EEUU. C4ISR es un marco para integración de actividades de tipo militar a nivel de comando, control, comunicaciones, sistemas de cómputo, inteligencia militar, vigilancia y reconocimiento.

La finalidad de este framework es que toda la información a nivel de comando militar y servicios de defensa, puedan interrelacionarse entre sí y con otras organizaciones a través de un conjunto de vistas comunes. C4ISR como arquitectura orientada a las vistas está compuesta de 3 elementos: Operacional, sistemas y Tecnología.

4.1.3.4. Modelo de Referencia para Procesamiento Distribuido Abierto (RM-ODP)

Está definido por 5 puntos de vista o perspectivas (viewpoints). ODP define un marco específico para arquitecturas de sistemas distribuidos. El diseño de la arquitectura provee soporte para diseños a gran escala que requieren de la integración de aspectos como distribución, interoperabilidad o portabilidad de los sistemas, objetos y componentes.

Según la ISO (International Standards Organization) e ITU (International Telecommunication Union , conocido antes como CCITT) en una de tantas definiciones resaltan a ODP como un referente arquitectónico que proporciona un Framework conceptual y una arquitectura que integra aspectos relacionados con la distribución, interoperabilidad y portabilidad de sistemas software, de tal forma que dicha heterogeneidad del entorno (hardware, sistemas operativos, redes, lenguajes de programación, bases de datos, etc.), sean transparentes al usuario.

Las vistas que integran el modelo ODP son: Empresa, Información, Computación (servicios de cómputo), Ingeniería y Tecnología.

4.1.3.5. The Open Group Architecture Framework (TOGAF)

TOGAF es una arquitectura empresarial que proporciona un enfoque para el diseño, planificación, implementación y gobierno TI. Se basa en cuatro dimensiones:

- **Arquitectura de Negocios** (o de Procesos de Negocio), la cual define la estrategia de negocios, la gobernabilidad, la estructura y los procesos clave de la organización;

- **Arquitectura de Aplicaciones**, representadas por las interacciones entre estos sistemas y sus relaciones con los procesos de negocio de la organización.

- **Arquitectura de Datos**, que describe la estructura de los datos físicos y lógicos de la organización, y se encarga de la gestión de los mismos.

- **Arquitectura Tecnológica**, hace referencia a la estructura de hardware, software y redes requerida para dar soporte a la implantación de las aplicaciones críticas del negocio.

4.1.3.6. Modelo 4+1 de Rational Software Corp.

Es un Framework diseñado por Philippe Kruchten para describir la arquitectura de un sistema software intensivo basado múltiples vistas. 4+1 ha sido vinculado a la metodología de desarrollo Rational Unified Process (RUP), a través de 4 vistas:

- La **Lógica**, que comprende las abstracciones del sistema a partir del dominio de un problema;

- La de **Proceso**, que corresponde a un conjunto de elementos en ejecución a partir de las abstracciones;

- La **Física** que relaciona al software con el hardware y

- La de **Desarrollo** que hace referencia a la producción de módulos o componentes en un entorno de desarrollo.

- El nombre 4+1 se deriva de: ***vista lógica, vista de procesos, vista de despliegue y vista física y la vista "+1"***. El +1 hace referencia a la vista de **Escenarios** representado por los Casos de Uso.

4.1.3.7. Unified Modeling Language (UML)

Framework diseñado por los precursores de UML (Booch, Rambaugh, & Jacobson, 1999). Está compuesto de 5 vistas:

- La vista de **Diseño**, que se ocupa de las clases, las interfaces y esquemas de colaboración entre entidades o actores;

- La vista de **Procesos** (hilos, concurrencia, sincronización de procesos, colas),

- La vista de **Implementación** que fusiona los componentes y sistema de archivos con los medios físicos,

- La vista de **Despliegue** que hace referencia a la disposición de nodos y topologías de hardware mediante la cual se distribuirá la arquitectura y

- La vista de **Casos de Uso** que corresponde a una descripción de los pasos o las actividades que deberán realizarse para llevar a cabo algún proceso¨.

4.1.3.8. Pattern Oriented Software Architecure (POSA)

Los patrones de arquitectura expresan el esquema fundamental de organización para sistemas de software. Proveen un conjunto de subsistemas predefinidos; especifican sus responsabilidades e incluyen reglas y guías para organizar las relaciones entre ellos (Microsoft, 2013). POSA como arquitectura orientada a las vistas está compuesta de 4 elementos:

- La vista **Lógica** de la cual hace parte los objetos de diseño y el diagrama relacional;

- La vista de **Proceso** que se encarga de manejar los mecanismos de concurrencia y sincronización de procesos;

- La vista **Física** que relaciona al software con el hardware y otros soportes para artefactos y modelos en entornos distribuidos; y

- La vista de **Desarrollo** que hace referencia a la producción de módulos o componentes en un entorno de desarrollo.

Este modelo coincide con el de Kruchten, pero no hace énfasis en el 5 elemento denominado Escenarios o +1.

4.1.3.9. Modelo de Vistas de Microsoft

La propuesta de Microsoft[7] se centra en el uso de Perspectivas Arquitectónicas. Estas perspectivas incluyen el modelo de negocio (funcionamiento interno del negocio), el modelo de aplicación (aplicaciones empresariales, funcionalidades e interrelaciones), la información (de acuerdo a la necesidades del negocio y los interesados) y el soporte TI (uso de hardware y software que da soporte a las funciones y visión de la organización). Microsoft trabaja en base a 4 vistas: Conceptual, Lógica, Física e Implementación.

- La vista **Conceptual** define los requerimientos funcionales y la visión que los usuarios del negocio tienen de la aplicación y describir el modelo de negocio que la arquitectura debe cubrir. Esta vista muestra los subsistemas y módulos en los que se divide la aplicación y la funcionalidad que brinda dentro de cada uno de ellos. Esta vista está representada por Casos de Uso, Diagramas de Actividades, Procesos de Negocio y Entidades.

- La vista **Lógica** configura los componentes principales de diseño y sus relaciones de forma independiente de los detalles técnicos y de cómo la funcionalidad será implementada en la plataforma de ejecución. Se basa en los diseños de casos de uso, diseño de paquetes y diseño de clases.

- La vista **Física** ilustra la distribución del procesamiento entre los distintos equipos que conforman la solución, incluyendo los servicios y procesos de base. Los elementos definidos en la vista lógica se "mapean" a componentes de software (servicios, procesos, etc.) o de hardware que definen más precisamente como se ejecutará la solución. Está basado en Diagramas de Despliegue.

[7] La definición de Vistas y el marco conceptual que describe la Arquitectura de Microsoft, ha sido extraída del sitio http://www.itescam.edu.mx/principal/sylabus/fpdb/recursos/r56619.PDF del ITESCAM "Instituto Tecnológico Superior de Calkiní en el Estado de Campeche, México D.F."

- La vista de **Implementación** Describe cómo se implementan los componentes físicos mostrados en vista de distribución agrupándolos en subsistemas organizados en capas y jerarquías, ilustra, además las dependencias entre éstos. Básicamente, se describe el mapeo desde los paquetes y clases del modelo de diseño a subsistemas y componentes físicos.

Hacen parte de este modelo de Arquitectura los Diagramas de Componentes.

Aparte de los diseños de arquitectura citados anteriormente, la IEEE también ha realizado sus propios aportes a la descripción de arquitecturas software. IEEE-Std-1471-2000 más que ser una norma estándar para la industria del software, es una Práctica recomendada para descripción de arquitecturas de Software y Sistemas intensivos (Recommended Practice for Architectural Description of Software-Intensive Systems). Esta norma establece una arquitectura común en base a 3 elementos básicos: La Arquitectura en sí, las Vistas y las Perspectivas (Puntos de Vista). (ISO/IEC:1471, 2007).

- **Arquitectura**. Es la organización fundamental de un sistema, encarnada en sus componentes, las relaciones entre ellos y con su entorno, y los principios que gobiernan su diseño y evolución.

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

- **Punto de vista**. Define un patrón o plantilla (template) para representar un conjunto de incumbencias (concerns) relativas a una arquitectura.

4.1.4. Abstracción

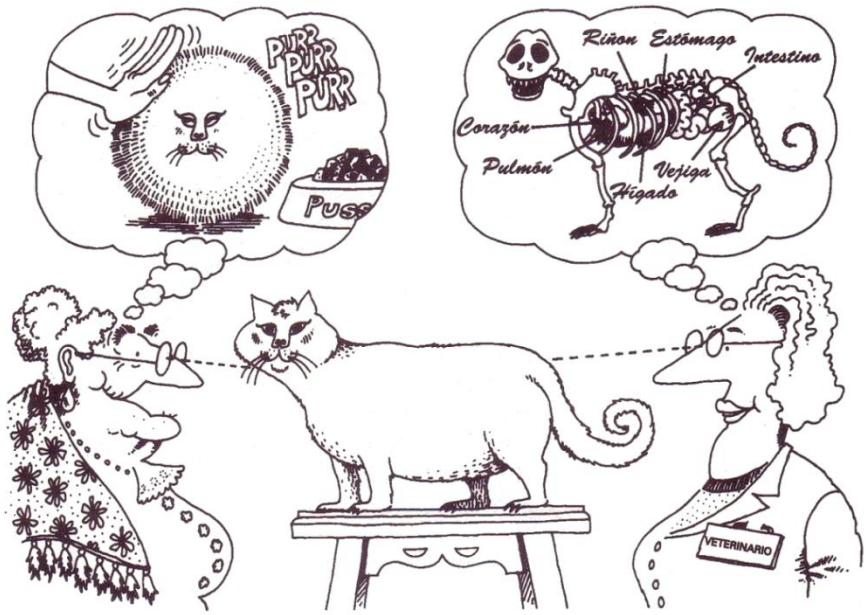
La RAE define el concepto Abstracción como: (Del lat. *abstractĭo, -ōnis*). ***1.*** *f. Acción y efecto de abstraer o abstraerse*. ***Abstraer*** consiste en separar por medio de una operación intelectual las cualidades de un objeto para considerarlas aisladamente o para considerar el mismo objeto en su pura esencia o noción.

Según (Graham, 1994) la Abstracción en una "*representación de las características esenciales de algo sin incluir antecedentes o detalles irrelevantes*".

Según (Booch, Rambaugh, & Jacobson, 1999), el principio de Abstracción "*…denota las características esenciales de un objeto que lo distinguen de otras clases de objeto y provee de este modo delimitaciones conceptuales bien definidas, relativas a la perspectiva del observador…*"

De acuerdo con (Blaha & Rumbaugh, 2004), la Abstracción corresponde a la "*Facilidad mental que permite a los humanos ver los problemas del mundo real con grados variables de detalle, dependiendo del contexto vigente del problema*".

Figura 4. Concepto de Abstracción



*La **Abstracción** se centra en las características esenciales de un objeto en relación a la perspectiva del observador.*

Fuente: https://sites.google.com/site/contenidospoo/abstracion.jpg

La definición aceptada mundialmente para el término Abstracción se encuentra publicada en la especificación de OMG para UML (La última versión para la especificación y el estándar UML de OMG, es la v2.4.1. de Agosto de 2011), la cual hace parte del glosario de términos y normas para el Std-ISO/IEC 19505-1 y 19505-2. Según (OMG®, 2003) la Abstracción se define como

> "***The essential characteristics of an entity that distinguish it from all other kinds of entities. An abstraction defines a boundary relative to the perspective of the viewer.***"

> "*La Abstracción se refiere a un conjunto de características esenciales que distinguen a una entidad de todas las demás entidades. Una abstracción define una frontera relativa a la perspectiva del observador*".

En pocas palabras la Abstracción trata de captar las características esenciales de un objeto, así como su comportamiento, y representarlas así mediante un modelo.

4.1.5. Procesos/Metodologías

Los procesos o metodologías en el marco de la arquitectura e ingeniería del software se refiere al uso de marcos o técnicas que se utilizan para estructurar, planear, y controlar el proceso de desarrollo de un sistema de información (CMS, 2008). A partir de esta concepción se han diseñado estrategias de desarrollo que difieren o coinciden entre sí a través de postulados, tipo de desarrollo, capacidades, ventajas, bondades y debilidades conocidas ampliamente y que se aplican a la mayoría de procesos de desarrollo de software. A partir de estas estructuras, existen varios modelos que permiten establecer pasos o estilos para el proceso de desarrollo de software, definiendo desde enfoques propios y a partir de una serie de actividades, la forma y el lugar donde se desarrolla cada proceso y la forma en cómo este interactúa con otros componentes de desarrollo y el contexto.

Todos los procesos de desarrollo se vinculan al ciclo de vida permite que los errores de programación sean detectados antes de que la aplicación esté implementada, y por lo tanto, permite a los desarrolladores concentrarse en la calidad del software, en los plazos de implementación y en los costos asociados.

El ciclo de vida básico de un software es un proceso cíclico y dispone de las siguientes fases:

- **Definición de Necesidades o Requerimientos**: En este componente se definen los objetivos del proyecto, los resultados esperados y las necesidades del negocio.

- **Análisis y Viabilidad**: Se considera la primera etapa en la producción de un componente de software. En esta etapa se define en realidad ***QUÉ*** va a hacer el software. Esta fase por lo general requiere de una constante comunicación entre los programadores y usuarios finales para definir con exactitud los objetivos y las características funcionales y no funcionales de un artefacto de software. Con el análisis se intenta descubrir dificultades, restricciones, limitaciones u otros factores que afecten el diseño y el tiempo de desarrollo y también identifica potenciales errores con anticipación incluso antes de estar terminado el producto. Esta fase recopila, examina y formula los requisitos del cliente y se modelan las restricciones que tendrá el software.

- **Diseño**: Corresponde a los requisitos generales de tipo arquitectónico (modelo de software). En la fase anterior se definió el ***Qué*** hará la herramienta de software, ahora el diseño determina el ***CÓMO*** deberá ser diseñada, (prototipos, modelos de bases de datos, lenguajes de programación, sistemas de gestión de bases de datos, ejecución simple o paralela, esquemas funcionales, formularios, etc.). El modelado de datos cobra aquí una importancia significativa, ya que de ella dependerá o no un diseño óptimo de las bases de datos que son el soporte de toda aplicación o sistema de información. También hace parte de esta fase el **Diseño en detalle**, que corresponde a la definición precisa de cada subconjunto de instrucciones o procedimientos que efectuará la aplicación.

- **Codificación** (Codificación e implementación): corresponde a la implementación del problema y sus requerimientos a través de lenguajes de programación específicos (C++, Java, .NET, php, Perl, Ruby, Phyton, etc.). En esta fase se escriben o codifican los algoritmos y estructuras de datos, definidos por la etapa de diseño. En muchos proyectos se pasa directamente a esta etapa sin agotar las instancias anteriores. Este tipo de proyectos adoptan un modelo de ciclo de vida de codificar y corregir en la marcha; lo que conlleva a la pérdida de control sobre la gestión del proyecto. Eliminar las etapas de especificaciones, análisis y diseño no contribuye a una buena práctica de desarrollo de software, al contrario, se corre el riesgo que el proyecto fallé o esté sujeto a cambios radicales, que incluya un modelado total del problema.

- **Pruebas**: En esta etapa se prueban de forma individual los componentes de la aplicación y se verifica su funcionamiento con bases en las especificaciones iniciales. Las pruebas de unidad tienen como fin probar los subprogramas, las subrutinas, los procedimientos individuales o las clases en un programa. Esta etapa es esencial ya que permite la revisión de unidades o bloques más pequeños de código evitando revisiones exhaustivas a códigos o programas completos.

- **Prueba de Integración**: Este tipo de pruebas se realiza para garantizar que los diferentes módulos que componen un artefacto de software, se integren con la aplicación de forma transparente. El propósito de la *prueba de integración* consiste en realizar pruebas para verificar que un gran conjunto de partes de software funcionan juntos. Las pruebas de integración (algunas veces llamadas integración y testing) sirven para combinar los módulos individuales para posteriormente ser testeados como una unidad. Este tipo de prueba se pueden realizar a la par con las unitarias sin que esto afecte el desarrollo del producto y la funcionalidad como tal.

- **Documentación**: Sirve para documentar información necesaria para los usuarios del software y para desarrollos futuros. la documentación es un conjunto de información que expresa detalles de estos sistemas, la forma adecuada de operar con él, nos permite interpretar los errores, conocer su proceso, etc. En la mayoría de los sistemas, la documentación está siempre expresada en un carácter técnico por excelencia, indicando algoritmos de cómo realizar alguna acción, registrar transacciones, imprimir reportes, resolver errores, u otros dependiendo de la naturaleza del sistema o su función. La documentación tiene una gran importancia dentro de la empresa, ya que esta ayuda a eliminar la posible dependencia que se pueda generar entre el proyecto realizado, y el ejecutor de éste.

- **Implementación**: El objetivo principal de la etapa de implementación es desarrollar la arquitectura y el sistema como un todo.

- **Depuración o Debugging:** El objetivo de esta etapa es garantizar que la aplicación de software no contiene errores de diseño o codificación. Todos los

programas contienen errores o son susceptibles a fallos. Lo ideal en esta fase es encontrar la mayoría, si no todos los errores posibles. Identificados los errores, se debe proceder con la respectiva documentación para implementar la solución.

- **Validación:** esta etapa tiene como objetivo la verificación del sistema desarrollado. Con ella se evalúa el impacto y se verifica si la herramienta de software cumple o no con los requerimientos expresados inicialmente por el cliente y que han dado lugar al desarrollo del proyecto.

- **Mantenimiento**: Se ocupa de procesos correctivos (mantenimiento correctivo) y de las actualizaciones secundarias del software (mantenimiento continuo). Esta fase es considerada como dos componentes independientes; por una parte se encuentra el ***Mantenimiento*** que se ocupa de la corrección y detección de errores que surgen al aplicar la solución de software; y por otro lado, se encuentra la ***evolución***, característica que agrega al diseño nuevas funcionalidades.

Figura 5. Ciclo Básico del Software

En términos técnicos, el Ciclo de Vida del Software se define como "*Una aproximación lógica a la adquisición, el suministro, el desarrollo, la explotación y el mantenimiento del software*"(ISO15504, 1998). En otros términos la ISO lo define como "*Un marco de referencia que contiene los procesos, las actividades y las tareas involucradas en el desarrollo, la explotación y el mantenimiento de un producto de*

software, abarcando la vida del sistema desde la definición de los requisitos hasta la finalización de su uso" (ISO12207, 2002).

Clasificar o comparar el conjunto de metodologías existentes es una tarea compleja debido a la diversidad de puntos de vista, alcances y propuestas dispuestas en cada una de ellas. En torno a su clasificación se toman como base aspectos como producto o proceso, tipo de filosofía y objetivo de desarrollo.

En relación al tipo de producto o artefacto derivado en cada una de las fases se consideran 2 tipos de metodologías; ***Estructuradas y Orientadas a Objetos***.

- **ESTRUCTURADAS**: se centra en el uso de funciones y estructuras de datos. Emplea una técnica conocida como Top-Down para realizar el proceso de análisis y desarrollo. Esta metodología se centra específicamente en los Datos. Sus inicios datan de 1968 con los primeros algoritmos descritos por Dijsktra. Utiliza ampliamente técnicas de programación estructurada como "*Jackson Structured Programming* (*JSP*)" y "*JSD* (*Jackson Structured Design*)" para diseño y desarrollo. De este tipo de metodología se destacan Yourdon y Constatine (1975), Gane y Sarson (1977), De Marco (1978), MERISE (1972), SSADM (1981-1986-1990), Métrica (1989-1993-1995) y la Metodología para aspectos de Ingeniería de la Información (Information Engineering - IE) de J. Martin y C. Finkelstein (1986).

- **ORIENTADA A OBJETOS**: Este tipo de metodologías surge con la evolución de los lenguajes de programación. En este tipo de metodologías se soporta ampliamente los lenguajes Orientados a Objetos. Sus inicios datan de 1960 con el primer lenguaje Orientado a Objetos al cual denominaron SIMULA. A finales de los 70's e inicios de los 80's aparece el compilador C++ considerado como el máximo exponente de los lenguajes Orientados a Objetos. A partir de 1990 se consolidan los lenguajes para POO con la aparición de JAVA y el mismo C Sharp "C#" de Microsoft.

  Los métodos Orientados a Objetos toman como base a UML (Booch, Rambaugh, & Jacobson, 1999) a partir de un conjunto de notaciones y especificaciones para el diseño basado en el uso de patrones. Algunas metodologías Orientadas a Objetos que utilizan la notación UML son: Rational Unified Process (RUP), OPEN, MÉTRICA (que también soporta la notación estructurada), Shlaer y Mellor, Coad y Yourdon, HOOD (ESA). El proceso Unificado surge de las siguientes propuestas; OMT (Rumbaugh), Booch, CRC (Wirfs-Brock) y OOSE (Jacobson). Mundialmente se conoce como Proceso Unificado (OMT, Booch, OOSE).

En relación a la filosofía de desarrollo, se considera 1 grupo específico de metodologías. A este grupo se les denomina ***Metodologías Tradicionales*** y se

caracterizan por una fuerte estructuración en los procesos de planificación y el estricto control de proyectos, requerimientos, procesos de modelado, análisis de entradas y salidas, etc. Por lo general se convierten en metodologías bastante rígidas donde existen muchos controles al proceso de desarrollo, lo que puede convertir a este tipo de método en algo costoso y complejo al momento del desarrollo. Las metodologías Estructuradas y Orientadas a Objetos se ajustan a este tipo de filosofía a excepción de RUP que puede ser ajustada al punto de convertirla en una metodología Ágil.

Si se puede clasificar como objetivo de desarrollo, al hecho de que cualquier implementación puede hacerse en tiempos considerables con un alto nivel de desarrollo y rendimiento, estamos hablando de metodologías recientes con un fuerte componente en la agilidad. A estas metodologías se les denomina ***Metodologías Ágiles***. La característica principal de este tipo de desarrollos se centra en la entrega de pequeños proyectos en ciclos cortos (denominados comúnmente iteraciones), se enfoca en desarrollo por equipos de trabajo (involucrando a todos los stakeholders en el proceso) y la característica más importante es que todo lo que se construya bajo el esquema de la agilidad, debe ser altamente reusable y adaptable. Se destacan de este grupo de metodologías:

- eXtreme Programming o XP
- SCRUM
- Familia de metodologías CRYSTAL
- Feature Driven Development - FDD
- Proceso Unificado Rational - RUP
- Dynamic Systems Development Method - DSDM
- Adaptive Software Development - ASD
- Open Source Software Development - OSSD

En cuanto a los ***Métodos Ágiles*** se puede concluir que éstos permiten la construcción de artefactos o desarrollos más eficientes en tiempos relativamente cortos a través de una serie de iteraciones, que de forma progresiva van agregando funcionalidades conforme a las necesidades y políticas del negocio. Las iteraciones representan en sí, un nivel donde se desarrollan las etapas esenciales de un proyecto de software (requerimientos, diseño, implementación y pruebas) y en las cuales se implementa un fuerte componentes de retroalimentación para la mejora del producto.

Continuando con la conceptualización de procesos/ metodologías en torno a la Arquitectura del Software, a mediados de los 80's, el Departamento de Defensa de los Estados Unidos de Norte América propuso la estandarización y creación de un modelo que permitiera evaluar los problemas asociados a la disciplina del software y a su posterior mejoramiento. Es así como en 1984 surgió el SEI[8] en la Universidad Carnegie Mellon, que correspondió a un grupo multidisciplinar de investigadores e ingenieros, que auspiciados por el Departamento de Defensa, diseñaron una propuesta para mejorar los aspectos de calidad en el software. El SEI desarrolló este modelo que definió una serie de etapas claves para el desarrollo de un producto y a su vez estableció un mecanismo para medir el

[8] **SEI**, por sus siglas en Inglés "***Software Engineering Institute***"

estado de madurez en un proceso de desarrollo de software; este modelo se denominó CMM "***C**apability **M**aturity **M**odel*". El modelo CMM surge como estándar a partir de 1991, y su aplicación involucra la aplicación del Ciclo PHVA propuesto por Shewart[9] y Deming[10].

El Modelo de Madurez y Capacidad del Proceso de Software (CMM) contribuye a que las organizaciones produzcan bienes y servicios consistentes y productos con un alto nivel de calidad. CMM realiza mediciones exhaustivas sobre los procesos y sus relaciones conforme a las metas y resultados planificados. En CMM, el objetivo primordial de un proceso de software con un alto nivel de madurez consiste en la efectividad de crear o planificar herramientas de calidad que cumplan no solo con los estándares industriales sino que cumplan específicamente con los requerimientos del cliente.

La ***madurez*** de un proceso de software se debe entender como el nivel de efectividad alcanzado por el proceso o el producto a largo plazo. Con CMM se evalúa la madurez de los procesos de desarrollo de software dentro de una organización y se proponen planes de mejoramiento en cada uno de los procesos de desarrollo de software en base a una serie de niveles que determinan la cualidad y efectividad del proceso, yendo desde un estado de inmadurez total a un proceso disciplinado, coordinado y de mejoramiento continuo con un alto grado de madurez.

La madurez de un proceso en CMM, permite determinar en qué grado se encuentra un proceso de software, cuáles son sus definiciones, dimensiones y como éste debe ser definido, administrado, medido, controlado puesto a punto para su implementación. La madurez se puede interpretar también como un indicador donde se puede medir y cuantificar el impacto y los resultados del proceso de desarrollo partiendo de unos objetivos y los resultados obtenidos o esperados. Un negocio u organización logra una madurez efectiva cuando todos los procesos de desarrollo de software se convierten en políticas institucionales y se establecen a lo largo y ancho de todo el nivel jerárquico, apoyando la función gerencial y organizativa de la empresa.

A finales de los 80's, surgió una nueva propuesta denominada CMMI "***C**apability **M**aturity **M**odel **I**ntegration*", una evolución del Modelo CMM. El propósito de CMMI consiste en evaluar la calidad de los procesos de software, dotando a la organización de rutas para el mejoramiento de los procesos tanto internos como externos de la organización gestionando de manera eficiente los recursos e incrementando la habilidad de desarrollo, adquisición y mantenimiento de cualquier producto bien o servicio. Actualmente CMMI se encuentra en desarrollo y la última versión corresponde a la 1.3, liberada el 1 de Noviembre de 2010.

CCMI se fundamenta al igual que CMM en aspectos claves que permiten catalogar a una organización como maduras o inmadura dependiendo del grado de madurez alcanzado a nivel de procesos y desarrollo y por otro lado otorga las herramientas

[9] **Walter A. Shewhart**, físico, ingeniero y estadístico estadounidense, a veces conocido como el *padre del control estadístico de la calidad*.

[10] **William E. Deming,** Estadístico estadounidense, consultor y difusor del concepto de calidad total.

necesarias para que las organizaciones apunten sus esfuerzos y planes de acción que le permitan evolucionar y que pasen de un estadio inmaduro a una empresa sólida y con un alto grado de madurez.

Como conclusión en lo referente a procesos o metodologías de desarrollo se puede afirmar que sin importar el tipo de metodología implementada en los proyectos de diseño y desarrollo, más de ser una especificación o un estándar, las metodologías se convierten en herramientas esenciales para el Arquitecto de Software, ya que a través de sus diversos marcos de trabajo sirven para estructurar, planificar y controlar el proceso de desarrollo en cualquier sistema de información.

4.1.6. Requerimientos

Un **Requerimiento** es una característica que el sistema debe tener o es una restricción que el sistema debe satisfacer para ser aceptada por el cliente. (Booch, Rambaugh, & Jacobson, 1999). Un requerimiento a nivel de software es una propiedad o restricción determinada con precisión, la cual permite satisfacer un conjunto de reglas y especificaciones para satisfacer una necesidad. (Sommerville, 2005). Existen 2 tipos de Requerimientos a nivel de Arquitectura e Ingeniería de Software; ellos son: los ***Funcionales*** y los ***No Funcionales***.

- **Funcionales**: Especifica las actividades de negocio del cliente y la funcionalidad para la cual será diseñado el artefacto o aplicación. Estas actividades definen el comportamiento específico de cada módulo o conjunto de aplicaciones en el modelo de negocio. Un Requerimiento funcional determina qué tipo de servicio proporcionará el sistema al usuario, cómo reaccionará éste frente a las entradas proveídas por el o los usuarios y cuáles son los posibles comportamientos en los que incurrirá el sistema ante una determinada variable o cambio introducida al sistema. Según (Sommerville, 2005), Los requerimientos Funcionales describen la interacción entre el sistema y su ambiente independientemente de su implementación (El ambiente incluye al usuario y cualquier otro sistema externo que interactúa con el sistema).

1) **No Funcionales**: Los requerimientos No Funcionales según (Sommerville, 2005) describen aspectos del sistema que son visibles por el usuario y que no incluyen una relación directa con el comportamiento funcional del sistema. Los requerimientos no funcionales incluyen restricciones como el tiempo de respuesta (desempeño), la precisión, recursos consumidos, seguridad, etc.

   Este tipo de requerimientos por lo general suelen ser más críticos que los mismos Funcionales, ya que a través de ellos se describen aspectos tan importantes como las reglas del negocio, los atributos de calidad, las interfaces externas y el conjunto de restricciones del sistema.

Los requerimientos pueden ser obtenidos a partir de 2 elementos: Los Casos de Uso (Abstracción de un escenario determinado) y los Escenarios (conjunto de interacciones entre el usuario y el sistema). A continuación se describen ambos elementos y su posible documentación en lenguaje natural. No es común realizar la descripción de un Escenario o un Caso de Uso en este estilo, ya que ambos sugieren la aplicabilidad de UML como estándar para la descripción de los flujos e interacciones.

La verdadera intensión de la siguiente descripción es dar al lector una idea muy general de lo que se hace en cada proceso y lo que implica la definición de un requerimiento.

Tabla 4. Documentación de un Escenario

Nombre del escenario	Nombre dado al Escenario. Debe siempre responder a un tipo de acción, por esta razón debe iniciarse con verbos en infinitivo (ar, er, ir), ejemplo Consultar, Registrar, Distribuir, Procesar, etc. Ejemplo: Consultar listado de cursos
Instancias de los stakeholders (participantes)	Describe el Actor o Actores que intervienen en el Escenario. Ejemplo Sergio: Profesor
Flujo de eventos	Se considera como el flujo normal de procesos e información dentro del Escenario. "Factor de Éxito". Es una descripción detallada que permite tanto al sistema como a los stakeholders, seguir en detalle un conjunto de pasos para lograr el éxito y lograr la meta propuesta por el Escenario (sin errores). Ejemplo: 1) Sergio ingresa al sistema a través del sistema de autenticación 2) Una vez autenticado, el sistema muestra el menú al profesor indicando las opciones que éste puede tomar. 3) Sergio elige una de las opciones del menú, en este caso se dirige al sistema de reportes de la plataforma, para sacar un listado general del curso de Arquitectura de Software. 4) El sistema solicita la información del curso como código, sección y semestre de la materia, ciclo, etc. 5) Sergio ingresa los valores al formulario y llena todos los campos solicitados por la aplicación. 6) El sistema devuelve el listado a Sergio con la información requerida que contiene nombre del estudiante, id, carrera, código de la materia, correo electrónico de todos los estudiantes del curso de Arquitectura de Software.

Tabla 5. Documentación de un Caso de Uso

Identificador:	Id Asociado que se le otorga al Caso de Uso para identificarlo como único.
Fecha	Fecha de Elaboración.
Autor:	Persona responsable del diseño o modelado del Caso de Uso.
Prioridad	Mide el nivel de cumplimiento, ejecución o preponderancia de la tarea. Puede ser valorado como Alta/Media/Baja u otro tipo de valoración dada por el arquitecto o usuario.
Nombre Caso de Uso:	Nombre dado al Caso de Uso. Debe siempre responder a un tipo de

	acción, por esta razón debe iniciarse con verbos en infinitivo (ar, er, ir), ejemplo Consultar, Registrar, Distribuir, Procesar, etc.
Actores Involucrados	Describe el Actor o Actores que intervienen en el proceso o Caso de Uso.
Resumen:	Describe la interacción o acciones que ocurre en el caso de uso (contexto).
Curso Básico Eventos:	Se considera como el flujo normal de procesos e información dentro del Caso de Uso. "Factor de Éxito". Es una descripción detallada que permite tanto al sistema como a los actores involucrados, seguir en detalle un conjunto de pasos para lograr el éxito y lograr la meta propuesta en el Caso de Uso (sin errores).
Caminos Alternativos:	Es una derivación o alternativa que puede tomar en un momento dado el caso de uso si no se cumple la función básica o el curso normal de eventos, pero que al final proceden a la meta específica del Caso de Uso.
Caminos de Excepción:	Muestra el comportamiento y opciones que debe tomar el sistema cuando éste reporta algún error en el proceso.
Puntos de Extensión:	Indica que cierta funcionalidad en un Caso de Uso puede hacerse extensible "Extends" bajo ciertas condiciones. Un atributo Extendido es una especie de sub-clase con relación Hijo-Padre, donde Padre es la Clase superior.
Pre - Condiciones:	Son eventos o condiciones que deben ocurrir antes de iniciar el escenario.
Post- Condiciones:	Es la respuesta exitosa, lo que se espera que realice el Caso de Uso. Se considera como satisfecha cuando el Caso de Uso completa satisfactoriamente la tarea.
Criterios de Aceptación	Es una especie de validación que se hace al usuario para que acepte el procedimiento como válido.
Borrador de Interfaz Gráfica	Puede incluir una pequeña capa o interfaz de usuario GUI para mostrar como visualizará el usuario el mecanismo de interacción con el sistema.

4.1.7. Escenarios

Como se describió anteriormente en la Sección 4.1.3. "*Frameworks y Vistas*", existen diversas metodologías y arquitecturas que se han orientado sobre la base de los **Escenarios**. Una de tantas propuestas y que hoy ocupa el análisis en el presente informe corresponde a la Arquitectura experimental **ArchE** "Architecture Expert"; un Sistema Experto basado en Escenarios y atributos de calidad, responsabilidades, Marcos de Razonamiento y Tácticas Arquitecturales para el diseño de herramientas software desarrollado por el SEI de la universidad Carnegie Mellon.

ArchE al igual que otras arquitecturas basadas en escenarios, realizan sus procesos de análisis a partir de una serie de comportamientos "Responsabilidades" y la interrelación entre componentes. A partir de un "Análisis temprano", estas herramientas (incluida ArchE), buscan o permiten detectar posibles errores en la arquitectura mucho antes de que esta se empiece a desarrollar.

Un **Escenario** (*Scenario*) corresponde a una descripción de pasos o secuencias para llevar a cabo el desempeño normal (éxito) de un Caso de Uso. Puede incluir múltiples escenarios para abarcar extensiones, excepciones y caminos alternativos. Un escenario

permite evaluar el comportamiento de un modelo a partir de una serie de requisitos o estímulos proveídos por el usuario y procesa una salida en función de los atributos de calidad proporcionados por el arquitecto. El resultado final será la medida del escenario en función de dichos atributos, evaluando si cumple o no con el requerimiento.

De acuerdo con (Clements & Nortrhop, "Software architecture: An executive overview, 1996), los escenarios "*son algo así como libretos correspondientes a las distintas piezas de funcionalidad de un sistema y son útiles para analizar una vista determinada y sus relaciones*". Un escenario se puede considerar como una abstracción específica en función de una serie de requerimientos básicos (requisitos de calidad) o requerimientos principales del sistema. Los escenarios se dividen en dos categorías: casos de uso (secuencias de responsabilidades) y casos de cambio (modificaciones propuestas al sistema).

En las siguiente sección se presentan las definiciones adoptadas por el SEI y sus autores (Bass, Len; Clements, Paul; Kazman, Rick, 2003), en torno a los atributos de calidad, y que posteriormente serán implementados a través de la herramienta ArchE.

4.1.8. Concerns Arquitectónicos o "Asuntos de Interés"

El término **Concerns** se usa de forma habitual en el ámbito de la ingeniería de software para describir los aspectos críticos o importantes para el desarrollo de una herramienta software y la implicación de los actores involucrados en este proceso. Esta definición está dada por el estándar 1471 de la IEEE.

Los Concerns no solo están presentes a nivel arquitectónico, sino que estos van surgiendo conforme se avanza en cada ciclo de desarrollo del sistema (Filman, Tzilla, Mehmet, & Siobhan, 2004) y se van documentado de acuerdo al producto tras cada ciclo, así por ejemplo un Concern relativo a seguridad puede surgir en el ciclo de los requisito s y se documenta mediante una especificación del mismo, a nivel arquitectónico la seguridad se representa en otros Concerns como la autentificación y encriptación, y al nivel de implementación surgen otros más en la misma línea, como por ejemplo los mecanismos utilizados para la autentificación y encriptación. (Limón C., 2009).

A los Concerns que se presentan al inicio del ciclo de desarrollo se les conoce como tempranos y abarcan a los requisitos y a la arquitectura, a los que se presentan en el ciclo del análisis y diseño se les llama intermedios y a los que se presentan a nivel de implementación se les designa como Concerns tardíos(Limón C., 2009).

Los Concerns hacen parte de los requerimientos no-funcionales de una aplicación, es decir provienen de las características que deben cumplir el diseño, la implementación y el desarrollo.

5

ATRIBUTOS DE CALIDAD

"Es más fácil cambiar las especificaciones para que encajen con el software que hacerlo al revés" - Alan Perlis

Este capítulo se destinará exclusivamente a la definición de los Atributos de Calidad propuestos por (Bass, Len; Clements, Paul; Kazman, Rick, 2003) en su libro "*Software Architecture in Practice*" a través del uso de **Escenarios**, aplicables a una arquitectura software. Los autores del SEI reconocen seis (6) atributos que permiten la evaluación de cualquier arquitectura y describe así mismo el conjunto de mecanismos para adaptar, modificar, corregir o validar un conjunto de estímulos provistos en los distintos escenarios de la arquitectura.

Existen diversos mecanismos para la medición de atributos de calidad en software, pero este estudio se limitará al análisis de los 6 atributos propuestos. Ellos son: Disponibilidad (Availability), Modificabilidad (Modifiability), Desempeño (Performance), Seguridad (Security), Facilidad de Pruebas (Testability) y Usabilidad (Usability). La aplicación de estos atributos permite en alguna medida la corrección de errores en la arquitectura antes de iniciar el proceso de desarrollo.

Antes de entrar en detalle recordemos el concepto de calidad en el software según la IEEE-Std.1061.

> "*Software quality is the degree in which software possesses a desired combination of quality attributes. The purpose of software metrics is to make assessments throughout the software life cycle as to whether the software quality requirements are being met…. The use of software metrics reduces*

subjectivity in the assesment and control of software quality by providing a quantitative basis for making decisions about software quality….However, the use of metrics does not eliminate the need for human judgment in software assessment".

"*La calidad del software corresponde a un grado o cualidad en la cual el software posee una combinación de atributos de calidad deseados. El propósito de la medición del software es hacer que las mediciones o valoraciones respondan a los requisitos de calidad establecidos a lo largo del ciclo de vida del software… El uso de métricas reduce la subjetividad en los juicios de evaluación y el control de calidad del software a través de un conjunto de elementos cuantitativos para la tomar de decisiones en torno al desarrollo y la calidad del software… Sin embargo, el uso de las métricas no elimina del todo la necesidad del juicio humano en cuanto a la valoración del software*".

5.1. ISO/IEC 9126 Software Engineering – Product Quality (2001)

Es un estándar internacional aplicado a la evolución de la Calidad en el Software. Está dividido en cuatro partes: modelo de calidad, métricas externas, métricas internas y calidad en las métricas de uso. El estándar en sí mismo es una herramienta para validar la definición y completitud de una serie de requisitos y a partir de un conjunto de atributos, asegurar la calidad del producto software.

Los inicios del estándar ISO/IEC 9126 "*Software Product Evaluation: Quality Characteristics and Guidline for Their Use*" se remonta a 1977 con base en lo propuesto por McCall y Bohem en su modelo. El modelo de McCall organiza los factores en tres ejes o puntos de vista desde los cuales se puede evaluar la calidad de un producto software. Este modelo propuso 11 factores distribuidos en los 3 ejes y a su vez cada factor contiene otros criterios que describen algunas actividades para realizar con mayor facilidad la aplicación de métricas. La tabla 6 muestra la distribución de los atributos descritos por McCall y Boehm en su modelo de calidad.

Tabla 6. Modelo de McCall y la Calidad del Software

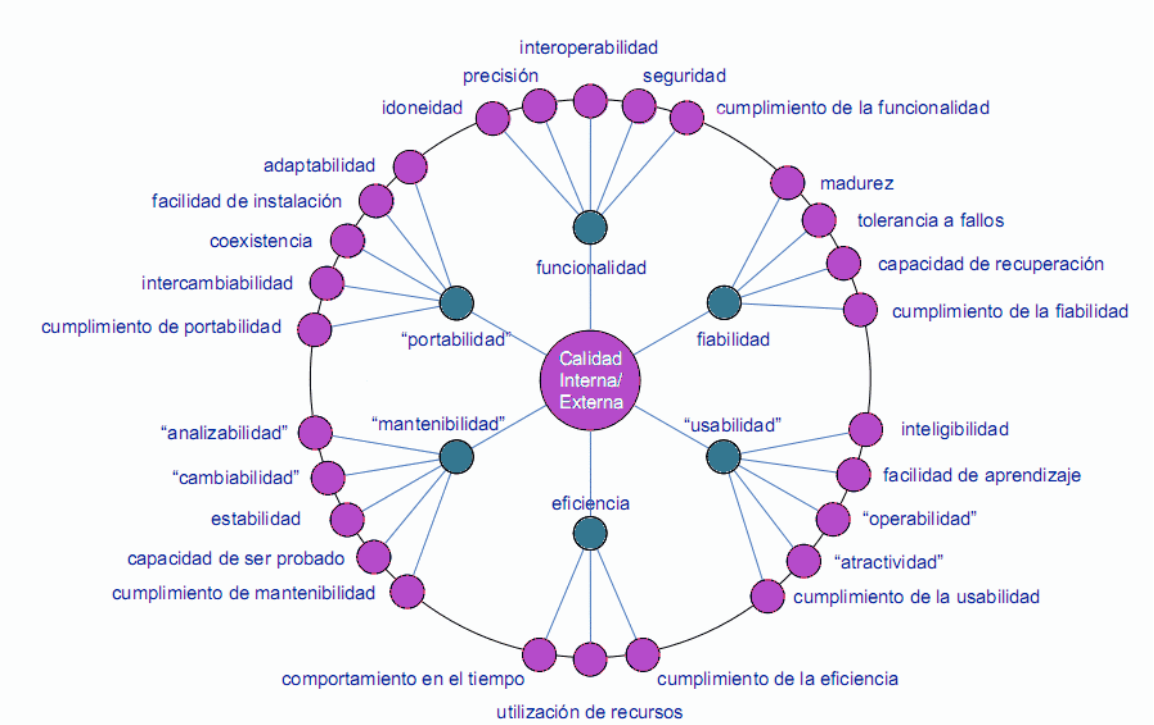
EJE	FACTOR	CRITERIOS DE EVALUACIÓN
OPERACIÓN DEL PRODUCTO	Facilidad de uso	• Facilidad de operación • Facilidad de comunicación. • Facilidad de aprendizaje. • Formación (Enseñabilidad).
	Integridad	• Control de accesos. • Facilidad de auditoría. • Seguridad.
	Corrección	• Completitud. • Consistencia. • Trazabilidad o rastreabilidad.

EJE	FACTOR	CRITERIOS DE EVALUACIÓN
OPERACIÓN DEL PRODUCTO	Fiabilidad	• Precisión. • Consistencia. • Tolerancia a fallos. • Modularidad. • Simplicidad. • Exactitud.
	Eficiencia	• Eficiencia en ejecución. • Eficiencia en almacenamiento.
REVISION DEL PRODUCTO	Facilidad de mantenimiento	• Modularidad. • Simplicidad. • Consistencia. • Concisión. • Auto descripción.
	Facilidad de prueba	• Modularidad. • Simplicidad. • Auto descripción. • Instrumentación.
	Flexibilidad	• Auto descripción. • Capacidad de expansión. • Generalidad. • Modularidad.
	Reusabilidad	• Auto descripción. • Generalidad. • Independencia entre sistemas y software. • Independencia entre sistemas y hardware.
	Interoperabilidad	• Interoperabilidad. • Compatibilidad de Comunicaciones. • Compatibilidad de Datos. • Estandarización de Datos.
	Portabilidad	• Auto descripción. • Modularidad. • Independencia entre sistemas y software. • Independencia del hardware.

La ISO/IEC 9126 es una variante del modelo de McCall. La norma hace una distinción entre fallos y no conformidades para validar y verificar el estado y calidad del producto. Los ***Fallos*** se catalogan como el NO cumplimiento de los requerimientos previos y las ***No Conformidades*** hacen referencia al incumplimiento de los requerimientos propiamente dichos (o especificados).

ISO 9126 clasifica los atributos de calidad de un producto software a través de un conjunto de características y sub-características y estos a su vez se catalogan en base a un conjunto de atributos. Los atributos son sujetos de medición, validación, verificación, estimación y evaluación.

Figura 6. Descripción del Estándar ISO/IEC-9126



Fuente: Consultoría de Sistemas SQA. http://www.sqa.es

Respecto a los modelos anteriores, ISO/IEC 9126 incluye en su descripción a la **Funcionalidad**, como una característica relevante para el proceso de calidad. De igual manera define con claridad cada una de las partes en las cual se divide el modelo. En cuanto a la "Calidad Interna" detalla aquellos atributos que pueden ser medidos durante el proceso de desarrollo. En cuanto a la "Calidad Externa" describe aquellos atributos que son medibles durante el proceso de pruebas y la "Calidad en Uso" corresponde a aquellos atributos que son percibidos por el usuario.

5.2. ESCENARIOS PARA ATRIBUTOS DE CALIDAD

Un escenario de acuerdo al SEI debe estar compuesto por seis partes. Ellas son: Fuente **de Estimulo, Estímulos, Ambiente, Artefacto, Respuesta y Medida de la Respuesta**.

Tabla 7. Representación de un Escenario

ELEMENTO		ATRIBUTO / VALOR
	**Fuente de Estimulo**	Está representado por las entidades (personas, un sistema de cómputo, o cualquier otro objeto o entidad que interactúa con el sistema). Su función principal es generar un estímulo (hacer que reaccione) al sistema.

ELEMENTO		ATRIBUTO / VALOR
	**Estímulos**	Corresponde a un conjunto de eventos o condiciones que tendrán que ser consideradas dentro del sistema y que responden a los estímulos introducidos por la fuente. Un estímulo es algo que altera el comportamiento del sistema y lo hace reaccionar ante dicho evento.
	**Ambiente**	Es el entorno o marco donde se desenvuelve el escenario. Cualquiera que sea el estímulo éste se desarrolla bajo determinadas condiciones. Cada ambiente está configurado de acuerdo a las especificaciones y a un estado determinado.
	**Artefacto**	Hace referencia a los componentes o conjunto de elementos que componen el sistema y que son afectados por los estímulos proporcionados por el usuario.
	**Respuesta**	Las respuestas son aquellas actividades resultantes que se generan tras la implementación de un determinado estímulo. Cada componente responde a un estímulo y este mismo produce una respuesta.
	**Medida de la Respuesta**	Toda respuesta está sujeta a su verificación. La efectividad del proceso radica en la capacidad de medición y su evaluación. En función de la Calidad se recopilan medidas, valores estimados, niveles de completitud en los requerimientos, efectividad de las tácticas aplicadas, etc.

Los escenarios también pueden ser diferenciados según su naturaleza; ***generalizados*** o ***específicos***. Los escenarios generales o generalizados son aquellos artefactos o elementos independientes del sistema que pueden ser reutilizados para evaluar o construir otros escenarios a partir de un escenario común y que pueden ser aplicados en otros sistemas. Los escenarios específicos son descripciones detalladas de los atributos de calidad dentro de un sistema en particular.

5.2.1. Disponibilidad (Availability)

Un escenario de Disponibilidad está relacionado directamente con los fallos ("failures") en el sistema y las consecuencias asociadas a dicho fallo. La disponibilidad se define como la propiedad o capacidad que tiene un sistema para operar en ciertas condiciones cuando le sea requerido. Un fallo en el sistema ocurre cuando un determinado servicio o aplicativo no es proporcionado en el tiempo establecido.

Se puede hacer un ejemplo comparativo de un atributo de disponibilidad a través del comando Ping, para indicar el tiempo de vida de un paquete o TTL. En una red de datos sobre IP "**Tiempo de Vida** o **Time To Live** (**TTL**) es un concepto usado en telecomunicaciones para indicar por cuántos nodos puede pasar un paquete antes de ser descartado por la red o devuelto a su origen".

El TTL es utilizado en el paquete IP de manera que los routers puedan analizarlo y actuar según su contenido. Si un router recibe un paquete con un TTL igual a uno o cero, no lo envía a través de sus puertos, sino que notifica vía ICMP a la dirección IP origen que el destino se encuentra "muy alejado" y procede a descartar dicho paquete. Si un paquete es recibido por un router que no es el destino, éste decrementa el valor del TTL en

uno y envía el paquete al siguiente router. En el protocolo IP, esta información se almacena en un campo de 8 bits. El valor óptimo para aprovechar el rendimiento en Internet es de 128. (Wikipedia, 2013).

En el ejemplo anterior, si existiera un fallo a nivel de disponibilidad de algún nodo, el sistema detectará tal anomalía e informará al usuario acerca de dicho fallo, para que éste tome las medidas pertinentes para corregir la anomalía en el sistema.

Para controlar un escenario de disponibilidad el arquitecto debe considerar ciertos comportamientos que puedan determinan en alguna medida el estado del servicio y las posibles variantes que puede tomar éste en caso de presentarse una determinada falla. Criterios como detección, recurrencia o persistencia del fallo, tiempo estimado de servicio, factores de desencadenamiento, afectación en otros componentes, tipo de anomalía, instrucciones y seguimientos, niveles de tolerancia, entre otros, se convierten en factores determinantes para controlar el atributo de calidad de disponibilidad.

Un ejemplo de escenario para el atributo de calidad de Disponibilidad se muestra en la siguiente tabla

Tabla 8. Escenario para el Atributo Disponibilidad

ELEMENTO	ATRIBUTO / VALOR
 **Fuente de Estímulo**	Sistema externo interactuando con la aplicación en desarrollo.
 **Estímulo**	El sistema externo realiza una petición a través de sockets para consultar una lista de datos asociados.
 **Ambiente**	El componente o sistema para la gestión de interfaces e interoperabilidad funciona en forma adecuada y sin contratiempos.
 **Artefacto**	El componente de interoperabilidad se encuentra en ejecución y está listo para gestionar la comunicación entre aplicaciones y servicios.
 **Respuesta**	Existe un fallo a nivel de interoperabilidad que impide la comunicación entre las aplicaciones.
 **Medida de la Respuesta**	El tiempo de inactividad para la conexión ha superado el tiempo estimado de 3 segundos.

En el ejemplo anterior se identifica claramente la métrica para controlar el escenario. Para ello se ha estimado un tiempo menor a 3 segundos para la espera de conexión. En este caso el escenario devolverá un mensaje al usuario informando acerca del fallo de comunicación entre componentes.

En un concepto muy general, la Disponibilidad está pensada en términos de **inactividad** de un componente o servicio. Las métricas para calcular la Disponibilidad de un sistema o componente deben ir centrados en;

- Concentrar esfuerzos donde son necesarios los cambios. En esta etapa se deben estipular de forma concreta los objetivos del cambio y la razón de ser de los mismos en función de los objetivos del negocio o el usuario.

- Detectar áreas críticas y dirigir todos los esfuerzos de desarrollo y costos de operación a aquellas tareas prioritarias donde el cambio introducido mejore la capacidad del sistema.

- Clasificar y detallar el componente o conjunto de éstos que puedan entrar en conflicto en un determinado momento y trazar planes para gestionar los posibles conflictos y tratar en la medida de resolverlos en forma simultánea para no afectar de forma general al servicio.

- Priorizar aquellos fallos detectados como inesperados cuando éstos se presenten y realizar los respectivos cambios ajustando las métricas y definiendo los objetivos de disponibilidad para dicho componente o servicio.

A continuación se describe una serie de métricas en torno al atributo de Disponibilidad, que igualmente son aplicados a los esquemas de ITIL y desarrollo de software.

Tabla 9. Métricas para la Gestión de Disponibilidad

REF	KPI	CÁLCULO
1	Total costes no planificados relacionados con disponibilidad	A
2	Índice de recuperación de disponibilidad	1-(C/B)
3	Promedio de fiabilidad	1-(E/D)
4	Nivel de instrumentación de la gestión de disponibilidad	F
5	Nivel de madurez de la gestión de disponibilidad	G
6	Media de confianza en el proveedor interno	1-(J/H)
7	Media de confianza en el proveedor externo	1-(K/I)
8	Índice de vulnerabilidades de seguridad	L/B
9	Índice de utilidad	N/M
10	Índice de riesgos de disponibilidad	P/O
11	Índice de mejora continua de disponibilidad	1-(Q/O)

REF	MÉTRICAS OPERATIVAS
A	Total costes no planificados
B	Número total de incidencias
C	Número total de incidencias impactando sobre el cliente
D	Disponibilidad total en minutos de todos los servicios entregados
E	No disponibilidad total en minutos de todos los servicios entregados

F	Nivel de instrumentación de la gestión de disponibilidad
G	Nivel de madurez de la gestión de incidencias
H	Número total de objetivos de servicios de proveedores internos
I	Número total de objetivos de servicios de proveedores externos
J	Número de objetivos perdidos de proveedores internos
K	Número de objetivos perdidos de proveedores externos
L	Número de incidencias relacionadas con seguridad
M	Número de CIs HW, SW y Red
N	Número de CIs HW, SW y Red no soportados por proveedores
O	Número de servicios en el Catálogo de Servicios
P	Número de servicios no cubiertos por un plan activo de disponibilidad
Q	Número de servicios sin revisión de disponibilidad los últimos 3 meses

CSF (Factores Críticos de Éxito)	**KPI**
Proveer servicios con disponibilidad apropiada a las necesidades del negocio	2,3,5,6,7,8,9,10
Demostrar costes efectivos en el plan de capacidad	1,4
Mejora continua de disponibilidad de los servicios entregados	11

Fuente: http://albinogoncalves.files.wordpress.com/2011/03/metricas-de-los-servicios-de-ti.pdf

El atributo de Disponibilidad se encuentra ampliamente ligado a los conceptos de ITIL, específicamente en lo concerniente a sistemas de Gestión de Incidencias a través de los SLA's "**acuerdo de nivel de servicio** o ***Service Level Agreement"***.

Un ***Incidente*** [Incident] (OCGa, 2007) (Operación del Servicio) es una Interrupción no planificada de un Servicio de TI o reducción en la Calidad de un Servicio de TI. También lo es el Fallo de un Elemento de Configuración que no ha impactado todavía en el Servicio. AENOR[11] en base a la Norma ISO/IEC 20000-1: 2005, define un incidente "*como cualquier evento que no es parte de la operación estándar de un servicio y que causa, o puede causar, una interrupción del servicio o una reducción en su calidad*". El principal objetivo del proceso de Gestión de Incidencias en consecuencia se encarga de devolver a una situación normal un proceso al menor costo en tiempo y recursos, minimizando el impacto sobre los procesos de negocio y asegurando el factor más importante, La integridad de la información.

Un **SLA** es un contrato escrito entre un proveedor de servicio y su cliente con objeto de fijar el nivel acordado para la calidad de dicho servicio.

Para resolver un incidente se debe tener en cuenta la **prioridad** del evento. La prioridad establece la secuencia lógica para la resolución del problema, determinada por el coste (impacto al negocio, urgencia y esfuerzo) del proceso o incidencia. La Prioridad se basa en el Impacto y la Urgencia, y es utilizada para identificar los plazos requeridos para la realización de las diferentes acciones. (OCGa, 2007).

[11] Asociación Española de Normalización y Certificación

Prioridad = Impacto + Urgencia

Según la (OCGa, 2007), el **impacto** es una medida o tendencia del efecto de un Incidente, Problema o Cambio en los Procesos de Negocio. El Impacto está a menudo basado en cómo serán afectados los Niveles de Servicio. El Impacto y la Urgencia se emplean para asignar la Prioridad.

La **Urgencia** es una medida de del tiempo en que un Incidente, Problema o Cambio tendrá un Impacto significativo para el Negocio. (OCGa, 2007).

En las Tablas 10 y 11 se muestra un ejemplo de un sistema de codificación de prioridad de incidentes, considerando el impacto y la urgencia y que puede ser aplicado a cualquier sistema.

Tabla 10. Escala de Calificación para Impacto y Urgencia

		Impacto		
		Bajo	**Medio**	**Alto**
Urgencia	Alto	3	2	1
	Medio	4	3	2
	Bajo	5	4	3

Tabla 11. Escala de Prioridades

		Descripción	**Tiempo de Resolución Admisible**
Prioridad	1	Crítica	1 Hora
	2	Alta	8 Horas
	3	Media	24 Horas
	4	Baja	48 Horas
	5	Re-Planificación	Planificado – a Mejorar

Es importante resaltar que este atributo es aplicado tanto al software como al hardware. El diseño de una determinada arquitectura deberá contemplar los aspectos de disponibilidad en ambos componentes para minimizar la interrupción del servicio y dotar de mayor capacidad a la herramienta software para aumentar los niveles de tolerancia a fallos para hacerlo más eficiente.

Los Concerns asociados a este atributo de Calidad son:

- Clases de servicios (QoS) (Quality of service).
- Tiempo planeado (Downtime – Planned, Unplanned).
- Tiempo de Recuperación (Time to repair).

- Tasa de fallas.
- Recuperación de desastres. (Disaster recovery).

5.2.2. Modificabilidad (Modifiability)

La Modificabilidad hace referencia al costo (tiempo/dinero) que podría significar un cambio en uno o varios componentes del sistema. El atributo de Modificabilidad determina cual será el costo promedio de esas nuevas implementaciones sin que éstas modifiquen otros componentes asociados al modulo sobre el cual se está efectuando la modificación o reestructuración.

El atributo de Modificabilidad como se ha mencionado en el párrafo anterior, está relacionado con el costo del cambio y está estrechamente ligada con otros aspectos de calidad como la Facilidad de Mantenimiento o Mantenibilidad, Extensibilidad y la Reusabilidad. Estos aspectos de calidad se deben enfocar desde las siguientes perspectivas: qué *se debe cambiar*, *cuándo cambiar* y *quién hará el cambio* (Bachman, Bass, & Nord, 2007);

- ***¿Qué se debe cambiar*?** Antes de realizar cualquier cambio, el arquitecto debe analizar las consecuencias que traerá dicho cambio. Se debe identificar qué componentes se verán afectados de forma directa o indirecta y qué deberá modificarse para que dichos componentes se ajusten a la nueva configuración del sistema y que permitan un mejor desempeño o performance de la rutina, algoritmo o modelo.

  Es muy frecuente que en los sistemas se den cambios en un momento dado, afectando no solo a los parámetros del software, sino también a aquellos relacionados con el hardware. Consideraciones como Interoperabilidad, plataformas de desarrollo, protocolos, interfaces, portabilidad, compatibilidad, etc; son aspectos a tener en cuenta antes de realizar cualquier cambio o modificación. Es obvio que este cambio causará un costo que podrá ser medido posteriormente.

- ***¿Cuándo cambiar y Quién hará el cambio*?** Anteriormente todos los cambios se hacían sobre código y por lo general se realizaba a la medida en la fase de desarrollo o cuando se presentaba alguna sugerencia de cambio, lo que llevaba a largos tiempos de implementación e inoperatividad casi total de un aplicativo. Hoy en día es posible realizar los cambios en cualquier momento, incluso por varios actores a la vez "programadores, usuarios y arquitectos", sin que esto afecte significativamente la funcionalidad del componente o servicio. Un caso particular son los ambientes web dinámicos.

El atributo de Modificabilidad o Modifiability según (Taylor, Medidovic, & Dashofy, 2010), "*es la habilidad del sistema para ser flexible frente a cambios inevitables*

durante su desarrollo y luego del despliegue. (en balance costo de construcción dentro de estos términos versus el costo de cambio)".

El atributo de Modificabilidad se puede caracterizar a través de los siguientes atributos: Complejidad, Portabilidad y Adaptabilidad. (Taylor, Medidovic, & Dashofy, 2010).

- **Complejidad**: Es una propiedad del sistema proporcional al tamaño del mismo, el número de sus elementos constitutivos, el tamaño y estructura de cada elemento y el número y naturaleza de sus interdependencias.

- **Portabilidad**: Es la habilidad que tiene el sistema para ser ejecutado en múltiples plataformas con mínimas modificaciones y sin degradación significante de sus características funcionales y no funcionales producto de su implementación.

- **Adaptabilidad**: Es la capacidad que tiene un sistema para satisfacer nuevos requerimientos y ajustarse a nuevas condiciones operativas durante su tiempo de vida.

Un ejemplo de escenario para el atributo de calidad de Modificabilidad se muestra en la siguiente tabla.

Tabla 12. Escenario para el Atributo Modificabilidad

ELEMENTO	ATRIBUTO / VALOR
 **Fuente de Estímulo**	Administrador de Base de Datos, Programador
 **Estímulo**	La definición de formato de fechas no se está almacenando correctamente en la Base de Datos.
 **Ambiente**	El aplicativo se encuentra en etapa de producción y requiere un ajuste inmediato. Hasta el momento la funcionalidad no se ha visto comprometida pero debe modificarse el atributo para que el motor de consultas funcione y retorne los datos esperados por el usuario y el SGBD.
 **Artefacto**	Código SQL asociado con la consulta y almacenamiento del tipo de dato y descriptores en las bases de datos.
 **Respuesta**	Los cambios realizados al modelo de datos no han afectado a otros componentes de desarrollo, por lo que la aplicación puede seguir funcionando normalmente.
 **Medida de la Respuesta**	Los cambios en el modelo de datos se estiman en tiempos no mayores a 1 día.

Los Concerns asociados a este atributo de Calidad son:

- Probabilidad de cambio.
- Magnitud y dimensión de cambio.
- Costo del cambio.
- Complejidad del cambio.
- Conservación del conocimiento de la solución.
- Confiabilidad del cambio.

5.2.3. Desempeño (Performance)

El Performance es la habilidad del sistema para ejecutar una tarea en forma predecible dentro de un perfil de desempeño definido. (Taylor, Medidovic, & Dashofy, 2010). El atributo de Desempeño está relacionado con todos los eventos que ocurren dentro del sistema y aquellos a los cuales debe responder el mismo. Un listado de eventos puede incluir, mensajes, excepciones, interrupciones, peticiones de los usuarios, peticiones de otros clientes o aplicativos, etc.)

El performance se caracteriza por cuanto considera explícitamente el tiempo que tomará al sistema responder a un determinado evento. El atributo de Performance por lo general se atribuye por ejemplo a número de transacciones realizadas por espacio de tiempo, o la optimización de recursos en base a dichas consultas o transacciones. El atributo de Desempeño "Performance" se puede caracterizar a través de 3 principales atributos o ***Concerns***: Latencia o tiempo de respuesta, Throughput o volumen de información y Escalabilidad.

- **Latencia**: es el tiempo que toma un dato en estar disponible desde que se realiza su petición. Este tiempo puede ser estimado por ejemplo en un grupo de transacciones entre un usuario y un sistema o un evento interno (nivel de respuesta para la interacción entre componentes, etc.).

- **Throughput**: corresponde al volumen de información que fluye a través de un sistema. Puede también denotarse como la capacidad que tiene el sistema para resolver un conjunto de requerimientos en un tiempo definido. Un mayor tiempo de respuesta implica un Throughput mayor. Este atributo puede ser analizado en conjunto con procesos de Predicción "Predictability" (tiempo máximo tolerado para producir una respuesta) y niveles de Fluctuación o Jitters (variaciones de tiempo en base al cálculo de un promedio dado).

- **Escalabilidad**: Es la capacidad que tiene el sistema para adaptarse a mayores volúmenes de trabajo o mayor carga operativa.

Un ejemplo de escenario para el atributo de calidad de Performance se muestra en la siguiente tabla.

Tabla 13. Escenario para el Atributo Rendimiento

ELEMENTO	ATRIBUTO / VALOR
 **Fuente de Estímulo**	Usuario
 **Estímulo**	Mostrar un Listado de Itinerarios
 **Ambiente**	En condiciones Normales el sistema devuelve el listado de los itinerarios, con el detallado de destinos, tiempos de viaje y alternativas de ruta al usuario.
 **Artefacto**	Sistema de Administración y Gestión de Itinerarios
 **Respuesta**	El itinerario es mostrado al usuario.
 **Medida de la Respuesta**	El tiempo de respuesta para que el listado sea desplegado al usuario no debe superar los 5 segundos tras su consulta.

Los Concerns asociados a este atributo de Calidad son:

- Tiempo de respuesta. (Thinking time, Response time).
- Latencia. (Throughputs).
- Picos de carga. (Peak Load)
- Cuellos de Botella. (Knee)

5.2.4. Seguridad (Security)

La **seguridad de la información** es el conjunto de medidas preventivas y reactivas de las organizaciones y de los sistemas para resguardar y proteger la información manteniendo la confidencialidad, la disponibilidad e integridad de la misma.

ISO, en su norma 7498, define la *seguridad informática como una serie de mecanismos que minimizan la vulnerabilidad de bienes y recursos*, donde un bien se define como algo de valor y la vulnerabilidad se define como la debilidad que se puede explotar para violar un sistema o la información que contiene. En el estándar 15408, la ISO define la seguridad dentro del marco denominado "*Information technology: Security Techniques Evaluation criteria for IT security*" como un conjunto de funcionalidades dentro de un esquema de evaluación de seguridad para TI. Algunos aparte de la Norma citan:

"(ISO/IEC 15408-1, 2009) *is useful as a guide for the development, evaluation and/or procurement of IT products with security functionality… ISO/IEC 15408 does not contain security evaluation criteria pertaining to administrative security measures not related directly to the IT security functionality. However, it is recognised that significant security can often be achieved through or supported by administrative measures such as organizational, personnel, physical, and procedural controls".*

***Un atributo de seguridad se define como**: ""**Security Attribute** is an property of subjects, users (including external IT products), objects, information, sessions and/or resources that is used in defining the SFRs and whose values are used in enforcing the SFRs*[12]

"ISO/IEC 15408 es una guía para el desarrollo, evaluación y adquisición de productos IT con funcionalidades de seguridad. ISO/IEC 15408 no contiene criterios de evaluación que permiten valorar los niveles de seguridad o las funcionalidades asociadas a estos u otros requerimientos. Sin embargo, se reconoce la importancia de la seguridad y los medios para alcanzar la misma apoyados en medidas administrativas y procesos de control".

"Un ***Atributo de Seguridad*** corresponde a un conjunto de propiedades de entidades, usuarios (internos y externos), objetos, información, sesiones y/o recursos que pueden ser utilizados para definir los Requisitos Funcionales a nivel de seguridad y que también son usados para hacer cumplir dichos requerimientos".

La Seguridad es la capacidad que posee un sistema para restringir el uso de servicios a usuarios no autorizados. La seguridad también se encarga de proveer los mecanismos necesarios para garantizar la disponibilidad de contenidos y servicios a aquellos usuarios catalogados como legítimos y que pueden hacer uso de los servicios del sistema. . Se caracteriza por previsión, confidencialidad, integridad, seguridad, disponibilidad, no repudio y de auditoría en el sistema.

Un intento para traspasar la seguridad es llamado ataque o amenaza y este puede tomar un gran número de formas con distintos objetivos específicos. Algunos de estos objetivos pueden ser: intentar de manera no autorizada acceder a datos o servicios, modificar información restringida, denegar el acceso o servicios a usuarios legítimos (registrados con distintos permisos de operar el sistema). (Bass, Len; Clements, Paul; Kazman, Rick, 2003).

La seguridad puede ser caracterizada como un sistema que provee cualidades principalmente de no-rechazo, confiabilidad, integridad, confianza, disponibilidad y verificación. (Bass, Len; Clements, Paul; Kazman, Rick, 2003).

[12] ***SFR***: Security Functional Requirement – Requerimientos Funcionales en Seguridad.

- **No-Rechazo (No-Repudio).** El No Repudio o irrenunciabilidad es un servicio de seguridad que permite probar la participación de las partes en una comunicación. Según la Norma OSI/ISO-7498-2 el No Repudio es un servicio de seguridad que previene que un emisor niegue haber remitido un mensaje (cuando realmente lo ha emitido) y que un receptor niegue su recepción (cuando realmente lo ha recibido).

- **Confidencialidad.** Es un mecanismo que garantiza que la información o conjunto de servicios están protegidos para usuarios no autorizados o sin privilegios. Cualquier dato o servicio que se encuentre bajo el marco de la confidencialidad, debe estar protegida o parametrizada por un conjunto de reglas que limiten el acceso a estos datos. De acuerdo con los niveles de acceso dispuestos en el sistema, un usuario solo podrá hacer uso de sus capacidades en función de los privilegios concedidos tanto por el administrador como por la aplicación.

- **Integridad.** Es la propiedad que busca mantener los datos libres de modificaciones por parte de personas o procesos no autorizados, conservando la exactitud y estructura de los mismos, sin que éstos se alteren. La integridad permite mantener con exactitud la información tal cual fue generada, sin ser manipulada o alterada, a menos que una persona autorizada proceda a realizar un determinado cambio en la información, lo que asegure su disponibilidad, precisión y confiabilidad.

- **Autenticación.** Esta propiedad garantiza que solo los individuos autorizados tengan acceso a los recursos o servicios. La autenticación puede ser considerada como un aspecto más de la integridad "*si está firmado por Juan, realmente es Juan quien lo ha enviado*". En seguridad informática, la autenticación puede ser obtenida a través de los siguientes mecanismos: Mensajes mediante Firma Digital (que consiste en asegurar que el mensaje m proviene del emisor y no de otro); autenticación mediante Contraseñas (a través de claves privadas y cifradas) y por medio de Dispositivos (como llaves electrónicas).

- **Disponibilidad.** El objetivo de la disponibilidad es garantizar el acceso a un servicio o a un conjunto de recursos, en base a unas reglas y condiciones de acceso.

- **Verificación.** Cualquier sistema que contenga un conjunto de reglas y parámetros de seguridad, deberá proveer los mecanismos necesarios para realizar las auditorías tanto a los sistemas como a los usuarios. Esta propiedad permite validar la información y preservarla en el tiempo y su finalidad se orienta a la Recuperabilidad de los sistemas e información cuando ocurra una vulneración o se requiera de una reconstrucción parcial o total de los datos.

La seguridad es fundamental para enfrentar las distintas vulnerabilidades a las cuales se ve avocado el software. Un modelo de seguridad bien establecido permitirá proteger no solo a los datos sino también a los usuarios. Cuando intercambiamos información, estos datos que por lo general viajan en formato digital a través de una red de comunicaciones la cual circula por un número no precisado de servidores, plataformas y dispositivos, de los cuales se desconoce si poseen o no mecanismos para el control de los datos. Tampoco es predecible saber si donde se está realizando la conexión el propietario de dicho nodo es confiable y no posee algún mecanismo que usurpe la información y utilice esta para fines delictivos o para destruir la información o un aplicativo de software. Una de las tareas del mecanismo de seguridad es precisamente prever este tipo de ataques y daños a los sistemas informáticos.

En el mundo de la informática existen diversos mecanismos y malas prácticas que hacen que los sistemas sean vulnerables a ataques. Los fallos de seguridad pueden darse tanto en la etapa de desarrollo como en la etapa de producción y pueden ser provocados de forma intencional o accidental, tanto por usuarios internos o desarrolladores como por usuarios externos. Los fallos más comunes a los que se enfrenta el software se derivan de:

- Fallos debidos a errores desconocidos en el software, o conocidos sólo por usuarios hostiles (por lo general se evidencia en una baja calidad del código. Es una falla muy compleja ya que de ella se derivan robos, fraudes electrónicos y sabotajes a los sistemas informáticos de la organización);

- Fallos debidos a errores o bugs conocidos pero que no han sido corregidos. (afán de los desarrolladores por vender un producto sin el debido control de de calidad, control de pruebas y control de versiones);

- Fallos debidos a una mala configuración del software. (por lo general se debe a una escasa documentación en el software, instalando características inapropiadas u otro elemento no compatible lo que vulnera el funcionamiento del software. Por lo general este tipo de procedimiento da origen a lo que se conoce como brecha de seguridad);

- Fallos de despliegue. (es una práctica aprovechada por los piratas informáticos que utilizan los sitios de descarga de los desarrolladores para bajar el software y realizar en ellos procesos de ingeniería inversa "Keygens, Seriales, Bloqueos de archivos Host, entre otras técnicas" para romper cualquier mecanismo de seguridad en el paquete de instalación).

- Fallos a nivel de mantenimiento. (cuando una entidad desarrolladora no genera los mecanismos de actualización que permiten eliminar ciertas vulnerabilidades detectadas tanto por el grupo de desarrolladores como por otros agentes que utilizan el software). El uso continuo de los Service-Pack o Parches de actualización es una práctica que permite no solo corregir problemas en los algoritmos, sino que también permite agregarle ciertas funcionalidades, actualizar las existentes y lo más importante introduce en

ellos mecanismos para mejorar el performance y la seguridad de la herramienta software.

- Fallos a nivel de virus informático u otro tipo de vulnerabilidades como:

    - Ataques informáticos.
    - Bluejacking.
    - Ingeniería social.
    - Exploits y ataque de día cero.
    - Hacking.
    - Cracking.
    - Malware.
    - Spyware.
    - Ataque de denegación de servicios.
    - Man in the middle.
    - Ataques de REPLAY.
    - Ataque por fuerza bruta.
    - SQL injection.
    - Cross-site scripting y Cross Site Request Forgery.
    - Spoofing: IP, web, ARP, DNS, etc.
    - Desbordamiento de búfer.

Un escenario de seguridad se presenta en la siguiente tabla.

Tabla 14. Escenario para el Atributo Seguridad

ELEMENTO	ATRIBUTO / VALOR
 **Fuente de Estímulo**	Administrador de la Plataforma.
 **Estímulo**	Cambio en el Rol de un Usuario.
 **Ambiente**	El cambio de Roles se puede dar en cualquier momento, bajo cualquier circunstancia, siempre y cuando el usuario haya obtenido los permisos y el Rol para acceder a determinada información.
 **Artefacto**	Sistema de Gestión de Usuarios y Directivas de Seguridad.
 **Respuesta**	El cambio en el Rol se debe reflejar inmediatamente al usuario. Es probable que este proceso exija de una confirmación vía mail con los nuevos parámetros y accesos.
 **Medida de la Respuesta**	Una vez confirmado el nivel de acceso y el Rol del usuario, se procederá a realizar el cambio de forma inmediata.

Los Concerns asociados a este atributo de Calidad son:

- Políticas.
- Amenazas.
- Detección de Ataques.
- Recuperación de Ataques.
- Las demás citados por (Bass, Len; Clements, Paul; Kazman, Rick, 2003) "Nonrepudation, Confidentiality, Integrity, Assurance, Abailability y Audit"

5.2.5. Facilidad de Pruebas - Verificabilidad (Testability)

El atributo de pruebas se refiere a la facilidad que tiene un sistema para ser probado en sus unidades o módulos o de forma integral. Por lo general el 40% de un proyecto de desarrollo tiene que ver con la aplicación de pruebas de unidad y pruebas de integración de componentes.

Las pruebas corresponden a una práctica importantísima al momento de desarrollar líneas de producto software con altos componentes de calidad. Estas pruebas han demostrado ser eficientes en la valoración y validación de código a lo largo del proceso de ingeniería de software. En la actualidad dichas pruebas se han elevado al nivel de procesos automatizados, ya que existen sinnúmero de herramientas que permiten medir y realizar seguimientos estrictos a la implementación de métodos o clases por ejemplo.

La elaboración de pruebas totalmente automatizadas y software que realice mediciones sobre el código y el producto, se están convirtiendo en un estándar de desarrollo para asegurar la calidad de los productos en una factoría de software, independiente del número de desarrolladores. Todos estos esfuerzos están direccionados hacia un mismo objetivo, y ese objetivo se llama Reutilización.

Las pruebas descritas en un proceso de ingeniería de software se clasifican en Unitarias y de Integración.

- **Pruebas unitarias (*Unit Tests*).** Las pruebas de unidad tienen como fin probar los subprogramas, las subrutinas, los procedimientos individuales o las clases en un programa. Esta etapa es esencial ya que permite la revisión de unidades o bloques más pequeños de código evitando revisiones exhaustivas a códigos o programas completos. La razón principal para realizar pruebas unitarias de software se concentra en la necesidad de encontrar una falla en un componente que aún no ha sido integrado a la totalidad del sistema. De esta manera es más efectiva la búsqueda y depuración y el tiempo de corrección será relativamente corto y fácil de implementar.

- **Pruebas de Integración (*Integration Tests*).** Este tipo de pruebas se realiza para garantizar que los diferentes módulos que componen un artefacto de software, se integren con la aplicación de forma transparente. El propósito de la *prueba de*

integración consiste en realizar pruebas para verificar que un gran conjunto de partes de software funcionan juntos. Las pruebas de integración (algunas veces llamadas integración y testing) sirven para combinar los módulos individuales para posteriormente ser testeados como una unidad. Este tipo de prueba se pueden realizar a la par con las unitarias sin que esto afecte el desarrollo del producto y la funcionalidad como tal.

La detección de fallos en un sistema ya integrado resulta caótica en la mayoría de los casos incrementando no solo los tiempos de desarrollo sino que también se incurre en elevados costos de producción, mano de obra lo que afecta la calidad del producto.

El escenario de verificabilidad se basa, según (Bass, Len; Clements, Paul; Kazman, Rick, 2003), en que un Tester "probador" ejecuta una unidad de prueba sobre un componente del sistema, que consta de un interfaz para controlar su comportamiento y observar el resultado obtenido.

Un escenario de Testability o Verificabilidad se presenta en la siguiente tabla.

Tabla 15. Escenario para el Atributo Verificabilidad

ELEMENTO	ATRIBUTO / VALOR
 **Fuente de Estímulo**	Tester – "Probador".
 **Estímulo**	Realizar Pruebas Unitarias sobre un Componente.
 **Ambiente**	Etapa de Desarrollo y Mantenimiento.
 **Artefacto**	Componente que calcula las deducciones de una nómina de pago a empleados con base en nuevos ajustes salariales y de retenciones.
 **Respuesta**	80% de Realización de la Prueba
 **Medida de la Respuesta**	Entre 3 y 4 Horas.

5.2.6. Usabilidad (Usability)

El atributo de calidad de usabilidad especifica la facilidad con la cual el usuario puede interactuar con el sistema, y la facilidad con la que éste puede realizar una determinada tarea.

De acuerdo con la (ISO-9126, 1982), el estándar 9126 (*Software Product Evaluation - Quality Characteristics and Guidelines for the User*), considera la usabilidad como: *"... un conjunto de atributos de software, que se sostienen en el esfuerzo necesitado para el uso y en la valoración individual de tal uso, por un conjunto de usuarios declarados o implicados…"* también la definen en ese mismo estándar como *"...la eficacia, eficiencia y satisfacción con la que un producto permite alcanzar objetivos específicos a usuarios específicos en un contexto de uso específico".*

En 1993, Jakob Nielsen definió los cinco atributos básicos de la **usabilidad**:

- **Facilidad de aprendizaje:** rapidez con que un usuario aprende a utilizar un sistema sin tener conocimientos previos de éste o que lo pueda operar sin necesidad de manuales de usuario. (este atributo permite al usuario interactuar con la herramienta de forma sencilla, rápida e intuitiva);

- **Eficiencia:** que producto de esa facilidad, el usuario pueda alcanzar un alto nivel de productividad;

- **Retención en el tiempo:** si por algún motivo el usuario dejara de emplear el sistema por un tiempo determinado, éste atributo permite que el usuario pueda retomar su utilización como una tarea casi que cotidiana sin necesidad de entrenamiento.

- **Tasas de error de los usuarios:** al cometer un error el usuario, el sistema debe estar en capacidad de informar al usuario y ayudarle a solventar su solución y

- **Satisfacción subjetiva:** hace referencia al gusto que siente el usuario al emplear la herramienta (comodidad y satisfacción).

Ben Schneiderman sumó a los principios de la **usabilidad** de Nielsen uno más en 1998:

- **Velocidad de utilización:** este atributo trata de minimizar los tiempos de consulta y acceso para que el usuario pueda realizar una tarea en forma rápida, una vez conoce el funcionamiento del sistema. Este atributo está asociado al concepto de Atajos o Shortcuts en las aplicaciones.

En 2001, John Cato sugirió tres atributos más para asegurar la **usabilidad** de un producto:

- **Control:** los usuarios deben tener el control sobre el sistema, y no a la inversa;

- **Habilidades:** el sistema debe responde con calidad a la experticia y experiencia del humano y no al revés. El sistema debe convertirse en una extensión del pensamiento y la habilidad humana y

- **Privacidad:** el sistema debe ayudar a los usuarios a proteger su información. (Confidencialidad y Seguridad).

Otros atributos igualmente importantes y que deben tenerse en cuenta al momento de procesar un atributo de usabilidad corresponden a:

- **Flexibilidad:** existencia de variantes para realizar una o más tareas y

- **Robustez:** convertirse en una herramienta que de soporte al usuario facilitando el cumplimiento de sus objetivos.

(Bass, Len; Clements, Paul; Kazman, Rick, 2003), hacen referencia a la usabilidad en los términos definidos por ISO y agregan: aparte de estas características, la usabilidad debe incluir mecanismos de comprensibilidad, operabilidad y facilidad de aprendizaje y el diseño amigable o intuitivo de las interfaces.

Un escenario de Usabilidad se presenta en la siguiente tabla.

Tabla 16. Escenario para el Atributo Usabilidad

ELEMENTO	ATRIBUTO / VALOR
 **Fuente de Estímulo**	Usuario
 **Estímulo**	Minimizar el impacto de error al momento de cancelar una operación.
 **Ambiente**	Proceso Normal del Sistema en Condiciones Normales de Ejecución.
 **Artefacto**	Modulo Y
 **Respuesta**	La Operación ha Sido Cancelada
 **Medida de la Respuesta**	Invariabilidad del sistema antes y después del proceso.

Los Concerns asociados a este atributo de Calidad son:

- Interfaz de usuario.
- Proceso de negocio.
- Calidad de la información.
- Alineamiento de capacidad de usuario con interfaz.
- Crecimiento de productividad acorde a aprendizaje en el uso.

Tabla 17. Ejemplos de algunos Escenarios

ATRIBUTO DE CALIDAD	FUENTE DE ESTÍMULO	ESTÍMULO	AMBIENTE	ARTEFACTO	RESPUESTA	MEDIDA
Los usuarios inician 1.000 transacciones X, por minuto (probabilidad) bajo condiciones normales, de 9 a 1 8:00hs, el sistema debe procesarlas (resultado en pantalla) en una latencia menor a 3 segundos.						
Rendimiento	Usuarios. Definir el tipo de Usuario si es necesario.	Inicio de Tx X. Probabilístico. 1.000 Rx por minuto	Procesamiento Normal. De 9 a 18. Carga Normal.	Todo el Sistema	Transacción procesada	La latencia debe ser menor a 3 segundos
Si el Middleware tiene alguna falla y no recibe pedidos, para transacciones que necesitan ser procesadas de manera asincrónicas se debe auditar en un archivo de log y se debe reintentar X veces con un intervalo de Y segundos						
Disponibilidad	El Middleware está caído	Transacciones asincrónicas a procesar	En todo momento	Manejador de Transacciones e Interfaz con el Middleware	Guardar en Log y Reintentar	X veces con un intervalo de Y
Un stakeholder desea que el sistema publique algún servicio para ser consumido por otra aplicación (y no por pantalla) luego de que el sistema esté en su primera release de producción y el tiempo de desarrollo (analysis, design, code, test y deploy) no supere los 25 días desde la aprobación del Change Request						
Modificabilidad	Stakeholder	Desea agregar funcionalidad	Etapa de Mantenimiento	Core Service e Interfaces	Cambio de Requerimiento correctamente implementado	Que no supere los 25 días
Si el administrador de accesos (seguridad) realiza, agrega o quita algún rol a cualquier usuario, estos cambio deben ser reflejados de manera inmediata en la sesión del usuario a la cual se le modificaron los accesos						
Seguridad	Administrador de Accesos	Cambio de roles	En cualquier momento	Todo el Sistema	Reflejar los cambios	Inmediatamente
Un tester unitario que realiza el test de un componente X debe poder ejecutar los scripts y debe poder llegar a un nivel de completitud del 70% en 4 horas						
Verificabilidad	Tester unitario	Testear el componente	En etapa de desarrollo y mantenimiento	Componente X	70% de completitud	4 Horas
Los usuarios del modulo X, una vez que ejecutan cualquier acción, el sistema debe proveer posibilidad de cancelarlas durante su ejecución y pueda como antes de la ejecución de la misma						
Usabilidad	Usuarios del módulo X	Minimizar el impacto de error cancelando operaciones	Procesamiento Normal. Carga Normal	Módulo X	Transacción cancelada	Sistema integro antes de la ejecución

Fuente: http://apit.wdfiles.com/local--files/start/02_apit_atributos_de_calidad.pdf

6

ESTRATEGIAS ARQUITECTÓNICAS

"Programar sin una arquitectura o diseño en mente es como explorar una gruta sólo con una linterna: no sabes dónde estás, dónde has estado ni hacia dónde vas"
-- Danny Thorpe

Una Estrategia Arquitectónica corresponde a un ***CONJUNTO DE TÁCTICAS*** aplicadas al diseño y desarrollo de sistemas. Este conjunto de estrategias conlleva a la aplicación de una serie de atributos que permiten controlar los comportamientos y respuestas de un sistema en función de una respuesta o necesidad de un modelo de negocio.

Según (Bass, Len; Clements, Paul; Kazman, Rick, 2003), en su libro Software Architecture in Practice, una Táctica responde a *"…una decisión de diseño que influye en el control de una respuesta a un atributo de calidad…"*. Dichas decisiones están basadas en un conjunto de requerimientos que describen el comportamiento del sistema a través de escenarios de calidad, los cuales especifican las respuestas de cualquier artefacto o componente de software en función de un estimulo o conjunto de éstos. El conjunto de atributos o reglas que definen el software en torno a la calidad se denomina **Táctica**.

Una **Táctica** es la manera en que los diseñadores y arquitectos deciden organizar los componentes del sistema, así como asignarle características específicas a cada uno de ellos para controlar sus comportamientos y salidas (posibles respuestas a eventos) para así soportar uno varios atributos de calidad específicos. (Bass, Len; Clements, Paul; Kazman, Rick, 2003).

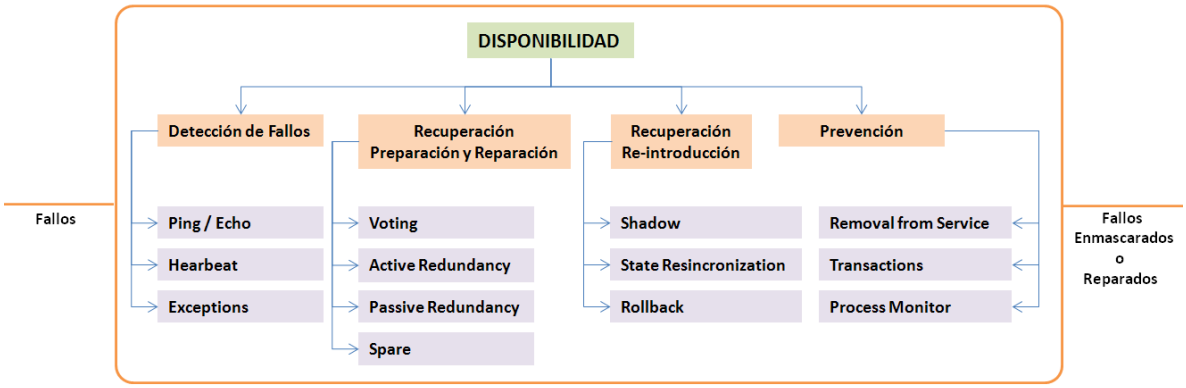
La principal funcionalidad de las tácticas consiste en dar soporte a los posibles fallos del sistema, y a través de ellas, hacer que el arquitecto prevea algunos comportamientos indeseados evitando un mayor esfuerzo en procesos de re-diseño. Otra característica de las tácticas responde a que la aplicabilidad de las mismas asegura o garantiza que el software preste su servicio de forma ininterrumpida a partir de sus especificaciones.

6.1. TÁCTICAS PARA EL CONTROL DE ATRIBUITOS DE CALIDAD

6.1.1. Tácticas aplicadas a la Disponibilidad (Availability)

Las tácticas asociadas a la disponibilidad tratan de disminuir el impacto provocado por los potenciales fallos en la arquitectura del sistema y que de alguna manera ponen en riesgo la funcionalidad de un componente o el sistema en su conjunto. Las tácticas de disponibilidad tratan de limitar al máximo estos efectos y provee un conjunto de estrategias o concerns que hagan posible su reparación en lo posible. Estas tácticas se orientan a 3 aspectos: ***Detección, prevención y recuperación***.

Figura 7. Tácticas Arquitectónicas de Disponibilidad



6.1.1.1. Detección de Fallos (Fault Detection)

- **Ping/Echo**: el método Ping responde a una técnica para el envío de paquetes ICMP a través de un conjunto de redes basadas en IP. Esta técnica permite evaluar la calidad y disponibilidad de la conexión y la calidad del servicio en términos de latencia y performance en la conexión. Para poder establecer este mecanismo es indispensable el diseño de componentes que evalúen dichos canales a través del conjunto de capas del modelo de red o un modelo determinado de arquitectura.

- **HeartBeat** (Latido, Frecuencia, Pulsación): es una técnica que se utiliza para comunicar dos nodos mediante tráfico multicast y se caracteriza por ser un método basado en altas disponibilidades de servicio tanto del lado cliente (activo) como del servidor (pasivo). Está técnica se basa en el envío y escucha de los mensajes entre nodos. Si el mensaje HeartBeat enviado por uno de los componentes falla, se considera que éste no está controlando de forma adecuada el evento. A partir de ello el componente deberá emitir el fallo y si procede realizar las respectivas correcciones al fallo.

- **Excepciones** (Exceptions[13]): es una técnica que permite manejar un conjunto de fallos ocurridos durante la ejecución de un sistema. En el ámbito de la programación la utilización de excepciones hace que las herramientas de software sean más tolerantes a fallos y que éstas describan de alguna manera cierta robustez frente al control de dichas excepciones, sin que dichos fallos afecten de forma general al sistema o componente.

  Las excepciones pueden ser disparadas a través de clases o Wrappers. Una clase de tipo exception contiene un conjunto de algoritmos que detallan un caso de fallo probado y documentado. Este mecanismo permite una

[13] El manejo de excepciones ayuda a lidiar con los errores de una aplicación por medio de la manipulación del código para hacer programas más robustos

ejecución de errores en forma limpia y depurada sin recurrir a fallos generales. La técnica wrapper es una estrategia que permite la interpretación de datos primitivos como clases u objetos.

Otros mecanismos para Detección de Fallas pueden ser:

- WatchDog / Centinels (Monitoreo de Procesos),
- Paramenter Fence,
- Parameter Typyng.
- TRM Triple Modular Redundancy (control de Redundancias).

6.1.1.2. Recuperación de Fallos (Fault Recovery)

- **Voting**: es un mecanismo utilizado en ambientes Cliente/Servidor en accesos concurrentes y de multiprogramación. Esta técnica posee un conjunto de algoritmos que comparan los valores en dos sistemas paralelos (clúster de Microprocesadores o Discos Duros) para corroborar las respectivas salidas. El voter evalúa la consistencia de dichos datos y si son equivalentes, no se reporta fallo; si por el contrario al computar ambas salidas éstas son diferentes, el mecanismo detecta tal anomalía y verifica el estatus de los registros e intenta recuperar el fallo.

- **Redundancia Activa** (Active Redundancy): la redundancia es una de las técnicas más utilizadas para mejorar la tolerancia a fallas en los sistemas de información. La tolerancia a fallos es la capacidad que tiene un sistema para continuar funcionando normalmente después de producirse un fallo. Cuando esto sucede, se requiere de una reconfiguración del sistema o servicio a través de mecanismos redundantes para sustituir al componente causante del fallo. Este tipo de mecanismos es muy utilizado a nivel de Bases de Datos para Replicación de información y manejo de estructuras redundantes para evitar pérdidas de información al momento de acceder a los datos en los sistemas de gestión.

- **Redundancia Pasiva** (Passive Redundancy): las técnicas de redundancia **pasiva** se relacionan con el concepto de **enmascaramiento de fallos** y la capacidad de poder encubrir dichos fallos. Mediante estos métodos no se evitan fallos, pero sí se reduce el impacto de sus consecuencias.

- **Spare** (Redundancia, de reserva, de recambio, de respaldo en caliente): es una técnica de redundancia utilizada como mecanismo de respaldo en caso de que alguno de los componentes produzca un determinado fallo. Tan pronto se produce la anomalía, un componente de iguales características sale al escenario como medio de respaldo para evitar la desconexión del servicio. Esta técnica requiere por lo general reinicios de sistema ya que el proceso se asemeja a un estatus de actualización del componente, por lo que

el sistema debe iniciar y comprobar el componente. Es una técnica que se asemeja a los puntos de restauración en un sistema operativo. Los puntos de restauración son copias de seguridad que el sistema operativo hace cuando detecta algún cambio en el sistema, al instalar programas nuevos, al desinstalarlos, etc.

- **Shadow** (Sombra): es una táctica que permite reintroducir cambios en los componentes que han reportado algún fallo, de tal manera que éstos puedan ser puestos a prueba por un periodo determinado. Esta técnica "a la sombra" busca determinar si el componente es apto o no para continuar su funcionalidad. Una vez el componente dispuesto en modo de respaldo pasa las pruebas de funcionalidad, el servicio será restaurado de forma automática por el sistema; de lo contrario se procesará un error crítico en la aplicación, desestabilizando a uno o más componentes.

- **Mecanismo de Re-sincronización** (State Resincronization): es un mecanismo que permite la sincronización adecuada de los sistemas de redundancia activa y pasiva en un sistema antes de que el servicio sea restaurado en su totalidad. Es una técnica que permite la actualización del estado de un componente y realiza sincronizaciones periódicas a los mismos para verificar su funcionalidad.

- **Checkpoint / Rollback**: es una táctica que permite al sistema revertir un proceso a un estado previo conocido (o funcional), después de haberse detectado un fallo. La técnica Rollback a menudo se combina con la táctica de redundancia y TRM "Triple Modular Redundancy" para el manejo de estados "***En Espera***" a estado "***Activo***". El éxito del rollback radica en que las copias o versiones anteriores se hayan guardado en forma íntegra y sin fallos.

Otros mecanismos para Recuperación de Fallas pueden ser:

- Function Patches (Parches y Funcionalidades),
- Class Patches,
- Hitless In-Service Software Upgrade (ISSU),
- Coordinated Checkpointed (Rollback),
- Reinicio escalado,
- Redundancia basada en rutas críticas,
- Procesos de intercambio ('switching', 'live switching', 'hotswap').

6.1.1.3. Prevención de Fallos (Fault Prevention)

- **Remoción de Servicios** (Removal from Services): es una táctica orientada a la suspensión del servicio para un componente determinado mucho antes

de que éste presente un fallo. Esta táctica disminuye los riesgos potenciales y las anomalías que se pueden presentar en el sistema afectando el servicio.

- **Transacciones** (Transactions): consiste en una técnica para asegurar la integridad y consistencia de los datos. Las transacciones responden a mecanismos secuenciales y en conjunto conforman una sola funcionalidad. Esta táctica se asegura de proporcionar el mecanismo adecuado para controlar las secuencias del conjunto de componentes para prevenir alteraciones en los datos antes de que éstos sean soportados por la funcionalidad de los componentes, y también controla el acceso compartido a dichos datos optimizando los procesos basados en Threads o hilos de procesos.

- **Monitorización de Procesos** (Process Monitor): es una técnica que permite monitorizar el funcionamiento de un componente en tiempo de ejecución. Una vez se ha detectado el fallo en el componente, el sistema de monitorización debe ser capaz de finalizar la ejecución del componente e iniciar una nueva instancia del mismo para asegurar la operatividad del sistema.

Otros mecanismos para Prevención de Fallas pueden incluir:

- Atomic Commit Protocol (Transacciones),
- Exception Classes (Conjunto de clases para manejo global de excepciones),
- Smart Pointers,
- Wrappers.

6.1.2. Tácticas aplicadas a la Modificabilidad (Modifiability)

La táctica de Modificabilidad se basa en dos conceptos: **Cohesión** y **Acoplamiento**. De acuerdo con (Stevens, Myers, & Constantine, 1999)

> El **Acoplamiento** es *"…the measure of the strength of association established by a connection from one module to another… the coupling is reduced when the relationships among elements not in the same module are minimized…"* y la **Cohesión** es *"…the measure of the relationship among the responsibilities of a specific module…"*. (Bachman, Bass, & Nord, 2007).

> El **Acoplamiento** es *"… la medida o tendencia que permite establecer la fortaleza de las asociaciones establecidas entre módulos y su correspondencia entre el uno y el otro… el nivel de Acoplamiento se obtiene cuando las relaciones entre dichos elementos se separan del módulo en sí, minimizando el grado de dependencia entre módulos…"* y la

Cohesión es *"… la medida de las relaciones entre responsabilidades de un módulo específico…"*. (Bachman, Bass, & Nord, 2007).

Una táctica de Modificabilidad corresponde a un proceso de transformación en una arquitectura de software para mejorar sus atributos de calidad. La Modificabilidad está basada en la Cohesión y Acoplamiento "***maximum Cohesion and minimum Coupling***", para indicar que un sistema o componente debe poseer una alta cohesión y un mínimo grado de acoplamiento para considerarlo como un modelo escalable y con alta usabilidad. Las tácticas de Modificabilidad en esencia persiguen la reducción de los acoplamientos y la reducción de costos por hacer una modificación a un determinado módulo en una arquitectura de software.

En la táctica de Modificabilidad los 2 atributos mencionados anteriormente se orientan a la definición de funciones en base a un conjunto de responsabilidades que describen de alguna manera el cómputo y eficiencia del módulo.

La Modificabilidad se basa en las Responsabilidades. Una responsabilidad es una acción o decisión arquitectónica que está sujeta a ser mantenida o resuelta en términos de salida (respuesta) fuera del sistema o dentro de éste. Una responsabilidad es asignada a un módulo y a un conjunto de relaciones entre dichas responsabilidades.

En cada iteración, cada responsabilidad tiene un costo asociado a ciertos atributos de modificación. En el caso de ArchE, los costos de modificación se basan en funciones probabilísticas para determinar el costo global de la transformación calculando el costo y la reducción de sus aspectos tras la modificación.

La táctica de Modificabilidad es un proceso que reorganiza o descompone un conjunto de responsabilidades en base a la granularidad y fusiona dichos atributos en un nuevo juego de responsabilidades. En otros casos, la Modificabilidad afecta al módulo trasladando ciertas responsabilidades a otros módulos para solventar la carga operativa del módulo.

Una relación entre dos responsabilidades se define en función del grado de Acoplamiento, un juego probabilístico que modifica los atributos de un módulo y los asigna a otro de forma autónoma, para disminuir los grados de dependencia entre módulos y responsabilidades. Los niveles de Acoplamiento revisten un comportamiento asimétrico ya que el nivel de acoplamiento de un componente a otro no es conmutable, es decir; no es igual el grado de Acoplamiento entre responsabilidades de A a B que el grado de Acoplamiento entre B y A. Siempre podrán existir diferencias en el grado en que una depende de otra.

Para el caso de la Arquitectura ArchE, el atributo de Modificabilidad está basado en modelos predictivos que involucran;

- **Costos promedios por modificar una simple responsabilidad**: este cálculo permite reducir el costo promedio de una modificación a una responsabilidad sin que estos cambios repercutan en el costo promedio del

escenario. Esta táctica divide las responsabilidades para disminuir el costo promedio bajo el esquema de "divide y vencerás".

- **Acoplamiento**: como se ha descrito anteriormente, este método permite reducir las dependencias entre componentes.

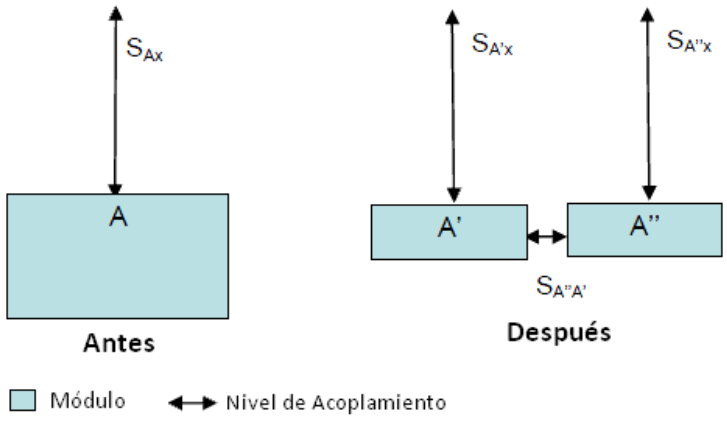
6.1.2.1. Reducción de Costos de Modificación en una Simple Responsabilidad

La forma más común para reducir costos de modificabilidad en una responsabilidad consiste en transformar dicha arquitectura en un conjunto derivado de responsabilidades. La táctica Split divide las responsabilidades y la asigna a otro escenario dentro del mismo modelo. Si el modelo posee una definición concreta de varias responsabilidades, este proceso se realizará de forma iterativa hasta hallar los cambios considerados como óptimos y que incrementan la funcionalidad del modelo.

Para poner en contexto un ejemplo de la táctica Split, se tienen un escenario donde una de las responsabilidades exige un alto grado de capacidad. Si asociamos esto a lo complejo del proceso, podremos estimar que los costos por producir dicho módulo serán elevados. Para mejorar el rendimiento del escenario, se refina la responsabilidad en unidades de responsabilidad más pequeñas para reducir el impacto de la modificación.

Al dividir las responsabilidades y transformar la arquitectura el modelo se parecerá al siguiente esquema:

Figura 8. División de Responsabilidades (Split)

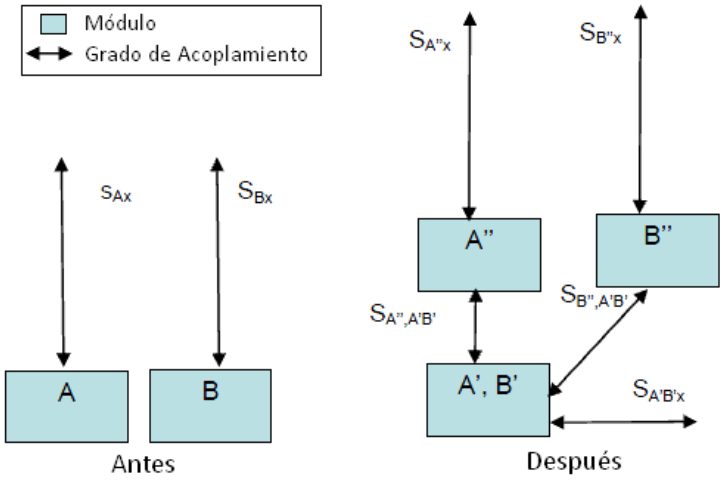


Fuente: Adaptada de (Bachman, Bass, & Nord, 2007).

6.1.2.1.1. División de Responsabilidades (Split Responsability)

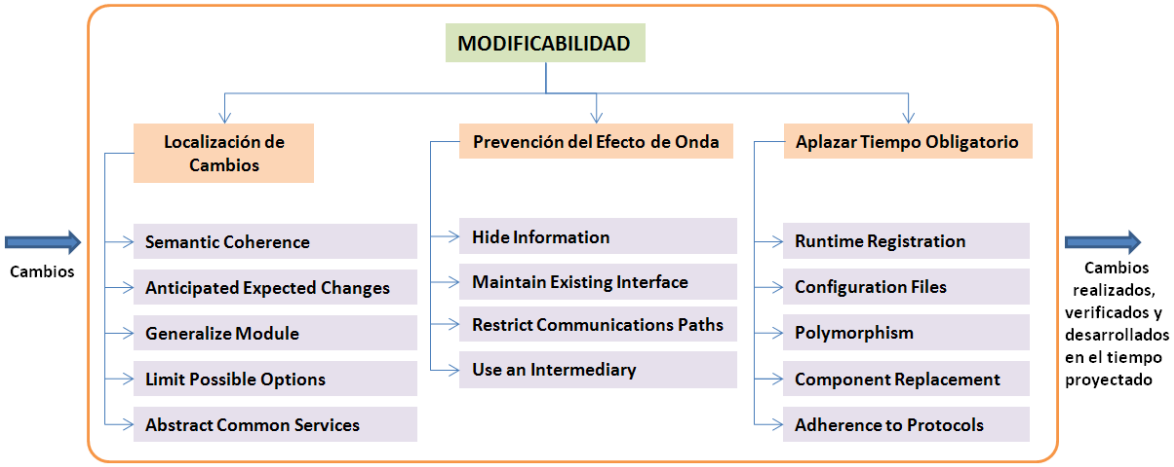
Otra táctica aparte de la división de responsabilidades consiste en la técnica que permite incrementar el grado de Cohesión en un módulo o componente. El propósito de esta táctica consiste en trasladar una responsabilidad de un módulo a otro para reducir los efectos de la responsabilidad en el módulo original. Básicamente lo que detalla este procedimiento está en dividir las responsabilidades, crear un nuevo módulo y asignar a éste las responsabilidades tanto de A como de B para incrementar la cohesión en el modelo. Un esquema que ilustra esta táctica se muestra a continuación.

Figura 9. Incremento de la Cohesión



Fuente: Adaptada de (Bachman, Bass, & Nord, 2007).

Figura 10. Tácticas Arquitectónicas de Modificabilidad



6.1.2.2. Incremento en la Cohesión

- **Coherencia Semántica** (Maintain Semantic Coherence): el objetivo de esta táctica consiste en garantizar que todas las responsabilidades trabajen en forma sincronizada y sin fuertes dependencias entre módulos. Los costos de una modificación en base a la coherencia se da siempre y cuando los costos de modificar las responsabilidades de A' y B' (Destino) son menos costosos que las modificaciones de A y B (Fuente) y cuando los costos no afectan a los módulos A'' y B'' tras las transformación (Ver Fig.9).

- **Servicios Comunes Abstractos** (Abstract Common Services): consiste en realizar instancias o clases comunes con aquellas funcionalidades más recurrentes dentro de un modelo. La idea de esta táctica es producir una clase u objeto a partir de una serie de reglas bien definidas y que pueden ser reutilizadas por otros componentes.

  Un paralelo o punto de referencia para esta técnica consiste en la creación de librerías para un determinado sistemas las cuales puedan ser invocadas por otros procesos o arquitecturas. Ejemplo; la API para proceso de grabación de formato MP3 "*Nero.dll*" puede ser invocada desde una aplicación diferente a la original, y a través de procesos delegados por la API, la otra aplicación utilizará de forma transparente los servicios e la API a través de sus clases comunes.

6.1.2.3. Reducción de Acoplamientos (Reducing Coupling)

- **Encapsulación**: es una técnica utilizada por el paradigma Orientado a Objetos para definir un conjunto de datos y modos de acceso a su estructura a partir de un conjunto de especificaciones y parámetros que hacen que el acceso sea exclusivo solo a través de dicho mecanismo, garantizando la integridad de los datos que contiene un objeto. El propósito de la encapsulación es reducir la probabilidad de que los cambios introducidos a uno o más módulos no se propaguen a otros módulos que no han sido afectados por la Modificabilidad en el modelo anterior (inicial o anteriormente iterado).

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

- **Aumento en los niveles de Abstracción** (Raise the Abstraction Level): Según (OMG®, 2003) la Abstracción se define como "*…un conjunto de*

características esenciales que distinguen a una entidad de todas las demás entidades. Una abstracción define una frontera relativa a la perspectiva del observador". En pocas palabras la Abstracción trata de captar las características esenciales de un objeto, así como su comportamiento, y representarlas así mediante un modelo. Esta táctica permite el diseño de módulos genéricos que puedan ser modificados en forma sencilla sin que dichos cambios afecten o involucren cambios relevantes en el módulo.

- **Uso de Intermediarios** (Use an Intermediary): la función de un intermediario consiste en romper una dependencia entre módulos. El uso de esta táctica crea una nueva responsabilidad la cual actuará como intermediario entre el conjunto de módulos y asigna dicha responsabilidad al módulo creado. Aparte de crear el módulo intermediario y su asociación a determinada responsabilidad, la técnica elimina los fuertes acoplamientos entre los módulos originalmente analizados. Un ejemplo de intermediario se asemeja a los Plug-ins utilizados en Eclipse.

- **Restringir Rutas de Comunicación** (Restrict Communications Paths): corresponde a un caso específico de intermediario. Esta técnica permite la eliminación de una dependencia causada por necesidades específicas de comunicación y la canaliza a través de un nuevo intermediario para disminuir el acoplamiento.

6.1.3. Tácticas aplicadas al Rendimiento (Performance)

La ***Performance*** se refiere a las respuestas del sistema, ya sea el *tiempo requerido* para responder a *eventos específicos* o la *cantidad de eventos* procesados en un intervalo de tiempo dado. (Brown, y otros, 1995). La Performance normalmente está asociada en un sistema de cómputo a la cantidad de transacciones por unidad de tiempo o se estima como el tiempo que toma un sistema para completar una transacción. Las tácticas de rendimiento se encargan de controlar el tiempo que se tarda en generar una respuesta a un determinado evento.

Existen diversos factores que intervienen en el rendimiento de un sistema de cómputo, desde el consumo de recursos, comunicaciones a nivel de dispositivos, procesamiento, acceso a memorias dinámicas y físicas, algoritmos y condición de competencia entre dispositivos, disponibilidad, sincronización de procesos hasta sistemas más complejos que incluyen planificación y accesos a elevadas tasas de transferencia. La performance se asocia normalmente con el término Latencia (retardos temporales).

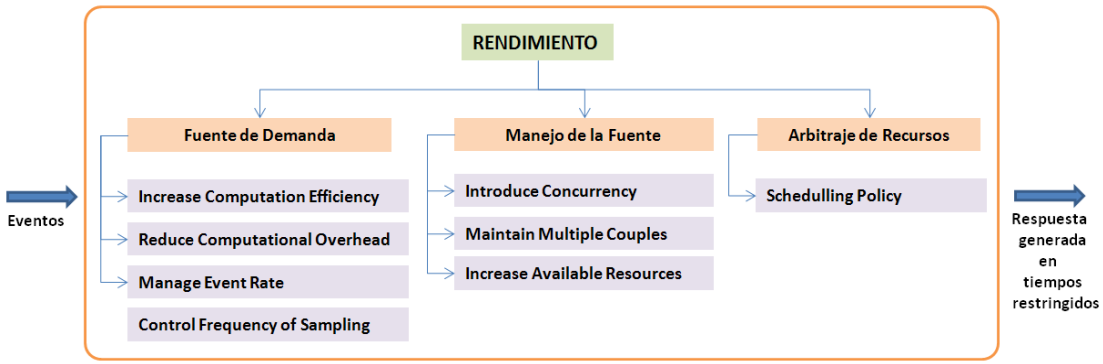
En el ámbito de ArchE, los requerimientos para el atributo de calidad Performance se obtienen a partir de un escenario. Estos escenarios reciben estímulos tanto internos como externos y se pueden clasificar como ***periódicos***, ***estocásticos*** o ***esporádicos***.

- Los estímulos **Periódicos** son aquellos que registran tiempos equivalentes tras cada intervalo de proceso;

- los estímulos **Estocásticos** responden a eventos que llegan a una determinada tasa de probabilidad y

- los estímulos **Esporádicos** que obedecen a eventos no programados o que se presentan con menor frecuencia.

Figura 11. Tácticas Arquitectónicas de Performance



6.1.3.1. Demanda de Recursos (Resource Demand)

- **Incremento de la Eficiencia Computacional** (Increase Computation Efficency): se logra mejorando los algoritmos y utilizando rutas criticas para la definición de servicios y prioridades en los componentes y su acceso al sistema de memorias y microprocesadores.

- **Reducir Sobrecargas Computacionales** (Reduce Computational Overhead): la idea central en este tipo de procedimientos se orienta hacia la gestión de hebras o hilos. Los hilos a diferencia de los procesos se configuran a través de un conjunto de estados e interactúan con el sistema operativo a través de mecanismos de comunicación. El Overhead hace referencia al tiempo desperdiciado por el microprocesador al cambiar de estado en (*ejecución)* al estado de espera (*waiting*) y colocar el nuevo proceso en ejecución. Los hilos se pueden sincronizar entre sí para garantizar condiciones de competencia equitativa entre dispositivos y compartimiento de recursos.

- **Manejo de Eventos** (Manage Event Rate): consiste en manejar unos niveles de Latencia aceptables para disminuir el número de eventos procesados.

- **Control para Frecuencias de Muestreo** (Control Frequency Sampling): consiste en el manejo de colas que permitan la gestión y control de eventos externos.

Otros mecanismos para la Demanda de Recursos pueden incluir:

- Reducción de intermediarios para mecanismos de comunicación,
- Reducción de requerimientos y funcionalidades en el esquema de comunicaciones,
- Splitting o particionamieno de funcionalidades.

6.1.3.2. Manejo de Recursos (Resource Managment)

- **Introducción de Concurrencias** (Introduce Concurrency): se logra creando múltiples procesos para que el grupo de eventos sean manejados no solo por un proceso sino por varios en forma paralela, para reducir tiempos de bloqueo entre procesos. La concurrencia comprende un gran número de actividades que inclyen: comunicación entre procesos, compartición y competencia por recursos, sincronización, ejecución simultánea de procesos y asignación del tiempo de procesador a los mismos.

- **Parejas Múltiples** (Maintain Multiples Couples): esta táctica se obtiene al realizar copias exactas de datos y procesos en memorias extendidas o de tipo Caché. Estas réplicas reducen sustancialmente el tiempo de procesamiento y distribuye uniformemente los tiempos administrando mejor el acceso y la disputa en los recursos del sistema. Esta táctica requiere de una constante sincronización entre parejas para asegurar la consistencia en los datos tanto de las fuentes como de las replicas.

- **Incrementar la Disponibilidad de Recursos** (Increase Available Resources): se logra implementando procesadores con alta capacidad de procesamiento, memorias con mayor capacidad y redes de alta velocidad para reducir el factor de Latencia.

Otros mecanismos para el manejo de recursos pueden incluir:

- Flexibilidad de asignación de hardware,
- Mecanismos para monitorización interna y externa para el producto software,
- Sistemas de programación en tiempo real,
- Herramientas software para gestionar la performance.

6.1.3.3. Arbitraje de Recursos (Resorurce Arbitration)

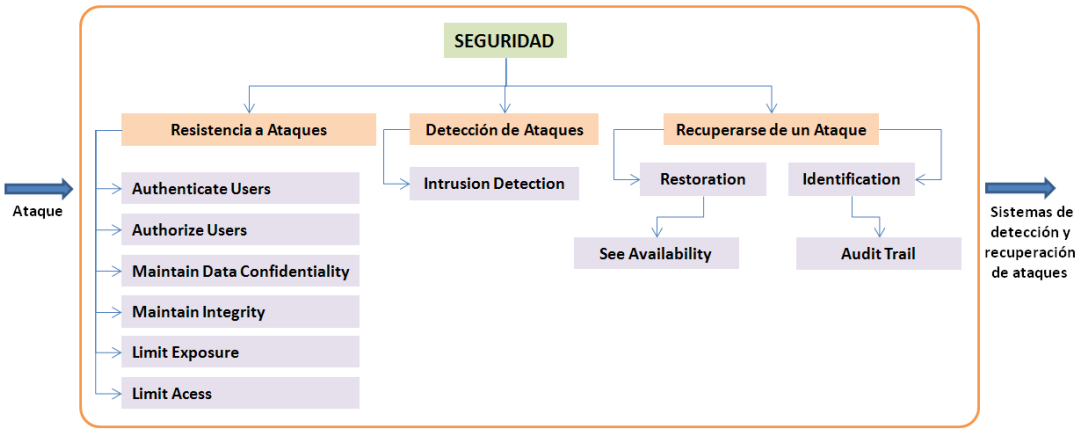
- **Políticas de Planificación** (Schedulling Policy): las políticas de planificación hacen referencia al conjunto de medios o mecanismos que son incorporados a un sistema para ejecutar un conjunto de instrucciones o procesos los cuales compiten por un tiempo de ejecución. Un sistema de planificación debe poseer un planificador, el cual se encarga de decidir cuál de todos los procesos se encuentra en condiciones para ser ejecutado y cuál de todos debe ser atendido de primeras o de forma simultánea para aprovechar al máximo los recursos.

  Dentro de las políticas de planificación más comunes encontramos: algoritmos FIFO, mecanismos de prioridad dinámica (Round Robin y Earliest Deadline First), estrategias para planificación autónoma o estática y mecanismos para gestión de prioridades fijas (Max Time Execution y sistemas de periodicidad); los cuales son aplicados bajo un esquema de orden y prioridades.

6.1.4. Tácticas aplicadas a la Seguridad (Security)

Las tácticas de seguridad se orientan hacia la protección e integridad de los datos y los componentes del sistema. La integridad permite manejar los datos y los sistemas bajo el esquema de la coherencia. De acuerdo a su funcionalidad, las tácticas de seguridad pueden clasificarse como: mecanismos para restricción de ataques, detección de ataques y recuperación de ataques. En la actualidad los ingenieros de software se centran mucho en la mejora de las capacidades en función de la seguridad, es importante en este sentido que el diseño de estas estrategias estén bien formadas y fundamentadas para no afectar otro tipo de atributos en el software como la Performance, la Usabilidad, entre otros (en algunos casos la seguridad hace que los sistemas sean engorrosos, poco flexibles ya que se requieren de demasiadas interacciones y restricciones).

Figura 12. Tácticas Arquitectónicas de Seguridad



6.1.4.1. Resistencia a Ataques (Resisting Attacks)

- **Autenticación de Usuarios** (Authenticate Users): son tácticas que garantizan que un usuario o un servicio remoto tenga acceso a las funciones y descripciones del sistema a través de un conjunto de reglas de identificación y verificación de accesos. Estos mecanismos normalmente están representados por contraseñas, sistemas biométricos, certificados o firmas digitales y sistemas de cifrado acompañados por parámetros de acceso o niveles tipo Grant (Acceso a Niveles de Seguridad y parámetros de Usuario).

- **Autorización de Usuarios** (Authorize Users): son mecanismos que garantizan la prestación de servicios en una aplicación o sistema previa revisión de derechos y credenciales de autenticación. Estos mecanismos garantizan a los usuarios el acceso y modificación de la información con base en los roles especificados para el mismo. Estos mecanismos se pueden implementar a base de listas de usuarios, roles del sistema, grupos de trabajo, dominios, etc.

- **Confidencialidad** (Maintain Data Confidentiality): estos mecanismos de confidencialidad se logran a través de la implementación de controles de Cifrado (DES, 3DES, AES) o algoritmos de cifrado (simétricos, cuánticos, asimétricos, de curva elíptica o híbridos) para la protección de los datos y las comunicaciones en una red de datos. La misión de estas tácticas es proteger la información de usuarios no autorizados.

- **Integridad** (Maintain Integrity): el objetivo de este tipo de tácticas es proveer la información a un destino tal y como ha sido enviada por la fuente, sin que el mensaje sufra alguna modificación por parte de servicios o usuarios no autorizados. Por lo general estos mecanismos están acompañados de códigos de comprobación o redundancia CRC[14] (Cyclic Redundancy Check). Las implementaciones más comunes son los mecanismos de Checksum y códigos Hash "un tipo de estructura de datos que asocia *llaves* o *claves* con *valores*."

- **Limitación/Restricción de Servicios** (Limit Exposure): esta táctica sugiere la implementación de servicios única y exclusivamente en equipos o sistemas de cómputo que requieran del componente como tal, conservando los esquemas de seguridad y acceso a dichos servicios. Ejemplo de ello, la implementación de un servicio de Antivirus en un Servidor (desde donde se actualizan las listas de definición de virus, spyware y malware) y en el cual se ha instalado el software completo de gestión para detección de amenazas; y el acceso a través de Clientes los cuales se instalan en los host de la red y

[14] La **Comprobación de Redundancia Cíclica** (CRC) es un código de detección de errores usado frecuentemente en redes digitales y en dispositivos de almacenamiento para detectar cambios accidentales en los datos, también es aplicado a los sistemas software para verificar la integridad de los datos.

los cuales solo acceden a ciertos servicios del antivirus instalado en el servidor. En cierta medida, los parámetros de configuración del Cliente pueden tener injerencia en procesos como instalación y desinstalación de aplicaciones, lo que se convierte en una forma eficiente para controlar este tipo de vulneraciones o cambios no autorizados al sistema.

- **Limitar Acceso** (Limit Access): por lo general, este tipo de tácticas se implementan a través de Firewalls y Zonas Desmilitarizadas o DMZ. Una DMZ es un diseño conceptual de red donde los servidores de acceso público se colocan en un segmento separado, aislado de la red. La intención de DMZ es asegurar que los servidores de acceso público no puedan comunicarse con otros segmentos de la red interna, en el caso de que un servidor se encuentre comprometido. (TP-Link, 2013). Un Firewall o Cortafuegos es un mecanismo que hace parte de un sistema (a nivel de software o hardware y redes) para bloquear el acceso no autorizado, permitiendo al mismo tiempo comunicaciones autorizadas.

6.1.4.2. Detección de Ataques (Detecting Attacks)

- **Detección de Intrusos** (Intrusion Detection): la función de estos mecanismos consiste en detectar posibles intrusiones al sistema. Estos mecanismos comparan los patrones de tráfico de una red, los paquetes o tramas de mensajes y direcciones MAC. Las tácticas para detección de intrusos deben poseer mecanismos de filtrado que trabajen sobre el protocolo, reconocimiento de direcciones físicas, control de puertos y tamaños de tramas transmitidas a través de la red de datos.

6.1.4.3. Recuperación de Ataques (Recovering from an Attack)

- **Restauración del Sistema** (Restoration): este tipo de técnicas se focalizan en la recuperación del sistema ante cualquier suceso inesperado. Para ello se hace necesario que el arquitecto mantenga copias redundantes de los datos del sistema (listas, claves, perfiles de acceso, dominios, servicios, etc) para poder recuperar con facilidad el sistema, cuando se presente un ataque.

- **Identificación de Atacantes** (Identification): estas tácticas se orientan a la auditoría y control de sucesos en un sistema. En la informática forense, este tipo de controles son el mecanismo más eficiente para recuperar eventos, buscar responsables y verificar el estatus y errores generados por el acceso a los servicios y datos de una aplicación o sistema. Este tipo de tácticas sugiere al arquitecto copias de respaldo de cada transacción aplicada a los datos junto con el registro de logueo o identificación de quien realizó el proceso.

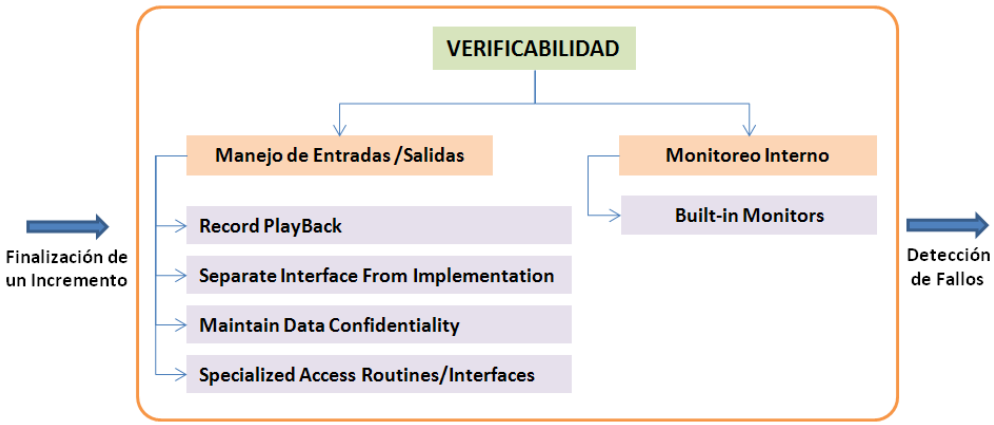
Otros mecanismos para el manejo de recursos pueden incluir:

- Tunneling y configuración de VPN o protocolos SSL (Secure Sockets Layer) y SSH (Secure Shell),
- Uso de patrones para identificación de tráfico entre una red y un Sistema de Gestión de Base de Datos,
- Documentación de ataques (históricos) y uso de patrones de corrección,
- Sistemas de Sensorica,
- Copias redundantes,
- Monitores de Red y Registro de Eventos,
- Kernels (núcleos) y shells (capas) de seguridad

6.1.5. Tácticas aplicadas a la Verificabilidad (Testability)

El objetivo de las tácticas consiste en facilitar las pruebas sobre componentes y módulos del sistema. Las dos categorías en las que se diferencian estas técnicas son; Controles de Entrada/Salida y Monitorización interna de procesos. El principal objetivo de las tácticas de testeabilidad es descubrir fallos a nivel general o en alguno de los módulos de la herramienta software.

Figura 13. Tácticas Arquitectónicas de Verificabilidad



6.1.5.1. Manejo de Entradas/Salidas (Manage Input/Output)

- **Grabación y Retroceso** (Record PlayBack): es una técnica que consiste en almacenar el estado de un componente, que lo haga predictivo tanto para la entrada como para la salida de otros sistemas. El comportamiento del componente es comparado con los resultados obtenidos tras la prueba y si devuelve los valores almacenados en el repositorio de pruebas, el módulo satisface los requerimientos operacionales.

- **Separación de interfaces** (Separate Interfaces from Implementation): esta táctica consiste en separar los módulos de la implementación global del sistema para probarse por aparte como componentes y posterior a ello probarlos en conjunto. La separación de interfaces hace que las pruebas sean mucho más flexibles y se puedan comprobar con mayor eficiencia las unidades de software.

- **Accesos Especializados** (Specialized Access Routines Interfaces): esta táctica consiste en la definición de niveles o jerarquías para pruebas, de tal forma que los resultados obtenidos por cada capa o nivel de pruebas, pueda ser utilizado por otras para su posterior evaluación y control de variables.

6.1.5.2. Monitorización Interna (Internal Monitoring)

- (Built-in Monitors): esta táctica consiste en utilizar componentes o software especializado para monitorear el comportamiento de los componentes durante su ejecución.

Otros mecanismos para el manejo de recursos pueden incluir:

- Mecanismos para monitoreo interno, captura, registro y reporte de eventos,
- Herramientas de simulación,
- Herramientas de testeo de aplicaciones,
- Sistemas para Debugging (Depuración),
- Desarrollo paso a paso (incremental),
- Separación de Interfaces.

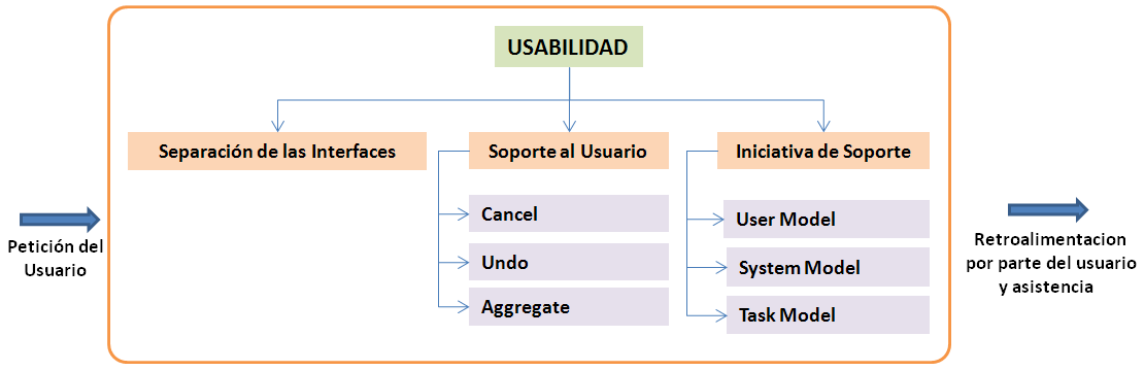
6.1.6. Tácticas aplicadas a la Usabilidad (Usability)

Las tácticas de usabilidad se enfocan en facilitarle al usuario la interacción con el sistema. Entre más intuitiva sea la herramienta, más fácil será su utilización. En esta técnica se agrupan dos grupos de estrategias: tácticas en tiempo de diseño y tácticas en tiempo de ejecución.

Los aspectos más importantes de la usabilidad son;

- El fácil aprendizaje,
- La Eficiencia producto de la facilidad de su utilización,
- Facilidad de Memorización.
- Interfaces amigables, intuitivas, enriquecidas con textos, gráficos y multimedia (en algunos casos).

Figura 14. Tácticas Arquitectónicas de Usabilidad



6.1.6.1. Tácticas en Tiempo de Ejecución

- **Tácticas de tiempo de ejecución**: este conjunto de tácticas ayudan al usuario a realizar sus actividades de forma ágil, eficiente y con una gran facilidad. Estas tácticas agrupan a todos aquellos mensajes, advertencias, cuadros de diálogo que informan al usuario acerca del estado de su petición u operación en el sistema. Hacer que tareas como copiar, pegar, cancelar, rehacer cambios, etc., sea una tarea fácil al alcance de cualquier usuario con pocos o escasos conceptos en programación y manejo de herramientas específicas.

Otros mecanismos para el manejo de tácticas en tiempo de ejecución incluyen:

- Vistas de Diseño,
- Control de sintaxis (corrección automática de errores léxicos y sintácticos) Spelling Check,
- Scrolling y control de eventos del mouse,
- Informe de eventos (tiempo de copia, transferencias de archivos, estimación de tiempo de descarga, etc.).

6.1.6.2. Tácticas en Tiempo de Diseño

- **Tácticas de tiempo de diseño**: este tipo de tácticas utilizan al patrón de Arquitectura MVC "Modelo-Vista-Controlador" para separar las interfaces del sistema. El patrón MVC, es una herramienta eficaz para asegurar de igual manera el atributo de calidad de coherencia semántica para el atributo de modificabilidad.

Otros mecanismos para el manejo de tácticas en tiempo de diseño incluyen:

- Implementación de Patrones de diseño separando capas de la Arquitectura (Presentation-Abstraction-Control (PAC),
- Interfaces Gráficas,
- Implementación de Frameworks
- Utilización de UI's (interfaces de usuario) comunes.

6.2. PATRONES ARQUITECTÓNICOS

Los patrones de diseño representan decisiones de diseño potenciales para mejorar los atributos de calidad del sistema. Los **patrones arquitectónicos**, o **patrones de arquitectura** ofrecen una descripción de los elementos, restricciones, interacciones y responsabilidades de un sistema y la forma en cómo éstos pueden ser usados en el diseño de modelos y arquitecturas de software. Un patrón arquitectónico responde a un esquema estructural de un sistema con sus funciones o subsistemas.

> Según (Buschmann, Meunier, Rohnert, Sommerland, & Stal, 1996) *"... **A pattern for software architecture** describes a particular recurring design problem that arises in specific design contexts, and presents a well-proven generic scheme for its solution. The solution scheme is specified by describing its constituent components, their responsibilities and relationships, and the ways in which they collaborate..."*
>
> (Buschmann, Meunier, Rohnert, Sommerland, & Stal, 1996), *"... **Un Patrón para la Arquitectura de Software** describe un problema particular de diseño que se presenta en un contexto específico a través de un conjunto de elementos genéricos probados y bien formados para la solución de un problema. El esquema final o solución está descrita por sus componentes, responsabilidades, relaciones y las interacciones mediante las cuáles éstos colaboran entre sí..."*

Según (Giraldo, Acevedo, & Moreno, 2011) un patrón de diseño es una abstracción de una solución en un nivel alto. Los patrones solucionan problemas que existen en muchos niveles de abstracción. Hay patrones que abarcan las distintas etapas del desarrollo.

Los Patrones de Diseño (Design Patterns) son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces. (Merlino, Vranic, Rodríguez, Pytel, & García-Martínez, 2009).

Los patrones de diseño pretenden:

a) proporcionar catálogos de elementos reusables en el diseño de sistemas software,

b) evitar la reiteración en la búsqueda de soluciones a problemas ya conocidos y solucionados anteriormente,

c) formalizar un vocabulario común entre diseñadores de software,

d) estandarizar el modo en que se realiza el diseño de software y

e) facilitar el aprendizaje de las nuevas generaciones de diseñadores de software condensando conocimiento ya existente.

Los Patrones arquitectónicos se dividen en 4 categorías: **From Mud to Structure** (Del lodo a la estructura), (Múltiples Dimensiones a las Estructuras), **Distributed Systems** (Sistemas Distribuidos), **Interactive Systems** (Sistemas Interactivos) y **Adaptable Systems** (Sistemas Adaptables).

6.2.1. From Mud To Structure

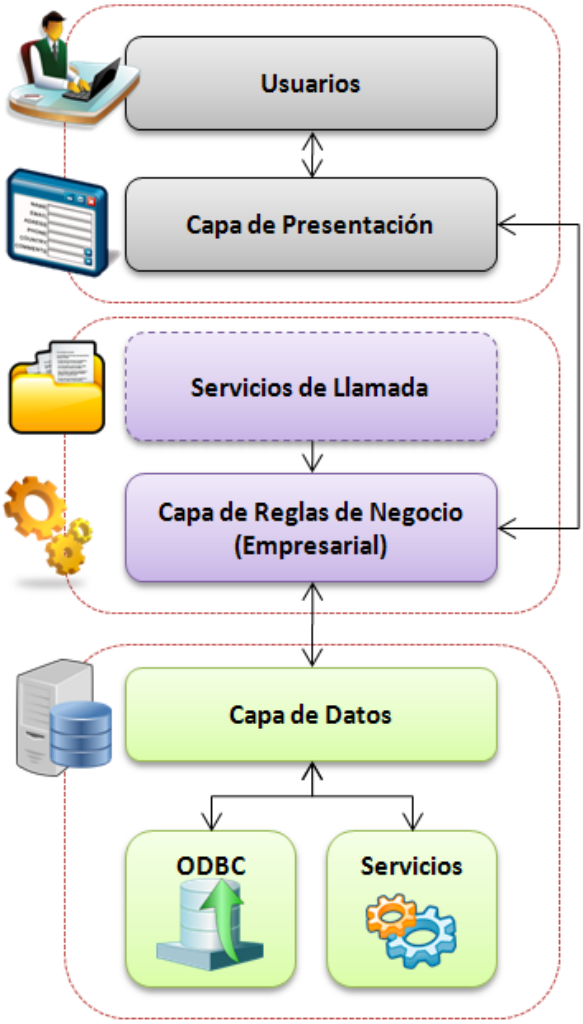
From Mud to Structure (Del lodo a la estructura), (Múltiples Dimensiones a las Estructuras): La traducción literal al idioma español no es fácil de acuñar. Se dice que este tipo de patrones están diseñados para evitar un "mar" de componentes u objetos. En otras palabras este tipo de patrones ayudan a controlar los niveles de descomposición para evitar la sobrecarga de clases en un modelo y las relaciones de cooperación entre éstos. Esta categoría incluye las tuberías y filtros "Pipes and Filters", los patrones de capas "Layers" y el patrón de tablero "Blackboard".

6.2.1.1. Capas (Layers)

Un Patrón por Capas o Layers permite la estructuración de aplicaciones en base a tareas o responsabilidades. Cada tarea corresponde a una descomposición dentro de un grupo de sub-procesos que corresponde a un determinado nivel de abstracción.

Este patrón describe sus funcionalidades en base a diagramas de bloques (por ejemplo como la pila de protocolos de OSI y TCP/IP). Este patrón arquitectónico propone el fraccionamiento de responsabilidades a través de unidades o estructuras lógicas. Este tipo de arquitectura se caracteriza por que las capas adyacentes en cada uno de los bloques corresponde entre sí a un conjunto de relaciones y responsabilidades que son compartidas entre si y que sirven como soporte de servicios entre las capas adyacentes. El éxito de esta arquitectura radica en la técnica del Top-Down, lo que minimiza los altos acoplamientos entre capas inferiores y capas superiores, ya que cada capa adyacente se encarga de entregar el servicio requerido por la sub-capa o capa contigua.

Figura 15. Arquitectura por Capas



CAPA DE PRESENTACIÓN

Representa a la interacción entre el usuario y el software. Puede estar contenido por formularios, interfaces, menús, líneas de comando o GUI's. Su principal responsabilidad es mostrar información al usuario, interpretar los comandos y realizar algunas validaciones simples de los datos ingresados.

CAPA DE REGLAS DE NEGOCIO (EMPRESARIAL)

También denominada Lógica de Dominio, contiene la funcionalidad que implementa la aplicación. Involucra cálculos basados en la información almacenada y en algunas validaciones. Controla la ejecución de la capa de acceso a datos y servicios externos. La lógica de negocios se puede modelar "encapsular" a partir de componentes o servicios y a través de llamados de otras aplicaciones o interfaces. Es una tarea básica en los ambientes Cliente/servidor.

CAPA DE DATOS

Esta capa contiene la lógica de comunicación con otros sistemas que llevan a cabo tareas por la aplicación. Estos pueden ser monitores transaccionales, otras aplicaciones, sistemas de mensajerías, etc. Para el caso de aplicaciones empresariales, generalmente está representado por una base de datos, que es responsable por el almacenamiento persistente de información. Esta capa debe abstraer completamente a las capas superiores (negocio) del dialecto utilizado para comunicarse con los repositorios de datos (PL/SQL, Transact-SQL, SQlite, Informix, DBase, etc.).

La arquitectura por capas define una estructura de bajo nivel y otra de alto nivel. Las estructuras de bajo nivel por lo general se asocial a datos o componentes de bajo nivel, mientras que las capas superiores hacen parte del modelo de aplicación. Como ejemplos de implementación de este patrón encontramos a las máquinas virtuales como JVM, Las API's y los Frameworks de desarrollo como Net Framework de Microsoft.

Una máquina virtual aísla los niveles más altos de una arquitectura y los separa de la arquitectura física (hardware). En esencia una máquina virtual permite ejecutar un sistema y simular sus procesos como si estuviese totalmente instalado en la máquina. Un ejemplo de ellos es el Virtual Box de Oracle que permite instalar más de un sistema operativo en un host sin traslapar sus núcleos y funcionalidades y la máquina virtual de Java que es una máquina virtual de procesos que se ejecuta en código nativo ByteCode y que es independiente de la arquitectura de la máquina.

Otro uso común de este patrón corresponde al uso de API's y Frameworks. Ambas corresponden a desarrollos por capas y su funcionalidad consiste en encapsular un conjunto de sub-capas o subprocesos que pueden ser utilizados en forma recurrente por un sistema o servicio. Este conjunto de funcionalidades por lo general contienen colecciones de bibliotecas o ensamblados para llamados a funciones del sistema operativo o de una determinada aplicación. Estas librerías proporcionan una alta **Portabilidad** entre los distintos sistemas operativos y provee servicios adicionales, aparte de que contiene a un conjunto de rutinas probadas que pueden ser reutilizadas en otros desarrollos.

Otra aplicación común de este patrón lo componen los sistemas de Información, en especial los sistemas orientados al dominio de negocios, los cuales están compuestos por bases de datos, capa de aplicación, capa de aplicación, servicios de dominio, entre otros servicios. Una de las características fundamentales de este patrón, es que su implementación permite una mayor escalabilidad o estandarización de servicios.

Este patrón implementa las siguientes tácticas

- **Coherencia Semántica**: esta táctica la realiza a través de la distribución de responsabilidades en capas adyacentes, generando bajos acoplamientos entre capas superiores e inferiores.

- **Mayor nivel de Abstracción**: si bien es cierto que el modelo está representado por un conjunto de Capas que no netamente Abstracciones, los servicios igualmente corresponden a implementaciones abstractas y un conjunto de responsabilidades de acuerdo al nivel y la adyacencia para cada una de las capas.

- **Servicios Comunes**: esta táctica es normalmente empleada a nivel de las API y los marcos de desarrollo como .NET; e este contexto, las responsabilidades se agrupan en servicios para que éstos puedan ser accedidos desde cualquier punto en cualquier momento aumentando el factor de **Disponibilidad** y **Flexibilidad** de las aplicaciones.

- **Encapsulación**: la encapsulación se hace en base a dos esquemas; de tipo público y de tipo privado. Las responsabilidades de tipo público son visibles y son compartidas con el exterior mientras que las privadas solo pueden ser accesadas por servicios específicos siempre y cuando estos posean parámetros de modificabilidad sin que estos cambios afecten la integridad del componente o sistema.

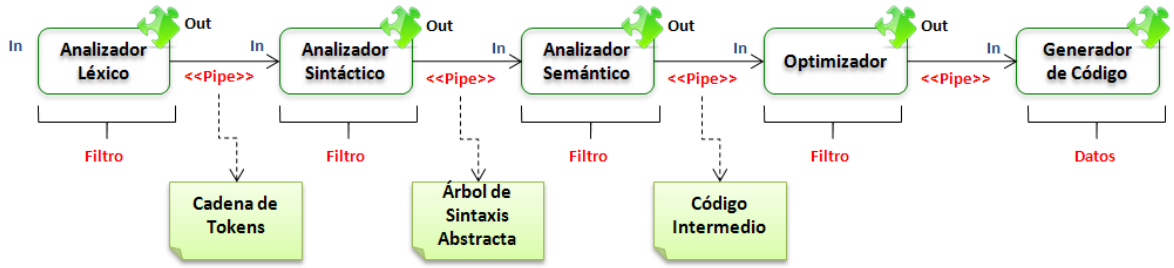
- **Servicios Intermediarios** (Brokers): esta táctica consiste en delegar un conjunto de responsabilidades a un intermediario y que a la vez sirva éste como puente entre una capa y otra.

6.2.1.2. Tuberías y Filtros (Pipes and Filters)

Pipes and Filters (Tuberías y Filtros) es una patrón arquitectónico que procesa una corriente de entrada en un sistema a través de un conjunto de sub-sistemas denominados Filtros. Cada sub-proceso o filtro corresponde a un mecanismo de encapsulación de un componente. Cada dato procesado por cada componente es pasado al componente adyacente a través de conductos o Pipes, para continuar con el proceso. En esta arquitectura cada filtro corresponde a un nivel particular de abstracción.

Este tipo de patrón es ideal cuando se diseñan sistemas complejos que requieren ciertos procesos de refinamiento o cuando los procesos se dan en forma incremental. Esta arquitectura divide una cantidad de procesos en tareas más pequeñas y que responden a cierta secuencia. Una de las aplicaciones más difundidas para esta arquitectura, corresponde a la construcción de Compiladores. Un esquema general de esta arquitectura se muestra a continuación.

Figura 16. Arquitectura Pipes and Fliters



El funcionamiento de la arquitectura es muy simple; cada filtro expone su interfaz, ésta recibe los mensajes de su filtro adyacente a través de su conexión Pipe y procesa el mensaje y posteriormente entrega los resultados del proceso al Pipe adyacente. El Pipe se encarga de conectar el filtro adyacente enviando una respuesta hasta que se procesa el último filtro. La salida corresponderá a la suma de todos los procesos en una sola salida. La conexión entre filtros y tuberías a menudo es denominada Puerto; es decir cada componente o filtro posee un puerto de entrada y otro puerto de salida.

El patrón Pipes and Filters implementa las siguientes tácticas:

- **Coherencia Semántica**: esta táctica define los filtros como unidades o componentes. Cada filtro posee mecanismos para procesar, refinar y transformar información que es compartida con su filtro adyacente. El factor de coherencia se extiende al conjunto de responsabilidades delegadas en torno a su utilización y especificación. Aquí cada responsabilidad opera de forma independiente.

- **Flexibilidad**: al poseer una interfaz simple, los filtros suelen intercambiar información de forma sencilla a través de un conjunto de tuberías o PipeLine. El intercambio de Filtros no es posible en tiempo de ejecución debido a que el proceso del patrón es incremental.

- **Encapsulación**: este patrón está fundamentado en la propiedad de encapsulación, manteniendo las características de tipo privado y públicas sobre los datos y sus funcionalidades.

- **Restricción de Comunicación**: al contener cada filtro un puerto de entrada y otro de salida, la comunicación se restringe y prácticamente las dependencias se restringen en el orden de 1 a 1 entre filtros lo que genera un bajo acoplamiento.

- **Uso de Intermediarios**: las tuberías pueden romper algunas dependencias entre filtros adyacentes mediante procesos de sincronización o con el uso de memorias intermedias (buffering).

- **Reusabilidad**: esta táctica se puede ejecutar a partir de los procesos de intercambio entre filtros, lo que maximiza el reuso de los componentes. Esta táctica permite la creación de nuevas tuberías de procesos, reestructurando los filtros existentes y agregando nuevos componentes.

6.2.1.3. Tablero (Blackboard)

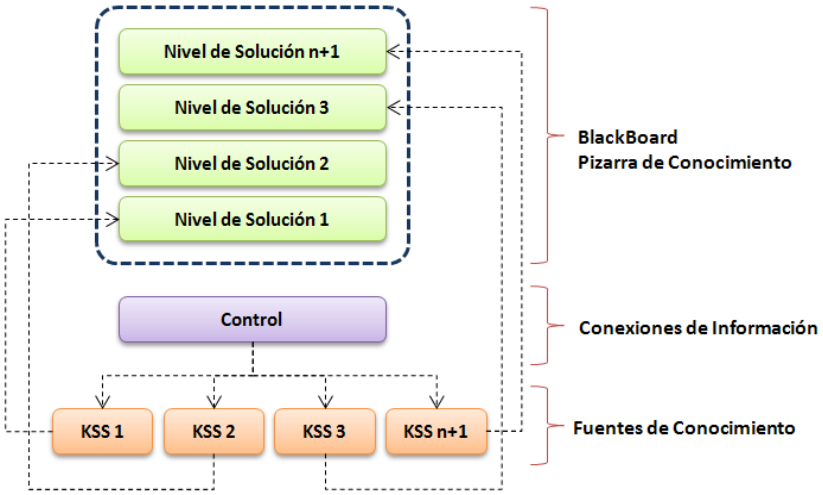
BlackBoard es un patrón arquitectónico que puede ser utilizado para la resolver problemas de tipo No Determinísticos con base en estrategias de solución ya conocidas. El sistema de divide en tres componentes, una llamada Blackboard "Pizarra", una colección de fuentes de conocimiento y un componente llamado control.

- **Blackboard**: contiene la estructura de datos (Vocabulario o Semántica) sobre la cual trabajarán los módulos cooperantes. Este componente alberga las reglas y los datos de control del sistema. Además de almacenar las reglas de negocio, administra las interfaces que permiten a los demás módulos leer, escribir e interactuar con la base de conocimiento. Cada vez que se procesa una salida (conocimiento) éste es considerada como solución y son puestos en escena al componente Blackboard como una solución hipotética o de Hipótesis.

- **Knowledge Sources KSS** o fuentes de conocimiento: corresponde a un conjunto de módulos que alberga un conocimiento "Experto" que aporta en determinado momento una solución en particular. El KSS es quien se encarga de escribir, leer y modificar las reglas o conocimiento en el componente BlackBoard.

- El componente **Control** es un elemento que se encarga de monitorear el comportamiento y los cambios realizados en el módulo de BlackBoard. La tarea de este mecanismo es planificar y evaluar en forma redundante las KSS de acuerdo a una estrategia de conocimiento planteada. Las responsabilidades de este componente se orientan a la estimación de las heurísticas, cálculo de costos computacionales por la ejecución de las KSS y la coordinación de los demás agentes cooperadores "Expertos". El resultado de su proceso a menudo se denomina "datos de control" y son puestos en el blackboard.

Figura 17. Arquitectura BlackBoard



La arquitectura Blackboard está compuesta por un conjunto de programas independientes que trabajan cooperativamente en una estructura de datos común. Cada uno de estos programas es especialista en resolver una parte específica de la tarea completa, y todos los programas trabajan juntos en la obtención de la solución. Se considera una arquitectura basada en datos (Driven Data). (Quian, Fu, Tao, Wei Xu, & Diaz-Herrera, 2009).

Unas de las estrategias utilizadas por este patrón es la construcción de conocimiento a partir de sus sub-sistemas quienes ensamblan el conocimiento para construir posibles soluciones, aproximaciones o soluciones parciales a un problema determinado (Una solución parcial puede requerir de diferentes interpretaciones y paradigmas). En un sistema No Determinísticos no existen soluciones predeterminadas ni se obtienen soluciones reales. Las soluciones en este tipo de sistemas son parciales y la optimalidad de su solución depende en parte de procesos de transformación y refinamiento.

Tras cada proceso de transformación se generan diversas soluciones o alternativas. En este orden de ideas es muy factible encontrar soluciones óptimas para una parte o para la mayor parte de los casos. En contraposición a una solución optimizada, también se puede

encontrar con soluciones parciales o en el peor de los casos no encontrar solución alguna. Esto depende del grado de **Heurísticas** que maneje la arquitectura.

El patrón BlackBoard proviene de la disciplina de la Inteligencia Artificial. Este modelo combina el conocimiento procesado por cada uno de sus componentes y produce una sola salida a partir de ese conjunto de reglas (sin importar si son o no estrategias optimizadas). La idea de utilizar este patrón es poder aprovechar la capacidad de cooperación entre componentes los cuales trabajan de forma independiente y sobre las mismas estructuras de datos. A través del elemento de Control, la arquitectura se encarga de ensamblar y evaluar el estado de procesamiento del conocimiento y se encarga de coordinar la comunicación e interacción entre los componentes o programas especializados.

Una de las áreas donde se emplea con mayor frecuencia este patrón, es en los sistemas de reconocimiento de voz o **Speech Recognition**. En este tipo de sistemas por ejemplo, un procedimiento divide la forma de onda e interpreta los segmentos de la misma y que son significativos "prioritarios" en el contexto del reconocimiento del patrón de voz. Al final de la secuencia de procesos, otro procedimiento comprueba la sintaxis o frases candidatas, y al final se produce una salida o patrón de reconocimiento. En este caso ambos procedimientos trabajan en dominios diferentes. Otras aplicaciones comunes incluyen; sistemas de predicción meteorológica, sistemas de autenticación y seguridad

El patrón Blackboard también puede representa las KSS como un conjunto de reglas, que son empleadas por lenguajes de alto nivel. Esta variante de representación resulta de aplicar mayores niveles de abstracción y la implementación de servicios comunes abstractos y su respectivo enlace en tiempo de ejecución.

Este patrón arquitectónico implementa las siguientes tácticas:

- **Escalabilidad**: fácil de agregar o actualizar fuentes de conocimiento. Este atributo hace parte del atributo de calidad de Modificabilidad.

- **Verificabilidad o Testeabilidad**: las hipótesis hacen parte del proceso de solución.

- **Concurrencia**: Todas las fuentes de conocimiento pueden trabajar en paralelo dado que son independientes entre sí.

- **Performance o Rendimiento**: al utilizar diversas fuentes de conocimiento, el sistema aumenta el performance disminuyendo el overead "Latencia" computacional. A pesar de que implementa esta estrategia, es difícil establecer buenos controles para la gestión de esta estrategia. Esta táctica no es soportada por el Paralelismo.

- **Reusabilidad** de los agentes fuentes de conocimiento.

- **Coherencia semántica**: la coherencia semántica agrupa las responsabilidades dentro de una fuente de conocimiento a partir de un conjunto de componentes "Expertos".

- **Intermediarios**: las KSS no se comunican entre sí, pero emplean intermediarios en forma de BlackBoards para su comunicación.

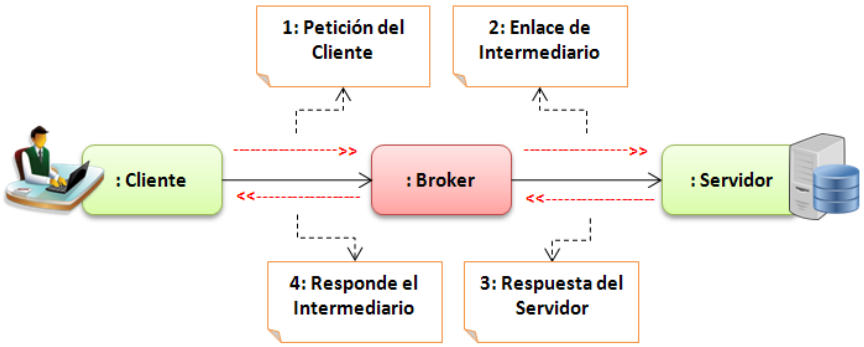
6.2.2. Sistemas Distribuidos (Distributed Systems)

Distributed Systems (Sistemas Distribuidos): esta categoría incluye al patrón Intermediario "Broker". Este patrón proporciona una infraestructura completa para aplicaciones distribuidas. En la actualidad, plataformas como Microsoft OLE (Object Linking and Embedding) y la arquitectura CORBA (**C**ommon **O**bject **R**equest **B**roker **A**rchitecture) de OMG comparten un núcleo común de arquitectura basada en el patrón Broker.

6.2.2.1. Intermediarios (Broker)

El patrón Intermediario "Broker" es una arquitectura aplicada a sistemas distribuidos. Este patrón utiliza un conjunto de componentes desacoplados que pueden interactuar entre sí a través de servicios remotos o invocación de los mismos. Un Intermediario es un mecanismo responsable de coordinar las comunicaciones entre componentes (peticiones, respuestas, transmisión y control de excepciones).

Figura 18. Patrón Intermediario (Broker)



CORBA (**C**ommon **O**bject **R**equest **B**roker **A**rchitecture) está definido como una tecnología distribuida para sistemas heterogéneos. En esta arquitectura se encuentra presente un lenguaje común para soportar los procesos de interoperabilidad entre los objetos del Cliente y los objetos del Servidor.

Microsoft **OLE 2.x** es una tecnología que proporciona un conjunto de protocolos y estándares para la exposición y acceso a interfaces de Servidor. La Internet es uno de los ejemplos que hace uso de este patrón arquitectónico. La WWW utiliza sistemas de Hipertexto, y preprocesadores que actúan como intermediarios entre los servicios web y las peticiones de los Clientes o Host.

Este patrón arquitectónico implementa las siguientes tácticas:

- **Modificabilidad**: con la aplicación de esta táctica, la arquitectura logra cierta transparencia derivado del fraccionamiento de responsabilidades a través de componentes dispersos. Cualquier sugerencia o modificación a uno de los componentes o protocolos, no altera el funcionamiento del sistema en general.

- **Portabilidad**: esta táctica se logra con la implementación del bróker como tal combinando los servicios con Proxys o Puentes de conexión. La portabilidad permite una alta interoperabilidad entre diferentes intermediarios.

- **Reusabilidad**: es una de las principales características ya que cualquier intermediario puede ser utilizado en una o más arquitecturas de forma separada o en conjunto.

- **Verificabilidad**: debido al fraccionamiento de componentes y funcionalidades, las pruebas y servicios de testing se realizan de forma más transparente y de fácil mantenimiento. En alguna medida, las pruebas pueden verse afectadas y se pueden convertir en un factor negativo ya que las pruebas pueden incluir muchos intermediarios.

Aunque es un patrón muy flexible, los atributos de Performance y Disponibilidad se pueden ver afectados ya que pueden restringir la eficiencia del sistema al presentar bajos niveles de tolerancia a fallos lo que puede desestabilizar el sistema en un momento determinado.

6.2.3. Sistemas Interactivos (Interactive Systems)

Interactive Systems (Sistemas Interactivos): en esta categoría se describen 2 patrones; el patrón MVC "Modelo-Vista-Controlador" y el patrón PAC "Presentation-Abstraction-Control". La particularidad de estos patrones radica en el soporte de sistemas que ofrecen para las interacciones hombre-máquina a través de herramientas software.

6.2.3.1. MVC (Modelo-Vista-Controlador)

MVC es una arquitectura que permite la construcción de aplicaciones interactivas. Es un patrón que separa los ***datos*** y la ***lógica de negocio*** de una aplicación de la ***interfaz de usuario*** y el módulo encargado de gestionar los eventos y las comunicaciones. Este patrón se basa en la reutilización y la usabilidad, factores de calidad que facilitan el desarrollo de aplicaciones versátiles, robustas y de fácil mantenimiento.

Los componentes del patrón MVC responden específicamente a ciertas funcionalidades. Dichos componentes por separado ejecutan las siguientes tareas o responsabilidades:

- Capa de **Modelo**: representa los datos y la información contenida en el sistema. Esta capa se encarga de gestionar los accesos a dicha información tanto a nivel de consultas, como de actualización, eliminación, adición y creación de accesos. Esta capa se encarga de administrar los privilegios de acceso descritos en las especificaciones de la aplicación (lógica de negocio). Su tarea es la de comunicar a la vista con el usuario desplegando la información contenida en el modelo. Los controles de acceso a la aplicación son gobernadas por el Controlador.

- Capa de **Controlador**: esta capa se encarga de responder a los eventos o llamados realizados por el usuario sobre la aplicación. La capa Controladora invoca las peticiones a la capa de Modelo cuando el usuario realiza cierta operación sobre los datos. El Controlador actúa como Broker o intermediario entre la capa de Vista y la capa de Modelo.

- Capa de **Vista**: presenta al modelo la información contenida en la capa de negocio, en un formato legible para el usuario. Este formato no es más que la interfaz GUI con la que el usuario interactúa con el sistema y representa las salidas del sistema (consultas, reportes, estadísticas, etc.).

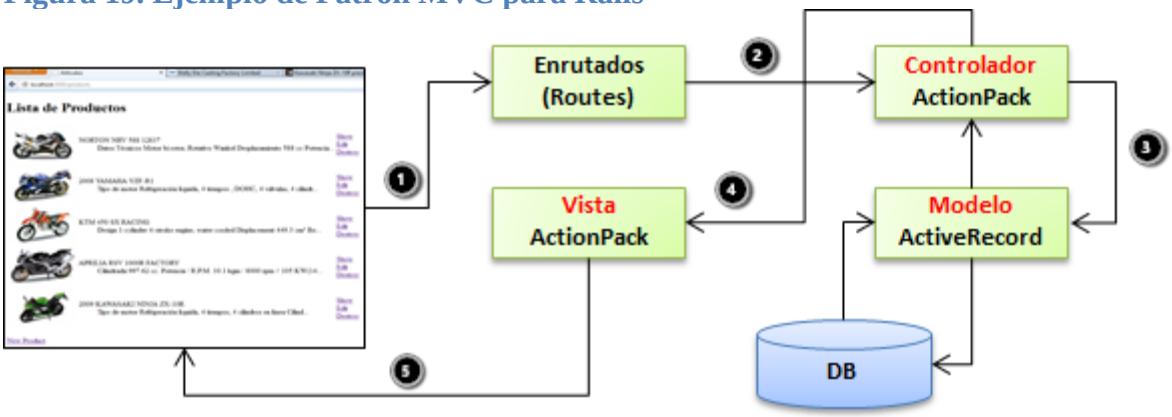
En la Fig.19 se muestra un ejemplo de implementación del patrón MVC en el lenguaje Ruby. Este lenguaje como muchos utiliza la capacidad de este patrón de diseño para modelar arquitecturas escalables y con altas prestaciones. Ruby on Rails por ejemplo; facilita el diseño y desarrollo de aplicaciones orientadas a ambientes Web separando de forma automática los componentes de una aplicación (el Modelo, la Vista y el Controlador). En Ruby on Rail o RoR como se conoce en el ámbito de los lenguajes de programación separa los componentes de la siguiente manera:

- ***Modelo*** hace referencia a las definiciones de acceso a las bases de datos y las funcionalidades descritas por el modelo de negocio.

- La ***Vista*** es la encargada de mostrar la información al usuario a través de una interfaz de formularios en rHTML (Ruby embebido en HTML) y XML.

- El ***Controlador*** comunica la Vista con el Modelo, y en resumen es el componente que alberga la lógica de programación y proporciona las funcionalidades a las vistas, y genera las validaciones necesarias para mostrar de forma adecuada los elementos de la vista.

Figura 19. Ejemplo de Patrón MVC para Rails



1. El navegador envía la petición http.
2. El sistema de enrutamiento (Routes), se encarga de enviar la petición al respectivo controlador.
3. El controlador interactúa con el modelo de base de datos producido por las Migraciones (SQLite, MySQL, Oracle, etc.).
4. El controlador llama a la vista y le devuelve la petición (respuesta http) a través de un conjunto de elementos dispuestos en formularios rHTML.
5. El navegador muestra en pantalla lo generado a través de la vista GUI.

A continuación se muestra la definición de códigos para un ejemplo programado en Rails y sus respectivas responsabilidades en cada una de las capas del modelo.

MODELO

Aut_Prg/app/models/Product.rb

```
Class Product < ActiveRecord::Base
default_scope :order=>'title'
  has_many :line_items
before_destroy :ensure_not_referenced_by_any_line_item
# ensure that there are no line items referencing this product
Def ensure_not_referenced_by_any_line_item
If line_items.count.zero?
Return true
else
        errors.add(:base, 'Existen items en la BD para desplegar')
return false
    end
```

En el modelo de producto, se ha definido en este caso que muchos productos pueden ser agregados a una carrito de compra (Relación de atributos)

```
end
attr_accessible :description, :image_url, :precio, :prod
validates :produc, :description, :image_url, :presence =>true
validates:precio, :numericality => {:greater_than_or_equal_to =>0.01}
validates:prod, :uniqueness =>true
validates:image_url, :format => {
:with=>%r{\.(gif|jpg|png)$}i,
:message=>'la imágen debe proceder de una URL en formatos GIF, JPG o
}
end
```

Aquí se detallan la validación de los objetos en el formulario y la forma en cómo deben ser suministrados los datos para ser almacenados en la base de datos.

VISTA

Aut_Prg/app/views/Layouts/application.html.erb

```
<!DOCTYPE html>
<html>
<head>
<title>Modelos a Escala .:.Maisto.:.</title>
<%= stylesheet_link_tag    "application", :media=>"all"%>
<%= javascript_include_tag "application"%>
<%= csrf_meta_tags %>
</head>
<body id="store">
        <div class="Main_Header">
                <h1>Ruby Source</h1>
                <%unless session[:user_id]%>
                        <ul class="nav">
                                <li><%= link_to :Signup , :signup, class: "nav_links"%></li>|
                                <li><%= link_to :Login , :login, class: "nav_links"%></li>
                        </ul>
                <%else%>
                        <ul class="nav">
                        <li><%= link_to :Cartilla , :home, class: "nav_links"%> |</li>
                        <li><%= link_to :Productos , :products, class: "nav_links"%> |</li>
                        <li><%= link_to :Pedidos , :compras, class: "nav_links"%> |</li>
                        <li><%= link_to :Logout , :logout, class: "nav_links"%></li>
                        </ul>
                <%end%>
        </div>
        <%if flash[:color]== "valid"%>
                <div class="valid_notice">
                        <p><%=flash[:notice]%></p>
                </div>
        <%elsif flash[:color]== "invalid"%>
                <div class="invalid_notice">
                        <p><%=flash[:notice]%></p>
                </div>
        <%else%>
                <div class="notice">
                        <p><%=flash[:notice]%></p>
                </div>
        <%end%>
<%=yield %>
</body>
</html>
```

Esta vista corresponde al menú de la aplicación, contiene los objetos que serán desplegados al usuario en la capa de presentación o GUI.

CONTROLADOR

Aut_Prg/app/controllers/Products_controller.rb

```
# POST /products
# POST /products.json
def create
@product = Product.new(params[:product])

respond_todo |format|
if@product.save
        format.html { redirect_to @product, notice: El producto se creó satisfactoriamente.'}
```

En el código del método create veremos que una vez que se ha guardado un Producto válido la acción establece un mensaje y luego redirige a la acción show del producto recién creado.

```
        format.json { render json: @product, status: :created, location: @product }
else
        format.html { render action: "Nuevo Producto"}
        format.json { render json: @product.errors, status: :unprocessable_entity }
        end
    end
end
```

El patrón MVC implementa las siguientes tácticas:

- **Coherencia semántica**: esta táctica la realiza a través del componente de Modelo en el cual se programan las distintas funcionalidades del sistema.

- **Encapsulación**: el componente Modelo encapsula las funcionalidades del modelo como tal y se encarga de realizar la abstracción de los datos. Este componente contiene la lógica del negocio y las bases de datos que interactúan con la vista.

- **Intermediarios**: el controlador se encarga de realizar el puente entre el modelo de negocio y el usuario. Esta funcionalidad está articulada en su totalidad por la Vista que es el verdadero intermediario entre el modelo y el usuario.

- **Enlace en tiempo de ejecución**: como se trata de una interfaz abierta (por lo general de tipo Web), los elementos de vista pueden ser invocados en cualquier momento por el modelo y desplegados a través de un conjunto de variables en tiempo de ejecución que permitiendo pasar parámetros entre las diferentes vistas.

- **Modificabilidad**: esta táctica es implementada gracias a la separación de responsabilidades. Los cambios que se requieran en alguno de sus componentes, especialmente las vistas, no afecta al modelo de negocio y al sistema en general. Se puede intercambiar con facilidad los componentes.

- **Usabilidad**: este atributo se logra a través de la separación de interfaces. Una vista puede ser utilizada para múltiples diseños o esta puede estar sincronizada a múltiples accesos como vista en un mismo modelo (formularios web por ejemplo).

Aunque el patrón MVC puede llegar a convertirse en un Framework potencial para el desarrollo de aplicaciones empresariales, existen algunos inconvenientes que se pueden presentar al momento de elegir esta arquitectura. Aunque la modificabilidad se visualiza como una de las tácticas más aplicada por la arquitectura, ésta puede verse afectada cuando los niveles de acoplamientos entre la vista y el controlador son demasiado bajos.

La táctica de Performance también se puede ver afectada cuando el acceso a los datos es deficiente o cuando estos no pueden ser accesados a través de una determinada

vista. Una deficiente configuración y restricción de accesos concurrentes a las vistas, a los datos con una excesiva carga de transacciones (a mayor complejidad, menor flexibilidad de la herramienta software), también puede afectar al atributo de Performance.

6.2.3.2. Control-Abstracción-Presentación (PAC)

PAC, "**P**resentación **A**bstracción **C**ontrol" es un patrón de arquitectura de software para sistemas interactivos basada jerarquías de agentes que cooperan entre sí. Cada agente es responsable de una tarea específica "funcionalidades del sistema". Este patrón es similar al MVC (Modelo-Vista-Controlador), ya que se distribuye en 3 capas; ***Presentación***, ***Abstracción*** *y* ***Control***. Esta división de componentes separa los aspectos de interacción ente la computadora y el usuario y la funcionalidad de los agentes y la comunicación de éstos con otros agentes. Los componentes de este modelo responden a las siguientes funcionalidades:

- **Presentación**: el componente se encarga de presenta la información y/o lógica de negocio en un formato adecuado para el usuario, como lo hace de forma análoga el patrón MVC en su capa de Vista. La representación de la información corresponde a la salida del sistema a través de una interfaz de usuario o GUI.

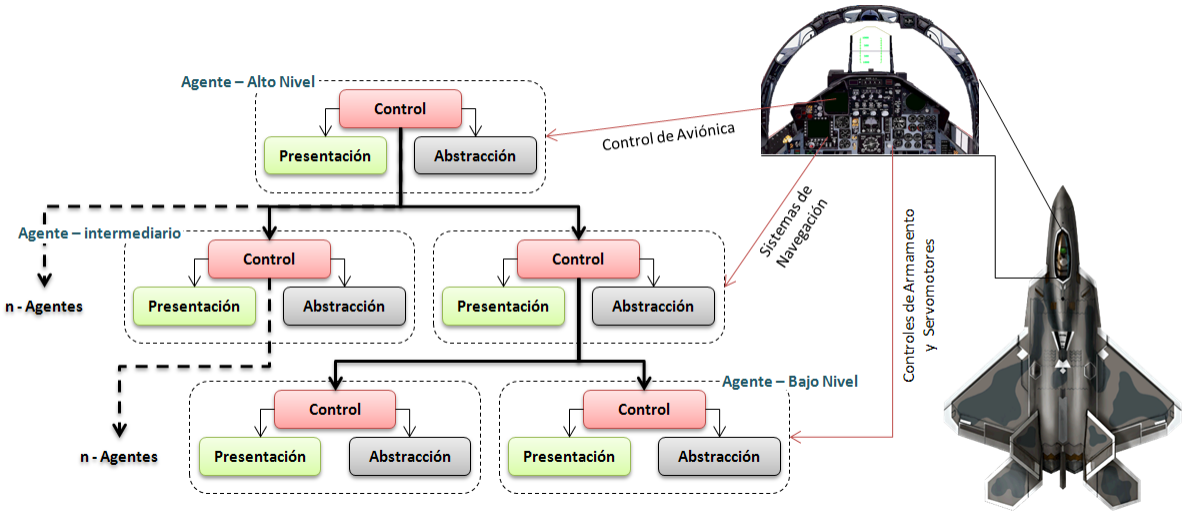
- **Abstracción**: al igual que MVC en su componente Modelo, este componente de la arquitectura contiene los datos y su funcionalidad. A diferencia de MVC este contiene una parte de las funcionalidades siendo más completo MVC en este contexto.

- **Control**: la funcionalidad de este componente es conectar la capa de presentación con los componentes de la capa de abstracción. Contiene un conjunto de elementos y funcionalidades para cooperación entre agentes a través de la descripción de jerarquías.

El patrón PAC es útil cuando se diseñan arquitecturas que contienen subsistemas independientes. Las implementaciones más conocidas de este modelo corresponden a sistemas para el control de tráfico en redes de datos y sistemas para operación de Robots o autómatas con manejo de sensores (equipos de búsqueda y rescate, detección de minas antipersonales, etc.). Otros sistemas pueden incluir a los sistemas de aviónica en un Jet de Combate donde interactúan múltiples sistemas, aunque este tipo de sistemas en su mayoría se modela a partir del patrón basado en eventos ***Publisher-Subscriber*** que permite desacoplar el procesamiento en sensores de los aspectos computacionales y de navegación del Avión.

El siguiente esquema muestra a modo de ejemplo, una posible implementación de este patrón arquitectónico.

Figura 20. Patrón Arquitectónico PAC (Presentación-Abstracción-Control)



El patrón PAC implementa las siguientes tácticas:

- **Coherencia semántica**: esta táctica se pone en escenario a partir de la separación de agentes, los cuales ejecutan una tarea específica en el sistema. Cada agente mantiene su estado y sus datos y se relacionan "cooperan" entre sí con otros agentes pero de forma independiente. Cada Agente posee un mecanismo que le permite interactuar con el usuario (display, mensaje, alertas). Este atributo de coherencia permite el desarrollo de capas de datos y modelos de acuerdo a cada concepto o semántica y los mecanismos de cooperación entre agentes.

- **Modificabilidad**: cambios realizados en alguno de los componentes a nivel de capa de presentación o de abstracción no afecta la funcionalidad de los otros agentes. Debido a esta característica, es muy fácil acoplar o integrar nuevos agentes sin mayores cambios en los demás, a partir de una interfaz predeterminada.

- **Usabilidad**: este atributo se logra a través de la separación de interfaces. Al soportar múltiples tareas, la arquitectura puede distribuir en forma concreta los diferentes hilos, procesos o estados, extendiendo la funcionalidad de los agentes afectando exclusivamente al componente y no al sistema en general.

El patrón presenta algunos inconvenientes; se puede evidenciar que tras cada semántica o concepto de aplicación para cada agente, puede incrementarse el factor de complejidad del sistema lo que conlleva a la generación de mecanismos de paralelismo más eficientes y con mayores capacidades en función de la granularidad del sistema. Esto hace que el sistema robustezca sus funcionalidades pero se requiere de una planificación y

procesos de refinación en cada agente para comunicarlos y hacer que cooperen de acuerdo a las responsabilidades asignadas (a través de concurrencia de procesos y sistemas de sincronización). La calidad de los componentes de Control en la arquitectura debe ser óptima para asegurar la cooperación entre agentes para garantizar la funcionalidad global y total de la arquitectura del sistema.

Otro factor que se puede ver amenazado en esta arquitectura es la Eficiencia. Esto ocurre cuando el overhead computacional "comunicación entre agentes" requiere de demasiados filtros o agentes intermediarios ya que al pasar los datos de un agente de bajo nivel, puede requerir el uso de diversos Intermediarios. Al intercambiar información entre agentes y comprobar el estado de sus procesos por separado, esta actividad incrementa considerablemente el tiempo de procesamiento hasta llegar a su destino "Bottom Agent" o agente de alto nivel. Si los agentes se encontraran distribuidos esto requeriría de procesos complejos para fragmentar y reunir los datos a través del canal de comunicación, lo que incrementa el costo computacional.

6.2.4. Sistemas Adapatables o Adaptativos (Adaptable Systems)

Adaptable Systems (Sistemas Adaptables): esta categoría está representada por 2 patrones; El MicroKernel o Micronúcleo y la arquitectura de Reflexión "Reflection". La potencia de estos patrones radica en la capacidad que proveen para adaptarse a los cambios y la extensibilidad de las aplicaciones en nuevos entornos, tecnologías y nuevos requerimientos.

6.2.4.1. MicroKernel

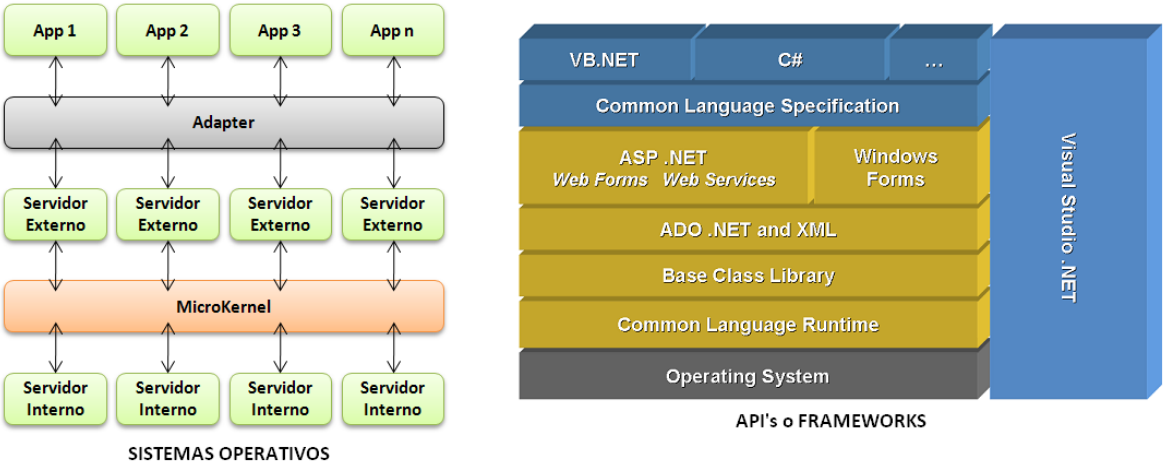
El patrón arquitectónico MicroKernel se aplica a aquellos sistemas de software que son susceptibles a cambios "nuevos requerimientos" en el tiempo. La arquitectura separa las funcionalidades principales del sistema y las encapsula en un componente denominado **Core** de las demás funcionalidades o partes específicas del cliente. Esta arquitectura es muy utilizada en el diseño de Sistemas Operativos y otros Kernels para sistemas específicos (Windows NT, Unix, Amoeba, Mach, Chorus, MDE (Microkernel Datenbank Engine), entre otros.

La arquitectura MicroKernel define 5 componentes que interactúan entre sí. Estos son; ***Internal servers***, ***External servers***, ***Adapters***, ***Clients*** y ***Microkernel***. El MicroKernel se encarga de la plantificación de hilos "threads" y no de los procesos en sí. Esta capacidad de la arquitectura hace posible la implementación de sistemas multitarea, ya que el MicroKernel orquesta y planifica en su totalidad al código que se ejecuta en el sistema.

El microkernel provee dos tipos de componentes "servidores". Unos internos, que extienden la funcionalidad del microkernel, y unos externos que proveen interfaces de programación a los clientes. Los "servidores" internos sólo son utilizados por los externos a través del microkernel como una extensión de este. Los externos resuelven a través del

MicroKernel la funcionalidad invocada por los clientes. El cliente no interactúa directamente sobre los servidores externos, sino que lo hace a través de los Adapters.

Figura 21. Patrón Arquitectónico MicroKernel



El patrón de MicroKernel implementa las siguientes tácticas:

- **Portabilidad**: una arquitectura de este tipo ofrece un alto grado de portabilidad. En la mayoría de los casos no se necesita definir servidores externos o aplicaciones de clientes pues son estos quienes se conectan a través de los puertos comunes del MicroKernel ajustando sus funciones a lo dispuesto por el Kernel. Si se quisiera migrar el Kernel a una arquitectura de hardware por ejemplo, solo se requiere modificaciones en las especificaciones del hardware y en el MicroKernel como tal.

- **Flexibilidad y Extensibilidad**: una de las mayores fortalezas de esta arquitectura radica en estos dos atributos de calidad. A modo de ejemplo se puede definir una nueva implementación de vistas para una aplicación. Lo único que necesita el arquitecto para conectar esta nueva vista con el sistema, es la agregación de un servicio externo, lo que extenderá las capacidades del sistema de acuerdo a las funcionalidades y nuevos requerimientos.

- **Mantenibilidad y Modificabilidad**: al utilizar una estructura modular, la arquitectura permite la activación de servicios externos y la implementación de nuevas prestaciones. La separación entre políticas y mecanismos incrementa la mantenibilidad y modificabilidad del sistema. A través de estos atributos, la arquitectura también permite la generación de servicios externos los cuales implementan sus propias vistas y parámetros.

- **Escalabilidad**: esta arquitectura como se describió anteriormente es ideal para el desarrollo de Sistemas Operativos y APIs de desarrollo. Además de esto, también es posible concebir SGBD "Sistemas de Gestión de Bases de Datos" y computación paralela. A partir de una configuración base, se puede extender la arquitectura y mejorar la capacidad de sus funcionalidades.

- **Fiabilidad (Disponibilidad y Tolerancia a Fallos)**: la disponibilidad es otro factor crítico en esta arquitectura. Una de las ventajas principales es la disponibilidad ya que este patrón permite la ejecución de un conjunto de servicios instanciados en un servidor y que pueden ser ejecutados en distintas máquinas a la vez. El hecho de que una estación falle, esto no afecta el funcionamiento de la aplicación. La tolerancia a fallos se enmascara fácilmente en la arquitectura y son detalles que se ocultan del cliente haciendo el proceso transparente. Todos estos procesos de intercomunicación, fallos y localización de servicios son gestionados por los Adapters quienes ocultan los detalles de implementación a los usuarios.

- **Performance**: se obtiene gracias a la utilización de diferentes vistas, las cuales se pueden accesar en forma concurrente desde cualquier punto de una red de servicios.

6.2.4.2. Reflexión (Reflection)

El patrón Reflection proporciona un mecanismo para cambiar estructuras y comportamientos de un sistema software en forma dinámica. Esta arquitectura soporta el atributo de modificabilidad para estructuras básicas como llamadas de función y tipos de estructuras. Este patrón divide una aplicación en 2 mecanismos; uno de ellos se encarga de las propiedades del sistema y hace que éste mismo reconozca sus propiedades; por otra parte el otro se encarga de manejar la lógica de la aplicación a bajo nivel.

Este patrón implementa las siguientes tácticas:

- **Encapsulación**: esta táctica es especificada a través de las diferentes interfaz que se encargan de manipular los meta-objetos.

- **Modificabilidad**: para realizar una modificación a un sistema basado en esta arquitectura, no es necesario realizar el cambio al código existente en su totalidad. Solo basta con cambiar los llamados a función o los protocolos de comunicación de los meta-modelos. Un buen diseño y un conjunto de protocolos para meta-objetos asegura o previene fallos no deseados a nivel de la semántica de la aplicación. Con la ayuda de este patrón es posible adaptar el software para que el mismo caracterice las necesidades específicas del mismo con base en el entorno y a partir de éstos pueda integrar el conjunto de requerimientos específicos del cliente.

7

ARCHITECTURE EXPERT DESIGN ASSISTANT .:.ArchE.:.

"Unas buenas especificaciones incrementarán la productividad del programador mucho más de lo que puede hacerlo cualquier herramienta o técnica" - Milt Bryce

ArchE es un Sistema Experto basado en atributos de calidad para herramientas software. Como herramienta experimental, el sistema desarrollado por el SEI de la universidad Cernegie Mellon, es una herramienta que ayuda a explorar diversas alternativas de diseño a través de un conjunto de tácticas y escenarios para mejorar la calidad del producto. ArchE como tal no se considera una solución de última milla, solo se ha concebido como una herramienta soporte para la toma decisiones, donde el arquitecto de software es quien elige en base a las variantes de diseño propuestas por ArchE la mejor solución al modelo.

La versión de ArchE sobre la cual se ha construido el presente informe de investigación corresponde a la versión 3.0.

Como se describe al inicio del párrafo de este capítulo, ArchE es una herramienta software basada en atributos de calidad. A partir de un conjunto de escenarios y responsabilidades, ArchE evalúa los parámetros de arquitectura en base a Marcos de Razonamiento "*Reasoning Framework – RF*".

Los Marcos de Razonamiento o Motor de Inferencia como se denomina a menudo en otros sistemas expertos, utiliza como corrientes de entrada para el análisis de la arquitectura, un conjunto de atributos basados en las **Responsabilidades** del sistema. El proceso de evaluación de Responsabilidades se efectúa a partir de la parametrización de requisitos funcionales denominados por ArchE como **Escenarios**. A partir de estos escenarios se construyen las funcionalidades del sistema y las dependencias que pudieran existir entre dichas funcionalidades.

A partir de las Responsabilidades, el arquitecto define las relaciones bidireccionales que existirán entre sí en base a un Marco de Razonamiento específico (Performance (ICM) Reasoning Framework ó Modifiability (Change Impact) Reasoning Framework.). Estos parámetros son modificables y es potestad del arquitecto definirlas según su necesidad. Desde la perspectiva de los escenarios, el sistema ArchE puede realizar el análisis de un módulo o dependencia a partir de datos como esfuerzo, tiempo de ejecución, costes por cambio en parámetros de la arquitectura, entre otros atributos y en base a estos requerimientos ArchE determina a través de los RF las estrategias o tácticas a utilizar para mejorar y optimizar el diseño.

Antes de iniciar cualquier análisis, los RF sobre los cuales trabaja ArchE deben estar activos (Ver Fig. 26). Esto hará que cualquier cambio realizado en la parte de requisitos funcionales y dependencias sea actualizado en tiempo real ya que cada cambio incluido en el modelo supone una o varias alternativas para el arquitecto, y es aquí donde los RF juegan un papel importante.

El funcionamiento del sistema experto ArchE se enmarca en 4 conceptos: ***Responsabilidades***, ***Tácticas***, ***Marcos de Razonamiento*** *y* ***Escenarios***.

A partir de cada proceso de evaluación, ArchE construye un modelo de referencia de la arquitectura a partir de los atributos programados por el arquitecto a través de los distintos escenarios y como resultado ArchE devuelve el análisis de dichas variables a partir de un conjunto de métricas y reglas definidas en el sistema experto. El grado de cumplimiento en cada uno de los escenarios es evaluado por el RF y en base a la aplicación de las métricas de calidad, ArchE sugiere al arquitecto las posibles mejoras a introducir en el modelo. A este conjunto de mejoras se les denomina ***TÁCTICAS***[15].

7.1. RESPONSABILIDADES EN ArchE

Las Responsabilidades representan las funcionalidades del sistema y caracterizan a la arquitectura del software. "Las responsabilidades, al igual que las funciones, se pueden añadir, eliminar, modificar o dividir, para llegar a cumplir los requisitos de la arquitectura, y más en concreto, de los atributos de calidad correspondientes"(Pérez-Campanero, 2010).

[15] Una ***TÁCTICA*** es una decisión de diseño que influyen en la respuesta al atributo de calidad. Un conjunto de tácticas se refiere a una estrategia de arquitectura.

Las Responsabilidades incluyen en sí mismas un conjunto de parámetros de distribución/asignación requeridos por los marcos de razonamiento para el análisis de tácticas y mejora del producto.

7.2. TACTICAS ARQUITECTÓNICAS UTILIZADAS POR ArchE

Una **Táctica** se puede definir como un mecanismo que sirve para satisfacer los atributos de calidad de una herramienta software a partir de la introducción de cambios en las dependencias y responsabilidades de un conjunto de elementos o estilos arquitectónicos.

En la versión 3.0 de ArchE, las tácticas se aplican de acuerdo al Marco de Razonamiento que desee aplicar el arquitecto, estas tácticas pueden ser a nivel de ***PERFORMANCE*** a través del Marco Performance (ICM) Reasoning Framework o a nivel de ***MODIFICABILIDAD*** a través del Marco Modifiability (Change Impact) Reasoning Framework.

7.3. REASONING FRAMEWORK - RF

La función principal de un Marco de Razonamiento consiste en encapsular el conocimiento el cual es aplicado por ArchE sobre una arquitectura específica con base en atributos de calidad. Un RF proporciona al arquitecto una visión de mejora para el producto software de acuerdo al comportamiento y las dependencias existentes en una herramienta o modelo.

Una de las características que más se destaca en los Marcos de Razonamiento utilizados por ArchE, es la reducción de interacciones (dependencias) entre los diferentes atributos de calidad. Cada Marco de Razonamiento emplea un conjunto de algoritmos de tipo analítico para realizar representaciones formales del modelo y en base a éstos produce una salida abstracta en base al razonamiento obtenido. La base del razonamiento no es más que un conjunto de restricciones que evaluadas en la arquitectura aseguran el cumplimiento de factores de calidad en el desarrollo de software.

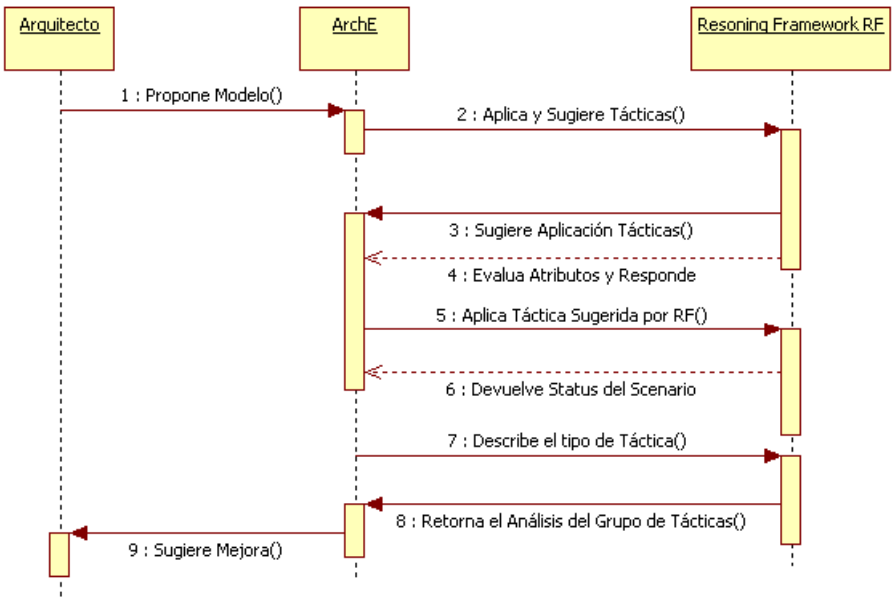
Las reglas de producción de ArchE se ciñen a supuestos fácticos, por esta razón el sistema experto se convierte en una herramienta de análisis y no de solución. Le corresponde al arquitecto decidir cómo y en qué forma deberá aplicar las tácticas hasta que el sistema sea lo más optimizado posible.

Para que el Marco de Razonamiento efectúe su tarea se precisa la definición de:

- Requisitos Funcionales y Arquitectónicos, Relaciones, Responsabilidades y capacidades que influyen en los atributos de calidad. (Los modelos de representación de ArchE extraen la información a partir de los escenarios de conocimiento y los transforma en salidas creando o aproximando un modelo).

- Conjunto de variables y algoritmos de tipo analítico “fórmulas” para realizar los cálculos en el modelo que permitan medir la efectividad de las tácticas aplicadas al modelo. (Estas formulas ya son proveídas por el Marco de Razonamiento y están representadas en ArchE por el MAST “Modelling and Analysis Suite for Real-Time Application”) que analiza los modelos y retorna los resultados al modelo inicial para su interpretación por el arquitecto de software.

Figura 22. Diagrama de Secuencia entre Reasoning Framework y ArchE



Una explicación de cómo interactúa el RF con ArchE se detalla a continuación:

1) El arquitecto planifica y modela los respectivos escenarios con los atributos de calidad para el análisis de ArchE.

2) ArchE traduce el modelo en base a la descripción de la arquitectura planteada por el arquitecto. Es el primer punto de partida donde se aplican las tácticas de calidad las cuales son proporcionadas por el arquitecto. Esta secuencia puede verse como un modelo de interpretación.

3) El Motor de Razonamiento evalúa los atributos de calidad propuestos en los escenarios en términos del modelo. Aquí se produce la primera evaluación. A partir de esta iteración, el RF sugiere o propone la táctica más adecuada para aplicar al modelo con base en los requerimientos establecidos, que conlleven a la mejora del modelo.

4) El Marco de Razonamiento al proporcionar la evaluación de los atributos, devuelve el status del escenario e informa a ArchE de las posibles inconsistencias en el modelo.

5) ArchE aplica las tácticas o conjunto de éstas sugeridas por el Marco de Razonamiento, modificando así los atributos de calidad que de una u otra forma están afectando de forma negativa o positiva al modelo. En este paso son aplicadas las tácticas.

6) Al ser aplicada(s) la(s) táctica(s), el RF actualiza el (los) escenario(s) para un nuevo análisis. Si todas las tácticas aplicadas satisfacen el modelo y sus atributos de calidad, el análisis concluye; de lo contrario se seguirán planteando tácticas por parte del RF al sistema ArchE hasta que se satisfaga la arquitectura. (este mecanismo no es automático y debe ser evaluado por el arquitecto tras cada iteración o proceso de evaluación).

7) El RF describe la táctica o grupo de éstas con mayor preponderancia y efectividad en el modelo y las cualidades de sus atributos.

8) y 9) Una vez solucionado y evaluado satisfactoriamente el conjunto de escenarios, RF y ArchE se encargan de comunicar al usuario sugiriendo la mejora y proponiendo nuevas dependencias o funcionalidades para optimizar el modelo.

Tabla 18. Elementos que componen al Framework de Razonamiento

ELEMENTO	DESCRIPCIÓN / FUNCIONALIDAD
Descripción del Problema	Corresponde al conjunto de atributos de calidad o funcionalidades empleadas por el Marco de Razonamiento para identificar los atributos de calidad, su medida y la propia mejora. El problema se calcula con base en un set de cualidades o requerimientos (restricciones de calidad) que serán utilizados posteriormente para el análisis y evaluación de las tácticas.
Teoría Analítica	Los Marcos de Razonamiento se basan en sistemas de Teoría Analítica. Disciplinas o técnicas como la teoría de colas, autómatas finitos, lógica formal y temporal, metodologías y algoritmos de planificación, RMS "Rate Monotonic Scheduling", entre otros algoritmos de planificación, hacen parte del núcleo analítico del RF.
Restricciones Analíticas	Corresponden al conjunto de Restricciones que satisfacen a los algoritmos dispuestos por la teoría analítica.
Representación del Modelo	Se encarga de condensar el conjunto de atributos mediante el uso

ELEMENTO	DESCRIPCIÓN / FUNCIONALIDAD
	de modelos específicos de representación "Orientado a Aspectos" para mejorar el uso de dependencias. Un modelo para RF responde a una abstracción del modelo teórico y a la producción de reglas para la evaluación de un proceso software.
Interpretación	Se refiere a cualquier procedimiento que genera un modelo optimizado y su representación en un modelo en base a la descripción de la misma arquitectura.
Procedimiento de Evaluación	Hacen parte de este, los algoritmos, cálculos y fórmulas matemáticas que permiten la evaluación y medición de los atributos de calidad a partir de un modelo de representación.

Fuente: Adaptada de (Bass, Ivers, Klein, & Merson, 2005).

7.3.1. Modifiability Reasoning Framework

Los atributos de **modificabilidad** en una arquitectura dependen directamente de la funcionalidad agregada a cada uno de los módulos y las dependencias entre éstos. Un atributo de modificabilidad mide en esencia el ***costo/esfuerzo*** en razón a un cambio introducido en el modelo. Para que una táctica de modificabilidad pueda ser introducida por ArchE al modelo debe poseer:

- Descripción de responsabilidades (con sus dependencias y su respectiva descomposición);
- Costos o esfuerzo que implica el cambio en la táctica. (La asignación de costos de cambio para cada nivel de responsabilidad, es potestad del arquitecto);
- Flujos de dependencia entre responsabilidades (niveles de acoplamiento) y
- Asignación de responsabilidades entre los distintos módulos que componen la arquitectura.

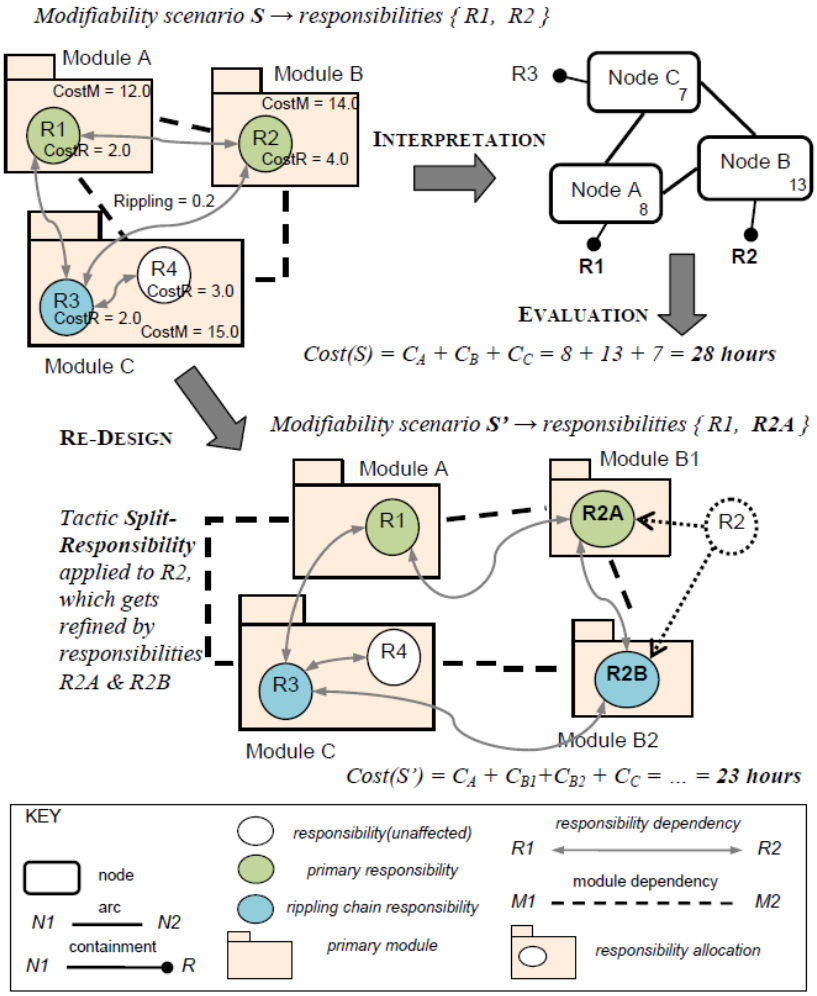
En la Arquitectura ArchE, el Arquitecto es quien modela y asigna los atributos de calidad al sistema. Una de las responsabilidades que debe introducir el Arquitecto como primera medida al modelo son los niveles de acoplamiento entre las dependencias y las responsabilidades. De no ser catalogadas inicialmente por el Arquitecto, ArchE asume como valor inicial de acoplamiento para cada una de ellas, la probabilidad de (0.7).

Cuando el marco de razonamiento es ejecutado, las reglas de inferencia producen un modelo de interpretación en base a los elementos de diseño dispuestos por el Arquitecto y las relaciones de los mismos a través de los distintos escenarios. Cada representación de modelo se asemeja a un grafo dirigido y es a partir de éste donde se realiza el proceso de evaluación de cada indicador por medio de formulas matemáticas y estimación de costos.

Para entender en detalle el proceso de aplicación de las tácticas, se hará referencia al documento *Integrating Quality-attribute Reasoning Frameworks in the ArchE Design Assistant* descrito por los autores (Diaz-Pace, Kim, Bass, Bianco, & Bachman, 2008). En la Figura 23 se muestra un conjunto de nodos que han sido impactados por un costo determinado de acuerdo a un escenario. Una vez finalizada la aplicación de las tácticas (en este caso de modificabilidad), se introducen 2 estrategias de mejora para el modelo. La aplicación de de la primera táctica ayuda a reducir el costo por modificaciones en uno de los módulos o escenarios específicamente en sus Responsabilidades. A cambio de ello, ArchE introduce la táctica ***divide y vencerás*** "Split" en una Responsabilidad derivada.

La segunda táctica ayuda a reducir el factor de acoplamiento entre los distintos módulos y se introduce un intermediario para romper las dependencias entre éstos. Una vez son gestionadas, se produce la respectiva transformación afectando a ambos módulos y por consiguiente a sus estructuras de responsabilidades.

Figura 23. Interpretación, Evaluación y Rediseño de atributos de Modificabilidad



Fuente: (Diaz-Pace, Kim, Bass, Bianco, & Bachman, 2008).

Cada introducción de tácticas lleva consigo a una nueva interpretación de una arquitectura producto de dicha transformación, de la cual se derivan diversos grafos de dependencia que varían en la medida en que se aplican los costos por modificación introducidos en cada iteración. El proceso se repite hasta que los escenarios satisfagan en su totalidad a los escenarios de calidad dispuestos por el arquitecto.

El procedimiento para dividir una responsabilidad se realiza de la siguiente manera:

1) Se crean "instancian" dos nuevas responsabilidades (R1 y R2) como producto del refinamiento de la responsabilidad primaria R.
2) La responsabilidad R es liberada de las dependencias asignadas inicialmente en el modelo (entre el módulo A y B).
3) Después de eliminar las dependencias de la responsabilidad primaria, ArchE procede a ubicar las dos responsabilidades hijas (R1 y R2) y las distribuye entre el módulo A y el módulo B.
4) Se asigna la respectiva relación de dependencia entre R1 y R2.
5) ArchE analiza la relación de dependencia de la responsabilidad primaria R, para verificar el grado de acoplamiento inicial.
6) ArchE remueve la dependencia de R y redefine las nuevas dependencias para R1 y R2, y analiza para ambas el grado de acoplamiento el cual no debe superar al grado de acoplamiento de la responsabilidad primaria.
7) El sistema actualiza las dependencias entre módulos conforme a los nuevos requerimientos.
8) Finalmente ArchE actualiza los costos y el acoplamiento de acuerdo a las necesidades propuestas en los escenarios. Estas decisiones se ponen a consideración del arquitecto quien en últimas decide cómo y de qué manera acoplará los distintos módulos con base en las nuevas responsabilidades generadas a partir de la modificación.

El procedimiento para insertar un intermediario se realiza de la siguiente manera:

1) Se crea una nueva responsabilidad R para que haga las veces de intermediario.
2) La nueva responsabilidad es asignada a un nuevo módulo M.
3) ArchE elimina del grafo de dependencias la instancia del módulo inicialmente acoplado con alguna otra responsabilidad en el sistema.
4) Una vez eliminada la dependencia inicial, se redirecciona la dependencia al módulo creado M.
5) El siguiente paso consiste en redireccionar las responsabilidades contenidas inicialmente en el módulo y sus responsabilidades asociadas y se redireccionan dichas dependencias a la nueva responsabilidad R.
6) Se crea el nuevo grafo de dependencias entre el módulo anterior y M.
7) Finalmente, ArchE actualiza los costos conforme a las necesidades propuestas en el escenario afectado por la modificabilidad.

7.3.1.1. Tácticas de Modificabilidad aplicadas por ArchE

Las Tácticas de Modificabilidad aplicadas actualmente por ArchE son las siguientes:

- ***Encapsulación***. (para reducir acoplamientos).
- ***Manejo de Niveles de Abstracción***. (para reducir acoplamientos).
- ***Uso de Intermediarios "Intermediary"***. (para reducir acoplamientos en algunas dependencias).
- ***Separación o División de Responsabilidades***. (para habilitar nuevos flujos de responsabilidades y creando así nuevos módulos con sus respectivos costos y atributos).
- ***Localización***. (para realizar cambios de responsabilidades a nivel de módulos).
- ***Empaquetado "Wrapping"***. (para reducir todos los niveles de acoplamiento resultantes tras cada proceso de iteración y análisis).

7.3.2. Performance (ICM) Reasoning Framework

El Framework de Razonamiento ICM se basa en el algoritmo de planificación RMS "Rate Monotonic Scheduling". Este tipo de algoritmo realiza un proceso de asignación de prioridad estático o dinámico en base a peticiones periódicas, donde la mayor prioridad es asignada a las peticiones con periodos más cortos.

El performance o Rendimiento se define como el tiempo que requiere el sistema para responder a un evento o estímulo (tiempo de respuesta), o bien el número de eventos procesados en un intervalo de tiempo. El performance de un sistema depende esencialmente de:

- Comunicación entre componentes;
- Asignación de funcionalidad en los distintos componentes;
- Algoritmos que implementan la funcionalidad y
- La codificación misma de los algoritmos.

En forma general, la Performance de un sistema o modelo puede ser analizada desde las siguientes perspectivas:

- Análisis de las tasas de llegada;
- Distribución de los requerimientos de servicios;
- Tiempos de procesamiento;
- Tamaño de las colas y
- Latencia (tasa de servicio)

Para realizar el análisis por medio del Marco de Performance, la Arquitectura ArchE trabaja sobre la base de los siguientes supuestos:

- Utilización de un solo procesador (sin hilos ni multiprogramación);
- Cada unidad computacional corresponde a una tarea básica y no a un conjunto de instrucciones a computar (Principio de Unidad Básica);
- Los recursos pueden ser compartidos entre tareas;
- Las tareas son independientes excepto por la explícita dependencia sobre los recursos compartidos;
- Una tarea solo puede ser ejecutada a la vez y solo esta puede utilizar un recurso compartido en un momento dado;
- La ejecución de tareas en el Microprocesador, vienen dadas por el algoritmo de planificación del mismo y por la asignación de la respectiva prioridad por parte del sistema operativo.

La performance depende en parte de la asignación de funcionalidades y tareas. La táctica más empleada para medir la efectividad del rendimiento en una aplicación está asociada a la tasa de servicio "Latencia" (que se entiende como el tiempo que toma el procesador para finalizar una tarea). Para que ArchE realice el respectivo análisis a partir del Framework ICM, es necesario que el arquitecto proporcione la siguiente información:

- Descripción de escenarios;
- Descripción de tiempos y periodos de ejecución y
- La asignación y trazado de escenarios a responsabilidades.

En este orden de ideas, el arquitecto debe construir el respectivo escenario que describa en forma detallada la arquitectura del modelo. Cada escenario representa en sí una tarea y se convierte en el objetivo base para al análisis de la Performance. Cada periodo refiere en esencia a una tarea y en base a ello a cada responsabilidad le es asignado un tiempo de ejecución determinado, y cada responsabilidad es asociada con un escenario. La asignación de prioridades son asignadas por ArchE dependiendo de la carga operativa de cada tarea y de las responsabilidades asociadas y a los recursos compartidos por cada tarea.

El Framework de Razonamiento ICM se fundamenta en 2 conceptos o framework de análisis. El primer Framework se basa en mecanismos de análisis predictivo GRMA (Generalized Rate Monotonic Analysis) al cual denominan Lambda-* (Moreno & Hansen, 2008) y el marco de análisis MAST (**M**odeling And **A**nalysis **S**uite For Real-**T**ime Applications) (Medina, 2005).

Lambda-* es un Framework de Razonamiento basado en el atributo de calidad Performance. Este Marco se basa en mecanismos de planificación para realizar análisis de tipo predictivo sobre variables periódicas. Este mecanismo de predicción evalúa los promedios y peores tiempos de latencia "***worst-case latency***" de un sistema en tiempo real. Además de esta capacidad, Lambda-* evalúa las funcionalidades de un sistema con base en mecanismos estocásticos.

Un mecanismo **estocástico** se basa en la teoría de la probabilidad para caracterizar una sucesión de variables aleatorias (estocásticas) que evolucionan en función de otra variable en el tiempo. Cada una de las variables aleatorias del proceso tiene su propia función de distribución de probabilidad y, entre ellas, pueden estar correlacionadas o no.

Lambda-* es un mecanismo que puede aplicarse a un sinnúmero de arquitecturas; los mecanismos más implementados que se basan en este framework se encuentran aplicados en sistemas de control embebidos en la industria automotriz, sistemas de robótica e inteligencia artificial y sistemas para aviónica y simulación en tiempo real. Otras aplicaciones pueden incluir sistemas multimedia, para multiplexación y mezcla de canales de audio y video.

Lambda-* Performance Reasoning Framework realiza la predicción a partir de:

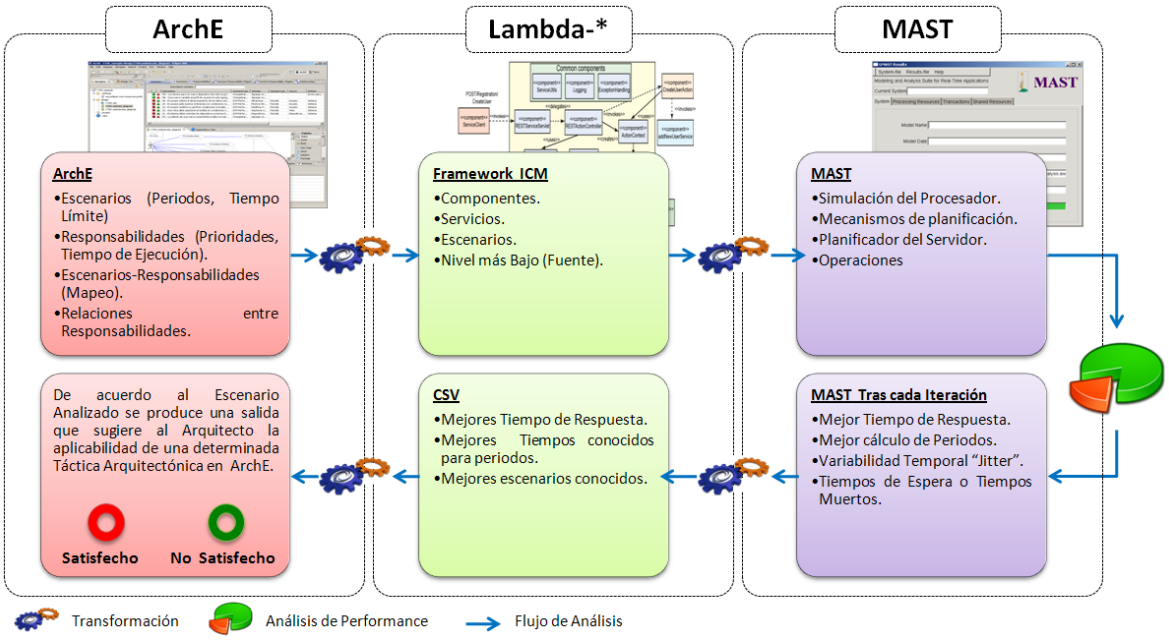
- Teorías analíticas que fundamentan la base del razonamiento,
- restricciones analíticas basadas en supuestos, desde las cuales se modelan nuevos escenarios a partir de este conjunto de restricciones, definiendo posibles variantes que el modelo puede analizar posteriormente,
- un modelo de representación captura los aspectos más importantes del sistema y que son relevantes para el análisis en una semántica entendible por la arquitectura en función de futuras predicciones,
- un modelo de interpretación que recrea el análisis del modelo a partir de las especificaciones del sistema (escenarios), conectando la semántica de las especificaciones del software con el modelo de análisis y
- un mecanismo de evaluación que soporta el proceso de análisis del modelo a partir de los esquemas monotónico para producir la respectivas reglas o predicciones. Este tipo de mecanismo es transparente para el usuario ya que los procesos de interpretación y evaluación son encapsulados en la misma herramienta.

El otro concepto bajo el cual trabaja el Framework ICM de ArchE es el MAST (Modelling and Analysis Suite for Real-Time Applications). MAST ofrece un entorno de herramientas de análisis y diseño de tiempo real, entre las que se incluyen herramientas para el cálculo de tiempos de respuesta, tiempos de bloqueo, para el análisis de sensibilidad a través del cálculo de los márgenes temporales (slacks), así como para la aplicación de técnicas de asignación óptima de prioridades (Medina, 2005).

MAST más que ser una herramienta para análisis de tácticas arquitectónicas en software, es una herramienta que contiene diversos recursos "elementos software" para facilitar el análisis de planificabilidad en sistemas mono-procesador y distribuido, así como para facilitar el cálculo de holguras, asignación óptima de prioridades, simulación de eventos discretos, etc. (Gonzáles Harbour, Gutiérrez, Palencia, & Drake, 2001). La mayor

parte de las herramientas disponibles en MAST operan sobre planificación basada en prioridades fijas.

Figura 24. Framework de Razonamiento ICM - Performance en ArchE



Fuente: Adaptada de (Champagne & Gagné, 2011).

7.3.2.1. Tácticas de Performance aplicadas por ArchE

Las Tácticas de Performance aplicadas actualmente por ArchE son las siguientes:

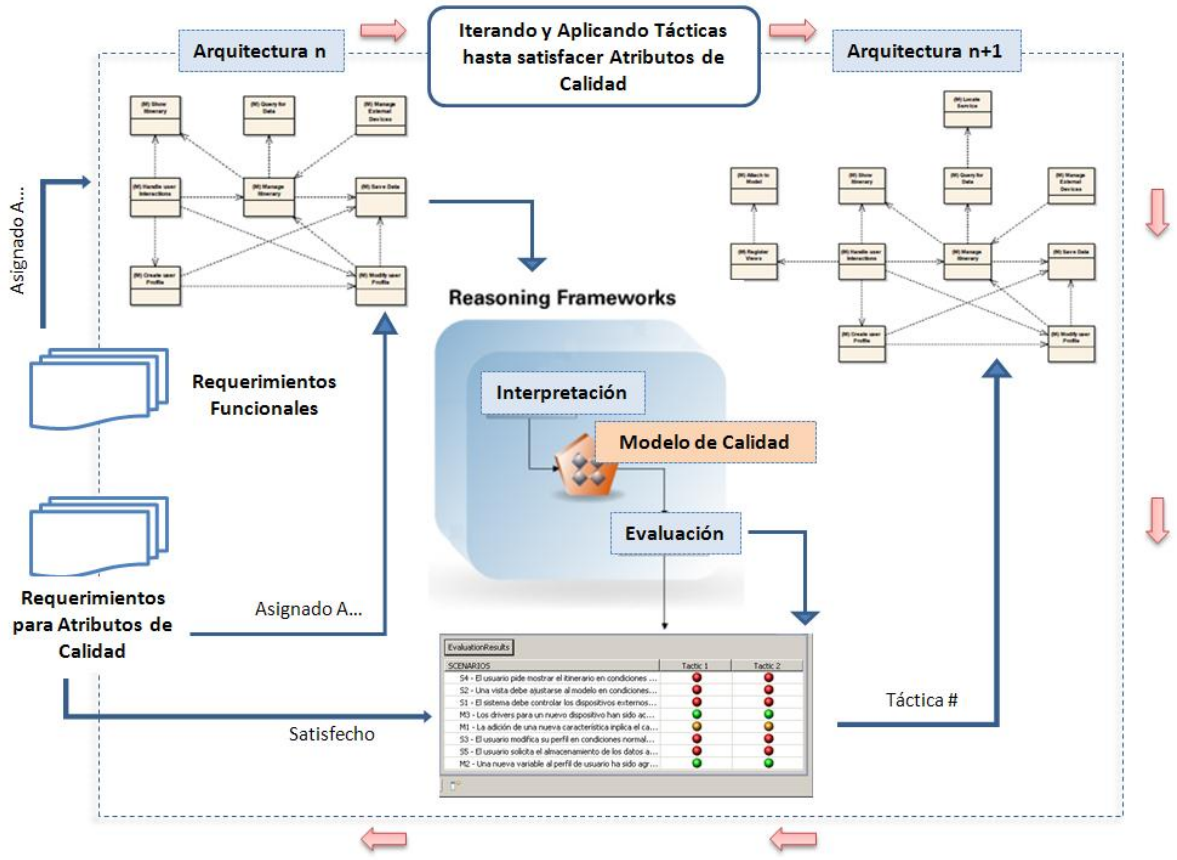
- ***Tácticas de requerimiento*** (incremento de periodos, incremento o extensión de plazos de entrega "deadlines").

- ***Tácticas de Planificación***. (se considera que la tarea con menor plazo o cumplimiento de tiempo, es aquella que tiene la mayor prioridad y su valoración supone un grado óptimo o solución factible.

- ***Tácticas sobre Tiempo de Ejecución***. (su tarea fundamental es reducir el tiempo de ejecución).

- ***Tácticas para la distribución y compartición de recursos***. (División de Responsabilidades para reasignar prioridades y tiempo de ejecución).

Figura 25. Principios Básicos de la Arquitectura en ArchE



Una táctica arquitectónica en resumen se considera como una pequeña transformación de una arquitectura existente. Una táctica es una decisión de diseño que influye en la respuesta al atributo de calidad.

7.4. ESCENARIOS EN ArchE

ArchE al igual que otras arquitecturas basadas en escenarios, realizan sus procesos de análisis a partir de una serie de comportamientos "Responsabilidades" y la interrelación entre componentes. A partir de un "Análisis temprano", ArchE detecta posibles errores en la arquitectura o diseño mucho antes de que esta se empiece a codificar (desarrollar).

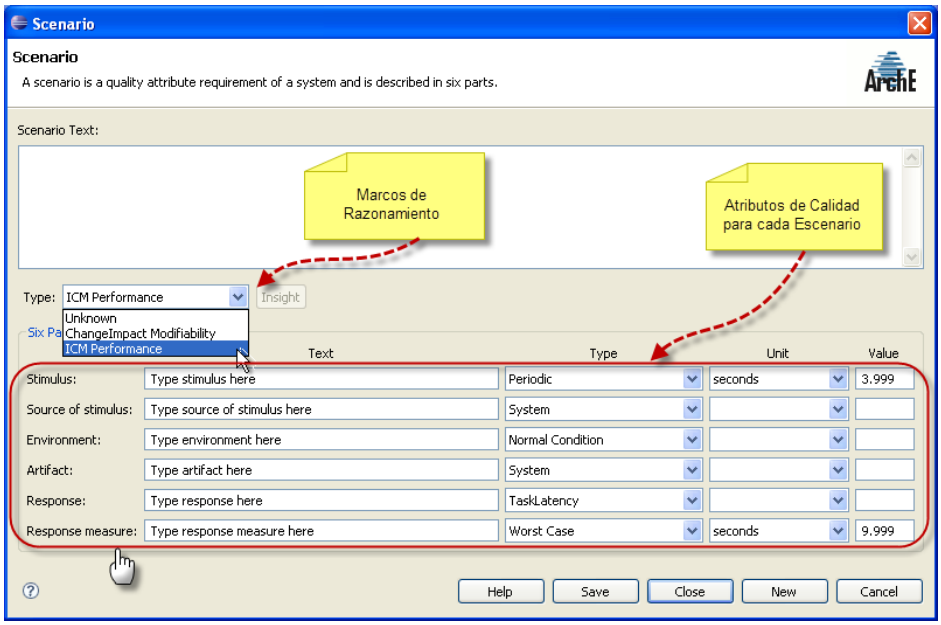
Un **Escenario** (*Scenario*) corresponde a una descripción de pasos o secuencias para llevar a cabo el desempeño normal (éxito) de un Caso de Uso. Puede incluir múltiples escenarios para abarcar extensiones, excepciones y caminos alternativos. Un escenario permite evaluar el comportamiento de un modelo a partir de una serie de requisitos o estímulos proveídos por el usuario y procesa una salida en función de los atributos de calidad proporcionados por el arquitecto. El resultado final será la medida del escenario en función de dichos atributos, evaluando si cumple o no con el requerimiento.

Un escenario está descrito en 6 partes:

- **Estímulo**: (Stimulus). evento que está efectuando el sistema.
- **Respuesta**: (Response). hace referencia a la actividad resultante en base a un estímulo.
- **Fuente de Estímulo**: (Source of Stimulus). corresponde a la entidad que está generando el estímulo.
- **Entorno**: (Environment). es la condición bajo la cual ocurre un estímulo.
- **Artefacto**: (Artifact). es el Artefacto que ha sido estimulado. Puede ser el conjunto total del sistema o un componente del mismo.
- **Evaluación**: (Response Measure). al ocurrir una respuesta, es necesario que la misma sea medida para verificar y evaluar si los objetivos "Requisitos" satisfacen el diseño.

A continuación se describe en la imagen las 6 partes en las cuales se fundamentan los escenarios de ArchE

Figura 26. Escenarios en la Arquitectura ArchE



7.5. FLUJO DE TRABAJO EN ArchE

De acuerdo a la Fig. 25, el flujo de información y análisis de tácticas se aplica de la siguiente manera:

- Entradas por parte del Arquitecto de Software
  - Rasgos o características que necesitan ser computadas por el sistema que se encuentra en desarrollo.
  - Especificación de requerimientos y atributos de calidad a través de escenarios.

- El sistema experto ArchE solicita al Arquitecto la información necesaria para determinar el comportamiento de los atributos de calidad, tales como tiempo de ejecución, costo por cambios y otros atributos necesarios para el análisis.

- A continuación, el sistema experto ArchE realiza las siguientes operaciones:

  - Proposición de un nuevo diseño ajustado a los atributos del Arquitecto;
  - Identifica posibles escenarios y atributos de calidad que no han sido tenidos en cuenta en el diseño inicial;
  - Procesa, ejecuta y Proporciona al Arquitecto una lista de posibles variantes de diseño representadas a través de **Tácticas**. La táctica o conjunto de éstas representan estrategias de diseño para mejorar los atributos de calidad. Un ejemplo de estas tácticas pueden incluir inserción de intermediarios para mejorar los atributos de modificabilidad en determinada área o introducir mecanismos de concurrencia en un área para mejorar la Performance.

- Es potestad del Arquitecto escoger y aplicar una o varias tácticas para la mejora del diseño. ArchE aplicará dichas tácticas para producir nuevos modelos a partir del diseño propuesto o a partir de cada iteración hasta que el modelo esté satisfecho por los atributos de calidad.

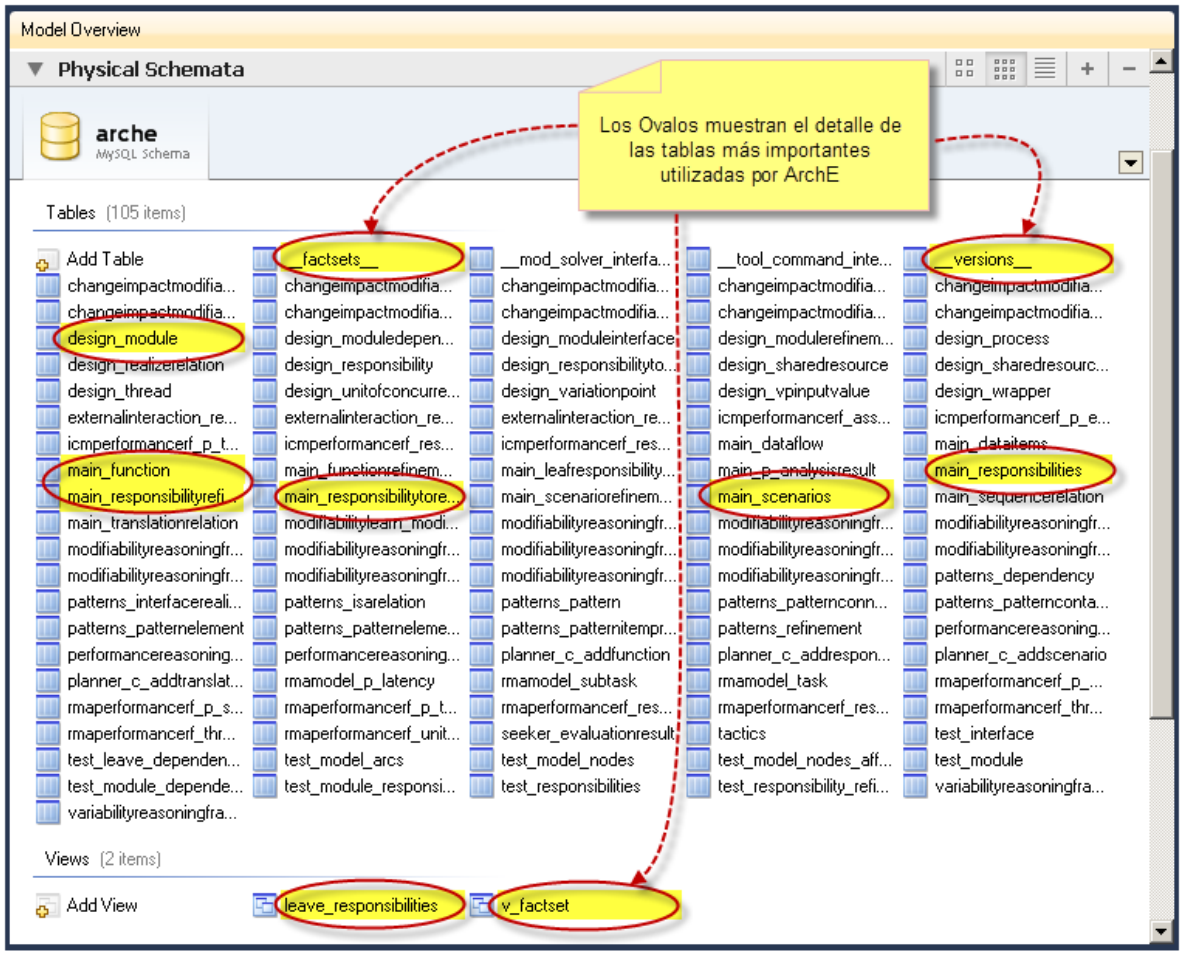
- Si es necesario, el Arquitecto debe proporcionar información adicional para el nuevo diseño propuesto por ArchE. Una vez introducidas las variables, ArchE entrará a computar nuevamente las entradas y los nuevos parámetros introducidos por el Arquitecto.

- El Proceso de Análisis se repite hasta que

  - Los atributos de calidad y los requerimientos se han cumplido satisfactoriamente;
  - El diseño se ajusta a las necesidades del Arquitecto y el modelo de negocio;
  - ArchE no tiene propuesta alguna de mejora para el modelo.

ArchE posee una robusta base de datos modelada en MySQL. El listado de entidades es extenso (105 tablas y 2 vistas) pero nos enfocaremos en las más importantes. Para las bases de datos de ArchE, los diseñadores del SEI han dispuesto como motor de gestión a la base de datos MySQL de Oracle. ArchE como prototipo y como desarrollo fundado en elementos de código abierto como xmlBlaster, Eclipse y el mismo MySQL, hace uso del mecanismo de almacenamiento de datos de código abierto para bases de datos MySQL denominado InnoDB que soporta transacciones CRUD y bloqueo de registros con manejo de Integridad Referencial. El listado completo de las bases de datos de ArchE se ilustra a continuación.

Figura 27. Esquema de Base de Datos .:. ArchE .:.



Las tablas que sobresalen en el modelo de datos y su funcionalidad se describen en la siguiente tabla.

Tabla 19. Principales tablas de ArchE en MySQL

TABLA	DESCRIPCIÓN
design_module	Esta tabla contiene los datos de las vistas generadas del modelo a partir de los esquemas de Razonamiento. ArchE modifica la forma en cómo accede a los datos registrados en esta tabla dependiendo del Marco referenciado por el modelo.
icmperformancerf_p_executiontime	Tabla que almacena los tiempos de ejecuciones planteadas por el framework ICM y su posterior análisis de tácticas.
leave_responsibilities	Esta tabla alberga al conjunto de responsabilidades que han sido generadas por ArchE a partir de nuevas iteraciones y responsabilidades iniciales.
main_function	Contiene las definiciones de la arquitectura (funcionalidades).
main_functionrefinementrelation	Esta tabla contiene las relaciones adicionales que han sido creadas por la aplicación de alguna táctica o por el Marco de Razonamiento.
main_responsibilities	Almacena el conjunto de responsabilidades descritas en la arquitectura y las cuales son definidas a partir de las funcionalidades del sistema.
main_scenarios	Almacena el conjunto de escenarios que han sido planificados por el Arquitecto.
main_responsibilityrefinementrelation	Almacena el conjunto de responsabilidades a partir de la aplicación de la táctica de Split o división de responsabilidades.
__factsets__	contienen las diferentes relaciones que admiten los marcos de razonamiento utilizados por ArchE
__versions__	Esta tabla registra todas las arquitecturas diseñadas por ArchE tras la aplicación de tácticas, así como las definidas en primera instancia por el arquitecto “modelo inicial”.

7.6. DESCRIPCIÓN DE COMPONENTES Y ARQUITECTURA EN ArchE

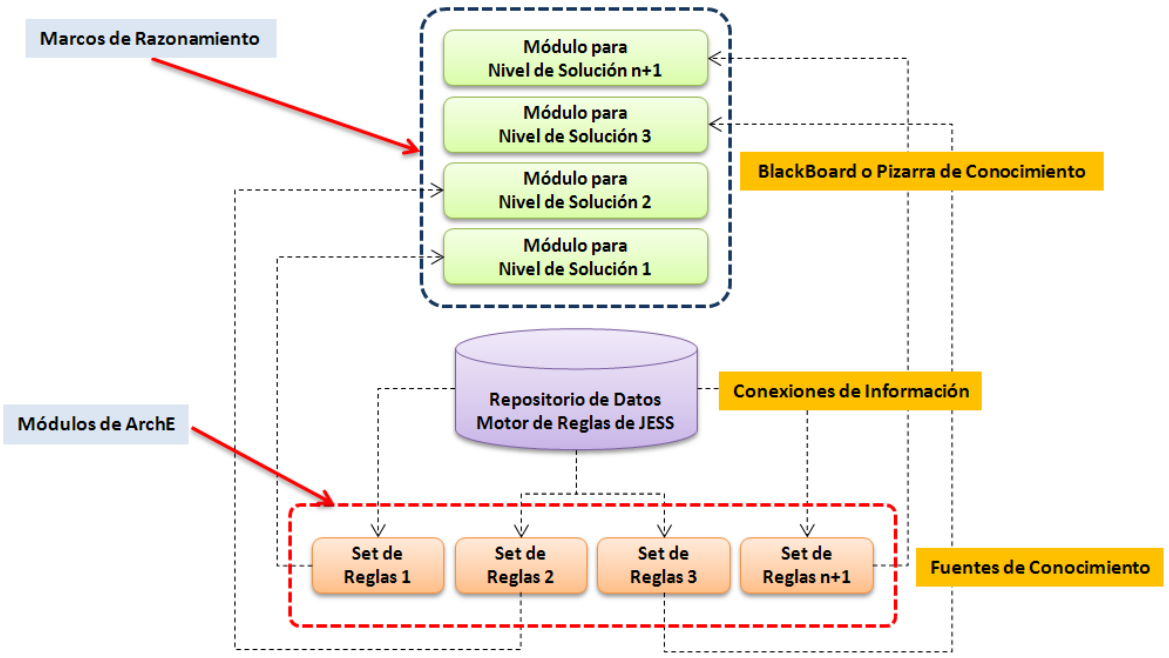
7.6.1. Estructura de ArchE

La Arquitectura ArchE fue diseñada como un componente o Plug-in de Eclipse. Al ser programada en este Framework de desarrollo, gran parte del desarrollo del SEI está basado en Java. ArchE como Sistema Experto utiliza para su ejecución al Sistema Experto JESS "***J**ava **E**xpert **S**ystem **S**hell*", un Shell o motor de reglas para plataformas Java.

ArchE fue diseñado bajo el patrón arquitectónico Blackboard (Fig.28). BlackBoard es un patrón arquitectónico que puede ser utilizado para la resolver problemas de tipo No Determinísticos con base en estrategias de solución ya conocidas. El sistema de divide en tres componentes, una llamada Blackboard "Pizarra", una colección de fuentes de conocimiento y un componente llamado control.

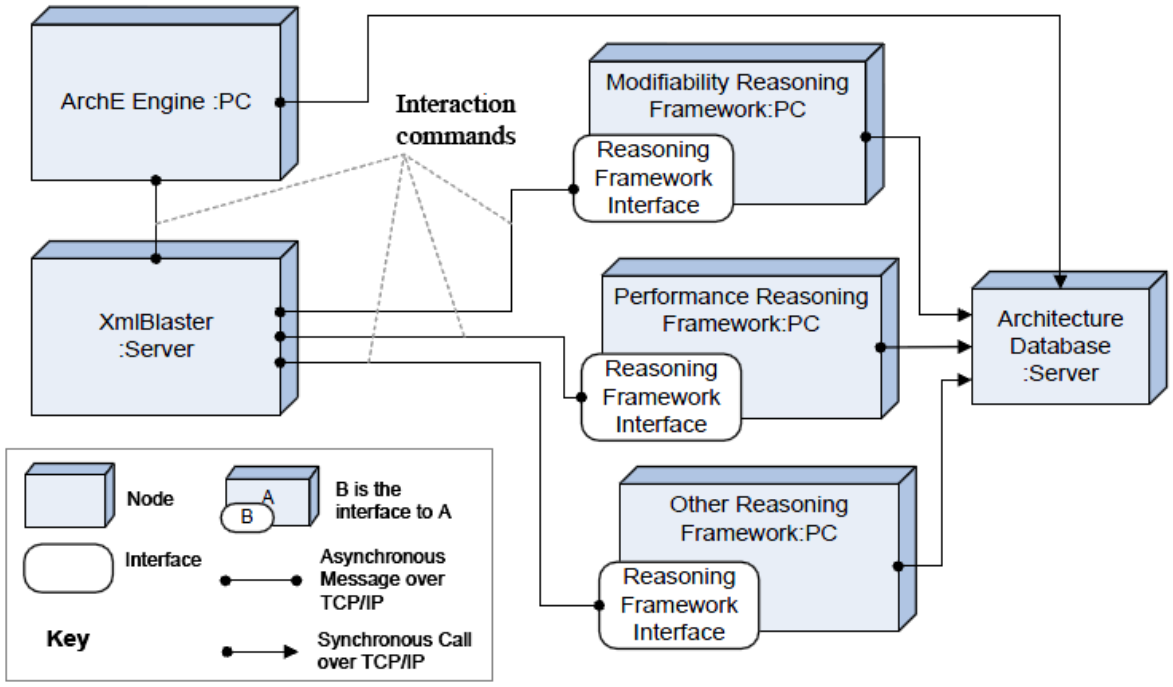
El Sistema Experto JESS, es el encargado de gestionar las estructuras de datos en la base de hechos y las reglas con las que ArchE interactúa con el razonamiento. La siguiente ilustración corresponde al diseño de la arquitectura ArchE en base al patrón Blackboard.

Figura 28. Arquitectura Blackboard para ArchE



En la siguiente imagen se muestra la interacción de los diferentes Frameworks, tanto internos como externos con el motor de Reglas de ArchE.

Figura 29. Interacción de Frameworks Internos y Externos en ArchE



Fuente: (Clements, y otros, 2003).

7.6.2. Especificación de Módulos en el Sistema Experto ArchE

A continuación se especificarán las características y funcionalidades más relevantes de los módulos de la arquitectura. La caracterización de los módulos se presentará en una lista de objetivos, reglas de interacción, secuencias de reglas, reglas de cómputo y salidas.

Tabla 20. Set de Reglas - Módulo Planner

	**DESCRIPCIÓN Y FUNCIONALIDAD**
OBJETIVOS (GOALS), RESPONSABILIDADES E INTERACCIONES	• Reunir los requerimientos en términos de escenarios • Interactuar con el arquitecto de software para mantener y priorizar la lista de escenarios para atributos de calidad. • Revisar la lista de escenarios uno a uno, sus argumentos y atributos iniciales necesarios para satisfacer un escenario determinado en base al Framework de Razonamiento utilizado. • Revisar las interacciones entre escenarios. • Comunicarse con el diseñador ante cualquier cambio sugerido por el arquitecto entre, durante o después de iniciar el proceso de modelado de los escenarios.

SECUENCIA DE REGLAS	• **AcquireRequirements** (Lista de Escenarios).
	• **RefineScenario** (Escenarios bien formados).
	• **ChooseReasoningFramework** (Determina que tipos de escenarios se encuentran actualmente en el modelo).
	• **BuildQualityAttributeModel** (Con base en el proceso de elección del escenario y sus atributos de calidad, selecciona el Framework de Razonamiento específico para cada caso.
	• **BuildDesign** (módulo que se encarga de traducir los elementos del modelo, relaciones, propiedades al modelo de ArchE. Al finalizar este proceso, el módulo muestra a través de la vista de diseño los escenarios en conjunto, para su análisis.
	
	Prioridad de Reglas
	
	`If (UnreadScenarios) then AcquireRequirements`
	`If (RawScenarios) then RefineScenario`
	`If (UnAttachedtoRF) then ChooseReasoningFramework`
	`If (UnmodelledScenarios) then BuildQualityAttributeModels`
	`If (UnDesignedScenarios) then BuildDesign`

CÓMPUTO DE REGLAS	Existencia y verificación de atributos.
	
	`If (ArchECompleted) then AcquireRequirements`
	;
	Revisión de nuevos requerimientos.
	;
	`(defrule CheckForNewRequirements`
	`(declare (auto-focus TRUE))`
	`()`
	`=>`
	`(focus AquireRequirements)`

REGLAS PARA MANIPULACIÓN DE LA BASE DE HECHOS	El módulo Planner no posee reglas para manipular la Base de Hechos.

Tabla 21. Set de Reglas – Módulo Acquire Requirements

	**DESCRIPCIÓN Y FUNCIONALIDAD**
OBJETIVOS (GOALS), RESPONSABILIDADES E INTERACCIONES	• La principal responsabilidad de este componente es ingresar los requerimientos y transformar los mismos en estructuras entendibles e interpretables por el motor de conocimiento
SECUENCIA DE REGLAS	• **AcquireRawScenarios** (este módulo se encarga de recuperar los escenarios que se han almacenado en un formato reconocible por la base de hechos y los muestra a través del formulario QAS o formulario para Atributos de Calidad.

	• **AcquireFunctionalRequirements** (este módulo se encarga de recuperar los requerimientos funcionales, abstrae las responsabilidades de dichos requerimientos e ingresa las responsabilidades a la estructura de responsabilidades de la base de hechos. Este módulo también se encarga de informar al usuario acerca de nuevas responsabilidades introducidas o generadas y de igual manera se encarga de gestionar la vista mediante la cual el arquitecto recupera las responsabilidades).
	
	Prioridad de Reglas
	
	`If (UnreadScenarios) then AcquireRawScenario`
	`If (UnreadRequirements) then`
	`AcquireFunctionalRequirements`

CÓMPUTO DE REGLAS	Existencia y verificación de atributos.

Tabla 22. Set de Reglas - Módulo Refine Scenario

	**DESCRIPCIÓN Y FUNCIONALIDAD**
OBJETIVOS (GOALS), RESPONSABILIDADES E INTERACCIONES	• Este componente se encarga de convertir los argumentos del escenario (mal formados) en un escenario refinado u optimizado caracterizando las entradas en las 6 partes para atributos de calidad de un escenario. • Clasifica las entradas o argumentos en un determinado conjunto de cualidades y lo asocia a un atributo de calidad en particular. • También se encarga de administrar las responsabilidades y sus interacciones que han sido introducidas por el diseñador.

SECUENCIA DE REGLAS	• DetermineScenarioType
	• IdentifyScenarioParts
	• IdentifyResponsibilities
	• DisplayResponsibilities (Corresponde a una implementación de un módulo que permite el despliegue de todas las responsabilidades y sus interacciones al usuario. Permite de igual manera que éste modifique algunos atributos como nombre de la responsabilidad y las relaciones entre responsabilidades).
	
	Prioridad de las Reglas
	
	`If (RawScenarios) then DetermineScenarioType;`
	`IdentifyScenarioParts;`
	`IdentifyResponsibilties; DisplayResponsibilities`

CÓMPUTO DE REGLAS	El resultado de refinar los escenarios contribuye a la formación de escenarios bien formados que posteriormente serán utilizados por la base de hechos. El escenario bien formado tendrá el siguiente formato

	de entrada para el análisis:
	
	`Scenario(1).Raw = "El sistema debe controlar los dispositivos externos bajo condiciones normales de carga y despliegue en un tiempo no mayor a 5 segundos "`
	`Scenario(1).State = "Parsed"`
	`Scenario(1).Quality = "Performance"`
	`Scenario(1).Type = "ICM"`
	`Scenario(1).Stimulus = "Dispositivo Externo"`
	`Scenario(1).Stimulus.Type = "Periodic"`
	`Scenario(1).Source = "Dispositivo Externo"`
	`Scenario(1).Context = "Normal"`
	`Scenario(1).Response = "Manejar y Operar"`
	`Scenario(1).Response_Measure = "TaskLatency, menor a 3.5 seg"`
	
	Las responsabilidades también son actualizadas en la base de hechos:
	
	`Responsibility(1).responsibility=Scenario.Stimulus(1)`
	`Responsibility(2).responsibility=cenario.Context(1)`
	`Responsibility(3).responsibility=Scenario.Response(1)`

REGLAS PARA MANIPULACIÓN DE LA BASE DE HECHOS	El módulo Refine Scenario no posee reglas para manipular la Base de Hechos.

Tabla 23. Set de Reglas - Módulo ChooseReasoningFrmework

	**DESCRIPCIÓN Y FUNCIONALIDAD**
OBJETIVOS (GOALS), RESPONSABILIDADES E INTERACCIONES	• Este módulo se encarga de construir el modelo para el cual se procesa cada escenario.

SECUENCIA DE REGLAS	• **DetermineScenarioType** (esta función incluye la identificación de "hard deadline" o límites de tiempo y costos de modificación. Los algoritmos implementados en el motor de reglas del marco de razonamiento y que calculan los tiempos límite corresponden a: planificación por prioridades fijas y ejecución cíclica.
	• **AssignReasoningFramework** (se encarga de diferenciar el mecanismo de cálculo en los marcos de razonamiento "prioridades fijas o ejecución cíclica").
	
	Prioridad de las Reglas
	
	`If (noScenariotype) then DetermineScenarioType`
	`If (noReasoningFramework) then AssignReasoningFramework`

CÓMPUTO DE REGLAS	El módulo solo se encarga de realizar el cálculo de tiempos en base a los algoritmos planificados en el marco de razonamiento para límites de tiempo y costos de modificación.

REGLAS PARA MANIPULACIÓN DE LA BASE DE HECHOS	El módulo ChooseReasoningFramework no posee reglas para manipular la Base de Hechos.

Tabla 24. Set de Reglas – Módulo DetermineScenario

	**DESCRIPCIÓN Y FUNCIONALIDAD**
OBJETIVOS (GOALS), RESPONSABILIDADES E INTERACCIONES	• Este módulo es invocado a través de un determinado escenario. Su principal característica es definir el tipo de escenario al cual pertenece el módulo. La versión 3.0 de ArchE solo cuenta con definición de tipos para escenarios de Performance y Modifiability (Deadline y costos de Modificabilidad).
SECUENCIA DE REGLAS	• Este módulo no reviste de sub procesos o secuencia de reglas.

CÓMPUTO DE REGLAS	• Este módulo no realiza cálculos, solo se especializa en designar el tipo de escenario donde se aplicarán los Reasoning Framework "ICM Performance y Modifiability"
	`If (scenario.quality = "Performance")`
	`  then scenario.type = "harddeadline"`
	`If (scenario.quality = "modifiability")`
	`  then scenario.type = "modifiability"`

REGLAS PARA MANIPULACIÓN DE LA BASE DE HECHOS	El módulo ChooseReasoningFramework no posee reglas para manipular la Base de Hechos.

Tabla 25. Set de Reglas - Módulo AssignReasoningFramework

	**DESCRIPCIÓN Y FUNCIONALIDAD**
OBJETIVOS (GOALS), RESPONSABILIDADES E INTERACCIONES	• Este módulo examina todos los escenarios e identifica cuales fueron asignados al marco de razonamiento ICM Performance y cuáles fueron asignados al marco de Modifiability. Esta división o selección de tipos de escenarios facilita el análisis al sistema experto JESS y la búsqueda en los motores de hechos y conocimiento, con base en un criterio establecido.
SECUENCIA DE REGLAS	• Este módulo no reviste de sub procesos o secuencia de reglas.

CÓMPUTO DE REGLAS	• Este módulo no realiza cálculos, solo se encarga de diferenciar los escenarios de acuerdo a los marcos de razonamiento ICM Performance y Modifiability.
	
	`If(TRUE)then`
	`  For each in (QueryFactBase(Scenario, Scenario.type-`
	`  "modifiability"))`
	`Scenario.ReasoningFramework = "modifiability"`
	`  If (TRUE) then`
	`  If (count of (QueryFactBase(Scenario, Scenario.type =`
	`"harddeadline"))>= 10)  then`
	`  For each in (QueryFactBase(Scenario, Scenario.type =`
	`"harddeadline"))`
	`  Scenario.ReasoningFramework = "Fixed-priority-`
	`scheduling"`
	`  If (count of (QueryFactBase(Scenario,`
	`Scenario.Stimulus.Type = "Sporadic") > 0 ) then`
	`  For each in (QueryFactBase(Scenario, Scenario.type =`
	`"harddeadline"))`
	`  Scenario.ReasoningFramework = "Fixed-priority-`
	`scheduling"`
	`  If (count of (QueryFactBase(Scenario,`
	`Scenario.Stimulus.Type = "Stochastic") > 0 ) then`
	`  For each in (QueryFactBase(Scenario, Scenario.type =`
	`"harddeadline"))`
	`  Scenario.ReasoningFramework = "Fixed-priority-`
	`scheduling"`
	`  If (Scenario.ReasoningFramework = "" AND`
	`"Scenario.type = "harddeadline")`
	`  then`
	`  Scenario.ReasoningFramework = "cyclic-executive"`
	
	Posteriormente a cada escenario se le asigna el respectivo algoritmo de cómputo
	
	`Scenario(1).ReasoningFramework ="fixed-priority-scheduling"`
	`Scenario(2).ReasoningFramework ="fixed-priority-scheduling"`
	`Scenario(3).ReasoningFramework ="modifiability"`
	`Scenario(4).ReasoningFramework ="modifiability"`

REGLAS PARA MANIPULACIÓN DE LA BASE DE HECHOS	El módulo AssignReasoningFramework no posee reglas para manipular la Base de Hechos.

Tabla 26. Set de Reglas - Módulo BuildQualityAttributeModel

	**DESCRIPCIÓN Y FUNCIONALIDAD**
OBJETIVOS (GOALS), RESPONSABILIDADES E INTERACCIONES	• Este modulo se encarga de crear el respectivo modelo en base a los escenarios descritos.
SECUENCIA DE REGLAS	• BuildFixedPrioritySchedulingModel • BuildCyclicExecutiveSchedulingModel • BuildModifiabilityModel

CÓMPUTO DE REGLAS	• Este módulo no realiza cálculos, solo se especializa en registrar los módulos que serán presentados como modelo en la vista de modelos de la arquitectura.
	
	`If (notconsistentmodel AND notinfiniteloop) then`
	`BuildFixedPrioritySchedulingModel`
	`BuildCyclicExecutiveSchedulingModel`
	`BuildModifiabilityModel`

REGLAS PARA MANIPULACIÓN DE LA BASE DE HECHOS	El módulo no posee reglas para manipular la Base de Hechos.

Tabla 27. Set de Reglas - Módulo BuildFixedPrioritySchedulingModel

	**DESCRIPCIÓN Y FUNCIONALIDAD**
OBJETIVOS (GOALS), RESPONSABILIDADES E INTERACCIONES	• Este modulo asigna las respectivas propiedades a las responsabilidades que han sido asociadas con los escenarios basados en límites fijos de tiempo “deadlines”.

SECUENCIA DE REGLAS	• DetermineBoundParametersforFixedPrioritySchedulingModel
	• InitialValuesofFreeParametersforFixedPrioritySchedulingModel
	• EvaluateFixedPrioritySchedulingModel
	• CreateFixedPrioritySchedulingModel
	
	Prioridad y secuencia de reglas:
	
	`If (FixedPrioritySchedulingResponsibilitiesIdentified) then`
	`DetermineBoundParametersforFixedPrioritySchedulingModel`
	`InitialValuesofFreeParametersforFixedPrioritySchedulingModel`
	`If (SchedulingModelNotSatisfied and`
	`FixedPrioritySchedulingParametersHaveValues) then`
	`CreateFixedPrioritySchedulingModel`
	`EvaluateFixedPrioritySchedulingModel`

CÓMPUTO DE REGLAS	• Este módulo no realiza cálculos, solo se especializa en registrar los módulos que serán presentados como modelo en la vista de modelos de la arquitectura.
	
	`If ( notconsistentmodel AND notinfiniteloop) then`
	`BuildFixedPrioritySchedulingModel`
	`BuildCyclicExecutiveSchedulingModel`
	`BuildModifiabilityModel`

REGLAS PARA MANIPULACIÓN DE LA BASE DE HECHOS	`If (TRUE) then`
	`Bind(QueryFactBase(Responsibilities deriving from`
	`Scenario.RF=”fixed-priority-scheduling”))`
	`If (Responsibility.schedulingpriority = “”) then`
	`Responsibility. schedulingpriority = “lowest”`

	Las responsabilidades también son actualizadas en la base de hechos:
	
	`Responsibility(1).responsibility = "Detect obstacle"`
	`Responsibility(1) ExecutionTime = 10ms`
	`Responsibility(1).schedulingpriority = 100`
	`Responsibility(3).responsibility = "Detect user request"`
	`Responsibility(3).ExecutionTime = 25ms`
	`Responsibility(3).schedulingpriority = 90.`

Tabla 28. Set de Reglas - Módulo CreatePrioritySchedulingModel

	**DESCRIPCIÓN Y FUNCIONALIDAD**
OBJETIVOS (GOALS), RESPONSABILIDADES E INTERACCIONES	• La principal tarea de este módulo es la creación de parámetros para los mecanismos de planificación de prioridades fijas. Este módulo analiza que escenarios se han satisfecho en base a las restricciones de límite de tiempo
SECUENCIA DE REGLAS	• Este módulo no reviste de sub procesos o secuencia de reglas.

CÓMPUTO DE REGLAS	• La creación de parámetros y prioridades está definida por el alcance de los escenarios en función de los deadline.
	
	`if (NOT UnsetParametersforFixedPriorityScheduling) AND`
	`GetSumofExecutionTimeofDeadlineScenarios LTE`
	`GetMinDeadlineofDeadlineScenarios) then`
	`AllocateScenariostoSingleUnitofConcurrency`
	`AllocateUnitsofConcurrencytoSingleProcessor`
	`else`
	`ApplyTacticIncreaseLogicalConcurrency`
	`AllocateUnitsofConcurrencytoSingleProcessor`
	`if (NOT UnsetParametersforFixedPriorityScheduling AND`
	`GetSumofExecutionTimeofDeadlineScenarios GT`
	`GetMinDeadlineofDeadlineScenarios)) then`
	`ApplyTacticBoundExecutionTime`

REGLAS PARA MANIPULACIÓN DE LA BASE DE HECHOS	El módulo no posee reglas para manipular la Base de Hechos.

Tabla 29. Set de Reglas – Módulo BuildModifiabilityModel

	**DESCRIPCIÓN Y FUNCIONALIDAD**
OBJETIVOS (GOALS), RESPONSABILIDADES E INTERACCIONES	• Este módulo se encarga de crear el respectivo modelo de Modificabilidad para el respectivo escenario basado en el atributo de calidad de Modificabilidad. La principal tarea del módulo es

	asignar las responsabilidades proporcionadas por el razonamiento "Marcos de Razonamiento" y procesa las mismas en un solo módulo.

SECUENCIA DE REGLAS	• DetermineBoundParametersforModifiabilityModel
	• InitialValuesofFreeParametersforModifiabilityModel
	• EvaluateModifiabilityModel
	• CreateModifiabilityModel
	
	Prioridad y secuencia de reglas:
	
	`If (ModifiabilityResponsibilitiesIdentified) then`
	`DetermineBoundParametersforModifiabilityModel`
	`InitialValuesofFreeParametersforModifiabilityModel`
	`If (ModifiabilityScenariosNotSatisfied and`
	`ModifiabilityParametersHaveValues) then`
	`CreateModifiabilityModel`
	`EvaluateModifiabilityModel`

CÓMPUTO DE REGLAS	• Depende de las reglas de cómputo establecidas por el marco de razonamiento de Modifiability.

REGLAS PARA MANIPULACIÓN DE LA BASE DE HECHOS	`If (TRUE) then`
	`Bind(QueryFactBase(Responsibilities deriving from`
	`Scenario.RF="modifiability"))`
	`If (Responsibility.moduleassignment = "") then`
	`Responsibility.moduleassignment = "other"`
	
	Las responsabilidades también son actualizadas en la base de hechos
	
	`Responsibility(4).responsibility = "processor independent"`
	`Responsibility(4) module = "processor independent"`
	`Responsibility(5).responsibility = "convert from processor`
	`independent to processor dependent"`
	`Responsibility(5).module = "virtual machine"`

Tabla 30. Set de Reglas - Módulo EvaluateModifiabilityModel

	**DESCRIPCIÓN Y FUNCIONALIDAD**
OBJETIVOS (GOALS), RESPONSABILIDADES E INTERACCIONES	• Evaluar y estimar los costos para los escenarios en consideración.
SECUENCIA DE REGLAS	• Este módulo prioriza la ejecución de las estimaciones en base al cómputo de reglas. No reviste de sub-procesos o secuencias de reglas.
CÓMPUTO DE REGLAS	• El costo de realizar un cambio específico a una responsabilidad se

	puede estimar a partir de las entradas del usuario o por entradas introducidas por el proceso mismo. • ArchE captura los eventos que producen efectos secundarios y efecto de onda "Ripple Efects" en el modelo a partir de los cambios realizados y el cómputo de sus dependencias.
REGLAS PARA MANIPULACIÓN DE LA BASE DE HECHOS	Las reglas para estimar el costo de un cambio en particular se calculan de la siguiente manera: • El arquitecto introduce el costo del cambio en otra responsabilidad asociada a la responsabilidad que está siendo modificada a la cual se le asocia una probabilidad e acuerdo al impacto del cambio agregado. • ArchE agrega los costos de estimación y realiza los respectivos cambios a las responsabilidades que afectan a otros módulos o dependencias que se unen a la responsabilidad en modificación. Los cambios dependen del cálculo de efecto de onda, el tipo de cambio sugerido, el tipo de dependencia a modelar y la probabilidad de cambio. • ArchE verifica a partir de estos cambios la propagación de los mismos entre dependencias y si es posible sugiere nuevos cambios para calcular nuevamente probabilidades.

Tabla 31. Set de Reglas - Módulo CreateModifiabilityModel

	**DESCRIPCIÓN Y FUNCIONALIDAD**
OBJETIVOS (GOALS), RESPONSABILIDADES E INTERACCIONES	• Este módulo de encarga de crear un conjunto de parámetros para un escenario de Modificabilidad. En este modelo basado en el atributo de calidad de Modificabilidad, este conjunto de parámetros deben servir como base para posteriores análisis y evaluación para determinar en cada iteración si el modelo se satisface de acuerdo a las restricciones dispuestas en los escenarios de Modificabilidad.
SECUENCIA DE REGLAS	• Este módulo no reviste de sub procesos o secuencia de reglas.
CÓMPUTO DE REGLAS	• Esta primera regla evalúa si se cumple o no el escenario a partir del costo de modificabilidad. Si la suma del costo de las responsabilidades es mayor que la respuesta de la medida, el escenario no se cumple. `If (Scenario.responsibility.cost > Scenario.responsemeasure) then ReportFailure` • A continuación, el motor de reglas evalúa varias tácticas de modificabilidad. Por defecto define un valor de desviación del 10% del total del costo estimado.

	If (MultipleResponsibilitiesAssignedSameModule AND ResponsibilityInteraction GT 10% * TotalCost) then
	ApplyTacticSemanticCohesion
	If (RippleCostFromModuleAtoB GT 10% * TotalCost) then
	ApplyTacticInsertIntermediary
	ApplyTacticHideInformation
	If (CostSingleResponsibilityModification GT 10% * TotalCost) then ApplyTacticRaiseAbstractionLevel
	If (ResponsibilityOverlap) then
	ApplyTacticAbstractCommonServices

REGLAS PARA MANIPULACIÓN DE LA BASE DE HECHOS	El módulo no posee reglas para manipular la Base de Hechos.

Tabla 32. Set de Reglas – Módulo BuildDesign

	**DESCRIPCIÓN Y FUNCIONALIDAD**
OBJETIVOS (GOALS), RESPONSABILIDADES E INTERACCIONES	• Este módulo de encarga de construir el modelo a partir de las responsabilidades. Cada diseño corresponde a un módulo central al cual se asocia un conjunto de responsabilidades y sus dependencias. Una vez concebido el modelo, este se exporta a la vista de Diseño que muestra el diagrama UML del modelo en cuestión.
SECUENCIA DE REGLAS	• Este módulo no reviste de sub procesos o secuencia de reglas.
CÓMPUTO DE REGLAS	• No realiza cómputo de reglas, solo limita su trabajo a procesar la vista de diseño (Modelo Sugerido).
REGLAS PARA MANIPULACIÓN DE LA BASE DE HECHOS	El módulo no posee reglas para manipular la Base de Hechos.

Tabla 33. Módulo Utilitario - DisplayDesign

	**DESCRIPCIÓN Y FUNCIONALIDAD**
OBJETIVOS (GOALS), RESPONSABILIDADES E INTERACCIONES	• La tarea de este módulo corresponde al despliegue en tiempo de diseño del modelo que se está evaluando. Este conjunto de vistas permiten al arquitecto modificar cualquier requerimiento o escenario a través de interfaces intuitivas (formularios, grillas de datos o los mismos escenarios) de tal manera que estos cambios afecten a los demás requerimientos o vistas (por ejemplo, un cambio en el costo de execution time, automáticamente hará que el

	framework de razonamiento actúe respecto a la modificación y dicho cálculo hará que los cómputos de los escenarios de calidad cambien de forma automática, tanto en diseño como en probabilidad). En este conjunto de vistas, el diseñador tiene la posibilidad de observar las tácticas aplicadas por ArchE, su impacto, costos asociados y modelos iterados.
SECUENCIA DE REGLAS	• Este módulo no reviste de sub procesos o secuencia de reglas.
CÓMPUTO DE REGLAS	• No realiza cómputo de reglas, solo limita su trabajo a desplegar la vista de diseño (Modelo Sugerido).
REGLAS PARA MANIPULACIÓN DE LA BASE DE HECHOS	El módulo no posee reglas para manipular la Base de Hechos.

Tabla 34. Módulo Utilitario - DisplayResponsabilities

	**DESCRIPCIÓN Y FUNCIONALIDAD**
OBJETIVOS (GOALS), RESPONSABILIDADES E INTERACCIONES	• Despliega el conjunto de responsabilidades y sus relaciones para el modelo actual en tiempo de ejecución y diseño. El arquitecto puede modificar parámetros como nombres de responsabilidades, descomposición de responsabilidades o especificar las relaciones entre estas responsabilidades.
SECUENCIA DE REGLAS	• Este módulo no reviste de sub procesos o secuencia de reglas.
CÓMPUTO DE REGLAS	• No realiza cómputo de reglas, solo limita su trabajo a desplegar la vista de diseño (Modelo Sugerido).
REGLAS PARA MANIPULACIÓN DE LA BASE DE HECHOS	El módulo no posee reglas para manipular la Base de Hechos.

8

CONFIGURACIÓN DE LA ARQUITECTURA .:.ArchE.:.

"Menos del 10% del código tienen que ver directamente con el propósito del sistema; el resto tiene que ver con la entrada y salida, validación de datos, mantenimiento de estructuras de datos y otras labores domésticas" - Mary Shaw

La Arquitectura Experta ArchE es una herramienta desarrollada por el SEI, la cual está soportada por el sistema experto JESS y plataformas Open Source como Java Eclipse y MySQL. ArchE ayuda en buena medida a resolver problemas de tipo arquitectónico, de diseño y modelado a través de la aplicación de un conjunto de métricas o tácticas que incrementan la performance y el rendimiento de los diseños.

ArchE se fundamenta en la especificación de Atributos de Calidad que se obtienen a partir de un conjunto de requerimientos (escenarios, funciones, responsabilidades, relaciones y tácticas) que son procesados por el motor de inferencia del sistema experto a través de un conjunto de Reglas y Goals.

Antes de iniciar con el proceso de análisis con ArchE, se precisa la instalación de los siguientes componentes o paquetes de software. El orden estricto responde a las características que deben ser instaladas en forma jerárquica para que la herramienta funcione de manera adecuada. Ellos son:

1) Java JDK 1.5 o JSDK jdk-1_5_0_22-windows-i586-p. (No se recomiendan otras máquinas virtuales o versiones posteriores al JDK 1.5).

2) IDE Elipse Europa SDK 3.3. Eclipse-SDK-3.3-win32.zip o el compilado que puede ser descargado directamente de la página del Máster. Eclipse.zip (188 MB).

3) Motor de Bases de Datos MySQL. Disponible desde la página del Máster a través del paquete de instalación Mysql-essential-5.0.67-win32.msi.

4) GEF (Graphical Editing Framework), un plug-in de Eclipse para el desarrollo de editores visuales. GEF utiliza todo el potencial de procesadores de texto basados en wysiwyg y la capacidad de modelado UML a través del uso de diagramas e interfaces gráficas para el usuario (GUI), entre otras características. Su descarga igualmente procede a través del repositorio del Máster. GEF-runtime-3.3.zip.

5) Sistema Experto JESS. El fichero corresponde a la versión 7.1. JESS está totalmente desarrollado en Java. Descárguelo a través del siguiente enlace en el hipervínculo del archivo Jess71p1.zip. (Es importante resaltar que este paquete requiere de licencia para poder ser utilizado).

6) Gestor de Comunicaciones XmlBlaster. Corresponde a un parser basado en XML que sirve como puente para la comunicación de ArchE con las demás herramientas (JESS, MySQL y MAST "Modelling and Analysis Suite for Real-Time Application"). Puede ser descargada desde la página oficial http://www.xmlblaster.org/. Se recomienda descargar la versión xmlBlaster_REL_1_6_1.zip.

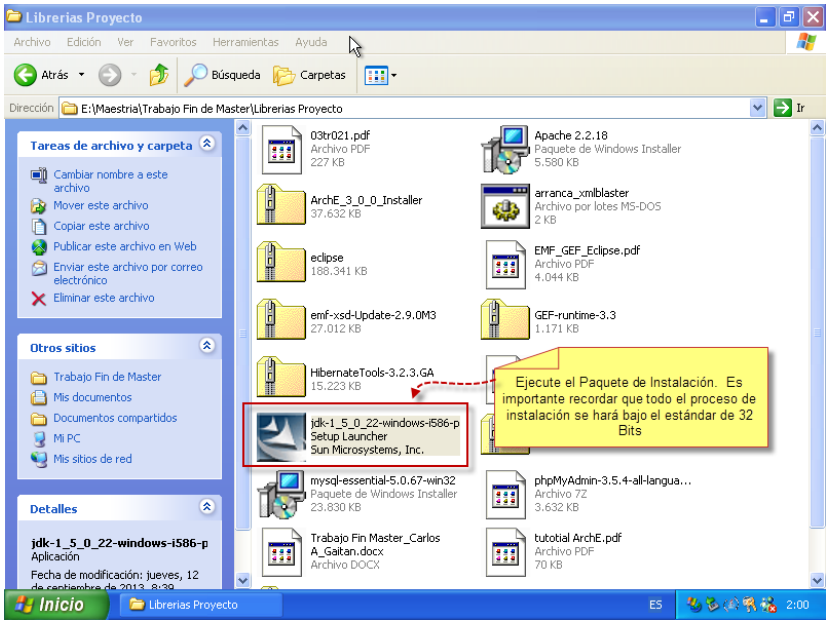
7) Por último se instalará ArchE_3_0_0_Installer.zip.

A continuación se describe en detalle el proceso de instalación de cada uno de los paquetes para un correcto funcionamiento de ArchE. Se recomienda al lector, que siga en forma estricta los pasos ya que el proceso de configuración es un poco engorroso y de no configurarse adecuadamente los paquetes, se corre el riesgo de fallos o inoperancia del sistema ArchE.

8.1. INSTALACIÓN DEL JDK

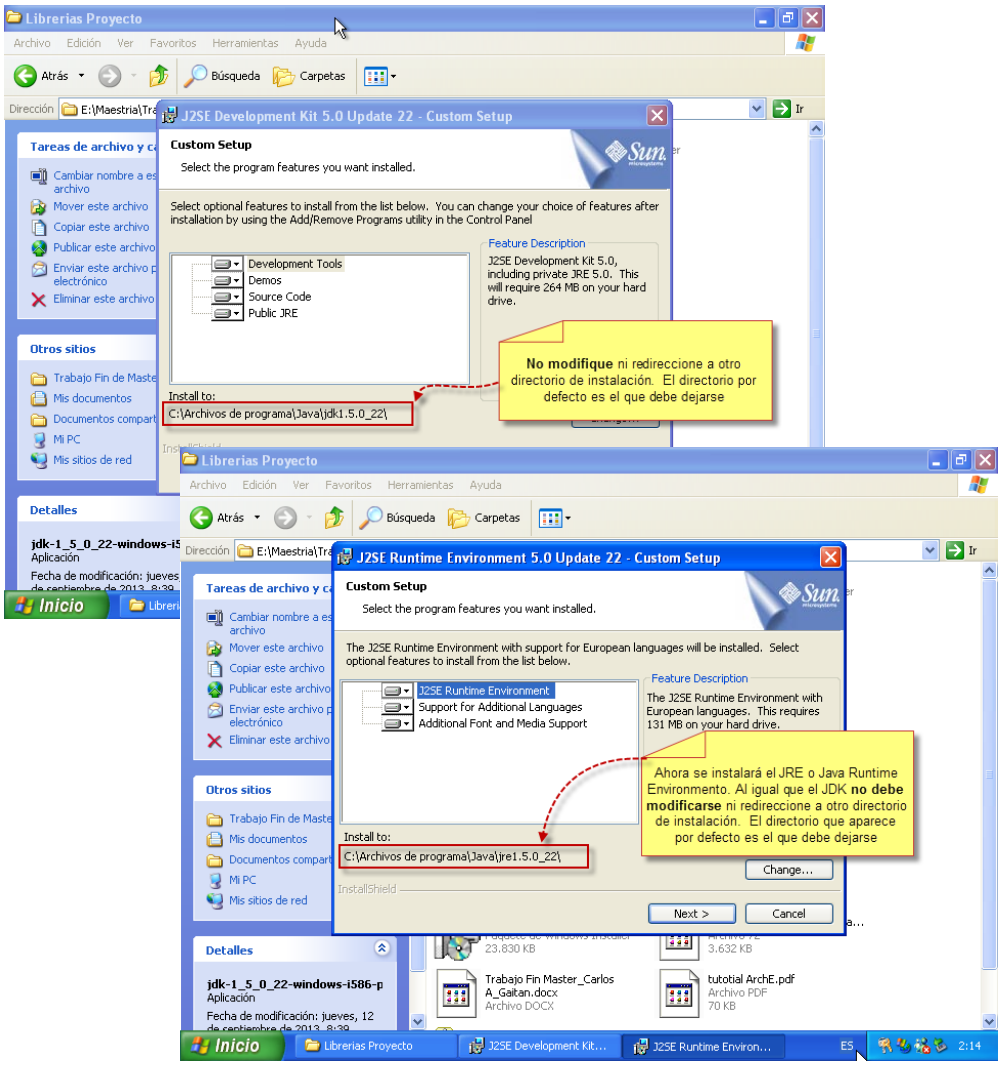
Seleccione el paquete de instalación del JDK. Recuerde que la versión de ArchE ha sido probada con el JDK 1.5. Asegúrese que no existan otras JDK instaladas en su computadora. Esta instalación debe realizarse por defecto.

Figura 30. Paquete de Instalación para Java JDK



Una vez iniciado el instalador, le mostrará la carpeta y las opciones de instalación. Asegúrese de instalar todos los componentes y dejar tal y como está el directorio que viene por defecto. Por lo general, la ruta corresponde a C:\Archivos de Programa\Java\JDK1.5.0_22 y C:\Archivos de Programa\Java\JRE1.5.0_22. En la primera ruta se almacenarán las clases y las bibliotecas propias para la ejecución de programas basados en Java y en el directorio del JRE se almacenarán los respectivos archivos de configuración y runtime (máquina virtual) para las aplicaciones con soporte de Java.

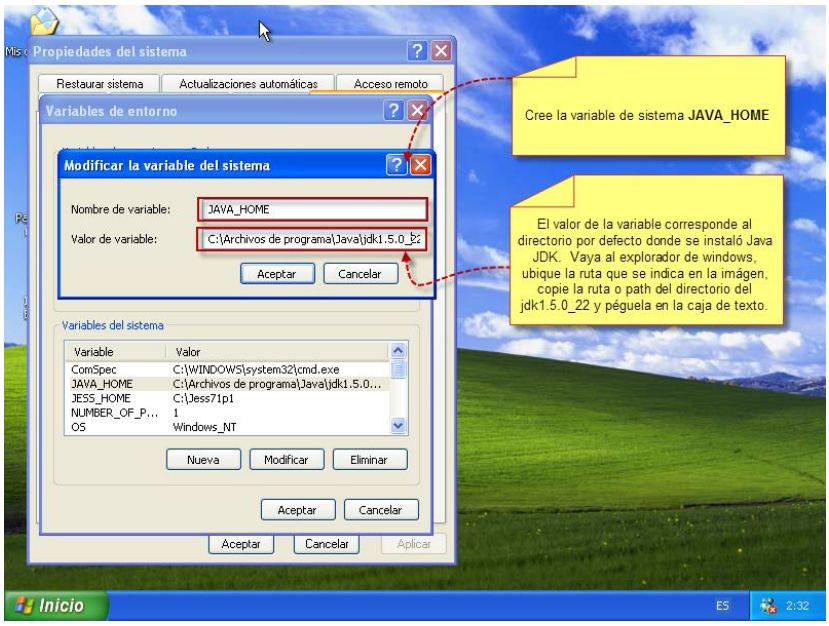
Figura 31. Directorios de instalación para Java JDK



Una vez instalado el JDK y JRE, se deben configurar las variables Path y Java_Home.

JAVA_HOME, es una variable de entorno del sistema que informa al sistema operativo sobre la ruta donde se encuentra instalado Java. Para acceder a las variables de entorno ubique el ícono de "Mi PC" en el escritorio. Selecciónelo y presione click derecho. Vaya a la opción **Propiedades**, luego presione la pestaña de **Opciones Avanzadas** y por último ubique el botón denominado **Variables de Entorno**.

Figura 32. Variables de Sistema para JAVA_HOME



El valor de la variable para el sistema bajo el cual se está instalando, se encuentra en la ruta ***C:\Archivos de Programa\Java\jdk1.5.0_22.*** En otros sistemas XP y sus variantes, es posible que la ruta de Java esté en paths diferentes al mencionado en este tutorial. Si este es su caso, ubique la carpeta en *C:\Program Files\Java\ jdk1.5.0_22* ó *C:\Program Files (x86)\Java\ jdk1.5.0_22*.

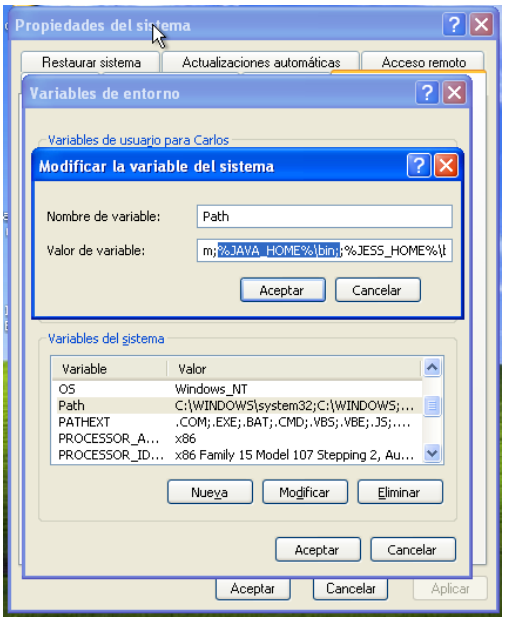
El siguiente paso corresponde a la configuración de la variable Path.

La variable **PATH** permite que el sistema operativo ejecute uno o más paquetes a través de una secuencia de comandos (por lo general de tipo Shell o DOS). Con la definición de la variable PATH, se garantiza que aquellos aplicativos compilados en un lenguaje determinado, puedan ser ejecutados en tiempo real. En el caso de ArchE, el cual está basado en JAVA, es necesario que esta característica se configure debido a que todas las bibliotecas y ejecutables .jar necesitan del compilador de clases **Javac.exe** y el intérprete **Java.exe** para ser ejecutados.

Para acceder a la variable PATH ubique el ícono de "Mi PC" en el escritorio. Selecciónelo y presione click derecho. Vaya a la opción **Propiedades**, luego presione la pestaña de **Opciones Avanzadas** y por último ubique el botón denominado **Variables de Entorno**. En las variables de sistema ubique la variable Path y presione el botón **Modificar**. Se desplegará una ventana denominada "Modificar la variable del sistema".

En este cuadro de diálogo va a escribir la siguiente cadena al final del valor de variable que encuentre en la misma. El valor a introducir es: **%JAVA_HOME%\bin;** (No olvide el **;** al final). Esto forzará al sistema operativo a compilar las clases directamente desde el directorio de origen.

Figura 33. Configuración de la Variable PATH para el JDK



Para comprobar que las variables han sido creadas satisfactoriamente, ejecute el editor de comandos del DOS o CMD "Command.COM" del sistema operativo. Vaya a Inicio – Ejecutar. En la caja de texto escriba **cmd**. Esto llamará al editor de comandos. A continuación escriba el comando **path**.

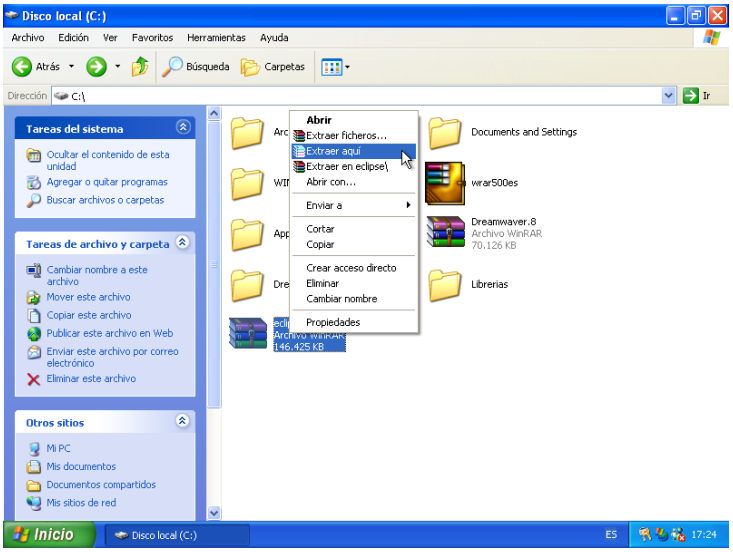
Figura 34. Comprobación de la Variable PATH para JAVA

Recuerde que es aconsejable no tener instaladas diversas versiones de máquina virtual. Es un detalle importantísimo a la hora de configurar ArchE ya que este solo es compatible con el JDK1.5 e incompatible con las API de otras versiones de JVM.

8.2. INSTALACIÓN IDE ECLIPSE

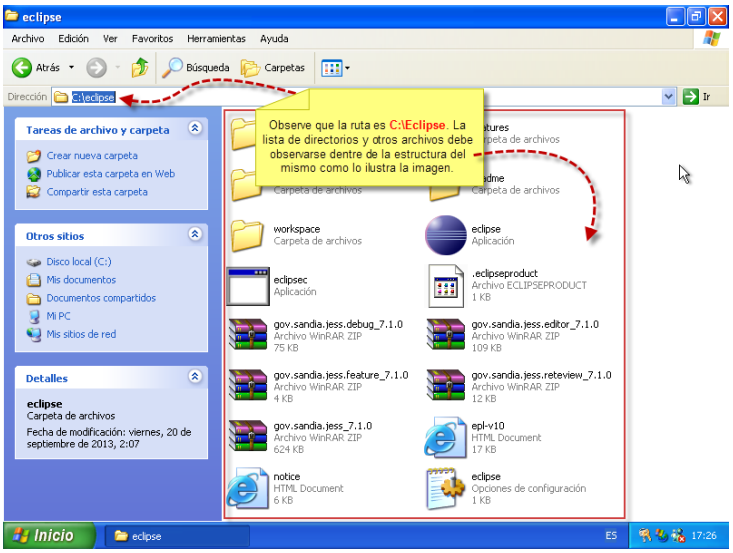
Para el IDE de Eclipse no se requiere proceso de instalación alguna. Descomprima el contenido del archivo en una carpeta y ya estará instalado el entorno de desarrollo.

Figura 35. Instalación del Entorno de Desarrollo Eclipse



Verifique que el contenido del archivo comprimido o se halla ubicado correctamente en el directorio C:\Eclipse.

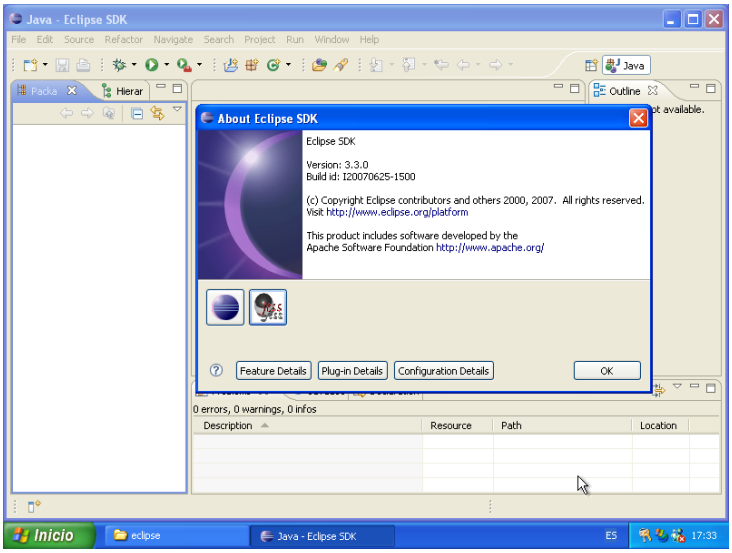
Figura 36. Directorio de Instalación y contenidos para el IDE Eclipse



Inicie la aplicación con el respectivo ejecutable. Recuerde que para el caso de ArchE, el IDE que ha sido probado corresponde al Eclipse Europa. Una vez ejecutado el entorno, verifique la integridad de la aplicación, atajos, configuración de paquetes o módulos, etc. Los íconos en el cuadro de diálogo acerca de… muestran algunos componentes (plug-ins) instalados. Un ejemplo de ellos es el JESS que ya se ha integrado a la herramienta de desarrollo.

Figura 37. Iniciando el IDE de Eclipse Europa

Figura 38. Cuadro de Diálogo Acerca de para el IDE de Eclipse

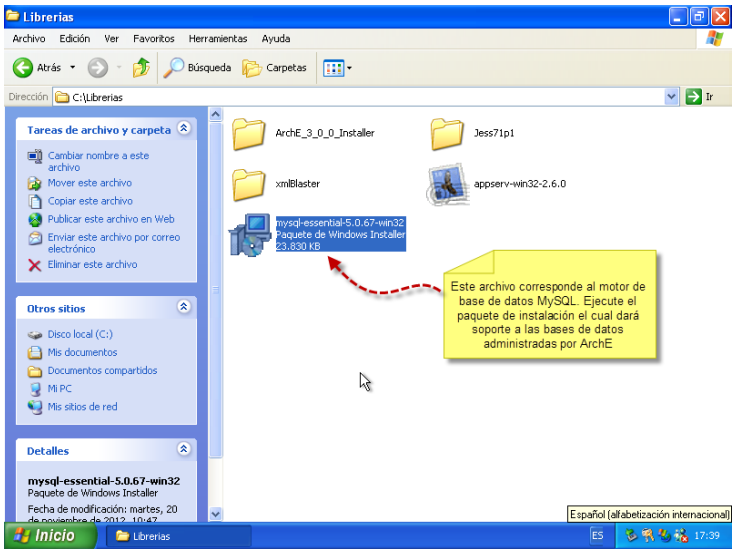


8.3. INSTALACIÓN DE MySQL DATABASE

Es un ***sistema administrativo relacional de bases de datos*** (RDBMS por sus siglas en ingles *Relational Database Management System*). MySQL es un servidor multi-usuarios de rápida ejecución con una excelente interfaz para la administración de transacciones en paralelo a lo largo de una red local o Internet. El proceso de instalación es sencillo pero habrá que configurarse ciertos parámetros para que ArchE funcione adecuadamente con la base de datos ya que ArchE registra en sí misma 105 tablas y 2 vistas que sirven como soporte para guardar o recuperar algunos datos necesarios en la ejecución y análisis de la herramienta ArchE.

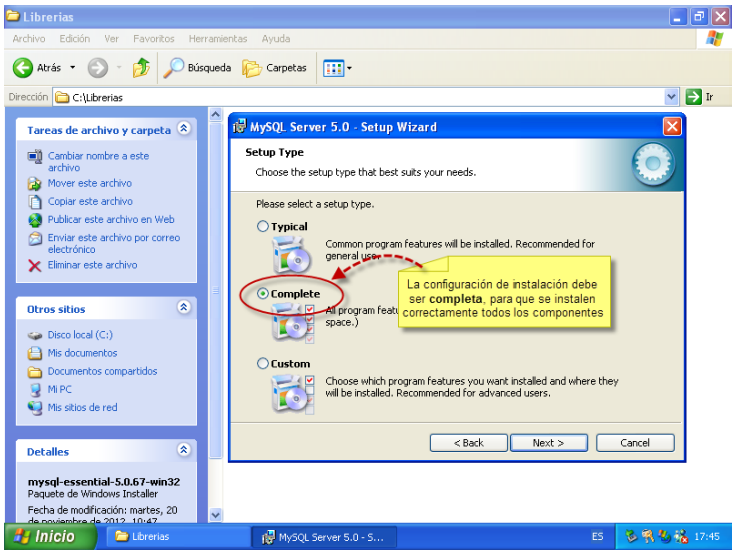
El proceso de instalación se inicia con la descarga del paquete MySQL essential 5.0.67 para plataformas Win32. Es importante tener en cuenta que ArchE solo funciona con plataformas de 32 Bits, ya que no se han documentado casos exitosos con plataformas de 64 Bits. Para instalar la base de datos relacional, ejecute el instalador y continúe con el protocolo de instalación.

Figura 39. Paquete de Instalación MySQL Database



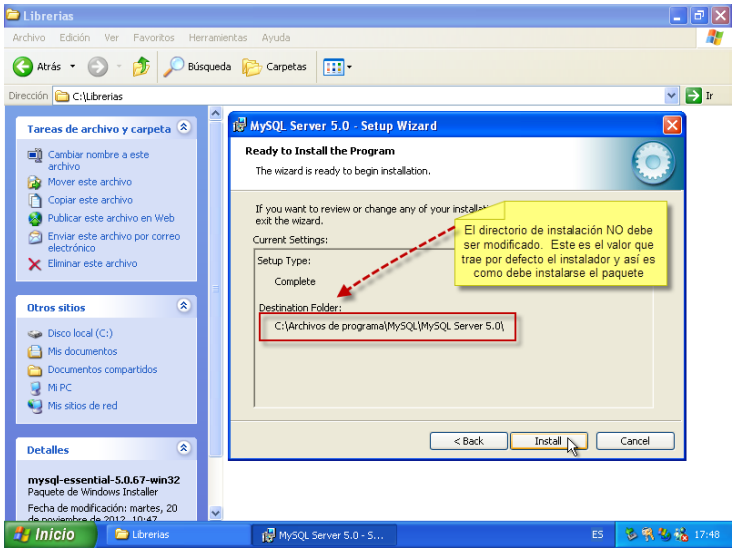
El proceso se ejecutará a través de una interfaz tipo Wizard. El proceso debe iniciarse en modo completo para instalar todos los componentes y para ampliar el performance del motor de gestión.

Figura 40. Tipo de Instalación para MySQL



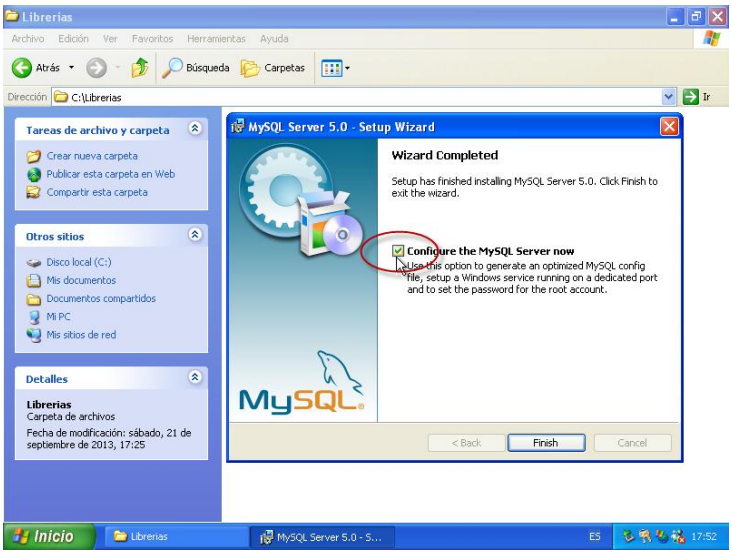
A continuación el instalador pedirá que confirme el directorio donde se instalará MySQL. El Directorio no debe ser modificado pues el instalador de ArchE buscará esta ruta por defecto para completar una lista de prerrequisitos antes de instalarse el sistema en sí.

Figura 41. Directorio de Instalación para MySQL



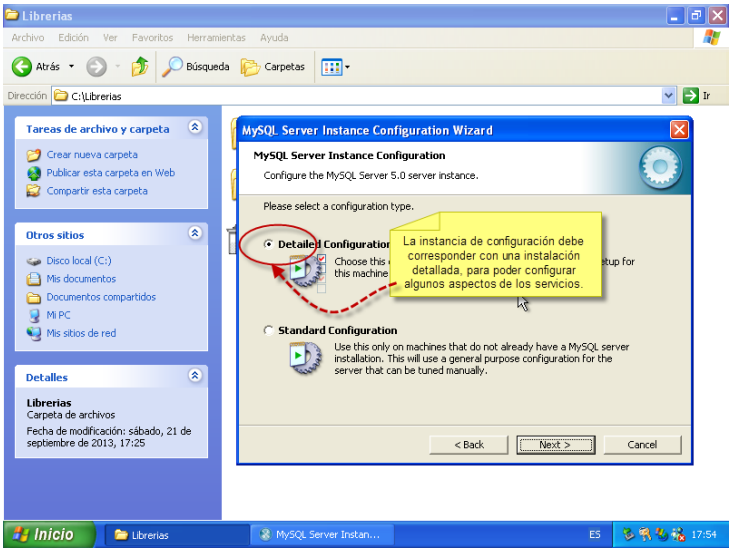
Una vez procesada la ruta de instalación por defecto, el sistema instalará todos los componentes y como paso adicional pedirá que se confirmen los parámetros del servidor. Para ello se desplegará la siguiente ventana (Parámetros del Servidor).

Figura 42. Parámetros y Configuración del Servidor



Seleccione a continuación el modo de configuración detallada. Esto hará que el asistente permita al usuario administrar y configurar sus propios parámetros para la instancia del servidor MySQL.

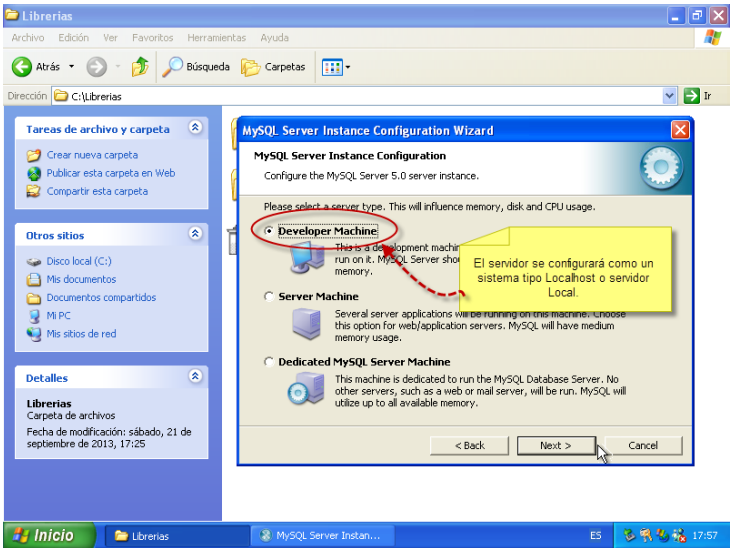
Figura 43. Instancia y Configuración Detallada del Servidor



El siguiente paso consiste en configurar la forma en cómo se iniciarán los servicios en el servidor. La configuración recomendada para utilizar ArchE con base en la instancia del servidor creada, corresponde a la opción “Developer Machine”, la cual configurará los

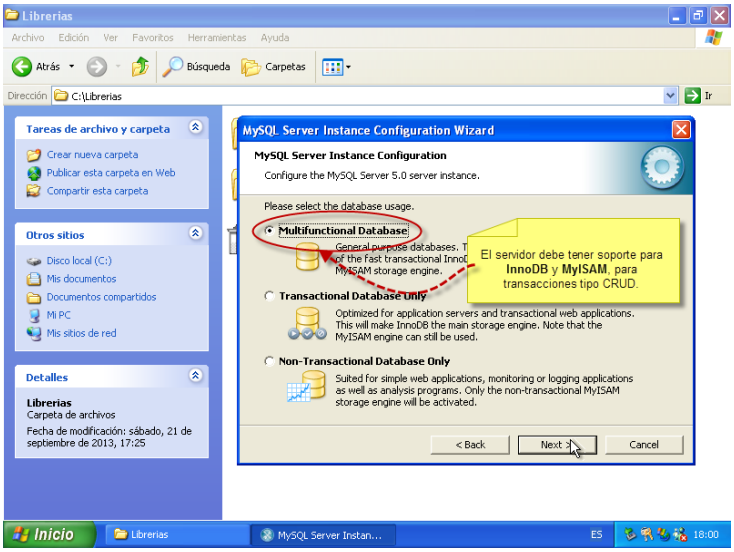
servicios y el localhost a través de los cuales se comunicarán los paquetes y sockets en la red local.

Figura 44. Configuración del Localhost



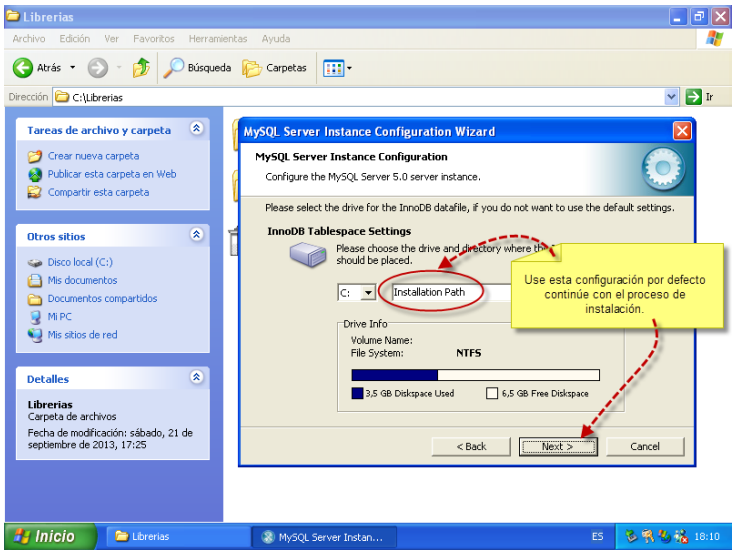
A continuación el instalador pedirá al usuario que configure el tipo de persistencia a los datos y el soporte de integridad para las bases de datos.

Figura 45. Configurando soporte para InnoDB y MyISAM en MySQL



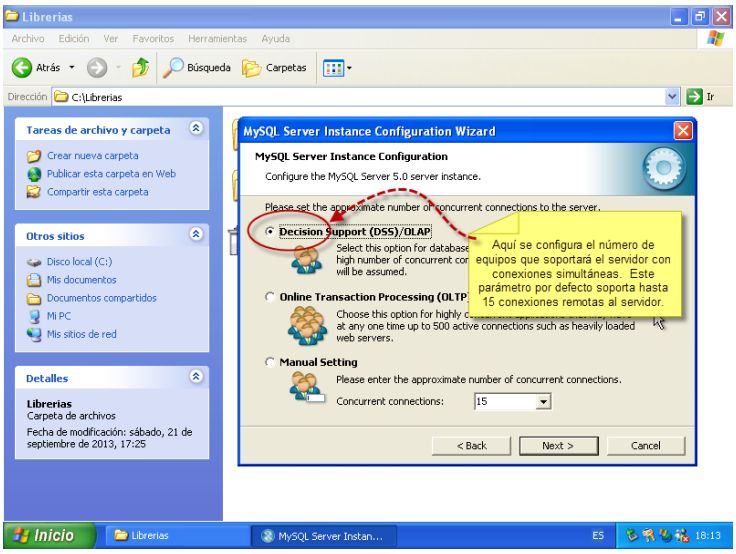
Ahora se configurará el Tablespace para MySQL. Un tablespace es una unidad lógica de almacenamiento dentro de una base de datos, la cual sirve como puente entre el sistema de ficheros del sistema operativo y las bases de datos alojadas en el servidor.

Figura 46. Configuración del Tablespace InnoDB para MySQL



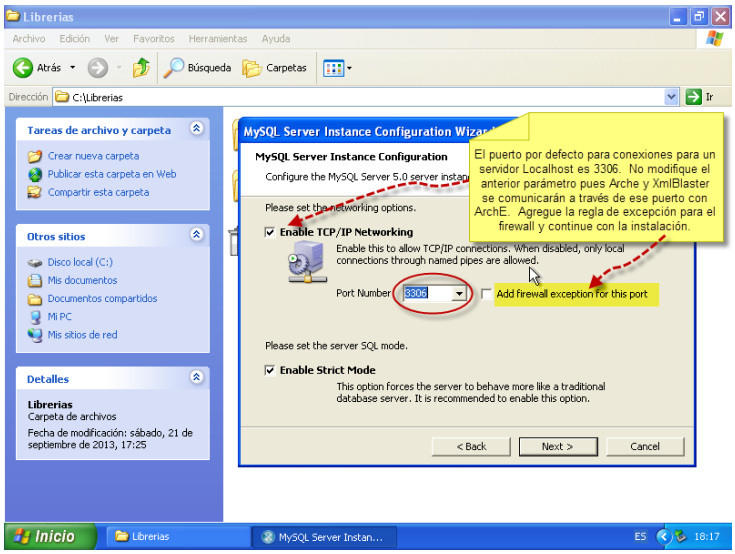
A continuación, se configurará el número de conexiones simultáneas que podrá soportar el servidor. Seleccione por defecto a OLAP Decision Support.

Figura 47. Configuración de Conexiones Simultáneas para MySQL



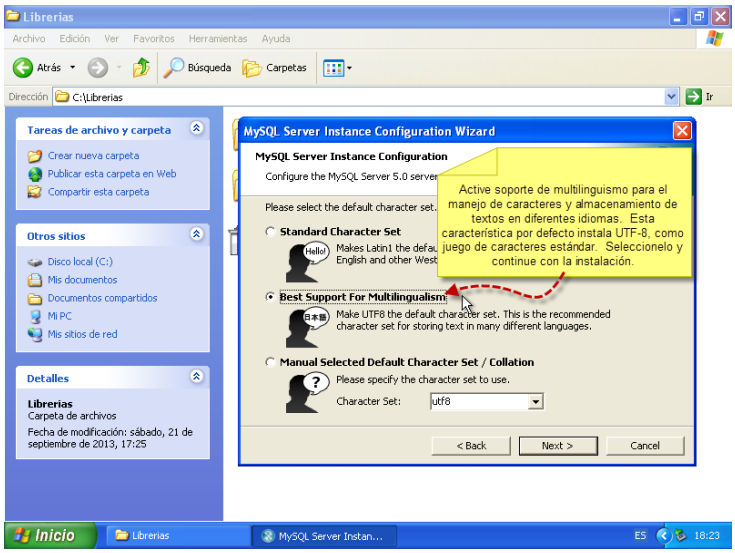
A partir de este momento se configurará el puerto TCP para el localhost. Por defecto el puerto 3306 es el utilizado por MySQL. Complete las opciones tal y como se detalla en la siguiente Figura y continúe con la instalación.

Figura 48. Soporte para Networking y TCP/IP en MySQL



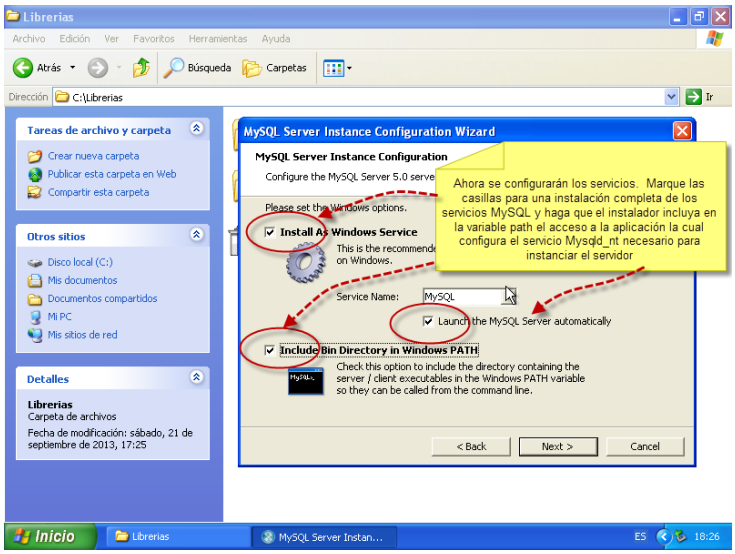
El siguiente paso es configurar el juego de caracteres (soporte de idiomas) que utilizará MySQL en sus bases de datos. Seleccione la opción de Multilingüismo.

Figura 49. Juego de Caracteres y Soporte de Idiomas en MySQL



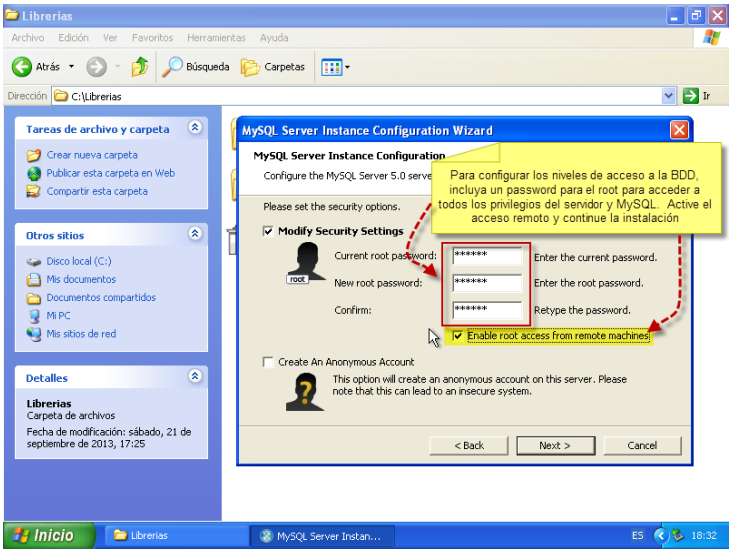
El siguiente paso configurará el path para la variable MySQL en el sistema operativo, el nombre de la instancia del servidor y el servicio de MySQL como tal. Configúrelo tal y como se indica a continuación.

Figura 50. Iniciación de Servicios e Instancia del Servidor



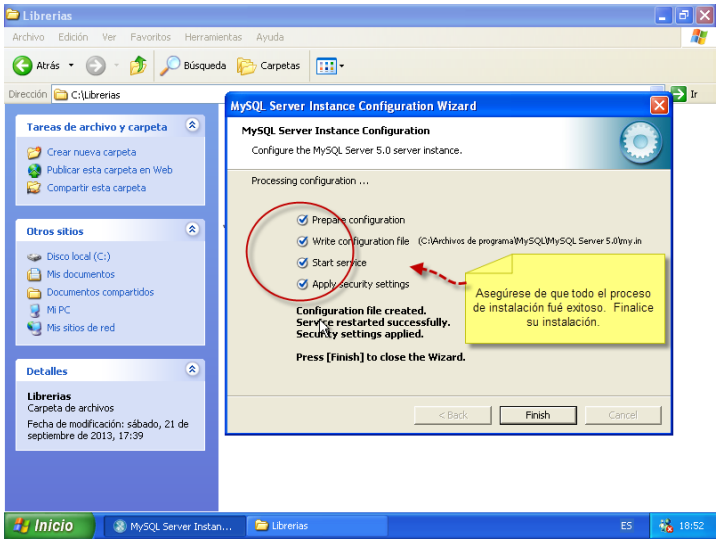
A continuación se configurará el sistema de autenticación al servidor.

Figura 51. Configuración de Seguridad para MySQL



Finalizado los parámetros de seguridad, el sistema procederá a ejecutar la configuración.

Figura 52. Finalizando la Configuración del Servidor



Para verificar que los servicios han sido iniciados correctamente, se verificará en el árbol de procesos la ejecución de la API de MySQL y la ejecución del servicio MySQL por DOS. La Fig. 53 muestra el servicio **Mysqld_nt.exe** registrado, el cual se encuentra en ejecución. La Fig. 54 ilustra el acceso al servidor a través del editor de comandos del DOS, sobre el cual se podrán ejecutar comandos propios de MySQL y manipular las bases de datos a través de lenguajes de consulta como SQL.

Figura 53. Ejecución de Servicios para MySQL

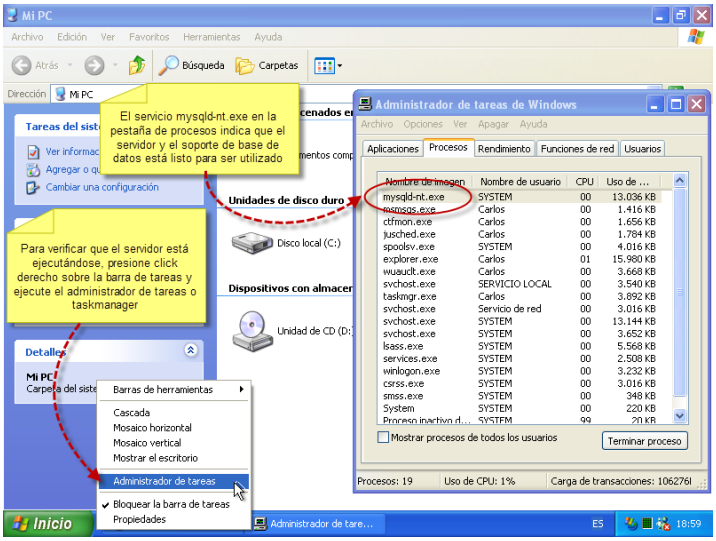
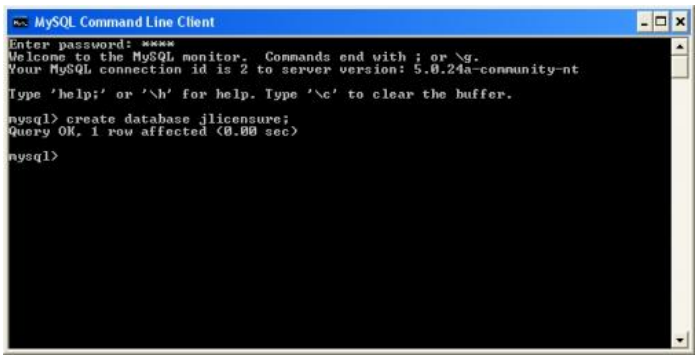


Figura 54. Línea de Comandos en MySQL



8.4. INSTALACIÓN DE GEF (Graphical Editing Framework)

Graphical Editing Framework (GEF) proporciona un conjunto de herramientas y tecnologías para crear recursos gráficos a través de interfaces *WYSIWYG* es el acrónimo de (**W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **G**et). Corresponde a un conjunto de componentes utilizados por el IDE Java Eclipse en el cual se agregan las siguientes características:

- Draw2d (org.eclipse.draw2d). Es una herramienta de diseño de tipo visual para el despliegue de gráficos. Aunque los recursos gráficos proveídos por Draw2d no se conectan con código Java en tiempo real, se pueden desplegar diagramas tipo UML en sus diseños.

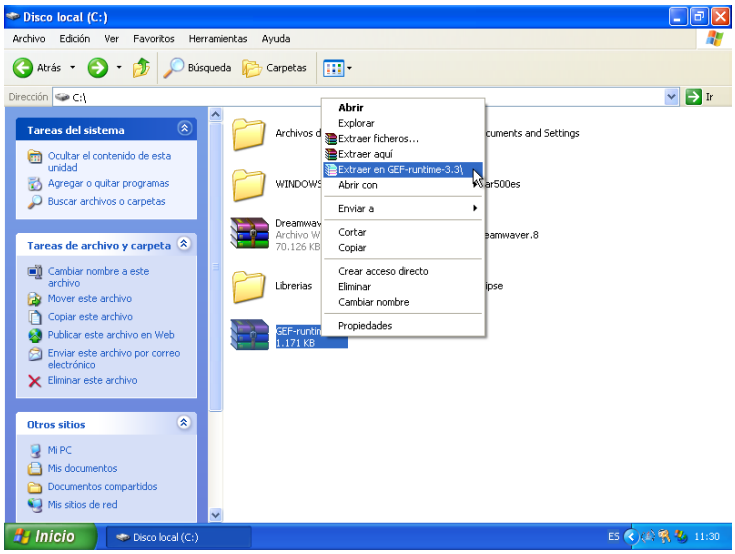
- GEF (MVC) (org.eclipse.gef). Corresponde a un framework basado en el patrón de diseño MVC. El patrón de arquitectura MVC (Modelo – Vista - Controlador) es un patrón que define la organización independiente del **Modelo** (Objetos de Negocio), la **Vista** (interfaz con el usuario u otro sistema) y el **Controlador** (controlador del workflow de la aplicación). Este plug-in de java Eclipse agrega ciertas características para diseño gráfico en ambientes SWT de Java como lo hace el plug-in Draqw2d.

- Zest (org.eclipse.zest). Es un plug-in que permite la integración de modelos Draw2d y SWT en un solo componente de diseño.

Existen diversas formas de instalación para los plug-ins de Eclipse. Los pasos descritos a continuación corresponden con la implementación más sencilla la cual permite fusionar el contenido de los plug-ins en una sola carpeta de componentes.

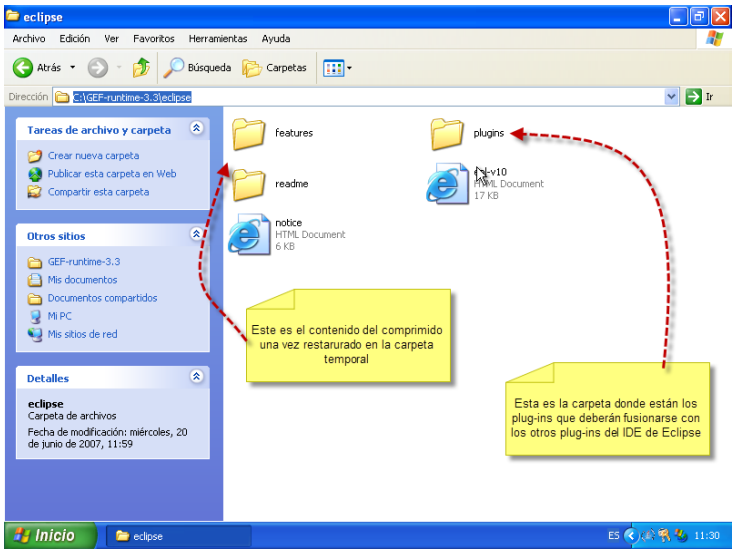
El procedimiento es muy sencillo, descomprima el contenido del archivo GEF-runtime-3.3.zip en una carpeta temporal.

Figura 55. Descompresión del Módulo GEF



Ingrese a la carpeta en la cual se descomprimieron los archivos. Observe que dentro de ésta se ha creado un directorio denominado Eclipse. Ingrese a dicho directorio y despliegue su contenido.

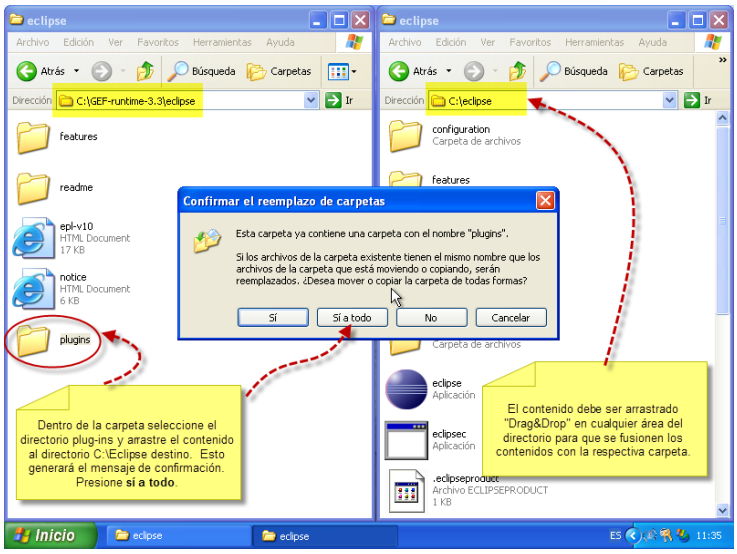
Figura 56. Despliegue de Contenidos de un Directorio



Observe que dentro de éste se encuentra un directorio denominado **plug-ins**. Seleccione la carpeta y realice un proceso de copiado con la técnica Drag&Drop sobre la carpeta de plug-ins del IDE Eclipse Instalado anteriormente.

Este proceso pedirá una confirmación al usuario que consiste en la fusión del contenido de la carpeta plugins de GEF-runtime-3.3.zip descomprimida, con el contenido de la carpeta plug-ins del IDE Eclipse. De esta manera quedarán instalados los plug-ins sin necesidad de recurrir a procesos de actualización desde el IDE de Eclipse. El proceso se detalla a continuación.

Figura 57. Copiando (Fusionando) el contenido del Plug-in GEF en Eclipse



El anterior procedimiento es el más recomendado ya que ArchE reconoce de manera satisfactoria el plug-in GEF instalado a través de éste método.

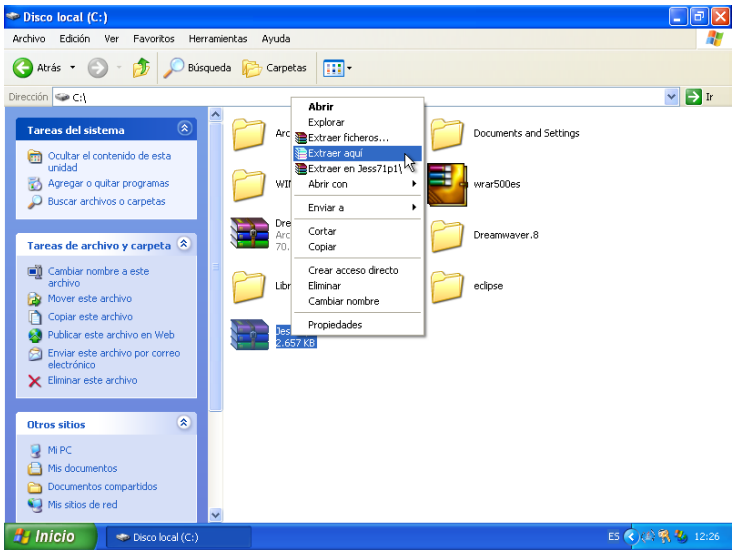
8.5. INSTALACIÓN DEL SISTEMA EXPERTO JESS

El procedimiento de instalación de JESS (**J**ava **E**xpert **S**ystem **S**hell) es similar al anterior, ya que el Sistema Experto ha sido programado como un componente más, el cual puede ser integrado al IDE de Eclipse.

JESS no es sólo un herramienta para sistemas expertos, como motor de reglas propiamente dicho (por ejemplo CLIPS), sino que también provee un conjunto de archivos .java y .class, los cuales permiten el desarrollo de sistemas expertos basados en reglas, que pueden acoplarse de diferentes formas con el lenguaje de programación Java. El procedimiento de instalación es el siguiente:

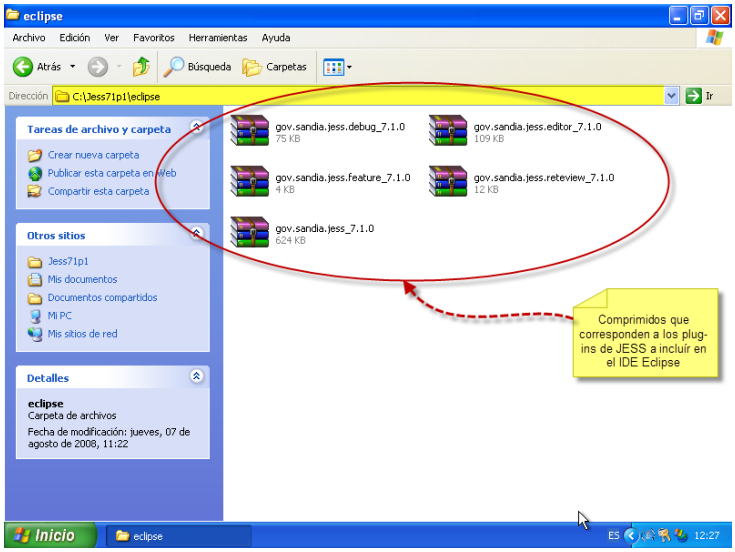
Descomprima el contenido del archivo Jess71p1.zip en una carpeta al mismo nivel de Eclipse.

Figura 58. Extracción e Instalación del Sistema Experto JESS



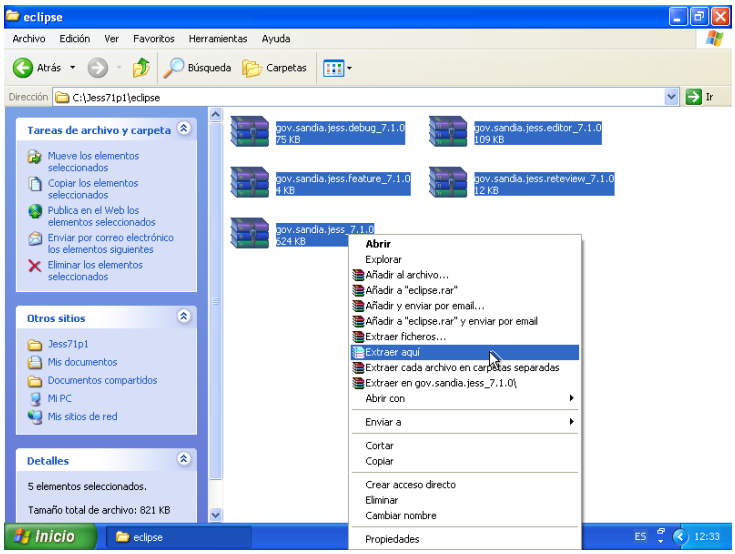
Ingrese al nuevo directorio y ubique la siguiente ruta ***C:\Jess71p1\Eclipse***. Dentro de este encontrará los archivos comprimidos que corresponden a los plug-ins.

Figura 59. Plug-ins del Sistema Experto JESS



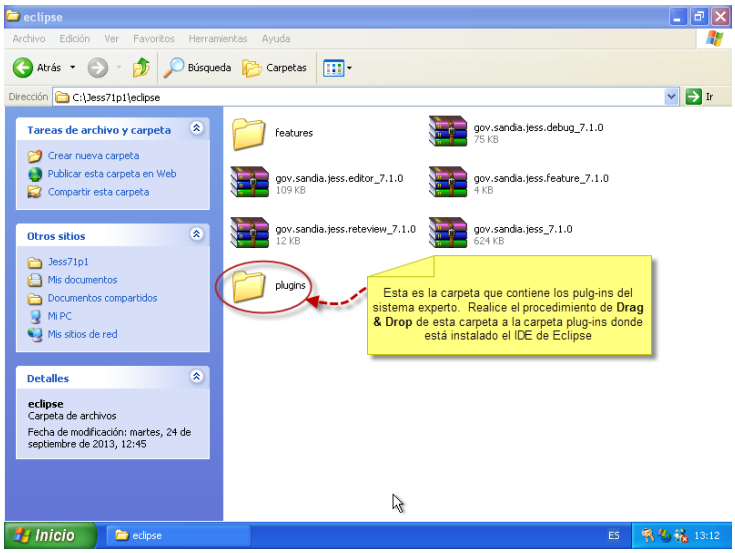
Seleccione todos los archivos y descomprímalos en esta misma carpeta como se muestra en la Fig. 58.

Figura 60. Descomprimiendo el Contenido de los Plug-ins JESS



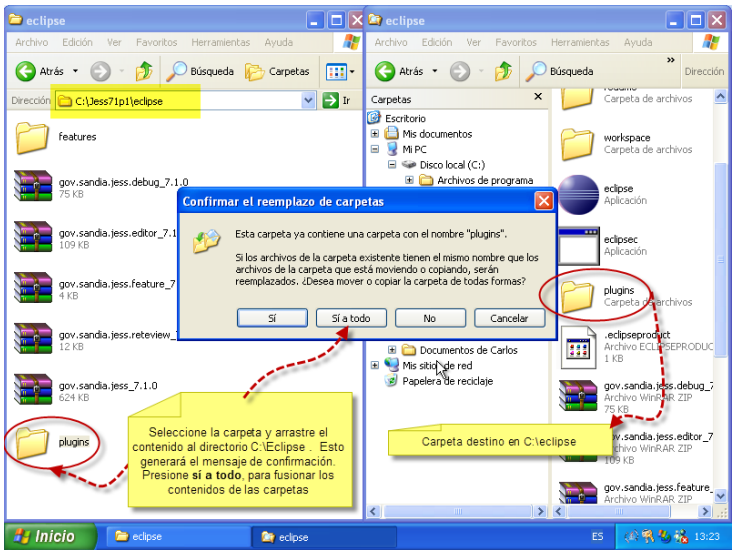
Estando en el directorio ***C:\Jess71p1\Eclipse***, seleccione la carpeta plugins y realice el proceso de fusión mediante la técnica de Drag & Drop entre la carpeta plugins (origen) y la carpeta plugins (destino) donde se encuentra instalado el IDE de Eclipse.

Figura 61. Contenidos y Carpeta de Plug-ins del Sistema Experto



El procedimiento Drag & Drop se muestra a continuación.

Figura 62. Fusión de Contenidos entre Plug-ins de JESS y Eclipse IDE



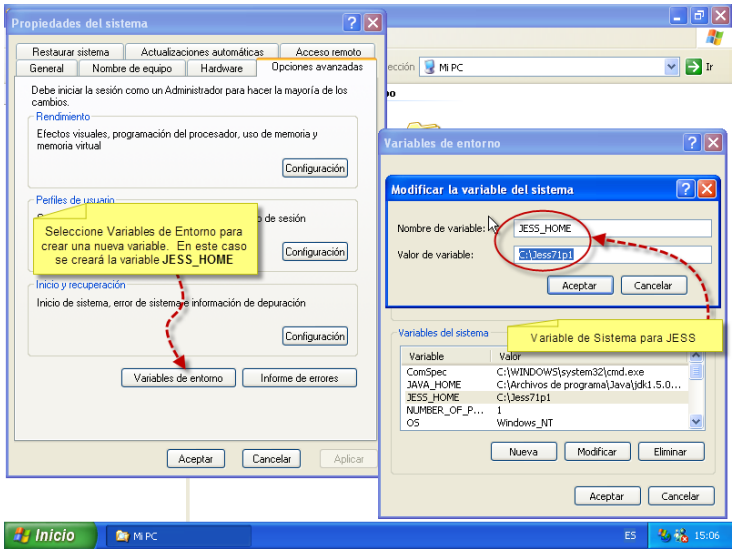
Tras el proceso de fusión de directorios, el sistema operativo pedirá una confirmación al usuario para unir los contenidos de ambas carpetas en una sola. De esta manera quedarán instalados los plug-ins de JESS sin necesidad de recurrir a procesos de actualización desde el IDE de Eclipse. (En el directorio de la instalación hay un directorio eclipse en cuyo interior se tiene el conjunto de clases **gov.sandia.jess_7.1.0.zip**, perteneciente al sistema experto.

El siguiente paso corresponde a la configuración de la variable Path para JESS.

Como se describió anteriormente, la variable **PATH** permite que el sistema operativo ejecute uno o más paquetes a través de una secuencia de comandos (por lo general de tipo Shell o DOS). Con la definición de la variable PATH, se garantiza que aquellos aplicativos compilados en un lenguaje determinado, puedan ser ejecutados en tiempo real. Ya que JESS está basado en JAVA, éste necesita del compilador de clases **Javac.exe** y el intérprete **Java.exe** para ejecutarse.

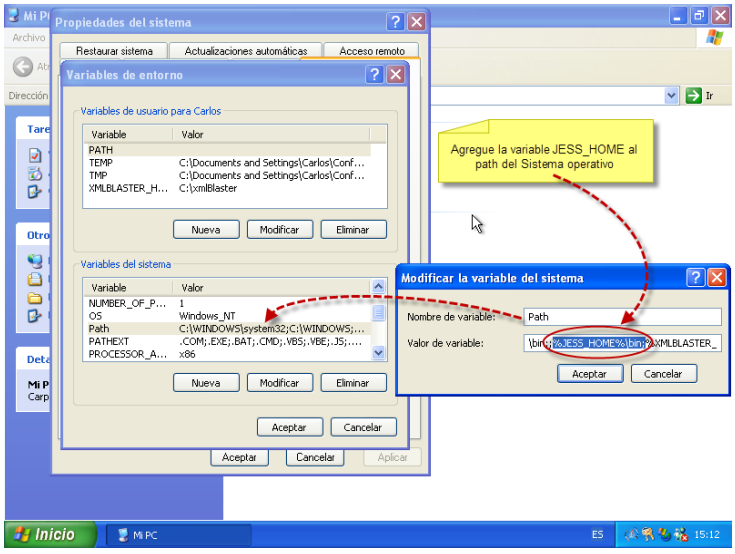
Para acceder a la variable PATH ubique el ícono de "Mi PC" en el escritorio. Selecciónelo y presione click derecho. Vaya a la opción **Propiedades**, luego presione la pestaña de **Opciones Avanzadas** y por último ubique el botón denominado **Variables de Entorno**. En las variables de sistema ubique la variable Path y presione el botón **Modificar**. Se desplegará una ventana denominada "Modificar la variable del sistema".

Figura 63. Variable de Sistema para JESS_HOME



En este cuadro de diálogo va a escribir la siguiente cadena al final del valor de variable que encuentre en la misma. El valor a introducir es: **%JESS_HOME%\bin;** (No olvide el **;** al final). Esto forzará al sistema operativo a compilar las clases directamente desde el directorio de origen.

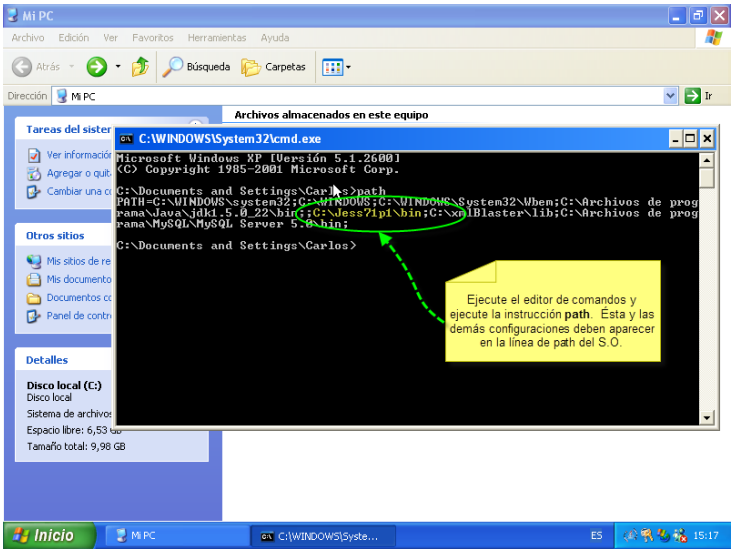
Figura 64. Configuración de la Variable PATH para JESS



Para comprobar que las variables han sido creadas satisfactoriamente, ejecute el editor de comandos del DOS o cmd "command.com" del sistema operativo. Vaya a Inicio

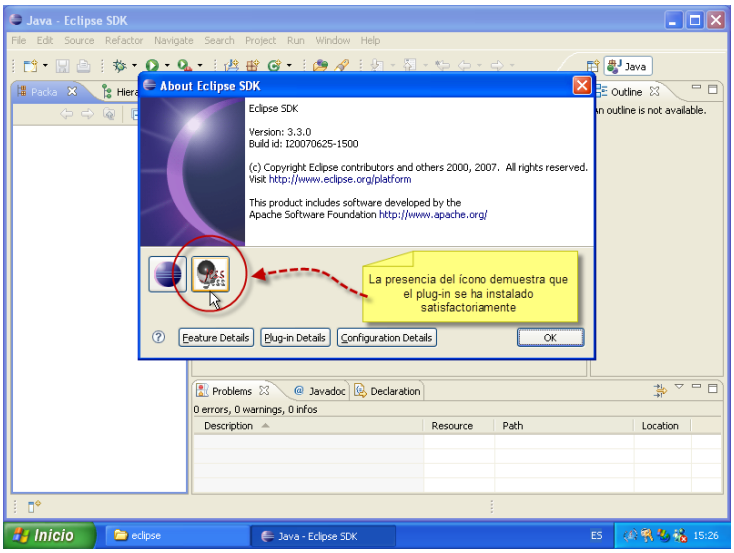
– Ejecutar. En la caja de texto escriba **cmd**. Esto llamará al editor de comandos. A continuación escriba el comando **path**.

Figura 65. Comprobación de la Variable PATH para JESS



Ahora comprobaremos que todos los componentes de JESS están debidamente instalados. Ejecute el entorno de desarrollo Eclipse, vaya a la opción Help > About Eclipse SDK.

Figura 66. Comprobando la Instalación del Plug-in JESS en Eclipse



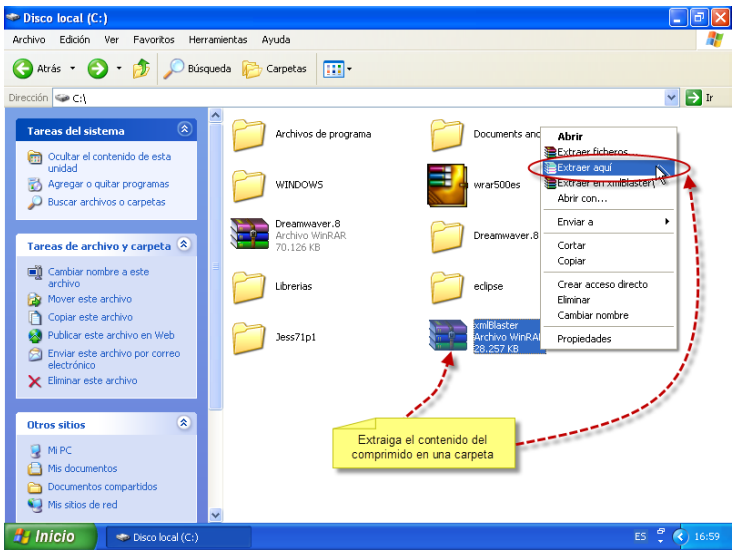
Si observa el ícono de JESS (Fig.66) en el cuadro de diálogo About Eclipse SDK, esto indicará que el plug-in se encuentra instalado correctamente.

8.6. INSTALACIÓN DE XmlBlaster

XmlBlaster es un Message Oriented Middleware. Este software se ejecuta en un servidor al cual se conectan los clientes para leer o publicar mensajes. Los mensajes se describen a través de meta información codificada en XML.

A continuación se documenta el proceso de instalación del parser XML utilizado por ArchE para pasar mensajes entre el servidor y las clases del Sistema Experto. El primer paso en el proceso de instalación de XmlBlaster consiste en descomprimir el contenido del archivo en la raíz del disco duro. Se sugiere que la ruta por defecto para ello sea **C:\xmlBlaster**.

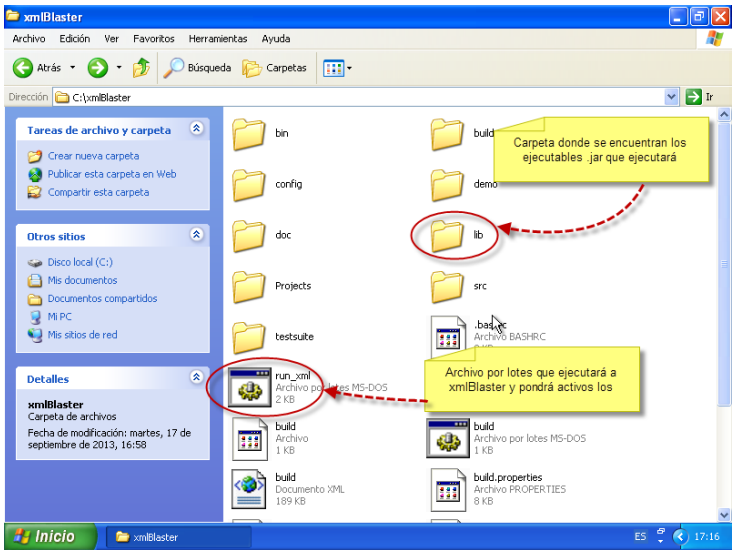
Figura 67. Instalación de XmlBlaster



La Fig. 68 muestra el contenido del directorio C:\xmlBlaster. Dentro de éste se encuentran las librerías de Java necesarias para ejecutar el XmlBlaster. Como las librerías no se compilan por si solas, es necesario que XmlBlaster sea agregado como una variable de entorno en el sistema. Para ejecutar el conjunto de librerías (especialmente **xmlBlaster.jar**), el sistema requiere de un conjunto de instrucciones de carga para que el sistema compile y ejecute las librerías de XmlBlaster en tiempo real.

Para realizar el proceso de carga, en primera medida se debe crear un archivo por lotes, por ejemplo (***Run_xml.bat***) para que dicho archivo ejecute el proceso de carga de XmlBlaster y sus dependencias. (El código del archivo por lote se incluye en el documento). El segundo paso es crear la respectiva variable de sistema.

Figura 68. Contenido del Directorio XmlBlaster



Si no cuenta con el archivo por lotes para lanzar XmlBlaster, a continuación se anexa el batching o lote de procesos. Para crear el archivo abra el editor de texto Notepad, copie y pegue el texto en el editor. Para guardar los cambios escoja la opción Guardar Como > Todos los Archivos y en el nombre de archivo coloque por ejemplo **arrancar_xml.bat**. A continuación salga del editor y verifique que en la carpeta ahora aparece un archivo de tipo MS-DOS parecido al de la Fig. 68. De esta manera quedará listo el archivo por lotes para su ejecución.

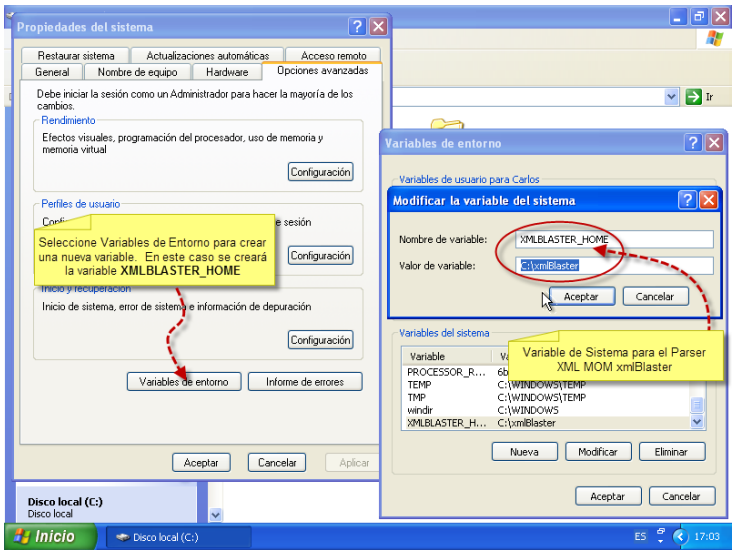
Run_xml.Bat


```
@echo off

if "%OS%" == "Windows_NT" setlocal
rem Guess XMLBLASTER_HOME if not defined

if not "%XMLBLASTER_HOME%" == "" goto gotHome
set XMLBLASTER_HOME=.

if exist "%XMLBLASTER_HOME%\lib\xmlBlaster.jar" goto okHome
set XMLBLASTER_HOME=..

if exist "%XMLBLASTER_HOME%\lib\xmlBlaster.jar" goto okHome
set XMLBLASTER_HOME="%~dp0.."

if exist "%XMLBLASTER_HOME%\lib\xmlBlaster.jar" goto okHome

:gotHome
        if exist "%XMLBLASTER_HOME%\lib\xmlBlaster.jar" goto okHome
        echo "%XMLBLASTER_HOME%"
        echo The XMLBLASTER_HOME environment variable is not defined correctly,
        echo and this script can't find the file "xmlBlaster.jar" without it.
        echo Please set this environment variable to point to your xmlBlaster installation
        echo directory, then try again
        goto end
```

```
:okHome
        rem Make sure JAVA_HOME is set correctly
        if not "%JAVA_HOME%" == "" goto gotJavaHome
        echo Warning: the JAVA_HOME environment variable is not defined
        echo If Jess fails to start, set this environment variable to
        echo point to your JDK installation directory, then try again.
        set RUN_JAVA=java.exe
        goto start

:gotJavaHome
        if not exist "%JAVA_HOME%\bin\java.exe" goto noJavaHome
        goto okJavaHome

:noJavaHome
        echo The JAVA_HOME environment variable is defined incorrectly.
        echo If set, it must represent the path to a J2SDK installation.
        echo If it is unset, then java.exe must be on your path.
        goto end

:okJavaHome
        set RUN_JAVA="%JAVA_HOME%\bin\java.exe"
        goto start

:start
        %RUN_JAVA% -jar "%XMLBLASTER_HOME%\lib\xmlBlaster.jar"
        :end
```

Una vez se tiene el archivo de carga, se procede a crear la variable **PATH** para XmlBlaster. Esto hará que las clases compiladas y ejecutadas bajo Java, sean ejecutadas desde el directorio ***lib*** de XmlBlaster. La definición de esta variable permite que el compilador Javac.exe y el interprete Java.exe compilen y ejecuten todos los paquetes, clases y dependencias de XmlBlaster.

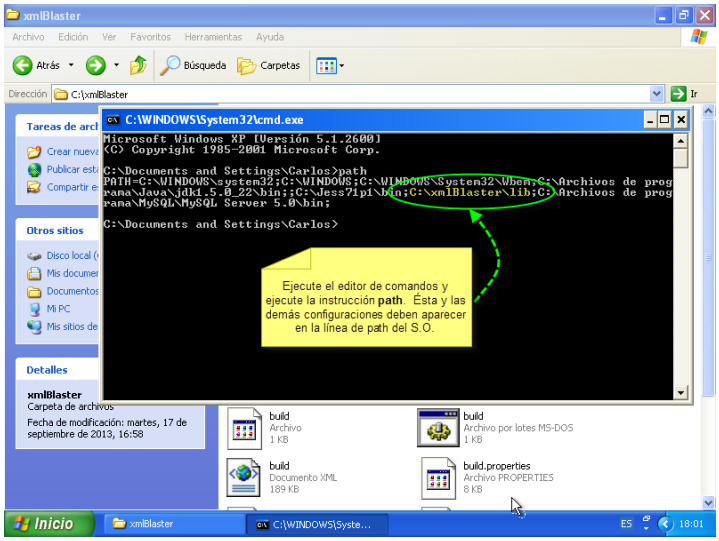
Figura 69. Configuración de la Variable XMLBLASTER_HOME



Para comprobar que las variables han sido creadas satisfactoriamente, ejecute el editor de comandos del DOS o cmd “command.com” del sistema operativo. Vaya a Inicio

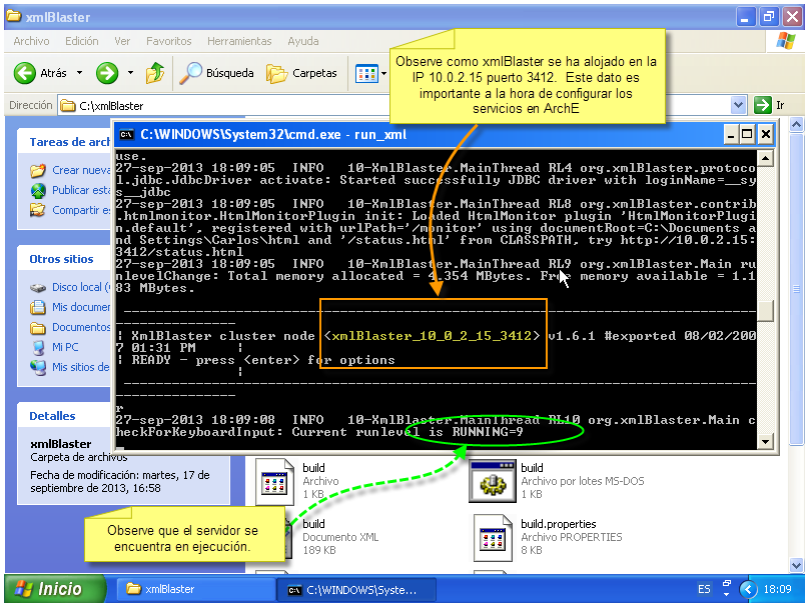
– Ejecutar. En la caja de texto escriba **cmd**. Esto llamará al editor de comandos. A continuación escriba el comando **path** y verifique que se encuentra en el path XmlBlaster.

Figura 70. Comprobando la Variable PATH para XmlBlaster



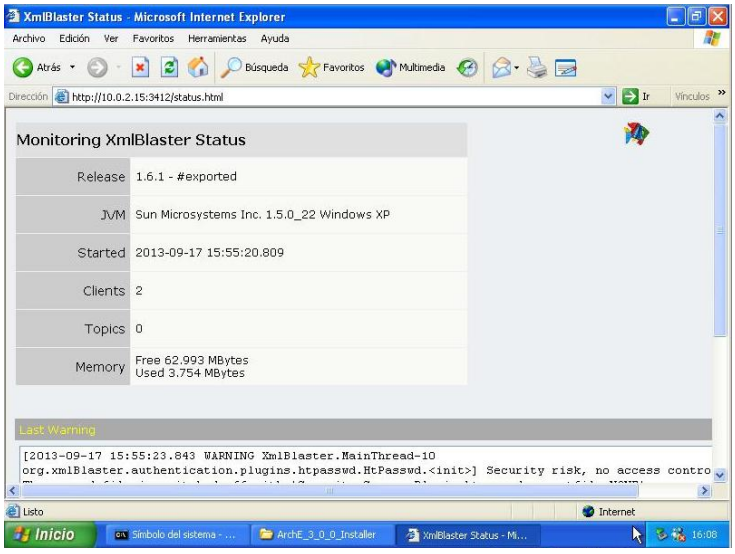
Ahora comprobaremos que XmlBlaster se ejecuta en el ordenador. En el editor de comandos a partir de la línea del prompt del DOS escriba el nombre del archivo .bat. Como ejemplo, se ha utilizado el archivo Run_xml para ejecutar el XmlBlaster. Una vez finalizada la carga presione la tecla ***r*** para ejecutar el MOM XmlBlaster.

Figura 71. Ejecución del Servidor XmlBlaster



Otra forma de comprobar que XmlBlaster está siendo ejecutado en el ordenador es el siguiente; abra el Browser de Internet Explorer u otro navegador y escriba en la casilla de dirección del navegador el siguiente formato de IP. http://ip_config.xmlBlaster:#Puerto/status.html. Para el caso del tutorial, XmlBlaster se instaló en la IP 10.0.2.15, puerto 3412. A continuación se muestra la ejecución del XmlBlaster. (http://10.0.2.15:3412/status.html)

Figura 72. Comprobación y Status de XmlBlaster



Como se mencionó anteriormente, ArchE necesita del servidor XmlBlaster para poderse ejecutar. Por esta razón es importante que cada vez que se desee trabajar con la Arquitectura ArchE, es necesario que XmlBlaster se ejecute antes y durante todo el proceso de carga del sistema ArchE.

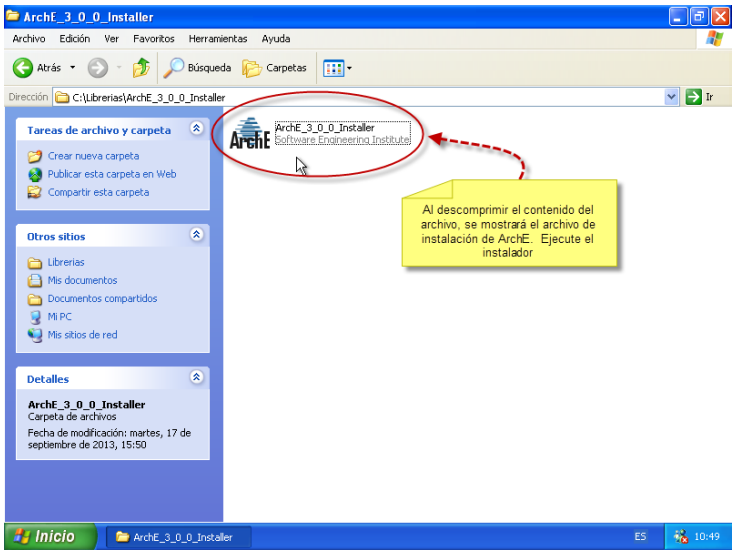
Recuerde que para ejecutar XmlBlaster, basta con iniciar el editor de comandos y llamar al archivo .bat desde la línea de comandos (Fig.71). Para el ejemplo; este archivo fue llamado ***Run_xml.bat***.

8.7. INSTALACIÓN DE ArchE

ArchE es una herramienta para ayudar al arquitecto a explorar distintos diseños de arquitecturas, en base a atributos de calidad; realizando propuestas para mejorar la arquitectura bajo evaluación, de manera que el arquitecto o diseñador pueda elegir la mejor solución. (Pérez-Campanero, 2010). A continuación se detalla el proceso de instalación de la Arquitectura.

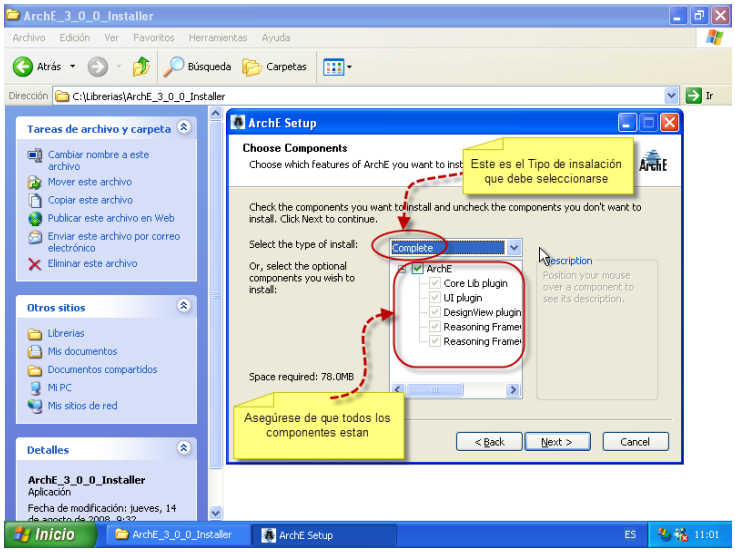
El primer paso para la instalación de ArchE consiste en descomprimir el contenido del archivo en una carpeta temporal.

Figura 73. Instalador de la Arquitectura ArchE



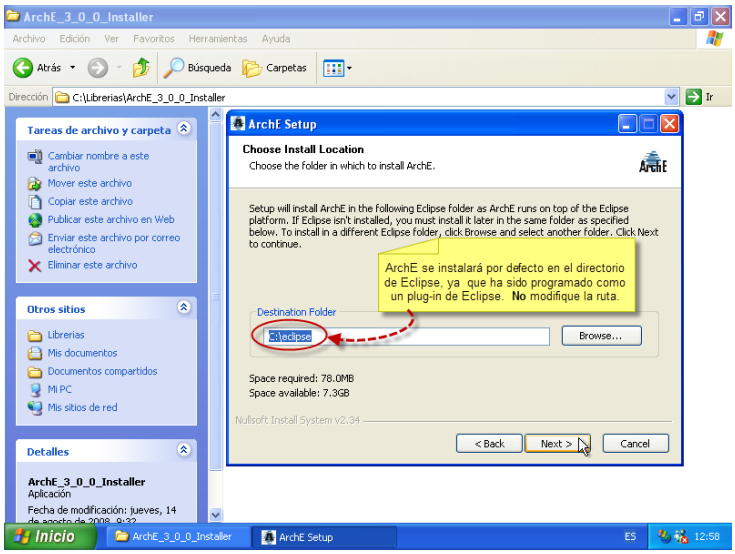
El proceso de instalación será guiado por el asistente. La configuración recomendada para la instalación del paquete del SEI es Completa. Este modo de instalación configurará los paquetes e interfaces de ArchE así como los marcos de razonamiento.

Figura 74. Tipo de Instalación y Configuración para ArchE



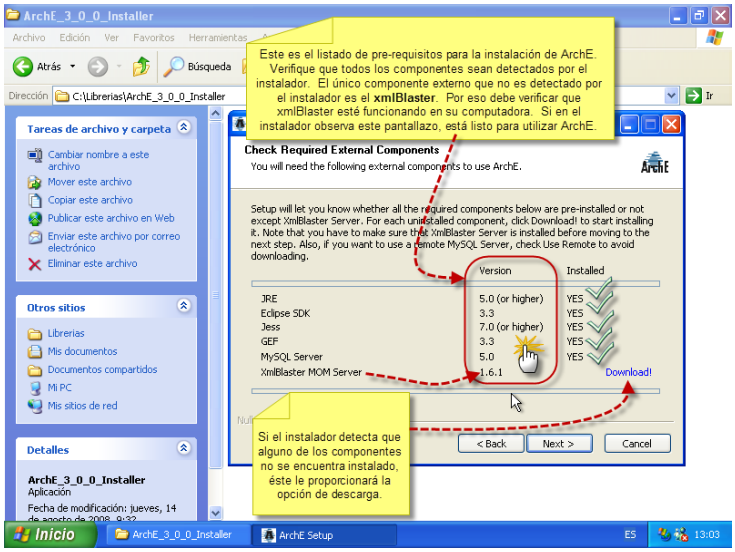
En el siguiente paso se debe configurar el directorio de instalación. ArchE se instalará como un plug-in de Eclipse; por este motivo el directorio de instalación ***no debe ser modificado***.

Figura 75. Directorio de instalación para ArchE



Antes de iniciar el proceso de instalación como tal, ArchE verifica algunas dependencias a componentes o software externo (Fig. 76). Si las herramientas que se han documentado anteriormente han sido instaladas en forma correcta, el instalador detectará las mismas e informará al usuario de su existencia.

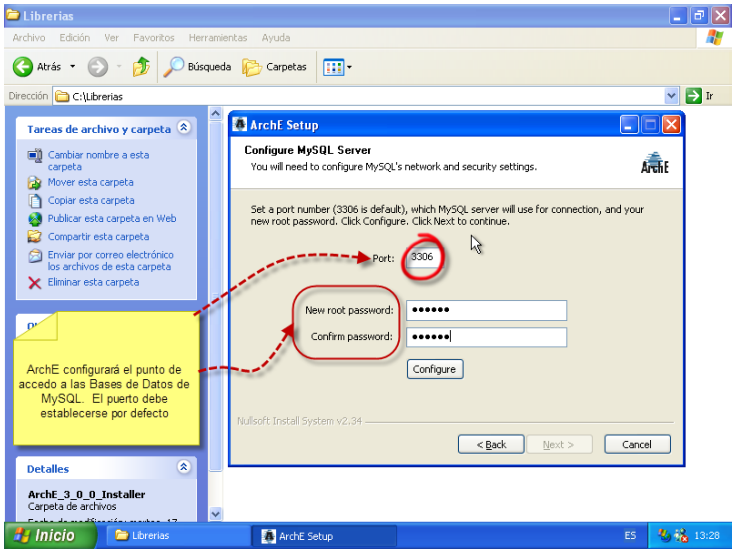
Figura 76. Pre-requisitos para la instalación de ArchE



Si por el contrario, los componentes o plug-ins no se encuentran instalados o sus versiones no corresponden con las predeterminadas por ArchE, el sistema sugerirá su instalación a través de un enlace de descarga <<***Donwload***>>. Esta opción proveerá un

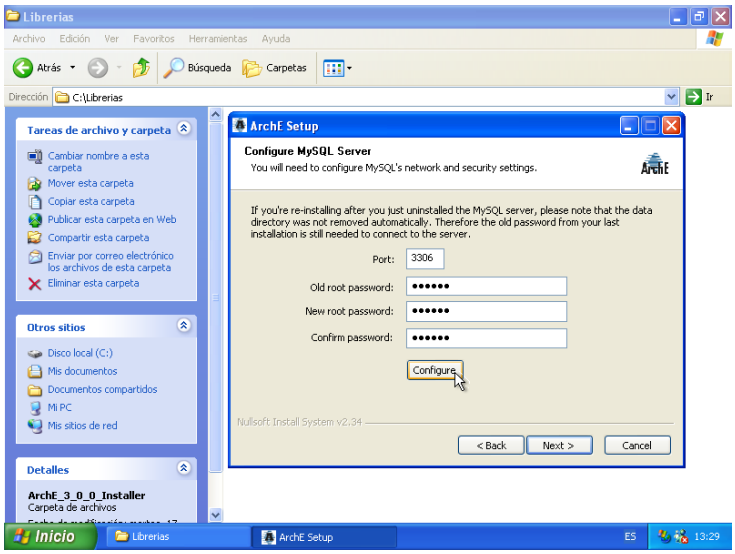
enlace de descarga o llevará directamente al usuario al sitio de descarga del componente. A continuación ArchE configurará los servicios para conexión entre el servidor de MySQL y ArchE.

Figura 77. Configurando Servicios MySQL para ArchE



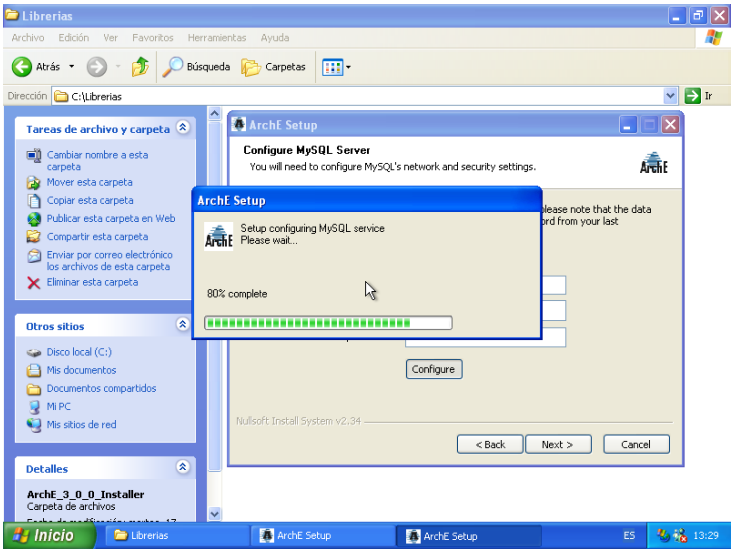
Para continuar con el proceso de instalación presione el botón ***Configure***.

Figura 78. Configuración de Acceso a MySQL desde ArchE



En el siguiente paso, ArchE configura tanto la conexión como la autenticación a las bases de datos MySQL.

Figura 79. Configuración de Servicios MySQL para ArchE



Una vez configurados los servicios, ArchE reportará al usuario el status de dicha configuración. Asegúrese de que los servicios estén configurados y continúe con el proceso de instalación. Al finalizar presione **Next**.

A continuación ArchE creará la base de datos de configuración. ArchE está integrada por 105 tablas y 2 vistas.

Figura 80. Base de Datos de ArchE

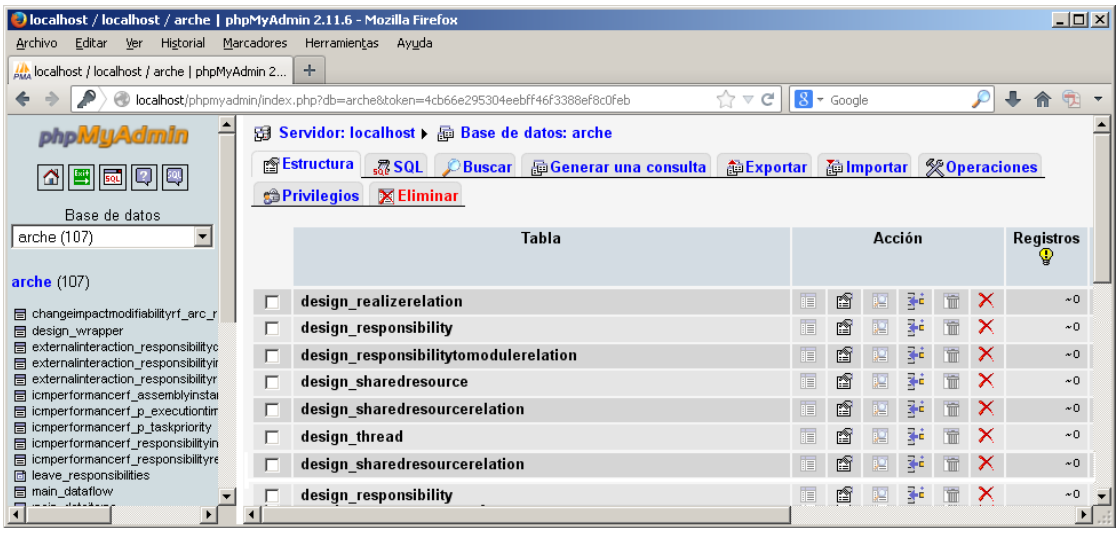


Figura 81. Creación de la Base de Datos para ArchE

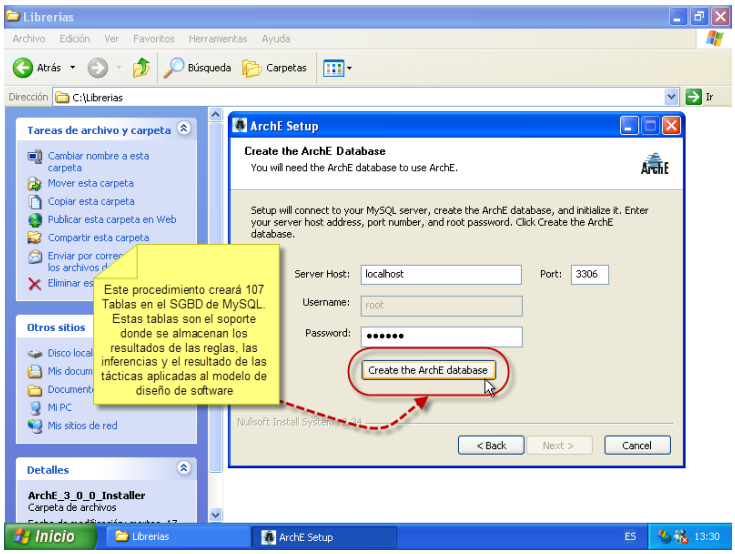
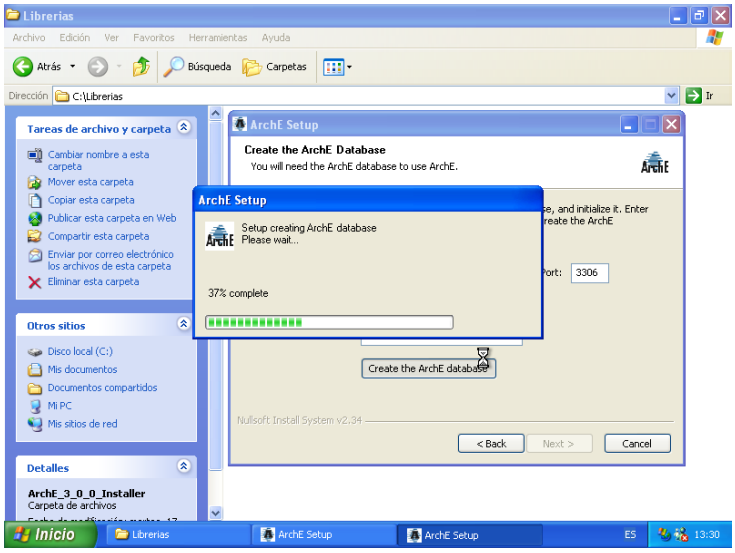
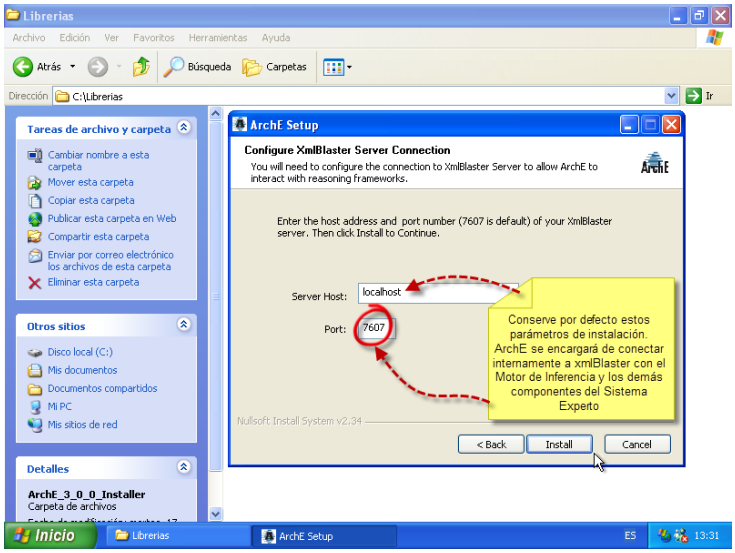


Figura 82. Registrando la Base de Datos de ArchE



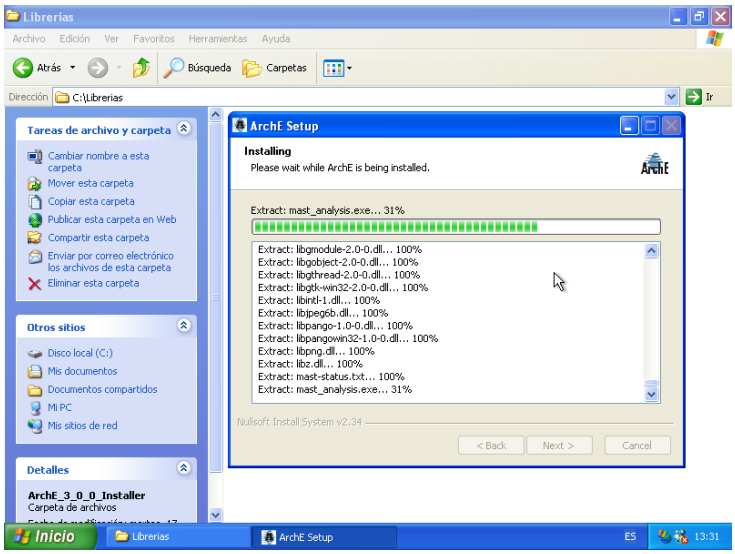
La Figura anterior muestra como el instalador realiza el proceso de creación de la base de datos. Una vez finalizado el proceso, el instalador pedirá al usuario que configure los servicios de conexión con el servidor MOM XmlBlaster. XlmBlaster es el responsable de comunicar al sistema ArchE con el Motor de Razonamiento a través de mensajes y objetos XML. Esta conexión se hace a través de TCP-IP por el puerto 7607. **No** modifique ningún parámetro y continúe con la instalación como lo sugiere el tutorial.

Figura 83. Configurando la Conexión al Servidor XmlBlaster



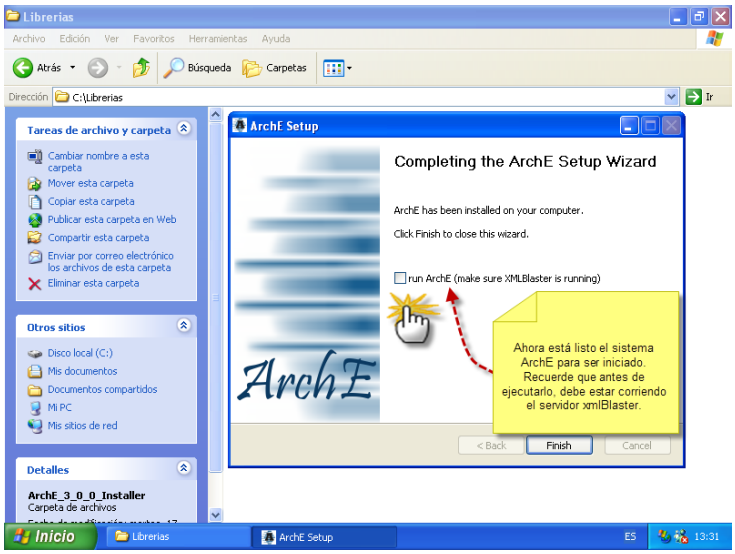
Tras el último paso de configuración de servicios, ArchE instalará todas sus características en el disco duro. Recuerde que ArchE se ubicará en el directorio de Eclipse.

Figura 84. Finalizando el proceso de instalación de ArchE



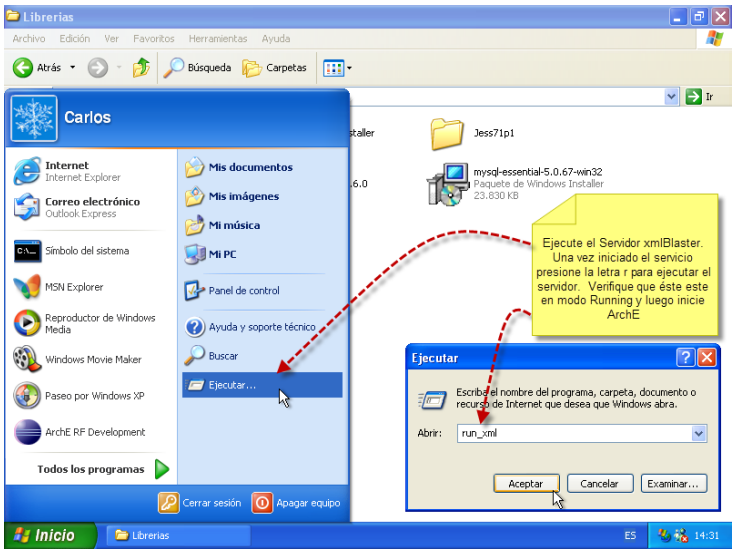
Al finalizar la copia de archivos, el usuario podrá ejecutar el sistema ArchE. Recuerde que antes y durante todo el proceso de carga de ArchE, XmlBlaster debe estar presente. Si este no es ejecutado, el Sistema ArchE no funcionará ya que el motor de inferencia y las bases de datos no serán operables y no se conectará MySQL con la interfaz de ArchE.

Figura 85. Asistente de instalación de ArchE finalizado



Ahora procederemos con la carga de ArchE. El primer paso es ejecutar a XmlBlaster. Presione simultáneamente las teclas + **r** y en la línea de comandos llame al archivo de lotes. Para el caso del tutorial, el archivo será Run_xml. Otra forma de ejecutarlo es a través del botón de inicio de Windows > Ejecutar. Recuerde al final de la carga del servidor XmlBlaster, presionar la tecla **r** para ejecutar el servidor.

Figura 86. Arranque de XmlBlaster



Ejecute ahora ArchE. Inicio > Todos los Programas > ArchE

Figura 87. Ejecutando el Sistema ArchE

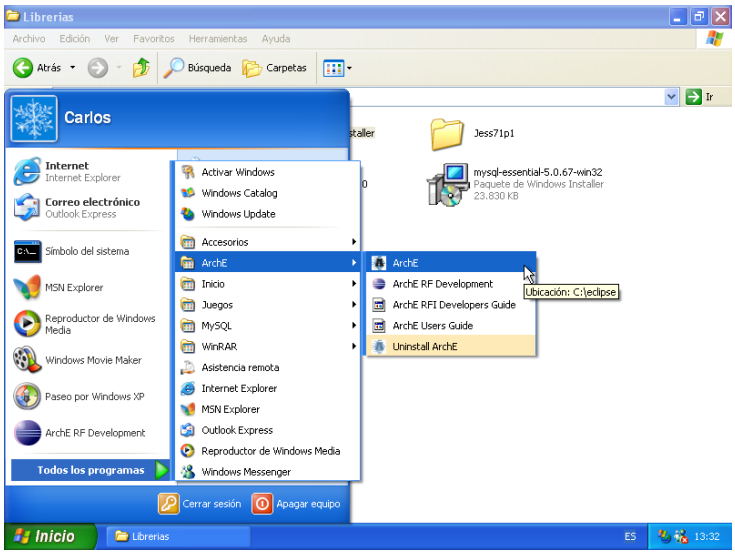
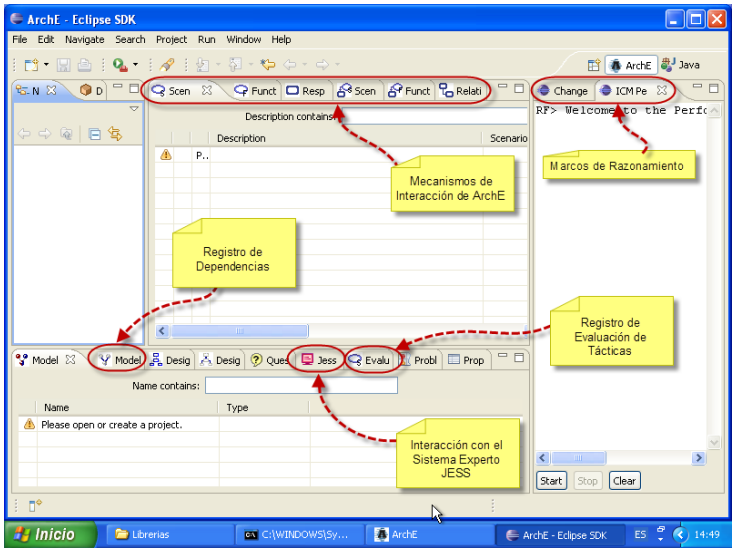
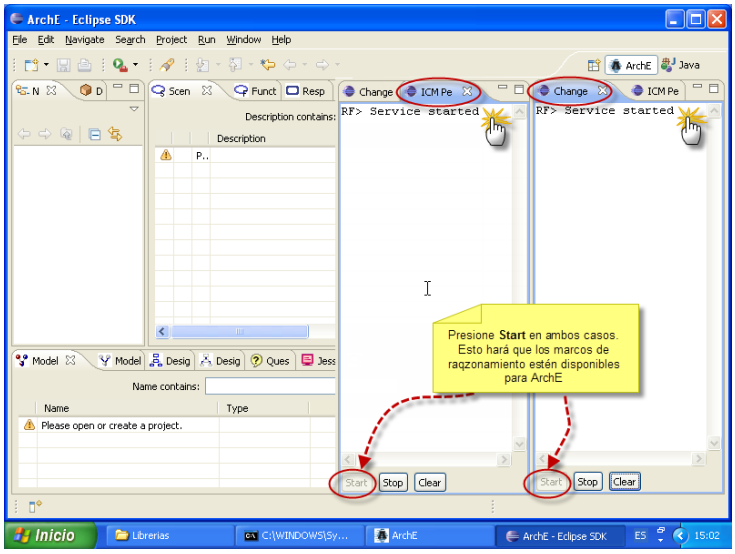


Figura 88. Interfaz de ArchE



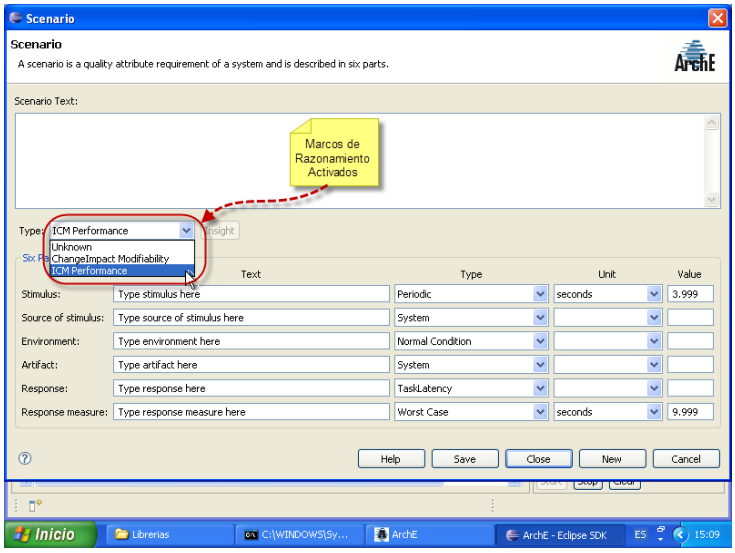
Una vez se ejecuta el entorno de desarrollo, se deben iniciar los dos marcos de razonamiento; el Performance (ICM) Reasoning Framework y el Modifiability (Change Impact) Reasoning Framework. En cada uno de ellos presione **Start**.

Figura 89. Iniciando los Marcos de Razonamiento ICM y CI en ArchE



Luego verifique su funcionamiento creando un nuevo proyecto y cargando un nuevo escenario. Si todo está correcto, ArchE desplegará los contenidos de los Marcos de Razonamiento y se habrá instalado correctamente la Arquitectura ArchE.

Figura 90. ArchE en funcionamiento



De esta manera se ha finalizado el proceso de instalación de ArchE y los demás pre-requisitos.

9

UTILIZACIÓN DE ArchE EN UN CASO PRÁCTICO – CTAS

“Si un proyecto no ha alcanzado el nivel de madurez adecuado a la arquitectura, incluido su análisis, el proyecto no debe proceder a su desarrollo completo. La especificación de la arquitectura en estadios tempranos permite la generación y desarrollo de herramientas con una alta capacidad de mantenimiento y reusabilidad” – El Autor.

Este capítulo se ha destinado al desarrollo del Caso Práctico CTAS, como parte del proceso de investigación y aplicación de los postulados desarrollados por (Bass, Len; Clements, Paul; Kazman, Rick, 2003) en su libro Software Architecture in Practice.

9.1. DESCRIPCIÓN DEL CASO PRÁCTICO CTAS

CTAS "Clemson Traveler Assistant System" es un modelo para la planificación de rutas, modos de transporte y viajes para entre dos o más puntos. El sistema corresponde a una implementación Wireless para dispositivos PDA. El objetivo principal de esta herramienta es proveer al usuario "viajero", de un mecanismo para actualizar la información respecto a un determinado itinerario de viaje.

El sistema cuenta con ciertas características proporcionadas por los interesados "stakeholders". Por una parte los diseñadores y usuarios están centrados en la capacidad y facilidad de uso del aplicativo y por otro lado los diseñadores están buscando los mecanismos necesarios para hacer que la aplicación provea la información al usuario con bajos costos y con una alta disponibilidad. Otra característica en la que está inmerso el desarrollo, es que la herramienta debe permitir al grupo de desarrolladores implementar nuevas tecnologías que permitan acelerar el proceso de evolución de la solución Wireless.

Una de las características que persiguen usuarios externos (propietarios de hotel, aparcaderos, restaurantes, etc.); es que el sistema proporcione en forma precisa una interfaz que permita ubicar al viajero en el escenario o ciudad elegida en su viaje y de esta manera maximizar sus ganancias, disponibilidad y masificación de sus servicios.

Los usuarios interactúan con la aplicación a través de las diversas vistas de la aplicación en las cuales se despliega la información referente al itinerario de viaje.

9.2. PASOS PARA LA DEFINICIÓN DE LA ARQUITECTURA

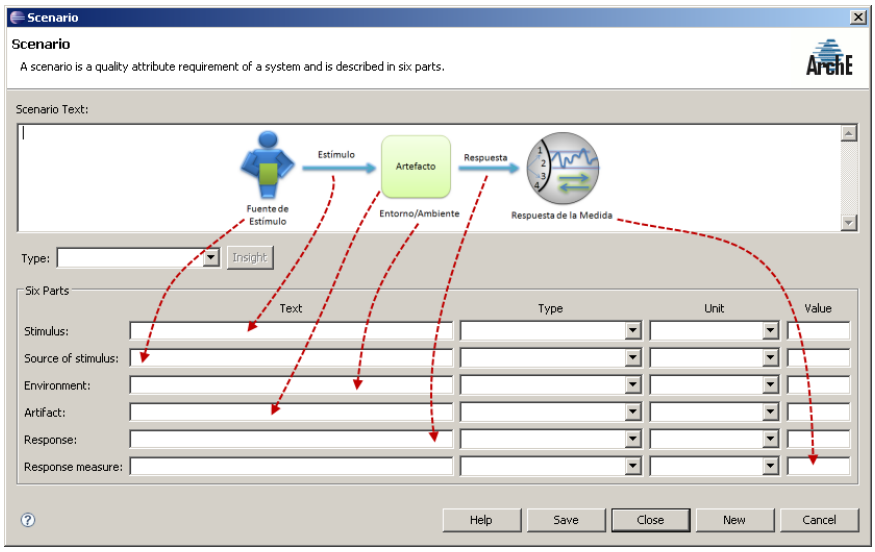
9.2.1. Paso 1 - Obtener Requerimientos

El primer paso consiste en obtener el conjunto de requerimientos para el sistema sujeto de evaluación. Hacen parte de los requerimientos, los atributos de calidad de cada uno de los escenarios así mismo como el conjunto de requerimientos funcionales.

9.2.2. Paso 2 - Refinamiento de Escenarios

Una vez el arquitecto ha modelado los distintos escenarios, es hora de convertir ese conjunto de requerimientos plasmados en papel en un modelo de análisis para ArchE. El refinamiento de los escenarios no es más que la definición de dichos escenarios en función de las 6 partes descritas por (Bass, Len; Clements, Paul; Kazman, Rick, 2003) y que hacen parte de un escenario de calidad.

Figura 91. Refinamiento de Escenarios en ArchE

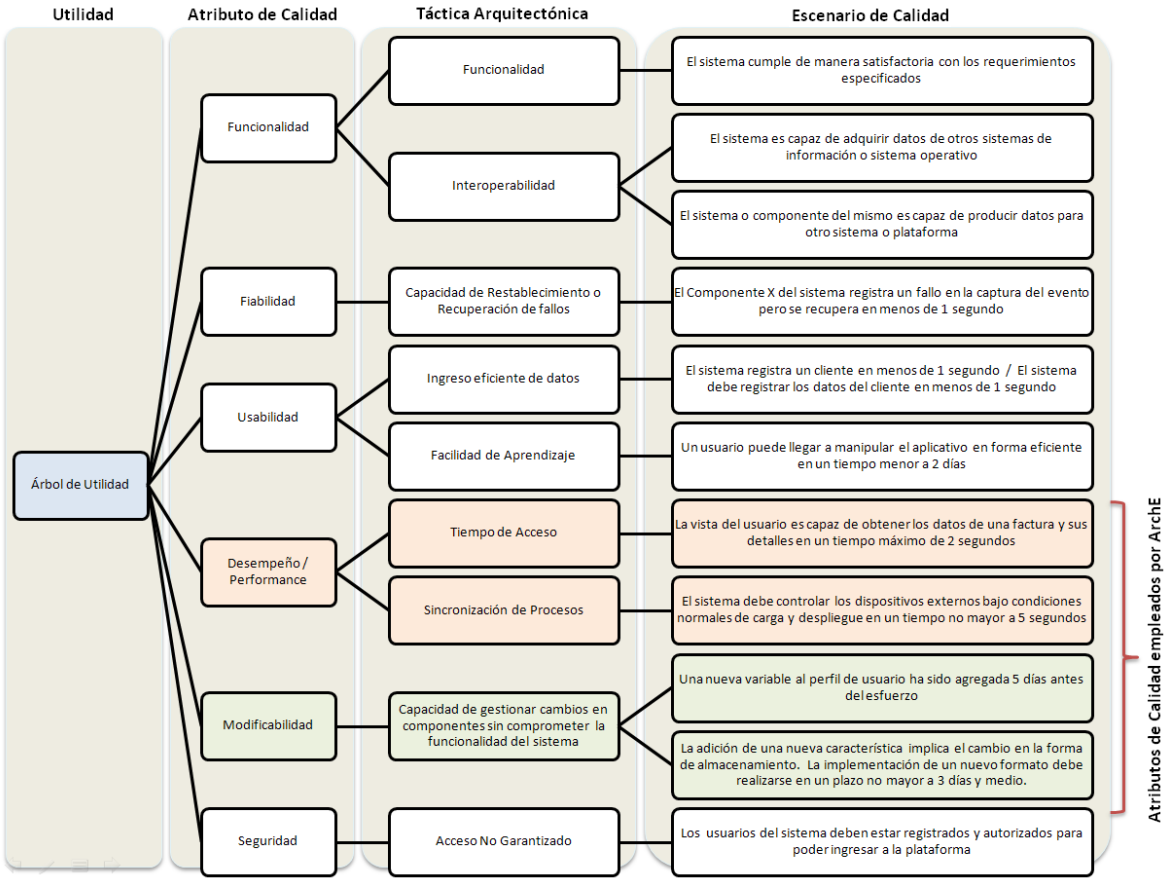


Este formulario, corresponde a la interfaz mediante la cual el arquitecto especifica los atributos de calidad para su modelo. Cada conjunto de requerimientos o la definición de los escenarios se basan en el análisis de los atributos y su mejora así como las posibles tácticas a emplear para el análisis. Del modelo ATAM (Architecture Tradeoff Analysis Method - Utilidad de la Arquitectura) se pueden abstraer algunos conceptos fundamentales que sirven como referente para que el diseñador construya los escenarios en base a los requisitos de mejora. ATAM utiliza como suministro de sus escenarios los ***Árboles de Utilidad***.

Un árbol de utilidad "Utility Tree" es un esquema funcional mediante el cual se representan los atributos de calidad más representativos de un sistema. La raíz representa en sí la utilidad, el primer nivel corresponde a los atributos de calidad requeridos. El siguiente nivel describe en detalle el tipo de táctica a utilizar que posibilitan el incremento en los factores de calidad del atributo y las hojas representan los requerimientos puntuales sobre los cuales actúan los respectivos atributos.

No existe un modelo o estándar establecido que permita utilizar o cuantificar la prioridad de cada uno de los atributos. La intensión fundamental de estos árboles es permitirle al arquitecto, una guía de cómo elaborar sus requerimientos y escenarios de calidad. Estos atributos siempre serán definidos por el arquitecto y difieren de cada modelo o arquitectura. En el siguiente esquema se muestra un ejemplo de ellos. El Análisis en ArchE solo se basa en 2 atributos de calidad; ***Performance*** y ***Modifiability***.

Figura 92. Árbol de Utilidad



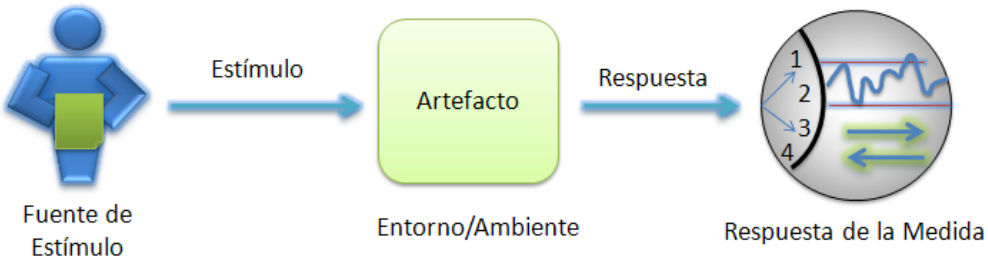
Tras cada proceso de refinamiento "iteración" del modelo, cada escenario se convierte en insumo de entrada para el próximo análisis de sí mismo. De acuerdo al Marco de Razonamiento elegido por el arquitecto, ArchE utilizará el respectivo proceso de análisis. Cada Framework razona en base a procesos de cálculo matemático y probabilísticos, los cuales son comparados contra los datos ingresados en los escenarios para verificar su capacidad "nivel de satisfacción".

- Los escenarios de calidad son el insumo base para el análisis de la arquitectura de ArchE. A través de estos escenarios ArchE introduce el concepto de restricciones de diseño que en conjunto con los demás requerimientos harán parte de la propuesta de mejora en la actividad de diseño del modelo. Los escenarios se encargan de articular las Funcionalidades y las Responsabilidades del sistema. Cada escenario de calidad es almacenado en la base de datos con sus respectivos atributos.

- ArchE interpreta cada requerimiento funcional como una Responsabilidad (la cual es almacenada en la tabla de responsabilidades de la base de datos MySQL), para su posterior trazabilidad con los elementos del modelo.

Para el caso concreto de la arquitectura, cada escenario debe modelarse en base al siguiente concepto:

Figura 93. Escenario de Calidad



Fuente: Adaptada de (Bass, Len; Clements, Paul; Kazman, Rick, 2003) .

9.2.3. Paso 3 - Seleccionar el Framework de Razonamiento

El tercer paso consiste en obtener el conjunto de requerimientos. En este contexto, ArchE determina en base al Marco de Razonamiento elegido por el arquitecto, el tipo de algoritmo para aproximar la solución de la arquitectura. Estos mecanismos incluyen; planificación de prioridades fijas, redundancia de procesamiento y mecanismos para el cálculo de modificabilidad. Como se ha descrito en el documento, ArchE solo utiliza dos (2) marcos de razonamiento; ***Performance*** y ***Modifiability***.

9.2.4. Paso 4 - Construir el Escenario de Calidad

El cuarto paso consiste en completar los datos del escenario en base al Framework de Razonamiento seleccionado. En esencia la contribución que hace este paso al proceso de modelado del escenario es la inclusión de variables de análisis como:

- ***Periodo de Llegada*** (Periódico, estocástico o esporádico);
- ***Tiempo de ejecución*** (en segundos o milisegundos);
- ***Nivel de Prioridad*** (Alta o Baja);
- ***Número de Procesadores*** empleados en el cómputo;
- ***Grado de Acoplamiento*** y
- ***Nivel de Cohesión***.

Este proceso de análisis compete al Framework de Razonamiento, la aplicación de cálculos solo podrán observarse a través de la vista del Framework donde se detalla el

proceso de cálculo de cada escenario. En la página 207 se describe un archivo Log producido por la herramienta de análisis MAST sobre un determinado escenario. En este proceso, ArchE realiza de forma simultánea el cálculo de cada escenario (uno a uno).

9.2.5. Paso 5 - Generar el Modelo

Una vez se han completado los requisitos (Escenarios, Funciones, Responsabilidades y Relaciones) en el modelo, ArchE procede a construir el primer modelo de diseño el cual está asociado a los parámetros de calidad inicialmente proporcionados en la arquitectura. Tras cada proceso o iteración, estos atributos y valores se van modificando conforme a la capacidad o propiedad afectada por el modelo de análisis de Razonamiento. El modelo se completa una vez se han aplicado las tácticas y el modelo satisface los requerimientos de diseño.

9.3. CASO PRÁCTICO CTAS (Clemson Traveler Assistant System)

9.3.1. Paso 1 – Requerimientos en el Sistema CTAS

Uno de los objetivos de este trabajo de Investigación consiste en la aplicación del caso práctico CTAS en la Arquitectura ArchE. De acuerdo con el texto del (SEI, 2007), CTAS corresponde a la implementación de un sistema para la planificación de itinerarios a través de diversas rutas y modos de transporte entre un nodo fuente y un nodo destino.

Una de las particularidades de este sistema es que su arquitectura corresponde a una aplicación con capacidad de ejecución en múltiples plataformas incluyendo a la ubicuidad como un mecanismo de difusión que permite a los usuarios actualizar sus itinerarios en tiempo real. El sistema CTAS cumple con las siguientes características:

- CTAS es un sistema multiplataforma.

- Es un sistema de fácil acceso, flexible y con capacidad de actualización de registros e itinerarios en tiempo real.

- Es un sistema que se ha desarrollado con base en la perspectiva de negocio y del desarrollo mismo involucrando desde programadores hasta usuarios finales. Desde esta perspectiva el diseño del sistema busca que la planificación de itinerarios sea lo más eficiente posible, que la actualización de la información se haga en un tiempo record optimizando servicios, que la información pueda ser accesada por sistemas de cómputo ubicuo y que sea lo suficientemente flexible para que sea usado por cualquier persona sin importar el conocimiento de la plataforma, y que el conjunto de componentes desarrollados sean lo suficientemente reutilizables como para generar nuevas expectativas de negocio y valor agregado.

9.3.1.1. Actores o Stakeholders en CTAS

Para el caso de estudio, los **stakeholders** se definen como todos los **actores sociales** involucrados en el desarrollo del producto CTAS. Este término agrupa a programadores, usuarios finales, usuarios abstractos, grupos de interés primario y secundario, gobierno, accionistas y entre muchos otros actores clave; que intervienen en el proceso y que de una u otra forma se verán afectados de forma positiva o negativa a partir de la implementación del producto software.

Tabla 35. Stakeholders para el Caso Práctico CTAS

ACTOR	DESCRIPCIÓN
CTAS user	Corresponde a aquellos usuarios que poseen un número limitado de itinerarios y rutas a partir de un destino predeterminado. Los usuarios tienen la potestad de revisar y planificar sus itinerarios a través del sistema de información o a través de consultas en bases de datos Ad-Hoc[16]. Para el usuario, es fundamental que el sistema sea flexible y altamente utilizable.
CTAS information provider	Este tipo de actor se responsabiliza de mantener y proveer los datos al sistema CTAS para posteriormente computar los itinerarios. Un ejemplo de este tipo de actor puede ser un vehículo equipado son sistemas de medición de tiempo/distancia, un sistema de parquímetro con control de GPS para localización o un sistema sensor que permita verificar la disponibilidad de la información.
CTAS smart unit	Corresponde a un tipo de actor especializado que provee información específica al usuario de CTAS. Esta información puede derivarse de mapas, mecanismos automatizados u otros sistemas computados.
CTAS device	Agrupa a todos aquellos dispositivos mediante los cuales se puede desplegar el aplicativo de software. Cualquier dispositivo electrónico diseñado específicamente para la consulta y actualización de itinerarios, así como otros dispositivos electrónicos como PDA's, Smartphones, Tablets, etc; hacen parte de este tipo de actores.
CTAS-related Hardware	Este tipo de actor está representado por cualquier dispositivo de hardware (y su respectivo middleware) que de una u otra forma está o se puede relacionar con los dispositivos de CTAS.
CTAS peripheral	Es un actor secundario que agrega una capacidad específica al dispositivo de CTAS. Por ejemplo un GPS. Estos periféricos tienen la capacidad de entablar comunicación con los dispositivos de CTAS en ausencia o presencia de información. Si se encuentran por ejemplo conectados a sistemas de navegación, el sistema debe proveer la ubicación; si por el contrario el escenario se encuentra desconectado el sistema tendrá la capacidad de preguntar al usuario por su ubicación.
CTAS expansion Device	Corresponde a un mecanismo capaz de proporcionar mayor cobertura de los servicios para CTAS.

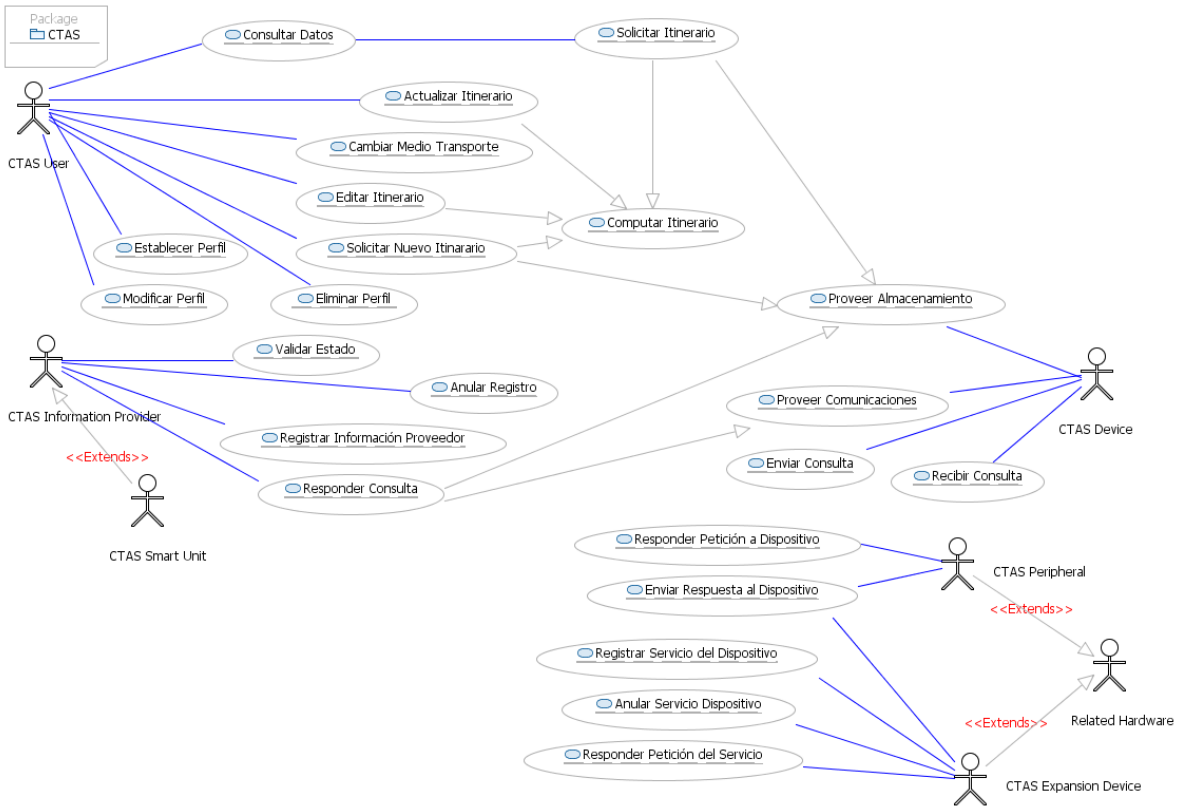
Fuente: Using ArchE in the Classroom. (Bachmann, Bass, Bianco, & Klein, 2007).

[16] El término ***Ad-hoc*** se refiere al uso de consultas personalizadas en tiempo real por parte de los usuarios, lo que implica el uso de interfaces potentes para consulta sin atar diseños a consultas prediseñadas.

9.3.1.2. Caso de Uso para el Sistema CTAS

Un **Caso de Uso** corresponde a una descripción detallada de actividades y a una secuencia de pasos que un sistema debe realizar para lograr un objetivo. Los casos de uso interactúan con los usuarios a través de mensajes y responden a ellos a través de una serie de eventos. Estos usuarios son denominados a menudo como Actores.

Figura 94. Caso de Uso CTAS



En el caso práctico, el diagrama de casos de uso que se presenta en la Fig. 94 ilustra de una manera global los requerimientos del sistema y cómo éste responde al usuario a través de mensajes y eventos. A continuación se muestra el diagrama general de casos de uso para el sistema CTAS. A continuación se lista el conjunto de Casos de Uso para CTAS.

Tabla 36. Listado General de Casos de Uso para CTAS

Actor	Casos de Uso en los que interviene
 **CTAS user**	• Consultar datos • Establecer Perfiles • Modificar Perfiles • Eliminar Perfiles • Solicitar Nuevo Itinerario • Editar Itinerario • Cambiar Medio de Transporte • Actualizar Itinerario
 **CTAS information provider**	• Validar Estado • Anular Registros • Registrar Información de Proveedores • Responder Consultas
 **CTAS device**	• Proveer Almacenamiento • Proveer Comunicaciones • Enviar Consulta • Recibir Consulta
 **CTAS peripheral**	• Responder Petición a Dispositivos • Enviar Respuesta al Dispositivo
 **CTAS expansión Device**	• Registrar Servicio del Dispositivo • Anular Servicio del Dispositivo • Responder a Petición del Servicio

9.3.1.3. Atributos de Calidad para el Sistema CTAS

Los atributos de calidad para el sistema CTAS, está fundamentado en la norma ISO/IEC:9126 "*Estándar Internacional para la evaluación de la calidad del software*".

En el caso propuesto, la Arquitectura ArchE solo trabaja con 2 atributos de calidad. Estos son el atributo de **Rendimiento** o performance y el atributo de **Modificabilidad**.

Los atributos de ***Rendimiento*** o ***Performance*** a los cuales hace referencia el ejemplo CTAS establecen que el sistema debe poseer en su estructura un mecanismo capaz de interactuar con otros dispositivos en tiempos relativamente cortos y sin necesidad de comunicación por hilos de procesos, y el segundo atributo corresponde a la velocidad y tiempo de proceso y optimización de la ruta más optima frente al itinerario planteado.

En lo que respecta al factor de ***Modificabilidad***, CTAS debe ser capaz de permitir una mayor funcionalidad a medida que el dominio del problema evoluciona. Es decir; CTAS debe tener la capacidad de interactuar con otras Arquitecturas como ArchE para modificar la capacidad del producto y las posibles variantes para mejorar los diseños arquitectónicos.

Para el caso práctico CTAS, los atributos de calidad aplicados al sistema se resumen en la siguiente tabla:

Tabla 37. Atributos de Calidad para el Sistema CTAS

Característica:	***FUNCIONALIDAD***
Descripción:	Esquema que provee funcionalidades que cumplen con necesidades específicas o implícitas del negocio. Evalúa la forma en cómo interactúa el software cuando es utilizado bajo ciertas condiciones.
Sub-Características	
Exactitud	El itinerario proporcionado por el sistema CTAS debe precisar de exactitud tanto a nivel de procesos como de respuestas. El sistema debe controlar el flujo de errores mediante el uso de mensajes si se excede la capacidad del sistema.
Interoperabilidad	El sistema CTAS debe estar en capacidad de procesar información procedente de otros tipos de proveedores de información. Cualquier estándar o mecanismo de interoperabilidad debe ser identificado y se debe realizar sobre éste el respectivo seguimiento.
Seguridad	Debido a que la comunicación entre el sistema CTAS y los proveedores de información a menudo está disponible a cualquier usuario, el sistema CTA debe ofrecer las políticas de seguridad para mantener la confidencialidad y los datos.

Característica:	***FIABILIDAD***
Descripción:	Esta característica permite identificar potenciales fallos en el diseño para evitar a futuro fallos en el producto. La Tolerancia a Fallos y pruebas de factoría son indispensables en el contexto del diseño. Esta capacidad se relaciona directamente con el rendimiento del producto software y la capacidad que tiene éste de continuar su ejecución a pesar de presentarse errores inesperados.
Sub-Características	
Recuperabilidad	Cualquier itinerario debe estar disponible para su uso incluso en situaciones en las que el sistema se reinicializa de forma espontánea.
Tolerancia a Fallos	No se aplica para el Caso CTAS

Característica:	***USABILIDAD***
Descripción:	Capacidad del producto de software para ser atractivo, entendido, aprendido y utilizado por el usuario bajo condiciones específicas.
Sub-Características	
Aprendizaje	Esta característica hace que cualquier usuario sea capaz de utilizar y aprender a manipular la herramienta a partir de una serie de instrucciones básicas.
Comprensión	El sistema debe ser fácil de usar y no debe requerir de procesos extensivos de entrenamiento o manuales complejos para la operación del sistema.
Operatividad	En la mayoría de los casos se emplearán interfaces de teclado. El usuario debe tener acceso a dispositivos de texto QWERTY u otro tipo de texto para estilos en su PDA, Smartphone o cualquier otro dispositivo desde donde se pueda desplegar la información de CTAS.

Característica:	***EFICIENCIA***

Descripción:	Esta característica interviene directamente en el producto software proporcionando interfaces apropiadas que mejoran el rendimiento tanto en capa de datos, accesos y utilización de recursos tanto de red como de sistemas operativos.
Sub-Características	
Temporalidad	El sistema CTAS debe estar en capacidad de procesar una salida "itinerario" en un tiempo no mayor a 30 segundos.
Utilización de Recursos	El sistema CTAS debe funcionar con arquitecturas donde la memoria dinámica se encuentre en el orden de los 256 MB

Característica:	***MANTENIBILIDAD***
Descripción:	Esta característica se asocia con la capacidad que tiene el software para evolucionar en el tiempo, dependiendo de los requerimientos del cliente, del negocio o las mejoras introducidas a lo largo del proceso de desarrollo o sus iteraciones. Su potencial radica en la capacidad del producto para ser modificado.
Sub-Características	
Facilidad de Análisis	Una modificación al sistema CTAS, debe realizarse con un tiempo de esfuerzo no mayor a 4 horas por el equipo de desarrollo.
Facilidad de Cambios	Los cambios estructurales en el sistema CTAS se deben estimar en un tiempo no mayor a 3 días.
Facilidad de Pruebas	El sistema CTAS debe poder ser probado con un nivel de esfuerzo en razón a un tercio del esfuerzo total del desarrollo.

Característica:	***PORTABILIDAD***
Descripción:	Se define como la característica que posee un componente software para ejecutarse en diferentes plataformas. Es una capacidad del producto de software para ser transferido de un ambiente a otro. Esta característica es importante para el desarrollo en función de reducción de costos.
Sub-Características	
Instalación	El sistema CTAS debe poseer su propio mecanismo de instalación, que lo haga portable y sencillo de implementar.
Sustitución	El sistema CTAS debe tener la capacidad de ser mejorado o versionado. Debe preservar el mecanismo de instalación y portabilidad.
Adaptabilidad	El sistema CTAS debe poder ser portado a un nuevo dispositivo reemplazando los drivers de conexión externos.

9.3.1.4. Especificación de Requerimientos y Relaciones

Antes de iniciar con el proceso de diseño, ArchE debe conocer en detalle los requerimientos de la arquitectura. Estos requerimientos describen la relación entre un conjunto de responsabilidades y su evaluación a partir de los escenarios de calidad. ArchE utiliza una descripción en colores "**Gris**, **Verde** y **Rojo**" en el caso de los escenarios. Cuando el usuario ingresa un escenario y este se torna de color **Verde** este estatus indica o representa las salidas o acciones por defecto elegidas por ArchE. El status **Verde** indica que el escenario ha sido analizado por ArchE y sugiere que dicho escenario no requiere de ninguna modificación a sus atributos ya que satisface las métricas de calidad. Cuando ArchE despliega en la columna un icono de color **Gris**, indica que dicho escenario no ha sido analizado por ArchE. Si el icono visualizado se torna de color

Rojo, indicará que este escenario no se cumple y por tanto se deben aplicar tácticas para corregir sus atributos.

9.3.1.5. Responsabilidades y Dependencias en CTAS

Tabla 38. Matriz DSM para el Conjunto de Responsabilidades en CTAS

Uso de Responsabilidades / Responsabilidades	1. Agregar al modelo	2. Crear Perfil de Usuario	3. Manejar Interacciones Usuarios	4. Localizar Servicios	5. Manejar Dispositivos Externos	6. Manejar Itinerarios	7. Modificar Perfiles de Usuario	8. Consultar Datos	9. Registrar Vistas	10. Almacenar Datos	11. Mostrar Itinerarios
1. Agregar al modelo									1		
2. Crear Perfil de Usuario			1								
3. Manejar Interacciones Usuarios											
4. Localizar Servicios								1			
5. Manejar Dispositivos Externos											
6. Manejar Itinerarios			1		1		1				
7. Modificar Perfiles de Usuario		1	1								
8. Consultar Datos						1					
9. Registrar Vistas			1								
10. Almacenar Datos		1				1	1				
11. Mostrar Itinerarios			1			1					

Nota: La matriz anterior ilustra el conjunto de responsabilidades del sistema CTAS con sus respectivas dependencias.

A continuación se realizará la implementación del sistema CTAS en la arquitectura ArchE. Como se ha descrito en el texto del (SEI, 2007), CTAS corresponde a la implementación de un sistema para la planificación de itinerarios a través dispositivos

PDA para viajeros entre un nodo fuente y otro destino. El sistema describe un conjunto de rutas y modos de transporte entre la fuente y el destino.

9.3.2. Paso 2 - Refinamiento de Escenarios en CTAS

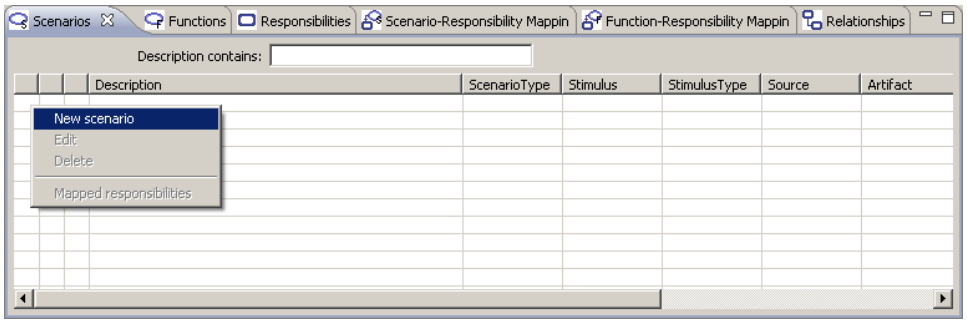
9.3.2.1. Definición de Escenarios para el sistema CTAS

El primer paso corresponde a la definición de los escenarios en Arche. Como se describió en apartados anteriores, un escenario corresponde a un caso de éxito en un determinado proceso o Caso de Uso. Al crear cada escenario, ArchE brinda al usuario la posibilidad de crear el escenario de acuerdo a las especificaciones del modelo. A través de este escenario se asigna el respectivo Framework de Razonamiento (Fig.98) con el cual se cruzará el escenario, dependiendo de ello se desplegaran las opciones de los marcos disponibles para ArchE. Por el momento ArchE solo implementa el Framework ICM Performance y el Framework para el atributo de Modificabilidad. Cada escenario puede ser editado, modificado o eliminado conforme a las necesidades del arquitecto.

9.3.2.1.1. Agregar un nuevo Escenario

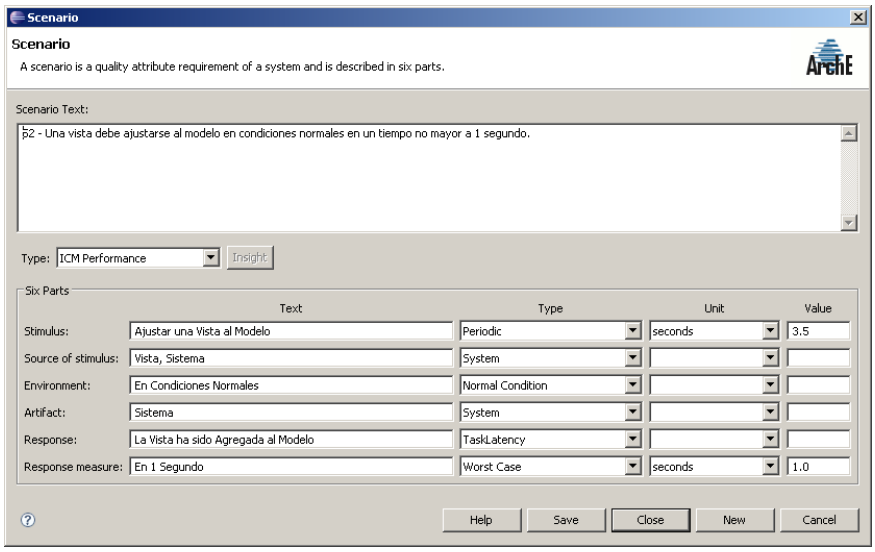
Para agregar un nuevo escenario diríjase a la vista de escenarios, presione click derecho. Se desplegará el contenido de un menú emergente. Presione como le indica la siguiente ilustración.

Figura 95. Agregar un Nuevo Escenario



Al crear un nuevo escenario, inmediatamente se desplegará la vista que da acceso a los escenarios de calidad. Cada escenario es almacenado con sus atributos, los cuales podrá modificar y consultar a lo largo del proceso. Todos los escenarios se modelan de forma independiente, por ese motivo es importante que cada escenario contenga específicamente los valores, medidas y atributos de calidad necesarios para la evaluación de ArchE. A continuación se muestra uno de los escenarios Refinados en la vista de Escenarios.

Figura 96. Refinamiento de Escenarios



Nota: Es aconsejable que tras cada proceso de adición o modificación de los atributos en los escenarios u otras funcionalidades del sistema, el usuario realice el grabado o almacenamiento de las bases de datos y hechos. Para ello diríjase a la opción ***Project*** del menú y sobre el menú desplegable seleccione la opción **Persist Fact Base**.

Una vez finalizada la inclusión de escenarios, ArchE mostrará en la vista de Escenarios el registro de los mismos. Cada uno de ellos puede ser modificado o consultado las veces que se requieran. El color **Gris** inicialmente le indicará al arquitecto que ArchE aún no posee los suficientes argumentos para evaluar el modelo. El color también persiste cuando los Frameworks de Razonamiento no han sido iniciados. Vaya a la vista de cada uno de los marcos y reinícielos con el botón Start. Si no lo encuentra, diríjase al menú de ArchE en la opción ***Windows > Show View > Other*** y Agregue los dos marcos de **Razonamiento en ArchE External RF Samples**.

Figura 97. Definición de Escenarios para el Sistema CTAS

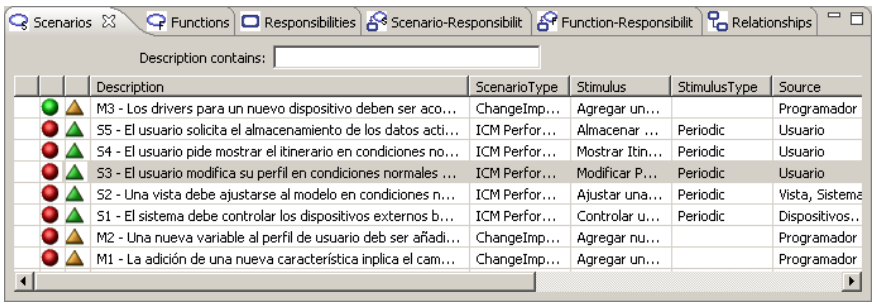
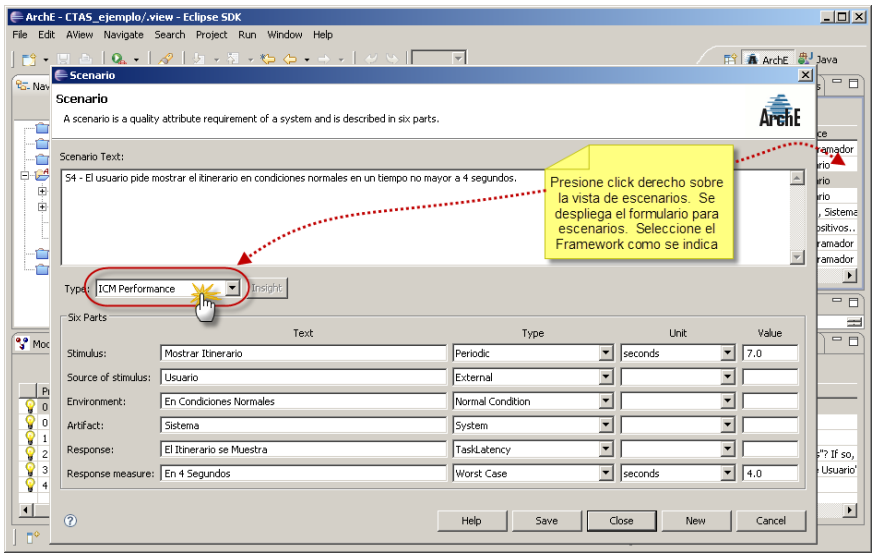
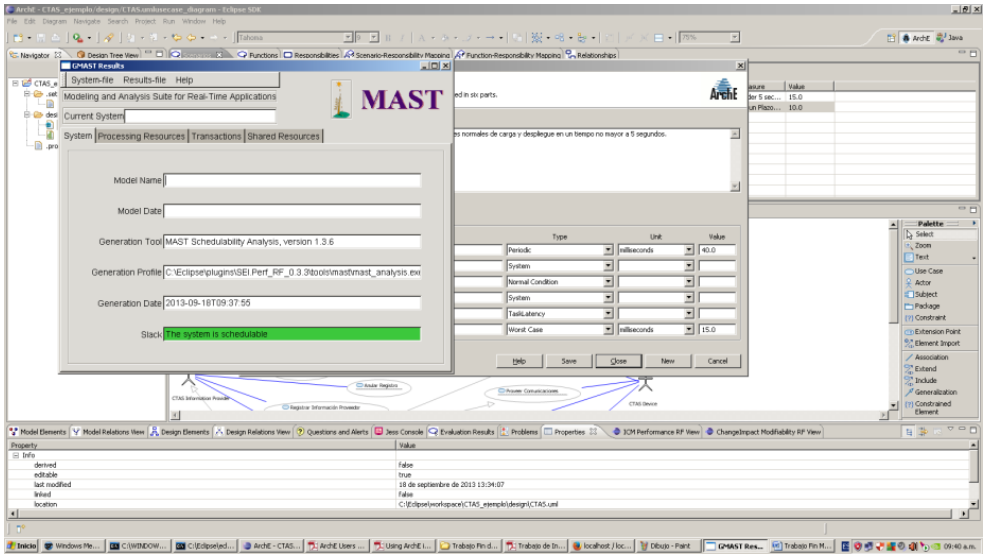


Figura 98. Creación de un Escenario basado en el Atributo Performance



Cuando se crea un escenario a partir del atributo de Performance, la herramienta de ArchE invoca a la herramienta de Análisis MAST. Esta herramienta (Fig. 99) inicia los procesos de simulación, asigna los mecanismos de planificación para el escenario inicial almacenando el estado del mismo. Aparte de esta responsabilidad, MAST genera el conjunto de operaciones y cálculos estocásticos que servirán de referente para el análisis de calidad de la herramienta ArchE a través del Marco de Razonamiento ICM Performance.

Figura 99. Interacción de MAST con el ICM Reasoning Framework de ArchE



A continuación se describirá en detalle cada uno de los escenarios utilizados en el ejemplo, con el respectivo Marco de Razonamiento empleado en cada uno de los escenarios de calidad.

Tabla 39. Escenario M1 para CTAS (ChangeImpact Modifiability RF)

ELEMENTO	ATRIBUTO / VALOR
Descripción del Escenario de Calidad	**M1** - La adición de una nueva característica implica el cambio en la forma de almacenamiento. La implementación de un nuevo formato debe realizarse en un plazo no mayor a 3 días y medio.
Fuente de Estímulo	Programador
Estímulo	Agregar una Nueva Característica.
Ambiente	En Tiempo de Diseño
Artefacto	Sistema
Respuesta	Se ha implementado el formato de almacenamiento
Medida de la Respuesta	En 3.5 días.

Tabla 40. Escenario M2 para CTAS (ChangeImpact Modifiability RF)

ELEMENTO	ATRIBUTO / VALOR
Descripción del Escenario de Calidad	**M2** - Una nueva variable al perfil de usuario debe ser añadida al modelo en un tiempo no mayor a 5 días.
Fuente de Estímulo	Programador
Estímulo	Agregar nueva Variables al Perfil
Ambiente	En Tiempo de Diseño
Artefacto	Sistema
Respuesta	Una Nueva Variable ha sido Agregada
Medida de la Respuesta	En 5 días

Tabla 41. Escenario M3 para CTAS (ChangeImpact Modifiability RF)

ELEMENTO	ATRIBUTO / VALOR
Descripción del Escenario de Calidad	**M3** - Los drivers para un nuevo dispositivo deben ser acoplados por el equipo de desarrollo en un plazo de 8 días.
Fuente de Estímulo	Programador
Estímulo	Agregar un Nuevo Dispositivo
Ambiente	En Tiempo de Diseño
Artefacto	Driver para un Nuevo Dispositivo
Respuesta	El Driver funciona y está acoplado
Medida de la Respuesta	En un Plazo de 8 días

Tabla 42. Escenario S1 para CTAS (ICM Performance RF)

ELEMENTO	ATRIBUTO / VALOR
Descripción del Escenario de Calidad	**S1** - El sistema debe controlar los dispositivos externos bajo condiciones normales de carga y despliegue en un tiempo no mayor a 5 segundos.

ELEMENTO	ATRIBUTO / VALOR
Fuente de Estímulo	Dispositivos Externo
Estímulo	Controlar un Dispositivo Externo
Ambiente	En Condiciones Normales
Artefacto	Sistema
Respuesta	El Driver Controla el Dispositivo
Medida de la Respuesta	En 5 Segundos

Tabla 43. Escenario S2 para CTAS (ICM Performance RF)

ELEMENTO	ATRIBUTO / VALOR
Descripción del Escenario de Calidad	**S2** - Una vista debe ajustarse al modelo en condiciones normales en un tiempo no mayor a 1 segundo.
Fuente de Estímulo	Vista, Sistema
Estímulo	Ajustar una Vista al Modelo
Ambiente	En Condiciones Normales
Artefacto	Sistema
Respuesta	La Vista ha sido Agregada al Modelo
Medida de la Respuesta	En 1 Segundo

Tabla 44. Escenario S3 para CTAS (ICM Performance RF)

ELEMENTO	ATRIBUTO / VALOR
Descripción del Escenario de Calidad	**S3** - El usuario modifica su perfil en condiciones normales en un tiempo no mayor o igual a 5 segundos.
Fuente de Estímulo	Usuario
Estímulo	Modificar Perfil
Ambiente	En Condiciones Normales
Artefacto	Sistema
Respuesta	El Perfil ha sido Modificado
Medida de la Respuesta	En 5 segundos

Tabla 45. Escenario S4 para CTAS (ICM Performance RF)

ELEMENTO	ATRIBUTO / VALOR
Descripción del Escenario de Calidad	**S4** - El usuario pide mostrar el itinerario en condiciones normales en un tiempo no mayor a 4 segundos.
Fuente de Estímulo	Usuario
Estímulo	Mostrar Itinerario
Ambiente	En Condiciones Normales
Artefacto	Sistema
Respuesta	El Itinerario se Muestra
Medida de la Respuesta	En 4 Segundos

Tabla 46. Escenario S5 para CTAS (ICM Performance RF)

ELEMENTO	ATRIBUTO / VALOR
Descripción del Escenario de Calidad	**S5** - El usuario solicita el almacenamiento de los datos activos de la pantalla de su PDA en condiciones normales en un tiempo no mayor a 3 segundos.
Fuente de Estímulo	Usuario
Estímulo	Almacenar Datos
Ambiente	En Condiciones Normales
Artefacto	Sistema
Respuesta	Los Datos han sido Almacenados
Medida de la Respuesta	En 3 segundos

9.3.3. Paso 3 – Selección del Framework de Razonamiento para CTAS

El Framework de Razonamiento para los escenarios en CTAS se fundamenta en los Marcos ICM Performance RF para el atributo de calidad Rendimiento, y ChangeImpact Modiafibility RF, para el atributo de Modificabilidad. Al diseñar los escenarios, estos Frameworks son seleccionados para completar el escenario.

9.3.4. Paso 4 – Construcción del Escenario de Calidad en CTAS

9.3.4.1. Definición de Funcionalidades para el sistema CTAS

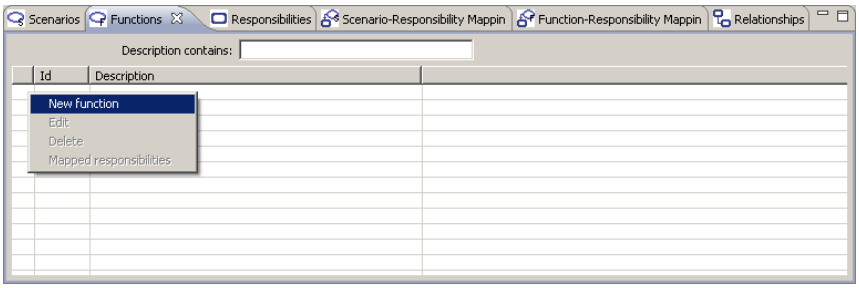
El segundo paso para el diseño de la arquitectura, consiste en la definición de las funcionalidades del sistema. Las funcionalidades o funciones corresponden a una descripción de los rasgos o características que posee el sistema y que satisfacen los requerimientos del modelo. En ArchE, las Funciones están representadas por un valor numérico o identificador y una descripción (Tabla 47).

Las Funciones de un sistema pueden sufrir descomposiciones y tras cada descomposición ArchE agrega el valor del indicador para indicar que existe una relación padre-hijo entre este grupo de funciones. Las funciones son traducidas por ArchE posteriormente como una Responsabilidad. Cada escenario para Funcionalidades puede ser editado, modificado o eliminado conforme a las necesidades del arquitecto

9.3.4.2. Agregar Funcionalidades

El paso a continuación corresponde al ingreso de las funciones, las cuales harán parte del conjunto de Responsabilidades del sistema. Diríjase a la vista Functions de ArchE, presione click derecho y proceda a crear una a una las funcionalidades.

Figura 100. Agregar una Nueva Funcionalidad



El proceso anterior desplegará una ventana de diálogo para ingresar la funcionalidad. Esta asocia un Id a una Descripción. Procure que la descripción de la funcionalidad inicie con verbos infinitivos "con terminación en ar – er – ir ".

Figura 101. Conjunto de Funcionalidades en CTAS

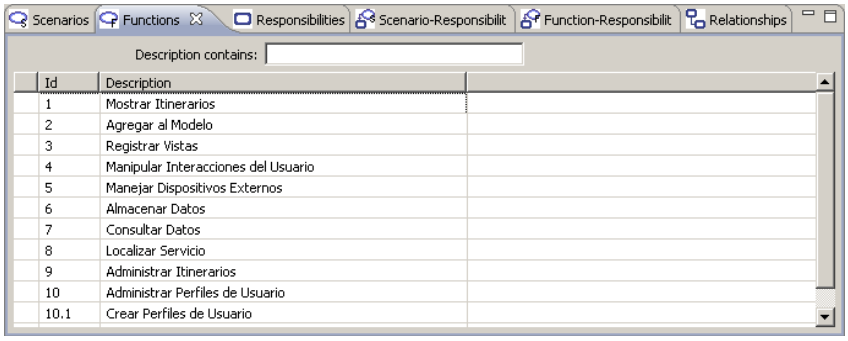
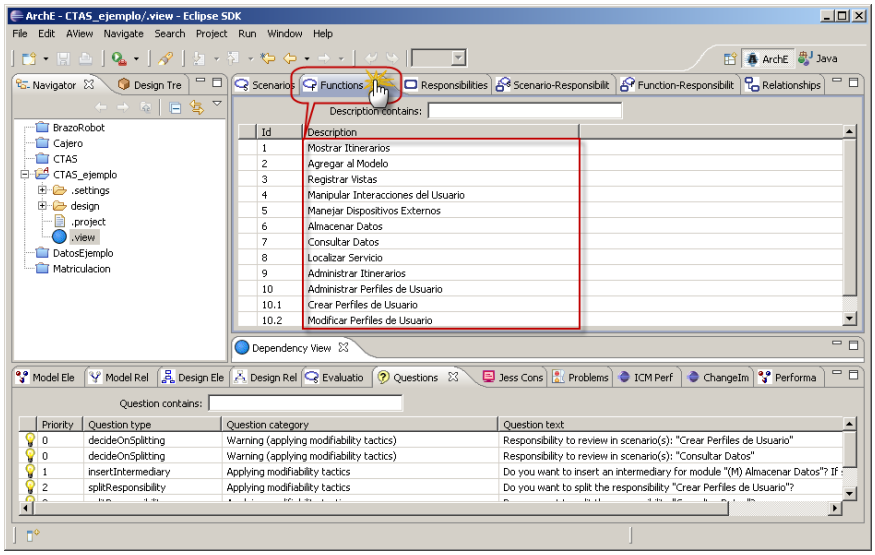


Tabla 47. Listado de Funcionalidades para el Sistema CTAS

	ID	DESCRIPTOR / DESCRIPCIÓN
Funciones	**1**	Mostrar Itinerarios
	2	Agregar al Modelo
	3	Registrar Vistas
	4	Manipular Interacciones del Usuario
	5	Manejar Dispositivos Externos
	6	Almacenar Datos
	7	Consultar Datos
	8	Localizar Servicio
	9	Administrar Itinerarios
Función y su respectiva Descomposición	**10**	Administrar Perfiles de Usuario
	10.1	Crear Perfiles de Usuario
	10.2	Modificar Perfiles de Usuario

La representación de las funciones para el sistema CTAS se muestra a continuación.

Figura 102. Registro de Funcionalidades para el sistema CTAS



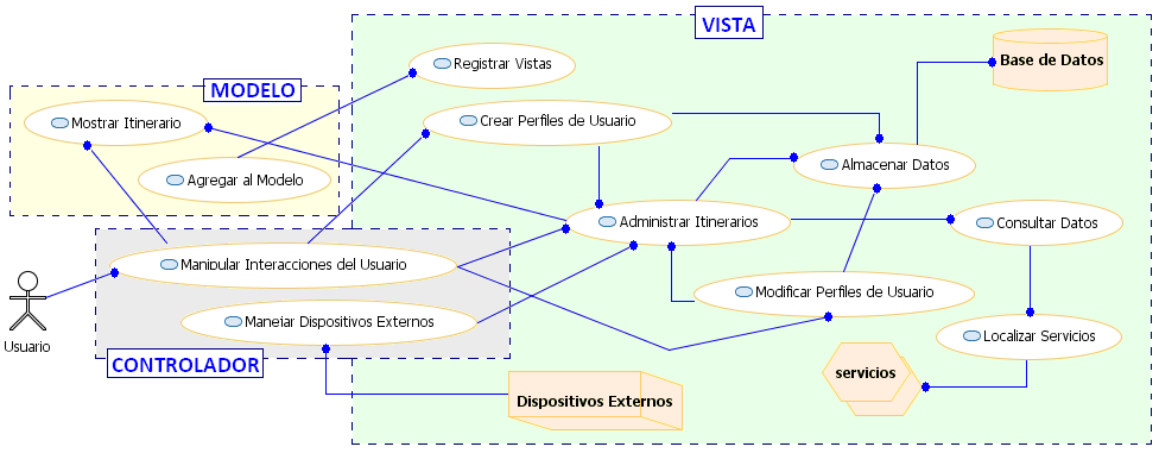
9.3.4.3. Definición de Responsabilidades para el sistema CTAS

El tercer paso consiste en definir el conjunto de Responsabilidades. Una Responsabilidad corresponde a una tarea específica que el sistema debe realizar. Las Responsabilidades son un concepto básico en ArchE y se generan de forma automática a partir de la definición de las Funcionalidades o tras la aplicación de una determinada Táctica arquitectónica. Estas responsabilidades pueden ser introducidas por el arquitecto y serán desplegadas a través de las vistas de diseño y de igual manera pueden ser modificadas conforme a un atributo de calidad específico.

9.3.4.3.1. Ingresando Responsabilidades

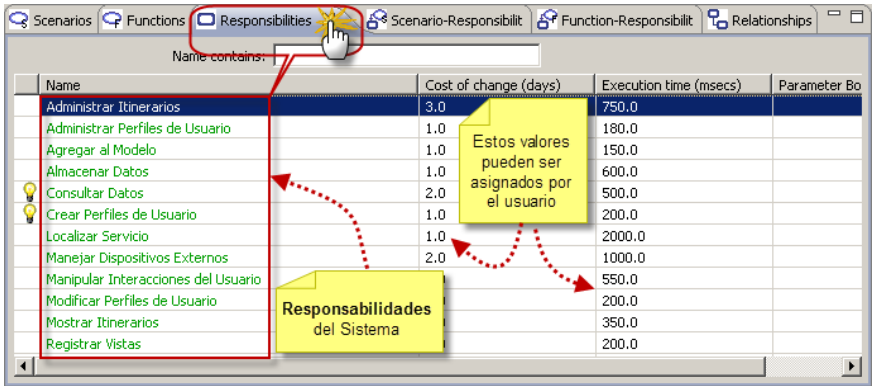
El sistema CTAS adopta la arquitectura MVC (Modelo-Vista-Controlador), a partir de la cual se define el conjunto de responsabilidades del sistema. La siguiente gráfica representa dichas responsabilidades en el modelo.

Figura 103. Gráfico de Responsabilidades para CTAS



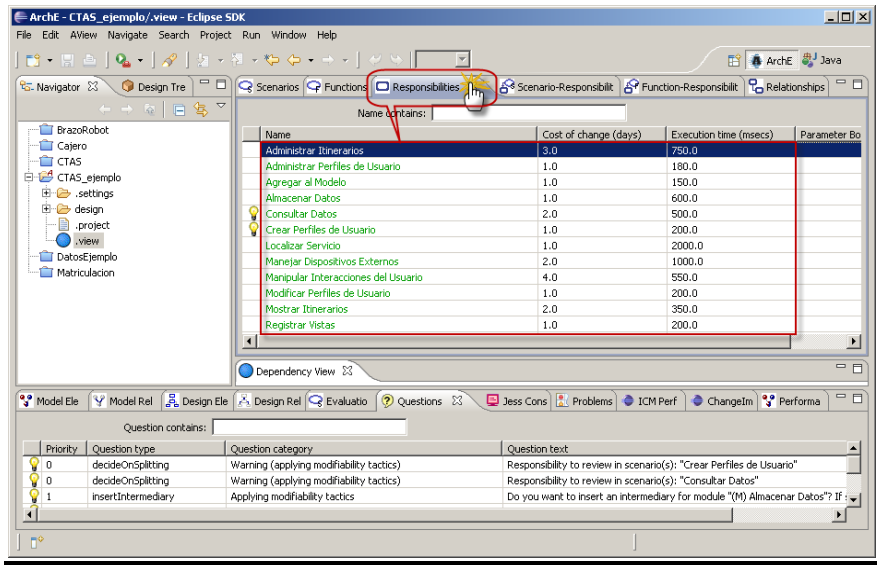
Una Responsabilidad corresponde a una actividad en particular que realiza el sistema. Las Responsabilidades son generadas de forma automática por ArchE y ésta misma asigna a cada una de ellas un costo promedio de ejecución "Execution Time", que por lo general es equivalente para todas las Responsabilidades. Dependiendo del caso el usuario podrá modificar el valor asignado sin sobrepasar el valor de 10, ya que el algoritmo del marco de razonamiento usa este límite para asignar tiempo en el procesador.

Figura 104. Conjunto de Responsabilidades



Una vez generada la interfaz para las Responsabilidades del sistema (las cuales se generan de forma automática), ArchE toma la Respuesta de Medida de Los escenarios y los coloca como valor de referencia para evaluar el Costo del cambio "***Cost of Change***" el cual se visualiza en la columna No.2 de la interfaz de Responsabilidades (Fig. 104). En la tercera columna se presenta el atributo "***Execution Time (msec)***", una medida que inicialmente es asignada por ArchE para la evaluación de las Funciones, pero que puede ser modificada por el usuario para mejorar los aspectos de calidad en el modelo de diseño. Los valores para este atributo deben estar entre 0.0 y 10.0 milisegundos.

Figura 105. Registro de Responsabilidades para el sistema CTAS

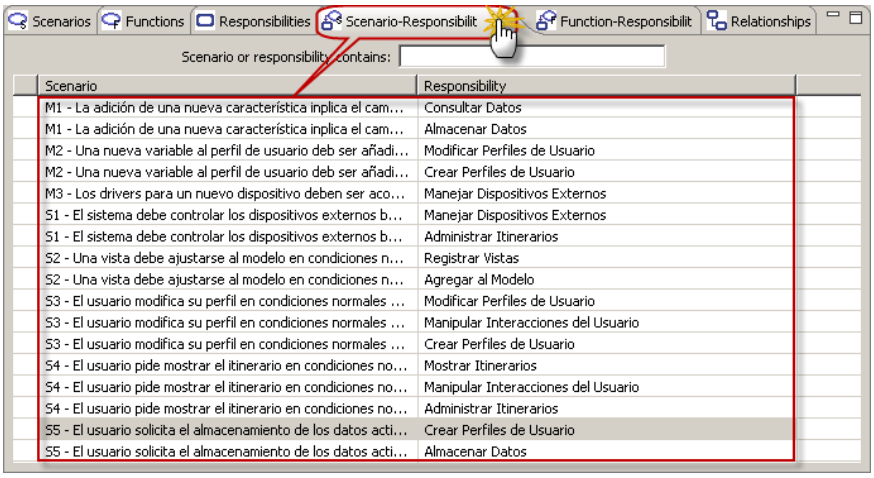


Las Responsabilidades se utilizan para conectar el modelo con los Marcos de Razonamiento. Al aplicar un conjunto de Tácticas se produce un nuevo conjunto de responsabilidades que serán incluidas en el modelo y que será analizadas por la arquitectura de ArchE. Las relaciones dispuestas por el arquitecto para cada responsabilidad contienen la materia prima para el análisis de los Marcos de Razonamiento (relevancia de responsabilidades en base a prioridades). Si una responsabilidad está contenida en otra, el Marco de Razonamiento utiliza el mecanismo de Modificabilidad para guiar el proceso de descomposición de los módulos.

9.3.4.4. Mapeo de Escenarios / Responsabilidades para el sistema CTAS

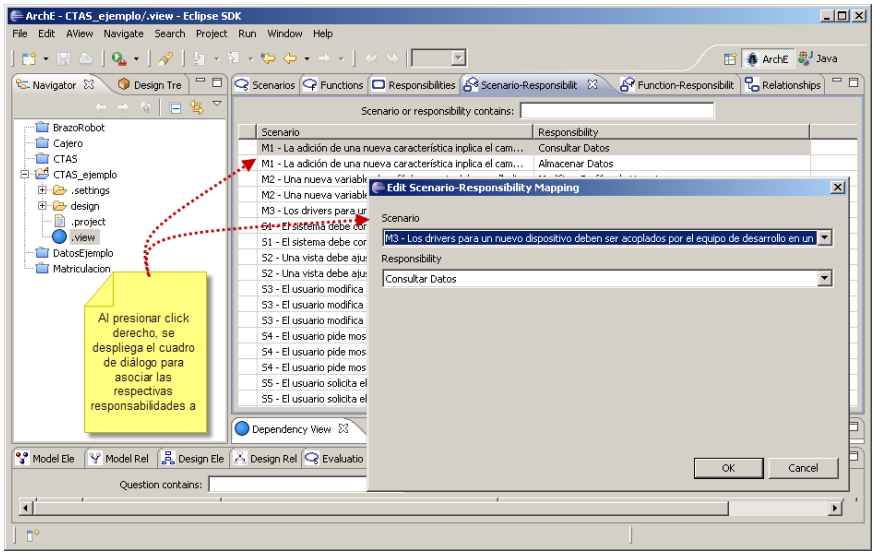
El siguiente paso consiste en definir el mapeo de las responsabilidades con respecto a los escenarios. Este procedimiento especifica cuáles son las Responsabilidades que son afectadas en un determinado escenario. El mapeado de Responsabilidades /Escenarios se muestra en la Figura anterior.

Figura 106. Listado de Asociaciones entre Escenarios y Responsabilidades



Para agregar una asociación entre Escenarios y Responsabilidades, en la vista presione click derecho y selecciones de acuerdo al criterio del modelo las respectivas asociaciones como se muestra a continuación. Cada escenario puede estar asociado a muchas responsabilidades y viceversa.

Figura 107. Mapeando Escenarios a Responsabilidades

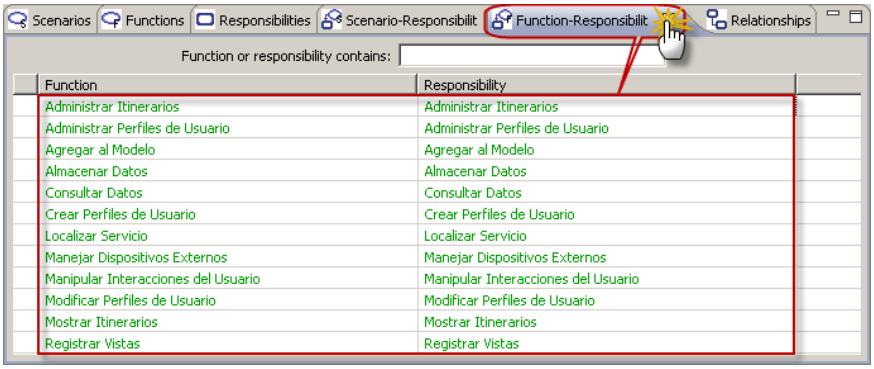


9.3.4.5. Mapeo de Funciones / Responsabilidades para el sistema CTAS

El siguiente paso consiste en definir el mapeo de las responsabilidades con respecto a los Funcionalidades del sistema. Este procedimiento especifica cuáles son las Responsabilidades que son afectadas por una determinada funcionalidad. Este proceso

se genera de forma automática. La vista le permitirá visualizar los datos referidos al mapeo entre Funcionalidades y Responsabilidades del sistema.

Figura 108. Listado de Asociaciones entre Funciones y Responsabilidades



El último paso en el proceso de requerimientos para la arquitectura ArchE corresponde a la definición de Relaciones. A partir de éstas se definen las dependencias entre las diferentes Responsabilidades. Existen dos posibilidades de relación para este conjunto de elementos: de ***Dependencias***, de ***Contenencia*** y de ***Reacción***.

9.3.4.6. Descripción de Relaciones "Relationships" para el sistema CTAS

La vista de Relaciones ArchE se utiliza para ingresar las dependencias entre las distintas responsabilidades. Una Responsabilidad puede contener a otras Responsabilidades o una Responsabilidad puede proveer información a otra Responsabilidad o conjunto de éstas. La columna ***Valor*** de la vista de Relaciones, muestra la probabilidad de que al realizar un cambio en una responsabilidad esta se propague a través de las relaciones o dependencias forzando un cambio en la respectiva responsabilidad.

Las relaciones entre responsabilidades se pueden dar en base a 3 características:

- **Contenencia**: Indica que una Responsabilidad tiene varios hijos;
- **Dependencia**: Indica que una responsabilidad depende de otra y;
- **Secuencia o Reacción**: Indica que la primera responsabilidad debe ejecutarse antes de que la segunda sea ejecutada; A → B.

Para ingresar las respectivas relaciones, presione click derecho sobre la vista de Relationship de ArchE. No se preocupe si asocia mal a las responsabilidades, el sistema permite modificarlas. Recuerde para todos los casos Salvar los cambios.

La inclusión de una relación entre una o más funcionalidades, generará las respectivas dependencias entre clases, y a partir de ellas, ArchE genera el modelo que será

mostrado a través de la vista de diseño del Plug-in GEF de Eclipse, desde el cual se realizarán las modificaciones al modelo cada vez que se aplique una nueva Táctica. A continuación se muestra el listado de Relaciones para el sistema CTAS.

Figura 109. Relaciones / Dependencias en CTAS

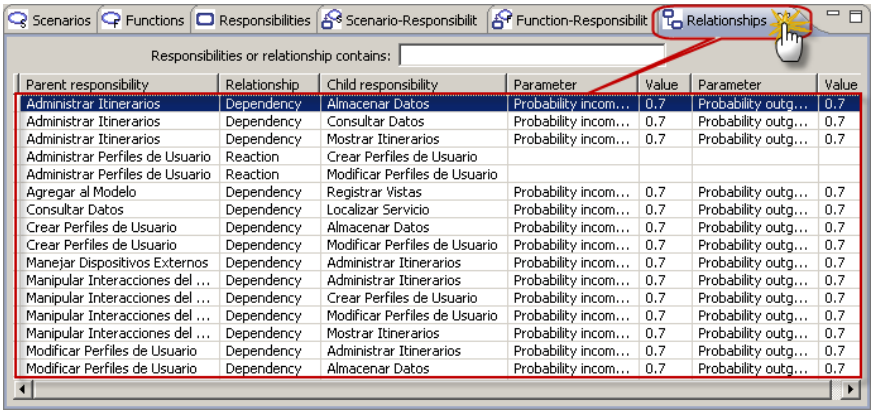
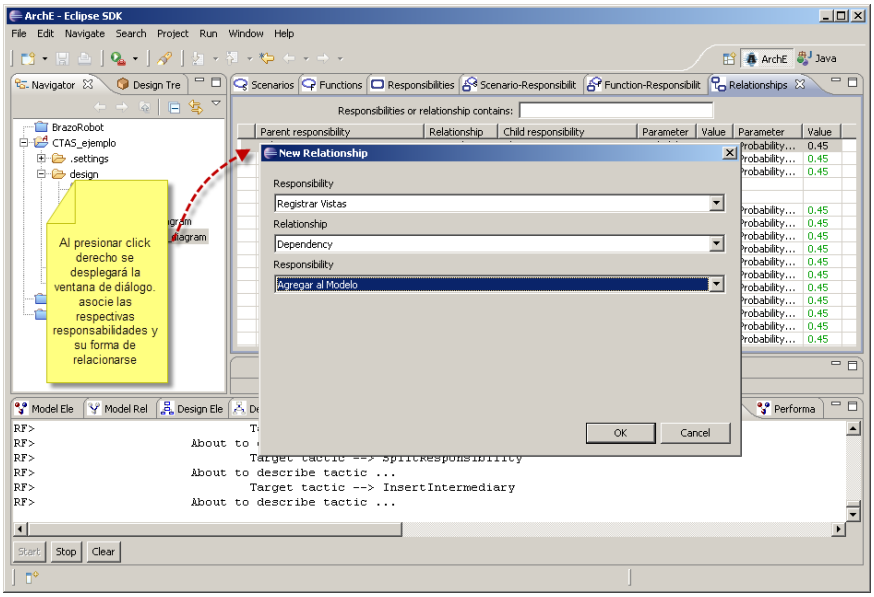


Figura 110. Agregando un Conjunto de Relaciones para el Caso Práctico



9.3.5. Paso 5 – Generación del Modelo para CTAS

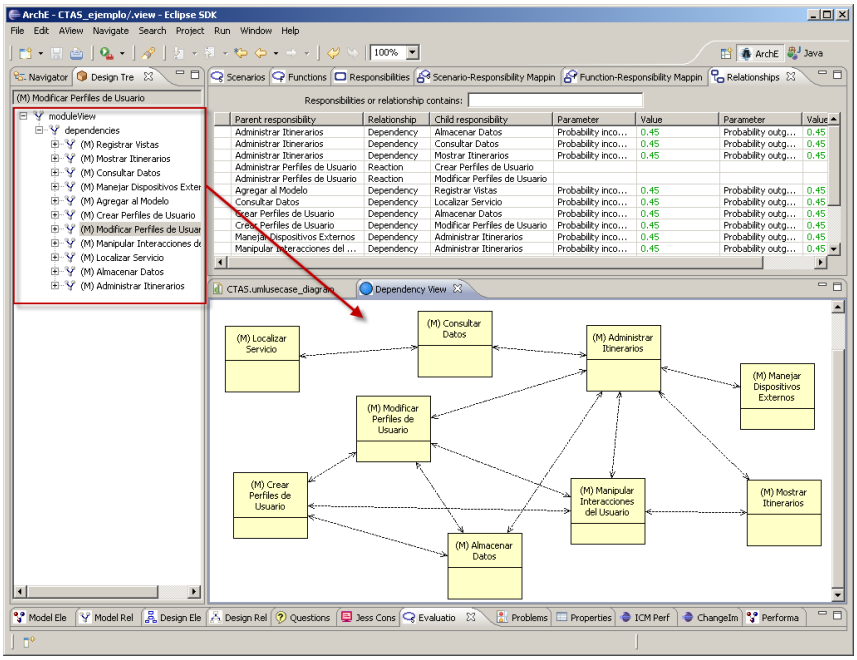
9.3.5.1. Generación y Optimización del Modelo

El paso final consiste en verificar los resultados del proceso de análisis y dar aplicación a las tácticas sugeridas por ArchE. El registro de análisis muestra como ArchE

ejecuta el proceso de testing sobre los 4 Escenarios propuestos en el modelo de ejemplo. Para acceder al conjunto de Tácticas sugeridas ubique la vista Questions y allí encontrará las sugerencias que hace ArchE para mejorar la arquitectura.

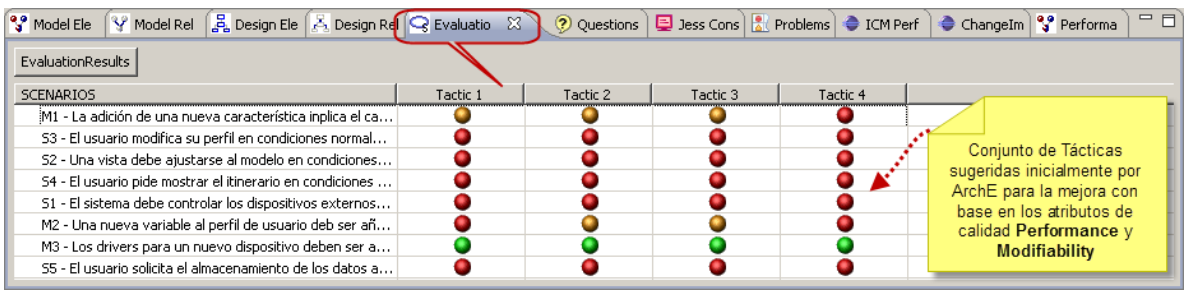
Recuerde que tras cada proceso de adición o modificación de los atributos en los escenarios u otras funcionalidades del sistema, el usuario debe salvar los cambios en las bases de datos y en la base de hechos del Sistema Experto. Para ello diríjase a la opción ***Project*** del menú y sobre el menú desplegable seleccione la opción **Persist Fact Base**.

Figura 111. Modelo Inicial proporcionado por ArchE



Una vez se ha concluido con la lista de requerimientos, el arquitecto puede verificar el estado de las tácticas y el análisis hecho por ArchE sobre la arquitectura. En la vista inferior "Evaluation" se puede observar el resultado inicial del análisis del modelo.

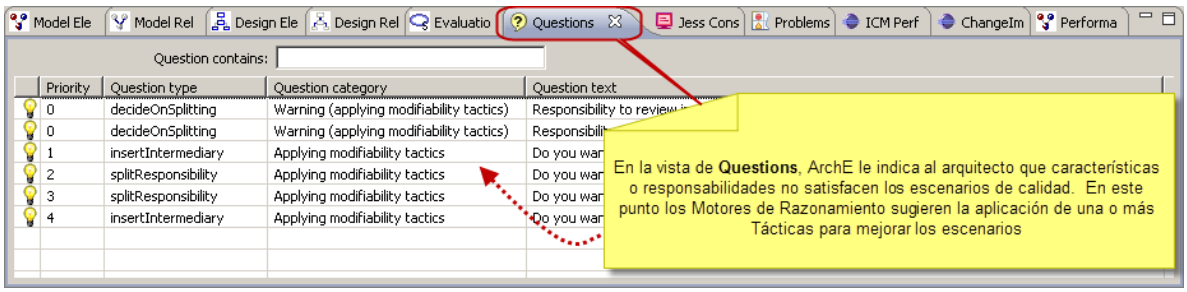
Figura 112. Vista "Evaluation" de ArchE en CTAS



Una vez ArchE ha capturado todos los requerimientos proporcionados por parte del arquitecto, el Sistema Experto propone un diseño inicial e identifica las posibles inconsistencias en la información proporcionada a través de los diferentes escenarios. Si el modelo iterado necesita de alguna variable o dato específico, el Sistema Experto pedirá al diseñador la introducción de tales elementos para evaluar nuevamente las responsabilidades y dependencias.

Para verificar los ajustes o sugerencia de Tácticas a aplicar, vaya a la parte inferior en la vista "Questions". En este interfaz el sistema experto presentará las inconsistencias del modelo y sugerirá la Táctica a aplicar. La vista "***Questions***" se encuentra en la parte inferior de la vista de dependencias donde se desplegó el diagrama UML. Observe como en la vista de Alertas y Preguntas, el sistema ArchE indica con un ícono en forma de bombilla encendida el ajuste y la táctica a sugerir para la mejora del modelo.

Figura 113. Vista de Preguntas y Alertas en ArchE



La siguiente tabla relaciona algunas preguntas formuladas al arquitecto por parte del Sistema Experto una vez se ha iniciado la primera iteración o análisis preliminar de la arquitectura.

Tabla 48. Propuesta Inicial de Tácticas aplicadas por ArchE a CTAS

PRIORIDAD	TIPO	CATEGORÍA	DESCRIPCIÓN
 0	Decidir si se Divide o no las Responsabilidades	Advertencia (Aplicar táctica de Modificabilidad)	Dividir la Responsabilidad "Crear Perfiles de Usuario" en 2 partes no siempre decrementa el costo del escenario relacionado. Es necesario que decida posteriormente cuales responsabilidades resultantes (hijas) estarán afectadas por el cambio.
 1	Insertar Intermediario	Aplicar táctica de Modificabilidad	La dependencia de responsabilidades en el módulo "Almacenar Datos" impacta a uno de los escenarios. La Inserción de un intermediario entre dos o más módulos, puede reducir el costo del escenario.
 2	Reasignar / Dividir Responsabilidades	Aplicar táctica de Modificabilidad	La Responsabilidad "Crear Perfiles de Usuario" tiene múltiples dependencias o se relaciona con múltiples Responsabilidades.

	Eliminando las fuertes dependencias, se reduce el acoplamiento entre componentes. El costo en los escenarios y responsabilidades puede reducirse Sí se divide la Responsabilidad "Crear Perfiles de Usuario" en 2 Responsabilidades hijas, lo que minimizará las dependencias.

El orden y numeración de la lista inicial de tácticas sugeridas, responde a un conjunto de prioridades asignadas por ArchE dependiendo de la relevancia del proceso. (Éstas prioridades con asignadas por el conjunto de Reglas contenidas en la base de hechos y de conocimiento del sistema experto). La idea general de aplicar una determinada Táctica es mejorar el diseño.

Para verificar el estado de inconsistencias en el modelo y el costo que causará aplicar la determinada Táctica, ArchE permite al arquitecto consultar el estado de la misma a través de un cuadro de diálogo completo que se despliega presionando click derecho sobre la línea donde se encuentra la respectiva salida. Este procedimiento desplegará un menú emergente con la opción "***Open***" la cual mostrará en detalle lo que el sistema experto modificará, que Táctica aplicará y cuál será el costo de aplicar dicha modificación. Este procedimiento actualizará los comportamientos y las dependencias de los componentes del modelo y de esta manera se producirá la respectiva iteración del modelo.

Figura 114. Desplegando Vistas para Aplicación de Tácticas

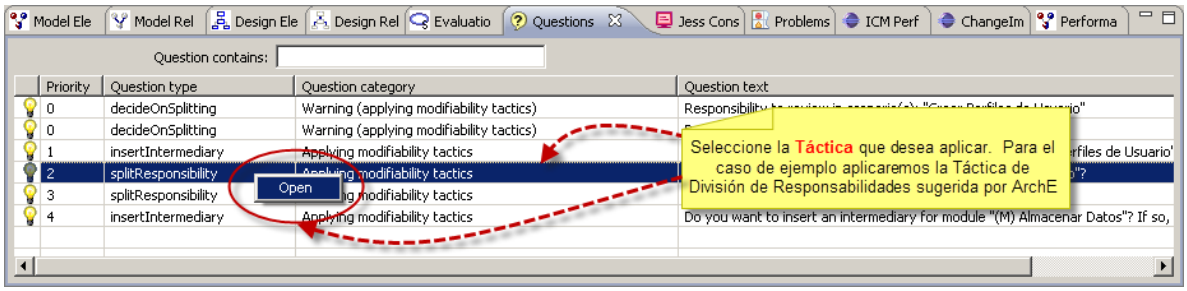


Figura 115. Aplicando Táctica Sugerida (Modificabilidad) a CTAS

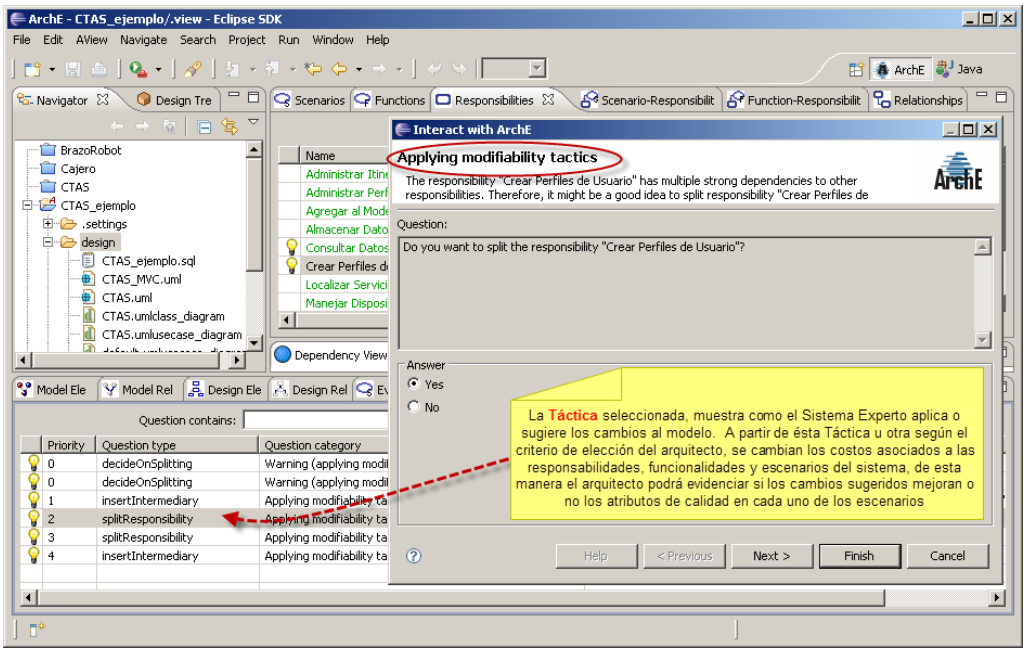
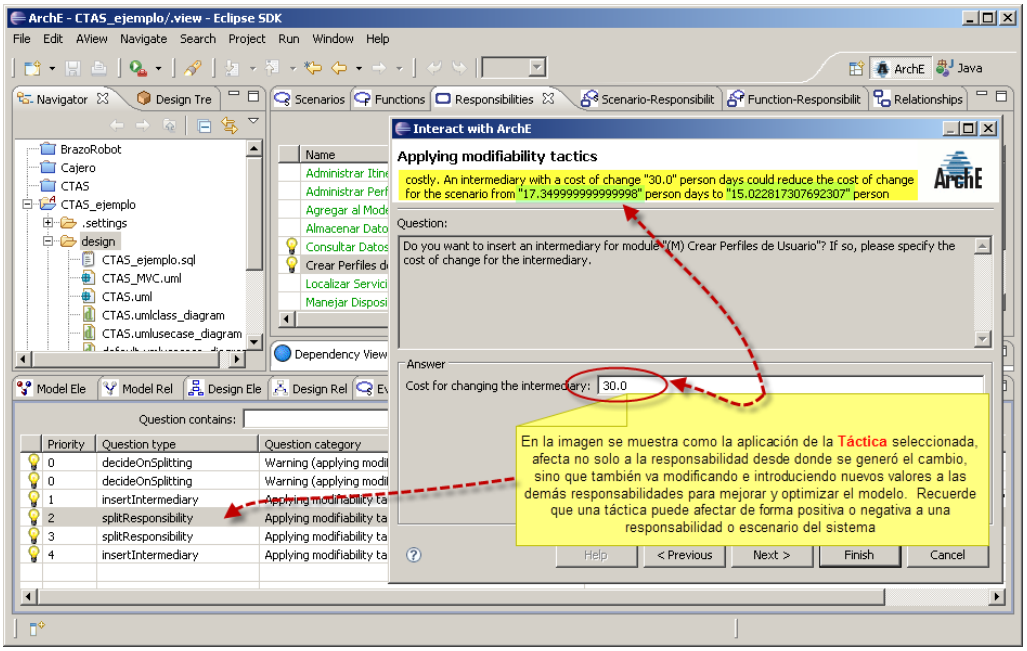


Figura 116. Ejemplo de Una Táctica de Modificabilidad Aplicada a CTAS



En el caso anterior (Fig. 115 y 116), ArchE ha sugerido la aplicación de la Táctica que dividirá una determinada responsabilidad para disminuir el acoplamiento entre responsabilidades. Para ello el sistema experto sugiere en su orden los siguientes cambios y afectaciones en el modelo:

- La división “Split” de responsabilidades en el escenario "*Crear Perfiles de Usuario*" ya que ésta posee fuertes dependencias con otros módulos o componente del modelo.

- La responsabilidad “*Consultar Datos*” también posee fuertes dependencias con otras responsabilidades y por ello ArchE sugiere dividir dicha Responsabilidad en dos hijos (nuevas responsabilidades heredadas). El costo estimado por ArchE para esta Táctica minimizará la dependencia entre responsabilidades padre-hijo y creará una nueva clase para reducir la dependencia entre módulos.

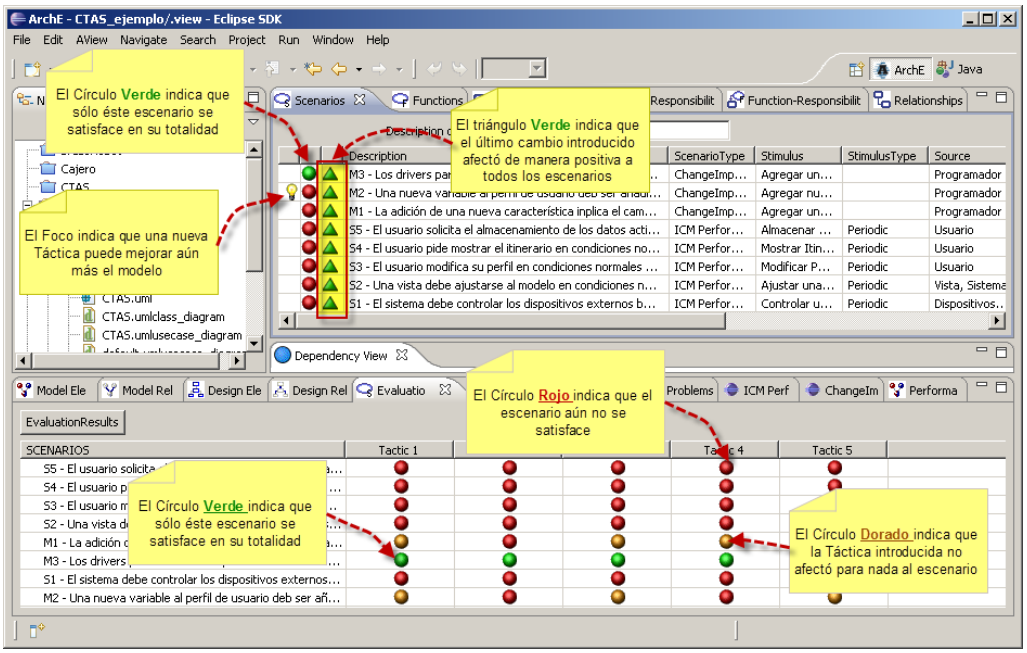
- El cambio en las dependencias del módulo “Crear Perfiles de Usuario” dará como resultado un impacto positivo en uno de los escenarios, si se inserta un módulo intermediario para reducir los costos. Un módulo intermedio “intermediario” con un costo de 30 personas/día reducirá el costo del escenario de "17.349999999999998" personas/día a "15.022817307692307" personas/día.

- Las dependencias de la Responsabilidad “Almacenar Datos” impacta a uno de los escenarios. Este cambio positivo puede ser benéfico si se inserta un intermediario para reducir el costo. Un intermediario con costo de 30 personas/día, reducirá el costo del cambio en el escenario de "21.179999999999996" personas/día a "13.480192307692306" personas/día.

- Al dividir la Responsabilidad “Consultar Datos” en 2 partes, no siempre asegura la disminución en los costos totales de los escenarios de calidad. El Arquitecto debe decidir posteriormente si aplica o no las Responsabilidades (hijas) generadas al modelo.

Es importante que el arquitecto tenga en cuenta que cada modificación o Táctica aplicada no corresponde a una solución de última milla. Para ArchE, cada Táctica es una posible sugerencia de mejora al modelo y la aplicación de una o más Tácticas no solo modificará o afectará al componente donde se aplica la Táctica, sino que estas decisiones pueden afectar a otras responsabilidades, por esta razón en el modelo de análisis para la arquitectura ArchE es posible encontrar diversas iteraciones que terminarán cuando los atributos y las Tácticas satisfagan por completo el modelo. Para verificar si la introducción de Tácticas ha afectado positiva o negativamente al modelo, diríjase a la vista “***Evaluations***”, como lo indica la siguiente imagen. Allí verá las tácticas aplicadas y las afectaciones al modelo “Mejora o Degradación”.

Figura 117. Vista de Evaluación de Tácticas en CTAS



En la vista de Evaluación, un círculo **Verde** indica que el escenario será satisfecho si se aplica la táctica sugerida. El círculo **Rojo** indica que de acuerdo a los parámetros especificados por el escenario, la aplicación de la táctica degradará la funcionalidad del escenario, mientras que el círculo **Dorado** indica que los cambios introducidos no afectarán en ningún momento al modelo y no tendrán impacto alguno sobre la arquitectura. En esta primera iteración se produce una nueva Táctica para mejora.

La vista "Evaluación de Resultados" resume el resultado de aplicar las tácticas potenciales a nuestro diseño. Los escenarios se muestran en filas y las columnas representan las tácticas que están siendo evaluadas por ArchE. Esta vista contiene además un botón "***EvaluationResults***" o "Evaluación de Resultados", Al hacer clic sobre este éste hará que ArchE cambie el aspecto de los íconos "de círculos a Triángulos". Este cambio indica o supone aspectos de Mejora o Degradación en el modelo dependiendo de si se aplica o no la Táctica.

La aplicación de Tácticas concluye cuando todos los escenarios de calidad han sido satisfechos en su totalidad. La imagen a continuación solo muestra la aplicabilidad de 2 tácticas "tras cada iteración", pero el modelo se extiende hasta tanto no se cumplan todos los escenarios; es decir, hasta que todos los escenarios y su respectivo círculo esté en color **Verde** como lo indica la vista de Evaluación.

Figura 118. Vista de Evaluación "Evaluación y Resultados"

Cada cambio introducido a los escenarios o a las relaciones entre las distintas responsabilidades del modelo es detectado por los Marcos de Razonamiento. Tras cada modificación el sistema, por ejemplo en el Marco ICM Performance, se realizarán los cálculos respectivos del escenario y se actualizará la información y procesará la salida en su respectivo archivo .mast. A continuación se lista el log de procesos del ICM Performance, como parte de los resultados entregados por ArchE al arquitecto. La información actualizada también afecta los registros en las bases de datos de MySQL donde se guardan los resultados de cada escenario, tácticas aplicadas, lista de responsabilidades, etc.

ICM File - Architecture1.icm

```
RF> Service started
RF> [AnalyzeAndSuggest starting...]
RF>     Current Project Name = CTAS_ejemplo
RF>     Current Architecture Name = Architecture1
RF>     Creating design model inputs...
RF>             Recovering related responsibilities: 2 - analyze on version= 16570
RF>             Number of responsibility dependencies (structure) --> 14
RF>             Running analysis for the scenario "M1 - La adición de una nueva característica implica
                el cambio en la forma de almacenamiento.  La implementación de un nuevo formato debe
                realizarse en un plazo no mayor a 3 días y medio."
RF>                     Number of modules in the view --> 11 for 2 primary responsibilities
RF>                     Number of module dependencies in the view --> 14
RF>                     Scope rate for modules --> 0.5454545454545454 (primary ones versus total)
RF>                     Scope rate for responsibilities --> 0.5 (primary ones versus total)
RF>                     Average module cost (initial) --> 10.0
RF>                     Average module cost (computed) --> 6.495909090909091
RF>                     Average responsibility cost (initial) --> 4.833333333333334
RF>                     Average responsibility cost (computed) --> 4.833333333333334
RF>                     Average module cohesion --> 0.8284848484848485
RF>                     Average module coupling --> 0.5848484848484848
RF>                     Average rippling --> 0.05206611570247935
RF>             Analysis result = 16.455 (NOT SATISFIED) reference= 3.0
RF>                     Setting responsibility target 1 --> Consultar Datos
RF>                     Setting module target 1 --> (M) Almacenar Datos
RF>                             tactic --> SplitResponsibility
RF>                             tactic --> InsertIntermediary
RF>                     Tactics suggested = 2
RF>             Recovering related responsibilities: 2 - analyze on version= 16570
RF>             Number of responsibility dependencies (structure) --> 14

RF>             Running analysis for the scenario "M2 - Una nueva variable al perfil de usuario ha sido
                agregada 5 días antes del esfuerzo"
RF>                     Number of modules in the view --> 11 for 2 primary responsibilities
RF>                     Number of module dependencies in the view --> 14
RF>                     Scope rate for modules --> 0.45454545454545453 (primary ones versus total)
RF>                     Scope rate for responsibilities --> 0.4166666666666667 (primary ones versus
                        total)
RF>                     Average module cost (initial) --> 10.0
RF>                     Average module cost (computed) --> 7.0375000000000005
RF>                     Average responsibility cost (initial) --> 5.166666666666667
RF>                     Average responsibility cost (computed) --> 5.166666666666667
```

```
RF>                     Average module cohesion --> 0.8050909090909092
RF>                     Average module coupling --> 0.6472727272727273
RF>                     Average rippling --> 0.05950413223140497
RF>             Analysis result = 13.412499999999998 (SATISFIED) reference= 15.0
RF>             Recovering related responsibilities: 1 - analyze on version= 16570
RF>             Number of responsibility dependencies (structure) --> 14

…… // continua el proceso de análisis de otros escenarios

RF>     Sent analysis results!!!
RF>         [DescribeTactic starting...]
RF>     Current Project Name = CTAS_ejemplo
RF>     Current Architecture Name = Architecture1
RF>                     About to describe tactic ...
RF>                             Target tactic --> SplitResponsibility
RF>                     About to describe tactic ...
RF>                             Target tactic --> InsertIntermediary
RF>     Sent a user question list!!!
```

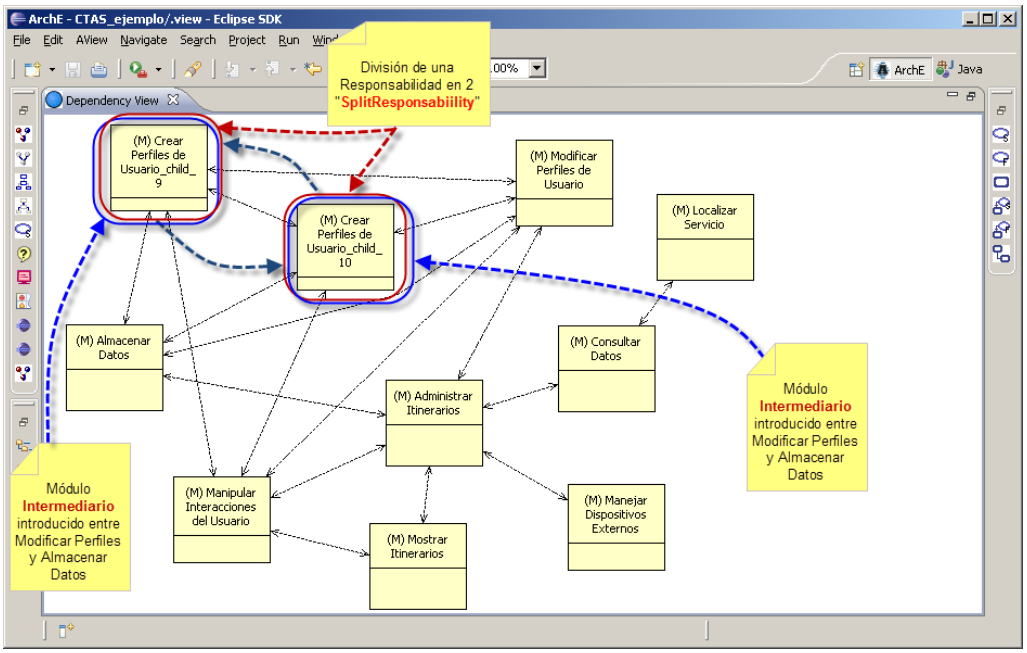
A partir de estos contextos, el arquitecto puede revisar el resultado de los procesos realizados por el marco de Razonamiento en la vista ChangeImpact Modifiabilitty RF View de la aplicación, que traducidas a un nivel más abstracto, corresponde a las tácticas sugeridas por ArchE y que pueden ser revisadas por el arquitecto en la vista "***Questions***" de ArchE (Fig.112). La vista ICM Performance RF view hace lo propio y despliega los cálculos del contexto al usuario a través de la vista.

9.3.5.2. Modelo propuesto por ArchE en CTAS tras aplicar una Táctica

Cada vez que ArchE sugiere un conjunto de Tácticas, estas decisiones de diseño afectan a todo el modelo. Como consecuencia de ello ArchE producirá tras cada iteración (entiéndase por iteración al proceso de cambio tras cada aplicación de una determinada Táctica), un nuevo modelo con sus respectivas dependencias. Este modelo puede visualizarse en forma de Grafo UML, como se muestra en la Fig.119.

Hay que recordar que cada Táctica es solo una **DECISIÓN** de diseño que puede ser aplicada o no por el arquitecto. La herramienta del SEI jamás pretende cambiar la perspectiva de diseño y la capacidad de modelar sistemas en el ser humano. Esta iniciativa que aún está en desarrollo es una herramienta que sirve como soporte para el diseño de arquitecturas a partir de modelos, responsabilidades y un conjunto de escenarios.

Figura 119. Propuesta de diseño introducida por ArchE tras una Iteración



De esta forma la evaluación de las posibles arquitecturas es mucho más rápida. El descubrimiento y la eliminación de conflictos entre los requisitos de los atributos de calidad se pueden realizar tan pronto como estos aparecen.

9.3.5.3. Guardar Cambios en la Arquitectura

ArchE no específica en sus manuales de referencia, el proceso de almacenamiento de los datos y escenarios dispuestos en el modelo de análisis. Uno de los inconvenientes al inicio de la práctica fue precisamente que dicha información no se almacenaba en un registro físico para recuperar más adelante el modelo y sus datos (escenarios, modificaciones o iteraciones hechas al modelo). La solución se encontró después de horas de exploración de la herramienta y es muy sencilla la implementación. Solo basta con almacenar la **FactBase**. A partir de este proceso se guardarán todos los procesos realizados en la herramienta y el usuario podrá recuperar los escenarios de calidad con sus correspondientes atributos y mejoras realizadas.

Figura 120. Guardar Cambios en el Sistema

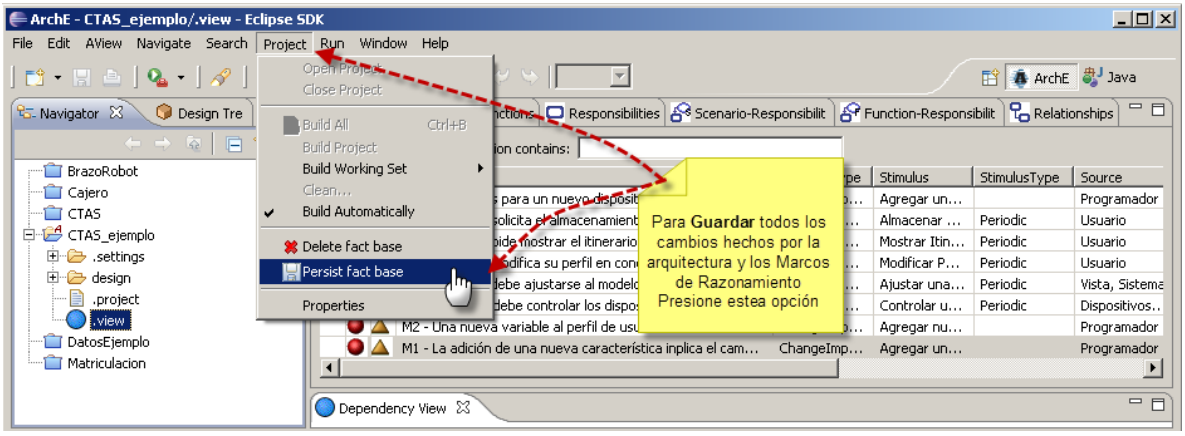
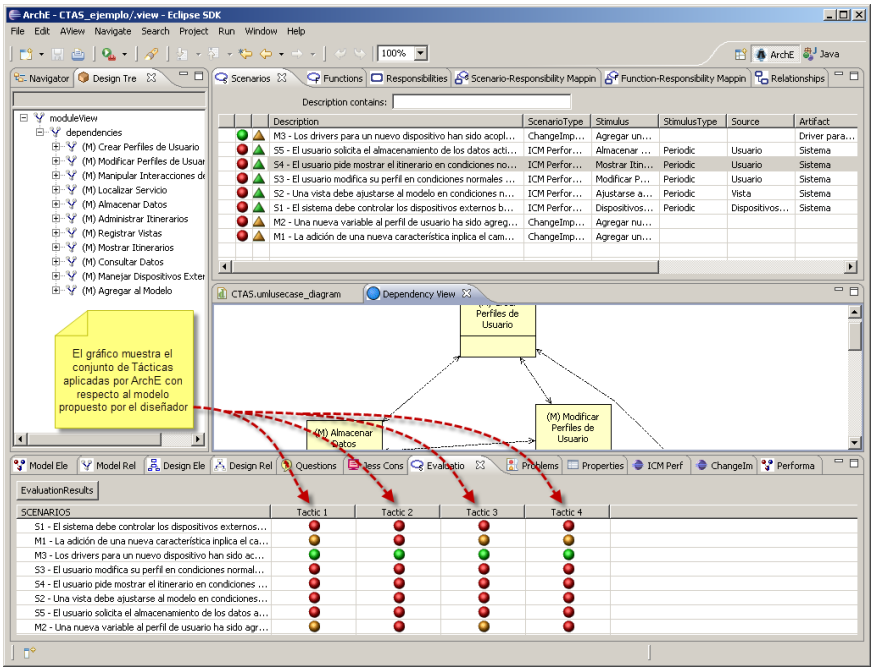
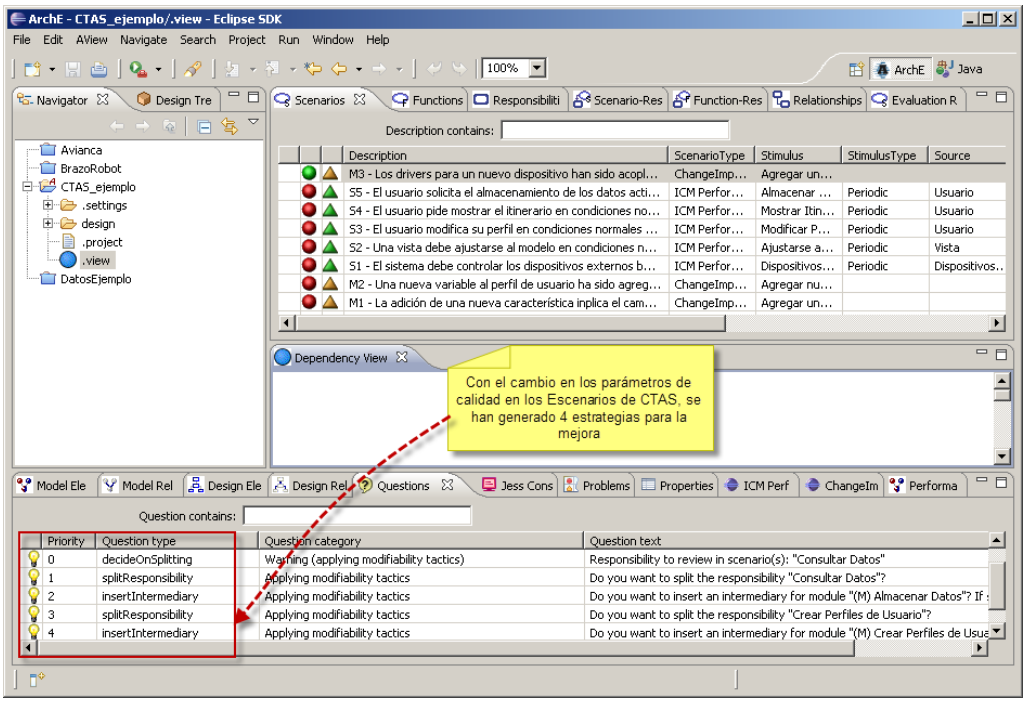


Figura 121. Aplicando un conjunto de Tácticas en CTAS



Cada vez que se realiza un cambio "aplicación de una determinada Táctica", ArchE va proponiendo mejoras al modelo. Para el caso práctico, se realizaron pruebas con más de 12 iteraciones o modificaciones, lo que dio lugar a un conjunto de división de responsabilidades, incremento en el conjunto de tácticas y desde luego se complicó bastante el diseño, lo que demuestra inicialmente una mala proposición de escenarios y distribución de responsabilidades por parte del arquitecto.

Figura 122. Tácticas aplicadas al modelo CTAS



La Tabla 49, muestra el registro de algunas Tácticas aplicadas al caso de estudio CTAS, con el análisis de costos y resultados obtenidos tras cada iteración o aplicación de Tácticas.

Tabla 49. Registro de Tácticas Aplicadas en CTAS

Estado Inicial												
Sat	**Ipc**	**Esc**	**Tipo**	**T1**	**T2**	**|T3**	**T4**	**T5**	**T6**	**T7**	**Táctica Aplicada**	**Observación**
		S4	Per								Estado Inicial	Sugiere aplicar Tácticas
		S3	Per								Estado Inicial	Sugiere aplicar Tácticas
		S2	Per								Estado Inicial	Sugiere aplicar Tácticas
		S1	Per								Estado Inicial	Sugiere aplicar Tácticas
		M1	Mod								Estado Inicial	Sugiere aplicar Tácticas
		M2	Mod								Estado Inicial	Sugiere aplicar Tácticas
		M3	Mod								Estado Inicial	Escenario Satisfecho con un 8.93 personas/día
La división “Split” de responsabilidades en el escenario "Crear Perfiles de Usuario" es recomendable ya que ésta posee fuertes dependencias con otros módulos o componente del modelo. La responsabilidad “Consultar Datos” también posee fuertes dependencias con otras responsabilidades y por ello ArchE sugiere dividir dicha Responsabilidad en dos hijos (nuevas responsabilidades heredadas). El costo estimado por ArchE para esta Táctica minimizará la dependencia entre responsabilidades padre-hijo y creará una nueva clase para reducir la dependencia entre módulos. El cambio en las dependencias del módulo “Crear Perfiles de Usuario” dará como resultado un impacto positivo en uno de los escenarios, si se inserta un módulo intermediario para reducir los costos. Un módulo intermedio “intermediario” con un costo de 30 personas/día reducirá el costo del escenario de "17.349999999999998" personas/día a "15.022817307692307" personas/día. Las dependencias de la Responsabilidad “Almacenar Datos” impacta a uno de los escenarios. Este cambio positivo puede ser benéfico si se inserta un intermediario para reducir el costo. Un intermediario con costo de 30 personas/día, reducirá el costo del cambio en el escenario de "21.179999999999996" personas/día a "13.480192307692306" personas/día. Al dividir la Responsabilidad “Consultar Datos” en 2 partes, no siempre asegura la disminución en los costos totales de los escenarios de calidad. El Arquitecto debe decidir posteriormente si aplica o no las Responsabilidades (hijas) generadas al modelo. La Aplicación de la Táctica genera una nueva (5 en total para la siguiente iteración).												
Sat	**Ipc**	**Esc**	**Tipo**	**T1**	**T2**	**|T3**	**T4**	**T5**	**T6**	**T7**	**Táctica Aplicada**	**Observación**
		S4	Per								SplitResposabilities	Hubo un cambio que impactó de manera positiva al escenario pero aún no satisface
		S3	Per								SplitResposabilities	Hubo un cambio que impactó de manera positiva al escenario pero aún no satisface
		S2	Per								SplitResposabilities	Hubo un cambio que impactó de manera positiva al escenario pero aún no satisface
		S1	Per								SplitResposabilities	Hubo un cambio que impactó de manera positiva al escenario pero aún no satisface
		M1	Mod								SplitResposabilities	Hubo un cambio que impactó de manera positiva al escenario pero aún no satisface
		M2	Mod								SplitResposabilities	Hubo un cambio que impactó de manera positiva al escenario pero aún no satisface
		M3	Mod								SplitResposabilities	Escenario Satisfecho con un 8.45 personas/día
Después de aplicar la táctica de división de responsabilidades o la táctica de abstracción, algunas funcionalidades de la responsabilidad “Crear Perfiles de Usuario” fueron refinadas en las responsabilidades hijo "Crear Perfiles de Usuario_child_9" y "Crear Perfiles de Usuario_child_10". Este cambio debe tener cierta influencia sobre el costo computacional del escenario y en algunos casos éste puede reducirse. Al dividir la Responsabilidad "Consultar Datos" y “Modificar Perfiles de Usuario” en 2 partes quizás no se afecte el costo del escenario. Posterior a ello el arquitecto debe decidir cuál de las responsabilidades hijo afectarán de forma positiva al escenario y cuáles deberán ser rechazadas.												
Sat	**Ipc**	**Esc**	**Tipo**	**T1**	**T2**	**|T3**	**T4**	**T5**	**T6**	**T7**	**Táctica Aplicada**	**Observación**
		S4	Per								AdjustRefinedResponsability	Hubo un cambio que impactó de manera positiva al escenario pero aún no satisface
		S3	Per								AdjustRefinedResponsability	Hubo un cambio que impactó de manera positiva al escenario pero aún no satisface
		S2	Per								AdjustRefinedResponsability	Hubo un cambio que impactó de manera positiva al escenario pero aún no satisface
		S1	Per								AdjustRefinedResponsability	Hubo un cambio que impactó de manera positiva al escenario pero aún no satisface
		M1	Mod								AdjustRefinedResponsability	El cambio no surtió ningún efecto y aún no se satisface el escenario
		M2	Mod								AdjustRefinedResponsability	El cambio no surtió ningún efecto y aún no se satisface el escenario
		M3	Mod								AdjustRefinedResponsability	El cambio no surtió ningún efecto pero aún se satisface el escenario con 8.45 personas/día
La responsabilidad “Consultar Datos” también posee fuertes dependencias con otras responsabilidades y por ello ArchE sugiere dividir dicha Responsabilidad en dos hijos (nuevas responsabilidades heredadas). El costo estimado por ArchE para esta Táctica minimizará la dependencia entre responsabilidades padre-hijo y creará una nueva clase para reducir la dependencia entre módulos. El cambio en las dependencias del módulo “Crear Perfiles de Usuario” dará como resultado un impacto positivo en uno de los escenarios, si se inserta un módulo intermediario para reducir los costos. Un módulo intermedio “intermediario” con un costo de 30 personas/día reducirá el costo del escenario de "17.349999999999998" personas/día a "15.022817307692307" personas/día. Las dependencias de la Responsabilidad “Almacenar Datos” impacta a uno de los escenarios. Este cambio positivo puede ser benéfico si se inserta un intermediario para reducir el costo. Un intermediario con costo de												

30 personas/día, reducirá el costo del cambio en el escenario de "21.179999999999996" personas/día a "13.480192307692306" personas/día. Al dividir la Responsabilidad “Consultar Datos” en 2 partes, no siempre asegura la disminución en los costos totales de los escenarios de calidad. El Arquitecto debe decidir posteriormente si aplica o no las Responsabilidades (hijas) generadas al modelo. La Aplicación de la Táctica genera una nueva (5 en total para la siguiente iteración). Si inserta un intermediario en la responsabilidad “Modificar Perfiles de Usuario” obtendrá un impacto positivo en uno de los escenarios ya que al introducir un costo de cambio de “30.0” personas/día se podría reducir el escenario de "15.677392857142854" a "10.804958333333332" personas/día. Al dividir la Responsabilidad "Modificar Perfiles de Usuario" en 2 partes quizás no se afecte el costo del escenario. Posterior a ello el arquitecto debe decidir cuál de las responsabilidades hijo afectarán de forma positiva al escenario y cuáles deberán ser rechazadas, por no tener un impacto en el escenario.

Sat	Ipc	Esc	Tipo	T1	T2	|T3	T4	T5	T6	T7	Táctica Aplicada	Observación
●	▲	S4	Per	●	●	●	●	●			SplitResponsabililities	Hubo un cambio que impactó de manera positiva al escenario pero aún no satisface
●	▲	S3	Per	●	●	●	●	●			SplitResponsabililities	Hubo un cambio que impactó de manera positiva al escenario pero aún no satisface
●	▲	S2	Per	●	●	●	●	●			SplitResponsabililities	Hubo un cambio que impactó de manera positiva al escenario pero aún no satisface
●	▲	S1	Per	●	●	●	●	●			SplitResponsabililities	Hubo un cambio que impactó de manera positiva al escenario pero aún no satisface
●	▲	M1	Mod	●	●	●	●	●			SplitResponsabililities	Hubo un cambio que impactó de manera positiva al escenario pero aún no satisface
●	▲	M2	Mod	●	●	●	●	●			SplitResponsabililities	Hubo un cambio que impactó de manera positiva al escenario pero aún no satisface
●	▲	M3	Mod	●	●	●	●	●			SplitResponsabililities	El cambio tuvo un cambio positivo, aún se satisface el escenario con 8.45 personas/día

Al insertar un intermediario en las dependencias para el módulo “Manipular Interacciones de Usuario”, se obtendrá un impacto en el escenario, lo que reducirá el costo del mismo. Un intermediario con un costo de "15.500000000000004" personas/día reducirá el costo del escenario de "14.04415" a "12.560536764705883" personas/día. Al insertar un intermediario en las dependencias para el módulo “Administrar Itinerarios”, se obtendrá un impacto en el escenario, lo que reducirá el costo del mismo. Un intermediario con un costo de "12.600000000000003" personas/día reducirá el costo del escenario de "19.3509" a "17.36642352941176" personas/día. Después de aplicar la táctica de división de responsabilidades o la táctica de abstracción, algunas funcionalidades de la responsabilidad “Crear Perfiles de Usuario” fueron refinadas en las responsabilidades hijo "Crear Perfiles de Usuario_child_9" y "Crear Perfiles de Usuario_child_10". La responsabilidad “Crear Perfiles de Usuario_child_9” posee múltiples dependencias a otras responsabilidades y se sugiere dividir sus responsabilidades. Al dividirla en 2 partes, la responsabilidad " Crear Perfiles de Usuario_child_9" se afectará en parte a uno de los escenarios. Se estima que al dividir la responsabilidad “Crear Perfiles de Usuario_child_9” en dos, se reducirán los costos de "14.04415" a "13.379374999999998" personas/día para este escenario. Posterior a ello el arquitecto debe decidir cuál de las responsabilidades hijo afectarán de forma positiva al escenario y cuáles deberán ser rechazadas. La responsabilidad “Consultar Datos” también posee fuertes dependencias con otras responsabilidades y se sugiere dividir sus responsabilidades. Al dividirla en 2 responsabilidades se minimizarán las respectivas dependencias y se reducirá el costo de "19.3509" a "17.2579" personas/día. Al dividir las responsabilidades para “Crear Perfiles de Usuario_child_9” quizás no se afecte el costo del escenario. Posterior a ello el arquitecto debe decidir cuál de las responsabilidades hijo afectarán de forma positiva al escenario y cuáles deberán ser rechazadas, por no tener un impacto en el escenario.

Sat	Ipc	Esc	Tipo	T1	T2	|T3	T4	T5	T6	T7	Táctica Aplicada	Observación
●	▲	S4	Per	●	●	●	●	●			InsertIntermediary	El escenario no cambia, se satisface pero disminuye el costo del mismo
●	▲	S3	Per	●	●	●	●	●			InsertIntermediary	El escenario no cambia, se satisface pero disminuye el costo del mismo
●	▲	S2	Per	●	●	●	●	●			InsertIntermediary	El escenario no cambia, se satisface pero disminuye el costo del mismo
●	▲	S1	Per	●	●	●	●	●			InsertIntermediary	Se mejora el atributo, pero el escenario aún no se satisface con las Tácticas
●	▲	M1	Mod	●	●	●	●	●			InsertIntermediary	Se mejora el atributo, pero el escenario aún no se satisface con las Tácticas
●	▲	M2	Mod	●	●	●	●	●			InsertIntermediary	El cambio no surtió ningún efecto y aún no se satisface el escenario
●	▲	M3	Mod	●	●	●	●	●			InsertIntermediary	El cambio tuvo un cambio positivo, aún se satisface el escenario con 8.45 personas/día

La responsabilidad “Consultar Datos” también posee fuertes dependencias con otras responsabilidades. Al dividir las responsabilidades para “Consultar Datos” en 2 partes, se minimizarán los costos y las dependencias entre responsabilidades. Se estima que la aplicación de la táctica reducirá el costo de "19.086194117647057" a "17.02105789473684" personas/día. Este cambio también afecta a las dependencias en la responsabilidad “Almacenar Datos” y se mejorará su costo su se inserta un intermediario. La introducción de un intermediario con un costo de "15.500000000000004" personas/día reducirá el costo de cambio en el escenario de "14.063495098039215" a "12.532336111111112" personas/día. Este cambio también afecta a las dependencias en la responsabilidad “Almacenar Administrar Itinerarios” y se mejorará su costo su se inserta un intermediario. La introducción de un intermediario con un costo de "12.600000000000003" personas/día reducirá el costo de cambio en el escenario de "19.086194117647057" a "17.152044444444446" personas/día. Después de dividir las responsabilidades o realizar procesos de encapsulación, algunas funcionalidades deberán ser redefinidas entre la responsabilidad hijo “Crear Perfiles de Usuario_09” y la responsabilidad padre “Crear Perfiles de Usuario”; “Crear Perfiles de Usuario_10” y la responsabilidad padre “Crear Perfiles de Usuario”. Esto tendrá un costo positivo en el mecanismo de cómputo del escenario. Posterior a ello el arquitecto debe decidir cuál de las responsabilidades hijo afectarán de forma positiva al escenario y cuáles deberán ser rechazadas. La responsabilidad “Consultar Datos” también posee fuertes dependencias con otras responsabilidades y se sugiere dividir sus responsabilidades.

Sat	Ipc	Esc	Tipo	T1	T2	|T3	T4	T5	T6	T7	Táctica Aplicada	Observación
●	▲	S4	Per	●	●	●	●	●			SplitResponsabililities	Se mejora el atributo, pero el escenario aún no se satisface con las Tácticas
●	▲	S3	Per	●	●	●	●	●			SplitResponsabililities	Se mejora el atributo, pero el escenario aún no se satisface con las Tácticas

		S2	Per								SplitResponsabililities	Se mejora el atributo, pero el escenario aún no se satisface con las Tácticas
		S1	Per								SplitResponsabililities	Se mejora el atributo, pero el escenario aún no se satisface con las Tácticas
		M1	Mod								SplitResponsabililities	Se mejora el atributo, pero el escenario aún no se satisface con las Tácticas
		M2	Mod								SplitResponsabililities	Se mejora el atributo, pero el escenario aún no se satisface con las Tácticas
		M3	Mod								SplitResponsabililities	Se mejora el atributo, se satisface y se reduce de 8.45 personas/día a 7.65 personas/día

La responsabilidad "Crear Perfiles de Usuario_child_9" tiene múltiples dependencias con otras responsabilidades, y se sugiere dividir dicha responsabilidad. Se estima que una división de la responsabilidad "Crear Perfiles de Usuario_child_9" reducirá el costo de "13.63934649122807" a "13.029184523809521" personas/día para uno de los escenarios. La responsabilidad "Almacenar Datos" tienen múltiples dependencias con otras responsabilidades y debe ser dividida. Para minimizar las dependencias. Se estima que esta división reducirá el costo de "17.015807894736838" a "16.269239285714285" personas/día en uno de los escenarios. Estos cambios impactarán las dependencias en el modulo "(M) Almacenar Datos", el cual se beneficiará al insertar un intermediario. Un intermediario con un costo de "30.0" personas/día reducirá el costo del escenario de "17.015807894736838" a "10.497499999999999" personas/día. Después de dividir las responsabilidades o realizar procesos de encapsulación, algunas funcionalidades deberán ser redefinidas entre la responsabilidad hijo "Crear Perfiles de Usuario_09" y la responsabilidad padre "Crear Perfiles de Usuario"; "Crear Perfiles de Usuario_10" y la responsabilidad padre "Crear Perfiles de Usuario". Esto tendrá un costo positivo en el mecanismo de cómputo del escenario. Posterior a ello el arquitecto debe decidir cuál de las responsabilidades hijo afectarán de forma positiva al escenario y cuáles deberán ser rechazadas. La responsabilidad "Consultar Datos" también posee fuertes dependencias con otras responsabilidades y se sugiere dividir sus responsabilidades.

Sat	Ipc	Esc	Tipo	T1	T2	|T3	T4	T5	T6	T7	Táctica Aplicada	Observación
		S4	Per								SplitResponsabililities	
		S3	Per								SplitResponsabililities	
		S2	Per								SplitResponsabililities	
		S1	Per								SplitResponsabililities	
		M1	Mod								SplitResponsabililities	El cambio no surtió ningún efecto y aún no se satisface el escenario
		M2	Mod								SplitResponsabililities	
		M3	Mod								SplitResponsabililities	El escenario se satisface con 7.1442 personas/día

Se advierte al arquitecto que al dividir la responsabilidad "Crear Perfiles de Usuario_child_10" en 2 partes no se "optimizará" el costo de los escenarios. Posterior a ello el arquitecto debe decidir cuál de las responsabilidades hijo afectarán de forma positiva al escenario y cuáles deberán ser rechazadas. La responsabilidad "Consultar Datos" también posee fuertes dependencias con otras responsabilidades y se sugiere dividir sus responsabilidades.

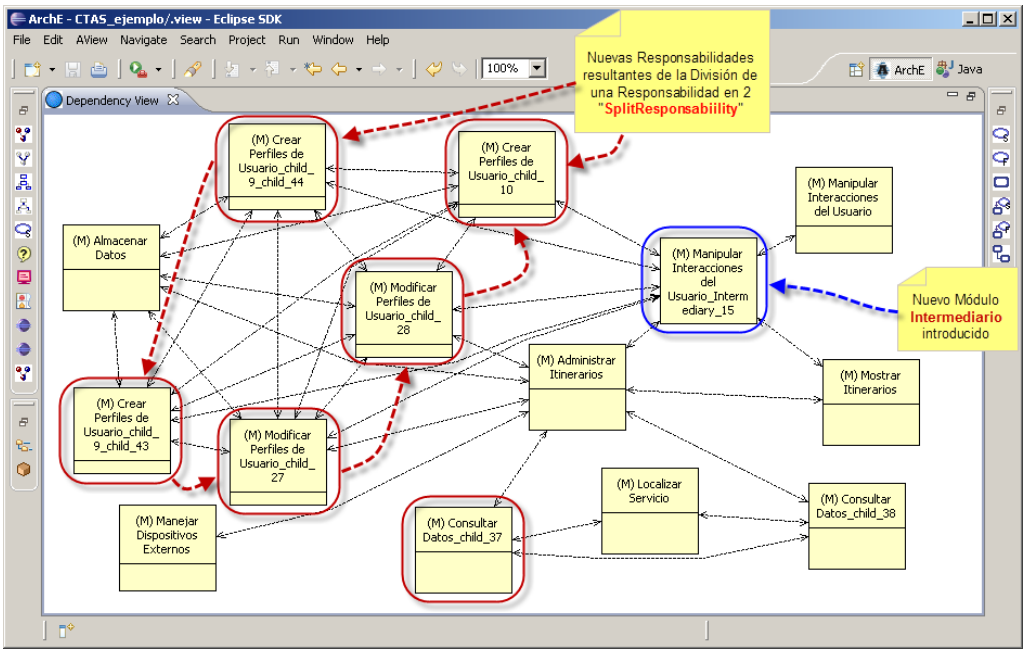
Sat	Ipc	Esc	Tipo	T1	T2	|T3	T4	T5	T6	T7	Táctica Aplicada	Observación
		S4	Per								decideOnSplitting	Se mejora el atributo, pero el escenario aún no se satisface con las Tácticas
		S3	Per								decideOnSplitting	Se mejora el atributo, pero el escenario aún no se satisface con las Tácticas
		S2	Per								decideOnSplitting	Se mejora el atributo, pero el escenario aún no se satisface con las Tácticas
		S1	Per								decideOnSplitting	Se mejora el atributo, pero el escenario aún no se satisface con las Tácticas
		M1	Mod								decideOnSplitting	El cambio no surtió ningún efecto y aún no se satisface el escenario
		M2	Mod								decideOnSplitting	El cambio no surtió ningún efecto y aún no se satisface el escenario
		M3	Mod								decideOnSplitting	El cambio no surte efecto pero el escenario aún se satisface con 7.1442 personas/día

Se cambian los tiempos de ejecución de las responsabilidades Crear Perfiles de Usuario_Child_10, Crear Perfiles de Usuario_Child_9, Modificar Perfiles de Usuario_Child_27, Modificar Perfiles de Usuario_Child_28, entre otros y se obtuvo el siguiente cambio.

Sat	Ipc	Esc	Tipo	T1	T2	|T3	T4	T5	T6	T7	Táctica Aplicada	Observación
		S4	Per								ExecutionTime	Se mejora el atributo, pero el escenario aún no se satisface con las Tácticas
		S3	Per								ExecutionTime	Se mejora el atributo, pero el escenario aún no se satisface con las Tácticas
		S2	Per								ExecutionTime	Se mejora el atributo, pero el escenario aún no se satisface con las Tácticas
		S1	Per								ExecutionTime	Se mejora el atributo, pero el escenario aún no se satisface con las Tácticas
		M1	Mod								ExecutionTime	El cambio no surtió ningún efecto y aún no se satisface el escenario
		M2	Mod								ExecutionTime	El cambio no surtió ningún efecto y aún no se satisface el escenario
		M3	Mod								ExecutionTime	El cambio no surte efecto pero el escenario aún se satisface con 7.14421 personas/día

Convenciones: **Sat**: Grado de Satisfacción del escenario. **Ipc**: Impacto **Esc**: Escenario. **Tipo**: Modifiability o Performance. **T**: Táctica

Figura 123. Diseño <<n+1>> propuesto por ArchE tras aplicación de Tácticas



En una primera impresión, considero que ArchE es una herramienta que permite al arquitecto gestionar un modelo a partir de un conjunto de Tácticas para mejorar los escenarios de calidad. Este análisis parte de la implementación de dos Marcos de Razonamiento para igual número de Tácticas "Modifiability y Performance"; (Algo que resulta como una potencial debilidad, ya que existen diversos atributos que en concurso con la Modificabilidad y el Rendimiento podrían orientar un mejor diseño en una arquitectura de software), en este caso para el sistema CTAS.

Es interesante como ArchE prioriza cada Táctica conforme a cierta probabilidad de éxito o cambio en los escenarios. En el caso concreto de este primer ejemplo, se escogieron en orden estricto a partir de las prioridades mostradas "0, 1, 2, 3…, n+1". Tras cada iteración o modelo sugerido, se observó como ArchE implementaba técnicas como inserción de intermediarios o división de responsabilidades (Fig. 123) entre otras (para tratar de aumentar la cohesión "funcionalidades bien establecidas dentro del modelo" y disminuir el acoplamiento entre elementos del modelo, para evitar la propagación de cambios a nivel de todo el sistema o componente).

Al realizar cada iteración y al analizar en detalle cada modificación, costo, y Táctica asociada, se observó como el modelo creció "generando demasiadas dependencias padre-hijo" entre nuevas responsabilidades derivadas de una modificabilidad. Tras cada aplicación de Tácticas en el caso práctico CTAS, algunos escenarios mejoraron su capacidad, pero otros por no decir que la mayoría fueron afectados de forma negativa una tras otra iteración, lo que empeoró notablemente el modelo de diseño propuesto inicialmente (si habláramos de componentes, el modelo creció en una proporción de 1 a 3,

es decir; se pasó de 9 componentes a 27, lo que indiscutiblemente describe un pésimo modelado a nivel de requerimientos por parte del arquitecto.

Si bien es cierto que ArchE proporciona una idea de cambio o mejora, derivada del proceso de análisis de los Frameworks de Razonamiento, no siempre estos cambios afectan de manera significativa al escenario afectado. Curiosamente se observó que estos cambios también modificaron los pesos o costos de los demás, incluyendo a los recién descompuestos "módulos o responsabilidades", recayendo estos nuevos conjuntos en fallos de diseño lo que dio origen a más tácticas (al final se procesaron 8 Tácticas), y se incrementaron las dependencias entre funcionalidades.

A modo general se puede concluir que tras la aplicación de la Arquitectura al modelo planteado en el caso de estudio ArchE exploró un conjunto de alternativas de mejora para el modelo propuesto pero no se encontró una solución factible, ya que tras cada proceso iterativo empeoraron las condiciones de uno o más escenarios (solo se logró satisfacer 3 escenarios de los 8 propuestos). Considero que la capacidad de la herramienta podría incrementarse al extender sus Frameworks de Razonamiento, ya que en definitiva un atributo de calidad no puede estructurarse como una unidad sino como la unión sinérgica de uno o más atributos para la mejora del modelo.

Algo que destacó en esta primera experiencia es que ArchE me permitió evaluar en forma global el diseño propuesto y me ayudó a entender cómo incide la valoración y peso de un elemento (artefacto, módulo, clase, etc.) en un diseño arquitectónico, dependiendo de las cualidades o atributos bajo el cual se ha diseñado el sistema. Aunque ArchE incluye una interfaz comprensible que permite determinar cambios y asociar mejoras, la herramienta no permite la restauración a un nivel determinado de arquitectura sugerida, ya que tras cada iteración, el sistema modifica todas sus dependencias y en la interfaz gráfica es complejo determinar cuáles fueron esos cambios y cómo pueden compararse con otros referentes (históricos de iteración).

A continuación se explorarán otros ejemplos donde se aplicará todo el proceso y con suerte encontraremos un modelo ideal de diseño a partir de ArchE.

10

CASOS PROPUESTOS Y SU IMPLEMENTACIÓN EN ArchE

"El mayor enemigo del conocimiento no es la ignorancia, sino la ilusión del conocimiento"
-- Stephen Hawking

10.1. SISTEMA DE MATRICULACIÓN

El sistema de Matriculas de la universidad se encuentra bajo una Arquitectura Web en 3 capas. El sistema se encuentra optimizado para trabajar como servicio Web en ambiente cliente/servidor, mediante protocolo https. Dentro de los nodos que componen al sistema, todos los procesos, servicios y/o componentes son agrupados en la ERP PeopelSoft.

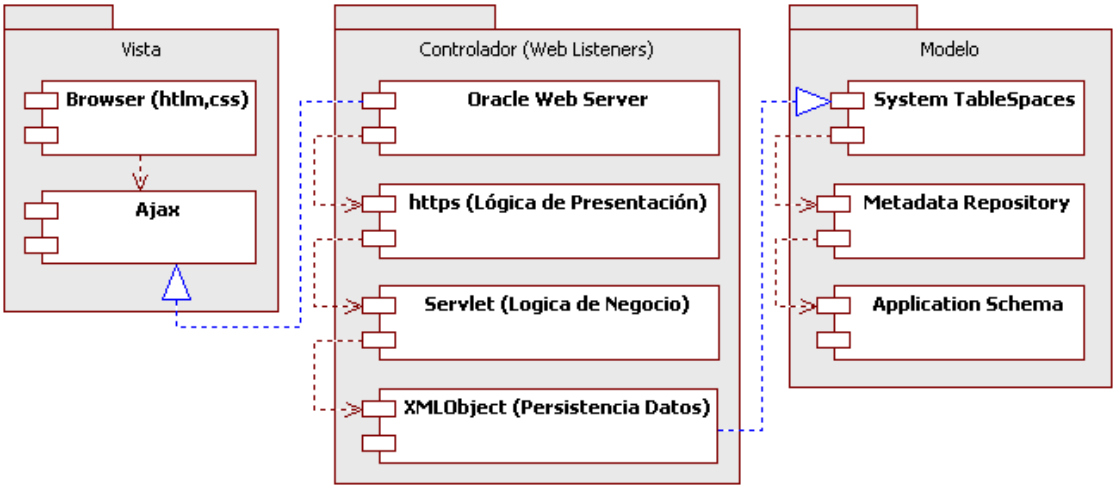
La ERP es un sistema en constante evolución y el manejo de bases de datos se sostiene bajo la lógica de procesos almacenados y Triggers que pueden ser parametrizados de acuerdo a las necesidades del negocio. El sistema maneja un conjunto de funcionalidades y clases las cuales invocan mediante servicios a los procesos almacenados desde las diferentes interfaces del usuario.

Para el caso de estudio, se sabe que la arquitectura de la ERP, y el respectivo módulo de Matrículas, están guiados por los siguientes atributos de calidad: Performance, Disponibilidad, Modificabilidad y Seguridad.

10.1.1. Definición de la Arquitectura para el Módulo de Matrículas

La arquitectura del sistema está diseñada bajo el esquema del patrón MVC. Para el caso concreto solo tomaremos como referencia al proceso de matrículas.

Figura 124. Arquitectura MVC para el sistema Matrículas



10.1.2. Planificación de Requerimientos

El primer paso consiste en obtener el conjunto de requerimientos para el sistema sujeto de evaluación. Hacen parte de los requerimientos, los atributos de calidad de cada uno de los escenarios así mismo como el conjunto de requerimientos funcionales, las relaciones, las responsabilidades y el mapeo entre dichos atributos.

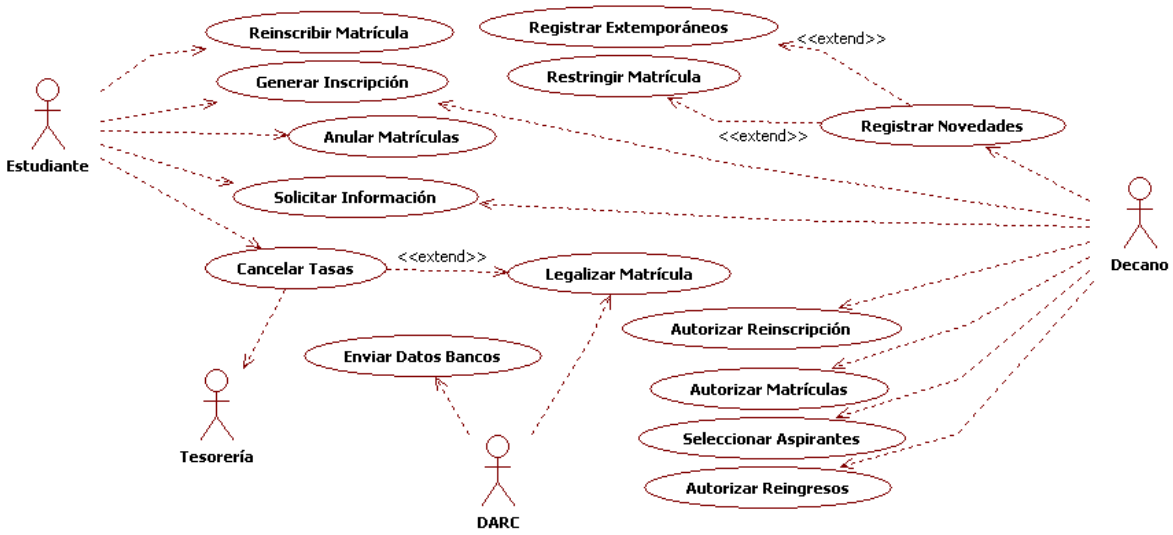
10.1.3. Funcionalidades del Sistema

Tabla 50. Descripción de Funcionalidades en el Módulo de Matrículas

NO.	FUNCIONALIDAD	DETALLE / UTILIDAD
1	Realizar Inscripción	Permite realizar una inscripción en el periodo académico activo. Requiere de la adición de materias
2	Consultar Registro Estudiante	Despliega la información básica del estudiante (código, nombre, programa, semestre, etc.)
3	Consultar Histórico de Matriculas	Corresponde al registro detallado de semestres cursados por un determinado estudiante
4	Consultar Solicitudes de Reintegro	Registra a aquellos estudiantes que han solicitado reintegro a la universidad
5	Verificar Restricciones a Matrícula	Identifica incompatibilidades o restricciones que impiden que el estudiante se matricule normalmente
6	Validar Pre-requisitos	Garantiza que los pre-requisitos de cada materia se cumplan conforme al pensum y al reglamento
7	Consultar Parámetros de Aula	Su función consiste en desplegar datos como tipo de aula, ubicación, cupo, disponibilidad, asignación, etc.
8	Guardar Registro de Matrícula	Almacena el registro de matrícula del estudiante para futuras consultas, modificaciones y control de pensum
9	Seleccionar Asignaturas	Permite seleccionar un conjunto de materias con base en pre-requisitos y número permitido de créditos
9.1	Seleccionar Parámetros de Asignatura	Selecciona un aula en específico donde ha sido asignada una determinada materia o asignatura
9.2	Seleccionar Horarios	Permite seleccionar el respectivo horario de la materia, puede ser en cualquiera de las dos jornadas existentes
9.3	Procesar Listado de Asignaturas	Una vez validado los pre-requisitos, este permite almacenar la totalidad de materias matriculadas
10	Validar Horario	El sistema debe realizar de forma automática la validación del horario para evitar cruces con otras asignaturas y aulas
10.1	Generar Horario	Genera el horario de clase para el estudiante (asignaturas, grupo, horario, aulas, docente asignado)
11	Generar Matrícula	Esta funcionalidad se asemeja al proceso de legalización de matrícula académica
11.1	Generar Recibo de Pago	La matrícula financiera se realiza a través del pago de tasas en el banco, solo hasta aquí se considera matriculado

10.1.4. Descripción de Casos de Uso (Modelo de Negocio)

Figura 125. Caso de Uso Matrículas



10.1.5. Lista de Involucrados "Stakeholders"

Tabla 51. Lista de Actores

ACTOR	CASOS DE USO EN LOS QUE INTERVIENEN
Estudiante	• Reinscribir Matrícula • Generar Inscripción • Anular Matrícula • Solicitar Información • Cancelar Tasas
Tesorería	• Legalizar Matrículas • Cancelar Tasas
DARC	• Enviar Datos de Pago a Bancos • Legalizar Matrícula
Decano	• Registrar Novedades (Extemporáneas y Restricciones a Matrículas) • Autorizar Reingresos • Seleccionar Aspirantes • Autorizar Matrículas • Autorizar Reinscripciones

10.1.6. Documentación de Casos de Uso para el sistema de Matrículas

Tabla 52. Funcionalidad Solicitar Información

CASO DE USO	***SOLICITAR INFORMACIÓN***
Actor (es)	Estudiante, Decano
Pre-Condiciones	La información del estudiante o aspirante debe estar registrada con anterioridad
Post-Condiciones	Disponibilidad de consulta de datos en cualquier momento
DESCRIPCIÓN Y FLUJOS DE INFORMACIÓN	

- Paso 1: El sistema inicia cuando se solicitan los datos personales del estudiante. A partir de la información contenida en la base de datos, el estudiante o decano de facultad puede actualizar la información del estudiante. las vistas de actualización, parametrización y modificación serán verificados por el sistema y se acudirá a los filtros de seguridad para garantizar que los cambios sean confiables y afecten exclusivamente al conjunto de registros pertenecientes al estudiante. En condiciones normales el sistema debe guardar la información.

- Paso 2: Se solicitan los respectivos datos académicos (programa, pensum académico, detalles de matrícula, pagos, novedades, etc.). En condiciones normales el sistema debe guardar la información.

- Paso 3: El sistema debe proporcionar el currículo o registro académico del estudiante. El decano se encarga de actualizar y validar el respectivo currículo con base en reglas y pre-requisitos. En condiciones normales el sistema debe guardar la información.

- Paso 4: El sistema despliega la lista de créditos por cada asignatura, y el número máximo de créditos. El sistema trae consigo datos de matriculaciones anteriores y record de materias cursadas. El sistema no podrá realizar modificaciones sobre anteriores periodos académicos a excepción del actual en donde si podrá realizar modificaciones antes de realizar la formalización total de la matrícula y procesos de adiciones o cancelaciones. A partir de este conjunto se despliega la respectiva malla curricular.

- Paso 5: Despliegue de la malla curricular.

- Paso 6: Se completan otros requerimientos en la vista de consulta y se procede a generar el respectivo informe en la cola de impresiones del servidor de impresión de PeopelSoft.

Tabla 53. Funcionalidad Generar Inscripción

CASO DE USO	***GENERAR INSCRIPCIÓN***
Actor (es)	Estudiante, Decano
Pre-Condiciones	La información del estudiante o aspirante debe estar registrada con anterioridad. El estudiante debe aparecer con registro de matrículas anteriores o haber solicitado procesos de reingreso. Tampoco debe encontrarse con pendientes (sanciones, multas, materias perdidas u otros impedimentos "pérdida simultánea por más de 3 periodos de una materia" o "pre-requisitos no cumplidos"). Antes de iniciar el caso de uso también se requiere de datos como cupos, horarios y tutores.
Post-Condiciones	Una vez validad y almacenada la información, ésta no podrá ser modificada.
DESCRIPCIÓN Y FLUJOS DE INFORMACIÓN	

- Paso 1: El sistema muestra al estudiante su record de matrículas. El sistema prioriza inicialmente las materias que éste podrá tomar, así como el número de créditos asociados a ésta. A partir de aquí se despliega un conjunto de opciones que incluyen el listado general de asignaturas, los horarios y posibles horarios a tomar dentro y fuera del programa (movilidad con otros programas académicos).

- Paso 2: Una vez elegida cada materia, el sistema va mostrando una a una con sus atributos (cupo, inscritos, horario, grupo, etc.). Al seleccionar la materia, el sistema valida que ésta cumpla con los pre-requisitos y gestiona el posible cruce de horarios.

- Paso 3: El sistema valida que se tomen los respectivos créditos académicos permitidos para el semestre y periodo lectivo. El comportamiento de este módulo se debe ajustar contantemente en función de los cambios realizados al reglamento académico-estudiantil.

- Paso 4: El estudiante finaliza el proceso de inscripción. Para ello el sistema despliega ayudas visuales que le indican que la transacción ha sido almacenada y que éstos datos solo podrán ser modificados por el decano.

- Paso 5: Una vez generada la inscripción con el conjunto de materias inscritas, el sistema procesa un reporte al cual el sistema asigna un valor de pago sugerido con base en el número de créditos matriculados (Pago de Tasas). Este estará sujeto a verificación en el departamento financiero. (éste no es modelado en el caso presentado).

- Paso 6: Una vez generado el Boucher de pago, el sistema almacena dicho recurso y el estudiante podrá acercarse a los puntos de pago autorizados por la universidad y las fechas límite para pago de las tasas con o sin recargo.

Tabla 54. Funcionalidad Reinscribir Matrícula

CASO DE USO	***REINSCRIBIR MATRICULA***
Actor (es)	Estudiante, Decano
Pre-Condiciones	La información del estudiante o aspirante debe estar registrada con anterioridad. Este proceso está sujeto a autorización previa del decano del programa. Cualquier reinscripción se debe hacer en el periodo y debe contener información como grupos disponibles, horarios, docentes, etc.
Post-Condiciones	Una vez validad y almacenada la información, ésta no podrá ser modificada.
DESCRIPCIÓN Y FLUJOS DE INFORMACIÓN	

- Paso 1: Una reinscripción se origina cuando el estudiante solicita el cambio de materia por otra, baja de alguna materia o por cambios en grupos de estudio.

- Paso 2: El sistema muestra al estudiante su record de matrículas. El sistema prioriza inicialmente las materias que éste podrá tomar, así como el número de créditos asociados a ésta. A partir de aquí se despliega un conjunto de opciones que incluyen el listado general de asignaturas, los horarios y posibles horarios a tomar dentro y fuera del programa (movilidad con otros programas académicos).

- Paso 2: Una vez elegida cada materia, el sistema va mostrando una a una con sus atributos (cupo, inscritos, horario, grupo, etc.). Al seleccionar la materia, el sistema valida que ésta cumpla con los pre-requisitos y gestiona el posible cruce de horarios.

- Paso 3: El sistema valida que se tomen los respectivos créditos académicos permitidos para el semestre y periodo lectivo. El comportamiento de este módulo se debe ajustar contantemente en función de los cambios realizados al reglamento académico-estudiantil.

- Paso 4: El estudiante finaliza el proceso de inscripción. Para ello el sistema despliega ayudas visuales que le indican que la transacción ha sido almacenada y que éstos datos solo podrán ser modificados por el decano.

- Paso 5: Una vez generada la inscripción con el conjunto de materias inscritas, el sistema procesa un reporte al cual el sistema asigna un valor de pago sugerido con base en el número de créditos matriculados (Pago de Tasas). Este estará sujeto a verificación en el departamento financiero. (éste no es modelado en el caso presentado).

- Paso 6: Una vez generado el Boucher de pago, el sistema almacena dicho recurso y el estudiante podrá acercarse a los puntos de pago autorizados por la universidad y las fechas límite para pago de las tasas con o sin recargo y con los detalles de la modificación o reinscripción.

Tabla 55. Funcionalidad Anular Matrícula

CASO DE USO	***ANULAR MATRÍCULA***
Actor (es)	Estudiante
Pre-Condiciones	El estudiante debe estar cursando un semestre en el periodo lectivo. La cancelación de la matrícula está sujeta a los horarios dispuestos por la universidad para realizar dicho proceso.
Post-Condiciones	La información debe estar disponible para futuras consultas y estadísticas del programa. Una vez realizado el proceso, este no podrá reversarse.
DESCRIPCIÓN Y FLUJOS DE INFORMACIÓN	

- Paso 1: El sistema muestra al estudiante el detalle de su matrícula.

- Paso 2: El estudiante debe argumentar mediante el formulario y el anexo de evidencia escrita, las razones que motivan la cancelación de su matrícula. Esta información es guardada automáticamente al terminar la edición del campo novedades de cancelación.

- Paso 3: El sistema procesa la petición e informa al usuario del procedimiento donde se le advierte la operación que va a realizar.

- Paso 4: El usuario acepta el contrato y el sistema procede a anular el registro. Si el usuario decide no realizar la anulación de matrícula, el sistema debe salir de la vista de anulación.

- Paso 5: Si el estudiante ha realizado con anterioridad pago alguno por concepto de matrícula, el sistema deberá proporcionar un saldo a favor el cual deberá ser cargado a la hoja de vida del estudiante para posteriores matrículas.

- Paso 6: El sistema almacena la cancelación de matrícula, libera el cupo del curso y genera la respectiva novedad.

Tabla 56. Funcionalidad Enviar Datos al Banco

CASO DE USO	*ENVIAR DATOS AL BANCO*
Actor (es)	DARC – Departamento de Admisiones, Registro y Control Académico
Pre-Condiciones	Tener un listado completo de los inscritos
Post-Condiciones	Una vez se genera el archivo plano, éste no podrá ser modificado.
DESCRIPCIÓN Y FLUJOS DE INFORMACIÓN	
• Paso 1: El DARC descarga la información de la base de datos de inscritos del programa (código, nombre, créditos y valor a pagar). • Paso 2: El departamento realiza la respectiva generación del archivo en formato CSV para enviar al banco para el respectivo pago de los estudiantes.	

Tabla 57. Funcionalidad Cancelar Tasas

CASO DE USO	*CANCELAR TASAS*
Actor (es)	Estudiante, Tesorería
Pre-Condiciones	Tener un listado completo de los inscritos
Post-Condiciones	La información debe estar disponible para futuras consultas y estadísticas del programa. Una vez realizado el proceso, este no podrá reversarse.
DESCRIPCIÓN Y FLUJOS DE INFORMACIÓN	
• Paso 1: El sistema reacciona cuando el estudiante realiza el respectivo pago en el banco. Cuando el estudiante le han sellado su Boucher, éste lo lleva a tesorería y al DARC para los respectivos descargos. • Paso 2: El Banco genera el registro de pagos y éste es procesado cada dos días para contrastar los datos de pagos. • Paso 3: El tesorero verifica en el listado de matrículas pagas, realiza la respectiva búsqueda y valida el pago realizado. Una vez confrontado con el formato de pago físico, se archiva y actualiza el registro de estudiantes tanto en el DARC como en el registro financiero.	

Tabla 58. Funcionalidad Legalizar Matrícula

CASO DE USO	*LEGALIZAR MATRÍCULA*
Actor (es)	DARC
Pre-Condiciones	El Caso de Uso requiere que exista el archivo plano generado en el proceso de pago en bancos.
Post-Condiciones	El sistema realiza la legalización de la matrícula en forma automática y actualizará el estado del estudiante (En Proceso a Activo).
DESCRIPCIÓN Y FLUJOS DE INFORMACIÓN	
• Paso 1: El sistema legaliza la matrícula y pasa un informe de pagos a la dirección para efectos financieros y estadísticos. • Paso 2: El sistema actualiza el estatus del estudiante que se encuentra en Proceso de Pago a Estudiante Activo, en el semestre y periodo activo.	

Tabla 59. Funcionalidad Seleccionar Aspirantes

CASO DE USO	*SELECCIONAR ASPIRANTES*
Actor (es)	Decano
Pre-Condiciones	Tener un listado completo de aspirantes
Post-Condiciones	La información debe estar disponible para futuras consultas
DESCRIPCIÓN Y FLUJOS DE INFORMACIÓN	

- Paso 1: El sistema despliega el listado de aspirantes inscritos para el periodo lectivo.
- Paso 2: El decano debe realizar uno a uno el registro de cada aspirante y actualizará la información del aspirante para la futura matrícula.
- Paso 3: El sistema despliega un cuadro de diálogo donde se le pide confirmar y aceptar al aspirante. El sistema asignará los cupos con base en puntajes obtenidos en las pruebas de estado y una vez completado el cupo máximo requerido por grupo, el sistema debe prohibir la adición de nuevos aspirantes.
- Paso 4: Una vez finalizada la carga de aspirantes, el sistema procede a generar el listado autorizado para matrícula por primera vez.

Tabla 60. Funcionalidad Registrar Novedades

CASO DE USO	*REGISTRAR NOVEDADES*
Actor (es)	Decano
Pre-Condiciones	Listado de Estudiantes sin matricular o por fuera del periodo, listado de novedades de otros departamentos y otros aspectos que impiden la matrícula del estudiante en el sistema. La información del estudiante debe estar registrada con anterioridad.
Post-Condiciones	
DESCRIPCIÓN Y FLUJOS DE INFORMACIÓN	

- Paso 1: El sistema muestra al decano, el listado de estudiantes con novedades de matrícula. El listado puede ser procesado a través de búsquedas generales o particulares << filtros >>.
- Paso 2: El sistema despliega la consulta del estudiante en particular.
- Paso 3: El decano realiza las respectivas anotaciones, actualiza el sistema y genera el respectivo impedimento del estudiante.
- Paso 4: Si la novedad corresponde a extemporaneidades, el decano marca una casilla con el respectivo atributo y éste se actualiza en la base de datos.
- Paso 5: Si el estudiante ha incurrido en multas, sanciones y otro tipo de impedimentos, éstos deberán ser soportados con la respectiva anotación y oficina que generó el impedimento para que el estudiante sepa de donde se originó la respectiva novedad. Una vez editada la novedad, el sistema procesará el requerimiento y actualizará el estatus del estudiante de Activo a Registra Novedad.
- Paso 6: Eventualmente el sistema podrá enviar un SMS al estudiante sobre el particular. (este no se ha modelado en el presente ejemplo).

Tabla 61. Funcionalidad Autorizar Reinscripciones

CASO DE USO	*AUTORIZAR REINSCRIPCIONES*
Actor (es)	Decano
Pre-Condiciones	Listado de Matriculados y la información específica del estudiante
Post-Condiciones	La información estará disponible para futuras consultas
DESCRIPCIÓN Y FLUJOS DE INFORMACIÓN	

- Paso 1: El sistema despliega la información de estudiantes registrados en el periodo lectivo. La lista también deberá poseer mecanismos para selección o por grupo o de forma individual a través de filtros y búsquedas.
- Paso 2: Una vez encontrado el registro, el decano selecciona al estudiante y realiza el proceso de autorización para que éste proceda a realizar su reinscripción.
- Paso 3: El sistema actualiza los datos.

Tabla 62. Funcionalidad Autorizar Matrículas

CASO DE USO	*AUTORIZAR MATRÍCULAS*
Actor (es)	Decano
Pre-Condiciones	Listado de Matriculados y la información específica del estudiante
Post-Condiciones	La información estará disponible para futuras consultas
DESCRIPCIÓN Y FLUJOS DE INFORMACIÓN	

- Paso 1: El sistema despliega la información de estudiantes registrados en el periodo lectivo. La lista también deberá poseer mecanismos para selección o por grupo o de forma individual a través de filtros y búsquedas.
- Paso 2: Una vez encontrado el registro, el decano selecciona al estudiante y realiza el proceso de autorización para que éste proceda a realizar su Matrícula.
- Paso 3: El sistema actualiza los datos, para que el estudiante pueda iniciar el proceso de matrícula en el respectivo programa académico.

Tabla 63. Funcionalidad Autorizar Reingresos

CASO DE USO	*AUTORIZAR REINGRESOS*
Actor (es)	Decano
Pre-Condiciones	La información del estudiante debe estar registrada con anterioridad. Cualquier reingreso se debe hacer en el periodo y debe contener información como grupos disponibles, horarios, docentes, etc.
Post-Condiciones	La información estará disponible para futuras consultas
DESCRIPCIÓN Y FLUJOS DE INFORMACIÓN	

- Paso 1: El sistema despliega la información de estudiantes registrados en el periodo lectivo. La lista también deberá poseer mecanismos para selección o por grupo o de forma individual a través de filtros y búsquedas.

- Paso 2: Una vez encontrado el registro, el decano selecciona al estudiante y realiza el proceso de autorización para que éste proceda a realizar su reingreso en el periodo lectivo.

- Paso 3: El sistema actualiza los datos, para que el estudiante pueda iniciar el proceso de matrícula en el respectivo programa académico.

10.1.7. Árbol de Utilidad para el sistema de Matrículas

El siguiente árbol de utilidad está basado específicamente en los 2 atributos de calidad utilizados por ArchE, Modifiability y Performance.

Figura 126. Árbol de Utilidad para el atributo Performance

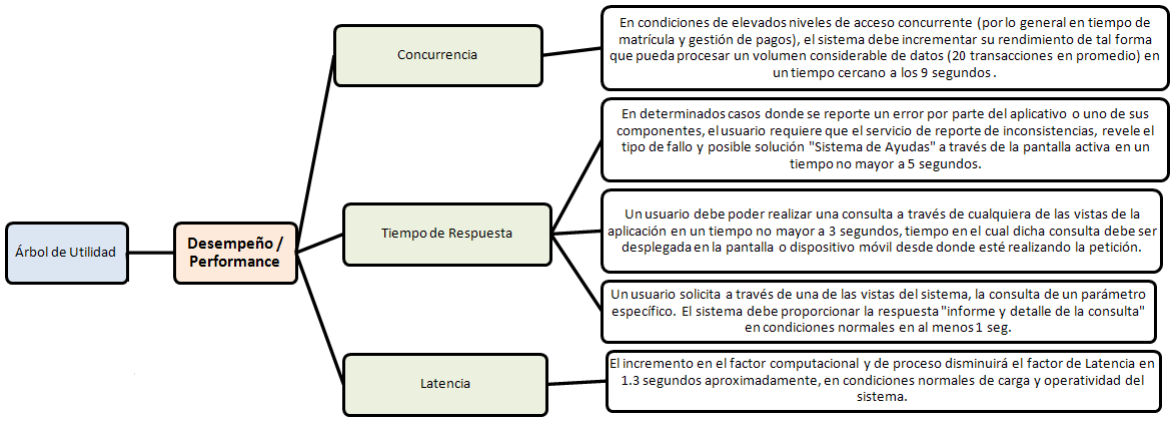
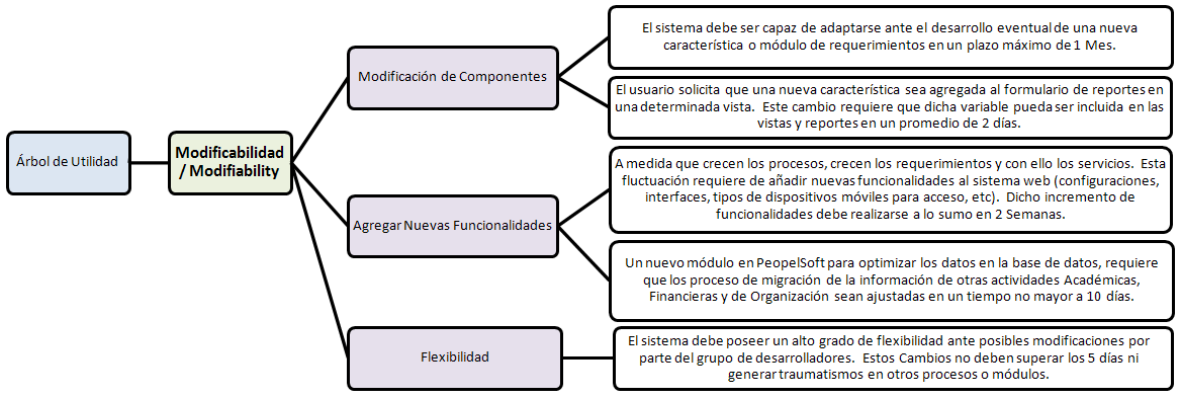


Figura 127. Árbol de Utilidad para el atributo Modifiability

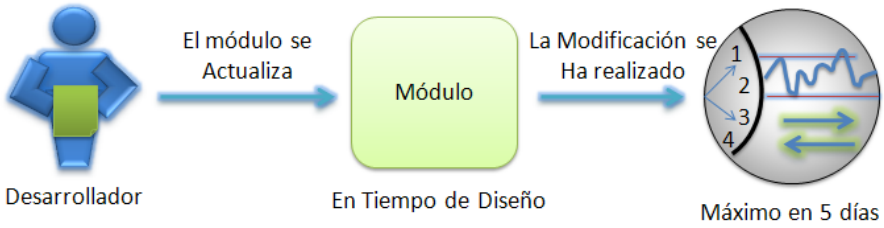


10.1.8. Refinamiento de Escenarios

Una vez el arquitecto ha modelado los distintos escenarios, es hora de convertir ese conjunto de requerimientos plasmados en papel en un modelo de análisis para ArchE. El refinamiento de los escenarios no es más que la definición de dichos escenarios en función de las 6 partes descritas por (Bass, Len; Clements, Paul; Kazman, Rick, 2003) y que hacen parte de un escenario de calidad.

ESCENARIO 1

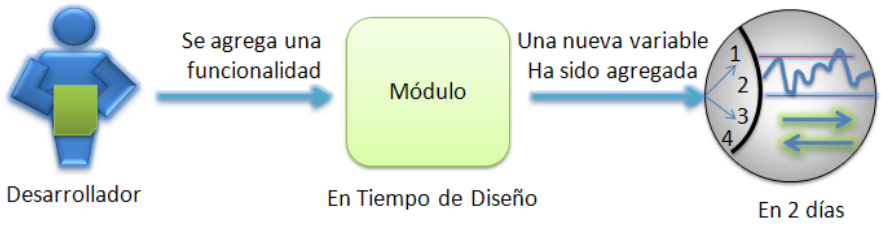
M1 - El sistema debe poseer un alto grado de flexibilidad ante posibles modificaciones por parte del grupo de desarrolladores. Estos Cambios no deben superar los 5 días ni generar traumatismos en otros procesos o módulos.



ESCENARIO	PARÁMETROS
Stimulus source	Desarrollador, Programador
Stimulus	Un Módulo requiere una Modificación específica
Artefact	Módulo
Environment	En Tiempo de Diseño
Response	La Modificación se ha realizado
Response measure	El costo de esfuerzo del cambio no debe superar los 5 días

ESCENARIO 2

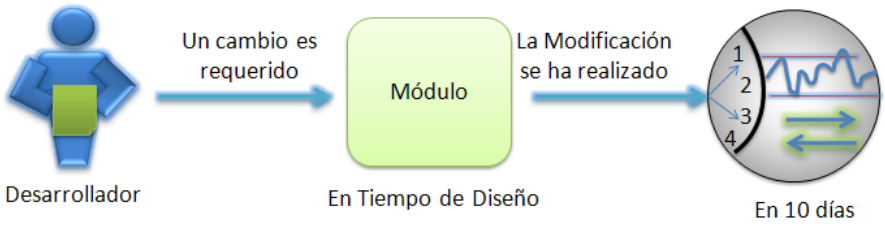
M2 - El usuario solicita que una nueva característica sea agregada al formulario de reportes en una determinada vista. Este cambio requiere que dicha variable pueda ser incluida en las vistas y reportes en un promedio de 2 días.



ESCENARIO	PARÁMETROS
Stimulus source	Desarrollador, Programador
Stimulus	Una nueva variable debe ser agregada a un Reporte
Artefact	Sistema, Módulo
Environment	En Tiempo de Diseño
Response	La Inclusión de la Variable se ha realizado sin contratiempos
Response measure	En 2 días

ESCENARIO 3

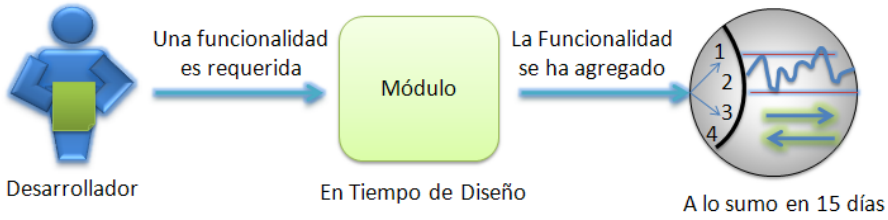
M3 - Un nuevo módulo en PeopelSoft para optimizar los datos en la base de datos, requiere que los proceso de migración de la información de otras actividades Académicas, Financieras y de Organización sean ajustadas en un tiempo no mayor a 10 días.



ESCENARIO	PARÁMETROS
Stimulus source	Desarrollador
Stimulus	Se requiere un cambio en la Migración de los Datos
Artefact	Módulo
Environment	En Tiempo de Diseño
Response	La Modificación se ha realizado
Response measure	Con un Límite de 10 días

ESCENARIO 4

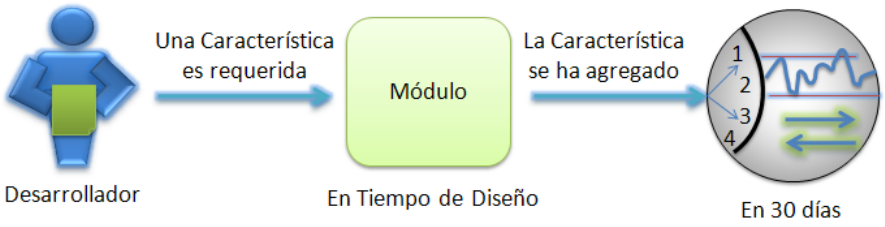
M4 - Una de las ventajas de la arquitectura de la aplicación es la provisión de servicios a través de la Web. A medida que crecen los procesos, crecen los requerimientos y con ello los servicios. Esta fluctuación requiere de añadir nuevas funcionalidades al sistema web (configuraciones, interfaces, tipos de dispositivos móviles para acceso, etc.). Dicho incremento de funcionalidades debe realizarse a lo sumo en 2 Semanas.



ESCENARIO	PARÁMETROS
Stimulus source	Desarrollador, Programador
Stimulus	Se debe Agregar una nueva Funcionalidad
Artefact	Sistema, Módulo
Environment	En Tiempo de Diseño
Response	Una Nueva Funcionalidad ha sido Agregada
Response measure	A lo sumo en 15 días

ESCENARIO 5

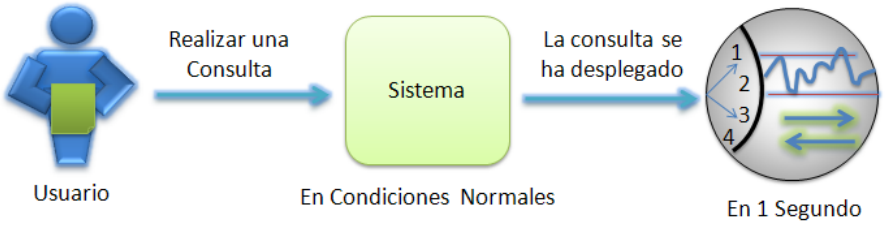
M5 - Para futuras implementaciones, el sistema debe ser capaz de adaptarse ante el desarrollo eventual de una nueva característica o módulo de requerimientos en un plazo máximo de 1 Mes.



ESCENARIO	PARÁMETROS
Stimulus source	Desarrollador, Programador
Stimulus	Se requiere de la Agregación de una nueva Característica
Artefact	Módulo
Environment	En Tiempo de Diseño
Response	La característica ha sido agregada en el tiempo establecido
Response measure	En un plazo máximo de 30 días

ESCENARIO 6

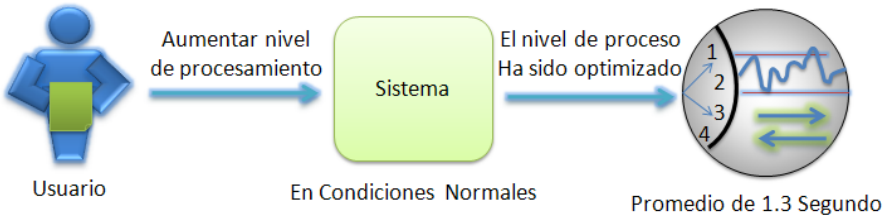
P1 - Un usuario solicita a través de una de las vistas del sistema, la consulta de un parámetro específico. El sistema debe proporcionar la respuesta "informe y detalle de la consulta" en condiciones normales en al menos 1 seg.



ESCENARIO	PARÁMETROS
Stimulus source	Usuario
Stimulus	El usuario realiza una Consulta y ésta se muestra aproximadamente en 4.5 seg.
Artefact	Sistema
Environment	En Condiciones Normales
Response	La Consulta ha sido desplegada
Response measure	Al menos en 1 segundo

ESCENARIO 7

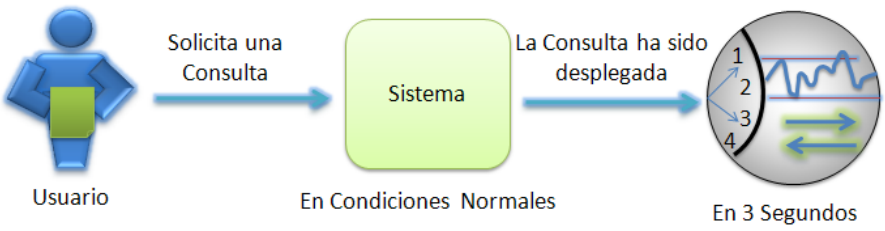
P2 - El grupo de Desarrollo y Administración han sugerido la implementación de un servidor de datos más potente para reducir la Latencia en el sistema de Memorias que se ha estimado en un rango de los 2.7 a 2.6 segundos actualmente. El incremento en el factor computacional y de proceso disminuirá el factor de Latencia en 1.3 segundos aproximadamente, en condiciones normales de carga y operatividad del sistema.



ESCENARIO	PARÁMETROS
Stimulus source	Usuario
Stimulus	Un Nuevo Servidor de Archivos será Instalado
Artefact	Sistema
Environment	En Condiciones Normales
Response	Se ha mejorado en nivel de procesamiento
Response measure	Disminución de factor de procesamiento en 1.3 Segundos

ESCENARIO 8

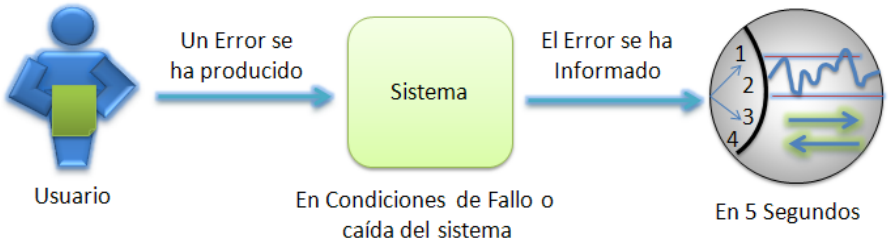
P3 - Un usuario debe poder realizar una consulta a través de cualquiera de las vistas de la aplicación en un tiempo no mayor a 3 segundos, tiempo en el cual dicha consulta debe ser desplegada en la pantalla o dispositivo móvil desde donde esté realizando la petición.



ESCENARIO	PARÁMETROS
Stimulus source	Usuario
Stimulus	Una Consulta es solicitada al Sistema
Artefact	Sistema
Environment	En Condiciones Normales
Response	La Consulta ha sido desplegada
Response measure	En 3 segundos

ESCENARIO 9

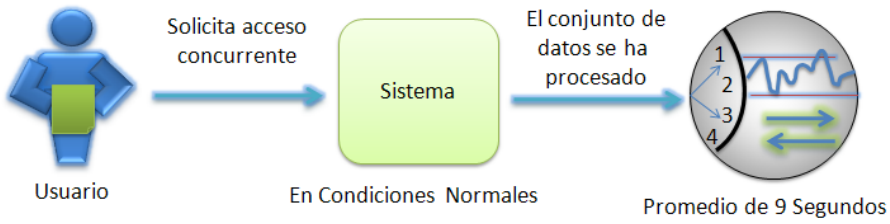
P4 - En determinados casos donde se reporte un error por parte del aplicativo o uno de sus componentes, el usuario requiere que el servicio de reporte de inconsistencias, revele el tipo de fallo y posible solución "Sistema de Ayudas" a través de la pantalla activa en un tiempo no mayor a 5 segundos.



ESCENARIO	PARÁMETROS
Stimulus source	Usuario
Stimulus	Un Error se ha Producido
Artefact	Sistema
Environment	En Condiciones de Fallo o Excepción de un módulo o ausencia de servicios
Response	Un Error específico ha sido reportado al usuario
Response measure	En un Máximo de 5 segundos

ESCENARIO 10

P5 – A nivel de acceso concurrente (por lo general en tiempo de matrícula y gestión de pagos), el sistema debe incrementar su rendimiento de tal forma que pueda procesar un volumen considerable de datos (20 transacciones en promedio) en un tiempo cercano a los 9 segundos.



ESCENARIO	PARÁMETROS
Stimulus source	Usuario
Stimulus	Un Petición de Acceso Concurrente a los Datos
Artefact	Sistema
Environment	En Condiciones Normales
Response	El Conjunto de Transacciones han sido procesadas
Response measure	En un promedio de 9 segundos

10.1.9. Mapeo de Escenarios/Responsabilidades en el sistema Matrículas

Figura 128. Mapeo de Responsabilidades

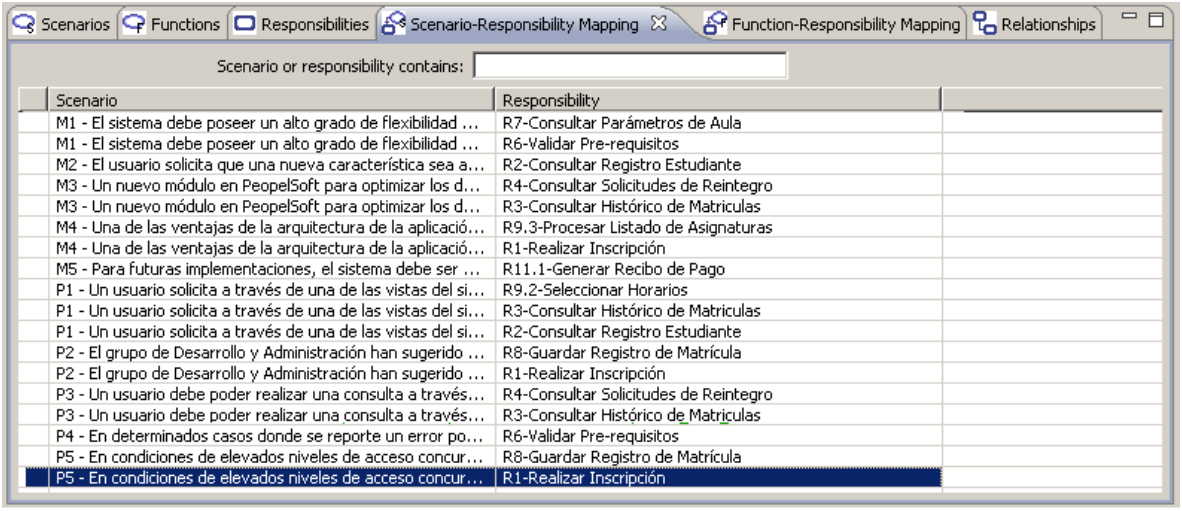
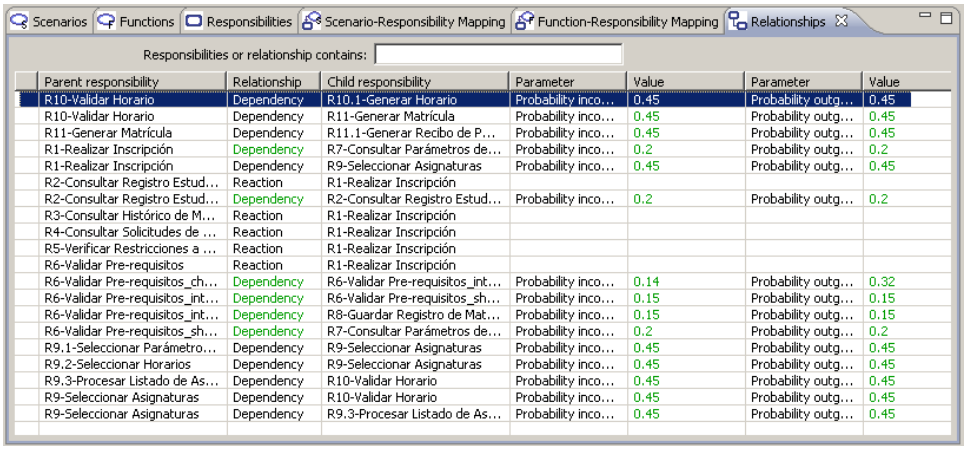


Tabla 64. Mapeo Escenarios / Responsabilidades

		ESCENARIOS									
		M1	M2	M3	M4	M5	P1	P2	P3	P4	P5
RESPONSABILIDADES	R1-Realizar Inscripción				M			P			P
	R2-Consultar Registro Estudiante		M				P				
	R3-Consultar Histórico de Matriculas			M			P		P		
	R4-Consultar Solicitudes de Reintegro			M					P		
	R5-Verificar Restricciones a Matrícula										
	R6-Validar Pre-requisitos	M								P	
	R7-Consultar Parámetros de Aula	M									
	R8-Guardar Registro de Matrícula							P			P
	R9-Seleccionar Asignaturas										
	R9.1-Seleccionar Parámetros de Asignatura										
	R9.2-Seleccionar Horarios						P				
	R9.3-Procesar Listado de Asignaturas				M						
	R10-Validar Horario										
	R10.1-Generar Horario										
	R11-Generar Matrícula										
	R11.1-Generar Recibo de Pago					M					

10.1.10. Conjunto de Relaciones en el Sistema Matrículas

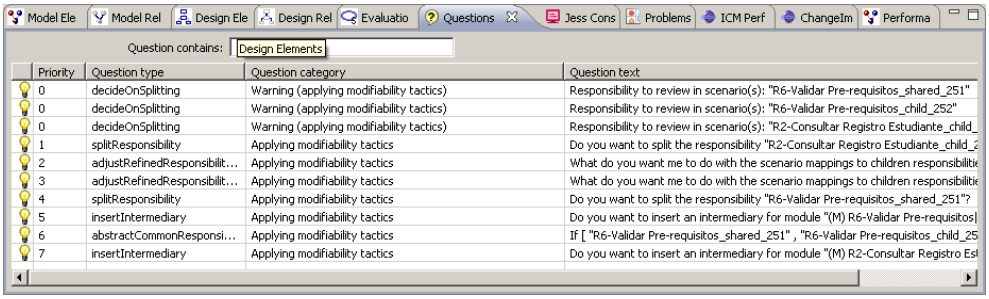
Figura 129. Conjunto de Relaciones para el sistema Matrículas



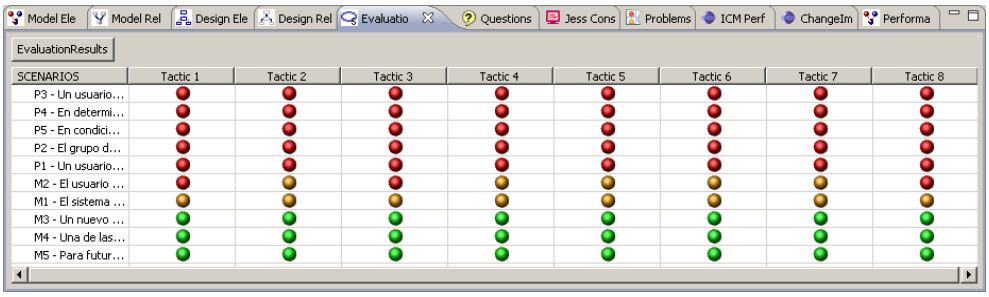
Parent responsibility	Relationship	Child responsibility	Parameter	Value	Parameter	Value
R10-Validar Horario	Dependency	R10.1-Generar Horario	Probability inco...	0.45	Probability outg...	0.45
R10-Validar Horario	Dependency	R11-Generar Matrícula	Probability inco...	0.45	Probability outg...	0.45
R11-Generar Matrícula	Dependency	R11.1-Generar Recibo de P...	Probability inco...	0.45	Probability outg...	0.45
R1-Realizar Inscripción	Dependency	R7-Consultar Parámetros de...	Probability inco...	0.2	Probability outg...	0.2
R1-Realizar Inscripción	Dependency	R9-Seleccionar Asignaturas	Probability inco...	0.45	Probability outg...	0.45
R2-Consultar Registro Estud...	Reaction	R1-Realizar Inscripción				
R2-Consultar Registro Estud...	Dependency	R2-Consultar Registro Estud...	Probability inco...	0.2	Probability outg...	0.2
R3-Consultar Histórico de M...	Reaction	R1-Realizar Inscripción				
R4-Consultar Solicitudes de ...	Reaction	R1-Realizar Inscripción				
R5-Verificar Restricciones a ...	Reaction	R1-Realizar Inscripción				
R6-Validar Pre-requisitos	Reaction	R1-Realizar Inscripción				
R6-Validar Pre-requisitos_ch...	Dependency	R6-Validar Pre-requisitos_int...	Probability inco...	0.14	Probability outg...	0.32
R6-Validar Pre-requisitos_int...	Dependency	R6-Validar Pre-requisitos_sh...	Probability inco...	0.15	Probability outg...	0.15
R6-Validar Pre-requisitos_int...	Dependency	R8-Guardar Registro de Mat...	Probability inco...	0.15	Probability outg...	0.15
R6-Validar Pre-requisitos_sh...	Dependency	R7-Consultar Parámetros de...	Probability inco...	0.2	Probability outg...	0.2
R9.1-Seleccionar Parámetro...	Dependency	R9-Seleccionar Asignaturas	Probability inco...	0.45	Probability outg...	0.45
R9.2-Seleccionar Horarios	Dependency	R9-Seleccionar Asignaturas	Probability inco...	0.45	Probability outg...	0.45
R9.3-Procesar Listado de As...	Dependency	R10-Validar Horario	Probability inco...	0.45	Probability outg...	0.45
R9-Seleccionar Asignaturas	Dependency	R10-Validar Horario	Probability inco...	0.45	Probability outg...	0.45
R9-Seleccionar Asignaturas	Dependency	R9.3-Procesar Listado de As...	Probability inco...	0.45	Probability outg...	0.45

Una vez se ha concluido con la lista de requerimientos, el arquitecto puede verificar el estado de las tácticas y el análisis hecho por ArchE sobre la arquitectura. En la vista inferior "Evaluation" se puede observar el resultado inicial del análisis del modelo.

Figura 130. Registro de Tácticas aplicadas al modelo de Matrículas



Priority	Question type	Question category	Question text
0	decideOnSplitting	Warning (applying modifiability tactics)	Responsibility to review in scenario(s): "R6-Validar Pre-requisitos_shared_251"
0	decideOnSplitting	Warning (applying modifiability tactics)	Responsibility to review in scenario(s): "R6-Validar Pre-requisitos_child_252"
0	decideOnSplitting	Warning (applying modifiability tactics)	Responsibility to review in scenario(s): "R2-Consultar Registro Estudiante_child_
1	splitResponsibility	Applying modifiability tactics	Do you want to split the responsibility "R2-Consultar Registro Estudiante_child_2
2	adjustRefinedResponsibilit...	Applying modifiability tactics	What do you want me to do with the scenario mappings to children responsibilitie
3	adjustRefinedResponsibilit...	Applying modifiability tactics	What do you want me to do with the scenario mappings to children responsibilitie
4	splitResponsibility	Applying modifiability tactics	Do you want to split the responsibility "R6-Validar Pre-requisitos_shared_251"?
5	insertIntermediary	Applying modifiability tactics	Do you want to insert an intermediary for module "(M) R6-Validar Pre-requisitos|
6	abstractCommonResponsi...	Applying modifiability tactics	If ["R6-Validar Pre-requisitos_shared_251" , "R6-Validar Pre-requisitos_child_25
7	insertIntermediary	Applying modifiability tactics	Do you want to insert an intermediary for module "(M) R2-Consultar Registro Est



SCENARIOS	Tactic 1	Tactic 2	Tactic 3	Tactic 4	Tactic 5	Tactic 6	Tactic 7	Tactic 8
P3 - Un usuario...	●	●	●	●	●	●	●	●
P4 - En determi...	●	●	●	●	●	●	●	●
P5 - En condici...	●	●	●	●	●	●	●	●
P2 - El grupo d...	●	●	●	●	●	●	●	●
P1 - Un usuario...	●	●	●	●	●	●	●	●
M2 - El usuario ...	●	●	●	●	●	●	●	●
M1 - El sistema ...	●	●	●	●	●	●	●	●
M3 - Un nuevo ...	●	●	●	●	●	●	●	●
M4 - Una de las...	●	●	●	●	●	●	●	●
M5 - Para futur...	●	●	●	●	●	●	●	●

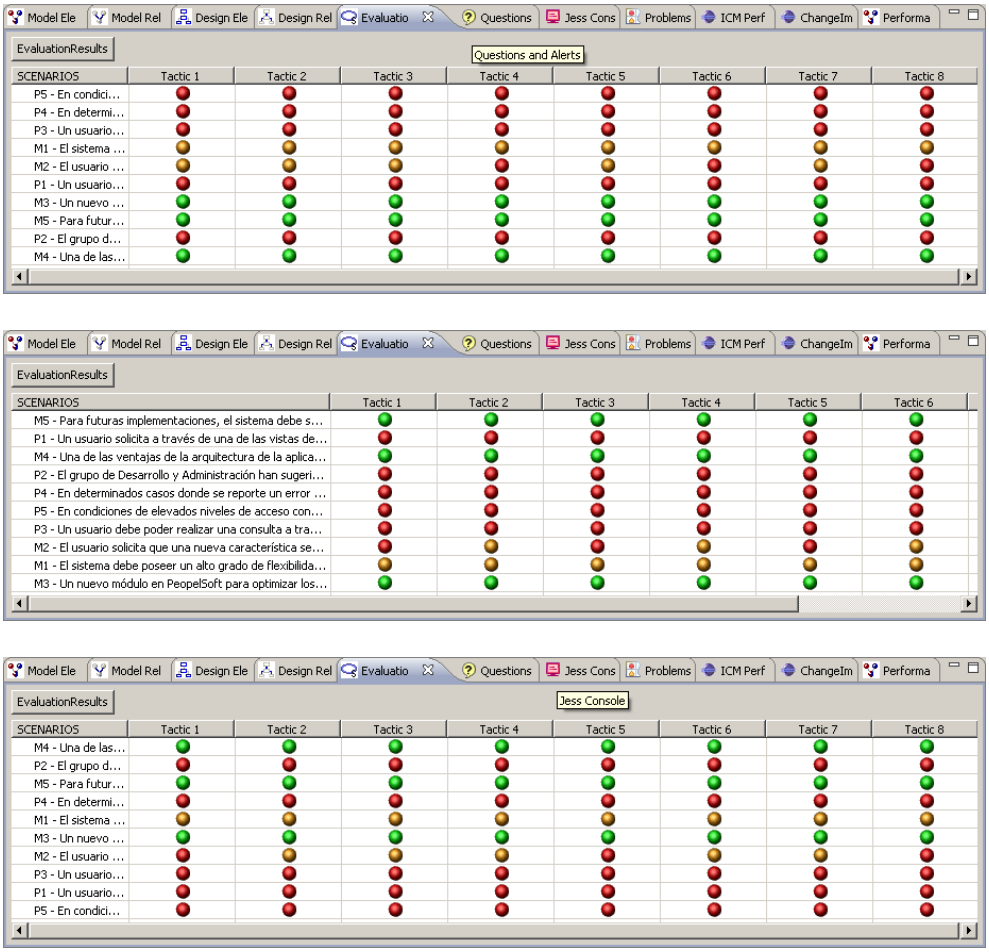
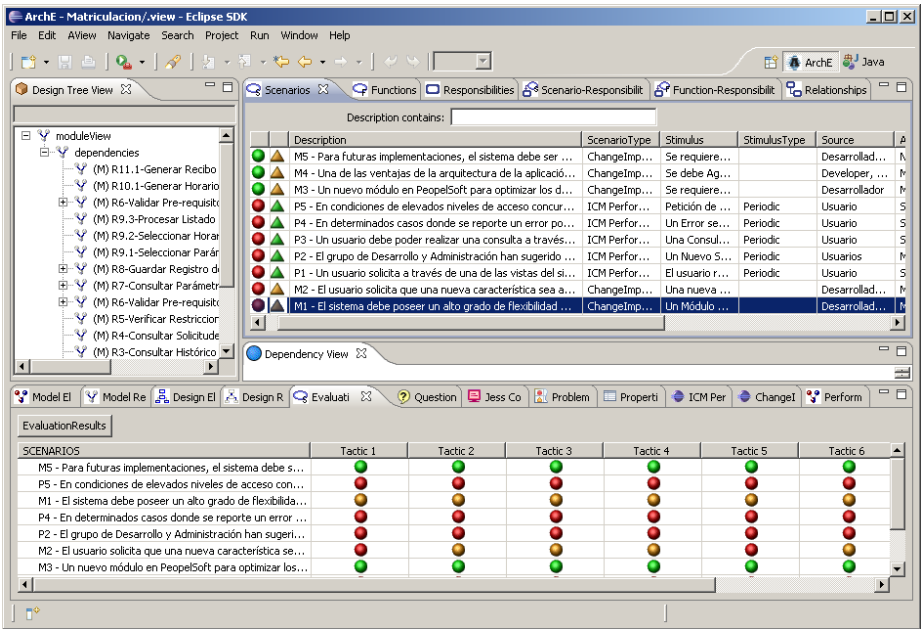


Figura 131. Registro de Tácticas aplicadas al modelo de Matrículas



Tras cada aplicación de Tácticas, los pesos o costos de las Responsabilidades fueron modificadas así mismo como los pesos de las Responsabilidades generadas "descompuestas". Tras reiteradas iteraciones, el comportamiento del modelo creció y su complejidad hizo que se generaran una tras otra nuevas Tácticas (procesando al final 8 Tácticas). Al finalizar el ejercicio solo se logró satisfacer 3 escenarios de los 10 propuestos.

Tabla 65. Resumen de algunas Tácticas aplicadas al caso de estudio Matriculación

Sat	Ipc	Esc	Tipo	T1	T2	T3	T4	T5	T6	T7	T8	Táctica Aplicada	Observación
		M5	Mod									Estado Inicial	Satisfecho con un valor de 3.8374
		M4	Mod									Estado Inicial	Satisfecho con un valor de 10.59
		M3	Mod									Estado Inicial	Satisfecho con un valor de 6.673
		P5	Per									Estado Inicial	Sin Cambios, no afectado
		P4	Per									Estado Inicial	Sin Cambios, no afectado
		P3	Per									Estado Inicial	Sin Cambios, no afectado
		P2	Per									Estado Inicial	Sin Cambios, no afectado
		P1	Per									Estado Inicial	Sin Cambios, no afectado
		M2	Mod									Estado Inicial	Sin Cambios, no afectado
		M1	Mod									Estado Inicial	Sin Cambios, no afectado
Aplciando Tácticas…. Iteración 1 : R2, R6 y R7 se dividen por múltiples dependencias, costo de cambio 3.90													
		M4	Mod									SplitResponsability	Satisfecho con un valor de 10.41
		M3	Mod									SplitResponsability	Satisfecho con un valor de 6.55
		M5	Mod									SplitResponsability	Satisfecho con un valor de 3.775
		M1	Per									SplitResponsability	Al dividir las Responsabilidades, hay un impacto positivo en el escenario
		P5	Per									SplitResponsability	Al dividir las Responsabilidades, hay un impacto positivo en el escenario
		P4	Per									SplitResponsability	Al dividir las Responsabilidades, hay un impacto positivo en el escenario
		P3	Per									SplitResponsability	Al dividir las Responsabilidades, hay un impacto positivo en el escenario
		P2	Per									SplitResponsability	Al dividir las Responsabilidades, hay un impacto positivo en el escenario
		P1	Mod									SplitResponsability	Al dividir las Responsabilidades, hay un impacto positivo en el escenario
		M2	Mod									SplitResponsability	La palicación de la Táctica Degradará el escenario
Se refinará un conjunto de Responsabilidades, para evaluar costo computacional por costo de nuevas Responsabilidades derivadas de la División de Responsabilidades en R2, R6 y R7													
		M5	Mod									AdjustRefinedResponsability	Satisfecho con un valor de 10.41
		M4	Mod									AdjustRefinedResponsability	Satisfecho con un valor de 6.55
		M3	Mod									AdjustRefinedResponsability	Satisfecho con un valor de 3.775
		P5	Per									AdjustRefinedResponsability	Al refinar el conjutno de Responsabilidades, se impacta positivamente el escenario
		P4	Per									AdjustRefinedResponsability	Al refinar el conjutno de Responsabilidades, se impacta positivamente el escenario
		P3	Per									AdjustRefinedResponsability	Al refinar el conjutno de Responsabilidades, se impacta positivamente el escenario
		P2	Per									AdjustRefinedResponsability	Al refinar el conjutno de Responsabilidades, se impacta positivamente el escenario
		P1	Per									AdjustRefinedResponsability	Al refinar el conjutno de Responsabilidades, se impacta positivamente el escenario
		M2	Mod									AdjustRefinedResponsability	Sin Cambios, no afectado
		M1	Mod									AdjustRefinedResponsability	Sin Cambios, no afectado
Se refinará un conjunto de Responsabilidades, para evaluar costo computacional. Se introducen intermediarios para R2, R6 y R7. Costo Intermediario=3.900 para R2 y 9.70 para R6													
		M5	Mod									AdjustRefinedResponsability	Satisfecho con un valor de 10.41
		M4	Mod									AdjustRefinedResponsability	Satisfecho con un valor de 6.55
		M3	Mod									AdjustRefinedResponsability	Satisfecho con un valor de 3.775
		P5	Per									AdjustRefinedResponsability	Al refinar el conjutno de Responsabilidades, se impacta positivamente el escenario
		P4	Per									AdjustRefinedResponsability	Al refinar el conjutno de Responsabilidades, se impacta positivamente el escenario
		P3	Per									AdjustRefinedResponsability	Al refinar el conjutno de Responsabilidades, se impacta positivamente el escenario
		P2	Per									AdjustRefinedResponsability	Al refinar el conjutno de Responsabilidades, se impacta positivamente el escenario
		P1	Per									AdjustRefinedResponsability	Al refinar el conjutno de Responsabilidades, se impacta positivamente el escenario

		M2	Mod									AdjustRefinedResponsability	Sin Cambios, no afectado
		M1	Mod									AdjustRefinedResponsability	Sin Cambios, no afectado
Se produjo un impacto positivo en todos los escenarios, al dividir los atributos de las nuevas Responsabilidades quienes compartíen algunas características comunes. Se agruparon en nuevas Respon.													
		M4	Mod									AbstractCommonResponsabil	Satisfecho con un valor de 8.6207
		M3	Mod									AbstractCommonResponsabil	Satisfecho con un valor de 6.4071
		M5	Mod									AbstractCommonResponsabil	Satisfecho con un valor de 3.7035
		M1	Mod									AbstractCommonResponsabil	Aunque se mejora el atributo en M1, no satisface al escenario. Se crea nueva Táctica
		M2	Mod									AbstractCommonResponsabil	Aunque se mejora el atributo en M2, no satisface al escenario. Se crea nueva Táctica
		P5	Per									AbstractCommonResponsabil	Aunque se mejora el atributo en P5, no satisface al escenario. Se crea nueva Táctica
		P4	Per									AbstractCommonResponsabil	Aunque se mejora el atributo en P4, no satisface al escenario. Se crea nueva Táctica
		P3	Per									AbstractCommonResponsabil	Aunque se mejora el atributo en P3, no satisface al escenario. Se crea nueva Táctica
		P2	Per									AbstractCommonResponsabil	Aunque se mejora el atributo en P2, no satisface al escenario. Se crea nueva Táctica
		P1	Per									AbstractCommonResponsabil	Aunque se mejora el atributo en P1, no satisface al escenario. Se crea nueva Táctica
Se refinará un intermediario para R6, con costo de 3.90, reduciendo el escenario de 9.70 a 8.70 personas/dia. Este cambio disminuye el costo de escenarios M4, M3 y M5 (satisfechos)													
		M4	Mod									InsertIntermediary	El escenario no cambia, se satisface pero disminuye el costo del mismo
		M3	Mod									InsertIntermediary	El escenario no cambia, se satisface pero disminuye el costo del mismo
		M5	Mod									InsertIntermediary	El escenario no cambia, se satisface pero disminuye el costo del mismo
		M1	Mod									InsertIntermediary	Se mejora el atributo, pero el escenario aún no se satisface con las Tácticas
		M2	Mod									InsertIntermediary	El escenario no cambia, aún no se satisface pero disminuye el costo del mismo
		P5	Per									InsertIntermediary	Se mejora el atributo, pero el escenario aún no se satisface con las Tácticas
		P4	Per									InsertIntermediary	Se mejora el atributo, pero el escenario aún no se satisface con las Tácticas
		P3	Per									InsertIntermediary	Se mejora el atributo, pero el escenario aún no se satisface con las Tácticas
		P2	Per									InsertIntermediary	Se mejora el atributo, pero el escenario aún no se satisface con las Tácticas
		P1	Per									InsertIntermediary	Se mejora el atributo, pero el escenario aún no se satisface con las Tácticas

Convenciones: **Sat**: Grado de Satisfacción del escenario. **Ipc**: Impacto **Esc**: Escenario. **Tipo**: Modifiability o Performance. **T**: Táctica

- El escenario está satisfecho.
- El escenario No está satisfecho.
- El escenario ha sido Degradado y por lo tanto disminuye su utilidad.
- El último cambio realizado (Tácticas aplicadas) en los escenarios de calidad, impactó de forma positiva al escenario.
- El último cambio realizado (Tácticas aplicadas) en los escenarios de calidad, impactó de forma negativa al escenario.
- El Cambio no impacta ni positiva ni negativamente al escenario.

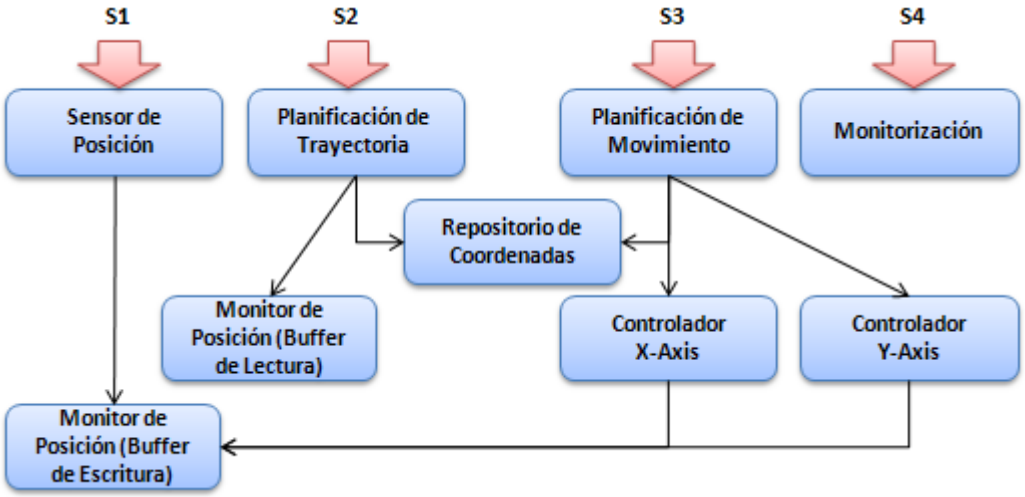
10.2. EVALUACIÓN DE UN SISTEMA BASADO EN ICMPerformance RF

El siguiente modelo ha sido adaptado del documento "***Towards automation of architectural tactics application – an example with ArchE***" (Champagne & Gagné, 2011) por considerarse un excelente modelo para la aplicación del atributo de performance en un sistema determinado (en este caso a un sistema robótico).

Los valores han sido modificados y los escenarios se han tratado de aproximar al modelo real para efectos de explicación y ampliación de los conceptos de los que trata el presente informe de investigación. Inicialmente el modelo se encuentra optimizado y ArchE no generará ninguna Táctica para la mejora de los escenarios. Como ejercicio práctico se va a incluir un nuevo escenario basado en el atributo Modifiablility y se realizará la trazabilidad con una de las funcionalidades incluidas en el modelo.

10.2.1. Requisitos Funcionales y Dependencias del Sistema

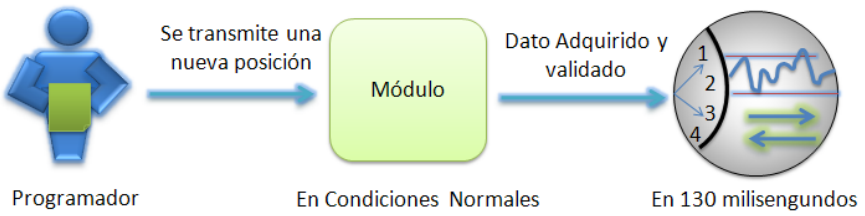
Figura 132. Requisitos Funcionales Dependencias para el Sistema Robótico



10.2.2. Refinamiento de Escenarios

ESCENARIO 1

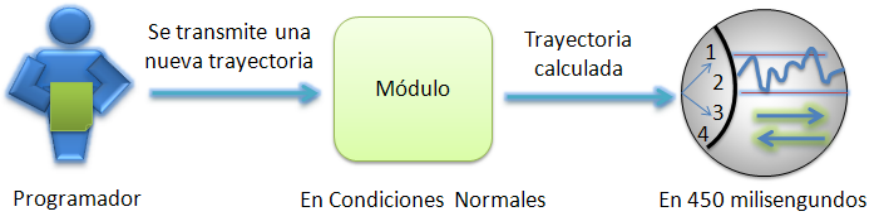
S1 - Un sensor de movimiento captura la posición del brazo del robot cuando el sistema se ejecuta en condiciones normales. Cada valor proveniente del sensor debe ser adquirido, validado y almacenado en el sistema antes de que la próxima actualización o nueva posición vector del sensor sea adquirida. El tiempo de respuesta debe ser menor a 130 milisegundos.



ESCENARIO	PARÁMETROS
Stimulus source	Programador
Stimulus	Se transmite una nueva posición del sensor
Artefact	Módulo
Environment	Bajo Condiciones Normales
Response	Cada valor adquirido, validado y Almacenado
Response measure	En 130 milisegundos

ESCENARIO 2

S2 - Para evitar exigencias inútiles de velocidad y aceleración en el brazo del robot (daños por sobre-esfuerzos en ejes y servomotores), se requiere de una planificación precisa de la trayectoria del mismo. El sistema debe ser capaz de calcular con precisión la trayectoria sobre los ejes o grados de libertad en un tiempo no mayor a 450 milisegundos antes de que se traslapen nuevas trayectorias procesadas por la interfaz y los sensores de posición. Cada trayectoria se transmite y es almacenada en un archivo base o "repositorio" de configuraciones para trayectorias. El componente de Planificación obtiene la información de la posición del brazo proporcionada por el monitor de posiciones.

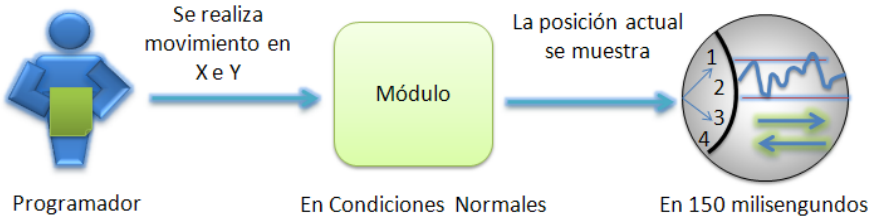


ESCENARIO	PARÁMETROS
Stimulus source	Programador
Stimulus	Se transmite una nueva trayectoria sobre los ejes
Artefact	Módulo
Environment	Bajo Condiciones Normales
Response	Trayectoria de los ejes calculada
Response measure	Hasta 450 milisegundos

ESCENARIO 3

S3 - El sistema que controla los mecanismos de movimiento (servos) dependen de las coordenadas X y Y proporcionadas por los sensores que capturan del contexto las posiciones relativas "axis" para el movimiento en el eje coordenado, los cuales son

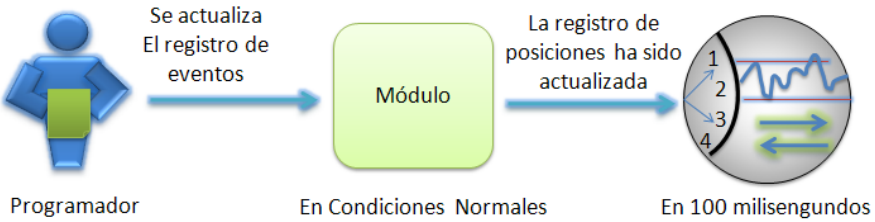
adquiridos y almacenados por el respectivo controlador de cada eje (X o Y). Los controladores de los servomecanismos dependen de la información almacenada en el archivo de trayectorias para poder realizar los movimientos en los servos para ambos ejes. "El repositorio debe siempre encontrarse con datos, pues la operación del brazo depende del buffer de datos almacenados para su ejecución, de lo contrario el brazo abortará el proceso de desplazamiento". Se estima que el promedio de tiempo por planificar cada movimiento debe ser de 150 milisegundos.



ESCENARIO	PARÁMETROS
Stimulus source	Programador
Stimulus	Realizar el movimiento en los ejes coordenados X y Y
Artefact	Módulo
Environment	Bajo Condiciones Normales
Response	El movimiento y posición actual se muestra en pantalla
Response measure	En 150 milisegundos

ESCENARIO 4

S4 - El componente Monitor se encarga de registrar y monitorear operaciones secundarias o de baja prioridad (estadísticas de movimiento, tiempo en ejecución, status de los sensores, errores y log de configuración). Esta es una tarea periódica del sistema que al ejecutarse en condiciones normales de funcionamiento actualiza su registro de eventos en un tiempo de 2000ms. El sistema debe registrar los eventos en un tiempo de 100 milisegundos antes de que se realice la próxima grabación de parámetros de eventos del brazo del robot.



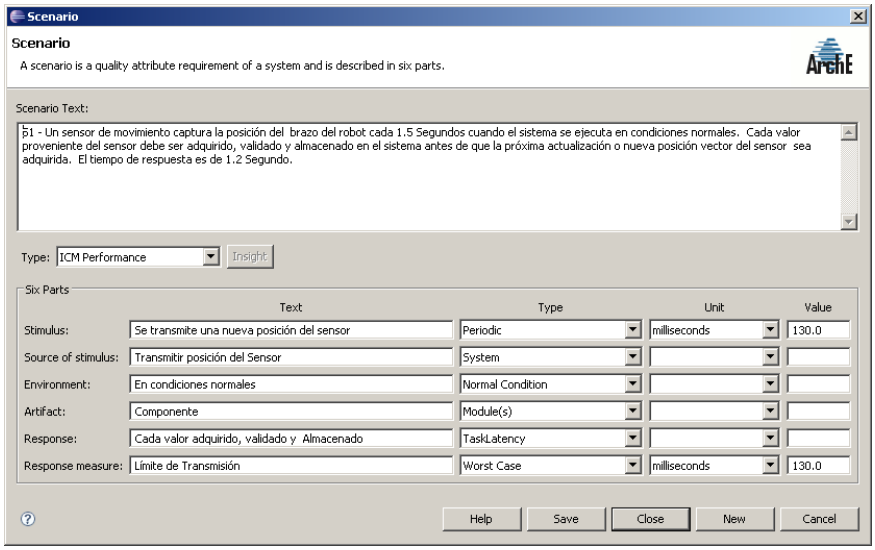
ESCENARIO	PARÁMETROS
Stimulus source	Programador
Stimulus	Registrar eventos secundarios
Artefact	Módulo
Environment	Bajo Condiciones Normales
Response	El registro de eventos ha sido procesado y almacenado.
Response measure	En un tiempo de 100 milisegundos

10.2.3. Selección del Framework de Razonamiento

El Framework de Razonamiento para todos los escenarios corresponde al ICM Performance para el atributo de calidad Rendimiento.

10.2.4. Especificación de los requerimientos en ArchE

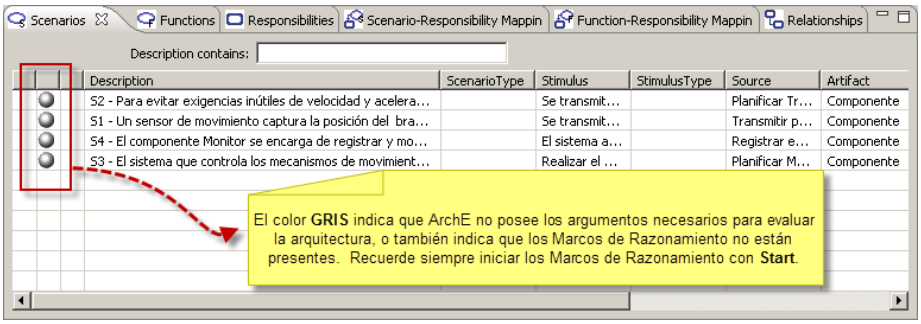
Figura 133. Refinamiento de Escenarios



Nota: Es aconsejable que tras cada proceso de adición o modificación de los atributos en los escenarios u otras funcionalidades del sistema, el usuario realice el grabado o almacenamiento de las bases de datos y hechos. Para ello diríjase a la opción ***Project*** del menú y sobre el menú desplegable seleccione la opción **Persist Fact Base**.

Una vez finalizada la inclusión de escenarios, ArchE mostrará en la vista de Escenarios el registro de los mismos. Cada uno de ellos puede ser modificado o consultado las veces que se requieran. El color **Gris** inicialmente le indicará al arquitecto que ArchE aún no posee los suficientes argumentos para evaluar el modelo. El color también persiste cuando los Frameworks de Razonamiento no han sido iniciados. Vaya a la vista de cada uno de los marcos y reinícielos con el botón Start. Si no lo encuentra, diríjase al menú de ArchE en la opción ***Windows > Show View > Other*** y Agregue los dos marcos de **Razonamiento en ArchE External RF Samples**.

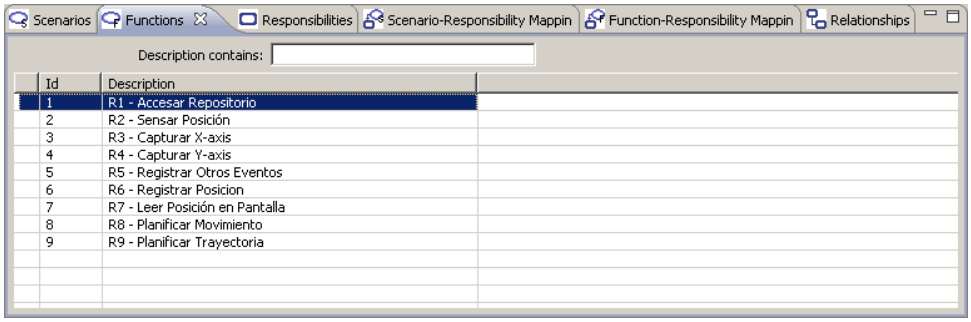
Figura 134. Vista de Escenarios



10.2.5. Agregar Funcionalidades

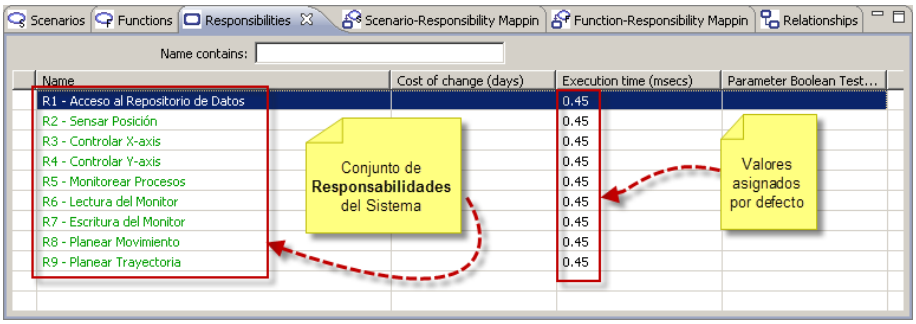
El paso a continuación corresponde al ingreso de las funciones, las cuales harán parte del conjunto de Responsabilidades del sistema. Diríjase a la vista Functions de ArchE, presione click derecho y proceda a crear una a una las funcionalidades.

Figura 135. Ingresando una Nueva Funcionalidad



10.2.6. Ingresando Responsabilidades

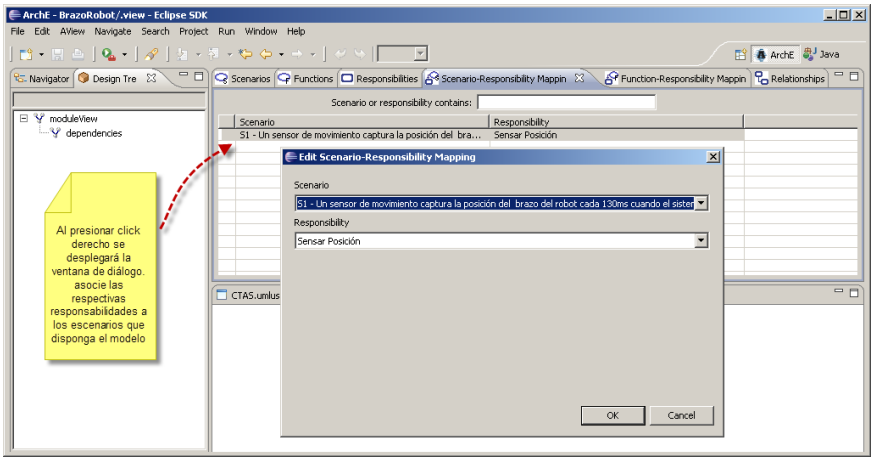
Figura 136. Conjunto de Responsabilidades



10.2.7. Mapeado de Escenarios a Responsabilidades

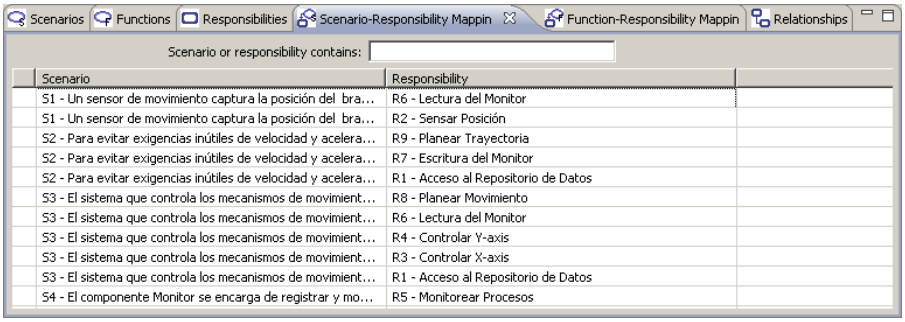
El siguiente paso consiste en mapear (asociar) los Escenarios con las respectivas responsabilidades.

Figura 137. Mapeando Escenarios a Responsabilidades



En el caso anterior, cada escenario puede estar asociado a muchas responsabilidades y viceversa. La lista de mapeo se muestra a continuación.

Figura 138. Listado de Asociaciones entre Escenarios y Responsabilidades

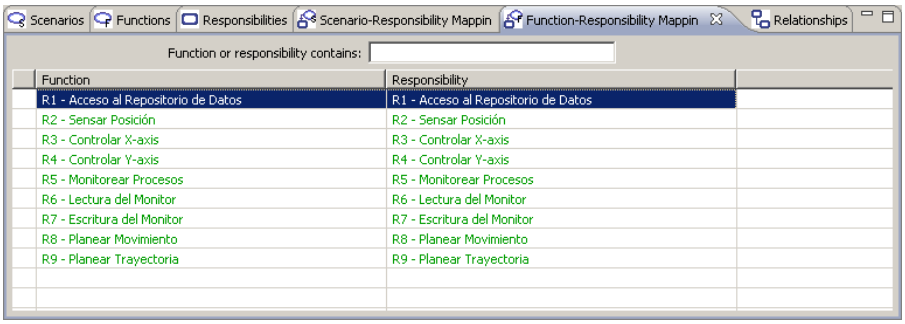


Scenario	Responsibility
S1 - Un sensor de movimiento captura la posición del bra...	R6 - Lectura del Monitor
S1 - Un sensor de movimiento captura la posición del bra...	R2 - Sensar Posición
S2 - Para evitar exigencias inútiles de velocidad y acelera...	R9 - Planear Trayectoria
S2 - Para evitar exigencias inútiles de velocidad y acelera...	R7 - Escritura del Monitor
S2 - Para evitar exigencias inútiles de velocidad y acelera...	R1 - Acceso al Repositorio de Datos
S3 - El sistema que controla los mecanismos de movimient...	R8 - Planear Movimiento
S3 - El sistema que controla los mecanismos de movimient...	R6 - Lectura del Monitor
S3 - El sistema que controla los mecanismos de movimient...	R4 - Controlar Y-axis
S3 - El sistema que controla los mecanismos de movimient...	R3 - Controlar X-axis
S3 - El sistema que controla los mecanismos de movimient...	R1 - Acceso al Repositorio de Datos
S4 - El componente Monitor se encarga de registrar y mo...	R5 - Monitorear Procesos

10.2.8. Mapeado de Funcionalidades a Responsabilidades

Este proceso se genera de forma automática. La vista le permitirá visualizar los datos referidos al mapeo entre Funcionalidades y Responsabilidades del sistema.

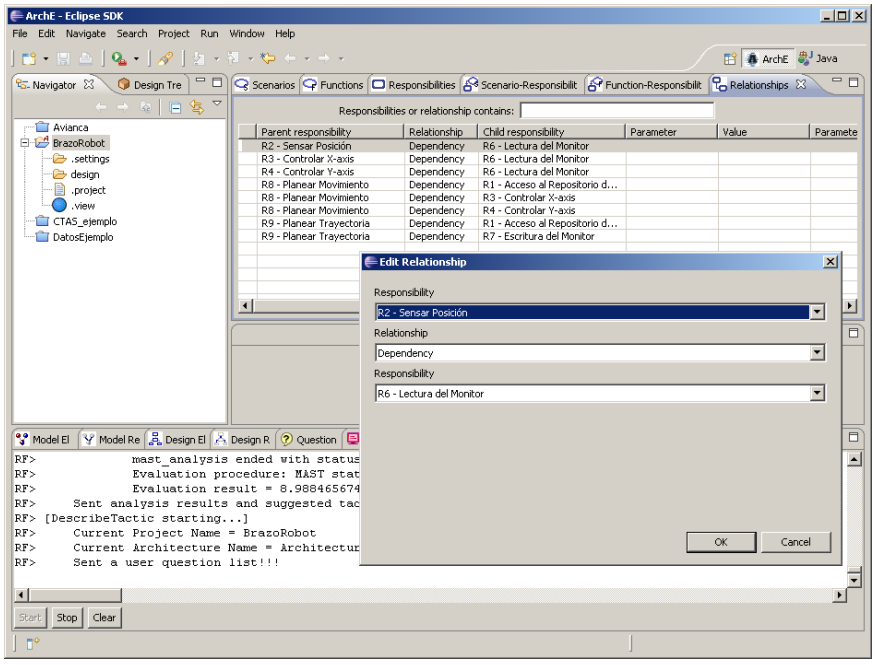
Figura 139. Listado de Asociaciones entre Funciones y Responsabilidades



El último paso en el proceso de requerimientos para la arquitectura ArchE corresponde a la definición de Relaciones. A partir de éstas se definen las dependencias entre las diferentes Responsabilidades. Existen dos posibilidades de relación para este conjunto de elementos: de ***Dependencias***, de ***Contenencia*** y de ***Reacción***.

10.2.9. Relaciones

Figura 140. Agregando un Conjunto de Relaciones para el Caso Práctico



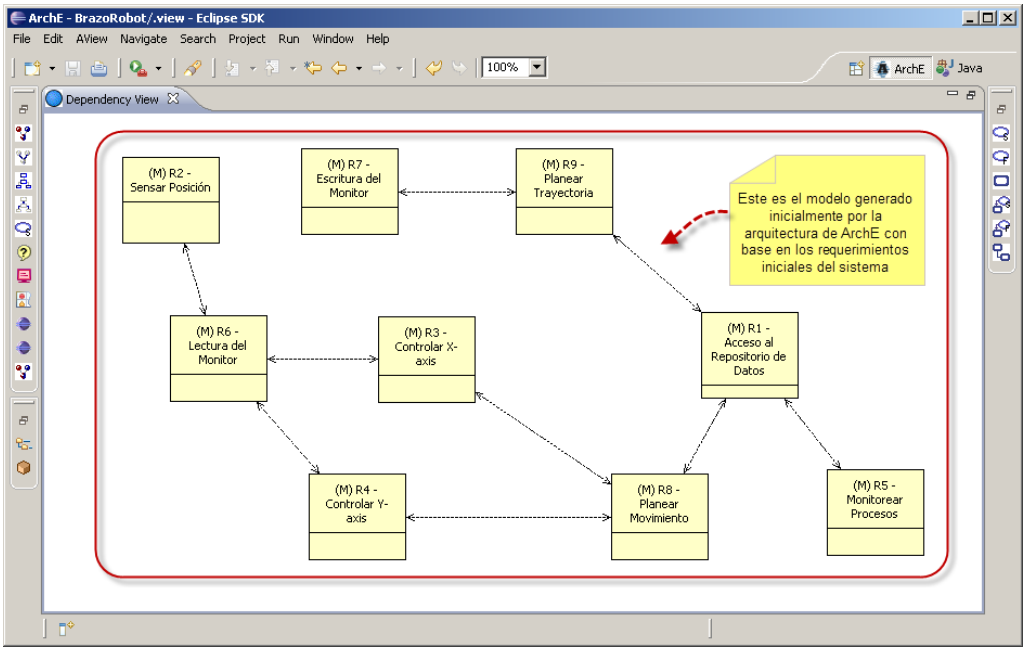
10.2.10. Generación y Optimización del Modelo

El paso final consiste en verificar los resultados del proceso de análisis y dar aplicación a las tácticas sugeridas por ArchE. El registro de análisis muestra como ArchE ejecuta el proceso de testing sobre los 4 Escenarios propuestos en el modelo de ejemplo. Para acceder al conjunto de Tácticas sugeridas ubique la vista Questions y allí encontrará las sugerencias que hace ArchE para mejorar la arquitectura.

En la siguiente sección se implementará el ejemplo CTAS como caso práctico para la evaluación de Arquitecturas con ArchE.

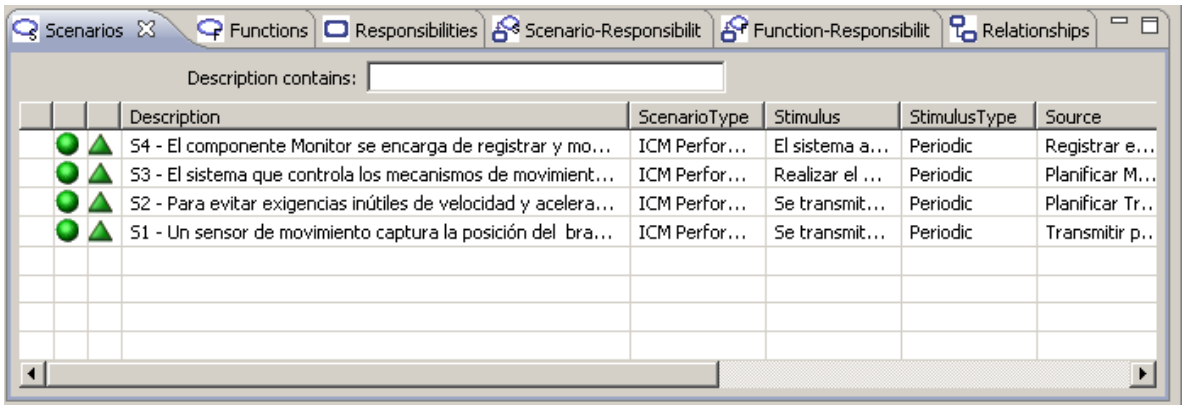
Nota: Recuerde que tras cada proceso de adición o modificación de los atributos en los escenarios u otras funcionalidades del sistema, el usuario debe salvar los cambios en las bases de datos y en la base de hechos del Sistema Experto. Para ello diríjase a la opción ***Project*** del menú y sobre el menú desplegable seleccione la opción **Persist Fact Base**.

Figura 141. Modelo Generado por ArchE



En este punto, el conjunto de requerimientos del sistema introducidos inicialmente a la arquitectura de ArchE no presenta sugerencias de mejora ya que el modelo es consistente debido a su bajo nivel de acoplamiento entre los componentes o módulos del sistema. A continuación se presenta el conjunto de escenarios con su respectiva evaluación.

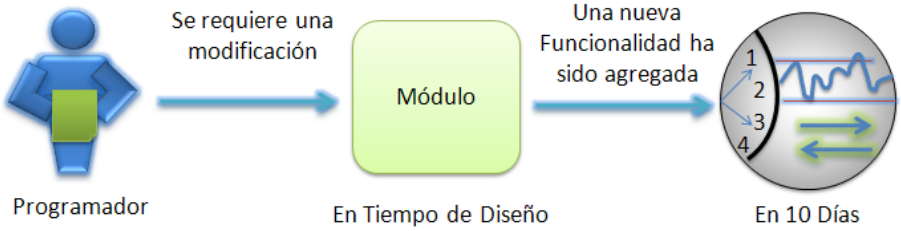
Figura 142. Modelo Inicial para el Sistema Robótico



Para efectos del ejercicio práctico aplicando un conjunto de Tácticas, se introducirá al sistema basado en Performance, un nuevo escenario basado en Modificabilidad. Este nuevo escenario describe en detalle una pequeña modificación que debe realizarse en un determinado módulo. El nuevo escenario se cita a continuación.

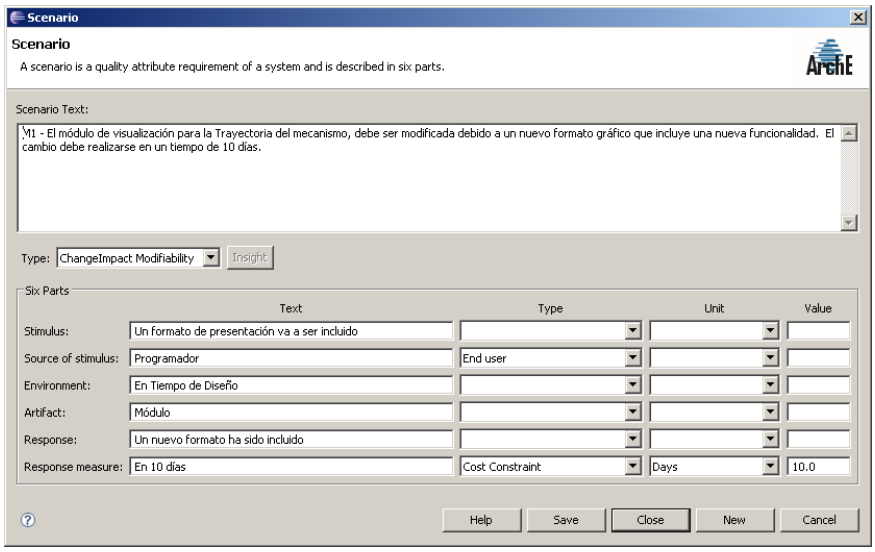
ESCENARIO 5

M1 - El módulo de visualización para la Trayectoria del mecanismo, debe ser modificado debido a un nuevo formato gráfico que incluye una nueva funcionalidad. El cambio debe realizarse en un tiempo de 10 días.



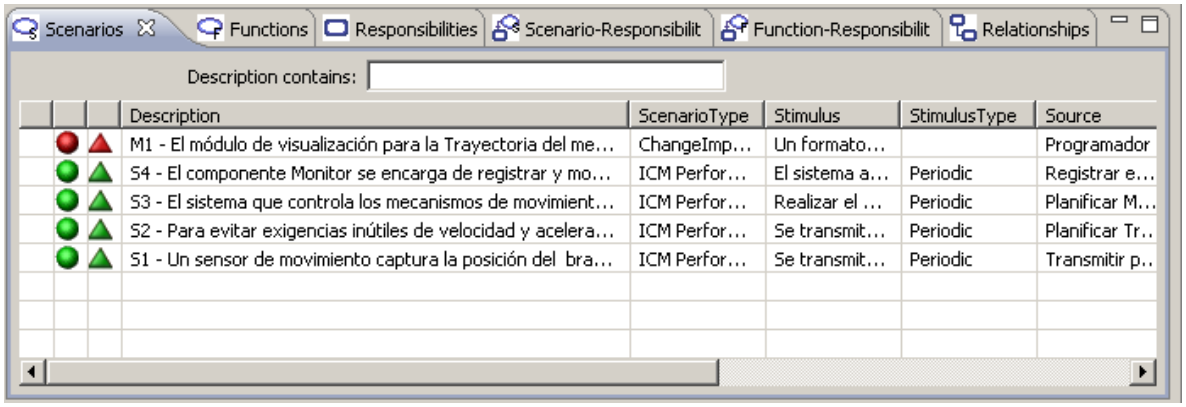
ESCENARIO	PARÁMETROS
Stimulus source	Programador
Stimulus	Un formato de presentación va a ser incluido
Artefact	Módulo
Environment	En Tiempo de Diseño
Response	Un nuevo formato ha sido incluido
Response measure	En 10 días

Figura 143. Refinamiento para el Nuevo Escenario en el sistema Robótico



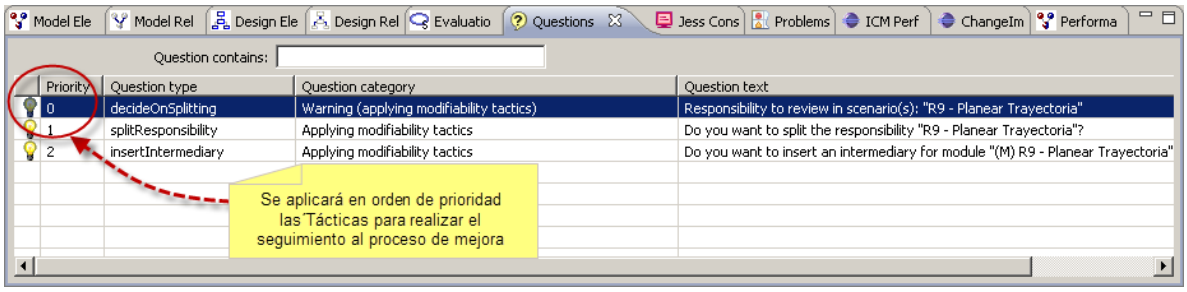
El marco general de escenarios ahora incluye un nuevo escenario, pero ahora este nuevo requerimiento se basa en el atributo de calidad de Modificabilidad.

Figura 144. Nuevo Conjunto de escenarios para el sistema Robótico



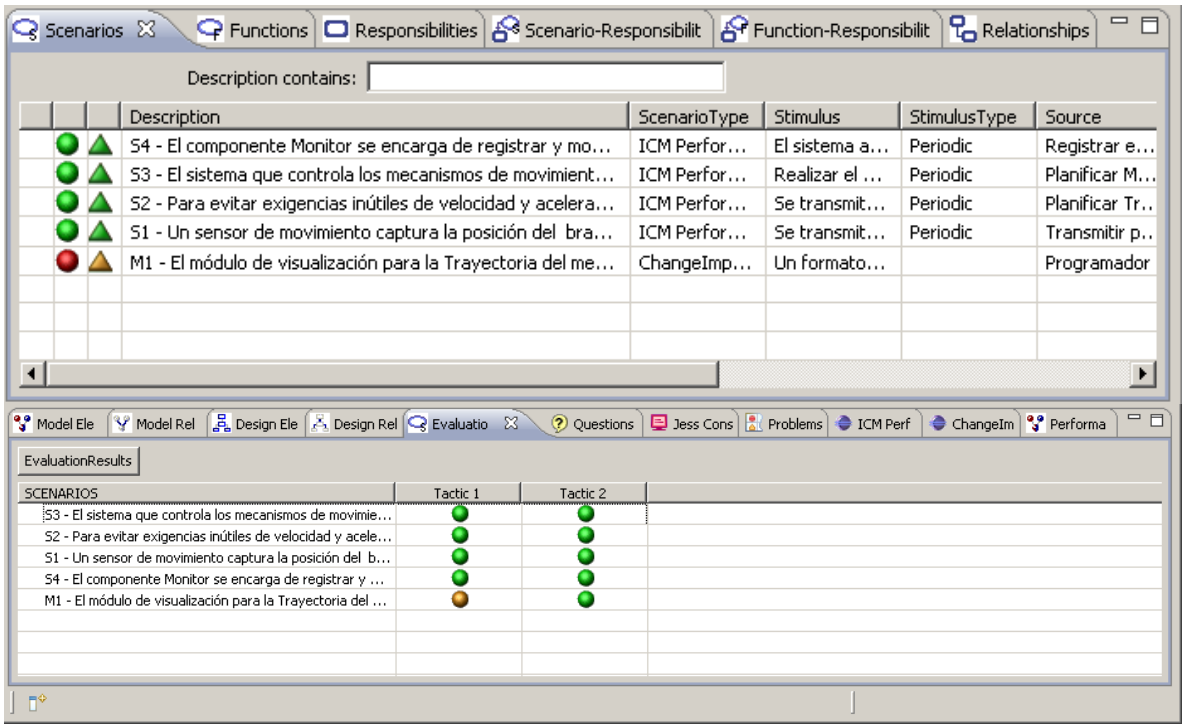
Una vez relacionado el escenario con una de las Responsabilidades del sistema, el sistema ArchE establece que el nuevo requerimiento No cumple con el escenario de calidad dispuesto para la modificación en una de las características del sistema. A partir de esta novedad, ArchE sugiere al arquitecto una serie de Tácticas a aplicar para la mejora de los atributos de calidad (Fig. 145).

Figura 145. Tácticas sugeridas por ArchE para el sistema Robótico



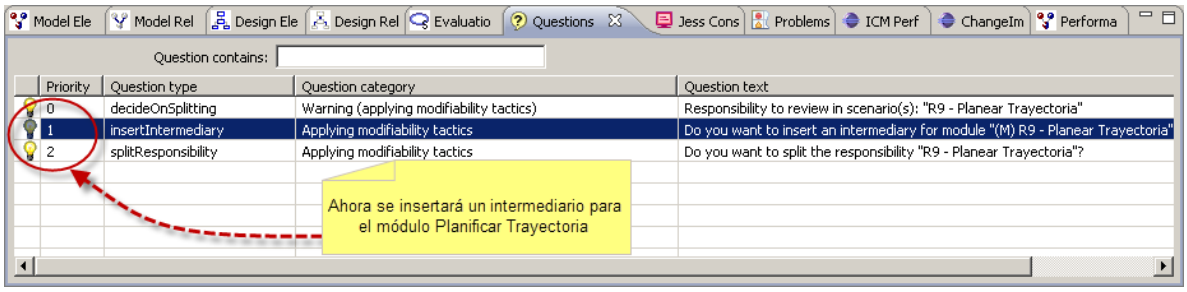
La primera Táctica que se aplicará corresponde a la que contiene la prioridad "0". Está táctica es más informativa, ya que al aplicarla no genera ningún cambio en el modelo. Este tipo de mensaje recomienda al arquitecto que la división de responsabilidades sobre "R9 - Planear Trayectoria" no siempre decrementa "optimiza" el costo total del escenario. Si se realiza la división en 2 Responsabilidades, el arquitecto debe decidir con posterioridad cuál de las dos responsabilidades impactará de forma positiva el cambio en el modelo general. El resultado obtenido se muestra a continuación.

Figura 146. Evaluación para Táctica aplicada al sistema Robótico



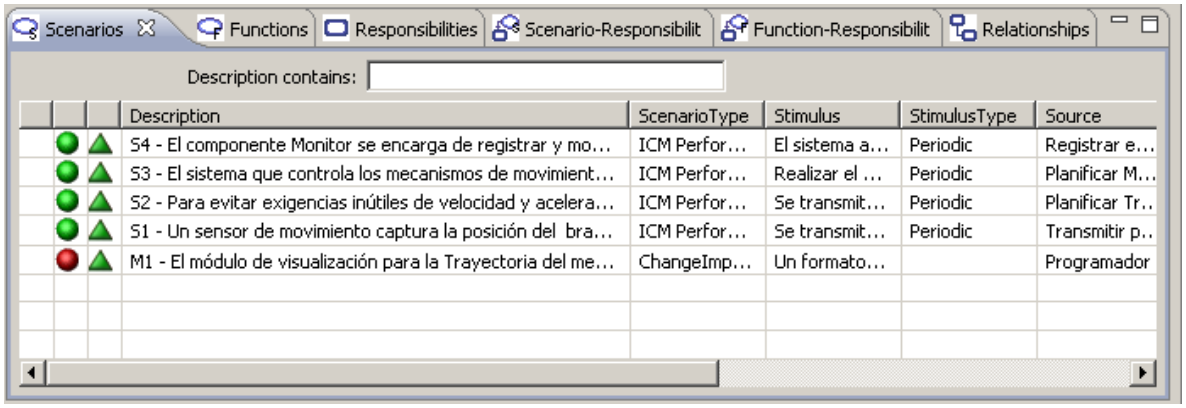
En el caso anterior no se gestionaron mejoras. A continuación se procederá a insertar un intermediario para el módulo "R9 – Planear Trayectoria".

Figura 147. Táctica de Inserción de Intermediario en el sistema Robótico



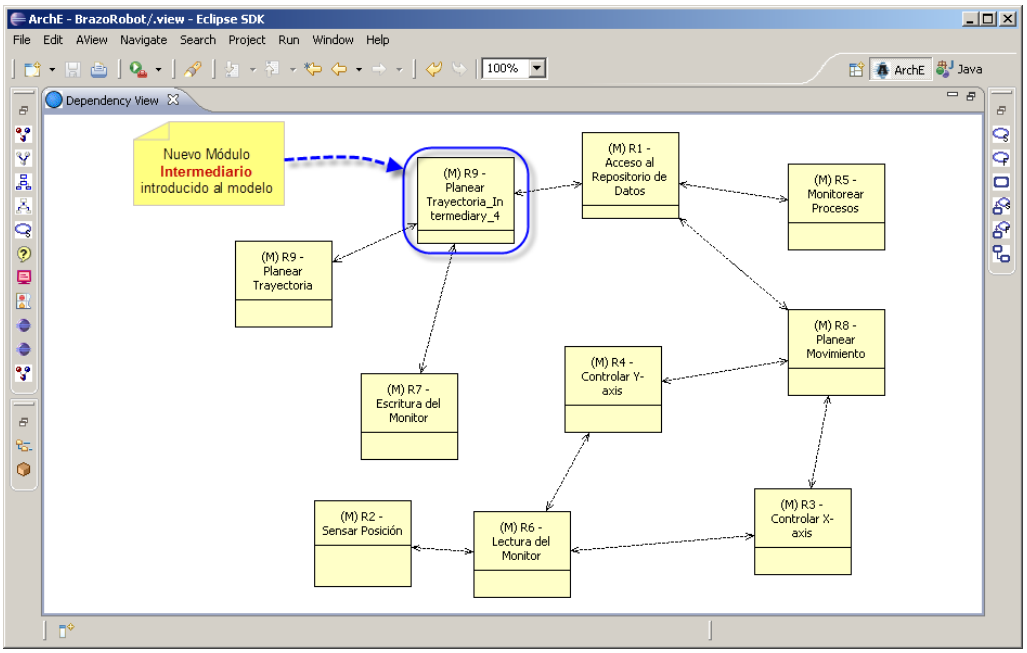
Las dependencias del módulo "(M) R9 - Planear Trayectoria", impactan a uno de los escenarios. La inserción de un intermediario podría disminuir el costo total del escenario. Un intermediario con un costo de cambio de "30.0" personas/día reducirá el costo del escenario de "11.577499999999999" personas/día a "9.9625" personas/día. Al dividir en 2 partes la Responsabilidad "R9 - Planear Trayectoria" no siempre se reduce el costo total en los escenarios de calidad. Se debe decidir más adelante cuál de las responsabilidades resultantes (hijas) impactará mejor con base en los cambios.

Figura 148. Escenarios con su respectiva mejora "Iteración n+1"



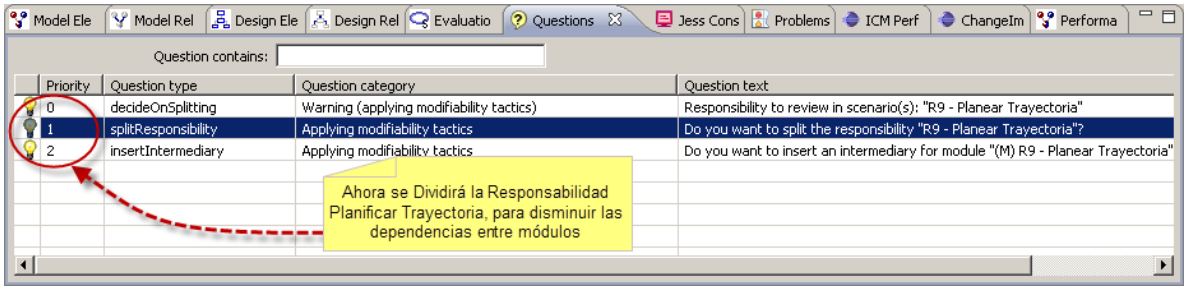
El modelo generado a partir de esta Táctica aplicada se muestra a continuación.

Figura 149. Modelo n+1 generado por ArchE en el sistema Robótico



La siguiente Táctica a aplicar corresponde a la división de Responsabilidades. Este tipo de Táctica se aplica para disminuir el acoplamiento entre componentes debido a un número mayor de dependencias entre responsabilidades.

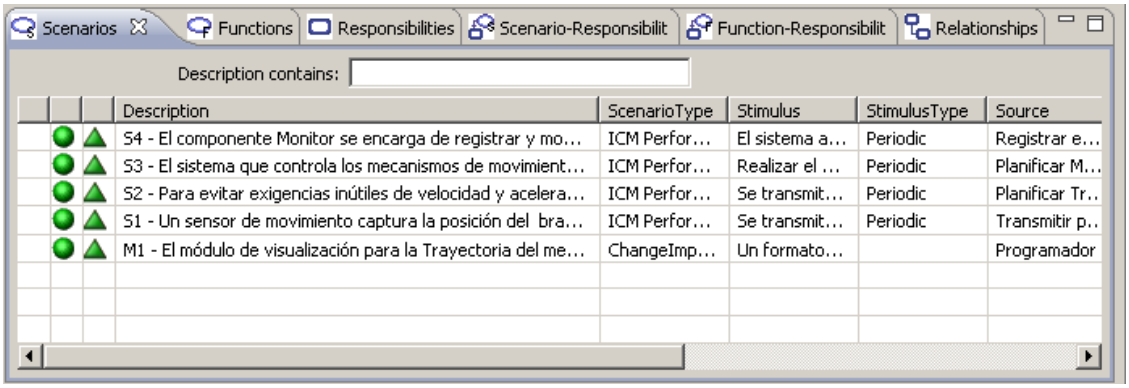
Figura 150. Táctica de División de Responsabilidades en el sistema Robótico



En este caso, ArchE ha detectado que la responsabilidad "R9 - Planear Trayectoria" tiene múltiples dependencias, lo que incrementa el acoplamiento entre módulos. Dividir en 2 nuevas responsabilidades a "R9 - Planear Trayectoria" minimizaría tales dependencias. ArchE estima que la aplicación de esta Táctica disminuirá el costo de "10.029" a "8.169041666666665" personas/día para el cambio en el escenario. Existe otra fuerte dependencia para el módulo "(M) R9 - Planear Trayectoria", lo cual sugiere también la inserción de un intermediario lo que reducirá el costo del escenario. Un intermediario con costo de "24.200000000000003" personas/día reduciría el costo del escenario de "10.029" personas/día a "8.815722727272728" personas/día. El cambio está sujeto a

revisión por parte del arquitecto para decidir cuál de las responsabilidades resultantes afectará de forma positiva el cambio.

Figura 151. Escenarios con su respectiva mejora "Iteración n+2"



Al cumplirse con todos los escenarios de calidad, se puede afirmar que el sistema ha sido optimizado. Como se ha enunciado en apartados anteriores, ArchE es una herramienta que sugiere la mejora de un sistema a partir de los atributos de calidad basados en escenarios. Es potestad del arquitecto o diseñador, decidir si el modelo generado corresponde a la arquitectura real "requerida", para poder implementarla posteriormente a su diseño. Cuando ArchE concluye y no determina más mejoras la vista de Questions y Evaluations no mostrará ningún resultado, de hecho; la vista de escenarios mostrará los círculos y triángulos en color verde lo que indica que el modelo está optimizado y que cada uno de los escenarios cumple con los respectivos atributos de calidad. Una vez concluido el proceso, el modelo resultante "arquitectura sugerida" se presenta en la Figura 153.

Figura 152. Resultado Final Evaluación del sistema Robótico

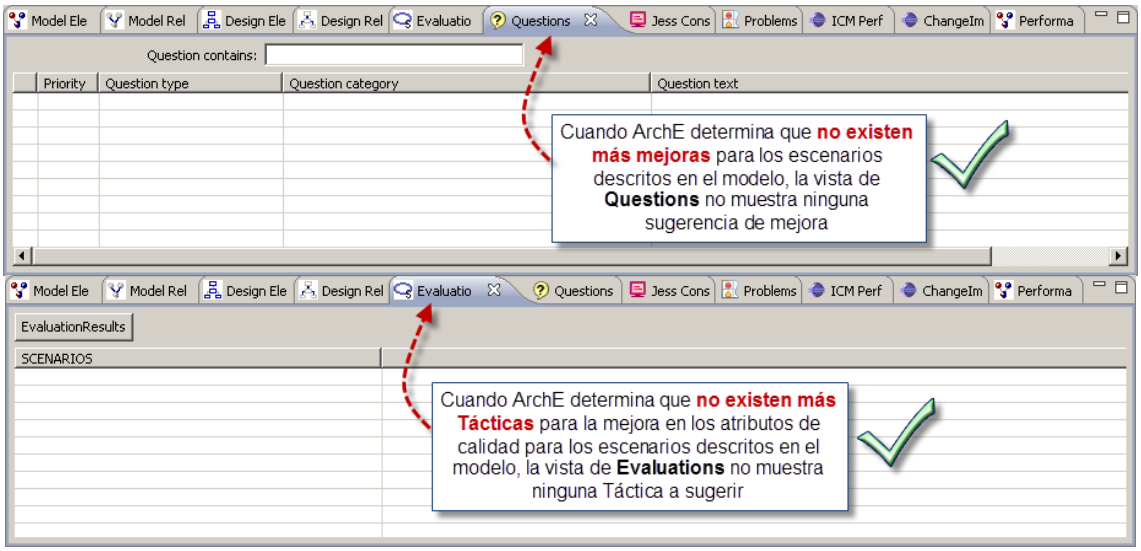
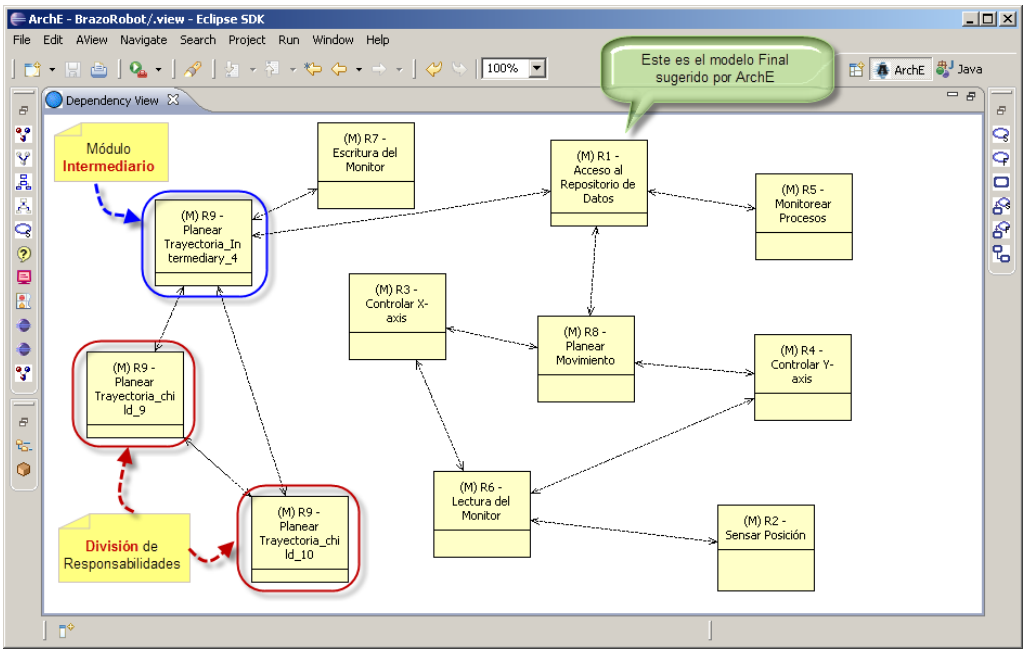


Figura 153. Modelo Final Sugerido por ArchE en el sistema Robótico



11. CONCLUSIONES

- La realización del presente trabajo de investigación se ha convertido en una experiencia enriquecedora para el autor, ya que con esta se ha logrado concatenar ideas en torno a un tema de interés específico. Antes de iniciar con el proceso de tesis, a título personal me sentí como un analfabeta más pese a que en mi experiencia como ingeniero y docente incursiono en la mayoría de aspectos cubiertos por la arquitectura de software, pero nunca había explorado con lujo de detalles como lo he hecho en este y otros documentos dispuestos en las asignaturas del Máster; por ello considero esta experiencia como el fruto de una juiciosa labor investigativa y la construcción de conocimientos significativos.

- La Arquitectura de Software es la "puerta de entrada" para lograr un buen diseño a partir de un conjunto de atributos de calidad del producto. La disciplina se enfoca en "Decisiones Tempranas", las cuales ayudan a diseñadores y programadores a concebir modelos escalables con un alto grado de reutilización, lo que conlleva a sistemas más flexibles, perdurables y con gran facilidad de mantenimiento. Al soportar su capacidad en el poder de la abstracción y la reutilización, se pueden incorporar un sinnúmero de componentes e incluso Frameworks no solo para validar nuevos diseños sino que también sirven para actualizar y dar soporte a los existentes sin que los cambios supongan grandes retos e inversiones a nivel de tecnología y de algoritmos de programación.

- Es indiscutible que una evaluación consciente de la arquitectura, conduce a una comprensión más clara de un sistema con base en los requerimientos, estrategias y valoración de riesgos antes y después de la implementación.

- Los atributos de calidad y la adopción de medidas tendientes a la mejora del producto, cada vez se convierte en un componente fundamental para la mejora y competitividad del negocio. La adopción de estándares relacionados con IT y el involucramiento de técnicas propias de la Arquitectura de Software no solo ayudan a incrementar la productividad y la mejora de los procesos operativos y el producto software como tal, sino que también buscan la plena satisfacción del cliente/usuario, con modelos cada vez más flexibles y reutilizables, adecuados a las necesidades propias del entorno.

- A pesar de que existe gran cantidad de estándares y atributos para la medición y evaluación de la calidad del software, no existe una medida universal que permita determinar con exactitud el comportamiento de un sistema ya que la aplicabilidad de una o varias estrategias están sujetas indiscutiblemente al entorno donde se desarrolla el sistema; por ello ha sido complejo determinar cómo y de qué manera se puede medir verdaderamente la calidad de un producto, de ahí que una constante mejora continua, debe estar implícita en cada uno de las fases de desarrollo y en la misma implementación para verificar y cuantificar el impacto y la calidad del producto de la mano de los stakeholders o involucrados.

- A título personal considero que un buen diseño no responde a que tan hábil se es en un determinado lenguaje de programación o que tan experto se es en la aplicación de un determinado estilo arquitectónico. El éxito de un modelo o sistema depende más de la especificación de sus atributos y requerimientos y la inclusión de éstos en los respectivos escenarios; desde los cuales el arquitecto ataca y planifica los comportamientos y toma las decisiones arquitecturales correctas.

- Los patrones (estilos) arquitectónicos se convierten en la base para el desarrollo de diversas arquitecturas a partir de la definición de sus propiedades y requerimientos. Como se describe en el documento, existen diversos estilos que se emplean dependiendo del grado de abstracción o entorno donde se determine el desarrollo. No todos los diseños incluyen a uno de ellos en específico, sino que su utilización depende de que tan complejo sea el desarrollo del componente software y como en base a dichas características, se pueden articular dos o más estilos para producir una salida satisfactoria.

- ArchE es una herramienta que permite la exploración de una amplia gama de soluciones para la optimización de modelos de software a partir de un conjunto de tácticas para soportar con mayor eficiencia las tareas de diseño. Desde la perspectiva académica, ArchE se constituye en una potente herramienta para mejorar los procesos de diseño ya que a partir de ella se incorporan características propias de la metodología orientada a objetos y el uso de patrones. ArchE incorpora mecanismos de solución optimizando los modelos con base en costos, utilidad, cohesión y probabilidades proveídos por el razonamiento de cada uno de sus marcos.

- Si bien es cierto que se pueden agregar nuevos marcos para evaluar más atributos y extender el árbol de utilidad de la arquitectura propuesta, actualmente no se evidencian nuevas propuestas a nivel académico o de la industria del software que permitan incrementar las prestaciones y capacidad de ArchE, lo que lo hace prácticamente inutilizable en escenarios a nivel de la industria del software y su estudio se limite más al entorno académico. Basado en trabajos anteriores (Fernández, 2010), también considero que el acceso restringido a los marcos de razonamiento, sus reglas y algoritmos no permite que se propongan nuevas especificaciones, reglas o mejoras a las bases de conocimiento y hechos del sistema experto.

- A pesar de que ArchE cuenta con una interfaz gráfica para modelado UML (Casos de Uso) a partir del Graphical Editing Framework (*GEF*), sería excelente, que la definición de este proceso de modelado, pudiera ser traducido de forma automática a la vista de Relationship o Relaciones (Reacción, Dependencia) que hace parte del conjunto de requerimientos de ArchE, al fin y al cabo es a partir de aquí donde se consideran las dependencias funcionales de la arquitectura.

- Una limitante para el diseño y aplicación de Tácticas está en el hecho de que ArchE solo realiza el análisis en base a 2 Frameworks de Razonamiento para igual número

de atributos de calidad; en este caso la **Modificabilidad** y el **Rendimiento**, y su aplicación se reviste de cierta irreversibilidad. Pienso a título personal que esta debió ser una funcionalidad contemplada por ArchE ya que nos permitiría explorar con mayor flexibilidad los cambios tras una y otra Táctica y analizar que tan beneficiosa es una de otra. Este es un proceso tedioso ya que se computan nuevamente muchas reglas "satisfechas o no" y es difícil tomar decisiones ya que los modelos generados pueden resultar de una u otra forma inmanejables. Todo dependerá de la perspectiva y experiencia del humano.

BIBLIOGRAFÍA

ACM. (2006). *Computing Degrees & Careers*. (ACM, Editor) Recuperado el 06 de Enero de 2013, de ACM - Association for Computing Machinery: http://computingcareers.acm.org/?page_id=12

American Psychological Association. (2009). *Publication Manual of the American Psychological Association.* Washington, DC.: 6th ed.

Arboleda, H., Casallas, R., & Royer, J. C. (2007). Implementing an MDA approach for managing variability in product line construction using the GMF and GME frameworks. *5th Nordic Workshop on Model Driven Software Engineering*, 67-82.

Bachman, F., Bass, L., & Nord, R. (2007). *Modifiability Tactics.* Carnegie Mellon University, Software Engineering Institute - SEI. Piitsburgh - PA: SEI.

Bachmann, F., Bass, L., & Klein, M. (2003). *Preliminary Design of ArchE: A Software Architecture Design Assistant.* Carnegie Mellon University. Pittsburgh - PA: Software Engineering Institute.

Bachmann, F., Bass, L., Bianco, P., & Klein, M. (2007). *Using ArchE in the Classroom: One Experience.* Carnegie Mellon University. Pittsburgh - PA: Software Engineering Institute.

Bachmann, F., Bass, L., Chastek, G., Donohue, P., & Peruzzi, F. (2000). *The Architecture Based Design Method.* Technical Report, Carnegie Mellon University, Software Engieneering Institute - SEI, Pittsburgh - PA.

Barbacci, M., Ellison, R., Lattanze, A., Stafford, J., Weinstock, C., & Wood, W. (2003). *Quality Attribute Workshops (QAWs), Third Edition.* Technical Report, Carnegie Mellon University, Software Engineering Institute - SEI, Pittsburgh - PA.

Bass, L., Ivers, J., Klein, M., & Merson, P. (2005). *Reasoning Frameworks.* Technical Report, Carnegie Mellon University, Software Engineering Institute - SEI, Pittsburgh - PA.

Bass, Len; Clements, Paul; Kazman, Rick. (2003). *Software Architecture in Practice* (Second Edition ed.). Boston - MA, USA: Addison Wesley.

Bishop, T., & Ramesh, K. (C de 2003). *A Survey of Middleware.* Towson University, Computer & Information Science Department, Towson - Maryland.

Blaha, M., & Rumbaugh, J. (2004). *Object-Oriented Modeling and Design with UML* (2nd Ed ed.). Prentice Hall.

Boehm, B. W., Brown, J. R., Kaspar, H., & Lipow, M. (1978). *Characteristics of software quality.*

Boehm, B., & Port, D. (1999). *Conceptual Modeling Challenges for Model-Based Architecting and Software Engineering (MBASE)* (Vol. 1565). Berlin, Alemania: Springer-Verlag Berlin Heidelberg.

Booch, G., Rambaugh, J., & Jacobson, I. (1999). *Guia de Usuario para el Lenguaje de Modelado UML*. México: Addison-Wesle.

Brown, A., Carney, D., Clements, P., Meyers, C., Smith, D., Weiderman, N., & Wood, W. (1995). Assessing the Quality of Large, Software-Intensive Systems: A Case Study. *European Conference on Software Engineering*, (pág. 7p). Madrid - España.

Buschmann, F., Meunier, R., Rohnert, H., Sommerland, P., & Stal, M. (1996). *Pattern-Oriented Software Architecture* (Vol. Vol. 1). Chichester, Inglaterra: John Willey & Sons.

Callaos, N., & Callaos, B. (1996). *"Designing with Systemic Total Quality", in International Conference on Information Systems.* Orlando, Florida: International Institute of Informatics and Systemics.

Carleton, A. D., & Florac, W. A. (1999). *Measuring the Software Process. Statistical Process Control for Software Process Improvement. SEI Series in Software Engineering.* Addison Wesley.

Champagne, R., & Gagné, S. (2011). *Towards automation of architectural tactics application – an example with ArchE.* Technical Report, ÉTS (University of Québec), Dept. of Software and IT Engineering, Burlingame - California.

Clements, P., & Nortrhop, L. (1996). *"Software architecture: An executive overview.* Technical Report, Carnegie Mellon Institute, Software Engineering Institute - SEI, Pittsburgh - PA.

Clements, P., Bachmann, F., Bass, L., Garlan, D., James, I., Reed, L., . . . Stafford, J. (2003). *Documenting Software Architectures: Views and Beyond.* Boston - MA: Addison-Wesley.

Clements, Paul. (2000). *Active Reviews for Intermediate Designs - Architecture Tradeoff Analysis.* Technical Note, Carnegie Mellon University, Software Engineering Institute - SEI, Pittsburgh - PA.

CMS. (Marzo de 2008). *Selecting a Development Approach.* Recuperado el 20 de Marzo de 2013, de Center for Medicare & Medicaid Services: https://www.cms.gov/Research-Statistics-Data-and-Systems/CMS-Information-Technology/XLC/Downloads/SelectingDevelopmentApproach.pdf

DeRemer, F., & Kron, H. (1976). Programming-in-the-large versus programming-in-the-small. *IEEE - Transaction in Software Engineering*, 80-86p.

Dewayne, P., & Wolf, A. (Octubre de 1992). Foundations for the study of software architecture. *ACM SIGSOFT - Software Engineering Notes*(17(4)), 40-52.

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

Diaz-Pace, A., Kim, H., Bass, L., Bianco, P., & Bachman, F. (2008). Integrating Quality-attribute Reasoning Frameworks in the ArchE Design Assistant. *Springer SVM*, 6-7p.

Eijogu, L. O. (1988). *A unified theory of software metrics.* ACM Conference on Computer Science.

Fenton, N. E., & Pflegger, S. L. (2001). *Software Metrics: A Rigorous & Practical Approach.* Boston, Mass: PWS Publishng Company.

Fernández C., O. M., & García L., D. (2013). *Un enfoque actual sobre la calidad del software.* ACIMED. La Habana: Ciencias Médicas.

Fernández, J. M. (2010). *Arquitecturas Software: Gestión de los atributos de calidad de un sistema y su incorporación en ArchE para la mejora del análisis, evaluación y soporte en la toma de decisiones durante el desarrollo arquitectónico.* Trabajo de Investigación, Universidad Nacional de Educación a Distancia - UNED, Escuela Técnica Superior de Ingeniería Informática - Ingeniería de Software y Sistemas Informáticos, Madrid - España.

Fielding, R. T. (2000). *Architectural Styles and he Design of Network-based Software Architectures.* Tesis Doctoral, Universidad de California, Irvine - CA.

Filman, R., Tzilla, E., Mehmet, A., & Siobhan, C. (2004). *Aspect-Oriented Software Development.* New York: Addison Wesley Professional.

Gamma, E. (1991). *Object-Oriented Software Development based on ET++: Design Patterns, Class Library Tools.* Tesis Doctoral, Universidad de Zurich, Institut für Informatik, Zurich - Suiza.

Gamma, E., Helm, R., & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software.* Boston - MA.: Addison Wesley.

Garbajosa, J. e. (2008). *Informe de vigilancia tecnológica Madrid. Tecnologías Orientadas a Servicios.* (Vol. ISBN:9788461268344). Madrid: CEIM.

Gil, R. H. (2007). *Metodología de Desarrollo de Software Basada en el Paradigma Generativo. Realización Mediante la Transformación de Ejemplares.* Universidad

Nacional De Educación A Distancia, Departamento de Ingeniería de Software y Sistemas Informáticos. Madrid - España: Escuela Técnica Superior de Ingeniería Informática.

Giraldo, G., Acevedo, J., & Moreno, D. (Diciembre de 2011). Una ontología para la representación de conceptos de diseño de software. (U. N. Colombia, Ed.) *Revista Avances en Sistemas e Informática, Vol. 8*(Num. 3), 10-110p.

Gonzáles Harbour, M., Gutiérrez, J., Palencia, J., & Drake, J. (2001). MAST: Modeling and Analysis Suite for Real-Time Applications. En IEEE (Ed.), *Proceedings of 13th Euromicro Conference on Real-Time Systems*, (págs. 125-124p). Netherlands: IEEE Computer Society Press.

Graham, I. (1994). *Metodología Orientada a Objetos* (2nd Ed ed.). Addison Wesley.

IEEE Computer Society. (2004). *Guide to the Software Engineering Body of Knowledge.* IEEE. Ney York, USA: SWEBOK executive editors.

IEEE Std729. (1983). *IEEE Standard Glossary of Software Engineering Terminology.* IEEE, Standars Coordinating Committe of the Computer Society of the IEEE, Ney York, USA.

Im, T. (2010). *A Reasoning Framework for Dependability in Software Architectures.* Tesis de Doctorado, Clemson University, South Carolina.

ISO. (1991). *"Information technology - Software product evaluation- Quality characteristics and guidelines for their use".* Montreal, Québec: JTC 1 Organization.

ISO 25000. (2005). *Software Engineering -- Software product Quality Requirements and Evaluation (SQuaRE).* Ginebra - Suiza: International Organization for Standarization.

ISO 8402. (1994). *Quality Management and Quality Assurance - Vocabulary.* Ginebra, Suiza: International Standards Organization.

ISO 9000. (2000). *Sistemas de Gestión de la Calidad- Fundamentos y Vocabularios.* Ginebra, Suiza: International Standization Organization.

ISO/IEC 15408-1. (2009). *Information technology - Security techniques - Evaluation criteria for IT security. Part 1: Introduction and general model.* International Organization for Standardization. Geneve - Switzerland: ISO.

ISO/IEC:1471. (2007). *ISO/IEC Standard for Systems and Software Engineering - Recommended Practice for Architectural Description of Software-Intensive Systems.* Std, IEEE, Institute of Electrical and Electronics Engineers, New York.

ISO12207. (2002). *ISO/IEC 12207:2002 / FDAM 2. Information Technology - Software Life Cycles.* International Organization for Standarization, Ginebra.

ISO15504. (1998). *ISO/IEC TR 15504:1998(E). Information Technology - Software Process Assestment. Parts 1-9.* Ginebra, Suiza: International Organization for Standarization.

ISO-9126. (1982). *SoftwareEngineering-Product Quality - Part 1: Quality Model.* Ginebra: Organización Internacional de Estándares.

Jazajery, M., Ran, A., & Linden, F. (2000). *Software Architecture for Product Families: Principles and Practice.* Addison Wesley.

Kazman, R., Bass, L., Abowd, G., & Webb, M. (2007). *SAAM: A Method for Analyzing the Properties of Software Architectures.* White Paper, Carnegie Mellon University, Software Engineering Institute - SEI, Pittsburgh - PA.

Kazman, R., Klein, M., & Clements, P. (2000). *ATAM: Method for Architecture Evaluation.* Technical Report, Carnegie Mellon University, Software Engineering Institute - SEI, Pittsburgh - PA.

Klein, M., & Kazman, R. (1999). *Attribute-Based Architectural Styles.* Carnegie Mellon University, Software Engineering Institute - SEI, Pittsburgh - PA.

Klein, M., & Kazman, R. (1999). *Attribute-Based Architectural Styles.* Technical Report, Carnegie Mellon University, Software Engineering Institute - SEI, Pittsburgh - PA.

Limón C., R. (2009). *Las vistas arquitectónicas de software y sus correspondencias mediante la gestión de modelos.* Universidad Politécnica de Valencia - UPV, Departamento de Sistemas Informáticos y Computación. Valencia - España: UPV.

McCall, J. A., & Richards, P. K. (1977). *Factors in software quality.* National Technology Information Service.

McGregor, J., Bachmann, F., Bass, L., Bianco, P., & Klein, M. (2007). *Using ArchE in the Classroom: One Experience.* Carnegie Mellon University. Pittsburgh - PA: Software Engineering Institute.

Medina D., F. (2010). *Repositorio Digital, Universidad Carlos III de Madrid.* Tesis Doctoral, Universidad Carlos III de Madrid, Departamento de Informática, Leganés. Recuperado el 12 de Enero de 2013, de e-archivo: http://e-archivo.uc3m.es/handle/10016/7433

Medina, J. L. (2005). *Metodología y Herramientas UML para el Modelado y Análisis de Sistemas de Tiempo Real Orientados a Objetos.* Universidad de Cantabria, Departamento de Electrónica y Computadores. UNICAN.

Mellor, S. J., Kendall, S., Uhl, A., & Weise, D. (2004). *MDA Distilled – Principles of Model-Driven Architecture.* Addison-Wesley.

Merlino, H., Vranic, A., Rodríguez, D., Pytel, P., & García-Martínez, R. (2009). *Ambientes de desarrollo de software basados en patrones de usabilidad.* Recuperado el 21 de Octubre de 2013, de Repositorio Institucional de la Universidad Nacional de La Plata: http://sedici.unlp.edu.ar/bitstream/handle/10915/19554/Documento_completo.pdf?sequence=1

Microsoft. (2013). *MSDN Library.* Recuperado el 269 de Septiembre de 2013, de Patrones y Antipatrones: http://msdn.microsoft.com/es-es/library/bb972251.aspx#M17

Miller, J., & Mukerji, J. (2003). *Tech. report, Object Management Group. MDA 1.0.1*.

Moreno, G., & Hansen, J. (2008). *Overview of the Lambda-* Performance Reasoning Frameworks.* Carnegie Mellon University, Software Engineering Institute - SEI. Pittsburgh, PA.: SEI.

OCGa. (2007). *Glosario de Términos ITIL®.* Londres - Inglaterra: Office of Government Commerce.

OMG:MDA. (2003). *MDA Guide Version 1.0.1.* Boston, MA.

OMG®. (2003). *Unified Modeling Language Specification.* Formal Specification, Object Managment Group - OMG, Needham - MA.

Osiatis. (2012). *ITIL - Gestión de Servicios TI*. Recuperado el 3 de Agosto de 2013, de Osiatis - The IT Service Experts: http://www.osiatis.es/

Parnas, D. (1972). On the Criteria for Decomposing Systems into Modules. *Communications of the ACM*(15(12)), 1053-1053p.

Parnas, David. (Marzo de 1976). On the Design and Development of Program Families. *IEEE - Transactions on Software Engineering*, 1-9p.

Parnas, David; Weiss, David. (1985). Active Design Reviews: Principles and Practices. (ACM, Ed.) *IEEE Computer Society Press*(1985), 132-136.

Pelayo García-Bustelo, B. C. (2007). *TALISMAN: Desarrollo ágil de Software con Arquitecturas Dirigidas por Modelos.* Tésis Doctoral, Universidad de Oviedo, Departamento de Informática, Oviedo - España.

Pérez-Campanero, J. A. (2010). *Desarrollo de Software y de las Arquitecturas Software.* Tésis de Máster - Trabajo de Investigación, Universidad Nacional de Educación a Distancia - UNED, Escuela Técnica Superior de Ingeniería Informática, Madrid - España.

Pressman, R. (2002). *Ingeniería del Software. Un enfoque práctico.* Madrid, España: McGraw-Hill Interamericana.

Quian, K., Fu, X., Tao, L., Wei Xu, C., & Diaz-Herrera, J. (2009). *Software Architecture and Design Illuminated* (1 Ed ed.). New York: Jones & Barlett Publishing Inc.

Sametinger, J. (1997). *Software Engineering with Reusable Components.* Berlin: Springer-Verlag.

Scott, J., & Kazman, R. (2009). *Realizing and Refining Architectural Tactics : Availability.* Carnegie Mellon University, Software Engineering Institute - SEI. Pittsburgh - PA: SEI.

SEI. (2006). *CMMI for Development, Version 1.2 CMU/SEI-2006-TR-008.* Pittsburgh - PA: Software Engineering Institute.

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

SEI. (2012). *Software Engineering Institute*. Recuperado el 10 de Mayo de 2013, de A Framework for Software Product Line Practice, Version 5.0: http://www.sei.cmu.edu/productlines/frame_report/index.html

Shaw, M., & Garlan, D. (1996). *Software Architecture: Perspectives on an emerging discipline.* Prentice Hall.

Sommerville, I. (2005). *Ingeniería de Software* (Séptima Ed. ed.). (P. Education, Ed., M. I. Alfonso Galpienso, A. Botía Martínez, F. Mora Lizán, & J. P. Tigueros Jover, Trads.) Madrird, Madrid, España: Addison Wesley.

Stevens, W., Myers, G., & Constantine, L. (1999). *Structured Design.* IBM. IBM Systems Journal.

Stoemer, C., O'Brien, L., & Verhoef, C. (2003). Practice Patterns for Architecture Reconstruction. *10th Working Conference on Reverse Engineering*, (págs. 46-56). Virginia.

Taylor, R., Medidovic, N., & Dashofy, E. (2010). *Software Architecture: Foundations, Theory, and Practice.* New Jersey: John Wiley & Sons, Inc.

TP-Link. (2013). *Qué es una DMZ.* Recuperado el 15 de Octubre de 2013, de TP-LINK Technologies Co: http://www.tp-link.es/article/?faqid=28

Van Der Linden, F., Schmid, K., & Rommes, E. (2007). *Software Product Lines in Action.* (P. I. Integra Software Services Pvt. Ltd., Ed.) New York, USA: Springer.

Wikipedia. (4 de Octubre de 2013). *Enciclopedia Social Wikipedia.* Recuperado el 03 de Agosto de 2013, de Tiempo de vida (informática): http://es.wikipedia.org/wiki/Tiempo_de_vida_%28inform%C3%A1tica%29

Wirth, N. (Abril de 1971). Program development by stepwise refinement. *Communications of the ACM,, 14(4),* 221-227p.

Wojcik, R., Bachmann, F., Bass, L., Clements, P., Merson, P., Nort, R., & Wood, B. (2006). *Attribute-Driven Design (ADD) - Version 2.0.* Technical Report, Carnegie Mellon Institute, Software Engineering Institute - SEI, Pittsburgh - PA.

SIGLAS, ABREVIATURAS Y ACRÓNIMOS

A

ACM (**A**ssociation for **C**omputing **M**achinery)

Sociedad científica y educativa fundada en 1947 por Richard Hamming. La ACM es una autoridad en informática y educación de las ciencias aplicadas a la informática a nivel mundial; consta de 34 SIGs (Special Group of Interest); o grupos de interés que profundizan en temas que van desde el uso de metodologías, implementación de modelos matemáticos y algoritmos, sistemas operativos, entre otros; hasta computación móvil, Inteligencia Artificial, Telecomunicaciones, Electrónica, Métricas de Calidad, Ingeniería de Software, etc.

API (***A**pplication **P**rogramming **I**nterface*)

Es el conjunto de funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción. Habitualmente estos componentes se agrupan a nivel de bibliotecas ddl (por ejemplo en un sistema basado en Windows) y ensamblados o Assemblies.

Arquitectura

(Del lat. *architectūra*). **1.** f. Arte de proyectar y construir edificios. **2.** f. *Inform.* Estructura lógica y física de los componentes de un computador. En Ingeniería de Software, la Arquitectura tiene que ver con el diseño y la implementación de estructuras de software de alto nivel. Es el resultado de ensamblar un cierto número de elementos arquitectónicos de forma adecuada para satisfacer la mayor funcionalidad y requerimientos de desempeño de un sistema, así como requerimientos no funcionales, como la confiabilidad, escalabilidad, portabilidad, y disponibilidad (Kruchten, Philippe).

Ataques de Replay

El replay attack, ataque por reenvío o reproducción, básicamente consiste en capturar información (transmisión de datos) que viaja en una red predeterminada, y luego enviarla al destinatario original saturando el canal sin que el receptor se percate de ello. El ataque de replay utiliza un método sencillo de explotar un paquete o paquetes capturados, y reenviar ese tráfico de datos para producir resultados inesperados. Aunque el canal de comunicación esté protegido con encriptación y otros mecanismos de autenticación como firmas digitales, es posible realizar este tipo de ataque que fundamentalmente lo que hace es duplicar la información en algunas veces a escala exponencial, lo que hace que el sistema se sature y falle consecuentemente.

Ataque de Denegación de Servicios

En seguridad informática, un **ataque de denegación de servicios**, también llamado ataque **DoS** (***D**enial **o**f **S**ervice*), es un ataque a un sistema de computadoras o red

que causa que un servicio o recurso sea inaccesible a los usuarios autenticados. Normalmente provoca la pérdida de la conectividad de la red por el consumo del ancho de banda de la red de la víctima o sobrecarga de los recursos computacionales del sistema de la víctima. Normalmente se catalogan como ataques lógicos (software) y ataques tipo Flood "inundación". El primero consiste en enviar al equipo remoto una serie de datagramas mal construidos; son fáciles de evitar actualizando el software a versiones que corrijan dichos fallos o añadiendo reglas al Firewall para filtrar paquetes mal construidos. El segundo consisten en bombardear un sistema con un flujo continuo de tráfico que acaba por consumir todos los recursos del mismo, especialmente el Ancho de Banda de la red del sistema atacado.

Ataque de Día Cero

Corresponde a un tipo de ataque realizado contra un ordenador, a partir del cual se explotan ciertas vulnerabilidades, o agujeros de seguridad de algún sistema. Por lo general este tipo de fallos en seguridad (desconocidos por el productor del paquete de software), de detectan después de ser implementada la herramienta, razón por la cual el diseñador siempre tendrá que solventar dicho fallo a través de Parches o actualizaciones para corregir la vulnerabilidad.

Ataque por Fuerza Bruta

Es el procedimiento de cracking utilizado para obtener las claves o códigos necesarios para acceder a un sistema, sitio web, etc. consistente en utilizar una serie de caracteres que obedecen a patrones aleatorios, que una vez formados, corresponden a claves o a una definición de diccionarios de datos definidos. Este tipo de mecanismo prueba todas las combinaciones posibles hasta encontrar la combinación correcta. Para evitar ataques por fuerza bruta, es importante utilizar contraseñas lo más seguras posible (combinación de caracteres).

Ataques informáticos

Es un método por el cual un individuo, mediante un sistema informático, intenta tomar el control, desestabilizar o dañar otro sistema informático.

Atomic Commit Protocol (Transacciones)

Es un protocolo que se encarga de analizar el estado de todos los servidores involucrados y que proporcionan una serie de conexiones y transacciones en un sistema distribuido. Un tipo de vulnberación a un sistema consiste en atacar directamente a este protocolo para que el mismo no pueda controlar el flujo de transacciones en una red de datos.

Atributo

(Del lat. *attribūtum*). **1.** m. Cada una de las cualidades o propiedades de un ser. En las ciencias de la informática un Atributo corresponde a una propiedad del producto, que cuando es asociada con la calidad se relaciona con los elementos que considera el cliente para aceptar o rechazar el producto. Estos atributos se modelan a partir de escenarios, los cuales permiten realizar mediciones y realizar mejoras al diseño.

Assemblie

Un ensamblado es principalmente una biblioteca de código compilado qe se utiliza para instalar, versionar y gestionar la seguridad en un paquete de software. Se agrupan en 2 categorías; ensamblados de proceso (EXE) y bibliotecas de ensamblados (DLL). Un ensamblado puede consistir en uno o más archivos. Los archivos de código son llamados módulos. Un ensamblado puede contener más de un módulo de código y es posible utilizar diferentes lenguajes en los diferentes módulos para crear el ensamblado de .NET.

B

BI

Inteligencia empresarial, inteligencia de negocios o **BI** (***B**usiness **I**ntelligence*) es el conjunto de estrategias y herramientas enfocadas a la administración y creación de conocimiento mediante el análisis de datos existentes en una organización o empresa.

Bluejacking

El Bluejacking se basa en una técnica mediante la cual se envían mensajes dañinos de un dispositivo a otro a través de una red Bluetooth (teléfonos móviles, Laptops, PDAs, entre otros). Este tipo de mensajes por lo general se catalogan como virus y pueden ir desde el envío de un simple SMS, hasta el acceso a la agenda de contactos y el apagado automático del equipo.. Para evitar este tipo de ataque, siempre se aconseja que una vez concluida la sesión se desactive la misma.

Brokers

Un Broker o Intermediario es un mecanismo responsable de coordinar las comunicaciones entre componentes (peticiones, respuestas, transmisión y control de excepciones).

C

CASE

(**C**omputer **A**ided **S**oftware **E**ngineering, Ingeniería de Software Asistida por Computadora). Corresponde a un conjunto de herramientas o aplicaciones informáticas diseñadas para aumentar la productividad y agilizar el proceso de desarrollo de software. Este tipo de herramientas se utilizan en casi todos los aspectos a lo largo del ciclo de vida del software (procesos de diseño, cálculo de costos, implementación de código, generación de código automático, compilación automática, documentación, depuración, modelado, etc).

CIs: (Incidencias)

La Norma ISO/IEC 20000-1: 2005, define un incidente "*como cualquier evento que no es parte de la operación estándar de un servicio y que causa, o puede causar, una interrupción del servicio o una reducción en su calidad*". Un ***Incidente*** [Incident] (OCGa, 2007) (Operación del Servicio) es una Interrupción no planificada de un Servicio de TI o reducción en la Calidad de un Servicio de TI. También lo es el Fallo de un Elemento de Configuración que no ha impactado todavía en el Servicio.

Componente

Un componente en otras palabras corresponde a un módulo de bajo acoplamiento y alta cohesión que denota una abstracción simple (Booch, Rambaugh, & Jacobson, 1999). Un componente es entonces cualquier elemento o recurso (por ejemplo, módulos, códigos, diseños, requerimientos, modelos de negocio, documentación, etc.), que puede ser reutilizado para posteriores desarrollos.

CORBA

(**C**ommon **O**bject **R**equest **B**roker **A**rchitecture): Es un estándar promovido por la OMG "***O****bject* ***M****anagement* ***G****roup*" y constituye un mecanismo que permite reducir la complejidad de los sistemas informáticos, los costos y la flexibilidad de los componentes software en un entorno distribuido. Este estándar es ideal para gestionar la interoperabilidad entre sistemas heterogéneos bajo un lenguaje común entre nodos cliente y su respectivo servidor. El lenguaje predominante para este estándar es el XML.

Class Patches

Consiste en una actualización del archivo ejecutable de un programa. Un Patch puede incluir rutinas para reparación de bugs o errores de programación, brechas de seguridad, actualizaciones, globalización "definición de un idioma en un determinado software", entre otras. Existen otros tipos de parches maliciosos denominados Cracks, que violentan los códigos de instalación y hacen que una aplicación pase de modo Trial a una versión completa.

CLIPS

Es una herramienta que provee un entorno de desarrollo para la producción y ejecución de sistemas expertos. Como otros lenguajes para sistemas expertos, CLIPS trabaja con reglas y hechos.

Coordinated Checkpointed

Un Punto de Control no es más que la coordinación y sincronización de hilos de procesos. Este mecanismo se implementa a través del uso de un determinado protocolo el cual garantiza igualdad de condiciones y competencia en tiempo de ejecución. Los protocolos de tolerancia a fallos no coordinados permiten que, cada proceso de la aplicación paralela guarde su estado independientemente de los demás procesos.

Cracking

Responde a un tipo de conducta delictiva mediante la cual un individuo "cracker" usurpa, altera, modifica o elimina los datos de un sistema informático con el fin de obtener un beneficio. Por lo general esta técnica de pirateo se focaliza en la rutura de códigos y claves de acceso a sistemas propietarios tipo Trial, para convertirlos en sistemas totalmente operativos con todas las funcionalidades a partir de la aplicación de un determinado crack.

Cross-site scripting

XSS, (***Cross-site scripting)*** es una vulnerabilidad o brecha de seguridad a nivel informático diseñada para ambientes web. Consiste en inyectar código de tipo Script en las páginas web. Este tipo de vulnerabilidades es causada por la incorrecta validación de los datos en un formulario y los controles de entrada que son usados en cierta aplicación. XSS es un vector de ataque que puede ser utilizado para robar información delicada, secuestrar sesiones de usuario, y comprometer el navegador, afectando la integridad del sistema.

Cross Site Request Forgery

El **CSRF** (***Cross-Site Request Forgery*** o falsificación de petición en sitios cruzados) es un tipo de *exploit* malicioso de un sitio web en el que comandos no autorizados son transmitidos por un usuario en el cual el sitio web confía. Esta vulnerabilidad es conocida también por otros nombres como XSRF, enlace hostil, ataque de un click, cabalgamiento de sesión, y ataque automático. En términos generales, el CSFR es un método mediante el cual un usuario malintencionado intenta hacer que los datos de otros usuarios, contengan otro tipo de información o datos privados sin que éste último se de por enterado.

D

Desbordamiento de búffer

En seguridad informática y programación, un **desbordamiento de buffer** (b*uffer overflow* o *buffer overrun*) es un error de software que se produce cuando un programa no controla adecuadamente la cantidad de datos que se copian sobre un área de memoria reservada a tal efecto (buffer). Un desbordamiento de búffer ocurre cuando los datos que se escriben en un búffer corrompen aquellos datos en direcciones de memoria adyacentes a los destinados para el búffer, debido a una falta de validación de los datos de entrada.

E

Earliest Deadline First

EDF "primero el más urgente", es un algoritmo de planificación que otorga mayor prioridad a la tarea cuyo plazo absoluto (límite o tiempo de ejecución) se venza antes que otros procesos con mayores tasas de ejecución.

Encapsulación

Es una técnica utilizada por el paradigma Orientado a Objetos para definir un conjunto de datos y modos de acceso a su estructura a partir de un conjunto de especificaciones y parámetros que hacen que el acceso sea exclusivo solo a través de dicho mecanismo, garantizando la integridad de los datos que contiene un objeto.

Estrategia

(Del lat. *strategĭa,* y este del gr. στρατηγία). **1.** f. Arte de dirigir las operaciones militares. **2.** f. Arte, traza para dirigir un asunto. **3.** f. *Mat.* En un proceso regulable, conjunto de las reglas que aseguran una decisión óptima en cada momento. En Ingeniería de Software corresponde a un conjunto de Tácticas aplicadas para la mejora de escenarios de calidad en un producto software.

Exploits

Es un pequeño fragmento de código, secuencia de comandos o pieza de software utilizada para explotar la información contenida en un servicio o aplicativo aprovechando al máximo la brecha de seguridad para robar, hacer espionaje o para corromper códigos de programas en ejecución. Los *exploits* pueden tomar forma en distintos tipos de software, como por ejemplo *scripts*, virus informáticos o gusanos informáticos.

Exception Classes (Conjunto de clases para manejo global de excepciones),

Corresponde a una clase o conjunto de clases base que dan soporte a las demás excepciones programadas por el usuario. A este conjunto de clases abstractas se les conoce también como excepciones estándar y están presentes en todos los lenguajes de programación orientados a objetos y eventos.

F

Fact Base: (Base de Hechos)

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

FIFO

Primero en entrar, primero en salir (***F**irst **I**n, **F**irst **O**ut)*, es un concepto utilizado en informática para denotar un conjunto de elementos "Arrays" que contienen un determinado orden y que son atendidos conforme van llegando a la cola de procesos. Uno de los usos de las colas es la exploración o búsqueda "en anchura" en un árbol binario y para la gestión de descargas en redes P2P.

G

GUI

La interfaz gráfica de usuario, conocida también como **GUI** (**G***raphical* ***U****ser* ***I****nterface*) es un software que actúa de interfaz entre el usuario y la computadora. Está representado por un conjunto de vistas "imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz y los aplicativos de software".

H

Hacking

Consiste en la búsqueda permanente de conocimientos en todo lo relacionado con sistemas informáticos, sus mecanismos de seguridad, las vulnerabilidades de los mismos, la forma de aprovechar estas vulnerabilidades y los mecanismos para protegerse de aquellos que saben hacerlo. Al sujeto que continene dicha habilidad se le conoce como Hacker o Pirata Informático.

I

ITIL

Desarrollada a finales de 1980, la Biblioteca de Infraestructura de Tecnologías de la Información (**ITIL**) se ha convertido en el estándar mundial para la Gestión de Servicios Informáticos. (Osiatis, 2012)**. ITIL** se ha formulado como un modelo para la mejora de procesos y contiene en sí misma un conjunto de "***Mejores Prácticas***" para garantizar la calidad de los servicios de TI en una organización. Además de esto, ITIL permite la descripción detallada de procesos y servicios adaptando dichas prácticas a las necesidades concretas de la compañía mejorando la gestión de su plataforma tecnológica para aumentar la competitividad de la misma.

Ingeniería social

El término "**ingeniería social**" hace referencia al arte de manipular personas para eludir los sistemas de seguridad. Esta técnica consiste en obtener información de los usuarios por teléfono, correo electrónico, correo tradicional o contacto directo. Los atacantes de la ingeniería social usan la fuerza persuasiva y se aprovechan de la inocencia del usuario haciéndose pasar por un compañero de trabajo, un técnico o un administrador, etc.

L

LSP o SPL

El Instituto de Ingeniería de Software de la Universidad Carnegie Mellon (SEI, 2012), define a una SPL como: "*…un conjunto de sistemas de software que comparten un conjunto común y gestionado de características que satisfacen las necesidades específicas de un segmento de mercado particular o misión, y que son desarrolladas de forma prescrita a partir de un conjunto común de elementos clave…*". En este contexto, una SPL permite acotar el tiempo de producción de componentes software, con altos niveles de calidad y una alta capacidad de reutilización.

M

Malware

Denominado también software malicioso/malintencionado "**Mal**icious Soft**ware**", es un tipo de software que tiene como objetivo infiltrarse o dañar una computadora o Sistema de información sin el consentimiento de su propietario. Los tipos más conocidos de malware son los virus informáticos, los Troyanos y los Gusanos, que se clasifican de acuerdo a su forma de infección y propagación.

Man in the Middle

MitM, es una situación donde un atacante supervisa (generalmente mediante un rastreador de puertos) una comunicación entre dos partes y falsifica los intercambios para hacerse pasar por una de ellas. En este tipo de ataque se introduce un intermediario (herramienta maliciosa) entre la víctima y la fuente: una página de banca online o una cuenta de correo electrónico. Estos ataques son realmente efectivos y, a su vez, muy difíciles de detectar.

MDA

La especificación (MDA) "Model Driven Architecture" es una especialización del desarrollo dirigido por modelos que separa la lógica del negocio del software y las plataformas tecnológicas (Pelayo García-Bustelo, 2007). ***M****odel* ***D****riven* ***A****rchitecture* (MDA) (OMG:MDA, 2003) (Mellor, Kendall, Uhl, & Weise, 2004) es una aproximación definida por el *Object Management Group* (OMG), mediante la cual el diseño de los sistemas se orienta a modelos.

MDD

Es una aproximación para solucionar el problema asociado a la complejidad de cada plataforma tecnológica y la inhabilidad que experimentan los lenguajes de propósito general en aliviar esta complejidad (Miller & Mukerji, 2003). Un enfoque de Desarrollo Dirigido por Modelos (***Model-Driven Development***), establece que para el desarrollo de un componente software, solo habrá de diseñarse un modelo específico que corresponda a la abstracción de la realidad que se desea representar y a partir de este modelo se generarán los componentes, especificaciones y código

para diversas plataformas separando los detalles arquitectónicos y de implementación, permitiendo la generación de uno o varios modelos a partir de un mismo PIM (***Platform Independent Model***). Con la implementación de este enfoque se adiciona mayor flexibilidad, interoperabilidad, flexibilidad y reutilización de componentes en una línea de producto software, minimizando el tiempo de desarrollo y ahorrando de manera sustancial la producción de líneas de código. La implementación más conocida de MDD se denomina Model-Driven Architecture (MDA).

Métrica

Una **métrica** es cualquier medida o conjunto de medidas destinadas a conocer o estimar el tamaño u otra característica de un software o un sistema informático. La *medición "es* el proceso por el cual los números o símbolos son asignados a atributos o entidades en el mundo real (Fenton & Pflegger, 2001). Una *medida* "proporciona una indicación cuantitativa de extensión, cantidad, dimensiones, capacidad y tamaño de algunos atributos de un proceso o producto. (Pressman, 2002). En la ingeniería de software las métricas se orientan a la aplicación continua de mediciones basadas en técnicas para el proceso de desarrollo del software y sus productos suministrando información relevante para la mejorar del proceso y el producto. Las métricas de software proveen la información necesaria para la toma de decisiones técnicas.

Middleware

Es un software que asiste a una aplicación para interactuar o comunicarse con otras aplicaciones, software, redes, hardware y/o sistemas operativos. Éste simplifica el trabajo de los programadores en la compleja tarea de generar las conexiones que son necesarias en los sistemas distribuidos. De esta forma se provee una solución que mejora la calidad de servicio, seguridad, envío de mensajes, directorio de servicio, etc. (Bishop & Ramesh, 2003).

MOM

MOM (**M**essage **O**riented **M**iddleware) es un tipo de protocolo o servicio de software que utiliza los mensajes como método de integración y provee mecanismos para crear, manipular, almacenar y transmitir esos mensajes. Estos sistemas permiten que las aplicaciones intercambien información en forma de mensajes compuestos por cabeceras y datos. En un MOM las aplicaciones intercambian mensajes a través de canales virtuales denominados destinations. Cuando se envía un mensaje no se envía a una aplicación concreta sino a un determinado destination. Es decir, en un MOM los mensajes son enviados de forma asíncrona. Los mensajes suelen ser muy útiles para la actualización de información y para la sincronización y coordinación de procesos. Un MOM asegura que los mensajes son distribuidos adecuadamente entre las distintas aplicaciones y por lo general suele proporcionar otras funcionalidades importantes como Tolerancia a fallos, Transacciones y Escalabilidad.

O

OOPSLA – "Object-Oriented Programming, Systems, Languages & Applications"

OOPSLA es la conferencia internacional sobre programación, lenguajes y aplicaciones que es realizada cada año por la ACM (Association for Computing Machinery), la asociación más importante en el ámbito de la computación a nivel mundial.

OLAP

OLAP es el acrónimo en inglés de **procesamiento analítico en línea** (*On-Line Analytical Processing*). Es una solución utilizada en el campo de la llamada Inteligencia de negocios o BI (*Business Intelligence*) cuyo objetivo es agilizar la consulta de grandes cantidades de datos en un sistema transaccional.

Overhedad

Es el desperdicio de ancho de banda, causado por la información adicional (de control, de secuencia, etc.) que debe viajar además de los datos, en los paquetes de un medio de comunicación. El overhead afecta al Throughput (cantidad de datos por unidad de tiempo que se entregan, mediante de un medio físio o lógico, en un nodo de la red), en una conexión.

P

POO

La programación orientada a objetos o **POO** es un paradigma de programación que usa los objetos y sus interacciones para diseñar aplicaciones y artefactos de software.

PIM

El Modelo Independiente de Plataforma o "**P**latform **I**ndependent **M**odel" sirve como soporte para la generación de uno o varios modelos empleando lenguajes específicos de dominio o lenguajes de propósito general como Java, C#, Python, etc. La traducción entre PIM y los PSM a generar se realizan a través de herramientas automatizadas como QVT.

Prioridad

(Del lat. *prior, -ōris*, anterior). **1.** f. Anterioridad de algo respecto de otra cosa, en tiempo o en orden. **2.** f. Anterioridad o precedencia de algo respecto de otra cosa que depende o procede de ello. Refiere al orden de ejecución de una tarea con respecto a la importancia y preponderancia de un proceso determnado. Por lo general las prioridades en informática se asocian a algoritmos como Prioridades fijas, colas, FIFO, Round Robin entre otras técnicas para mejorar el tiempo de respuesta en los sistemas y mecanismos de procesamiento de la CPU o dispositivo.

Q

QoS

Se refiere a cierto nivel de desempeño que ofrece un sistema de cómputo. A mayor QoS, el flujo de información entre emisor y receptos se hace cada vez más eficiente y en tasas de tiempo mucho menores. Cuando el QoS es deficiente, por lo general el usuario o servicio debe esperar y prolongar su espera lo que dificulta la transmisión y ubicación de la información.

R

REST

Propone orientar toda la capacidad de la Arquitectura de Software al desarrollo de soluciones en el marco de las nuevas tecnologías de internet y los modelos orientados a servicios o SAS y recursos distribuidos.

Riesgo

(Del it. *risico* o *rischio,* y este del ár. clás. *rizq*, lo que depara la providencia). **1.** m. Contingencia o proximidad de un daño. **2.** m. Cada una de las contingencias que pueden ser objeto de un contrato de seguro. Riesgo se puede definir como aquella eventualidad que imposibilita el cumplimiento de un objetivo. De manera cuantitativa el riesgo es una medida de las posibilidades de incumplimiento o exceso del objetivo planteado. Así definido, un riesgo conlleva dos tipos de consecuencias: ganancias o pérdidas. La ISO define el ***riesgo tecnológico*** (TI /TEC TR 13335-1, 1996) como: "La probabilidad de que una amenaza se materialice, utilizando vulnerabilidad existentes de un activo o un grupo de activos, generándole perdidas o daños".

Round Robin

Es un método para seleccionar todos los elementos en un grupo de manera equitativa y en un orden racional, normalmente comenzando por el primer elemento de la lista hasta llegar al último. Round Robin es un algoritmo de planificación de procesos simple de implementar, dentro de un sistema operativo se asigna a cada proceso una porción de tiempo equitativa y ordenada, tratando a todos los procesos con la misma prioridad. En Sistemas operativos, la planificación Round Robin da un tiempo máximo de uso de CPU a cada proceso, pasado el cual es desalojado y retornado al estado de listo, la lista de procesos se planifica por FIFO, primero llegado, primero atendido

S

SOA

Es un concepto de arquitectura de software que define la utilización de servicios para dar soporte a los requisitos del negocio. Permite la creación de sistemas de información altamente escalables que reflejan el negocio de la organización, a su vez brinda una forma bien definida de exposición e invocación de servicios. SOA permite acercar las TI al negocio, disminuyendo el riesgo operacional y aumentando el grado de conformidad con Base en estándares. También incrementa la flexibilidad y minimiza los niveles de acoplamiento, facilitando la reutilización y reducción en la logística de mercadeo, acelerando la innovación y facilitando la integración. (Garbajosa, 2008).

Spoofing

En seguridad informática hace referencia al uso de técnicas de suplantación de identidad. La técnica más conocida de spoofing trabaja directamente sobre el protocolo TCP-IP generando tramas TCP/IP y direcciones IP falsificadas; la idea de este ataque es muy sencillo; Desde un punto de acceso remoto, un hacker simula la identidad de otra máquina de la red para conseguir acceso a recursos de un tercer sistema con el cual ha establecido cierta confianza basada en el nombre o la dirección IP del host suplantado.

Spyware

Es un software que recopila información de un ordenador y después transmite esta información a una entidad externa sin el conocimiento o el consentimiento del propietario de la información. Un spyware típico se auto instala en el sistema afectado de forma que se ejecuta cada vez que se pone en marcha el ordenador (utilizando CPU y memoria RAM, reduciendo la estabilidad del sistema), y funciona todo el tiempo, controlando el uso que se hace de Internet y mostrando anuncios relacionados "publicidad de sexo, juegos online y otro tipo de información molesta al usuario". La idea fundamental de este software espía es saturar los puertos de conexión y de alguna manera relentizar la navegación y el ancho de banda en una red de datos.

SQL injection

Es un método de infiltración de código intruso que se vale de una vulnerabilidad informática presente en una aplicación en el nivel de validación de las entradas para realizar consultas a una base de datos.

T

Táctica

El conjunto de atributos o reglas que definen el software en torno a la calidad se denomina **Táctica.** Una **Táctica** es la manera en que los diseñadores y arquitectos deciden organizar los componentes del sistema, así como asignarle características

específicas a cada uno de ellos para controlar sus comportamientos y salidas (posibles respuestas a eventos) para así soportar uno varios atributos de calidad específicos. (Bass, Len; Clements, Paul; Kazman, Rick, 2003).

Tradeoff

Implica una decisión en la cual se comprende totalmente las ventajas y desventajas de cada elección.

V

Vista

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

W

Wrappers (Envolturas)

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

WYSIWYG: (What You See Is What You Get).

Es el acrónimo de ***W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **G**et* (en español, "lo que ves es lo que obtienes"). Se aplica a los procesadores de texto y otros editores de texto con formato (como los editores de HTML) que permiten escribir un documento viendo directamente el resultado final.

X

XML

XML, siglas en inglés de *e**X**tensible **M**arkup **L**anguage* ('lenguaje de marcas extensible'), es un lenguaje de marcas desarrollado por el World Wide Web Consortium (W3C) utilizado para almacenar datos en forma legible.