DESARROLLO DE UN ENTORNO SOFTWARE, DE MODELAMIENTO Y SIMULACIÓN, POR UNA COMUNIDAD (I+D) GEOGRÁFICAMENTE DISTRIBUIDA

ING. EMILIANO LINCE MERCADO

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICO MECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS
MAESTRÍA EN INGENIERÍA ÁREA INFORMÁTICA Y CIENCIAS DE LA COMPUTACIÓN
2009

DESARROLLO DE UN ENTORNO SOFTWARE, DE MODELAMIENTO Y SIMULACIÓN, POR UNA COMUNIDAD (I+D) GEOGRÁFICAMENTE DISTRIBUIDA

ING. EMILIANO LINCE MERCADO

TRABAJO DE INVESTIGACIÓN PRESENTADO COMO REQUISITO PARA OPTAR EL TÍTULO DE MAGÍSTER EN INGENIERÍA ÁREA INFORMÁTICA Y CIENCIAS DE LA COMPUTACIÓN

DIRECTOR:
HUGO HERNANDO ANDRADE SOSA
MAGÍSTER EN INFORMÁTICA

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICO MECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS
MAESTRÍA EN INGENIERÍA ÁREA INFORMÁTICA Y CIENCIAS DE LA COMPUTACIÓN
2009

DEDICATORIA

A Dios, a mi esposa y a mi hijo

AGRADECIMIENTOS

A Dios por su compañía

A mi familia por su paciencia y comprensión en los momentos difíciles

Al profesor Hugo Hernando Andrade Sosa por permitirme ser su alumno

Al grupo de investigación SIMON y todos sus integrantes por el apoyo incondicional

Al Ingeniero Jorge Moreno por el conocimiento compartido y el tiempo dedicado

A mis amigos por la motivación

RESUMEN

TITULO: DESARROLLO DE UN ENTORNO SOFTWARE, DE MODELAMIENTO Y SIMULACIÓN, POR UNA COMUNIDAD (I+D) GEOGRÁFICAMENTE DISTRIBUIDA.*

AUTOR: EMILIANO DE JESÚS LINCE MERCADO **

PALABRAS CLAVE: Ingeniería del Software, Proceso Distribuido de Desarrollo de Software, Desarrollo de Software, Comunidades de I+D, Entornos de Modelado y Simulación, modelos Integrados, Dinámica de Sistemas.

DESCRIPCIÓN:

Esta tesis de ingeniería del software hace parte del macro proyecto “Plataforma software de modelado y simulación - proyecto nacional interuniversitario” liderado por el grupo SIMON de la Universidad Industrial de Santander. Este macro proyecto promueve el desarrollo de un entorno para el modelado y simulación de modelos, que integran mediante la Dinámica de Sistemas a modelos en otras representaciones matemáticas.

El desarrollo de este nuevo entorno, ofrece mecanismos que posibilitan la integración de herramientas matemáticas como la lógica difusa, las redes neuronales, entre otras, por medio de la Dinámica de Sistemas. Estas herramientas, debido a sus características y naturaleza, normalmente son trabajadas de forma independiente. Por lo tanto, la posibilidad de integración brinda nuevos escenarios de investigación y desarrollo del modelado y la simulación.

La presente investigación tiene como objetivo principal, la creación de la arquitectura software del entorno de modelado y simulación de modelos integrados, a partir de la obtención de los requerimientos arquitectónicos. Esta arquitectura debe permitir el desarrollo del entorno por una comunidad distribuida geográficamente, haciendo uso de una plataforma software que facilite la comunicación, la integración y la consolidación de la comunidad desarrolladora. Esta plataforma software se debe implementar instanciando el proceso de desarrollo distribuido Agile-DISOP.

.

---

* Tesis de Maestría

** Facultad de Ingenierías Físico-Mecánicas. Programa: Maestría en Ingeniería (Área Informática y Ciencias de la Computación). Director: Mag. Hugo Hernando Andrade Sosa.

SUMMARY

TITLE: SOFTWARE DEVELOPMENT ENVIRONMENT, MODELING AND SIMULATION, BY A COMMUNITY (R & D) GEOGRAPHICALLY DISTRIBUTED.*

AUTHOR: EMILIANO DE JESÚS LINCE MERCADO**

KEYWORDS: Software Engineering, Distributed Process For Software Development, Software Development, R & D Communities, Environments Modeling and Simulation, Integrated models, System Dynamics.

DESCRIPTION:

This thesis software engineering is part of the macro-project “Platform modeling and simulation software - interuniversity national project" led by the group SIMON Industrial University of Santander. This macro-project promotes the development of an environment for modeling and simulation models, including using the System Dynamics models in other mathematical representations.
The development of this new environment provides mechanisms that enable the integration of mathematical tools such as fuzzy logic, neural networks, among others, through the System Dynamic. These tools, owing to their nature, are usually worked independently. Therefore, the possibility of integration offers new scenarios for research and development of modeling and simulation.
This research has as main objective the creation of software architecture modeling and simulation environment for integrated models, from the collection of architectural requirements. This architecture should enable development of the environment for a geographically distributed community, using a software platform to facilitate communication, integration and consolidation of the developer community. This platform must be deployed software development process instantiating distributed Agile-DISOP.

---

* Master’s Thesis.

** Faculty of Physical and Mechanical Engineerings. Degree: Masters of Computer Science. Director: Mag. Hugo Hernando Andrade Sosa.

CONTENIDO

Pág

LISTA DE TABLAS

LISTA DE FIGURAS

Pág

LISTA DE ANEXOS

INTRODUCCIÓN

El Grupo de Investigación en Modelos y Simulación (SIMON) adscrito a la Escuela de Ingeniería de Sistemas e Informática (EISI) de la Universidad Industrial de Santander (UIS), ha realizado investigaciones y desarrollos relacionados con el modelado y la simulación de enfoque estructural, bajo el paradigma Dinámico Sistémico, motivado por la preocupación por la difusión y aplicación del modelado y la simulación de enfoque estructural y su acercamiento al lenguaje natural. La ingeniería de software ha jugado un papel preponderante en el desarrollo de herramientas y aplicaciones software que apoyan el proceso de divulgación, aprendizaje y socialización de la Dinámica de Sistemas (D.S), el Pensamiento Sistémico (P.S) y en general de los enfoques relacionados con el modelado y la simulación de realidades complejas, como soporte a la resolución de problemas, generación de conocimiento y toma de decisiones, entre otros. El rol de la ingeniería software es el de brindar las herramientas que permitan implementar y dinamizar nuevos campos y formas de aplicación del modelado de enfoque estructural.

El presente informe sintetiza los resultados de la investigación en el área de ingeniería del software titulada “Desarrollo de un entorno software, de modelamiento y simulación, por una comunidad (I+D) geográficamente distribuida”, la cual pertenece al macro proyecto “Plataforma software de modelado y simulación - proyecto nacional interuniversitario” liderado por el grupo SIMON. Esta tesis es continuidad, en el contexto de un proceso de Investigación Acción, de la tesis de maestría “Diseño de una arquitectura para un entorno de modelamiento-simulación y creación de un proceso para su desarrollo por una comunidad (I+D)”.

Esta investigación se focaliza en la obtención de una arquitectura software, que permita el desarrollo en comunidad de un entorno para el modelado y simulación de modelos que integran, mediante la D.S, a modelos en otras representaciones matemáticas. El desarrollo de este nuevo entorno, ofrece mecanismos técnicos que posibilitan la integración de herramientas matemáticas como la lógica difusa (LD) y las redes neuronales (RN), entre otras, a través de la D.S, que por sus características y naturaleza son trabajadas de forma independiente.

La Arquitectura Software de un programa o sistema es la estructura o estructuras del sistema, la cual comprende los elementos de software, las propiedades visibles desde el exterior de estos elementos y las relaciones entre estos (Bass, y otros, 2007). Fundados en esta definición, el capítulo 6 describe la arquitectura del Entorno Software de Modelado y Simulación de Modelos Integrados (ESMS-MI) en función de sus componentes y la relaciones entre estos, a partir del estudio de diversos patrones arquitectónicos disponibles en la literatura.

Igualmente, se presenta en este informe los resultados del desarrollo de los objetivos que apuntan a la creación de las condiciones mínimas necesarias para el desarrollo por una comunidad distribuida geográficamente, por medio de una plataforma software que facilite la comunicación, la integración y consolidación de la comunidad desarrolladora del entorno.

Este informe se estructura en doce capítulos, los cuales dan cuenta del proceso de desarrollo de la tesis y sus resultados de la siguiente manera:

El capítulo 1, Descripción del proyecto, presenta la situación problema que generó la presente investigación, así como los objetivos propuestos para aportar a la problemática presentada y dar respuestas las preguntas de investigación planteadas.

El capítulo 2, ANTECEDENTES, presenta una revisión de las investigaciones y desarrollos previos a la presente tesis. Igualmente, este capítulo permite contextualizarla en el marco de un macro- proyecto liderado por el grupo SIMON en el área de ingeniería de software en el modelado y la simulación.

El capítulo 3, FUNDAMENTOS TEÓRICOS, proporciona el soporte conceptual al trabajo propuesto y está representado en el estudio de las teorías relacionadas a la presente investigación, como son: entornos de modelado y simulación, desarrollo de software en comunidad y arquitectura de software.

El capítulo 4, TESIS DE MAESTRÍA "DISEÑO DE UNA ARQUITECTURA PARA UN ENTORNO DE MODELAMIENTO- SIMULACIÓN Y CREACIÓN DE UN PROCESO PARA SU DESARROLLO POR UNA COMUNIDAD (I+D)", presenta los resultados de la revisión al principal antecedente de la presente tesis, estableciendo el punto de partida para dar continuidad a dicha investigación.

El capítulo 5, INGENIERÍA DE REQUISITOS PARA EL ESMS-MI[1], continúa con la obtención de los requisitos funcionales y no funcionales del nuevo entorno. Este capítulo desarrolla, en conjunto con el capítulo 6, el primer objetivo específico de la presente tesis: "*Continuar con la especificación de los requerimientos funcionales y elaborar la arquitectura (diseño de alto nivel) del Entorno Software de Modelamiento y Simulación de Modelos Integrados (ESMS-MI), garantizando la integración entre la Dinámica de Sistemas y otras herramientas matemáticas*[2]*"*

El capítulo 6, Arquitectura del ESMS-MI, propone un diseño arquitectónico para el entorno, definiendo los componentes, sus responsabilidades y relaciones entre dichos componentes. Este capítulo desarrolla, en conjunto con el capítulo 5, el primer objetivo específico de la presente tesis.

[1] Entorno Software de Modelado y Simulación de Modelos Integrados
[2] Continuar el desarrollo realizado por la tesis "Diseño de una arquitectura para un entorno de modelamiento-simulación y creación de un proceso para su desarrollo por una comunidad (I+D)".

El capítulo 7, NÚCLEO DEL ENTORNO SOFTWARE DE MODELAMIENTO Y SIMULACIÓN DE MODELOS INTEGRADOS, presenta las acciones realizadas con el propósito de promover el desarrollo del núcleo del entorno a través de la dirección de proyectos de pregrado. Este capítulo desarrolla el segundo objetivo específico de la presente tesis: *"Promover el desarrollo del núcleo del Entorno Software de Modelamiento y Simulación de Modelos Integrados (ESMS-MI), a partir de la herramienta Evolución 4.0".*

El capítulo 8, Orientaciones para la documentación, prueba y evaluación de los desarrollos independientes del ESMS-MI, desarrolla el tercer objetivo específico de la presente tesis: *"Formular las especificaciones metodológicas que orienten la realización de la documentación, prueba y evaluación de los desarrollos independientes a cargo de la creación de los componentes del ESMS-MI*[3]*"*

El capítulo 9, PLATAFORMA SOFTWARE PARA LA GESTIÓN Y COMUNICACIÓN DE LA COMUNIDAD DESARROLLADORA DEL ESMS-MI, presenta un conjunto de herramientas software que apoyan el desarrollo del entorno en un ambiente distribuido. Este apartado desarrolla el cuarto objetivo específico: *"Instanciar y soportar mediante una plataforma software el proceso de desarrollo en comunidad Agile-DISOP*[4]*, con el propósito de promover la puesta en marcha de la comunidad (I+D) que desarrollará el ESMS-MI. Ésta metodología guiará la creación del ESMS-MI permitiendo el trabajo distribuido entre la comunidad*[5] *que lo desarrolle"*.

El capítulo 10, Estructura y dinámica de la comunidad desarrolladora del ESMS-MI, establece las condiciones mínimas para que la comunidad funcione como una red humana, como son definir los medios de comunicación, contar con una plataforma para la integración de la comunidad, la organización de la comunidad y otras relacionadas con el desarrollo de software. Así mismo, se establecen criterios de homogeneidad, relación con la comunidad usuaria, gestión de errores y nuevos requerimientos, entre otros, que garanticen el desarrollo continuado del entorno. Este capítulo desarrolla los dos últimos objetivos específicos de la presente tesis: *"Orientar el inicio de las actividades de una comunidad interesada en desarrollar sostenidamente, a partir del núcleo del ESMS-MI implementado, las características previstas en los requerimientos funcionales y el diseño lógico del entorno*" y *"Definir las orientaciones para un desarrollo continuado de la plataforma de modelado, así como para el sostenimiento y consolidación de la comunidad desarrolladora"*.

El capítulo 11, Conclusiones y recomendaciones generales, resume las conclusiones de la tesis y las recomendaciones para su continuidad. Éstas serán tomadas por otras

[3] A partir de las especificaciones desarrolladas en la tesis "Diseño de una arquitectura para un entorno de modelamiento-simulación y creación de un proceso para su desarrollo por una comunidad (I+D)".
[4] Metodología propuesta por la tesis "Diseño de una arquitectura para un entorno de modelamiento-simulación y creación de un proceso para su desarrollo por una comunidad (I+D)" (2)
[5] Los integrantes de la comunidad estarán distantes espacialmente.

investigaciones y desarrollos en el marco del proceso de investigación- Acción que orienta el desarrollo del macro proyecto “Plataforma software de modelado y simulación - proyecto nacional interuniversitario”.

Por último el capítulo 12, mediante un conjunto de anexos, da cuenta de la labores de difusión de los resultados de esta tesis y de otros documentos que amplían e ilustran algunos de los demás capítulos.

CAPÍTULO 1. DESCRIPCIÓN DEL PROYECTO

1 DESCRIPCIÓN DEL PROYECTO

1.1 PLANTEAMIENTO DEL PROBLEMA

Durante 15 años el grupo SIMON ha desarrollado herramientas para apoyar su labor investigativa alrededor del modelado y la simulación. Una de estas herramientas y la de mayor trayectoria es Evolución(Cuellar Yeneris, y otros, 2003) herramienta para el modelado y la simulación con Dinámica de Sistemas, la cual ha venido creciendo durante varios años para satisfacer las necesidades de la comunidad dinámico sistémica nacional e internacional. Investigadores en el área de Dinámica de Sistemas, incluyendo al grupo SIMON de la UIS, han realizado estudios sobre la viabilidad de integrar a la dinámica de sistemas con otras herramientas matemáticas como son la lógica Fuzzy, redes neuronales, algoritmos genéticos, etcétera(El modelado con Dinámica de Sistemas, en su integración con otras herramientas matemáticas, 2005)- Por otra parte, la experiencia en el uso de algunas características de Evolución exige módulos cada vez más sofisticados para la presentación de los resultados de simulación, análisis de sensibilidad de modelos, generador de reportes en múltiples formatos, ayuda automatizada, etc. Además, es aconsejable agregar nuevas características como las planteadas por Sterman bajo el titulo “desafíos que enfrenta el desarrollo futuro de la Dinámica de Sistemas” (Sterman, 2000).

Lo anterior permite afirmar que existe la necesidad de crear un “Entorno Software de Modelamiento y Simulación de Modelos Integrados (ESMS-MI)”(Moreno Chaustre, 2006) que responda a las exigencias plateadas. Para emprender ésta tarea se ha diseñado una estrategia de desarrollo distribuido y continuado, entre varios grupos de investigación de universidades del país, que harán parte de la comunidad usuaria-desarrolladora lo cual ofrece varias ventajas como son: la oportunidad o desarrollo en menor tiempo, distribuye la capacidad tecnológica necesaria, mayor eficiencia debido a que cada módulo seria desarrollado por especialistas en el área, entre otras. Por otra parte, el desarrollo distribuido acoge varias de las ventajas del modelo de desarrollo de software libre donde la comunidad desarrolladora de la herramienta es la misma comunidad usuaria de esta, lo que repercute en más fiabilidad y mayor funcionalidad.

La presente investigación es continuación de la tesis de maestría “Diseño de una arquitectura para un entorno de modelamiento-simulación y creación de un proceso para su desarrollo por una comunidad (I+D)” (Moreno Chaustre, 2006), la cual realizó un análisis de la arquitectura de Evolución y propuso la creación de un nuevo entorno, a ser desarrollado por una comunidad distribuida geográficamente. Como primer paso, esta tesis soluciona las deficiencias encontradas en la arquitectura actual, requisito para abrirla (la arquitectura) a la comunidad que desarrollará el entorno, como segundo paso, definió un proceso de desarrollo de software en comunidades geográficamente distantes.

El desarrollo distribuido implica actividades que deben ser abordadas por ésta investigación como lo son: la definición de objetivos y el alcance del producto a desarrollar en comunidad, un análisis, diseño e implementación de un núcleo que se constituya la base para el desarrollo de la herramienta, identificar la forma como deben comunicarse sus componentes, definir los sub-proyectos que ejecutaran los equipos geográficamente distantes y que integren la comunidad de desarrollo, además se hace necesario la gestión para la conformación y el sostenimiento de la comunidad desarrolladora;

1.2 PREGUNTA DE INVESTIGACIÓN

El desarrollo de ésta tesis debe dar respuesta a las siguientes preguntas de investigación:

¿Cómo debe ser la arquitectura de una plataforma software para el modelado estructural y simulación, centrada en DS, que facilite su desarrollo distribuido? ¿Cuáles deben ser los requerimientos y desarrollos básicos (análisis, diseño, implementación herramientas de soporte, etc) para garantizar el inicio de un desarrollo distribuido por una comunidad dispersa geográficamente?

¿Cuáles deben ser las orientaciones de diseño para el desarrollo de software de usuario final basados en la arquitectura definida?

1.3 OBJETIVOS

1.3.1 Objetivo General

Formular la arquitectura de un entorno software de modelamiento y simulación centrado en Dinámica de Sistemas, que permita la integración con otras herramientas matemáticas para la construcción y reconstrucción del conocimiento y proponer las bases para el desarrollo continuado del entorno, por una comunidad distribuida geográficamente.

1.3.2 Objetivos Específicos

- Continuar con la especificación de los requerimientos funcionales y elaborar la arquitectura (diseño de alto nivel) del Entorno Software de Modelamiento y Simulación de Modelos Integrados (ESMS-MI), garantizando la integración entre la Dinámica de Sistemas y otras herramientas matemáticas[6].
- Promover el desarrollo del núcleo del Entorno Software de Modelamiento y Simulación de Modelos Integrados (ESMS-MI), a partir de la herramienta Evolución 4.0.
- Formular las especificaciones metodológicas que orienten la realización de la documentación, prueba y evaluación de los desarrollos independientes a cargo de la creación de los componentes del ESMS-MI[7].
- Instanciar y soportar mediante una plataforma software el proceso de desarrollo en comunidad Agile-DISOP[8], con el propósito de promover la puesta en marcha de la comunidad (I+D) que desarrollará el ESMS-MI. Esta metodología guiará la creación del ESMS-MI permitiendo el trabajo distribuido entre la comunidad[9] que lo desarrolle.
- Orientar el inicio de las actividades de una comunidad interesada en desarrollar sostenidamente, a partir del núcleo del ESMS-MI implementado, las características previstas en los requerimientos funcionales y el diseño lógico del entorno.
- Definir las orientaciones para un desarrollo continuado de la plataforma de modelado, así como para el sostenimiento y consolidación de la comunidad desarrolladora.

[6] Continuar el desarrollo realizado por la tesis “Diseño de una arquitectura para un entorno de modelamiento-simulación y creación de un proceso para su desarrollo por una comunidad (I+D)”.
[7] A partir de las especificaciones desarrolladas en la tesis “Diseño de una arquitectura para un entorno de modelamiento-simulación y creación de un proceso para su desarrollo por una comunidad (I+D)”.
[8] Metodología propuesta por la tesis “Diseño de una arquitectura para un entorno de modelamiento-simulación y creación de un proceso para su desarrollo por una comunidad (I+D)” (Moreno Chaustre, 2006)
[9] Los integrantes de la comunidad estarán distantes espacialmente.

CAPÍTULO 2. ANTECEDENTES

2 ANTECEDENTES

2.1 INTRODUCCIÓN

Esta sección presenta una revisión de los trabajos realizados que anteceden a la realización de la presente investigación. Estos trabajos se pueden clasificar en dos grandes grupos, el primero relacionado con la línea de investigación "Ingeniería de software en el modelado y la simulación". El segundo alrededor de las experiencias relacionadas con la integración entre la Dinámica de Sistemas (D.S) y otras herramientas matemáticas.

2.2 INGENIERÍA DE SOFTWARE EN EL MODELADO Y LA SIMULACIÓN: RECORRIDO POR LA EXPERIENCIA DEL GRUPO SIMON.

Desde su creación el grupo de investigación en modelos y simulación (SIMON) ha estado en contacto con el desarrollo de software y con las investigaciones en el campo de la ingeniería software, en procura de apoyar la consolidación de la D.S y de su aplicación en las diferentes áreas: Educación, epidemiología, agroindustria, ingeniería ambiental, ingeniería química, organizaciones, entre otras.

Como plantea (Navas Garnica, 2006) para que la D.S pudiera consolidarse en el ámbito universitario debía construirse una herramienta para el modelado y la simulación con D.S, de libre uso académico y que liberara a los usuarios de cualquier labor diferente a la del uso directo de la D.S, principalmente de labores de programación. Esta labor tiene un antecedente, previo a la constitución del grupo, la construcción del software SDS (Software de Dinámica de Sistemas), el cual se desarrolló en lenguaje BASIC y su objetivo fue colaborarle al usuario en la solución numérica de las ecuaciones, teniendo el mismo usuario que codificarlas de manera directa; el archivo de resultados generado por SDS era llevado a una hoja electrónica para su graficación. Este software se construyó para apoyar a la tesis de maestría titulada “Dinámica de Sistemas aplicada a la simulación de algunos fenómenos de transporte” (Andrade Sosa, y otros, 1990).

El software SDS se convertiría en la semilla que germinaría en el software que hoy se conoce como Evolución. Su primera versión se crea en el año de 1994 (al constituirse el grupo SIMON) y a diferencia de su predecesor es codificado en el lenguaje Pascal. Evolución es desarrollado para apoyar varios proyectos de pregrado de ingeniería de sistemas que aplicaron D.S, entre estos (Sotaquirá, 1994), (Ulloa, y otros, 1994) y (Gelvez Pinto, y otros, 1994) y con la colaboración adicional del ingeniero Juan Carlos García Díaz, miembro egresado del grupo. Evolución 1.0 facilitaba al modelador digitar las ecuaciones, las ordenaba y ejecutaba su solución numérica y graficación de resultados. Aún el progreso, para el usuario era dispendioso digitar las ecuaciones, debía ser de una manera particular.

Hasta ese momento, Evolución había sido un desarrollo en paralelo, que tenía como propósito generar una herramienta de apoyo para la obtención del objetivo principal de las tesis mencionadas, el cual fue la aplicación del modelado y la simulación con D.S. En el año de 1995 se desarrolla la primera tesis en el área de la ingeniería de software, la cual tiene como objetivo principal el desarrollo de la segunda versión de la herramienta. Evolución 2.0 (Ardila Arango, y otros, 1995) facilitaba la construcción del diagrama flujo nivel y a partir de éste, se generaban las rutinas de código con las ecuaciones ordenadas y se desarrollaba la solución numérica de las ecuaciones y la graficación de los resultados.

El mismo año se desarrolla otro proyecto de pregrado en la línea de la ingeniería de software aplicada al modelado y la simulación: SIMUIS (Arias Blanco, y otros, 1995). Este proyecto tiene como objeto el desarrollo de una herramienta software para la representación, solución y análisis de sistemas dinámicos en la forma espacio-estado.

En el año de 1997 se realiza una revisión a la versión 2.0, de la cual se genera Evolución 2.0a desarrollada por Carlos Angarita y Milena González. Esta versión contenía las características básicas de los software de D.S que circulaban a nivel internacional en dicha época, pero carecía de algunas como los análisis de sensibilidad y un editor de diagramas influencias (Dyner, y otros, 2004).

En el intento de llevar la D.S a la educación y a otras áreas del conocimiento, los nuevos desarrollos del grupo SIMON demandaron la reutilización del código de Evolución en otras aplicaciones. A raíz de esto, se genera un motor de simulación que permite la operación de modelos realizados con Evolución 2.0a en otros entornos y aplicaciones, aporte del ingeniero Carlos Angarita.

Lo anterior, posibilitó el desarrollo de otras herramientas, aplicaciones y software educativos con el uso del modelado y la simulación. Es así como se desarrollaron los micromundos, software educativos para el aprendizaje de diversas áreas con el uso del modelado y la simulación, como por ejemplo, los Micromundos para el Aprendizaje de Ciencias de la naturaleza- MAC. Hasta el momento, los software MAC implementaban las rutinas y algoritmos para resolver el sistema numérico asociado a cada modelo, limitando de esta manera la inclusión de nuevos modelos para su simulación[10]. Los primeros MAC que incluyen el motor de simulación son MAC 6-7 (Navas Garnica, y otros, 2000) y Mac 8-9 (Dueñas Amaris, y otros, 2000), este motor recibía los modelos y escenarios desarrollados en Evolución 2.0a, lo que facilitó la inclusión de nuevos contenidos y simulaciones, brindando una mayor flexibilidad a los micromundos. Otros desarrollos siguieron en esta línea, enmarcados en el macro proyecto MAC 1-11, una revisión detalla de estos trabajos pueden ser consultada en (Navas Garnica, 2006).

[10] El primer MAC se desarrolló en el 2000 y se denominó MACMedia 1.0 (Villa Abaunza, y otros, 2000)

En el año de 1998 nace otra herramienta denominada HOMOS 1.0 (Duarte Mogotocoro, y otros, 1998), la cual permite realizar modelos y simulaciones basados en objetos y reglas, creándose un paralelo entre el modelado con D.S y el modelado basado en objetos y reglas (MBOR). Pese a que ambas formas de modelado están enmarcadas en lo que se denomina el modelado estructural (Ingenieria de sistemas -realidad virtual y aprendizaje-, 2002), las herramientas software se han desarrollado de forma separada. Es así como al igual que con Evolución se han desarrollado micromundos para el aprendizaje con el uso de MBOR, creándose así MICRHO 1.0 (Cabarcas Alvarez, y otros, 2000). Actualmente, MICRHO se encuentra en su segunda versión (Ortiz Ramos, y otros, 2008).

Para retomar el desarrollo de Evolución, el grupo SIMON plantea una estrategia de desarrollo por proyectos. Es así como en el año de 1997 se realiza un proyecto en colaboración con la escuela de diseño industrial de la UIS, para realizar el diseño de la interfaz gráfica de usuario de Evolución (Martínez Beltrán, y otros, 1997). Otro proyecto sería el encargado de realizar el análisis y diseño de la siguiente versión de Evolución (Torres Cantillo, y otros, 2000), integrando las orientaciones del diseño de la interfaz de usuario realizadas por (Martínez Beltrán, y otros, 1997). En el año 2003, fruto de tres tesis de pregrado se generó la versión 3.5 de la herramienta (Ardila Arango, y otros, 2000), (Cuellar Yeneris, y otros, 2003) y (Alfonso, y otros, 2003). Esta versión posee un componente para la creación de los diagramas de influencia y uno para el análisis de sensibilidad por variación de parámetros y escenarios. Además, su diseño es orientado a componentes, en procura de facilitar la reutilización de código en otros software, necesidad ya sentida durante el desarrollo de los MAC. Es así como se crea un Framework con componentes de Edición de diagramas flujo-nivel, motor de simulación y presentación de resultados, utilizado no sólo en los MAC, sino en otros software como (Gómez Prada, y otros, 2002) y (Santamaria Pabon, y otros, 2004). Otros ejemplos del uso y aplicaciones de los componentes pueden ser consultados en (Framework para el desarrollo de ambientes software de aprendizaje y toma de decisiones con modelos en Dinámica de Sistemas, 2009).

Evolución 3.5, permite crear o diseñar una interfaz de presentación de los resultados o animadores, la cual presenta los resultados de la simulación de forma diferente a las gráficas y tablas tradicionales de Evolución 2.0a, por ejemplo, imágenes en movimiento, textos con los valores de las variables, etc. (previamente se realizó un intento por desarrollar la interfaz de este tipo tanto para Evolución como para SIMUIS en (Pineda Gomez, y otros, 2000)). Además, esta interfaz permite la interacción del usuario con la simulación por medio de controles, por ejemplo iniciar y pausar una simulación o cambiar un valor de una variable del modelo. Lo anterior, facilitó el desarrollo de un nuevo tipo de MAC (Ospino Reales, y otros, 2006), (Vera, y otros, 2006) y (Cala Amaya, y otros, 2008) que permite cargar dicha interfaz desde un archivo de Evolución 3.5, posibilitando a

los usuarios finales el diseño de simuladores y posteriormente agregarlo al contenido del MAC.

El proyecto titulado “Componente de Sistema de Inferencia Difusa (FIS) para Evolución 3.5”, (Machado Mendoza, y otros, 2006) genera la cuarta versión de Evolución. Se desarrolló un componente que permite integrar la Lógica Difusa (L.D) en el proceso de construcción de un modelo de D.S, siendo posible definir elementos del modelo en términos de un FIS. (Integración de la dinámica de sistemas y la lógica difusa. Evolución con FIS, 2007). Adicionalmente, se corrigen algunos errores detectados en el uso de Evolución 3.5.

El uso de Evolución cada vez mayor por parte de la comunidad académica y de las instituciones educativas beneficiadas por el convenio Computadores Para Educar-Universidad Industrial de Santander, genera nuevos retos y necesidades, al tiempo que permite identificar deficiencias y errores en las funcionalidades ya disponibles en el software.

A raíz de las deficiencias encontradas en el software y la intención de realizar una nueva versión, desarrollada por una comunidad distante geográficamente (grupos de investigación de varias universidades del país), se plantea la tesis de maestría titulada “Diseño de una arquitectura para un entorno de modelamiento-simulación y creación de un proceso para su desarrollo por una comunidad (I+D)” (Moreno Chaustre, 2006). Uno de los objetivos de la tesis fue la evaluación detallada del diseño de Evolución, identificando deficiencias a nivel de diseño y documentación. En el Capítulo 4 “TESIS DE MAESTRÍA ” se presenta más en detalle los resultados de esta tesis.

Actualmente, el grupo SIMON desarrolla el mantenimiento de Evolución 4.0 (Hernandez Cuadrado, y otros, 2009). Este proyecto de pregrado toma la evaluación de software planteada por Moreno Chaustre, realiza nuevas pruebas para identificar errores y deficiencias que permitan depurar el software y generar una nueva versión. Con la realización de este proyecto se abarcan todas las etapas del ciclo de desarrollo de software incluyendo la etapa de mantenimiento[11].

Con el propósito de continuar con la investigación de la tesis Moreno Chaustre y contemplar las necesidades identificadas, se plantea la presente tesis de maestría, que define la arquitectura de un nuevo Entorno de Modelado y Simulación de Modelos Integrados o ESMS-MI y crea las condiciones para su desarrollo por una comunidad separada geográficamente.

2.2.1 Conclusiones

El desarrollo de software por proyectos (que corresponden a proyectos de pregrado y tesis de maestría) ha posibilitado la realización de herramientas de

[11] El estándar 1219-1998 de IEEE define el mantenimiento de software como “La modificación de un producto software después de haber sido entregado, con el fin de corregir defectos, mejorar el rendimiento u otros atributos, o adaptarlo a un cambio en el entorno”.

gran tamaño y complejidad, lográndose abarcar parte del ciclo de vida del software. El grupo SIMON ha planteado proyectos donde el único objetivo ha sido el análisis y diseño inicial de una herramienta, posteriormente, propone uno o varios para realizar la implementación de esa herramienta.

Una de las dificultades para la generación de software de calidad, es el poco tiempo destinado a **la evaluación y prueba del software** desarrollado. Es común encontrar, proyectos que centran sus esfuerzos en la implementación de la herramienta, objetivo central del proyecto de grado, minimizando el tiempo destinado a la evaluación y prueba del software.

Otra de las dificultades presentadas es **la falta de documentación** tanto de los productos como de los procesos de desarrollo. Es poca la cultura para la implementación de técnicas de la ingeniería de software para documentar los procesos (gestión del cambio, control de versiones, trazabilidad, etc.) y la documentación del software tanto a nivel de desarrollo (modelos de análisis, modelos de diseño, documentación de código, manuales de programador, etc.) como de usuario (Ayudas, manual de usuario, sitio web, etc.).

Son muy escasos los proyectos de grado en los cuales se realiza la evaluación y prueba de la herramienta y aún más, los que realizan las labores correspondientes al **mantenimiento del software**. Esto sucede porque las necesidades de mantenimiento, surgen pasado un tiempo de estar implantado el software, producto de los cambios en el dominio y al uso por parte de los usuarios, por consiguiente un solo proyecto de grado no cuenta con el tiempo suficiente para concluir esta fase del ciclo de vida del software.

También se hace necesario examinar aspectos organizativos relacionados con **el soporte a usuarios**, lo cual es un criterio que tiene una notable influencia en la decisión del usuario, al momento de elegir un software entre varias opciones.

La mayoría de los desarrollos están propensos a presentar las dificultades mencionadas, ya sea porque el desarrollo es experimental y su principal objetivo es implementar o por apoyar una investigación o por algún otro motivo. Si el objetivo es producir software útil por una comunidad (empresa, organización, sector, etc), se debe tener en cuenta el problema del acabado y de la calidad del mismo. Para esto, antes de emprender un proyecto hay que ser consciente del punto en el cual se encuentra la herramienta en el ciclo de vida del software, del alcance del proyecto y de la forma como se pretende dar continuidad al desarrollo del mismo.

Para superar estas dificultades o barreras, que se presentan no sólo en los desarrollos del grupo SIMON, sino en general en la Escuela de Ingeniería de Sistemas de la Universidad Industrial de Santander - UIS, hay que pensar en cómo debe ser la organización para afrontar las necesidades de calidad y acabado del software; en definir y establecer metodologías apropiadas para soportar el

desarrollo en este tipo de organizaciones y en crear una cultura para la formalización del conocimiento adquirido o generado durante el desarrollo de los proyectos y tesis, relacionado con la ingeniería de software.

2.3 INTEGRACIÓN ENTRE LA DINÁMICA DE SISTEMAS Y OTRAS HERRAMIENTAS MATEMÁTICAS

En procura por encontrar nuevas posibilidades de aplicación de la D.S y el interés por acercar el modelado y la simulación al lenguaje natural, el grupo SIMON ha realizado investigaciones acerca de la relación entre la D.S y otras herramientas matemáticas. A continuación, se presenta un recorrido por las experiencias realizadas en el grupo.

A principios del año 2001, integrantes del grupo SIMON hacen uso de la Lógica Difusa (LD) y de las Redes Neuronales (RN) para definir multiplicadores, no linealidades y variables exógenas en modelos con D.S. Este trabajo se formalizó en la tesis de pregrado titulada “Modelo conceptual generalizado de un sistema para soportar la toma de decisiones. Enfoque integrador entre la Dinámica de Sistemas y la Inteligencia Artificial” (Luque Y Guzman Saenz, y otros, 2003). En particular, esta tesis propone el uso de los (FIS) Sistema de Inferencia Difusa como recurso para modelar multiplicadores y no linealidades en general y sistemas neuro-difusos para definir una variable exógena.

Posteriormente, durante el desarrollo de las labores académicas del profesor Hugo Andrade, en la universidad del Magdalena se desarrolla un ejercicio de clase en el que se integra la LD con la D.S mediante el uso del software UNFUZZY 1.2 y Evolución 3.5, ilustrando esta integración con el modelado del impacto ambiental ocasionado por la siembra de cultivos de coca en la Sierra Nevada de Santa Marta. En el modelo se utiliza L.D para la elaboración de elementos de la D.S como los multiplicadores, igualmente para definir funciones en términos de FIS que operan en tiempo de ejecución o de simulación. Al pretender definir las variables del modelo se apreció que era compleja la definición analítica y cuantitativa, por esto, se utilizó herramientas de la L.D, creando estructuras FIS para representarlas. La tesis de pregrado (Luque Y Guzman Saenz, y otros, 2003) ya había determinado esta posibilidad para los casos en los cuales el FIS requería una sola entrada, ya que es posible determinar todos sus posibles valores de entrada y obtener la función tabulada con todos los valores de salida. Pero para los casos en que las entradas al FIS son dos o más y no es posible definir todos sus posibles valores, es necesario que el FIS opere en tiempo de simulación del fenómeno, definiendo en cada iteración de la simulación el valor de salida correspondiente al conjunto de valores de entrada. El FIS (en tiempo de ejecución o de simulación) del modelo es posible operarlo en Evolución 3.5 si se agrega a éste mediante DLLs. Para esto, se implementa una función en Evolución 3.5 que tiene como parámetros las variables de entrada del FIS, y ésta debe retornar el

valor de la variable de salida de la respectiva estructura. (Integrando Dinámica de Sistemas y lógica Fuzzy, en tiempo de modelado y de simulación, un ejercicio de clase, 2004).

Más tarde, se realiza un segundo proyecto de pregrado en procura de aplicar los lineamientos iniciales sobre el uso de la LD y de las RN con D.S (Rivera, y otros, 2005)[12] . Este trabajo identificó tres posibilidades en la relación de modelado entre D.S. y otros herramientas matemáticas, Apoyo, Complemento e Integración, de las cuales se desarrollaron y aplicaron dos (Modelo Apoyado y Modelo Integrado), además, se avanzó en la determinación de los tipos de situaciones susceptibles a ser tratadas con este enfoque; se formuló una propuesta metodológica concreta para modelar situaciones con cada una de las posibilidades de relación desarrolladas y se plantearon algunos requisitos software a ser tenidos en cuenta para el desarrollo de un módulo que permitiera la creación e inclusión de los FIS como elementos de los modelos de D.S, para su operación en tiempo de simulación (ejecución), dentro de la herramienta de modelado y simulación Evolución, sin recurrir a software externo ni labores de programación. (El modelado con Dinámica de Sistemas, en su integración con otras herramientas matemáticas, 2005).

En el año 2006 se desarrolla el proyecto (Machado Mendoza, y otros, 2006) el cual implementa un FIS como un componente software para Evolución 3.5. En este proyecto se contempló las experiencias desarrolladas previamente y los lineamientos definidos en la tesis (Rivera, y otros, 2005). Este componente permite agregar un nuevo elemento en el diagrama flujo-nivel, el cual brinda la posibilidad al modelador de relacionar la D.S y la L.D, de una manera casi que intuitiva, librándolo de la necesidad de realizar algún tipo de programación y de utilizar otra herramienta software para crear los FIS (Integración de la dinámica de sistemas y la lógica difusa. Evolución con FIS, 2007).

2.3.1 Conclusiones

En ocasiones es apropiado el uso de otras herramientas matemáticas en conjunto con la D.S porque brinda nuevas posibilidades de aplicación en diversos campos y además permite su uso por un mayor número de usuarios. Estas posibilidades se ven limitadas por restricciones de los entornos computacionales, por lo cual es fundamental el desarrollo de software acorde a nuestras necesidades e intereses investigativos, y su continua adecuación a la par del desarrollo mismo de la Dinámica de sistemas.

[12] Los resultados de esta tesis fueron publicados en dos ponencias (Integración entre la Dinámica de Sistemas y otras matemáticas para la representación del conocimiento, 2004) y (El modelado con Dinámica de Sistemas, en su integración con otras herramientas matemáticas, 2005)

CAPÍTULO 3. FUNDAMENTOS TEÓRICOS

3 FUNDAMENTOS TEÓRICOS

3.1 DINÁMICA DE SISTEMAS (D.S)

La D.S es un lenguaje que nos facilita explicar y recrear los fenómenos de interés en términos de modelos de simulación. Con estos modelos y el computador podemos observar cómo se puede comportar el fenómeno bajo diferentes condiciones (experimentación simulada). Es decir, podemos responder a la pregunta por: ¿Qué pasaría en …. (un fenómeno) si ….. (se presentan determinadas condiciones…)?

Según esto, un modelo es una explicación que nos es útil para contestarnos preguntas sobre el fenómeno que explica. Además, la D.S asume el Paradigma de Pensamiento Dinámico Sistémico para pensar los fenómenos a modelar. Este paradigma es una manera de pensar sobre lo que nos interesa asumiendo los fenómenos como Sistemas Dinámicos, es decir, como cosas que están en permanente cambio y que para comprenderlas debemos explicar cómo cambian como sistemas dinámicos, constituidos por un conjunto de partes interrelacionadas generando una estructura realimentada que determina su comportamiento.(Andrade Sosa, y otros, 2009).

3.1.1 Herramientas para el modelado y la simulación con D.S

Existen varias herramientas para el modelado y la simulación con D.S. Estas herramientas permiten hacer modelos utilizando los diagramas de Forrester o Flujo Nivel, el cual es un lenguaje que permite representar un modelo matemático, basado en un sistema de ecuaciones diferenciales, de forma gráfica; además, estas herramientas integran métodos numéricos que permiten hacer simulaciones y presentar los resultados a través de diferentes medios como gráficas, tablas y animaciones. Cada herramienta tiene características que la diferencian del resto, así como algunas permiten el manejo de vectores, otras permiten la interacción en tiempo de simulación[13], análisis de sensibilidad, etc. Las herramientas más conocidas en nuestro medio son: Dínamo, Powersim, Stella, IThink y Evolución.

La mayoría de estas herramientas son software propietario con versiones de demostración o de evaluación, con capacidades de uso limitada. Una de éstas, Evolución (Cuellar Yeneris, y otros, 2003) fue desarrollada por el grupo SIMON se encuentra disponible en idioma español y gratuitamente para uso académico e investigativo.

Evolución, actualmente en su versión 4.0, ha venido desarrollándose desde el año 1995 (ver Capítulo Antecedentes). Ofrece varias capacidades dentro de las que se encuentran: editores de diagrama de flujo-nivel, diagrama de influencias, módulo de análisis de sensibilidad, inclusión de un FIS en el modelo, vectores de elementos, presentación de resultados de forma animada, entre otras.

[13]Permitir cambiar el valor de un variable durante la simulación.

3.2 ENTORNOS DE MODELADO Y SIMULACIÓN DE MODELOS INTEGRADOS.

3.2.1 Concepto de modelo

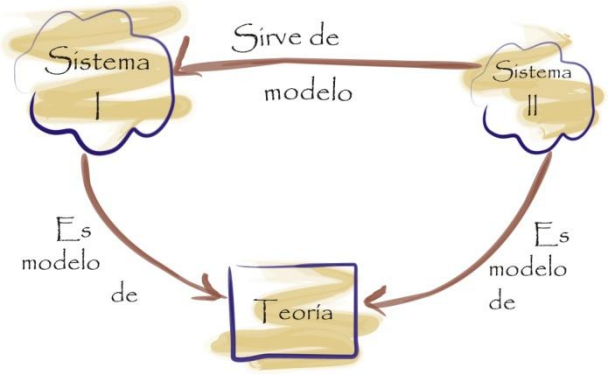
En el lenguaje ordinario, el uso del término "*modelo"* se emplea para referirse a lo pintado, esculpido, fotografiado, es decir, a lo representado (por ejemplo la *modelo* del fotógrafo). Pero en ocasiones el término "*modelo"* es empleado para referirse a lo opuesto, es decir, a la pintura, lo esculpido, la maqueta (por ejemplo el modelo o maqueta de un edificio). (Mosterín, 1987).

Para llegar al concepto de "modelo" es necesario precisar un conjunto de conceptos entre los que encontramos: S*istema*, *estructura*, *teoría*. Todos estos conceptos, incluido el de modelo, son reproducidos del análisis sobre los conceptos y teorías en la ciencia (Mosterín, 1987) realizada por el filósofo español Jesús Mosterín[14]. Estos conceptos serán referentes teóricos de la presente investigación. Por sistema se entiende: "Un conjunto de objetos bien definidos, acompañado de ciertas propiedades, posiciones e interrelaciones bien definidas entre ellos". La estructura se refiere a "ciertos rasgos más o menos formales comunes a varios sistemas". En cuanto a las teorías Mosterín afirma que "definir una estructura es lo mismo que formular su teoría. … En efecto, la matemática suele definirse como la ciencia de las estructuras. En este sentido, todas las teorías son matemática." "El estudio científico de un modelo aspira a elaborar una teoría del sistema, es decir, un conjunto de enunciados, ecuaciones, fórmulas, esquemas, etc. Que permitan describir adecuadamente el funcionamiento presente del sistema, así como explicar lo pasado y predecir lo que pasará en dicho sistema en el futuro" Todos los modelos de un teoría tienen en común una estructura la cual es caracterizada por esa teoría.

Un *modelo* no es una teoría, es a lo que se refiere la teoría, un sistema en el cual se cumple la teoría. En ocasiones se construye otro sistema más simple que *sirva de modelo* para el estudio de un sistema más complejo, en la medida que igualmente en éste se cumple la teoría. (Mosterín, 1987). Lo anterior se esquematiza en la **Figura 1**.

[14]Profesor de Investigación en el Instituto de Filosofía del Consejo Superior de Investigaciones Científicas de España y Catedrático de Lógica y Filosofía de la Ciencia en la Universidad de Barcelona. Miembro titular del Institut International de Philosophie (París), de la Academia Europaea (Londres) y de la International Academy of Philosophy of Science. Fellow del Center for Philosophy of Science – Pittsburgh. (Consejo Superior de Investigaciones Científicas)

Figura 1. Servir de modelo



Fuente: Autor

3.2.2 ¿Que son los modelos integrados?

Existen herramientas matemáticas para la representación de las realidades complejas a través de modelos de simulación, algunas de estas son: D.S, Lógica difusa - LD, Redes Neuronales - RN, Agentes, Análisis Estadístico, Investigación Operacional, Representación Espacio Estado, entre otras. Un modelo integrado es un macro modelo que integra modelos realizados con diferentes herramientas matemáticas. "Al buscar integración entre los útiles matemáticos de la D.S y aquellos correspondientes a otras matemáticas, como es el caso de L.D y las R.N, se llega a una interacción entre elementos que, debido a su diferenciación desde los fundamentos de cada uno, operan con distintos lenguajes formales y procedimientos para el modelado. En este sentido, se puede hablar del surgimiento de nuevas construcciones matemáticas que no corresponden del todo con las ecuaciones diferenciales lineales y no lineales empleadas por la D.S, pero que del mismo modo, no pueden ser asociadas completamente a las herramientas matemáticas de los demás medios para el modelado con que se integra." (El modelado con Dinámica de Sistemas, en su integración con otras herramientas matemáticas, 2005).

3.2.3 Entorno de Modelado y Simulación

El concepto de "entorno de modelado y simulación", combina varios aspectos del proceso relacionados con el desarrollo y la simulación de los modelos en una poderosa y completa colección de herramientas integradas. Conceptualmente, un entorno de modelado y simulación armoniza el lenguaje de simulación con las herramientas de soporte (A standard simulation environment: a review of preliminary requirements., 1994). Otra forma de concebir un entorno de modelado y simulación consiste en describirlo como aquellas partes del sistema que el usuario percibe (incluye el sentimiento psicológico del sistema basado en las bondades de su funcionalidad) más allá de la superficial interfaz de usuario hasta las herramientas de soporte al proceso de modelado y simulación. Más

específicamente, un ambiente es la integración sinérgica de módulos de software para proveer un esquema fuerte y cohesivo cuyo propósito consiste en formular y resolver un conjunto dado de tareas (An examination of environment design and the applicability of software engineering requirements to the standard simulation environment, 1994).

3.2.4 Requerimientos para un Entorno de Modelado y Simulación

El mundo real del modelado y la simulación, es una amalgama de diferentes entornos tratando de alcanzar los mismos objetivos. Aunque esos entornos proveen características similares, estos raramente utilizan el mismo lenguaje de modelado y más aún, obstáculos tales como el aprendizaje de un nuevo paradigma, modelos y formatos de datos incompatibles no permiten la explotación del trabajo previo mediante la reutilización de componentes ni el intercambio de modelos entre entornos. En consecuencia, definir requerimientos para un entorno de modelado y simulación debe ser un proceso que:

- Brinde soporte a las necesidades del usuario final, al mismo tiempo que minimiza la necesidad de aprender un nuevo lenguaje, mediante la especificación de características funcionales y no funcionales útiles
- Determine la correcta selección de la plataforma hardware/software y los lenguajes de desarrollo utilizados.
- Brinde soporte a la simultánea ejecución e interacción de múltiples modelos, permitiendo que modelos implementados en distintos lenguajes se integren en otros más grandes.
- Soporte la portabilidad de un modelo de simulación desde un sistema a otro, permitiendo que un modelo desarrollado por una persona (analista, docente, consultor o estudiante) u organización pueda ser re-utilizado por otras personas u organizaciones en otros sistemas.
- Brinde soporte al modelador en la adopción de un enfoque de múltiples perspectiva.

3.3 PROCESOS DE DESARROLLO DE SOFTWARE

Una metodología o proceso de desarrollo de software es el conjunto de actividades y estrategias que tienen como objetivo asegurar la calidad del software desarrollado.

La ingeniería de software dispone de un amplio abanico de modelos de procesos para el desarrollo de software, a continuación se presenta algunos de estos:

- Modelo en cascada
- Modelo por prototipos.
- Modelo Desarrollo Rápido de Aplicaciones (DRA)
- Modelo en espiral

- Desarrollo basado en componentes.
- Métodos formales.
- Proceso Unificado de Rational (RUP).
- Agile Unified Process (AUP).
- Extreme Programming (XP).

3.3.1 Metodologías ágiles

Dentro de la amplia gama de metodologías, un grupo de éstas son clasificadas como "ágiles". Martin Fowler (Fowler, 2000) plantea la diferencia entre las metodologías "monumentales"[15] y las ágiles, como la reducción de procesos que dilatan el desarrollo de software:

> Un nuevo grupo de metodologías ha surgido en los últimos años. Durante algún tiempo se conocían como las metodologías ligeras, pero el término aceptado ahora es metodologías ágiles. Para muchos el encanto de estas metodologías ágiles es su reacción a la burocracia de las metodologías monumentales. Estos nuevos métodos buscan un justo medio entre ningún proceso y demasiado proceso, proporcionando simplemente suficiente proceso para que el esfuerzo valga la pena.
>
> Algunas metodologías que encajan dentro de las ágiles son:
>
> - XP (Programación Extrema)
> - La Familia de Cristal de Cockburn
> - Código Abierto
> - El Desarrollo de Software Adaptable de Highsmith
> - Scrum
> - Desarrollo Manejado por Rasgos (FDD)
> - DSDM (Método de Desarrollo de Sistema Dinámico)
> - Proceso dX de Robert Martin

3.3.2 Desarrollo distribuido vs Desarrollo disperso

Con el auge de los desarrollos de código abierto y libre se ha popularizado el desarrollo en comunidad, donde el código fuente de un software se encuentra disponible para descargar. Un potencial desarrollador puede hacer una modificación o agregar alguna característica y si es aceptada por la comunidad desarrolladora se genera una nueva versión. Cualquier persona es virtualmente un desarrollador del software.

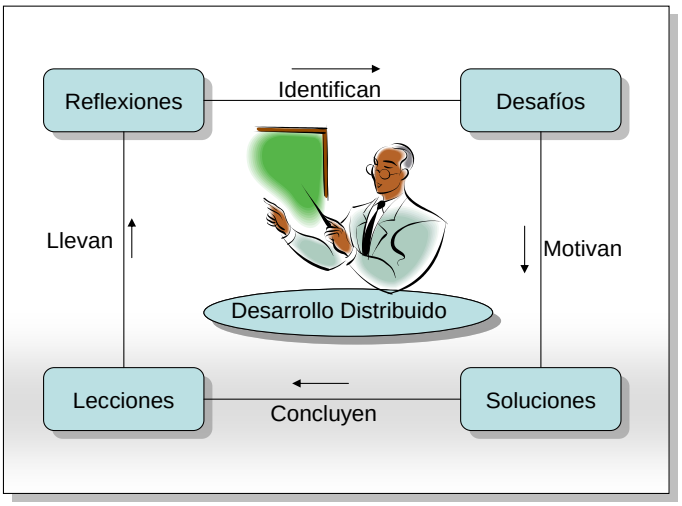
Es importante distinguir si el desarrollo en comunidad es distribuido o es disperso, ya que acarrean problemas diferentes. Los dos enfoques difieren principalmente en que en el desarrollo distribuido son *equipos* los que trabajan distribuidos

[15] Metodologías que son burocráticas. Hay demasiadas actividades para seguir la metodología que el ritmo global de desarrollo se ve afectado.

geográficamente, mientras que en el desarrollo disperso son *individuos* los que trabajan remotamente (Ambler, 2002).

3.3.3 Procesos de desarrollo distribuido

Figura 2. Modelo de Evolución del Pensamiento para procesos distribuidos



Fuente (Moreno Chaustre, 2006)

"El desarrollo distribuido establece desafíos significativos relativos a la cultura organizacional y la filosofía de la gestión, las arquitecturas de software, la comunicación, satisfacción de los clientes, los procedimientos de integración y prueba y muchas otras áreas." Dice Lockheed Martin Fred Palmer, quien se ha desempeñado como moderador del Open Forum Session durante el "The Executive Round Table. La evolución del concepto del desarrollo distribuido (ver **Figura 2. Modelo de Evolución del Pensamiento para procesos distribuidos**), tiene un ciclo de vida que inicia con las reflexiones suscitadas por la necesidad de superar con éxito los esfuerzos de desarrollo que no imponen restricciones respecto a la dispersión geográfica de todos los implicados. Estas reflexiones, se agrupan en tres categorías a saber: La gente, los Procesos y la Tecnología. A este respecto, se presenta a continuación, la revisión de algunos planteamientos y experiencias sobre los procesos de desarrollo distribuidos, basado en el Software Productivity Consortium(Systems and software consortium, 2009):

- **La Gente.** La postura más relevante identificada para el desarrollo geográficamente distribuido es la comunicación. Dada la necesidad de crear una visión compartida del producto y proceso software en un ambiente de desarrollo distribuido, el valor de la comunicación cara a cara

es vital, particularmente cuando se presenta un diseño complejo que represente una significativa dificultad técnica.

- **El Proceso.** La necesidad de procesos maduros en común a través de los proyectos de desarrollo geográficamente distribuidos es uno de los desafíos identificados más difíciles de superar. Sin embargo, el simple hecho de tener procesos en común no es suficiente, el verdadero reto de la gestión consiste en inculcar y hacer respetar rigurosamente esos procesos comunes entre todos los miembros del equipo y la gestión, y en asegurarse de que las herramientas seleccionadas para el proceso encajen todas en su lugar.
- **La Tecnología.** El groupware y las herramientas de comunicación son requisitos tecnológicos de apoyo a los procesos distribuidos de desarrollo. Así mismo, aplicaciones Intranet, servidores del proyecto y video conferencia se han consolidado como las tecnologías más útiles en la mayoría de esfuerzos de desarrollo distribuido. Debe tenerse cuidado sin embargo, en el uso de herramientas genéricas las cuales frecuentemente están pobremente documentadas y soportadas y en ocasiones hacen el uso del proceso algo poco amigable. Asimismo, se destaca la importancia de negociar y acordar un conjunto de herramientas comunes de apoyo al proceso siempre que sea posible, aunque en ocasiones es más crítico aún el establecimiento de procesos comunes y arquitecturas abiertas para garantizar el éxito del desarrollo distribuido. Por último, la necesidad de compartir información privada en un entorno geográficamente distribuido, motiva la seguridad de la red distribuida.

De otra parte (continuando con la explicación de la **Figura 2**), mediante un continuo trabajo de la comunidad del software, las anteriores reflexiones esclarecen nuevos desafíos que traen consigo los ambientes distribuidos de desarrollo, los cuáles fundamentalmente se concentran en formas alternativas para tratar los aspectos distribuidos de actividades clave tales como: La comunicación, la infraestructura, la coordinación, la disponibilidad y la gestión. El esclarecimiento de estos nuevos desafíos, provoca una reacción, cuyo propósito fundamental consiste en la concepción de soluciones que mitiguen los inconvenientes causados por las soluciones tradicionales al tratar con los aspectos distribuidos de las actividades clave arriba mencionadas. Sin embargo, estas soluciones son meras aproximaciones y son constantemente cuestionadas a la luz de su eficacia. En consecuencia, se pueden aprender muchas lecciones de su apropiación, uso y desempeño. A este respecto, se presenta la revisión de algunos de los paralelos Desafíos vs. Soluciones que implican los esfuerzos de desarrollo distribuido, así como las lecciones aprendidas de la implementación e implantación de las mismas en entornos distribuidos. Estos aportes fundamentalmente pertenecen a Michael Kircher, Prashant Jain, Angelo Corsaro y David Levine, quienes lideran la iniciativa del Dispersed Extreme Programming o DXP.

3.3.4 Desarrollo distribuido: Desafíos encontrados y soluciones propuestas.

Desafío: La Comunicación. Un aspecto importante en la comunicación para los humanos, es conocer cómo reaccionan las personas a las cosas que uno mismo dice. Para juzgar esa reacción, típicamente una persona podría leer el lenguaje corporal o facial y hasta la entonación de la voz. En el desarrollo distribuido las personas generalmente se encuentran en ubicaciones geográficamente distantes, ¿cómo podría alguien conocer las reacciones que otros tienen acerca de sus comentarios? A continuación, se plantean varias estrategias para superar el componente distribuido de la comunicación.

- Iniciar la creación del equipo de trabajo en una ubicación física única, con el propósito de permitir a las personas el tiempo suficiente para conocerse y constituir una cultura en común. Luego, es suficiente con una pequeña ventana de video, para percibir, las mutuas reacciones que en la comunicación remota tienen las otras personas acerca de los comentarios de otros. Al respecto podrían acordarse formas de comunicación remota basadas en video, teléfono, Chat, email o tele conferencia.
- Acordar reuniones periódicas para incrementar las relaciones interpersonales de todos los miembros del equipo, de tal manera que la cooperación remota se facilite gracias a los lazos de confianza creados.
- Finalmente, la capacidad y la habilidad para compartir los documentos mejora enormemente la comunicación y la cooperación remotas.

Desafío: La Coordinación. Cuando dos o más miembros del equipo trabajan juntos en un proyecto desde ubicaciones físicas diferentes, la coordinación de su trabajo, se convierte en todo un desafío. Sincronizar la disponibilidad ajustando las diferencias en los tiempos y cronogramas personales, con el propósito de coordinar la distribución, integrar las actividades y al mismo tiempo compartir documentos y aplicaciones es un desafío a superar. A continuación, se plantea una estrategia para superar el componente distribuido de la coordinación.

- La apropiada coordinación entre los miembros del equipo requiere un mínimo de planeación. Sin embargo, hacer un uso extensivo de varias líneas de comunicación puede facilitarlo. Por ejemplo: 2 miembros del proyecto ubicados remotamente podrían intercambiar sus agendas de actividades diarias vía e-mail, en éstas podrían asignar de común acuerdo algunas franjas para dedicarlas al proyecto. Lo anterior les obligaría a tomar en cuenta las diferencias de tiempo disponible existentes para cada uno.

Desafío: La Infraestructura. Tanto la comunicación como la coordinación, en el desarrollo distribuido dependen de la infraestructura disponible. Esto incluye tanto el hardware como el software, así como el ancho de banda de la red. Una

infraestructura pobre puede dificultar seriamente estos aspectos en un proceso distribuido. La disponibilidad de una infraestructura suficiente es vital. A continuación, se plantean varias estrategias para superar el componente distribuido de la infraestructura.

- Todos los miembros del equipo deben tener el software y hardware adecuado. Se constituyen como criterios importantes de selección del software los siguientes: El software debe ser seleccionado, teniendo en cuenta su: Facilidad de uso, interoperabilidad con otras herramientas y disponibilidad en diferentes plataformas. Igualmente, el hardware debe ser seleccionado de forma cuidadosa. De hecho cada desarrollador necesitará el poder de una estación de trabajo clásica con características multimedia.

Desafío: La Disponibilidad. Los miembros de un equipo distribuido de desarrollo pueden estar disponibles pero no necesariamente de forma sincrónica Algunos de ellos incluso podrían estar trabajando en múltiples proyectos y tener su dedicación restringida, mientras que otros simplemente tendrán una disponibilidad limitada debido a asuntos de índole personal. En otras ocasiones, podrían presentarse incluso limitantes de disponibilidad relacionadas con diferentes zonas horarias. A continuación, se plantean varias estrategias para superar el componente distribuido de la disponibilidad.

- No negar ayuda a nadie que la esté solicitando especialmente si es desde una ubicación remota.
- Publicar diaria o semanalmente las agendas de disponibilidad para cada miembro del equipo.

Desafío: La Gestión. El gestor del equipo, necesita confiar mucho en sus subordinados y más aún si ellos frecuentemente son remotos. Deben definirse nuevas estrategias alternativas al control directo que realiza el gestor. A continuación, se plantean varias estrategias para superar el componente distribuido de la gestión.

- Los líderes del proyecto necesitan aprender a manejar equipos distribuidos. En particular, deben aprender cómo administrar a los miembros ubicados en locaciones remotas, esto podría incluir: Requerir diaria o semanalmente reportes de todos los miembros del equipo ya sean locales o remotos.
- Proporcionar realimentación frecuente a todos los miembros del equipo para crearles un sentimiento de conexión y pertenencia. Eventos regulares pueden ayudar a construir confianza y motivación entre todos los miembros del equipo.

Finalmente (continuando con la explicación de la **Figura 2. Modelo de Evolución del Pensamiento para procesos distribuidos**), nuevas soluciones llevan a nuevas lecciones y a partir de estas, nuevas reflexiones son planteadas, no para finalizar el ciclo, sino para iniciar uno nuevo, incrementando el conocimiento y estabilizando el concepto siempre evolutivo del desarrollo distribuido.

3.3.5 AGILE-DISOP

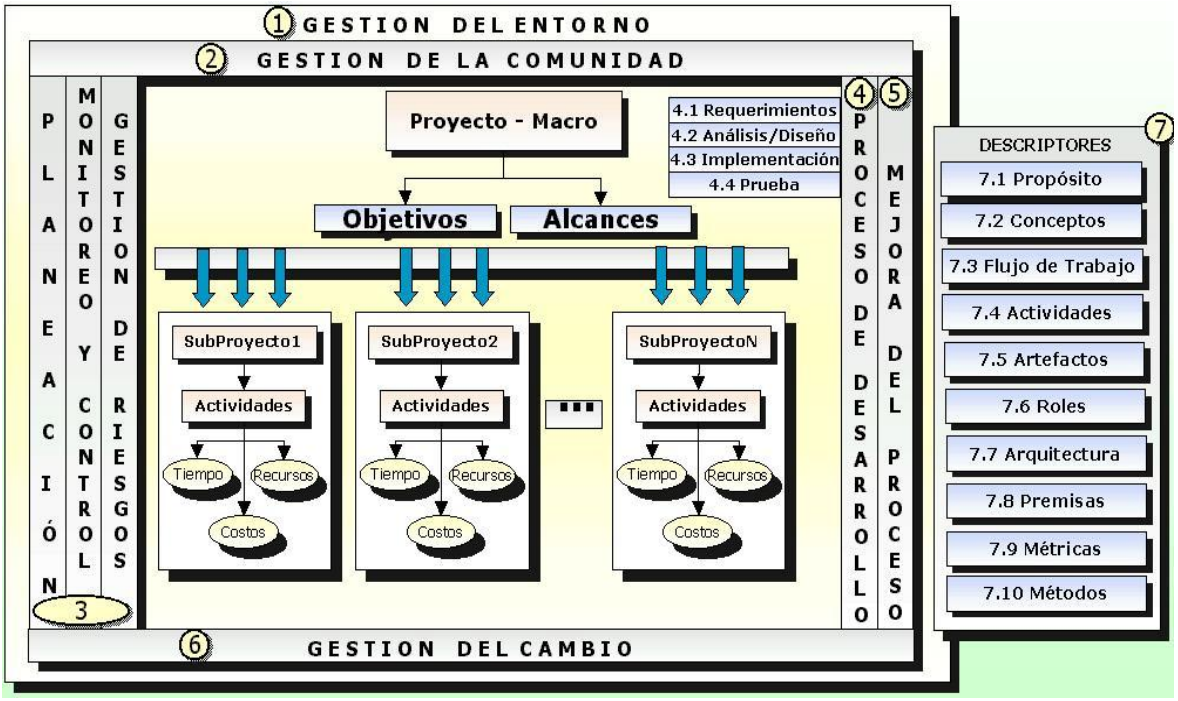
Uno de los objetivos de la tesis de investigación "Diseño de una arquitectura para un entorno de modelamiento- simulación y creación de un proceso para su desarrollo por una comunidad (I+D)" realizado por el Ing. Jorge Jair Moreno Chaustre (Moreno Chaustre, 2006), fue definir un proceso adecuado para el desarrollo distribuido de "Entorno software de Modelamiento y Simulación de Modelos Integrados ESMS-MI", se propuso la utilización del modelo **AGILE-DISOP** acrónimo de proceso ágil de desarrollo distribuido de software. Este proceso de desarrollo de software se basa en metodologías ya existentes mediante la selección, adecuación y armonización de aspectos claves para concebir un marco idóneo para el desarrollo ágil de software bajo un ambiente de dispersión geográfico. Como parte de dicha investigación se realizó una revisión de trabajos de investigación con características similares realizados a nivel mundial como: El Modelo de administración de proyectos de Zanony, Distributed eXtreme Programming (DXP), Proyecto Génesis; de cada experiencia se resaltan similitudes, diferencias y reflexiones. Producto de ésta revisión y estudio se hace la especificación del modelo Agile-Disop como proceso para el desarrollo del ESMS-MI.

En la **Figura 3. Visión General de Agile-DISOP** se proporciona un panorama global de la metodología de desarrollo de software **Agile-DISOP** propuesta. Como puede observarse, en la figura, la metodología **Agile-DISOP**, propone dos niveles de ejecución para cualquier emprendimiento de desarrollo de software. En primer lugar, se contempla el nivel "Macro", en el cual son determinados los objetivos y los alcances del producto a desarrollar en comunidad, todos estos de alto nivel. En segundo lugar, se contemplan los sub-proyectos de desarrollo, los cuáles pueden ser ejecutados por equipos de (i+d) geográficamente distantes que se integran a la comunidad de desarrollo mediante la plataforma de integración soportada por la tecnología de la información (TI). Cada sub-proyecto es ejecutado bajo un ambiente distribuido y tiene sus propias asignaciones respecto al tiempo, presupuesto y recursos que consume, así como las actividades y productos de los cuales es responsable. Estas asignaciones localizadas en cada sub-proyecto, alimentarían las asignaciones globales de los mismos aspectos para valorar el avance del proyecto en comunidad en el nivel "Macro".

La metodología **Agile-DISOP** (Figura 3), propone unos elementos integradores denominados *"Disciplinas",* las cuales agrupan basadas en su naturaleza, las actividades de la ingeniería del software y gestión de proyectos fomentando el

espíritu comunitario a través de todos los equipos de desarrollo y sub-proyectos geográficamente dispersos que ejecuta la comunidad. Las *disciplinas* pueden comportarse como integradoras transversales (a todos los *equipos* de desarrollo en la comunidad), verticales (a todos los sub-proyectos de un macro-proyecto que ejecuta la comunidad) y mixtos (para todos los equipos y proyectos).

Figura 3. Visión General de Agile-DISOP



Fuente (Moreno Chaustre, 2006)

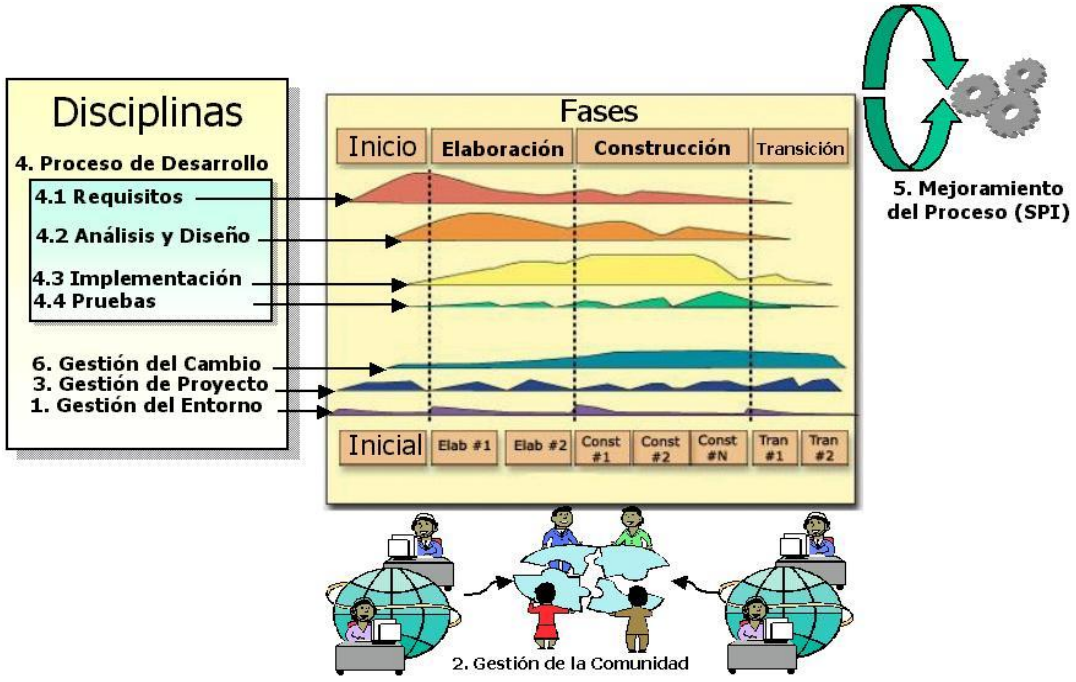
Mientras que las *disciplinas* transversales, favorecen la percepción de unidad en la comunidad, las verticales fomentan la percepción de la totalidad sobre los sub-proyectos distribuidos en un sólo Macro-Proyecto. Así mismo, cada *disciplina* estará soportada por servicios informáticos disponibles en la plataforma de integración seleccionada.

- **Organización del Proceso**

En la **Figura 4. Visión Detallada del Agile-DISOP**, se observa más concretamente los elementos internos del **Agile-DISOP** desde dos perspectivas. En primer lugar, el eje horizontal representa el tiempo e ilustra el aspecto dinámico de la metodología expresado en términos de *fases, iteraciones e hitos*. En segundo lugar, el eje vertical, representa el aspecto estático expresado en términos de las *disciplinas* (compuestas por sus *flujos de trabajo, artefactos y roles*) y su correspondiente

nivel de énfasis (medido en unidades de esfuerzo[16]) el cual está representado por las curvas de área que varían según su intersección con cada fase. Las *disciplinas* de mejoramiento del proceso y gestión de la comunicación tienen un énfasis que no es representable por las curvas de área, puesto que se encuentran en un nivel superior al del producto de software. Se hace énfasis en que **Agile-DISOP**, se inspira en el proceso unificado de desarrollo (RUP por las siglas de Rational Unified Process), enriquecido con algunas prácticas y recomendaciones de las metodologías de desarrollo de software libre.

Figura 4. Visión Detallada del Agile-DISOP



Fuente (Moreno Chaustre, 2006)

A continuación se detalla brevemente la organización del proceso en términos de las *disciplinas y las fases*.

- **Disciplinas del Agile-DISOP**

A continuación se describen brevemente las *disciplinas* contempladas por **Agile-DISOP** en la misma secuencia que se muestra en la **Figura 3** y **Figura 4**:

1. **Gestión del Entorno.** Su propósito consiste en fortalecer la adecuación, puesta en marcha y mantenimiento tanto del proceso mismo **Agile-**

[16] Por ejemplo: Horas Hombre al Mes.

DISOP como de la plataforma de integración para la comunidad de desarrollo seleccionada.

2. **Gestión de la Comunidad**. Su propósito consiste en soportar las necesidades de comunicación, coordinación y gestión de la disponibilidad para personas y equipos (i+d) bajo un ambiente de dispersión geográfica que pertenecen a la comunidad de desarrollo.
3. **Gestión del Proyecto.** Su propósito consiste en proveer un marco metodológico para la planificación, monitoreo, control y gestión de riesgos en macro-proyectos y/o sub-proyectos informáticos de la comunidad de desarrollo.
4. **Proceso de Desarrollo.** Su propósito consiste en proveer un marco metodológico para la ejecución de las *disciplinas* orientadas hacia la investigación de requisitos-análisis, diseño, implementación y pruebas para el desarrollo software.
5. **Mejora del Proceso (SPI).** Su propósito consiste en proveer los mecanismos para soportar el mejoramiento continuo de mismo modelo **Agile-DISOP** con el propósito de alcanzar de forma incremental un grado de madurez cada vez más alto y competitivo.
6. **Gestión del Cambio.** Su propósito consiste en proveer mecanismos que soportados informáticamente permitan controlar los cambios en los artefactos producidos por la comunidad de desarrollo. El control sobre la documentación ayudará a la comunidad a evitar la confusión, asegurando que los artefactos resultantes no tengan conflictos entre sí, debido a las actualizaciones simultáneas y las numerosas versiones que puede manejar la comunidad de desarrollo en un momento determinado.

- **Fases del Agile-DISOP**

Las fases en **Agile-DISOP** se entienden como los momentos que vive el software a través de su maduración desde una idea hasta un producto terminado durante el ciclo de vida de desarrollo. Según RUP (Rational Software Corporation, 2003) se proponen 4 de estas fases: Inicio, Elaboración, Construcción y Transición. Cada fase tiene un propósito y unos hitos que determinan cuándo la fase finaliza, dando paso a la siguiente. **Agile-Disop**, retoma el espíritu de estas fases de RUP las cuales se describen a continuación:

- ***Fase de Inicio***

Su objetivo fundamental consiste en alcanzar la estabilidad de los objetivos del sistema mediante el consenso general de toda la comunidad de desarrollo. Durante esta fase, la comunidad de desarrollo debe ponerse de acuerdo en cuestiones relativas a los alcances, riesgos, requerimientos y objetivos del sistema con el propósito de comprobar la viabilidad del proyecto teniendo en cuenta las condiciones del entorno de desarrollo.

- ***Fase de Elaboración***

Esta fase del proceso tiene como propósito fundamental la implementación de los requisitos funcionales que son críticos desde el punto de vista de la arquitectura, haciendo realidad la estabilización temprana de la misma. Además, determinará una concepción estable de los requerimientos y las estrategias relacionadas con la mitigación de riesgos para estimar de manera más fiable los costos y cronogramas para la culminación del proyecto. Durante esta fase, la comunidad debe ponerse de acuerdo sobre varias cuestiones riesgosas tales como: conflictos de requerimientos y diseño, reutilización de componentes, estrategias de demostración del producto a los clientes, asegurar una línea base de la arquitectura que soportará los requerimientos del sistema en un tiempo y costo razonables, establecer y configurar un entorno de soporte, interacción e integración para la comunidad.

- ***Fase de Construcción.***

Esta fase del proceso tiene como propósito fundamental el análisis, diseño, implementación y prueba del resto de casos de uso (y otros requerimientos) que agregan funcionalidad al sistema, los cuales crecen sobre la línea base de la arquitectura estabilizada durante la fase de elaboración. En cierto sentido, esta fase se parece a un proceso de manufactura, debido a que se concentra en la gestión de los recursos y las operaciones para optimizar los costos, cronogramas y la calidad durante la ejecución del resto del proceso. Además, se concentra en la determinación de la disposición de la comunidad de usuarios para recibir las primeras versiones de producción del software. De otro lado, es preciso determinar los mecanismos que permitan el trabajo paralelo para los equipos de desarrollo de la comunidad, siempre que los recursos lo permitan, con el propósito de incrementar la velocidad de desarrollo pero teniendo en cuenta el aumento en la complejidad de la gestión. Sin embargo, una arquitectura robusta influye en la asignación de trabajo paralelo.

- ***Fase de Transición.***

El propósito fundamental de esta fase consiste en asegurar que el software estará disponible para sus usuarios finales. Para iniciar esta fase es necesario que la línea base de funcionalidad del producto esté madura y sea suficiente para ser distribuida a través de la comunidad de desarrollo. Esto último, implica que usualmente exista un subconjunto de características del sistema implementadas y documentadas con calidad, listas para ser aprovechadas por los usuarios finales.

Durante la transición es posible iterar varias veces haciendo énfasis en la disciplina de pruebas con el objeto de preparar el producto para su liberación con base en la realimentación de los usuarios. En este punto, la realimentación del usuario ayuda a enfatizar en los ajustes de granularidad fina sobre el producto, su configuración e instalación, así como las dificultades relacionadas con la usabilidad. Al final de esta fase, el proyecto debe estar próximo a finalizar, sin embargo el final del proyecto puede coincidir con el inicio de otro para la próxima

versión del producto. También es posible que todos los artefactos del proyecto sean entregados a otro equipo encargado del mantenimiento del software durante su etapa de producción. La naturaleza del producto desarrollado determina la complejidad de esta fase. Las actividades de esta fase suelen ajustarse al objetivo perseguido para la transferencia del producto, ejemplo: Si se está depurando los errores, tan sólo es necesario ejecutar las disciplinas de implementación y prueba[17], sin embargo, si se requiere agregar nuevas características, entonces debe ejecutarse también las disciplinas del análisis y el diseño.

Igualmente, es importante socializar con toda la comunidad de desarrollo, las características importantes ofrecidas por la nueva versión del producto así como un informe ejecutivo de los errores corregidos. Finalmente, es primordial delimitar las estrategias relacionadas con el entrenamiento de los usuarios finales y puesta en marcha del plan de marketing y distribución así como el ajuste a los errores en el desempeño, usabilidad del producto y soporte al usuario.

3.3.6 Arquitectura software

En la actualidad no hay un consenso en la definición de una arquitectura software. Lo anterior se ve reflejado en la gran cantidad de definiciones diferentes de "arquitectura software". El Software Engineering Institute (SEI) ha recopilado y clasificado un conjunto de definiciones en Modernas (2 definiciones), Clásicas (9 definiciones), bibliográficas (17 definiciones) y más de 188 definiciones de la comunidad (Arquitectos software, administradores de sistemas, consultores, ingenieros de sistemas, etc.) (SEI, 2009).

Para el presente trabajo se tomará como referencia la siguiente definición de arquitectura:

"La Arquitectura Software de un programa o sistema es la estructura o estructuras del sistema, la cual comprende los elementos de software, las propiedades visibles desde el exterior de estos elementos y las relaciones entre estos" (Bass, y otros, 2007).

3.3.7 Patrones para arquitecturas software

Un patrón para arquitectura software describe un problema particular de diseño el cual surge en un contexto específico, y provee de un esquema genérico para su solución debidamente comprobada. El esquema de solución es descrito por los componentes, sus responsabilidades y relaciones y la forma como se colaboran entre sí. (Buschmann, y otros, 1996).

[17] Ejemplo: Pruebas beta del nuevo sistema contra las expectativas de los usuarios y operación en paralelo con el sistema que será reemplazado.

3.3.8 Redes humanas dispersas geográficamente apoyadas con la TI (comunidades virtuales)

También llamadas redes humanas telemáticas, son redes humanas cuyos individuos se encuentran separados geográficamente y utilizan las redes electrónicas como herramienta para afianzarse, hacerse más eficientes y fortalecerse.

Una red humana es una agrupación de personas que comparten conocimientos, habilidades, recursos, entre otros y unifican esfuerzos para trabajar por el logro de objetivos comunes.

Una red humana telemática es una construcción social, que al estar apoyadas con la interconexión de las redes de computadores eliminan barreras y expanden las posibilidades de investigación y desarrollo. Lo más importante de lograr una interconexión entre las redes humanas y tecnológicas es no dar prioridad a la infraestructura (aunque es necesaria) como medio y escenario de interacciones, sino que hay que reconocer y alimentar la capacidad que las personas desarrollan para transformar la información disponible y las relaciones existentes en la red, en conocimiento provechoso para mejorar su vida y relaciones de apoyo mutuo.

Las redes humanas telemáticas se caracterizan fundamentalmente por los siguientes elementos:

- **Las personas** interactúan socialmente para satisfacer sus propias necesidades o las de otros, desempeñando diversos roles en la comunidad.
- **Un propósito en común.** El interés, la necesidad, el intercambio de la información o los servicios de valor agregado que dan sentido a la existencia y a la continuidad de la comunidad.
- **Las políticas,** los rituales, protocolos, reglas y leyes que guían la interacción y regulan el comportamiento entre los miembros de la comunidad.
- **Los sistemas de computadora** soportan, facilitan y median la interacción social y el sentido de pertenencia a la comunidad, mediante el establecimiento de lazos electrónicos.
- **Límites permeables.** Una entidad puede pertenecer a varias organizaciones, las fronteras físicas no existen.
- **Una estructura reconfigurable.** Se pueden redefinir procesos, roles, asignaciones y carga de trabajo según las necesidades.

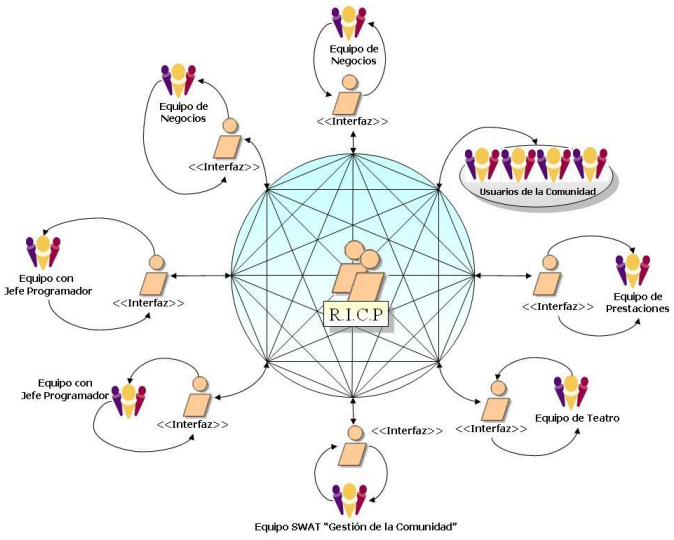
AGILE-DISOP es una metodología enfocada al desarrollo de software por medio de redes humanas distribuidas. Esta metodología define una estructura de organización en red como la presentada en la **Figura 5**. Cada equipo de desarrollo puede adoptar una configuración diferente dependiendo de las motivaciones de la comunidad[18].

[18] Estas configuraciones son descritas en (Moreno Chaustre, 2006)

- Programador Jefe
- En la sombra
- Prestaciones
- Equipo de teatro
- Negocios.
- SWAT
- De Especialistas
- Búsqueda-Rescate
- De teatro

Cada uno de estos equipos se debe acoplar a la comunidad mediante interfaz humano-computador, que son responsables de la garantía, mantenimiento y soporte de los canales de comunicación con el propósito de asegurar una coordinación limpia y eficaz al interior de la comunidad.

Figura 5. Organización de los equipos en la comunidad de desarrollo



Fuente (Moreno Chaustre, 2006)

El **R.I.C.P** es el acrónimo de: *"Responsable de la Integridad Conceptual del Producto"* y es la persona o grupo de personas responsable final de la integridad conceptual del producto. Este rol es desempeñado principalmente por el equipo promotor del proyecto.

CAPÍTULO 4. TESIS DE MAESTRÍA

Este capítulo presenta una sinopsis de los resultados obtenidos por la tesis de maestría "*Diseño de una arquitectura para un entorno de modelamiento- simulación y creación de un proceso para su desarrollo por una comunidad (I+D)"*(Moreno Chaustre, 2006). La presente tesis es continuidad de la referida, en el marco del macro-proyecto de investigación del grupo SIMON.

4 TESIS DE MAESTRÍA “DISEÑO DE UNA ARQUITECTURA PARA UN ENTORNO DE MODELAMIENTO- SIMULACIÓN Y CREACIÓN DE UN PROCESO PARA SU DESARROLLO POR UNA COMUNIDAD (I+D)”

4.1 INTRODUCCIÓN

El presente trabajo de investigación es continuidad de la tesis de maestría “Diseño de una arquitectura para un entorno de modelamiento- simulación y creación de un proceso para su desarrollo por una comunidad (I+D)” (Moreno Chaustre, 2006), por lo que se convierte en su principal antecedente. Este capítulo presenta los resultados obtenidos luego de realizar un estudio de la investigación mencionada.

El capítulo inicia con el análisis de la evaluación arquitectónica realizada al software Evolución 3.5 (Cuellar Yeneris, y otros, 2003), para lo cual se utilizó una metodología que consiste en diez actividades o etapas y en la que se aplica un conjunto de métricas para evaluar los atributos de calidad definidos previamente. Además, se presenta el conjunto de herramientas software utilizado para dicha evaluación. Finalmente, se plantean las recomendaciones para el mejoramiento del diseño y la arquitectura del software Evolución.

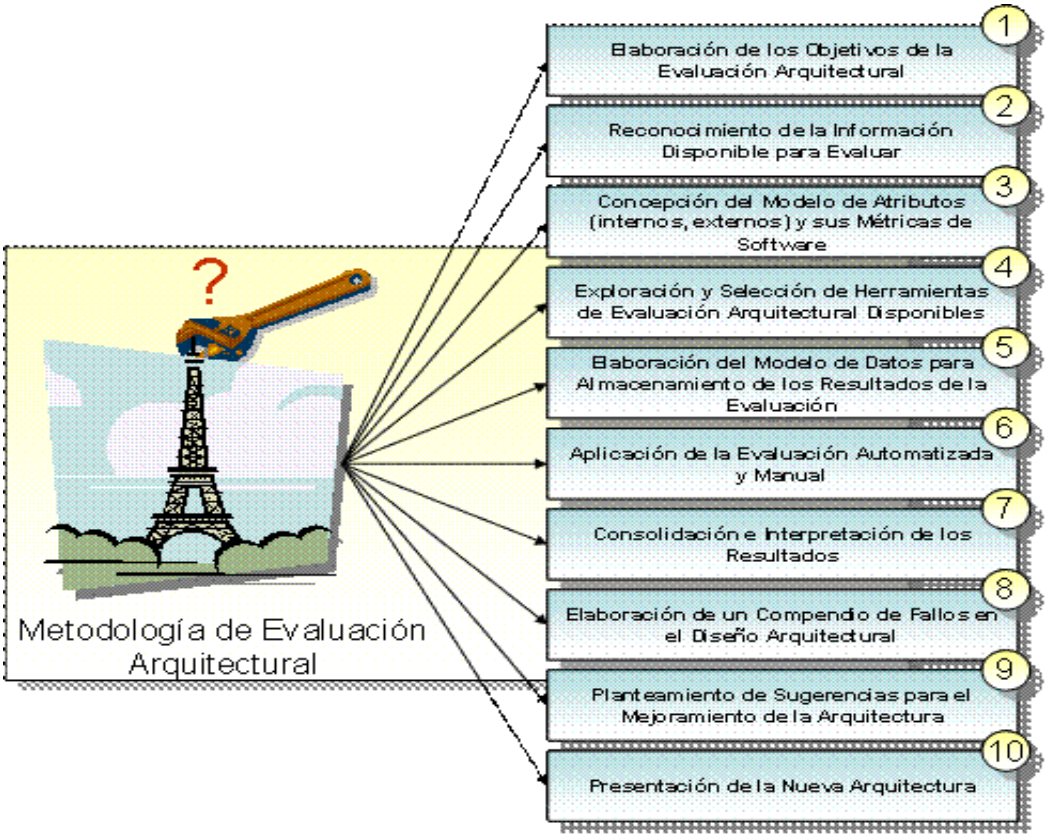
Por la poca documentación del software (como diagramas de clases), se requirió realizar una reconstrucción de la arquitectura utilizando ingeniería inversa, haciendo uso de herramientas software y como producto se presenta la arquitectura con su respectiva documentación (diagramas UML y especificación de requisitos). Dentro de la propuesta del nuevo entorno, se define un proceso de desarrollo software denominado AGILE-DISOP, el cual está diseñado para el desarrollo software por comunidades geográficamente distribuidas. Por último, se propone un conjunto de herramientas para la gerencia del proyecto y para la promoción, gestión y comunicación de la comunidad desarrolladora.

Finalmente, se presentan las conclusiones del análisis realizado a la tesis “Diseño de una arquitectura para un entorno de modelamiento- simulación y creación de un proceso para su desarrollo por una comunidad (I+D)”.

4.2 EVALUACIÓN ARQUITECTÓNICA DE EVOLUCIÓN

La evaluación realizada a la arquitectura de Evolución, consistió en un conjunto de 10 actividades las cuales se presentan en la **Figura 6. Metodología de evaluación arquitectural**.

Figura 6. Metodología de evaluación arquitectural



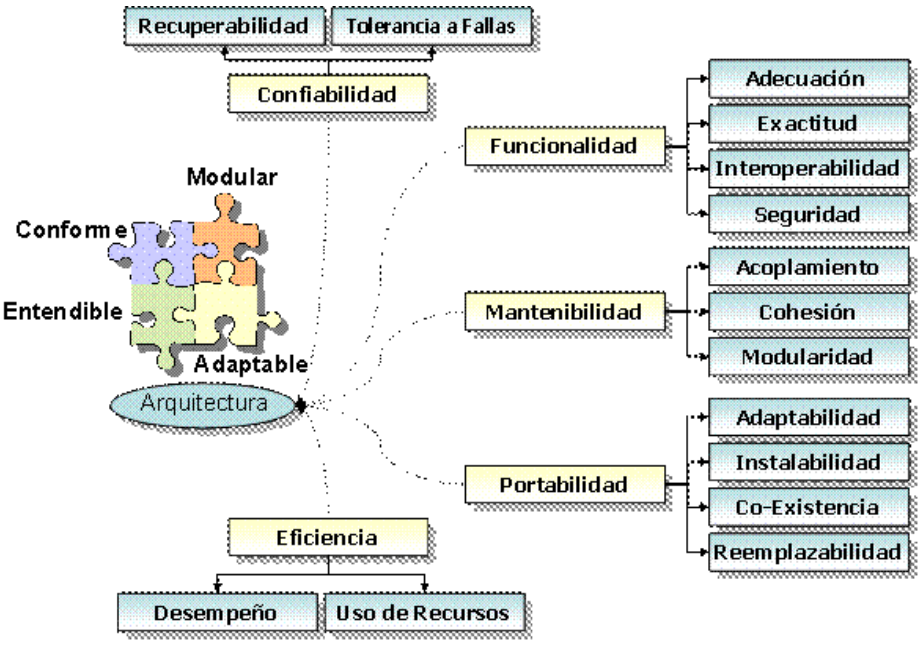
Fuente (Moreno Chaustre, 2006)

Para el desarrollo de la evaluación se realizaron la totalidad de las actividades descritas, pero aquí se resaltará el desarrollo de la concepción del modelo de atributos y las métricas de software empleadas para su evaluación, así como el conjunto de herramientas software empleadas para la evaluación de manera automática, semiautomática y manual. Igualmente, es de resaltar los resultados de dicha evaluación y las sugerencias dadas por el autor de la tesis, a la arquitectura de Evolución. A continuación, se presenta cada uno de los ítems que se acaban de mencionar.

4.2.1 Definición de los atributos de calidad

El modelo de atributos de calidad definido consta de 5 atributos y 15 sub-atributos, los cuales son presentados en la **Figura 7**.

Figura 7. Atributos de calidad utilizados en la evaluación de Evolución



Fuente (Moreno Chaustre, 2006)

4.2.2 Selecciona un conjunto de métricas para la evaluación de la arquitectura (el diseño)

Para evaluar el diseño de Evolución de generó un modelo de métricas de software. Este modelo está conformado por las métricas listadas a la **Tabla 1**.

Tabla 1. Modelo de métricas aplicado a Evolución

DIMENSIÓN	ELEMENTO
REQUISITOS	SIMPLICIDAD
	OBJETIVIDAD
	VALIDEZ
	ROBUSTEZ
	FÁCIL RECOLECCIÓN
NATURALEZA	DIRECTA
	INDIRECTA
ATRIBUTOS	MODIFICABILIDAD Y DEFECTOS POTENCIALES DE REQUISITOS
	COHESIÓN
	ACOPLAMIENTO
	HERENCIA
	POLIMORFISMO
	COMPLEJIDAD
	REUTILIZACIÓN
	ENCAPSULAMIENTO

DIMENSIÓN	ELEMENTO
	TAMAÑO
HEURÍSTICAS (REGLAS DE DISEÑO)	CLASE NO USADA
	CLASE NO REPRESENTADA
	PAQUETE NO REPRESENTADO
	CLASE ENORME
	OPERACIÓN DUPLICADA
	NOMBRE DE CLASE INCORRECTO
	REFERENCIAS CIRCULARES
	SOBRE- ESCRITURA DE ATRIBUTO HEREDADO
DISCIPLINA	REQUISITOS
	DISEÑO
	IMPLEMENTACIÓN
	PRUEBAS
GRANULARIDAD	MÉTODO
	CLASE
	PAQUETE
	SISTEMA
	MODELO DE CASOS DE USO

Fuente (Moreno Chaustre, 2006)

4.2.3 Revisión y selección de las herramientas para la evaluación

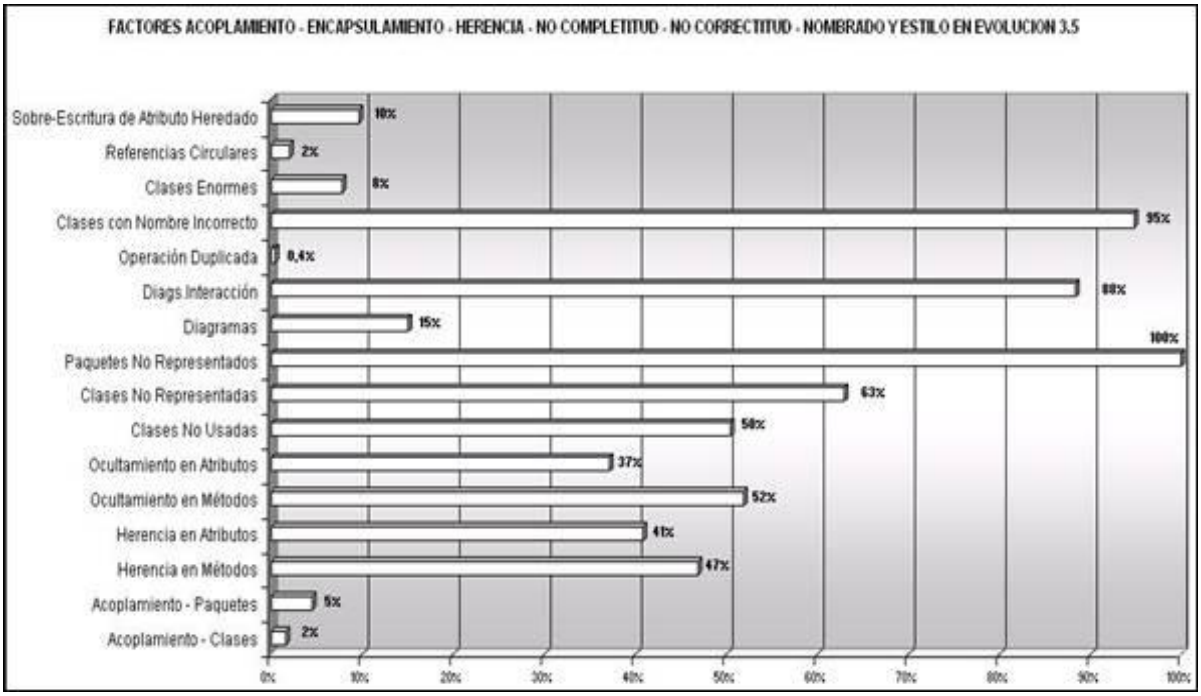
Para el desarrollo de la evaluación por medio de la implementación del modelo de métricas descrito en la sección anterior, se hizo necesario seleccionar un conjunto de herramientas que permitieron aplicar las métricas al software evaluado. La aplicación del modelo de métricas se desarrollo de manera automática, semi-automática y manual dependiendo de las métricas y el software utilizado.

- **UML Studio:** software de diseño en UML con en el que desarrolló la documentación (a nivel de diseño) de Evolución (EVL) 3.5.
- **SDMetrics:** Herramientas con capacidades automatizadas para analizar y valorar un modelo (métricas). Como entrada requiere un archivo XMI y como salida provee tres vistas: métricas, elemento y tabla, entre las métricas están: tamaño acoplamiento, complejidad y herencia. También la completitud, correctitud, nombrado y estilo.
- **EssModel**: Esta herramienta permite la generación de un modelo a partir de código fuente escrito en C, Java o Delphi. Este modelo puede ser exportado a un archivo en formato XMI
- **MS-Excel:** tabulación, organización e interpretación de los datos

4.2.4 Resultados de la evaluación

Los resultados de la Evaluación de EVL arrojaron un conjunto de deficiencias a nivel de diseño y documentación. Estos resultados son condensados en la **Figura 8**.

Figura 8. Resultados de la evaluación de Evolución



Fuente (Moreno Chaustre, 2006)

4.2.5 Sugerencias para el mejoramiento del diseño de Evolución

La tesis plantea un conjunto de sugerencias con el propósito de mejorar el diseño de Evolución a la luz de los resultados obtenidos en la evaluación realizada.

En cuanto a la Completitud (4081 problemas encontrados principalmente en los componentes animador, editor, influencias, motor, comunes, ZIP y FLAT):

- Documentar (en modelos) todos los paquetes del software
- Documentar (en modelos) todas las clases del software
- Realizar todos los diagramas de UML (casos de uso, diagramas de secuencias y de interacción)

Lo anterior mejora la facilidad de comprensión de la arquitectura de EVL 3.5 por parte de la comunidad desarrolladora

En cuanto a la correctitud (25 problemas encontrados principalmente en los componentes *principal, animador, editor, proyecto y comunes*):

- El 5% (23 de 448)de las clases no coinciden su nombre en el modelo y el código fuente.
- Eliminar las operaciones duplicadas (la clase TManejadorEvaluacion aparecen dos operaciones duplicadas)

Lo anterior aporta a la facilidad de comprensión y facilidad de mantenimiento debido a la coherencia entre el modelo y el código fuente

En cuanto al Nombrado y estilo (87 problemas encontrados):

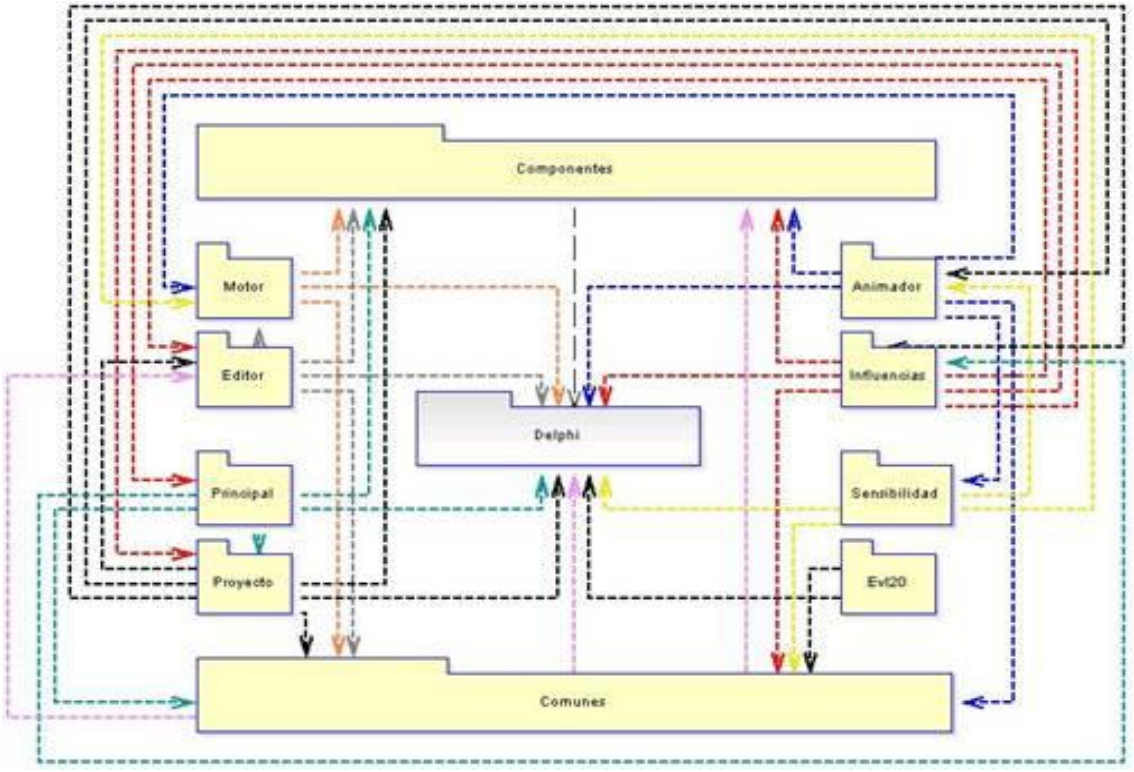
- 35 de las 448 clases (8%) son clases enormes, que se deben descomponer en clases más pequeñas si no existe una justificación.
- Las clases no deben tener referencias circulares ¿? (9 clases en EVL)
- No sobre-escribir atributos heredados (43 clases de EVL)

4.3 RECONSTRUCCIÓN DE LA ARQUITECTURA UTILIZANDO INGENIERÍA INVERSA

Para la reconstrucción de la arquitectura a partir del código fuente (ingeniería inversa) se utilizaron los siguientes herramientas software: EssModel, PowerDesigner y UMLStudio. El procedimiento realizado fue el siguiente:

1. Con EssModel se genera un modelo en formato .XMI
2. Se importa en el programa PowerDesigner (herramienta para modelado).
3. Se compara con el entregado en la documentación de EVL 3.5 (UMLStudio).
4. Se definen los paquetes lógicos de la arquitectura de EVL (que corresponden a las carpetas en las que son clasificadas las unidades de EVL 3.5).
5. Se distribuyen todas las clases en los paquetes lógicos.
6. Se crean todos los diagramas de clase y los diagramas de paquetes de EVL 3.5
7. Arquitectura resultante:

Figura 9. Arquitectura de Evolución



Fuente (Moreno Chaustre, 2006)

8. Se definen los diagramas de casos de uso, diagramas de clase, de paquetes, modelo de componentes y especificación de requisitos funcionales y no funcionales.

Estos modelos fueron realizados en la herramienta PowerDesigner

4.4 DEFINICIÓN DE UN PROCESO DE DESARROLLO DE SOFTWARE

Debido a la complejidad del desarrollo del nuevo entorno software de modelado y simulación, se propone su desarrollo por una comunidad geográficamente distribuida. Para lo cual Jorge Moreno define una metodología denominada AGILE-DISOP. Esta corresponde a las metodologías denominadas “Agiles” y está diseñada específicamente para proyectos de desarrollo software por medio de sub-proyectos geográficamente distribuidos.

4.5 DEFINICIÓN DE LA PLATAFORMA PARA LA INTEGRACIÓN DE LA COMUNIDAD

En procura de facilitar la promoción, gestión y comunicación de la comunidad, así como la gerencia del proyecto, se define una plataforma software con las siguientes herramientas:

Para la gerencia del proyecto se propone el uso del software DotProject, que permite hacer un cronograma de actividades por proyectos y hacer seguimiento del desarrollo de cada uno. Este software es una alternativa gratuita al software MS-Project de Microsoft, es software libre y plataforma web lo cual facilita su uso por una comunidad distribuida geográficamente.

Para la promoción, gestión y comunicación de la comunidad se propone el uso del software Moodle, el cual también es plataforma web y de libre distribución. Este software es un gestor de cursos el cual puede ser configurado para permitir la creación de grupos de trabajo y facilita diferentes medios para la comunicación entre estos, así como publicar y compartir diferentes tipos de documentos. También ofrece herramientas como Wikis, foros y mensajería.

4.6 CONCLUSIONES DEL ANÁLISIS A LA TESIS:

- Se realizó un análisis a todo el diseño del software Evolución, más que a la arquitectura en sí.

- Se define una metodología (métodos y herramientas) para la evaluación arquitectónica y el diseño de un software; dicha metodología puede ser usada para otras aplicaciones del grupo SIMON y de la Ea escuela de Ingeniería de Sistemas de la UIS.

- Se hace uso de UML como lenguaje para documentar la arquitectura, el cual no es propiamente un lenguaje para definir arquitecturas o ADL.

- La arquitectura de Evolución no es apropiada para los objetivos del nuevo ESMS-MI.

CAPÍTULO 5. INGENIERÍA DE REQUISITOS PARA EL ESMS-MI

El presente capítulo corresponde a la del objetivo:

> *Continuar con la especificación de los requerimientos funcionales y elaborar la arquitectura (diseño de alto nivel) del Entorno Software de Modelamiento y Simulación de Modelos Integrados (ESMS-MI), garantizando la integración entre la Dinámica de Sistemas y otras herramientas matemáticas*[19].

Este capítulo se centra en la obtención de los requerimientos del sistema, lo correspondiente a la arquitectura del sistema será tratado en el siguiente capítulo.

[19] Continuar el desarrollo realizado por la tesis "Diseño de una arquitectura para un entorno de modelamiento-simulación y creación de un proceso para su desarrollo por una comunidad (I+D)".

5 INGENIERÍA DE REQUISITOS PARA EL ESMS-MI

5.1 INTRODUCCIÓN

La ingeniería de requisitos es la etapa de la ingeniería de software encargada de obtener ("elicitation" en ingles) los requisitos funcionales y no funcionales, así como la documentación, comunicación y seguimiento de los mismos, a lo largo del proceso de desarrollo de un sistema software a desarrollar.

El término "obtener" requisitos se utiliza en lugar del término "capturar" o "suscitar" requisitos, con el objetivo de evitar la tentación a creer que los requisitos se encuentran a la espera de ser recogidos por medio de preguntas simples (Nuseibeh, y otros, 2000). Como definición de ingeniería de requisitos, se asumirá la propuesta por (Zave, 1997):

> "La ingeniería de requisitos es la rama de la ingeniería de software preocupada con los objetivos de la realidad, funciones y restricciones de un sistema software. También se preocupa con la relación de estos factores para precisar especificaciones acerca del comportamiento del software, y su evolución en el tiempo y a través de familias de software"

A continuación, se describen las etapas que involucra la ingeniería de requisitos (Nuseibeh, y otros, 2000):

- Obtener requisitos (eliciting requirements)
- Modelado y análisis de requisitos (modelling and analysing requirements)
- Comunicación de requisitos (communicating requirements)
- Acordar los requisitos (agreeing requirements)
- Evolución de los requisitos (evolving requirements)

5.2 OBTENCIÓN DE LOS REQUISITOS

Existen variadas técnicas para la obtención de los requisitos, las cuales son usadas dependiendo del dominio particular de la aplicación a desarrollar, como por ejemplo, las técnicas tradicionales (cuestionarios, entrevistas, entre otras), los casos de uso en el análisis orientado a objetos, realización de prototipos, máquinas de estado finito (para software de proceso crítico), escenarios, entre otros. Un estudio más riguroso sobre técnicas de obtención de requisitos es presentado en (Nuseibeh, y otros, 2000).

Los interesados en el proyecto[20] (El cliente, los usuarios, etc), los documentos y los procesos del dominio de la aplicación y software anteriores son fuentes de información importantes a la hora de obtener los requisitos del sistema a desarrollar. EL desarrollo del ESMS-MI parte del software Evolución, (Cuellar Yeneris, y otros, 2003), por lo tanto, éste es fuente de información para identificar requisitos para el nuevo entorno, junto con otras herramientas de su tipo.

En las metodologías de desarrollo de software iterativas, el proceso de obtener requisitos se realiza durante gran parte del ciclo de vida del software, principalmente en sus primeros ciclos o iteraciones de desarrollo, en los que se busca obtener la mayor cantidad de requisitos (sobre todo los de mayor relevancia).

Para el desarrollo del ESMS la obtención de gran parte de los requisitos durante la fase de inicio es vital porque permite a la comunidad tener un punto de partida para discutir y establecer los requisitos definitivos del sistema (durante el transcurso del desarrollo del entorno). Igualmente, la obtención inicial de requisitos es el punto de partida para el desarrollo de la arquitectura, la cual permitirá establecer cada uno de los sub-proyectos y cada una de sus responsabilidades.

Para la obtención de los requisitos del ESMS-MI se realizaron las siguientes actividades:

- Revisión de la documentación del software Evolución generada por la tesis (Moreno Chaustre, 2006).
- Revisión de otras herramientas para el modelado y la simulación con D.S.
- Entrevistas y charlas con los interesados en el proyecto.
- Elaboración y discusión de un boceto inicial del sistema con los interesados en el proyecto.
- Elaboración y discusión de modelos de escenarios descritos por casos de uso.

Posterior a la realización de las actividades antes mencionadas, se generó el conjunto de requisitos funcionales y no funcionales del nuevo entorno. Otro

[20] Conocidos como Stakeholders en la lengua inglesa

producto generado es el documento “Especificación de requisitos para el software ESMS-MI”[21] (

Anexo A), el cual cumple el estándar de la IEEE 830-1998 “*Recommended Practice for Software Requirements Specifications”* (IEEE Computer Society, 1998).

5.3 MODELADO Y ANÁLISIS DE REQUISITOS

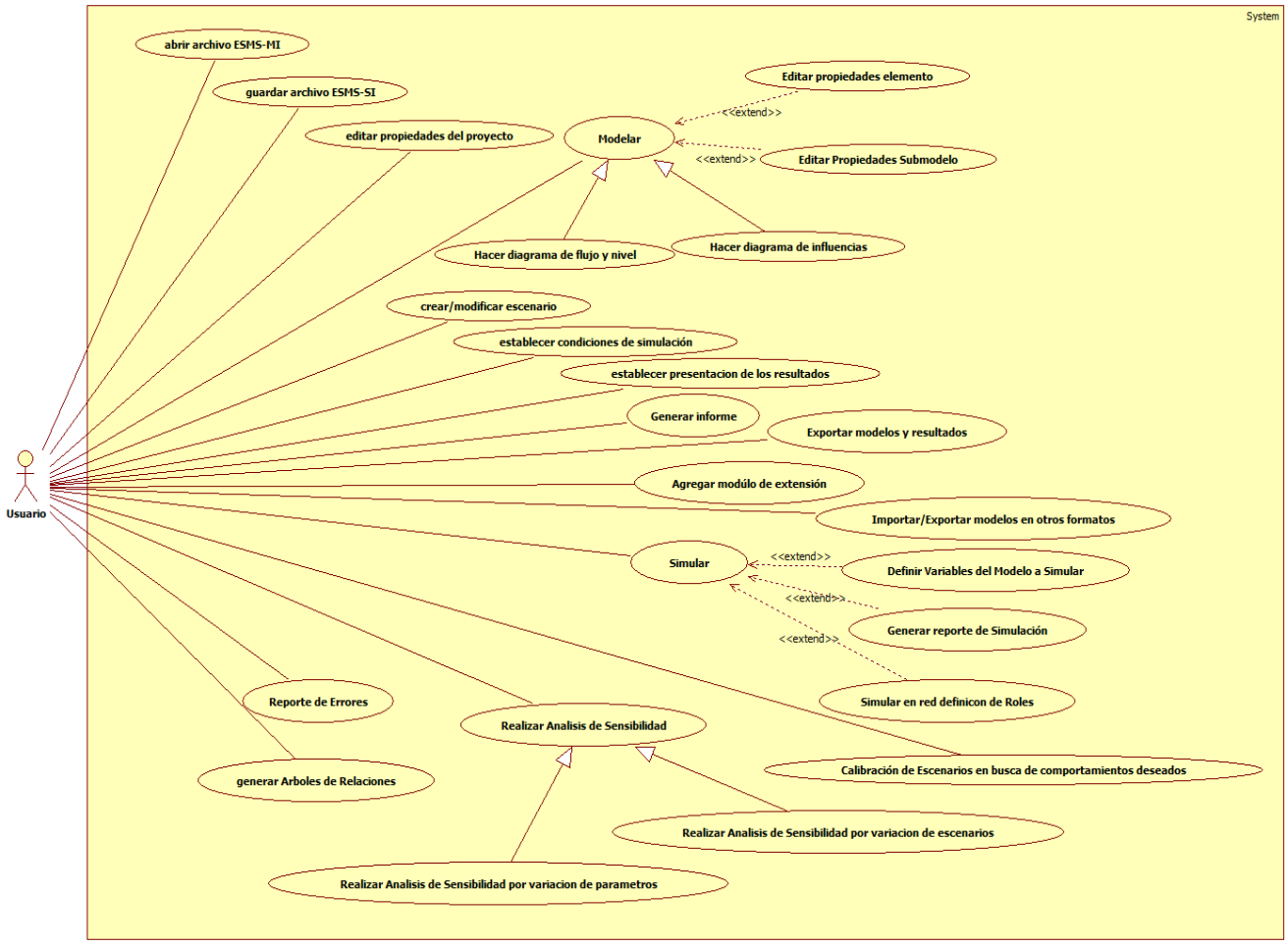
Existen dos tipos de requisitos: los requisitos funcionales y los no funcionales o atributos de calidad. Para modelar los requisitos funcionales se realiza dos tipos de diagramas, los modelos de escenarios y los modelos de requisitos funcionales, los cuales están basados en (Inquiry-based requirements analysis, 1994). Estos modelos están en el lenguaje de casos de uso de UML y para su desarrollo se utilizó el software StarUML, el cual es de uso gratuito y de código abierto. En cuanto a los requisitos no funcionales se emplea los “escenarios de atributos de calidad”, los cuales son descritos en las siguientes secciones. Los diagramas de estos últimos se realizaron en el software propietario Microsoft Visio, pero pueden ser realizados en cualquier software para elaboración de gráficos.

5.3.1 Requisitos funcionales

En la **Figura 10** se presenta los modelos de escenarios del ESMS-MI. Los modelos de requisitos funcionales no son presentados en este documento por su extensión, pero pueden ser consultados en el documento adjunto *Especificación de requisitos para el software ESMS-MI* (**Anexo A**)

[21] SRS por sus siglas en ingles

Figura 10. Casos de uso de escenarios del ESMS-MI



Fuente: Autor

5.3.2 Atributos del sistema de software (Requisitos no funcionales)

Existe una gran cantidad de taxonomías y definiciones para los atributos del sistema software (Bass, y otros, 2007), también denominados atributos de calidad o requisitos no funcionales. En este documento se hace uso de un conjunto de escenarios de atributos de calidad para ayudar a la obtención de los requisitos no funcionales, esta técnica es propuesta en (Bass, y otros, 2007).

Los escenarios de atributos de calidad están conformados por las siguientes partes (Ver **Figura 11**):

Fuente del estimulo: es alguna entidad (un humano, un sistema de computadora o cualquier otro actor) que genera el estimulo.

Estimulo: es el requisito que necesita ser considerado cuando este llega a un sistema.

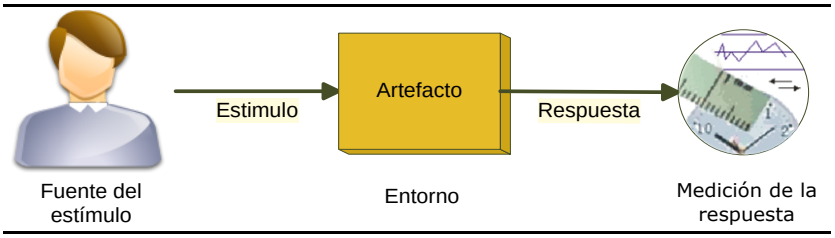
Entorno: el estimulo ocurre con ciertas condiciones. El sistema puede estar en una condición de sobrecarga o puede estar ejecutándose cuando el estimulo ocurre, o algún otro requisito o condición que pueda ser cierta. Esta parte contextualiza el requisito

Artefacto: el artefacto estimulado. Este puede ser todo el sistema o alguna parte de éste.

Respuesta: es la actividad realizada o emprendida después de la llegada del estímulo.

Medición de la respuesta: cuando la respuesta es generada, esta puede ser medida de alguna forma, de modo que los requisitos pueden ser probados.

Figura 11. Escenario de atributo de calidad



Fuente: Autor

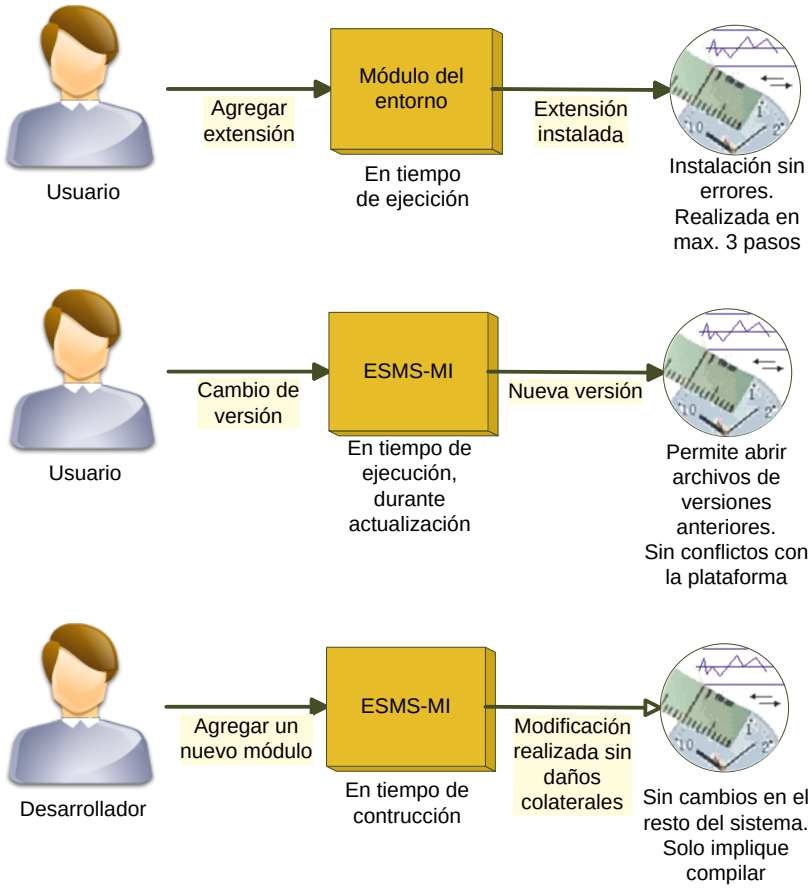
Dentro de los atributos de calidad más relevantes para el ESMS-MI se pueden enumerar los siguientes (en orden de importancia):

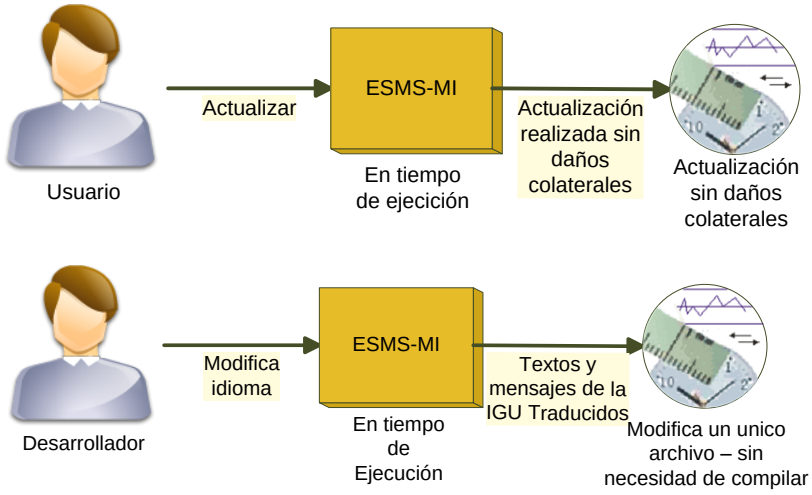
5.3.2.1 Modificable (Modifiability)

Facilidad para el cambio y para realizar mantenimiento. Esfuerzo necesario para localizar y arreglar un error en un programa o realizarle cambios.

Dentro de este atributo de calidad encontramos los siguientes escenarios:

Figura 12. Escenarios del atributo modificable



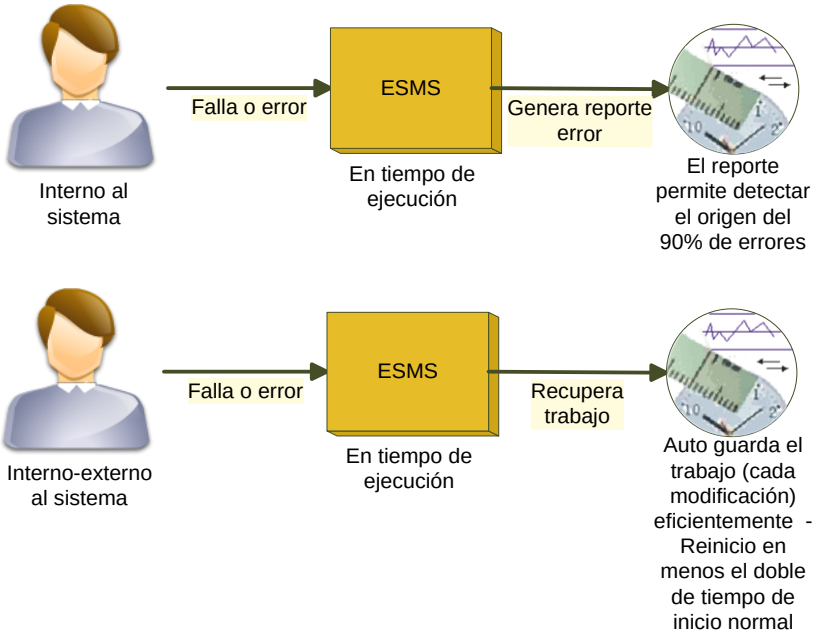


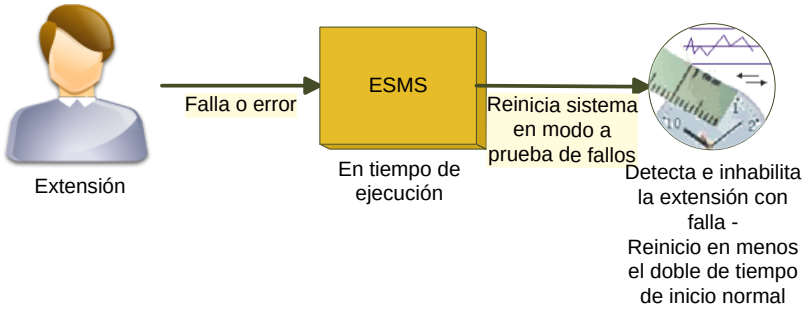
Fuente: Autor

5.3.2.2 Fiabilidad (Reliability)

Se evalúa midiendo la frecuencia y la gravedad de los fallos, la exactitud de las salidas (resultados), el tiempo medio de fallos (TMDF), la capacidad de recuperación de un fallo.

Figura 13. Escenarios de Fiabilidad



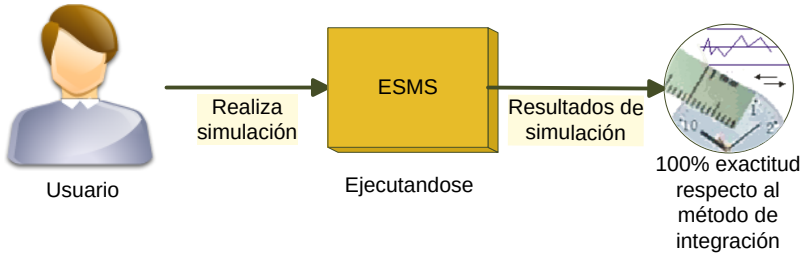


Fuente: Autor

5.3.2.3 Desempeño (Performance)

Es el grado en el cual un sistema o componente cumple con sus funciones designadas, dentro de ciertas restricciones dadas, como velocidad, exactitud o uso de memoria. El desempeño de un sistema se refiere a aspectos temporales del comportamiento del mismo. Se refiere a la capacidad de respuesta, ya sea el tiempo requerido para responder a aspectos específicos o el número de eventos procesados en un intervalo de tiempo. Además, se refiere a la cantidad de comunicación e interacción existente entre los componentes del sistema. El rendimiento se mide por la velocidad de procesamiento, el tiempo de respuesta, el consumo de recursos, el rendimiento efectivo total y la eficacia.

Figura 14. Escenarios de desempeño

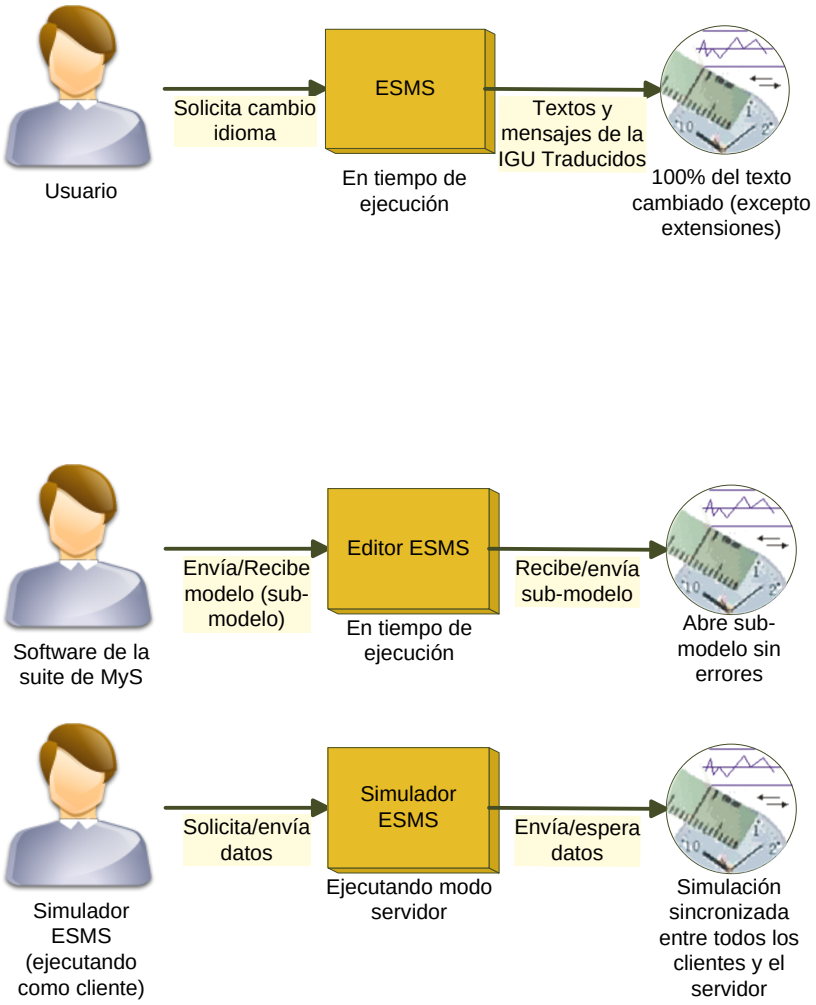


Fuente: Autor

5.3.2.4 Interoperabilidad (Interoperability)

Capacidad de operar en conjunto con otros software

Figura 15. Escenarios de interoperabilidad

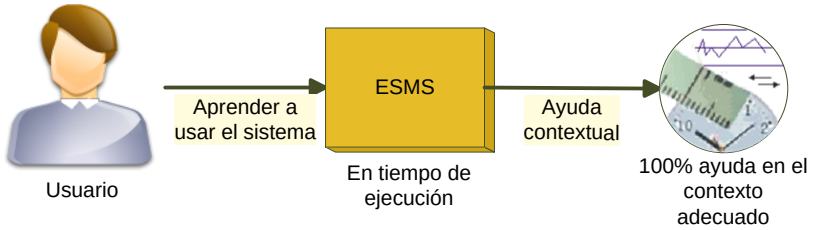


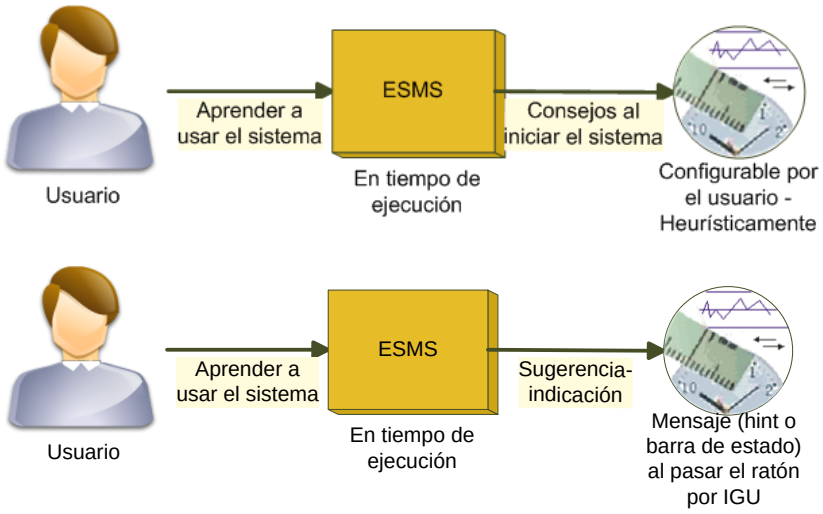
Fuente: Autor

5.3.2.5 Usabilidad

Relacionada con qué tan fácil es que el usuario realiza una tarea deseada y el tipo de soporte a usuario que provee el sistema.

Figura 16. Escenarios de usabilidad

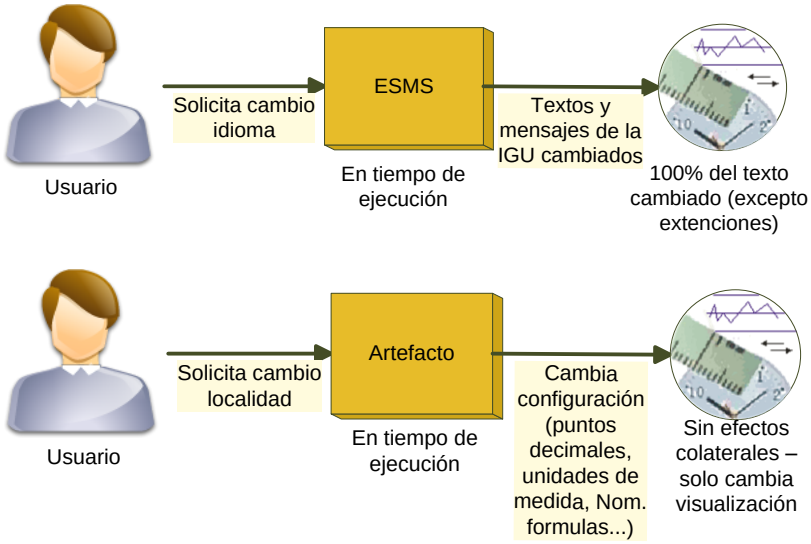




5.3.2.6 Internacionalización - Regionalización (localibity)

Disponibilidad para adaptarse a otras regiones (idioma, formatos de número y moneda).

Figura 17. Escenarios de internacionalización



Fuente: Autor

5.4 COMUNICACIÓN DE REQUISITOS

Porque el desarrollo del ESMS es en comunidad, la publicación, distribución y discusión de los requisitos al igual que otros tipos de documentación, se realiza través de la plataforma de comunicación de la comunidad. Específicamente, este tipo de documentos que deberán ser discutidos y modificados por la comunidad y al mismo tiempo son fuente de información para la misma, se publicarán por medio de un Wiki (ver capítulo 9). También se deberá publicar los archivos fuente de los modelos (por ejemplo archivos de StarUML para modelos de casos de uso).

5.5 ACORDAR LOS REQUISITOS

La obtención inicial de requisitos se ha acordado con la institución promotora, el grupo SIMON de investigación. Como se mencionó los requisitos han sido publicados en la plataforma de comunicación de la comunidad, por medio de la cual se realizarán los ajustes necesarios a los requisitos durante la realización de las iteraciones del proceso de desarrollo.

5.6 EVOLUCIÓN DE LOS REQUISITOS

Para manejar la evolución de los requisitos y de los documentos asociados se hace uso de Wikis (ver capítulo 9). Esta herramienta implementa un control de los cambios en las publicaciones realizadas, permitiendo regresar a una versión anterior del documento en caso de ser necesario. El control de versiones de los documentos fuente de los modelos (así como el código fuente del desarrollo) se realizará por medio del software subversión (ver capítulo 9). Subversion puede acceder al repositorio a través de redes, lo que le permite ser usado por personas que se encuentran en distintos ordenadores lo cual facilita el trabajo colaborativo (Collins-Sussman, y otros).

La evolución de los requisitos estará a cargo del equipo de desarrollo y los cambios serán supervisados y autorizados por el comité coordinador de la comunidad desarrolladora del ESMS.

CAPÍTULO 6. ARQUITECTURA DEL ESMS-MI

El presente capítulo corresponde al desarrollo del objetivo:

> *Continuar con la especificación de los requerimientos funcionales y elaborar la arquitectura (diseño de alto nivel) del Entorno Software de Modelamiento y Simulación de Modelos Integrados (ESMS-MI), garantizando la integración entre la Dinámica de Sistemas y otras herramientas matemáticas*[22]*.*

Este capítulo se centra en la elaboración de la arquitectura del sistema.

[22] Continuar el desarrollo realizado por la tesis "Diseño de una arquitectura para un entorno de modelamiento-simulación y creación de un proceso para su desarrollo por una comunidad (I+D)".

6 ARQUITECTURA DEL ESMS-MI

6.1 INTRODUCCIÓN

Durante muchos años la arquitectura software permaneció oculta dentro de los documentos de diseño de software, emergiendo de este alrededor del año 1970 que se hace uso del término para referirse a estructuras de software (Bass, y otros, 2007). Pero es solo hasta principios de los 90's que ésta comienza a tomar un papel importante en el proceso de desarrollo de software. A los años noventa se le denomina "la década de la arquitectura software" (Foundations for the study of software architecture, 1992), y es desde el año 1998 que el "Software Enginering Institute (SEI) y otros organismos comenzaron a elaborar métodos específicos de procesos de ingeniería que sistematizan el rol de la arquitectura en la totalidad del proceso" (Reynoso, 2004). Existe un número creciente de corrientes arquitectónicas, estilos, patrones arquitectónicos los cuales han atraído las miradas de las facultades de ciencias de la computación (Ej. Escuela de ciencias de la computación de la Carnegie Mellon University), grupos de investigación (Ej. Grupo ABLE, The USC Software Architecture), centros para el desarrollo para la ingeniería de software (Ej. SEI) y empresas del sector privado (Ej. Bredemeyer Consulting), entre otros.

El presente capítulo describe el proceso de desarrollo de la arquitectura del ESMS-MI, a partir del levantamiento de los requisitos y la revisión de diversos patrones arquitectónicos disponibles en la literatura. Los resultados obtenidos durante el proceso de desarrollo de la arquitectura, son presentados a lo largo de esta sección.

6.2 DESARROLLANDO LA ARQUITECTURA

Por ser una disciplina relativamente nueva, sobre la arquitectura software existen diferentes corrientes y taxonomías, creando un espectro de definiciones y métodos que orientan el desarrollo de la misma. El SEI realiza una compilación extensa de definiciones de arquitectura software que puede ser consultada en línea en (SEI, 2009). A continuación, se reproduce la definición de la IEEE y la de Paul Clements:

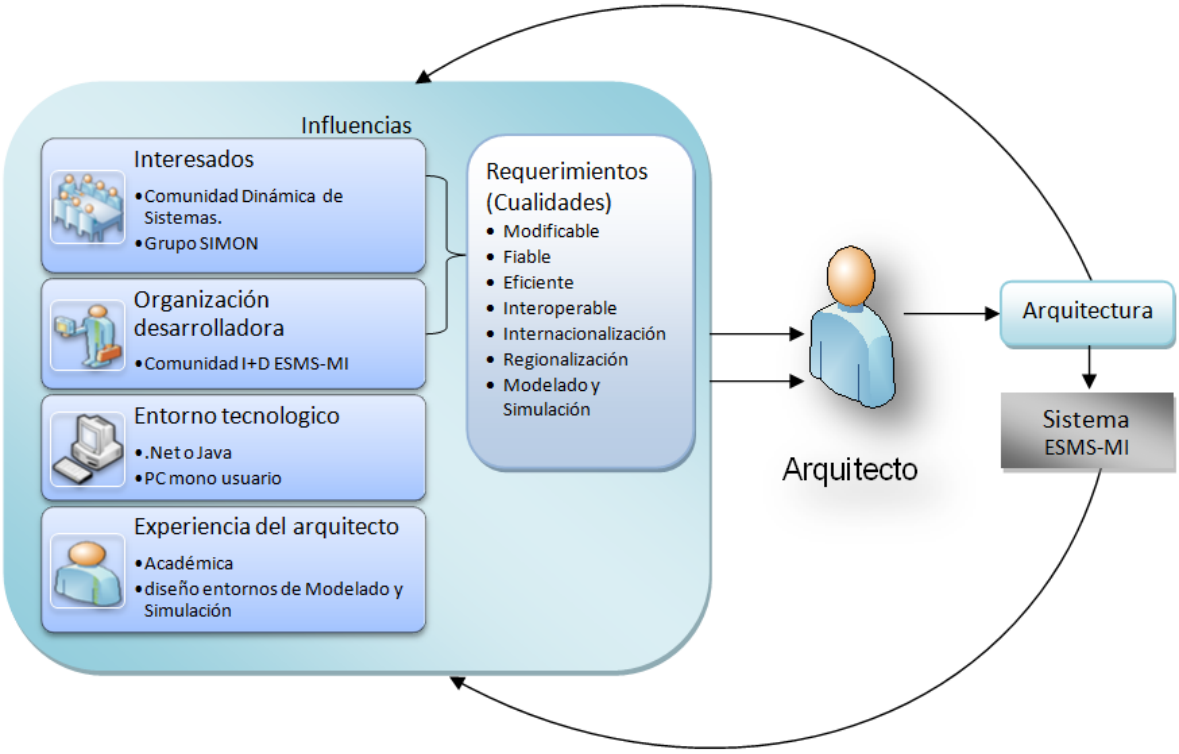
"La Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución." (IEEE Computer Society, 2000).

"La Arquitectura Software de un programa o sistema es la estructura o estructuras del sistema, la cual comprende los elementos de software, las propiedades visibles desde el exterior de estos elementos y las relaciones entre estos" (Bass, y otros, 2007).

De igual manera, existe un nutrido número de métodos como son el Architecture Based Design (ABD), Software Architecture Analysis Method (SAAM), Quality Attribute Workshops (QAW), Quality Attribute-Oriented Software Architecture Design Method (QASAR), Attribute-Driven Design (ADD), Architecture Tradeoff Analysis Method (ATAM), Active Review for Intermediate Design (ARID), Cost-Benefits Analysis Method (CBAM), Family-Architecture Analysis Method (FAAM), Architecture Level Modifiability Analysis (ALMA), y Software Architecture Comparison Analysis Method (SACAM). Igualmente, abundan los métodos y técnicas para su documentación (Reynoso, 2004).

La metodología ATAM propone un método para identificar las influencias y fuerzas ejercidas sobre un arquitecto las cuales determinan sobre la arquitectura diseñada. El Architecture Business Cycle (ABC) es un mecanismo para identificar y presentar los factores que influyen al arquitecto al momento de diseñar una arquitectura. La **Figura 18** presenta el ABC para el ESMS-MI.

Figura 18. Architecture Business Cycle para el ESMS-MI



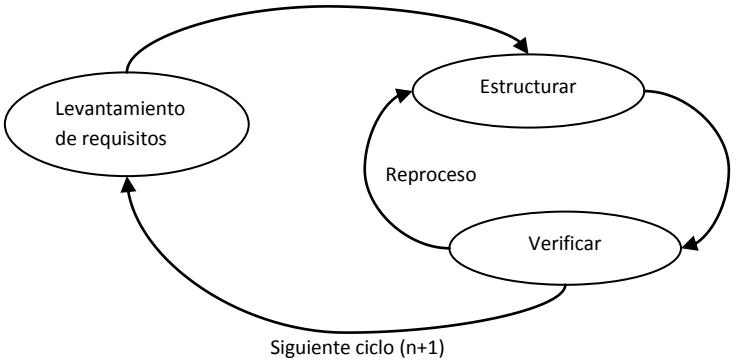
Fuente: Autor

Entre las metodologías para la creación de arquitecturas encontramos al *Proceso Arquitectónico Visual* ™ (PAV). Esta metodología plantea que una arquitectura debe tener las siguientes características (Malan, y otros, 2004):

- Buena: técnicamente representada de forma sólida y clara.
- Correcta: reúne las necesidades y objetivos de los interesados (Stakeholders)
- Exitosa: es usada actualmente en el desarrollo de sistemas exitosamente.

La **Figura 19** muestra de manera general las actividades del proceso, el cual es iterativo presentando dos ciclos de trabajo.

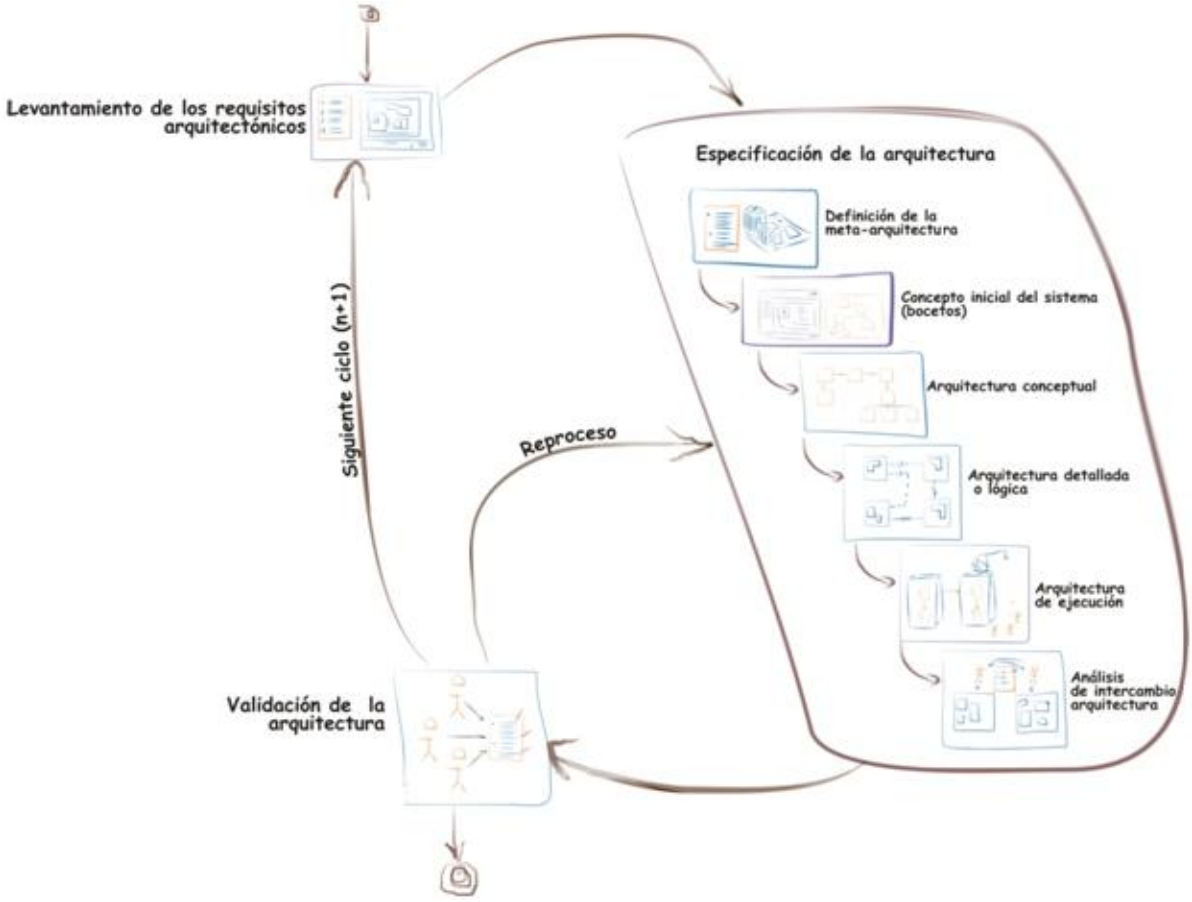
Figura 19. Proceso del PAV



Fuente: Autor

Para el desarrollo de la arquitectura del ESMS-MI se define el proceso descrito por medio de la **Figura 20**. Este proceso está basado en el proceso arquitectónico visual mencionado anteriormente y consta de 9 pasos o etapas.

Figura 20. Proceso de desarrollo de la arquitectura ESMS-MI



Fuente: adaptado a partir de (Malan, y otros, 2004)

A continuación, se describe cada una de las etapas del proceso desarrollo de la arquitectura:

1. **Levantamiento de los requisitos arquitectónicos**: es el conjunto de requisitos del sistema determinados por su relevancia arquitectónica. Las cualidades del sistema que tienen importancia arquitectónica y por otra parte, Los requisitos funcionales más relevantes23 (Malan, y otros, 2004). Las fuentes de información para esta actividad son: los interesados en un sistema los documentos de análisis e ingeniería de requisitos la experiencia del arquitecto, y el entorno del sistema. Como producto de esta etapa se obtiene un documento (ver **Anexo A**) con el conjunto de requisitos arquitectónicos especificados por casos de uso (requisitos funcionales) y escenarios (requisitos no funcionales)

2. **Especificación de la arquitectura:** En esta etapa se desarrolla la arquitectura del sistema. Esta etapa se compone de un conjunto de sub-etapas:
    a. **Definición de la meta-arquitectura:** La meta-arquitectura es un conjunto de decisiones de alto nivel que influirán fuertemente a la estructura del sistema. En este paso, se formula la visión arquitectónica, actúa como un faro que guía las decisiones durante el resto de la estructuración del sistema. Además, se define el estilo arquitectónico, conceptos clave, mecanismos y principios que guiarán al equipo de arquitectura durante los pasos siguientes de la estructuración. (Malan, y otros, 2004).
    b. **Concepto inicial del sistema (bocetos)**: en este paso se realizan los diseños preliminares de pantalla. Aquí no se pretende realizar diseños detallados de la interfaz de usuario, sólo se desarrollan algunos bocetos de lo que podría ser el nuevo sistema, con el fin de identificar relaciones entre requisitos funcionales y no funcionales. También, permite identificar cuáles requisitos funcionales son relevantes para la definición de la arquitectura. Estos bocetos se podrían ver como un primer prototipo con el cual los interesados del proyecto discuten el documento de los requisitos. Este recurso también permite la puesta a punto de algunos aspectos de la meta-arquitectura como los conceptos y mecanismos.
    c. **Arquitectura conceptual:** El propósito de la arquitectura conceptual es concentrarse en una descomposición apropiada del sistema antes de ahondar en los detalles de la especificación de interfaces y tipos de información. (Malan, y otros, 2004). Se deben identificar

[23] se debe tener en cuenta que la estructura del sistema falla si no soporta los servicios o funcionalidades que el usuario valora, o si las cualidades asociadas con esta funcionalidad inhiben al usuario para realizarla o descuide otros objetivos de las partes interesadas

principalmente los componentes, sus responsabilidades y las interconexiones entre estos (colaboradores).

d. **Arquitectura detallada o lógica:** La arquitectura lógica es la especificación detallada de la arquitectura, define de forma precisa los mecanismos de conexión y las interfaces entre componentes. Este paso parte de la arquitectura conceptual la cual puede ser redefinida en este paso. La arquitectura lógica puede contener una descripción resumida de los servicios que proveen los componentes, así como la descripción de las operaciones, sus precondiciones y poscondiciones y las restricciones para cada operación. Modelar la conducta dinámica de los componentes pude ser útil al momento de definir la arquitectura lógica. (Malan, y otros, 2004).
e. **Arquitectura de ejecución:** Una arquitectura de ejecución es creada para sistemas concurrentes o distribuidos. Corresponde a una vista de los componentes en término de los procesos del sistema físico (nodos del sistema), con principal atención en cuestiones tales como el rendimiento y la escalabilidad. (Malan, y otros, 2004).
f. **Análisis de intercambio arquitectura:** en este paso se analizan diferentes alternativas de arquitectura, reconociendo que cada una de estas producen diferentes grados de satisfacción de los requerimientos arquitectónicos. (Malan, y otros, 2004).

A lo largo de la etapa de especificación de la arquitectura se debe realizar la documentación de la arquitectura. Tenga en cuenta que la arquitectura también es un medio de comunicación, documentar la arquitectura de manera clara y comprensible, sirve como medio de comunicación entre perfiles técnicos y no técnicos de los interesados del proyecto (Anacleto, 2009).

De la etapa especificación de la arquitectura se debe generar toda la documentación necesaria para describir la arquitectura.

3. **Validación de la arquitectura**: Luego de estructurar la arquitectura, ésta pasa a la etapa de evaluación respecto a los requisitos definidos en la primera etapa. Con el fin de favorecer a la objetividad es recomendable que esta etapa sea realizada por personas ajenas al proceso de construcción de la arquitectura.

6.3 LEVANTAMIENTO DE LOS REQUISITOS ARQUITECTÓNICOS

Los requerimientos arquitectónicos son un conjunto de requerimientos del sistema determinados por su relevancia arquitectónica.

Puesto que uno de los objetivos de la presente tesis es continuar el levantamiento de los requisitos del sistema, el desarrollo de las actividades relacionadas con éste se presenta en un capítulo independiente (ver capítulo 5). Así, en este apartado

sólo se reproducirán los requisitos que son relevantes para la arquitectura del ESMS-MI. En cuanto a los requisitos de calidad se presenta una lista priorizada en orden de importancia para la determinar la arquitectura. El desarrollo del ESMS-MI también debe favorecer otros requisitos de calidad (como en todo desarrollo software) pero no influyen significativamente (poca relevancia arquitectónica) en la estructuración del sistema.

Modificable (Modifiability): Facilidad para el cambio

Fiabilidad (Reliability): Se evalúa midiendo la frecuencia y gravedad de los fallos, la exactitud de las salidas (resultados), el tiempo medio de fallos (TMDF), la capacidad de recuperación de un fallo.

Eficiencia (Efficiency): La cantidad de recursos informáticos y de código necesarios para que un programa realice su función.

Interoperability: El esfuerzo necesario para localizar y arreglar un error en un programa o realizarle cambios.

Internacionalización - Regionalización (localibity): disponibilidad para adaptarse a otras regiones (idioma, formatos de número y moneda).

En cuanto a las funcionalidades que son relevantes arquitectónicamente se definen los escenarios presentados por medio de casos de uso en la **Figura 21**

Figura 21. Requisitos funcionales de relevancia arquitectónica

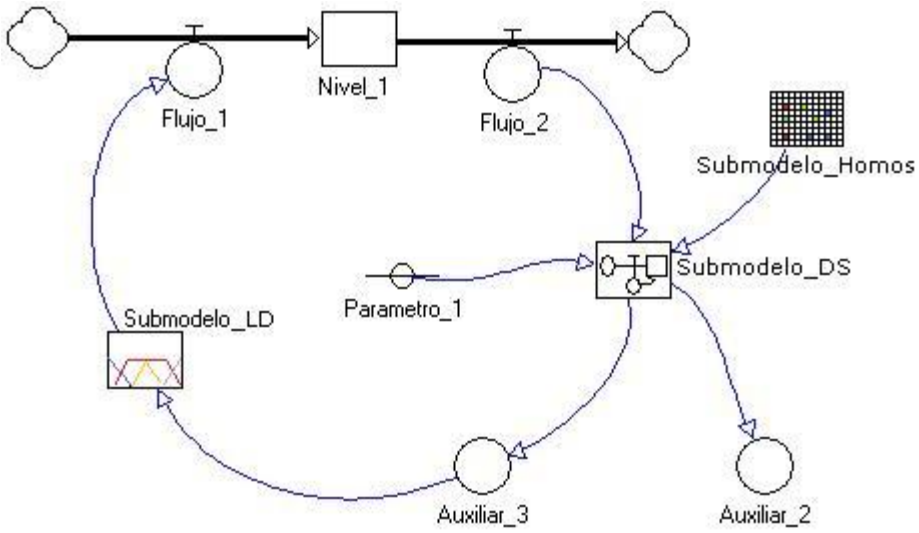
Fuente: Autor

6.4 CONCEPTUALIZACIÓN Y CONCEPTO INICIAL

6.4.1 Modelos integrados

El ESMS-MI, es un software que una vez desarrollado, integrará un conjunto de útiles para apoyar el modelado y la simulación de modelos expresados por medio de diferentes herramientas matemáticas, como la Lógica Difusa (LD), Redes Neuronales (RN), agentes, entre otras. Estos modelos se integran por medio de la D.S, convirtiéndoles en submodelos de un modelo más general.

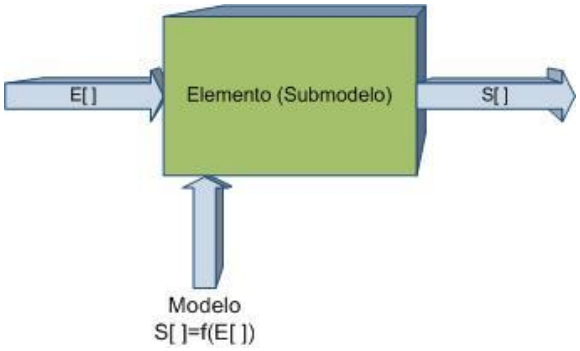
Figura 22. Modelo y submodelos integrados



Fuente: Autor

Cada uno de los submodelos actúa como un elemento dentro del modelo en D.S. En la **Figura 22** se aprecia que el Submodelo_DS recibe un conjunto de entradas, las cuales son procesadas para generar un conjunto de salidas (ver **Figura 23**).

Figura 23. Submodelo visto como un elemento



Fuente: Autor

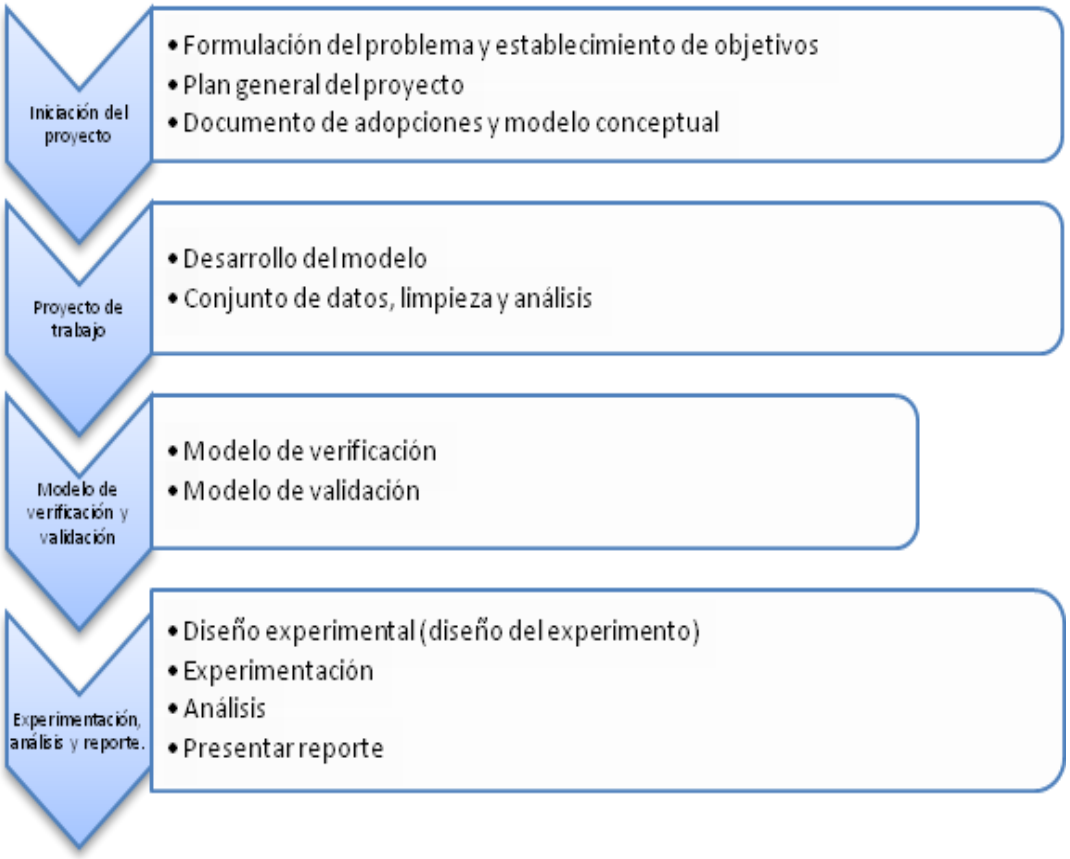
El elemento representa un submodelo o subsistema. Los Subsistemas son sistemas contenidos en otro sistema más general. El subsistema es representado por un elemento del sistema, pero no es una caja negra ya que el usuario debe poder acceder al subsistema que este representa. Un subsistema puede estar representado por medio de otro lenguaje *Y* diferente al lenguaje *X* del sistema que lo contiene.

6.4.2 Proceso de modelado y simulación

En procura de encontrar elementos comunes en los procesos de modelado y simulación, se ha realizado una revisión a diferentes autores. A continuación, se presentan los diagramas que representan cada uno de los procesos.

A continuación, se presenta el conjunto de fases y pasos del proceso de modelado y simulación propuestos por (Introduction to modeling and simulation, 2005) los cuales son considerados como necesarios para realizar el proceso de manera satisfactoria. Ver **Figura 24**.

Figura 24. Fases del proceso de modelado y simulación



Fuente: adaptado de (Introduction to modeling and simulation, 2005)

Para (Introduction to simulation, 1992) el proceso de simulación consta de 12 pasos, presentados en la **Figura 25**.

Figura 25. Pasos del proceso de modelado y simulación

Fuente: adaptado de (Introduction to simulation, 1992)

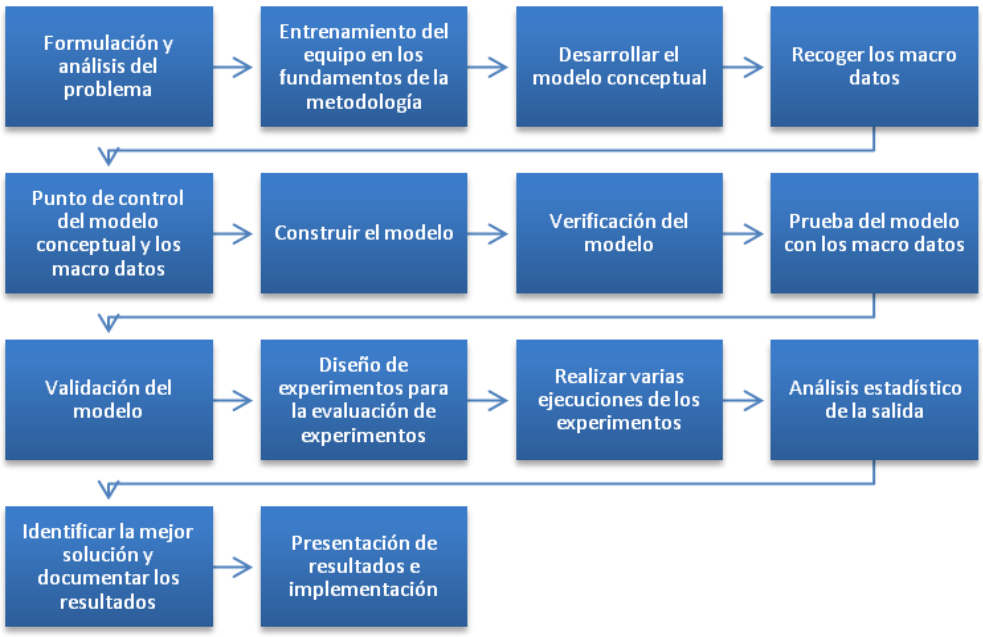
El proceso del uso de la simulación para analizar un sistema, (Introduction to simulation, 1995) los divide en los siguientes pasos lógicos:

Figura 26. Pasos del proceso de modelado y simulación

Fuente: adaptado de (Introduction to simulation, 1995)

La propuesta presentada por (Introduction to simulation, 1993) es una adaptación de los métodos de resolución de problemas utilizados en ingeniería.

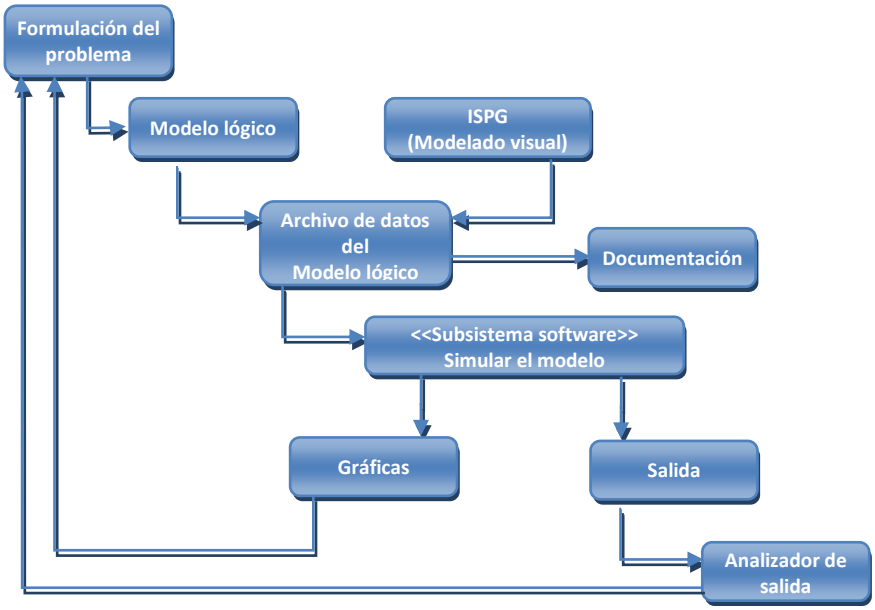
Figura 27. Pasos del proceso de modelado y simulación



Fuente: adaptado de (Introduction to simulation, 1993)

A continuación, se presenta el entorno de modelado propuesto por el proyecto CASM (Computer Aided Simulation Modelling) iniciado en 1982 por Balmer y Paul (The CASM enviroment revisited, 1994) (Modelling styles and their support in the CASM enviroment, 1987)

Figura 28. Entorno de modelado del proyecto CASM



Fuente: adaptado de (Modelling styles and their support in the CASM enviroment, 1987)

El modelado matemático se puede clasificar en dos enfoques, el enfoque conductual que intenta describir el comportamiento en función del comportamiento mismo y el enfoque estructural que describe, con una perspectiva sistémica, el fenómeno de la realidad en términos de las estructuras que conforman el modelo (Ingenieria de sistemas -realidad virtual y aprendizaje-, 2002). El grupo SIMON, ha trabajado principalmente en dos corrientes del modelado de enfoque estructural, la D.S y el MBOR.

El proceso de construcción de modelos en el Modelado Basado en Objetos y Reglas (MBOR) consta de cuatro acciones, las cuales se muestran en la **Figura 29**.

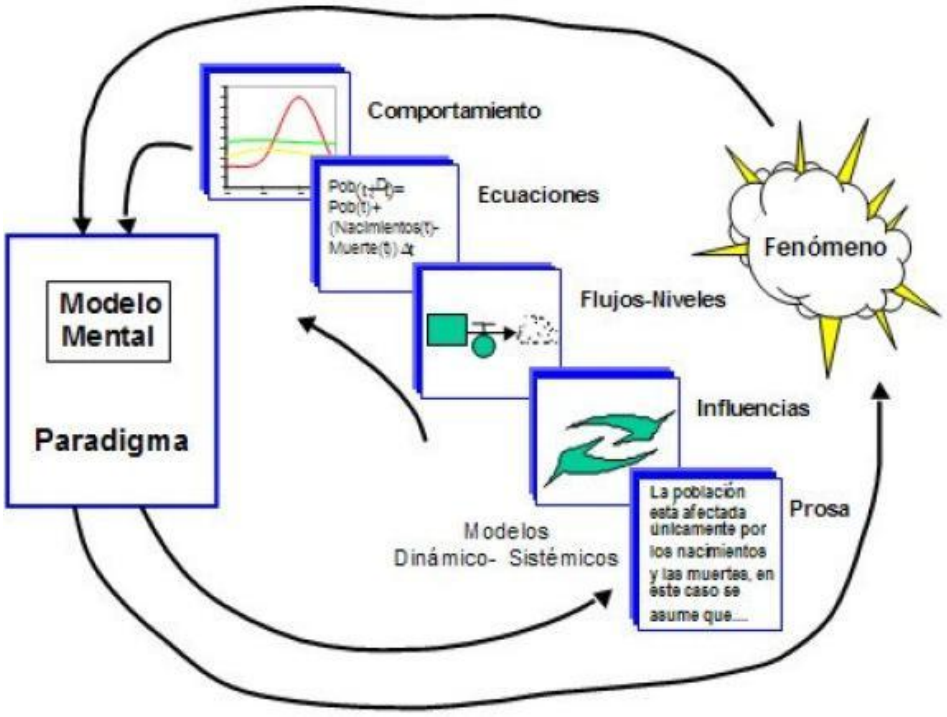
Figura 29. Etapas del MBOR

Fuente: Autor

El paradigma dinámico–sistémico es un paradigma de modelado que estudia una realidad o fenómeno representándolo como un sistema. Uno de los útiles del paradigma dinámico-sistémico es la D.S, la cual “ofrece herramientas metodológicas para expresar el modelo mental mediante un modelo visible…”, con estas herramientas se hace posible “desarrollar el proceso dirigido de reformulación del modelo mental hacia una mejor comprensión dinámico-sistémica del fenómeno” (Andrade Sosa, y otros, 2001).

Este proceso se presenta en (Andrade Sosa, y otros, 2001) como el transformar nuestros modelos mentales por medio de la D.S, a través del uso de los lenguajes mostrados en la **Figura 30**

Figura 30. Modelado y simulación con DS

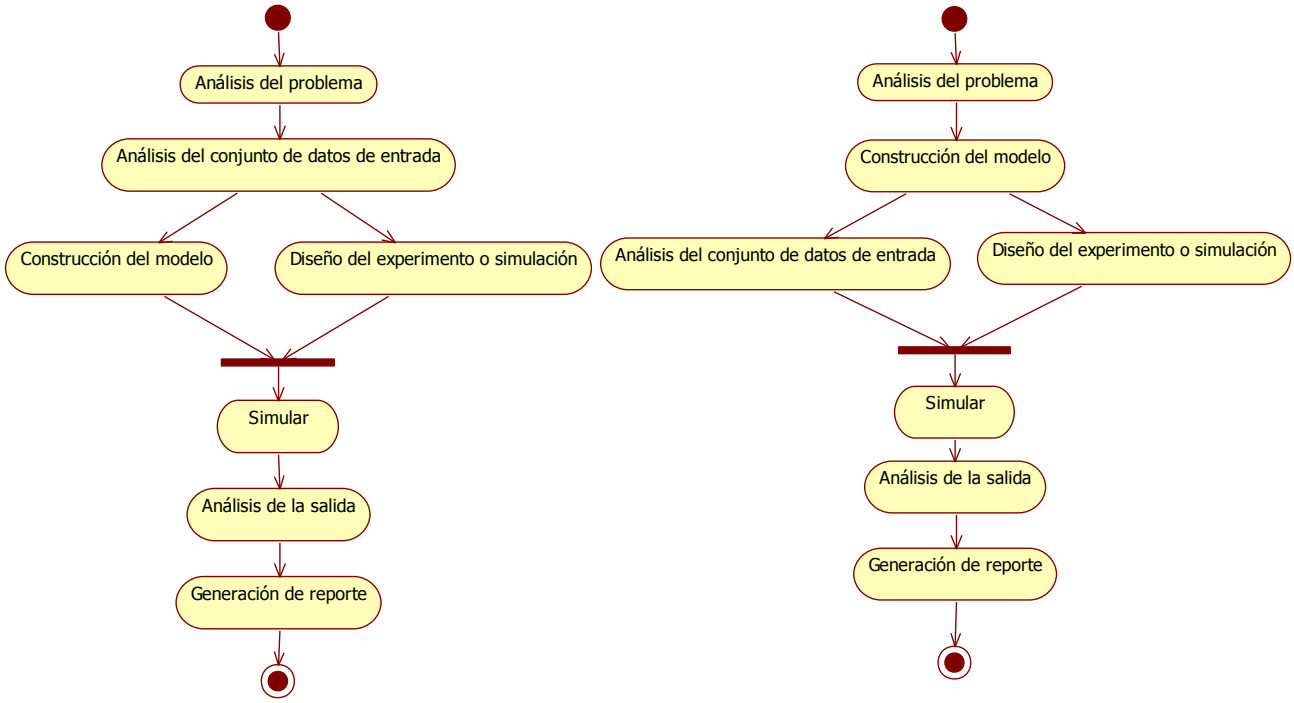


Fuente (Andrade Sosa, y otros, 2001)

El ciclo interno denominado ciclo de modelado, simulación e interpretación de resultados.

Teniendo en cuenta las actividades planteadas por los diferentes autores, se presenta el esquema de la **Figura 31**"**Proceso de modelado y simulación generalizado**", en el que se define las etapas, de manera general, que involucra el proceso de modelado para el enfoque conductual (**Figura 31**A) como para el enfoque estructural (**Figura 31**B)

Figura 31. Proceso de modelado y simulación generalizado



A. Enfoque conductual B. Enfoque estructural

Fuente: Autor

Este proceso puede ser *lineal* o *cíclico* dependiendo de la metodología empleada; a continuación se describen las actividades que se realizan en cada una de estas etapas.

En la etapa de ***análisis del problema*** existen diferentes herramientas que ayudan al análisis de una situación o fenómeno. Estas son herramientas (en varios casos) para elaboración de organizadores gráficos, representaciones generalmente visuales que organizan ideas de manera gráfica, como son los diagramas de influencias, mapas mentales, diagramas causa-efecto, mapas conceptuales, entre otros. Para el caso de D.S se cuenta con al menos dos herramientas como son: definir el modelo en prosa (texto) y los diagramas de influencias (gráfico).

En la etapa de ***construcción del modelo*** se deben brindar las herramientas para su creación ya sea de forma gráfica, un lenguaje matemático o cualquier tipo de representación. Este tipo de herramientas depende del paradigma empleado para modelar el fenómeno o problema de estudio. Generalmente, debe existir un lenguaje formal y herramientas visuales que permiten diagramarlo por medio de una representación gráfica, la cual es traducida al lenguaje formal antes mencionado. En algunos casos, existen diferentes vistas o representaciones del mismo modelo, o vistas parciales de algunos elementos que lo conforman. En este sentido (Problems for the future of system dynamics, 1998) manifiesta la

necesidad de tener herramientas que faciliten el entendimiento de modelos complejos; lo cual se ve reflejado en la dificultad que tienen los autores de artículos de la System Dynamic Review, para presentar modelos complejos en forma comprensible. Herramientas como sectores, submodelos, presentación y análisis de ciclos y diagramas de influencias, pueden ser de gran ayuda para dicho propósito. Cuando en el proceso no se cuenta con una herramienta para la elaboración de los modelos (herramienta de modelado), esta etapa se divide en dos: construcción del modelo lógico y diseño y la elaboración del software

En esta etapa pueden existir herramientas que permitan identificar elementos que pueden ser parte del modelo desde las representaciones realizadas en la etapa de análisis. En el caso de la D.S se pueden encontrar herramientas para identificar ciclos de realimentación. Así mismo, incluye algunas verificaciones estáticas al modelo tales como la consistencia dimensional e identificación de referencias circulares.

La etapa de ***análisis del conjunto de datos*** de entrada implica actividades tales como la limpieza, preparación y análisis del conjunto de datos de entrada y la definición de los escenarios con los cuales se van a realizar los experimentos. En estos datos se encuentran constantes, datos de inicialización del sistema, definición de variables exógenas al sistema, etcétera.

En la etapa de ***diseño del experimento*** se encuentran todas las actividades relacionadas con la preparación de la simulación como son: diseño del experimento, seleccionar la forma de presentar los resultados de la simulación (tabla de datos, gráficas, animaciones, etcétera), seleccionar posibles escenarios de simulación, diseño del análisis de la salida. Otra actividad de esta etapa es la de definir los parámetros o condiciones de la simulación. Algunas de estas actividades pueden no realizarse, lo cual depende de la metodología y de las herramientas utilizadas para simular el modelo.

La etapa de ***simulación*** es el conjunto de actividades que se realizan para que a partir del modelo, escenarios y condiciones iniciales de simulación, se obtengan los resultados de ésta. Cuando el proceso de simulación se realiza por computador, esta actividad es automática o semiautomática y los resultados se presentan al usuario por medio de los mecanismos seleccionados en la etapa anterior. Durante la simulación se pueden realizar actividades de seguimiento a los resultados en la medida en que se obtienen (en caso que el proceso sea iterativo), los cuales pueden ser procesados para brindar información al usuario en tiempo real; igualmente, esta información puede ser procesada luego de ser realizada la simulación y presentada al usuario por demanda. Un ejemplo de esto, en el caso de la D.S., es la identificación automatizada de ciclos de realimentación dominantes en cualquier punto de la simulación (Sterman, 2000), mostrando como la no linealidad en el modelo, cambia la estructura dominante de realimentación.

En la etapa de ***análisis de salida*** se incluyen actividades como análisis e interpretación de los datos de salida o comportamiento del modelo y análisis de sensibilidad, algunas de estas actividades pueden ser automatizadas. En el caso

de la D.S, (Sterman, 2000) plantea la necesidad de un análisis de sensibilidad que incluya la identificación automática de la política más ventajosa tomada durante una simulación e identificación de los parámetros más influyentes. En esta etapa también se realizan actividades relacionadas con la validación y verificación del modelo como el "reality check" (chequeo de realidad), mencionada en (Sterman, 2000) e implementada en Vensim (Ventana Systems, 2007).

La última etapa, denominada ***generación de reporte***, consiste en la generación de la documentación e informe final. Abarca el proceso de modelado y simulación en su totalidad. Las herramientas del entorno deben dar soporte a esta característica, generando reportes automatizados sobre el proceso realizado, éstos serán insumos para la elaboración del informe final del proceso.

Al igual que las herramientas de generación de reportes, existe un conjunto de éstas que son transversales a todas las etapas del modelado y la simulación descritas anteriormente. Por ejemplo, herramientas que ayuden a comprender el comportamiento del modelo, cuyo desarrollo es el principal problema técnico por resolver en la simulación de modelos según (Problems for the future of system dynamics, 1998).

Este tipo de herramientas transversales, son las más indicadas para que sean desarrolladas como módulos o plugin, que se agregan al entorno y se comunican con los módulos nativos por medio de la interfaz de aplicación (API).

Un entorno software de modelado y simulación debe dar soporte al usuario durante cada una de estas etapas, brindando diferentes herramientas integradas a todo el proceso en un único entorno.

6.4.3 Framework de modelado y simulación

Junto con el ESMS-MI se debe diseñar una herramienta software para la documentación, organización, publicación y socialización de los modelos. Los modelos del ESMS-MI contienen información básica, sin embargo, en procura de llegar a una audiencia más amplia y hacer los modelos más accesibles (Problems for the future of system dynamics, 1998), se hace necesaria una documentación más completa y personalizada, orientada a sectores o comunidades con distintos intereses sobre un mismo modelo (educativo, científico, empresarial). La herramienta recibe como insumo un archivo con todos los datos del modelo y un reporte generado por el entorno; a partir de esto, permite al usuario agregar información relacionada con el modelo y que va dirigida a una comunidad específica.

Entre las características de esta herramienta se cuentan: agregar imágenes, texto, videos, tutoriales de cómo hacer un experimento o simulación con el modelo, administración de metadatos, creación de documentos, administración de modelos, permitir una navegación por la documentación secuencialmente como un libro o híper-textualmente, creación de paquetes de distribución para facilitar su distribución. Además, esta herramienta debe facilitar la publicación de los

contenidos en un servidor; el cual es accedido remotamente por clientes que pueden descargar dichos paquetes o subir nuevos paquetes, estos paquetes y sus modelos se deben poder administrar localmente en el equipo del cliente.

Además, de esta herramienta descrita en el párrafo anterior, el ESMS-MI debe trabajar en conjunto (Interoperabilidad) con otras herramientas, conformando una Suite de Herramientas y aplicaciones de Modelado y Simulación. La suite debe estar soportada en un marco de trabajo (Framework). La suite consta de herramientas y aplicaciones software interoperables, a continuación se presentan alguna de estas:

Tabla 2. Listado de posibles herramientas y aplicaciones de la suite de modelado y simulación

Entorno Software de Modelado y Simulación de Modelos Integrados ESMS-MS
Software para la documentación de modelos
Software para la administración, socialización, documentación de modelos y simuladores
Visor de simulaciones del ESMS-MI
Micromundos de Aprendizaje Dinámico Sistémico
Herramienta software para el modelado basado en objetos y reglas
Herramienta software para el desarrollo de controladores difusos.

El objeto de la presente investigación es el desarrollo de la arquitectura del ESMS-MI, el cual aporta el núcleo del Framework.

6.4.4 Prototipos y vistas

La forma de como vemos al mundo está determinada por el paradigma de pensamiento, este paradigma está fuertemente ligado al modelador y funciona como un filtro que nos permite ver la realidad de una manera u otra.

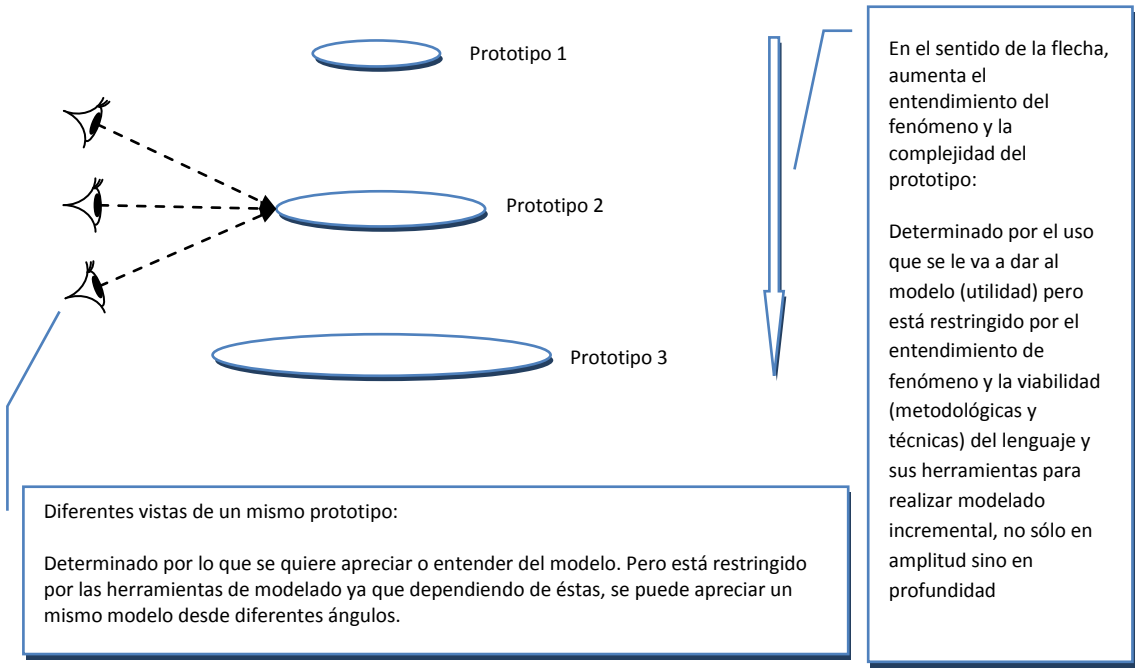
En el terreno del modelado surgen dos conceptos que muchas veces son confundidos y tomados como sinónimos, las vistas y los prototipos. Los prototipos son modelos de un mismo fenómeno que se diferencian entre sí, por las variables del fenómeno que se tienen en cuenta. Por otra parte, las vistas corresponden a las diferentes formas o perspectivas que tenemos de un mismo prototipo. Las vistas nos ayudan a ver características diferentes de un mismo modelo. Así como en los diferentes planos de la arquitectura de un edificio, existe la vista de las instalaciones eléctricas, de las tuberías, de la fachada, etc. Es posible que en algunas oportunidades se quiera observar el comportamiento dinámico del modelo, y en otras oportunidades queremos ver su comportamiento estático.

En D.S se cuenta con diferentes vistas de un prototipo, estas pueden ser expresadas en diferentes lenguajes (ver **Figura 30**). El lenguaje en prosa se centra en la definición del modelo y de sus restricciones expresadas en lenguaje natural.

Por su parte, el diagrama de influencias se centra en definir los elementos que hacen parte del sistema y las relaciones causales entre estos, es decir, cómo se influyen los elementos del modelo entre sí. El diagrama de flujo nivel se centra en las características propias de cada elemento, según su comportamiento dinámico, es decir, identificar si cada elemento es un Flujo de cosas, la acumulación de flujos (Nivel), una variable intermedia o una constante, entre otros. El lenguaje de las ecuaciones se centra en definir la relación entre los elementos que hacen parte del modelo, formalizadas en términos de expresiones matemáticas. El lenguaje del comportamiento permite ver el sistema desde múltiples vistas, las cuales expresan cómo es el comportamiento dinámico del modelo (sus diferentes variables) en la medida que avanza una simulación (en el tiempo).

Las vistas nos permiten ampliar la comprensión de un mismo modelo. Cada una de éstas permite observar diferentes aspectos del mismo modelo, posibilitando su estudio y entendimiento por parte del observador. A diferencia de los prototipos, que son versiones del mismo modelo que van contemplando nuevas características o desagregando las ya existentes. En la **Figura 32** se presenta algunas características que nos permiten distinguir entre vistas y prototipos.

Figura 32. Vistas y prototipos



Fuente: Autor

6.4.5 Mecanismos de agrupación de diagramas y elementos

Una de las dificultades con las que se enfrenta el modelado visual por medio de diagramas, es la organización de las estructuras (diagramas y elementos) que componen un modelo. El no existir mecanismos y herramientas que ayuden a dicha organización, un modelo suficientemente grande puede convertirse en todo un caos, entendible únicamente por su creador. Es por esto que en esta sección se examinarán mecanismos y herramientas que permiten realizar modelos más organizados y estructurados. Estos mecanismos de organización también deben favorecer uno de los objetivos fundamentales del ESMS-MI, la integración de elementos y submodelos.

Sector: parte de un diagrama cuyos elementos se encuentran en el mismo nivel que el diagrama principal.

Subdiagrama (submodelo): diagrama que hace parte del diagrama principal. Los elementos que lo componen están en una capa o nivel inferior al diagrama principal. Los submodelos son la base para la generalización de estructuras (estructuras genéricas).

Exógena: Elemento que representa un elemento externo al diagrama. El origen de sus datos no es de interés para el modelador.

Clon: es un link o enlace a otro elemento del modelo, este permite organizar relaciones entre elementos evitando cruces entre éstas, mejorando la presentación de un diagrama. También presta la funcionalidad de abstraer las características de un elemento, de esta manera puede jugar el rol de interfaz entre subdiagramas o submodelos, permitiendo tomar el valor de un elemento sin importar su tipo. Un tipo especial de clon es el cual hace conversiones del valor el cual representa, por ejemplo, el valor de origen puede ser real y el valor destino un entero, etcétera.

Estereotipo: es el rol elemento, diagrama o mecanismo de agrupación. Un mismo elemento puede jugar diferentes roles, por ejemplo un elemento de tipo auxiliar puede tener jugar el rol de una variable o una constante dependiendo del valor definido, en el caso de un mecanismo de agrupación puede tener diferentes roles para permitir el uso de diferentes jerarquías de diagramas.

La organización propuesta para los conceptos de un modelo se presenta en la **Figura 34**. Esta organización jerárquica propone que un modelo es un conjunto de vistas (expresadas por medio de diagramas), cada diagrama tiene elementos los cuales pueden estar agrupados ya sea en submodelos o sectores. Una vista puede contener un conjunto de capas las cuales filtran la información de la vista, permitiendo ver algunos elementos de ésta.

Antes de definir lo que es una vista, se presentan dos conceptos necesarios para su definición: lenguaje y diagrama.

Un lenguaje está conformado por un conjunto de signos o símbolos y unas reglas (sintaxis y semántica) para representar los elementos de una vista (Real Academia Española, 2009).

Un diagrama es un dibujo en el que se muestran las relaciones entre las diferentes partes de un conjunto o sistema (Real Academia Española, 2009).

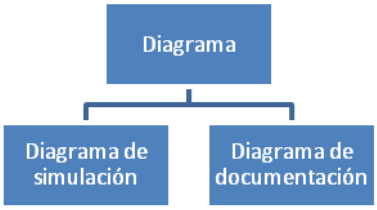
Las vistas representan el modelo "visto" desde una perspectiva diferente, permitiendo observar algunas características de dicho modelo (comportamiento estático, comportamiento dinámico, sistema eléctrico de una edificación, estructura de una edificación). Normalmente, el propósito de cada vista es la de analizar diferentes características del mismo modelo. Las vistas se relacionan entre sí, estas relaciones pueden ser restricciones (si existe un elemento *X* en la vista *A*, no es posible crear un elemento *Y* en la vista *B*) o implicaciones (al crear un elemento *X* en la vista *B*, se crea un elemento *Y* en la vista *A*). Las vistas pueden ser parciales o totales. Una vista es parcial si no representa a la totalidad del sistema-modelo, sino a un sector del mismo. Por lo tanto, varios diagramas corresponden a la misma vista, si representan parte del mismo sistema-modelo y están representados en el mismo lenguaje o lenguaje equivalente.

Una vista puede ser representada en diferentes lenguajes, sí y sólo sí, los lenguajes son equivalentes, en caso contrario los diagramas corresponden a vistas diferentes. Dos lenguajes son equivalentes si permiten expresar las mismas características que se quiere observar del sistema-modelo.

Decimos que un diagrama se puede traducir de un lenguaje a otro. Esta traducción puede ser de un lenguaje más específico (más general) a otro menos específico (más particular), o entre lenguajes equivalentes. En el primer caso para ir del diagrama expresado en un lenguaje B más general, al diagrama expresado en el lenguaje A más específico, se debe traducir utilizando un conjunto de reglas, pero para ir del diagrama en A al diagrama en B es necesario realizar actividades de diseño o modelado. En el segundo caso, cuando tenemos un diagrama en lenguajes equivalentes se puede traducir de A a B y viceversa aplicando un conjunto de reglas de traducción sin perder datos.

Existen dos tipos de diagramas, de simulación y de documentación (ver **Figura 33**). Los diagramas de simulación representan al modelo de simulación, puede estar conformado por submodelos enlazados entre sí o un modelo principal que integra a submodelos. Los modelos de documentación son modelos que tienen como función presentar información sobre el modelo, pero no representan al modelo de simulación.

Figura 33. Tipos de diagramas de modelos

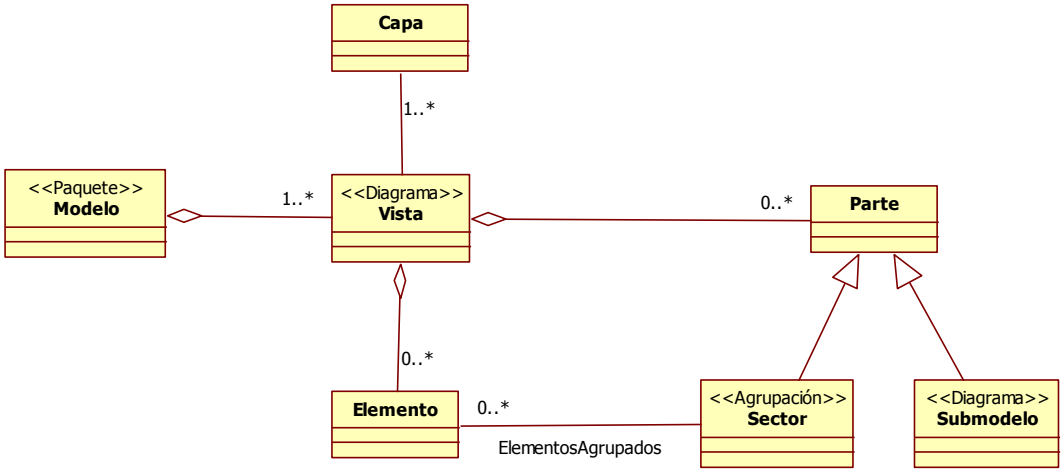


Fuente: Autor

Un ejemplo de diagrama de documentación y de simulación son los diagramas de influencias en las herramientas software Vensim y Powersim. En el primero se puede realizar un modelo y definirlo completamente por medio de un diagrama de influencia[24]. En PowerSim y Stella son diagramas informativos que hacen parte de la documentación del modelo.

Completitud de un modelo de simulación: un modelo está completamente definido cuando es expresado en su totalidad y puede ser simulado con unas condiciones iniciales, de frontera o escenario.

Figura 34. Organización de un modelo



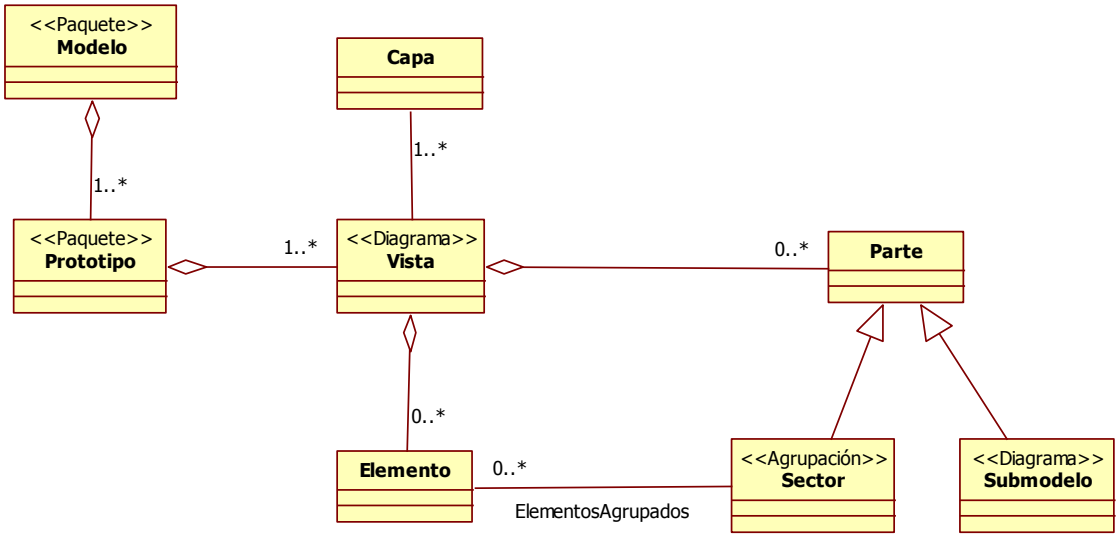
Fuente: Autor

Para contemplar diferentes prototipos de un mismo modelo, se agrega un elemento de agrupación de vistas el cual se denomina prototipo. Por lo tanto, la organización cambia a la presentada en la **Figura 35**. En esta nueva estructura, un

[24] También llamados diagramas causales

modelo está conformado por uno o varios prototipos, que a su vez contiene una o más vistas.

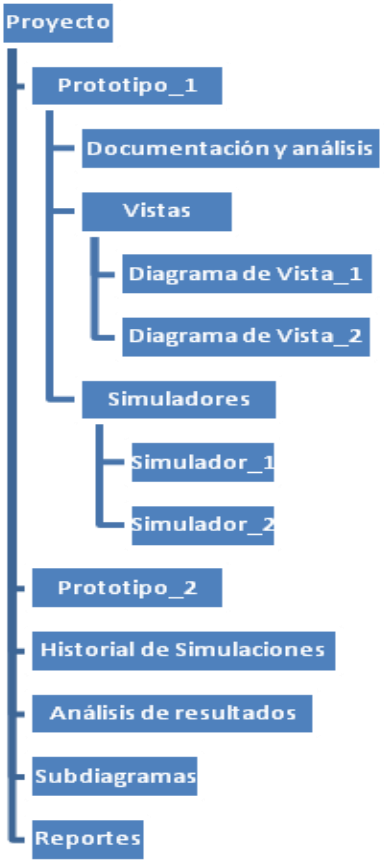
Figura 35. Organización de un modelo incluyendo prototipos



Fuente: Autor

Si se incluyen las demás etapas del proceso de modelado y simulación, esta agrupación debe ser extendida para abarcar otros elementos del proceso. Es así como surge la organización presentada en la **Figura 36**, la cual inicia con el elemento raíz llamado “proyecto”.

Figura 36. Estructura de un proyecto en el ESMS



Fuente: Autor

Escenarios

Por la naturaleza del modelado estructural, un modelo representa un fenómeno en términos de su estructura, por lo tanto, pueden existir diferentes instancias del fenómeno, las cuales se convierten en fenómenos concretos cuando es definido uno o varios escenarios específicos. En términos matemáticos se tomará el ejemplo del fenómeno denominado movimiento armónico simple. Este tipo de fenómeno se puede modelar por medio de ecuaciones diferenciales lineales de segundo orden (Purcell, y otros, 1992), más específicamente por medio de la ecuación:

$$\frac{d^2y}{dt^2} + B^2y = 0$$

Esta ecuación es la estructura del fenómeno.

Las condiciones iniciales

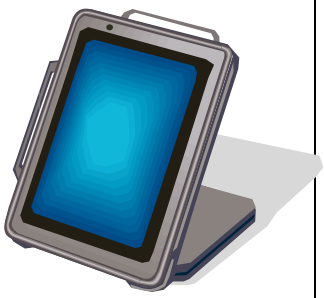
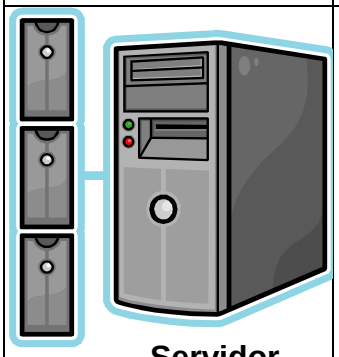
$$y_{(t=0)} = y_0 \quad \text{y} \quad y'_{(t=0)} = 0$$

Y el valor de la constante B, determinan un ***escenario*** del fenómeno, es decir, un caso concreto o específico del fenómeno.

Estas estructuras incluso pueden ser trasladadas a otros fenómenos que tienen características similares. Para el caso del ejemplo, encontramos que esta misma estructura puede ser empleada para modelar resortes, vibración de cuerdas y algunos circuitos eléctricos.

Entorno del sistema
El entorno o ambiente en el cual debe ejecutarse el software se define por las características del software

Tabla 3. Tipos de ambientes de trabajo

Escritorio 	Como un software mono usuario
Tablet PC 	Como software mono usuario para poder funcionar en un tablet PC. Se debe tener acceso a todas las funciones sin hacer uso del teclado (tablero con letras y números). En las propiedades de los elementos debe aparecer tableros para introducir fórmulas.
 **Servidor**	El motor de simulación ejecutándose en un ciclo infinito tomando datos de usuarios remotos.

Red 	Funcionando como Cliente-servidor. Es posible realizar diferentes animadores para un mismo modelo de simulación. Un PC es configurado como servidor en el que se ejecuta la simulación. Otros computadores se unen al servidor, según roles que se definieron previamente y los cuales han asignado a cada animador. Por medio de pasos de mensajes el servidor ejecuta una iteración y envía la información al resto de computadores, los cuales toman estos datos y realizan las respectivas animaciones o presentación de resultado. Luego los animadores de los clientes enviarán al servidor las modificaciones realizadas por los usuarios (interacción).

Fuente: Autor

6.5 BOCETOS DEL SISTEMA

En esta sección se presenta un boceto inicial del ESMS-MI, este boceto inicial consta de un conjunto de imágenes y un flujo de acciones, que en conjunto con los diagramas de casos de uso de escenarios y requisitos, conforman la primera aproximación al diseño del sistema. Las imágenes y procesos aquí presentados, no pretenden ser una representación fiel del sistema definitivo; tampoco pretende ser el primer prototipo del sistema. El boceto permite una primera visualización de lo que será el sistema software (visualización del sistema como sistema software, no una representación elaborada de diseño). De esta manera, el boceto se convierte en un instrumento para discutir los requisitos del sistema, los cuales fueron suscitados inicialmente en la etapa de análisis. Es así como el boceto se convierte en el punto de partida para realizar una profunda reflexión de los requisitos iniciales del sistema y su entorno, permitiendo adaptarlos, estimulando la aparición de nuevos requisitos y/o eliminando a requisitos innecesarios o riesgosos.

La interfaz de usuario que se presenta a continuación, son bocetos que sirven como guía para definir los mecanismos de interacción con el usuario. Por lo anterior, esta interfaz no corresponde al diseño de la interfaz final, etapa que debe ser asumida posteriormente y que cambia significativamente la presentación de la misma.

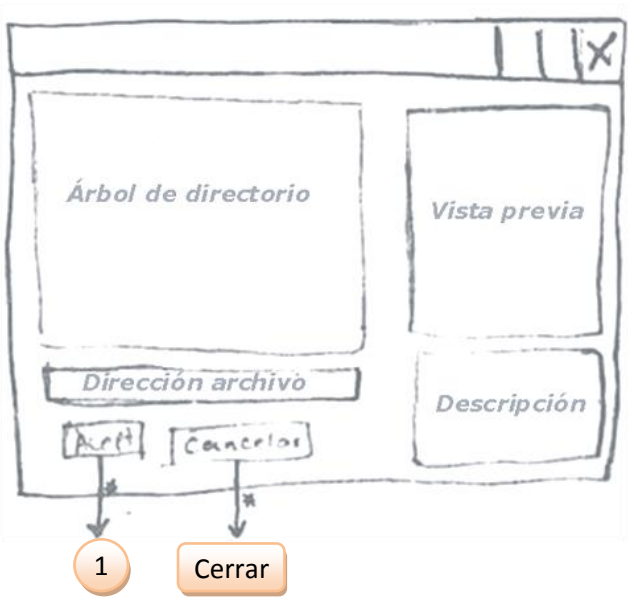
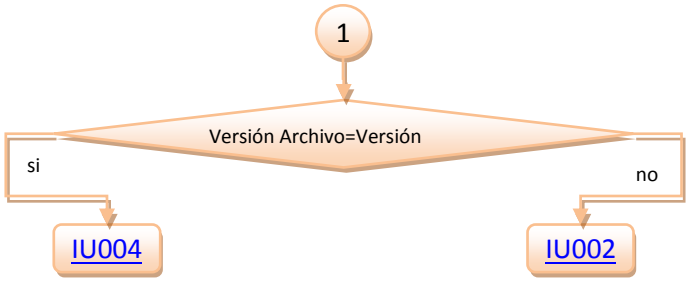
Esc = Escenario
IU = Interfaz de usuario
✸ = un asterisco indica “si el usuario hace clic”
✸✸ = dos asteriscos consecutivos indica “si el usuario hace doble clic”

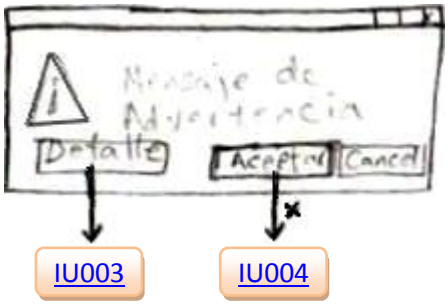
→

✸ = El asterisco con una flecha en la parte superior indica “si el usuario hace clic sostenido y arrastra”

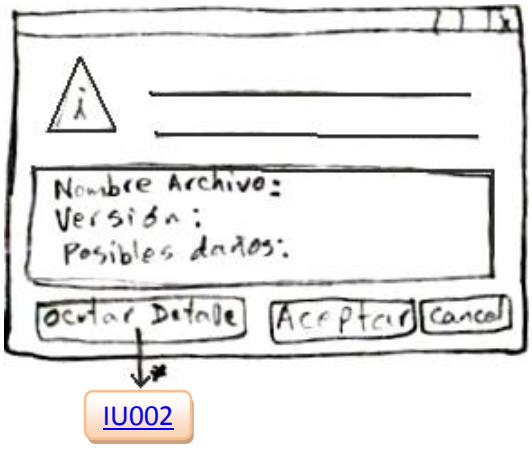
→ = La flecha indica transición a…

Tabla 4. Bocetos del sistema

Esc001: Abrir archivo ESMS-MI	
IU001	
	Esta ventana nos debe permitir abrir diferentes tipos de archivos: • Archivo de proyecto (Modelo + Simuladores, etc) • Archivo del Modelo F-N • Archivo del Modelo en DI • Archivo de Ecuaciones • Archivo de animadores • Paquete con proyectos prototipos
	El sistema comprueba si la versión del archivo corresponde con las versiones permitidas por la versión actual (el software puede tener compatibilidad con algunas versiones anteriores, pero puede estar condicionada a la pérdida de parte información, lo cual debe ser advertido al usuario)
IU002	



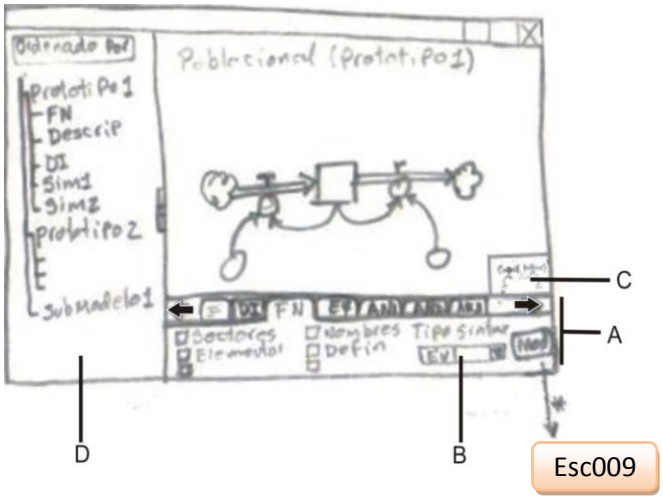
IU003



Amplía la IU002 para mostrar detalles del archivo que se está intentando abrir, con el fin de que el usuario se informe del porqué se está presentando este mensaje de advertencia. Igualmente, debe presentar información acerca de los inconvenientes que se pueden presentar al intentar abrir un archivo con una versión anterior.

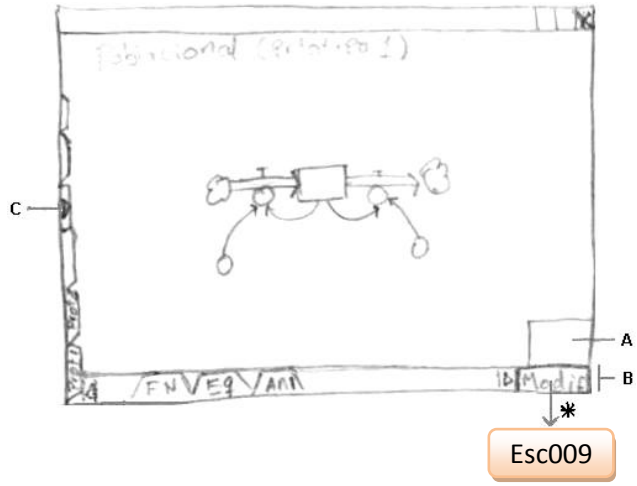
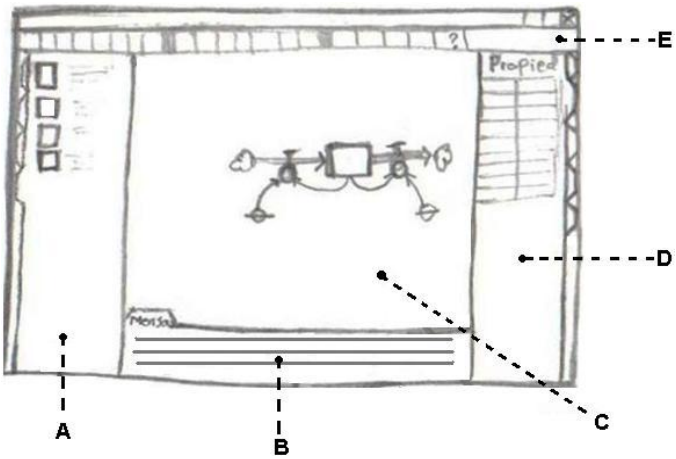
Esc004: Modelar

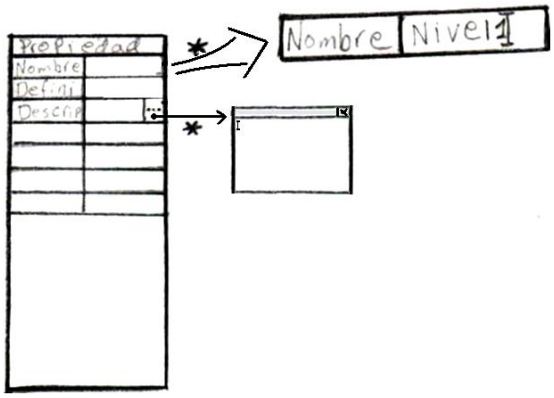
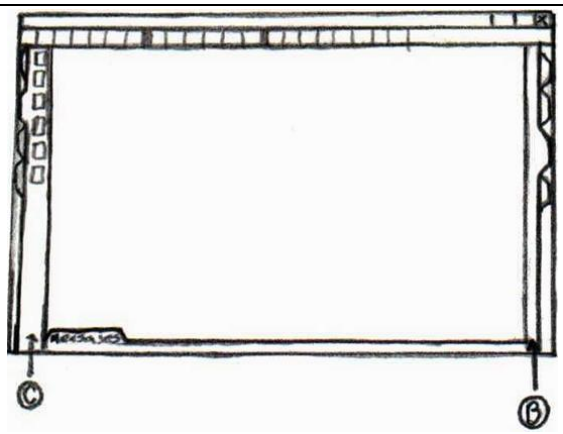
IU004



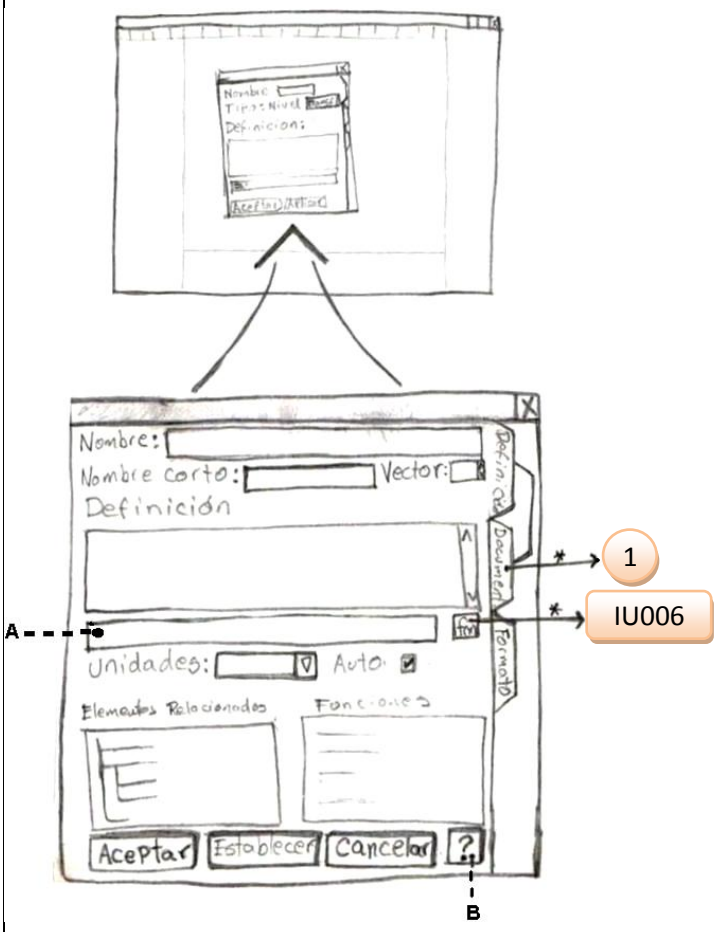
Esta IU permite navegar entre los diferentes diagramas y prototipos, visualizando el seleccionado por el usuario. Para modificar el diagrama se debe activar el modo de edición. Este concepto se basa en la premisa que, es más frecuente que un modelo se abra con el fin de verlo o examinarlo que con el fin de modificarlo.

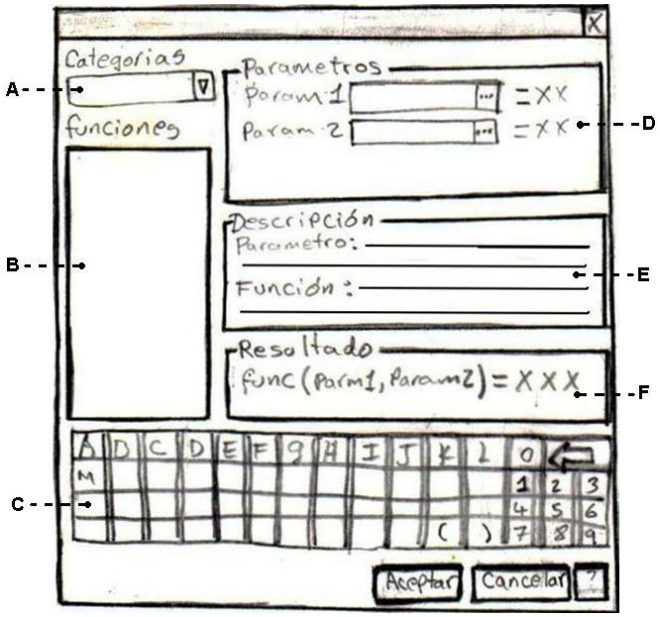
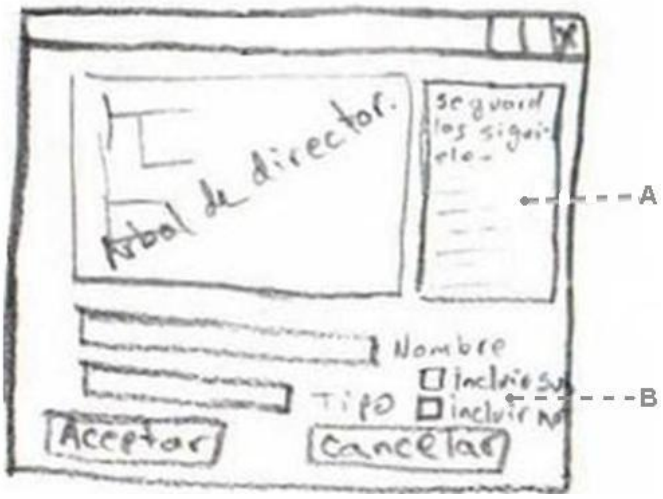
A → Banda que permite navegar entre los diferentes diagramas y simuladores de un mismo prototipo. Esta banda puede contener flechas de adelante y atrás para cuando existen muchos diagramas y/o simuladores. Además, contiene opciones de visualización del modelo, como mostrar/ocultar

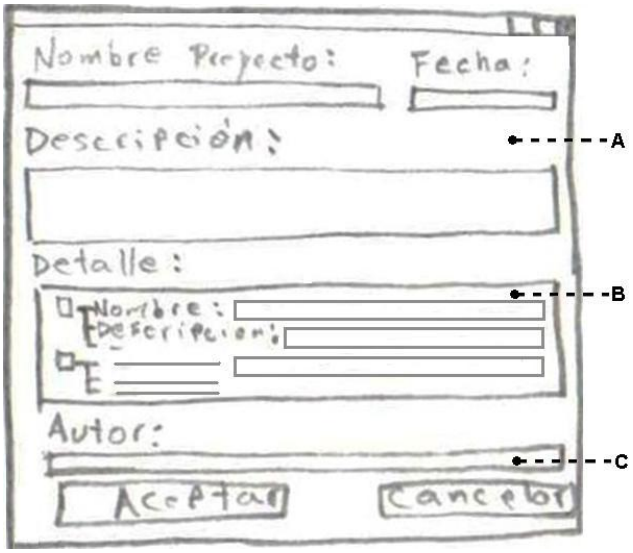
	capas. Esta banda contiene un botón el cual permite al usuario entrar en modo de diseño (modificar el diagrama). C → Zoom. Vista en miniatura del diagrama presentado, sirve para navegar fácilmente por modelos grandes. B → Tipo o estilo de sintaxis D → Árbol para navegar entre los diagramas, submodelos y demás carpetas organizadoras como lo son los prototipos.
IU004_1	
	A → Zoom. Vista en miniatura del diagrama presentado, sirve para navegar fácilmente por modelos grandes. B → Banda que permite navegar entre los diferentes diagramas y simuladores de un mismo prototipo. Esta banda puede contener flechas de adelante y atrás para cuando existen muchos diagramas y/o simuladores. C → Árbol para navegar entre los diagramas, submodelos y demás carpetas organizadoras como lo son los prototipos. En este caso se encuentra recogido.
IU009	
	A → Presenta los elementos disponibles para hacer el diagrama, estos elementos deben estar ordenados por categorías. Este panel se puede recoger (ver IU009_2). B → Panel para presentar mensajes, estos mensajes se agrupan en mensajes de verificación, historial de comandos, etc. C → Espacio destinado para realizar/editar el modelo, aquí se colocan los elementos desde el panel de elementos. D → Panel de propiedades del elemento seleccionado. En esta sección se apilan otros paneles

	como un panel para seleccionar elementos, paneles de herramientas (por ejemplo herramientas de alineación de elementos), entre otros. Este panel se puede recoger (ver IU009_2) E → Barra de herramientas que brinda acceso a las principales herramientas utilizadas por el usuario. Esta barra debe ser personalizable.
IU009_1	
	Detalle del panel de propiedades de los elementos. En este panel se pueden editar las propiedades haciendo clic en los cuadros de textos. Deben existir algunos asistentes para introducir textos largos o listas de valores. Las propiedades complejas que se componen por otras propiedades, deben ser presentadas en forma de árbol y deben poderse recogerse o comprimirse.
IU009_2	
	Sí los paneles C y B están recogidos y el usuario hace clic en las fichas (o lengüetas), el panel se extiende sobre el modelo (no se recoge). Estos paneles deben tener un botón para poder ser fijados (inmovilizar paneles).

Esc005: Editar propiedades elemento

IU005	
	Las propiedades de los elementos se muestran en el Panel de Propiedades de un elemento, pueden tener Subpropiedades (propiedades complejas) y tener editores de propiedades. No se contempla el uso de Vectores Multidimensionales, para definir elementos, con el objetivo de no agregar complejidad al modelado; también teniendo en cuenta que toda Matriz dimensional puede ser expresada en términos de Vector Dimensional. Se agrega el concepto de restricciones que corresponde a restricciones propias del concepto representado por el elemento. Las unidades pueden establecerse como “Automáticas”; es decir, el sistema las determina. También es posible agregar unidades definidas por el usuario. A → Vista previa del valor con las unidades según la definición. B → Ayuda contextualizada. 1 → Otros editores de propiedades como Documentación, formato del elemento, etc..
IU006	

	A → Lista las categorías de funciones. B → Lista las funciones de la categoría seleccionada. C → Teclado para facilitar la escritura de nombres y números en la definición de la función. El teclado puede ser como el de algunos equipos celulares (tres letras por tecla más números, signos y teclas especiales). D → permite ingresar los parámetros de la función. Por cada parámetro existe un botón que permite acceder a la lista de variables relacionadas con el elemento. Igualmente, por cada parámetro se debe mostrar el valor que tiene en ese momento. E → presenta una descripción a manera de ayuda sobre el parámetro y la función seleccionados. F → Resultado que la función obtiene en ese momento (si es posible de calcular y los valores de los parámetro se están disponibles).
IU007	
	A → Lista los elementos a guardarse, ya que se puede estar guardando solo un modelo, todo el proyecto o un simulador. B → Opciones para permitir al usuario agregar los submodelos, dll, funciones y otros al archivo; de esta manera el archivo contiene todo lo necesario para poderse abrir en su totalidad en otro computador.
IU008	

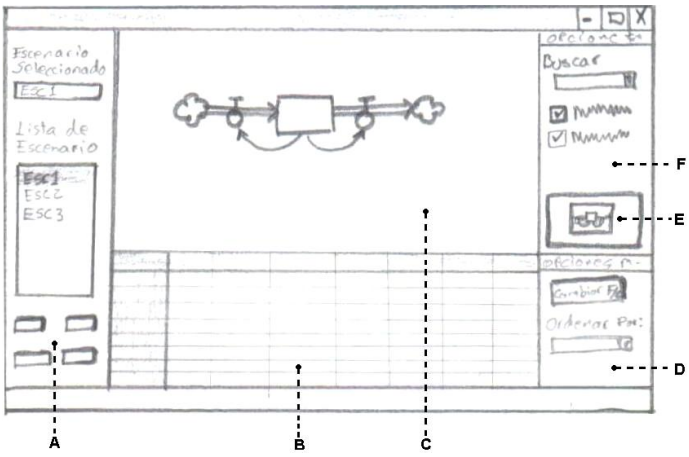


A → En esta sección aparecen las principales propiedades del proyecto como son el nombre, fecha y descripción.

B → Opciones para permitir al usuario agregar los submodelos, dll, funciones y otros al archivo; de esta manera el archivo contiene todo lo necesario para poderse abrir en su totalidad en otro computador.

Esc009: Crear/Modificar Escenarios

IU009



A → Administración de escenarios. En este panel se encuentran controles que permiten, entre otras cosas: visualizar la lista de escenarios disponibles, el nombre del escenario visualizado, crear un escenario nuevo, eliminar un escenario existente y duplicar un escenario.

B → Representación matricial del escenario actual.

C → Representación gráfica del escenario actual.

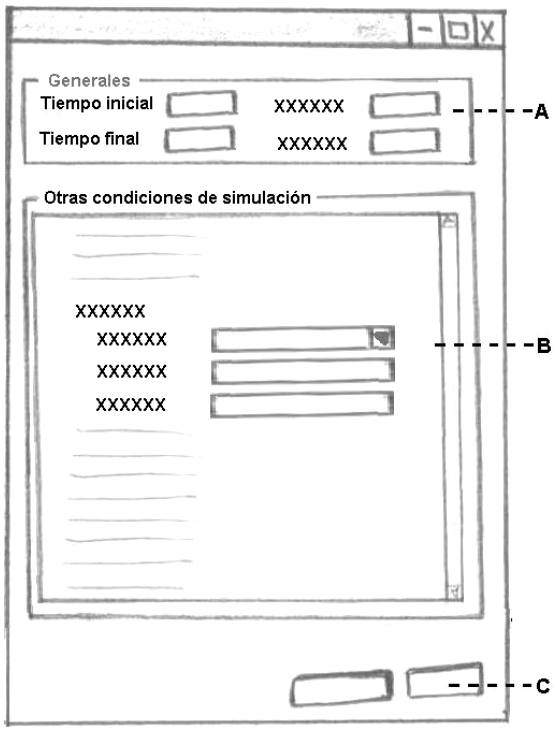
D → Panel de opciones de la representación matricial del escenario actual. Contiene controles que permiten: cambiar columnas por filas y ordenar por columna *X*, entre otros.

E → Zoom que permite desplazarse fácilmente por la representación gráfica del escenario actual.

F → Panel de opciones de la representación grafica del escenario actual. Contiene controles que permiten: buscar elemento, resaltar elementos no definidos, zoom, entre otros.

Esc010: Establecer condiciones de simulación

IU010



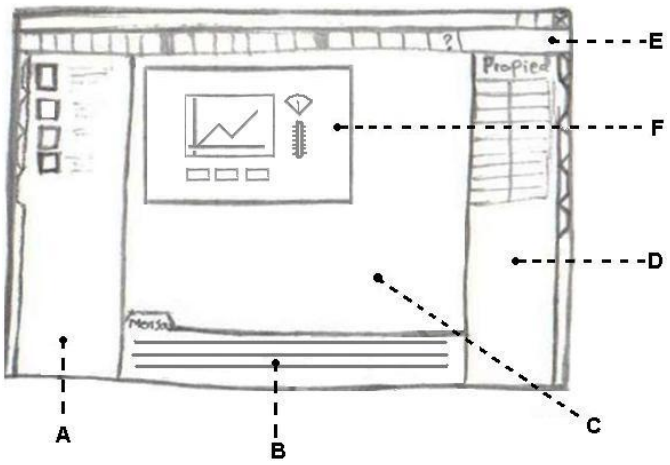
A → Condiciones de simulación generales (comunes en diferentes tipos de modelo)

B → Condiciones de simulación personalizada, dependiendo del tipo de modelo. Las propiedades y sus respectivas cajas de texto se colocarán en tiempo de ejecución y varían según el modelo.

C → Botones de comando para aceptar o cancelar.

Esc011: Diseño de la presentación de resultados

IU011



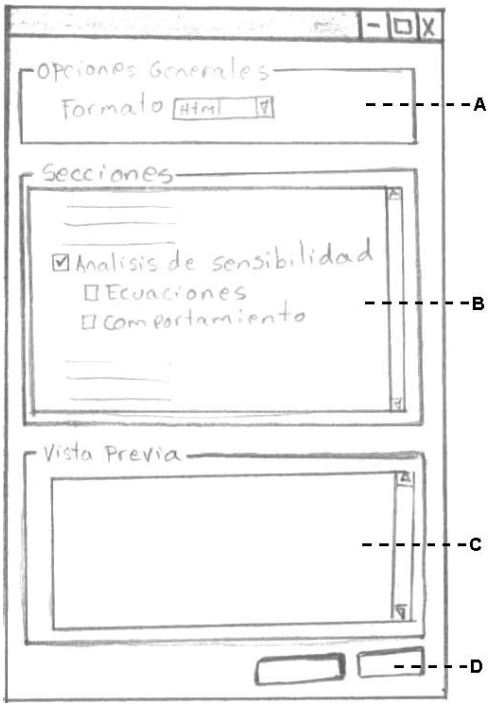
El diseño de los simuladores se realiza en el editor, el cual tiene las mismas herramientas del editor de diagramas (ver IU009). Este editor de simuladores permite crear interfaces gráficas con controles definidos para los simuladores, como graficas, tablas, track, dial, entre otros (para consultar la lista completa de componentes, ver sección 0).

F → Simulador con algunos controles.

Para los simuladores 3D se hace necesario modificar el editor para que contenga un control de navegación que permita ubicar los elementos en el espacio tridimensional.

Esc012:

IU012



A → En esta sección se establecen las opciones generales del reporte o informe, como por ejemplo el formato de salida del informe.

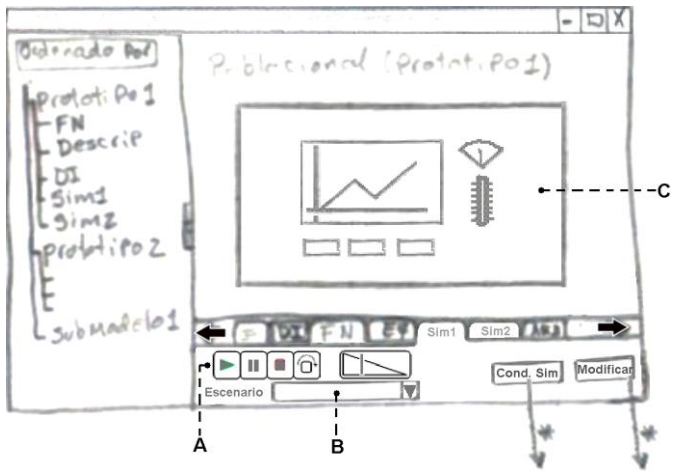
B → Aquí aparecen todas las secciones del informe las cuales pueden ser seleccionadas o deseleccionadas con el fin de que hagan o no parte del informe.

C → presenta una vista previa de ejemplo del informe, permitiendo ver al usuario como será la estructura del informe generado, es decir, cuáles secciones incluye y cuáles no.

D → Botones de comando para generar reporte, o cancelar.

Esc016: Simular

IU014

	A → Botones para controlar la simulación (en caso que los simuladores no los posea). Entre estos se encuentran iniciar la simulación, detener, pausar, paso a paso, control de velocidad, etc. B → Listado de todos los escenarios para seleccionar el empleado por el simulador. C → Simulador. Además, en este mismo panel aparece información relacionada con el simulador como: nombre, autor, descripción, fecha de creación, etc. Esta interfaz de usuario es la misma IU004, pero en lugar de estar posicionados en un modelo, se encuentra seleccionado un simulador

Fuente: Autor

6.5.1 Diseño del experimento o simulación

Una de las etapas del proceso de modelado y simulación es el *diseño del experimento o simulación*, en esta etapa se debe diseñar la forma de presentación de resultados, definir las condiciones de simulación y seleccionar un escenario.

Diseñar presentación

Consiste en diseñar la interfaz, formulario o presentación con la que el usuario podrá interactuar con una simulación. El proceso de diseñar esta interfaz debe ser posible de realizarse en el mismo componente editor de modelos. Esta interfaz del simulador está compuesta por controles de simulación.

Los controles de un simulador deben permitir modificar tanto el escenario en uso, así como interactuar con los elementos del modelo durante la simulación, además, algunos controles (como los botones) pueden modificar el Script de la simulación.

Cuando un control está asociado a una variable constante, definida en el escenario, ésta debe permitir regresar el valor definido en el escenario.

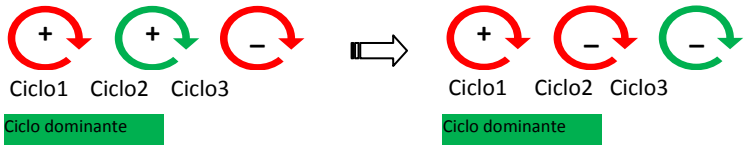
Para las características que lo requieran (por ejemplo para el valor máximo y el mínimo de un Track, o los ejes de una gráfica) deben tener escalas fijas o auto escala (el máximo cambia, si el valor de la variable excede el máximo original).

Algunos de los software de D.S, como Stella y Powersim, tiene presentaciones para la documentación de los modelos y simuladores, con textos, imágenes y enlaces con el propósito de documentar el modelo. En el grupo SIMON existen algunas herramientas que permiten hacer esto, por ejemplo los software MAC[25], HCAIAD[26], GAIA[27], entre otros. Además, incluyen funcionalidades para administrar la documentación en forma de contenidos temáticos acerca del modelo, es posible que se adapte este tipo de herramientas para que se pueda hacer un empaquetamiento del modelo y su documentación (en el menú herramientas), de esta manera, queda funcionando como un software independiente. Para lo anterior, se debe disponer de los módulos de motor y simuladores, de forma independiente para que así por cada modelo ejecutable no se repita el código. El diseño de la presentación de resultados es el diseño de la interfaz del simulador, con el cual interactúa el usuario, al simular con el modelo se puede tener cualquier cantidad deseada de simuladores.

Los controles mínimos que debe tener un simulador son:

- **Analizador de ciclos:** Muestra gráficamente los ciclos del modelo (los seleccionados por el usuario) indicando el tipo de realimentación (positiva o negativa) en tiempo de simulación (ver **Figura 37**); además, suministra información sobre cual ciclo predomina sobre los otros. Así mismo, se debe presentar una tabla histórica del estado de los ciclos seleccionados durante toda la simulación.

Figura 37. Control analizador de ciclos



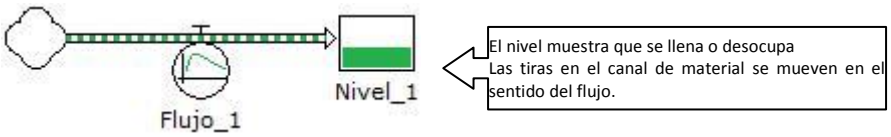
T	Ciclo1	Ciclo2	Ciclo3
0	+	+	-
1	+	+	-
2	+	-	-
3	+	-	-
…	+	-	-

Fuente: Autor

[25] En (Navas Garnica, 2006) se puede consultar una revisión completa de los MAC desarrollados en el grupo SIMON
[26] Software para la creación de ambientes educativos informáticos con D.S (Ospino Reales, y otros, 2006)
[27] Software para el estudio de fenómenos ambientales mediante la D.S (Santamaria Pabon, y otros, 2004)

- **Sector de Modelos:** permite mostrar un sector del modelo y visualizar de manera animada durante la simulación (ver valores, el llenado y vaciado de un nivel, ver el sentido en que circula el material por un canal de material…). Ref: Modelo en Stella[28] (ver **Figura 38**)

Figura 38. Control animador de modelos

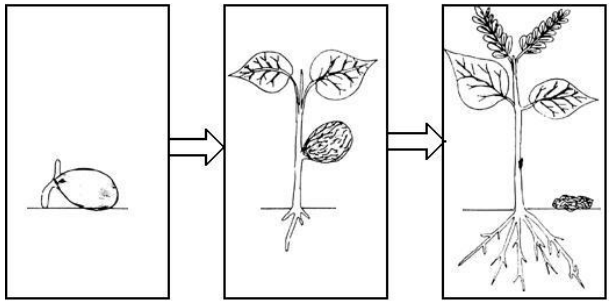


Fuente: Autor

- **Imágenes:** Estáticas o que se mueven o cambian su tamaño (ancho, alto) según los valores obtenidos por las variables o elementos asociados. Ref. Animadores en Evolución[29].

- **Secuencia de imágenes**: Es una imagen Gif que contiene una lista de imágenes. Se debe poder configurar entre qué rango de valores de una variable se muestra cada imagen (dentro de las herramientas se puede hacer un editor de Gifs) (ver **Figura 39**).

Figura 39. Componente secuencia de imágenes

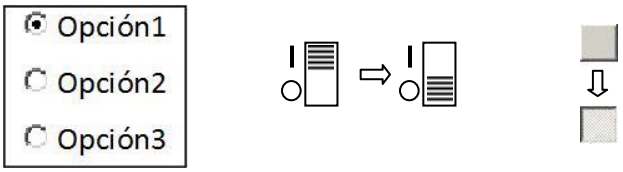


Fuente: Autor

- **Swiches**: Son botones, cajas de selección. Botones deslizables, etc. (ver **Figura 40**). que activan o desactivan alguna política o cambian algún valor del modelo. Ref. Swich en PowerSim[30] y Stella

[28] Versión 9
[29] Versión 3.5 en adelante
[30] Versión Studio 8

Figura 40. Control Swiches



Fuente: Autor

- **Indicadores o Led's**: encienden o apagan (o cambian de color) dependiendo de los valores obtenidos por la variable o elemento asociado, o en caso de cumplirse alguna condición especificada. Ref. Status indicator de Stella (ver **Figura 41** y **Figura 42**)

Figura 41. Indicador tipo diodo

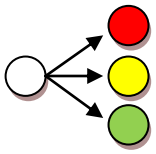
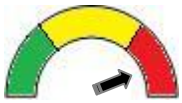


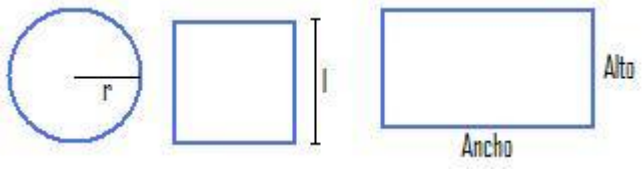
Figura 42. Indicador tipo Gauge o Dial

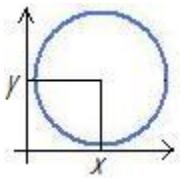


Fuente: Autor

- **Figuras parametrizadas**: Círculos cuadrados, líneas, hexágonos, rectángulos, etc.. que tienen sus características parametrizadas, como lo son la posición *x* y *y* del centro de la figura, o el radio de un círculo, o los lados de un rectángulo o cuadrado (ver **Figura 43**). Otras características parametrizables son las propiedades graficas de los elementos, como el color de relleno (tal color en un rango de valor y otro color en otro rango, el grosor del contorno, etc.).

Figura 43. Parámetros *X* y *Y* de las figuras

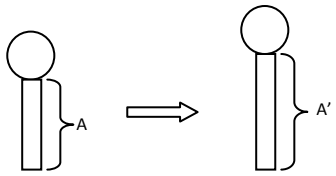




Fuente: Autor

Ahora, a estas figuras parametrizadas se les puede fijar sus parámetros *X* y *Y* a otro control, con el fin de formar figuras compuestas (ver **Figura 44**).

Figura 44. Control figuras compuestas

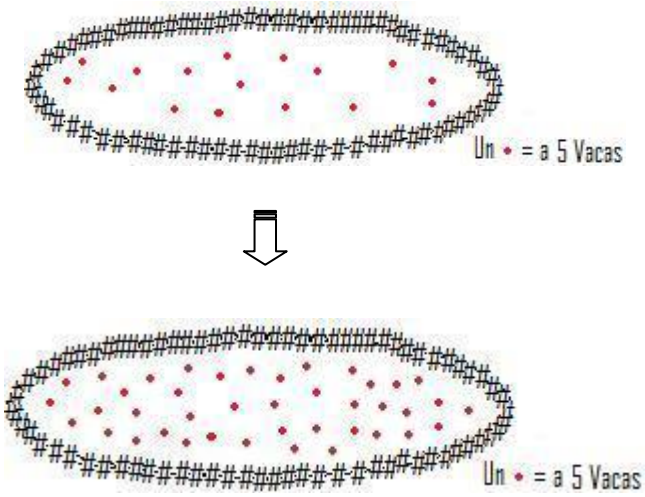


Fuente: Autor

Si la altura del rectángulo de la **Figura 44** cambia de valor (de *A* a *A'*), debido a que el círculo se ha fijado a la parte superior del rectángulo, debe desplazarse hacia arriba permaneciendo en la misma posición respecto al rectángulo.

- **Conjuntos:** Este control muestra un gráfico con elementos en marcados en un área, que representan una cantidad o valor de una variable asignada. El área del gráfico contiene elementos como puntos, triángulos, conos, etc. Cada uno de estos elementos representa un valor (por ejemplo, un elemento son 10 unidades) que al ser multiplicado por el número total de elementos, da como resultado el valor de la variable asociada (como los cuadros que representan sectores en el desfragmentador del disco duro) (ver **Figura 45**).
  Es posible que se creen varios conjuntos (o clases) que se presentan en el mismo gráfico y deben existir diferentes formas de distribución de los elementos en el gráfico. Estas áreas pueden tener diferentes formas y su contorno se puede definir con tramas (por ejemplo con "#" se formaría un corral y los elementos del conjunto equivalen a bovinos).

Figura 45. Control conjunto de elementos



Fuente: Autor

- **Cuadro de valores:** muestra un cuadro de textos con el valor de diferentes variables del modelo y permite su modificación (ver **Figura 46**).

Figura 46. Control cuadro de valores

NOMBRE	VALOR
Población	3520
Nacimientos	3
Muertes	1

Población	3520

Fuente: Autor

- **Botón de comando:** Este es un botón (ver **Figura 47**) que ejecuta un conjunto de acciones (o una acción) como: ejecutar un script de simulación, cambiar el valor de una variable, controlar la simulación (pausar, detener..) debe poder recibir funciones por ejemplo: población= población/2, es decir, cuando el botón es presionado, ésta disminuye a la mitad; Otra acción posible es mostrar un mensaje de texto multilínea en un cuadro de mensajes al usuario. Deben existir diferentes presentaciones de botones.

Figura 47. Control botón de comandos

Fuente: Autor

- **Figuras estáticas:** Decorativas como líneas, marcos, flechas, etc.

- **Reloj digital:** Para mostrar el tiempo de simulación (ver **Figura 48**). Este reloj podría incluir una barra de desplazamiento indicando el porcentaje de tiempo de simulación.

Figura 48. Control reloj

Fuente: Autor

- **Barra de herramientas de simulación:** conjunto de botones (ya configurados) para controlar la simulación, entre los que se pueden encontrar: Iniciar simulación, Pausar simulación, Detener Simulación, Paso de simulación y Control de velocidad de simulación (ver **Figura 49**).

Figura 49. Barra control de simulación

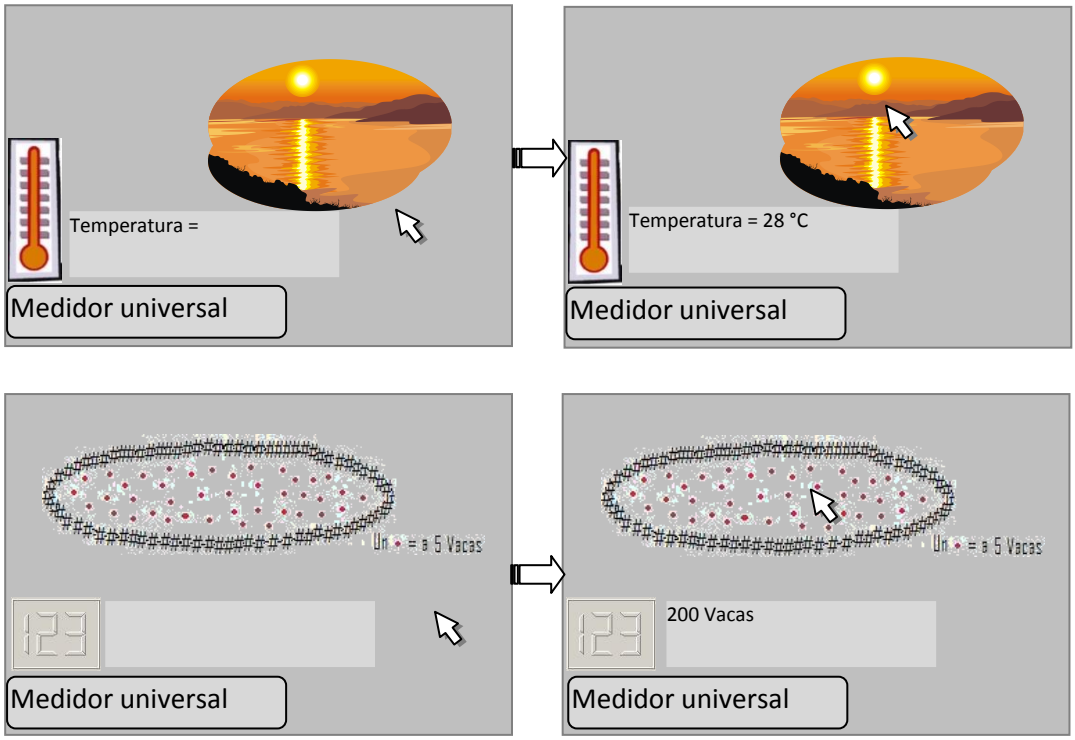
Fuente: Autor

- **Medidor universal:** es un dispositivo que permite medir (indica el valor de una variable) ciertas variables en el momento en que el usuario (habiendo seleccionado el botón del medidor) coloque el puntero del ratón sobre alguna imagen o control del Simulador (ver

- **Figura 50**). El usuario puede seleccionar el tipo de medidor (termómetro, barómetro, regla, etc…), previamente al diseñar el animador ya se ha

configurado en el control “medidor universal”, indicándole qué elementos el modelo (variables) debe medir para cada uno de los controles. También es posible configurarlo como un tipo de medidor específico (o permitir solo algunos tipos), también debe permitir colocar varios en el mismo Simulador. Uno de sus modos debe ser “universal” indicando que no tiene un tipo específico, por lo tanto cuando el usuario se coloca sobre un control del Simulador, debe mostrar el valor de todas las variables asignadas a dicho control.

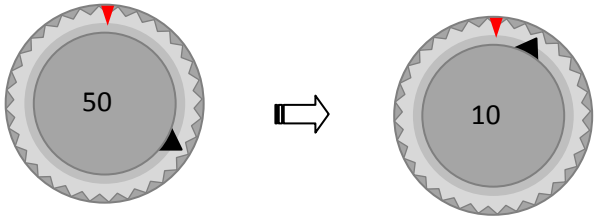
Figura 50. Control medidor universal



Fuente: Autor

- **Temporizador**: presenta el tiempo de la simulación como en forma de temporizador, en otras palabras, indica el tiempo restante para terminar la simulación (ver **Figura 51**). El usuario puede ampliar el tiempo de simulación, girando la perilla del temporizador.

Figura 51. Control temporizador



Fuente: Autor

6.5.1.1 Definir condiciones de simulación:

Debe permitir modificar las características de la simulación; cada simulador tiene sus propias características. Entre los parámetros de una simulación encontramos: tiempo inicial, tiempo final (puede establecerse como “indefinido”), delta o paso de simulación, método de integración, retorno al escenario inicial al terminar la simulación, tiempo en que inicia la grabación (grabación o impresión es cuando se presenta un resultado de manera visual por la IU) tiempo final de grabación, paso o Delta de grabación (en oportunidades se quiere tener exactitud en la simulación, pero un paso muy pequeño, puede colocar lenta la simulación), velocidad de simulación por defecto y escenario de simulación.

6.5.2 Simulación

Toda simulación se ejecuta por medio de un Script de simulación, este se puede crear y ejecutar, o en la medida que se van realizando acciones (iniciar, pausar, cambiar un valor), se va generando el Script. Un Script de simulación es el conjunto de comandos y configuraciones de una simulación, por ejemplo:

```
TI=0;
TF=100;
DT=0,1;
Iniciar simulación:
Pausarsimulación(t=50);
CambiarValor(Población=Población/2);
ReiniciarSimulción;
DetenerSimulacion;
```

Este Script debe permitir asignar un escenario diferente al por defecto, así como modificarlo (cambiar el valor de algún parámetro o valor inicial de algún nivel).

6.6 META ARQUITECTURA

La meta arquitectura es un conjunto de decisiones de alto nivel que influirán fuertemente a la estructura del sistema, pero no es en sí misma la estructura del sistema (Malan, y otros, 2004).

La arquitectura del ESMS-MI corresponde a una arquitectura heterogénea (Garlan, y otros, 1994) que combina varios estilos arquitectónicos. En principio, la arquitectura del entorno sigue el patrón arquitectónico PAC (Presentación-Abstracción-Control). Una arquitectura PAC define la estructura de un sistema software como una jerarquía de agentes PAC que cooperan entre sí. Cada agente está conformado por tres componentes: Presentación, Abstracción y Control. Estos componentes permiten separar los aspectos relacionados con la interfaz humano-computador de los aspectos de la lógica funcional y la comunicación con otros agentes (Buschmann, y otros, 1996).

La arquitectura PAC es una arquitectura para sistemas interactivos. Esta interacción con el usuario muchas veces es realizada por medio de interfaces gráficas de usuario (IGU). Cuando se diseña sistemas interactivos es deseable separar la lógica de la aplicación o núcleo funcional de la interfaz con el usuario. Entre los patrones arquitectónicos para sistemas interactivos sobresalen dos: Modelo-Vista-Control (MVC) y Presentación-Abstracción-Control (PAC), los cuales son descritos en (Buschmann, y otros, 1996).

El ESMS-MI es un entorno software que proveer diversos mecanismos de interacción con el usuario con el propósito de ofrecer una gran cantidad de herramientas para el modelado visual y la simulación interactiva con mecanismos avanzados de presentación gráfica y animada de resultados.

Entre los patrones arquitectónicos para sistemas interactivos, se ha seleccionado el patrón PAC. Este patrón define el sistema en función de agentes que cooperan entre sí. En nuestro sistema se logra identificar claramente agentes para el modelado, simulación, generador de reportes, entre otros. Otro requisito de nuestro sistema es la reutilización y los agentes son candidatos ideales para facilitar el cumplimiento de este requisito, permitiendo a los desarrolladores de otras aplicaciones hacer uso de los agentes para el propósito específico de su aplicación

Es de aclarar, que en este contexto un agente es un componente para procesar la información que incluye el envío y recepción de eventos, estructuras de datos para mantener el estado, un procesador que maneja los eventos recibidos, actualización de su estado y que pueda producir nuevos eventos. Los agentes pueden ser tan pequeños como un objeto, pero también tan complejos como un sistema software completo (Buschmann, y otros, 1996).

Esta arquitectura PAC presenta los siguientes beneficios y debilidades[31] (ver **Tabla 5**).

Tabla 5. Beneficios y debilidades de la arquitectura PAC

Beneficios	***Debilidades***
• Separación de intereses o conceptos. • Soporta el cambio (mantenimiento) y la extensión. • Soporta la multitarea (cada agente en un hilo o los componentes abstracción y presentación en hilos independientes). • Desacopla Interfaz de usuario del núcleo funcional del sistema	• Incrementa la complejidad del sistema. • Componentes "control" complejos. • Eficiencia. • Aplicabilidad.

A continuación, se presenta algunas conclusiones de la comparación entre dos arquitecturas orientadas a sistemas interactivos, el patrón PAC con el patrón MVC:

Para sistemas grandes (como lo en este caso) la estructura introducida por el modelo PAC provee mayor facilidad de mantenimiento que el provisto por el modelo MVC (Hussey, y otros, 1996).

En la arquitectura MVC la noción de control está diluida a través de los tres objetos relacionados (modelo, vista y controlador), mientras que en la arquitectura PAC el control es centralizado explícitamente en el componente *Control*. (The construction of user interfaces and the object paradigm, 1987).

El componente "presentación" de PAC es en esencia toda la interfaz de usuario tal como el usuario la percibe. En el modelo MVC, la interfaz de usuario está compuesta tanto por el componente "vista" como el componente "control"(Bass, y otros, 1991).

El modelo MVC es más simple que el modelo PAC y resulta en implementaciones y especificaciones menos complejas. (Hussey, y otros, 1996).

[31] Las debilidades pueden ser fortalecidas por medio de la combinación de patrones arquitectónicos o aplicación de patrones de diseño

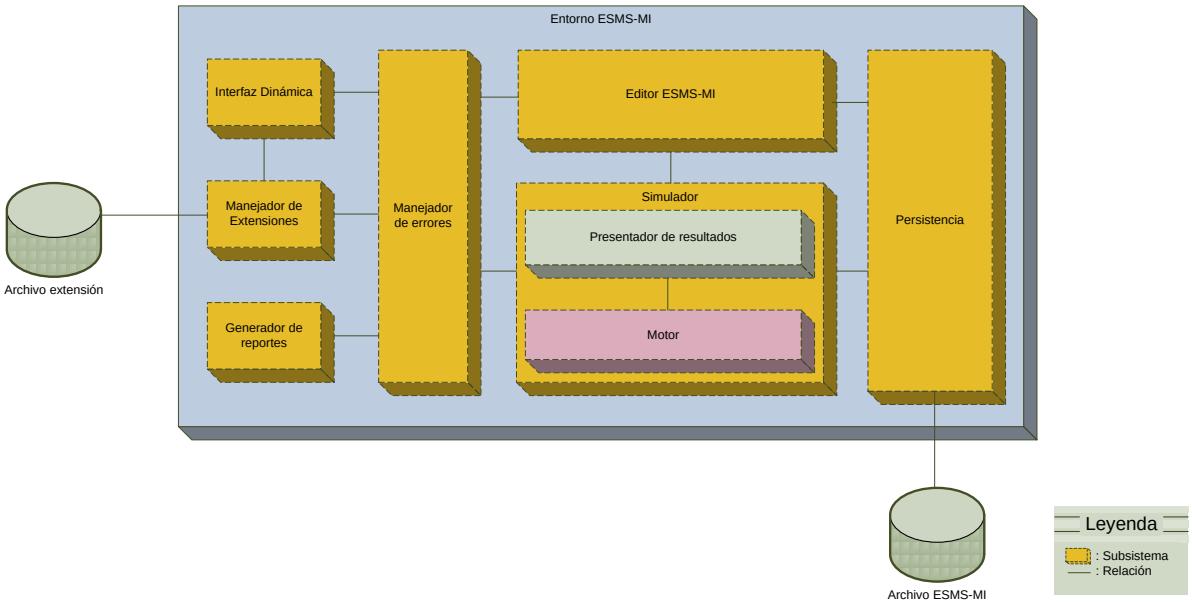
6.7 ARQUITECTURA CONCEPTUAL

En la arquitectura conceptual nos concentramos en una descomposición apropiada del sistema, antes de ahondar en los detalles de especificación de interfaces y tipos de información (Malan, y otros, 2004).

6.7.1 Arquitectura conceptual de alto nivel

La arquitectura conceptual de alto nivel contiene los principales subsistema del entorno y sus relaciones. En la **Figura 52** se presenta el diagrama de la arquitectura del entorno y sus elementos son descritos en los numerales 0 al 6.7.8.

Figura 52. Arquitectura conceptual de alto nivel del ESMS-MI

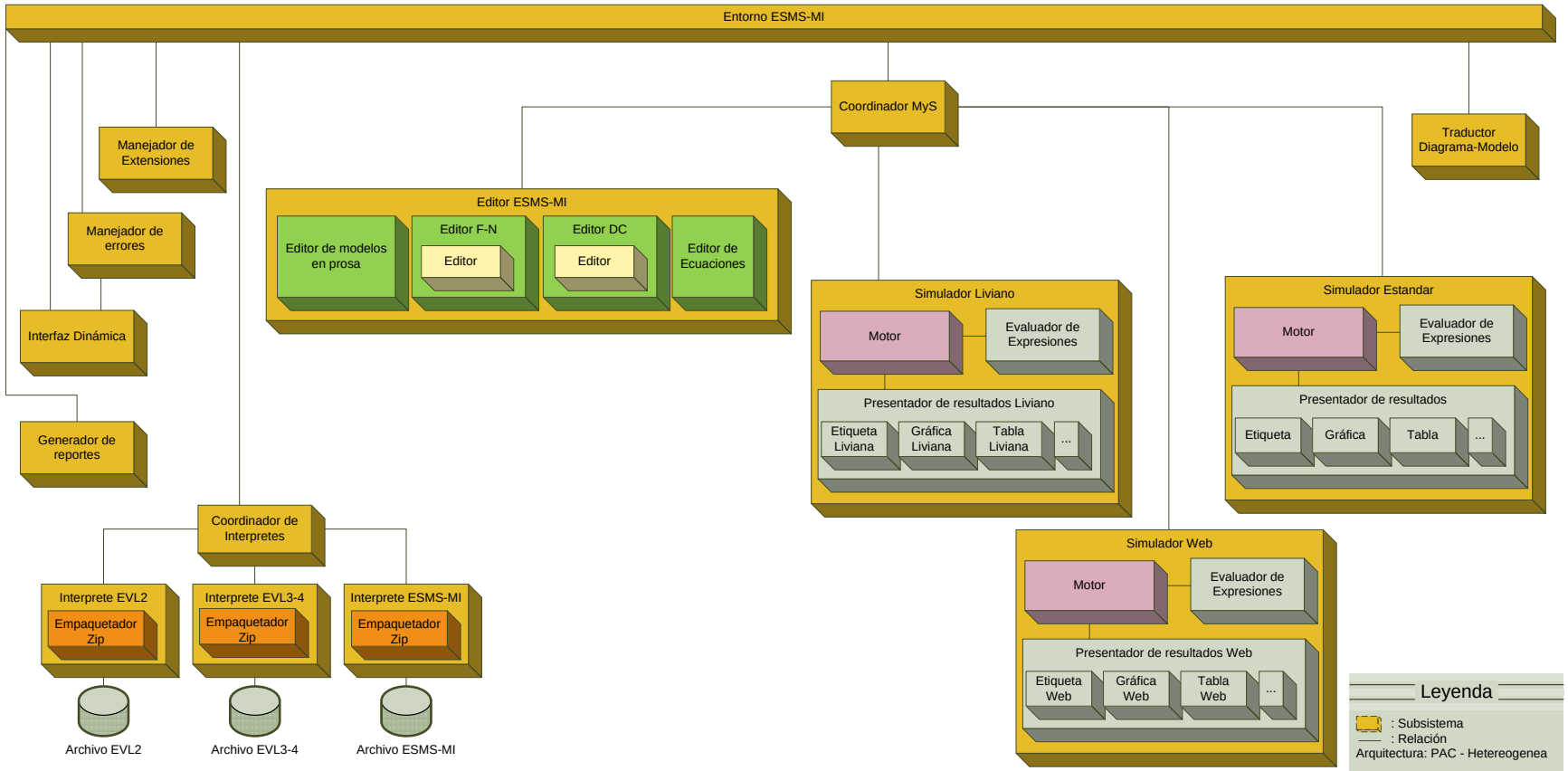


Fuente: Autor

6.7.2 Estilo arquitectónico

La arquitectura del entorno es una arquitectura basada en componentes, lo cual facilita su desarrollo por equipos independientes y separados geográficamente. El patrón arquitectónico es en esencia Presentación – Abstracción – Control (PAC), pero algunos de los agentes son subsistemas complejos con otros patrones arquitectónicos, es decir, su estructura interna no está compuesta por componentes PAC (Ver **Figura 53**)

Figura 53. Arquitectura del ESMS-MI

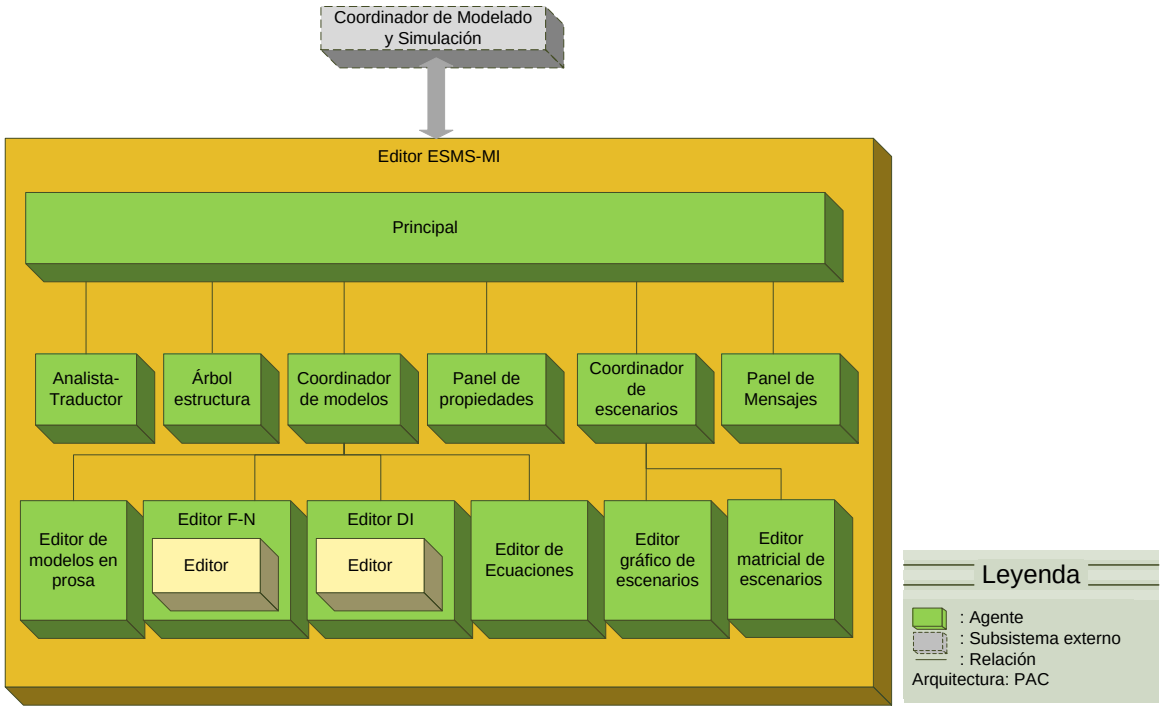


Fuente: Autor

6.7.3 Editor ESMS-MI

El subsistema Editor (ver **Figura 54**) permite al usuario realizar de forma gráfica e interactiva un modelo de simulación. Este subsistema contiene un conjunto de herramientas con los cuales el usuario puede realizar un modelo de forma gráfica. El editor debe permitir crear un modelo por el método de arrastrar-soltar o seleccionar-colocar y por medio de comandos de consola.

Figura 54. Editor del ESMS-MI



Fuente: Autor

Estilo arquitectónico

El estilo arquitectónico del Editor es PAC. Tanto como el editor de flujo-nivel (F-N) como el editor de diagramas de influencias (DI) hacen uso del editor Editor de modelos visuales.

Estructura

El editor está conformado por siguientes agentes:

- Principal
- Analista-Traductor
- Coordinador de modelos

- Árbol estructura
- Panel de propiedades
- Coordinador de escenarios
- Panel de mensajes
- Editor de modelos en prosa
- Editor Flujo-Nivel
- Editor Diagrama de Influencias
- Editor de ecuaciones
- Editor gráfico de escenarios
- Editor matricial de escenarios

Responsabilidades

- Proveer interfaz gráfica de interacción sistema-usuario.
- Permite hacer modelos en los diferentes lenguajes de la D.S.
- Coordinar y sincronizar los diferentes modelos y diagramas.
- Generar un modelo a partir del diagrama introducido por el usuario.
- Administración de escenarios.

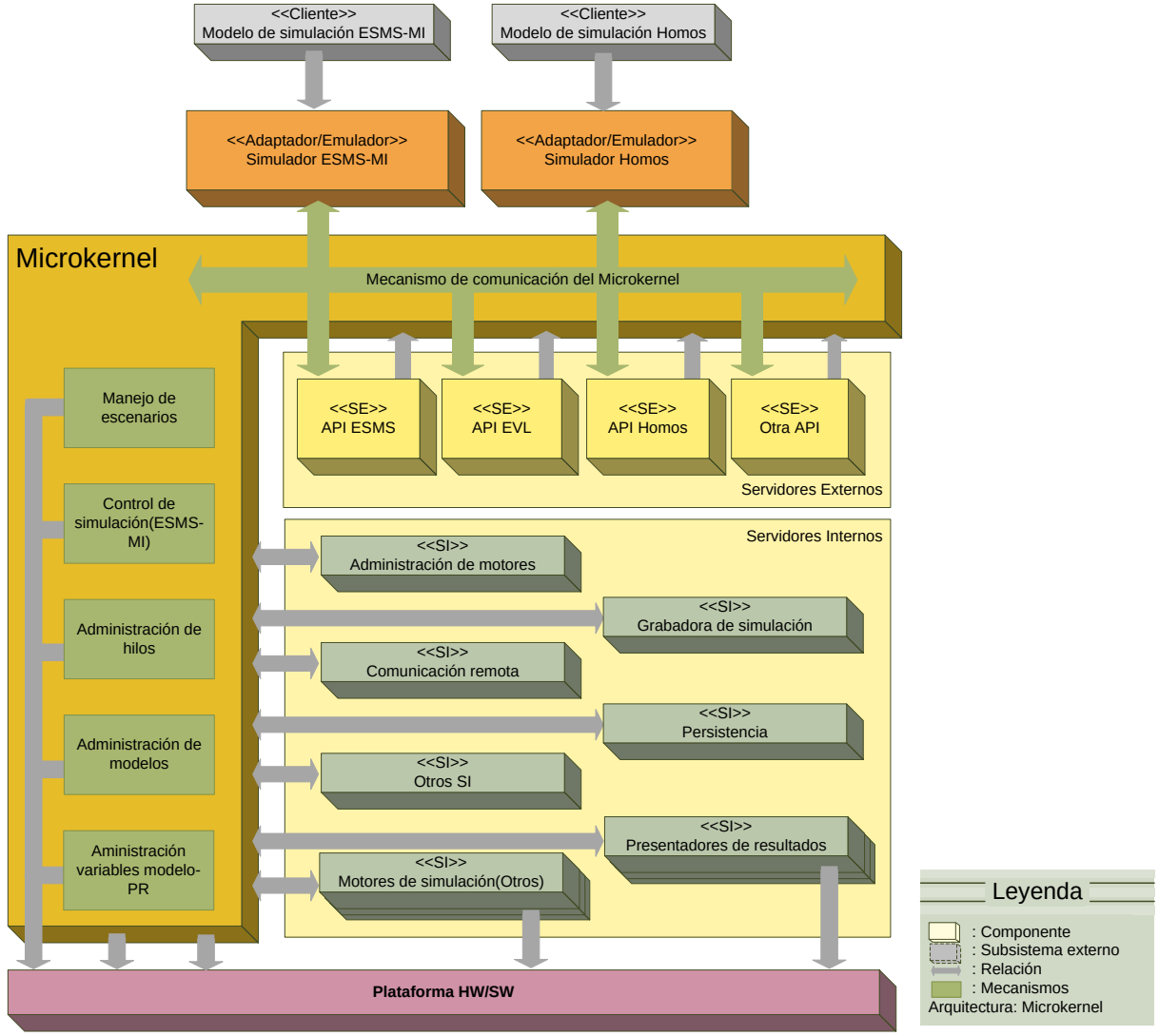
Relaciones y flujo de datos.

El Editor ESMS-MI se relaciona con el Coordinador de Modelado y Simulación (MyS), al cual le suministra los diagramas generados por el usuario. El diagrama correspondiente al modelo es convertido en un modelo entendible (incluyendo los escenarios) por el simulador y suministrado al coordinador de MyS. Igualmente, a través del coordinador envía los mensajes de error al manejador de errores. Por último, también por este medio envía los diferentes diagramas al subsistema de persistencia.

6.7.4 Simulador

El subsistema simulador (ver **Figura 55**) permite realizar simulaciones a partir de un modelo de simulación suministrado. El simulador tiene diversas formas de presentación de los resultados, ya sea de forma tabular, gráfica o por medio de animaciones parametrizadas, es decir, la animación está regida por los valores que se obtienen de la simulación del modelo. El simulador presenta una interfaz gráfica que le permite al usuario interactuar con la simulación a través de algunos controles. Unos controles permiten controlar la simulación y otros permiten modificar los valores de las variables del modelo. El simulador encapsula los diferentes motores de simulación y las diferentes interfaces o presentadores de resultados.

Figura 55. Simulador



Fuente: Autor

Estilo arquitectónico

El estilo arquitectónico del Simulador es Microkernel que es ideal para sistemas cambiantes. El componente principal es el Microkernel, el cual ofrece servicios atómicos denominados mecanismos, estos corresponden a la funcionalidad ofrecida por el núcleo del simulador. La funcionalidad de éste es extendida con los servicios ofrecidos por sus servidores internos. Los servidores externos colocan a disposición servicios más elaborados, creando plataformas de aplicación. Los clientes se comunican con los servidores externos por medio de las facilidades de comunicación ofrecidas por el Microkernel y que son encapsuladas por los adaptadores o emuladores. Los adaptadores también

pueden implementar el patrón de diseño Proxy para la comunicación con los servidores externos remotos.

Estructura

El Simulador está conformado por los siguientes componentes:

- Microkernel
- Servidores internos
- Administración de motores
- Grabadora de simulación
- Comunicación remota
- Persistencia
- Presentadores de resultados
- Presentador de resultados liviano
- Presentadores de resultados estándar
- Presentadores de resultados Web
- Motores de simulación
- Motor de simulación de HOMOS
- Motor de simulación de LD
- Servidores externos
- API ESMS-MI
- API EVL
- API HOMOS

El motor del ESMS-MI hace parte del componente Microkernel. Tanto los presentadores de resultados como los motores de simulación son subsistemas complejos. Se recomienda que el componente motor se diseñe haciendo uso del patrón arquitectónico Tuberías y Filtros, arquitectura para sistemas de flujo de procesos y muy utilizada en compiladores (otra opción de arquitectura puede ser Blackboard) y los presentadores de resultado el patrón arquitectónico PAC, arquitectura para sistemas interactivos.

Responsabilidades

- Realizar simulación de modelos integrados y otros tipos de modelos según los servidores externos implementados.
- Presentar por medio de interfaces gráficas y en diferentes plataformas Software-Hardware, los resultados de la simulación.
- Permitir a los usuarios interactuar con la simulación (cambiar valores de las variables durante la simulación y/o controlar la simulación) a través de controles.
- Interactuar con otros simuladores local y remotamente.
- Permitir interactuar con otras aplicaciones (enviar y recibir datos durante la simulación).

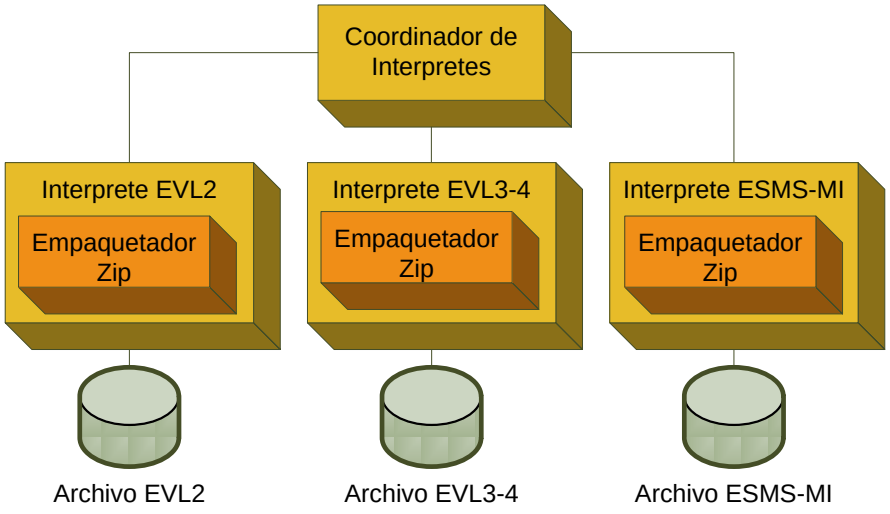
Relaciones y flujo de datos.

El simulador se relaciona con el Coordinador de Modelado y Simulación (MyS), el cual suministra la información necesaria para su funcionamiento como son el modelo y los escenarios de simulación.

6.7.5 Persistencia

El subsistema de persistencia (ver **Figura 56**) está conformado por Agentes encargados de gestionar los diferentes tipos de archivos soportados por el entorno. En la **Figura 56** se observa la estructura de este subsistema y los agentes encargados de abrir tres de los posibles tipos de archivos.

Figura 56. Subsistema de persistencia



Fuente: Autor

Estilo arquitectónico

Este subsistema continúa con el patrón arquitectónico del entorno, por lo que tiene un agente encargado de la coordinación, denominado Coordinador de intérpretes. Además, se cuenta con agentes denominados Intérpretes para los diferentes tipos de archivos soportados. Al menos se debe implementar intérpretes para los archivos de Evolución 2, 3.5-4 y el del entorno.

Estructura

El subsistema de persistencia está conformado por siguientes agentes:

- Coordinador de intérpretes.
- Intérprete EVL 2.

- Intérprete EVL 3-4.
- Intérprete ESMS-MI.

Responsabilidades

- Abrir y guardar archivos soportados por el entorno, suministrar y recibir los datos al entorno según corresponda.
- Seleccionar el procedimiento adecuado para poder abrir/guardar en los diferentes formatos.
- Compactar los datos (archivo de los diagramas, simuladores, modelo, etcétera) en un archivo comprimido según sea el caso.

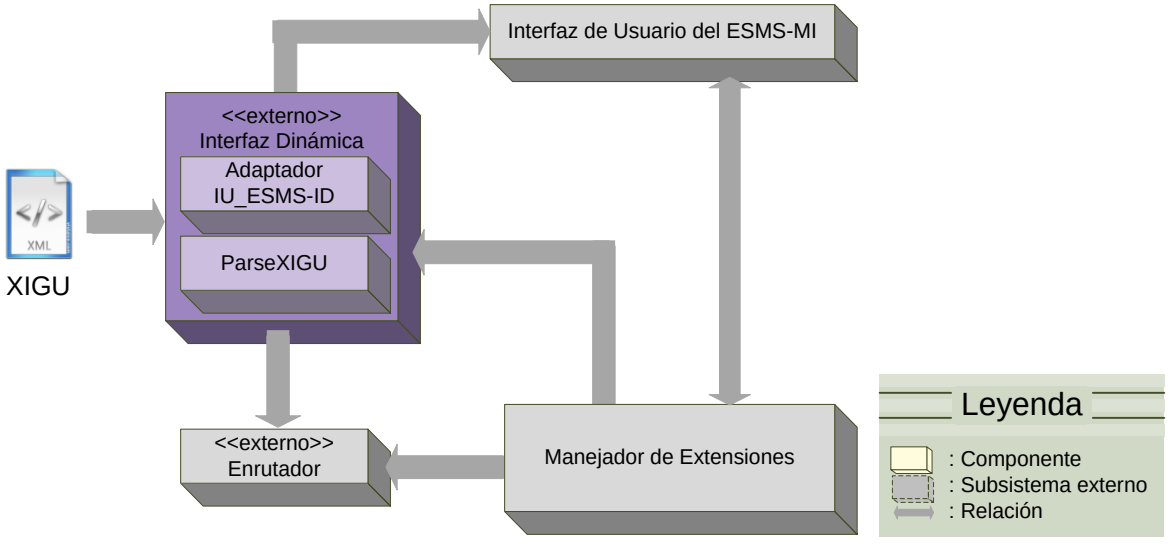
Relaciones y flujo de datos.

Toda la comunicación se debe realizar por medio del componente Control del entorno. Este subsistema recibe por parte del entorno del ESMS-MI los datos a guardar en el archivo, igualmente entrega a éste los datos del archivo que se le solicite abrir.

6.7.6 Interfaz dinámica

La interfaz dinámica (ver **Figura 57**) permite crear y administrar las Interfaz Gráfica de Usuario (IGU) creadas dinámicamente. Las IGU son creadas en tiempo de ejecución a partir de un archivo o cadena de texto con un script con código XIGU (Interfaz Gráfica de Usuario XML). Un archivo XIGU define una interfaz de usuario independiente de la plataforma, está escrito en el estándar XML y es interpretado por un ParserXIGU, el cual convierte el script en una interfaz real. La interfaz generada es acorde a la interfaz de todo el entorno.

Figura 57. Interfaz dinámica



Fuente: Autor

Estilo arquitectónico

Se recomienda utilizar el patrón arquitectónico Capas, separando los componentes de persistencia, procesamiento y presentación. La API XIGU debe cumplir con el estándar XML.

Estructura

El subsistema Interfaz dinámica está conformada por:

- ParserXIGU
- Adaptador IU_ESMS-ID

Responsabilidades

- Interpretar archivos XIGU.
- Crear GUI a partir del script XIGU.
- Gestionar las IGU creadas dinámicamente.

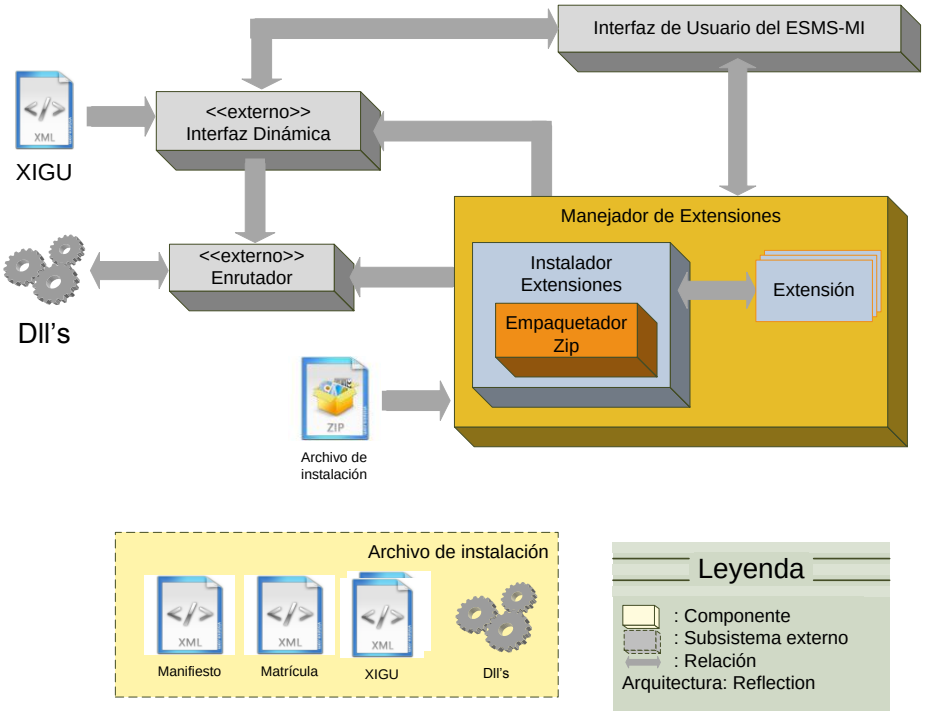
Relaciones y flujo de datos.

- El entorno
- Subsistema Manejador de extensiones
- Componente enrutador

6.7.7 Manejador de extensiones

Este subsistema es el encargado de gestionar la instalación de las extensiones, habilitarlas/deshabilitarlas, desinstalarlas y coordinar la ejecución su ejecución (ver **Figura 58**).

Figura 58. Manejador de extensiones



Fuente: Autor

Estilo arquitectónico

El estilo arquitectónico para el manejador de extensiones es patrón Reflection.

Estructura

El subsistema Manejador de extensiones está conformado por los siguientes componentes:

- Instalador de extensiones.
- Extensiones

Responsabilidades

- Instalación/desinstalación de extensiones
- Provee acceso a las librerías y códigos de extensiones
- Proveer a las extensiones de mecanismos para la creación a la interfaz de usuario.
- Desempaquetar archivos de instalación.
- Verificar consistencia de los paquetes de instalación.
- Habiliatar/deshabilitar extensiones instaladas.

- Administración de las versiones de extensiones (verificar versión, comprobar actualizaciones, verificar compatibilidad, entre otras).

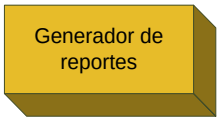
Relaciones y flujo de datos.

- Interfaz dinámica
- Enrutador

6.7.8 Generador de reportes

El subsistema Generador de reportes (ver **Figura 59**) permite generar un documento que contiene datos sobre el proceso de modelado y simulación, tales como imágenes del modelo, ecuaciones, descripciones de elementos, resultados de una simulación, gráficas de resultados, etcétera. Este reporte es personalizable, permitiendo al usuario seleccionar las partes que desea que aparezcan en este. También, soporta el uso de plantillas predefinidas para la elaboración de reportes. El documento generado se debe poder exportar a diferentes tipos de archivos como: Open Document, MS-Word y Html.

Figura 59. Generador de reportes



Fuente: Autor

Estilo arquitectónico

Se recomienda utilizar el patrón arquitectónico Capas.

Responsabilidades

- Generación de reportes en diferentes formatos.
- Configuración de reportes.
- Administrar plantillas de reportes

Relaciones y flujo de datos.

Se relaciona con el Entorno. El generador de reportes toma los datos para armar el reporte, de un archivo XML que genera el ESMS-MI, el cual solicita información a cada uno de los componentes o subsistemas.

6.8 ARQUITECTURA DETALLADA

La arquitectura detallada o arquitectura lógica define de forma precisa los mecanismos de conexión y las interfaces entre los componentes y los protocolos.

A continuación se detallan las interfaces de los principales componentes del entorno.

6.8.1 Interfaz Simulador

Información <<interfase>>
Nombre: string read FNombre write SetNombre;
Descripcion:TDescripcionAnimador //el usuario agrega la descripción del animador
FechaCreacion:TDate read FFechaCreacion; //se guarda la fecha en que se creó el animador
Autor:
Persistencia <<interfase>>
Cargar(S: TDatosEnMemoria):boolean;virtual; xml
Guardar(S: TDatosEnMemoria):boolean;virtual;
ScriptSimulación <<interfase>>
    CargarScript
    EjecutarScript
    EjecutarScriptPasoAPaso
    DetenerScript
ControlSimulación <<interfase>>
    AumentarVelocidad
    DisminuirVelocidad
ControlSimulacion <<interfase>>
RegresarAlEscenarioInicial: boolean
Iniciar;virtual;
    Detener;virtual;
    Pausa;virtual;
    PasoDeSimulacion;virtual;
EstadoSimulacion:TEstadoSimulacion read FEstadoSimulacion write SetEstadoSimulacion;
Animar(lista de variables modelo, lista variables sistema)
VentanaSimulacion <<interfase>>
    CrearVentana(CS:TCondicionesDeSimulacion):entero //retorna el código de la ventana
    EliminarVentana(idVentana:entero):booleano //retorna falso si hay error
    ModificarVentana(idVentana:entero; nomVariable: texto; Valor:texto) :booleano //retorna falso si hay error

TPropiedadesVentana <<tipo>>
TiempoInicial
TiempoFinal
Delta
Velocidad

CondicionesDeSimulacion <<interfase>>
CargarCondicionesSimulacion
EstablecerTiempoInicial
EstablecerTiempoFinal
EstablecerAlgoritmoSimulacion(texto) <<en DS corresponde al método de integración>>
EstablecerDeltaSimulacion

RolBack
SnapShoot(unPaso:integer)

6.8.2 Interfaz Motor

Modelo
DefinirEscenario(TObject) read getEscenario write SetEscenario;
ControlSimulación
    AumentarVelocidad
    DisminuirVelocidad
ControlSimulacion
RegresarAlEscenarioInicial: boolean
PuedeIniciar:boolean;virtual;
    Iniciar;virtual;
    Detener;virtual;
    Pausa;virtual;
    PasoDeSimulacion;virtual;
EstadoSimulacion:TEstadoSimulacion read FEstadoSimulacion write SetEstadoSimulacion;
Grabacion
ActivarGrabarSimulacion
DetenerGrabarSimulacion
IrAPasoAnterior
Persistencia
CargarModelo(S: TDatosEnMemoria): boolean;virtual;
    GuardarModelo(S: TDatosEnMemoria): boolean;virtual;
Consulta
    TiempoDeSimaulacion

Motor.iniciar
Motor.pausar
Motor.pasoSimulacion
Motor.detener
Motor.cargarSnapShoot(SnapShoot)

6.8.3 Interfaz Editor de modelos Visuales (diagramas)

Persistencia
AbrirModeloDeArchivo(Archivo:string):boolean;
    GuardarModeloEnArchivo(Archivo:string):boolean;
    AbrirDiagrama(EspacioMemoria)
    GuardarDiagrama(EspacioMemoria)
Metamodelo
    AgregarMetamodelo

ListaDeMetamodelos
EliminarMetamodelo
Edicion
AgregarElemento(tipo, nombre: string; posX, PosY: integer):true;
EliminarElemento
SeleccionarElemento
UbicarElemento
AgregarElementoASeleccion
EliminarElementoDeSeleccion
ModificarPropiedadElemento(elemento: string; propiedad:string; valor:string)
ElementosSeleccionados:Lista
CopiarSelección
CortarSelección
DeshacerUltimaAccion
Consulta
ExisteUnElementoSeleccionado: boolean
NumeroPropiedadesElementoSeleccionado: integer
Exportar
ExportarModeloGrafico(En_donde:string; Filtro:integer);
ExportarModeloTexto(En_donde:string; Filtro:integer);
Importar

6.8.4 Interfaz del entorno

Administracion/Seguridad
ProtegerModelo(contrasena)
CambiarContrasena(contrasenaAnterior; contrasenaNueva)
EditarElementoSeleccionado;
EditarEscenarios;
EnviarMensajeSector(ObligarAUno:boolean=false);
EnviarMensajeOrdenes;
ImprimirDiagramaForresterEnCanvas(Cual_canvas: TCanvas;
ImprimirDescripcionElementosForrester(Cual_Canvas: TCanvas;
HacerReporteDiagramaForresterEnCanvas(Documento : TDocumentoWord;
HacerReporteDescripcionElementosForrester(Documento:TDocumentoWord;
HacerReporteSoloDescripcionYAutor(Documento : TDocumentoWord;
KeyDown(var Key: Word; Shift: TShiftState); override;
MouseWheelHandler(var Message: TMessage); override;
Pegar;
ReHacer;
VerAdministracion;
ZoomMas;
ZoomMenos;
AcercaDe;
Guardado:boolean read GetGuardado write SetGuardado;
DatosProyecto:TDatosProyecto

```
ElementoSeleccionado:TTipoElemento
EscenarioActual:string
ZoomValor:single read GetZoomValor write SetZoomValor;
TipoPlumaSector: TPenStyle
AnchoPlumaSector: Integer
FormaSectorSector: TDrawingTool;
ColorSector: TColor;
ColoresPersonalizadosDelFormato: TStringList;
ManejadorElementos: TManejadorElementos;
MantenerTipoElemento:Boolean;
OnMensaje:TEventoMensaje;
OnNuevaPagina:TNuevaHojaEvent;
Visible: Boolean;
  end;
```

6.9 DOCUMENTACIÓN

Como parte de la documentación de la arquitectura, la presente tesis genera los siguientes documentos:

- El documento de descripción de la arquitectura (ver Anexo B)
- Ficha técnica de la arquitectura (ver Anexo C)

El primero es el documento principal para comunicar la arquitectura, contiene una descripción de los elementos del sistema y sus relaciones. El segundo corresponde a una ficha técnica para referencia rápida para los interesados en el proyecto, también puede funcionar como poster para facilitar una referencia visual de la estructura del sistema.

6.10 VALIDACIÓN DE LA ARQUITECTURA

Con el fin de favorecer a la objetividad es recomendable que esta etapa sea realizada por personas ajenas al proceso de construcción de la arquitectura (Malan, y otros, 2004). Por otra parte, para evaluar la arquitectura existen métodos que involucran un conjunto extenso de actividades. Por estas razones, la validación de la arquitectura desborda los alcances de la presente investigación. No obstante, se hace necesario que sea abordada rigurosamente, por lo que se recomienda que los proyectos encargados de los sub-proyectos realicen la evaluación de la arquitectura desde el punto de vista de los componentes involucrados en su respectivo sub-proyecto. También, es recomendada la realización de un proyecto encargado de aplicar un método de evaluación arquitectónica como the Architecture Tradeoff Analysis Method (ATAM) y the Cost Benefit Analysis Method (CBAM) descritos en (Bass, y otros, 2007). Como antecedente se encuentra la utilización de ATAM para evaluar la arquitectura de la

herramienta Evolución 3.5, realizada en la universidad del Cauca (Fernández de Valdenebro, 2005).

6.11 CONCLUSIONES

- El estudio formal de la arquitectura tiene pocos años respecto a otras áreas de la ingeniería software, por lo que es preciso realizar más investigaciones en este campo. Normalmente los trabajos se remiten a la escuela de Carnegie Mellon (CMU-SEI) con Mary Shaw, David Garlan, Paul Clements, Len Bass, Rick Kazman, quienes se han convertido en autoridades en el tema. Otro campo relativamente poco explorado es el de los patrones arquitectónicos, en donde se resalta el trabajo realizado en *Pattern-Oriented Software Architecture* (Buschmann, y otros, 1996).
- La realización de esfuerzos a nivel de la academia e investigación para la aplicación *consciente* en los desarrollos software de estos estilos arquitectónicos y la reutilización de soluciones a problemas arquitectónicos comunes, haciendo uso de patrones arquitectónicos, es poca. La aplicación de estilos arquitectónicos en nuestro contexto, ha estado más influenciada por la moda o tendencia a un estilo arquitectónico que por los atributos y características del sistema. Es así como encontramos que gran parte de los desarrollos dicen tener arquitectura de capas para las aplicaciones web y cliente servidor o el modelo MVC para aplicaciones de escritorio o monousuario y actualmente, la Service Oriented Architecture SOA para aplicaciones de integración.
- Es posible crear arquitecturas heterogéneas que beneficien diferentes requisitos. La arquitectura aquí definida combinan patrones para sistemas interactivos y altamente modificables (entre otros requisitos) en una sola estructura.
- EL modelado visual y los métodos formales para definir arquitecturas son dos fuerzas que debe equilibrar el diseñador de una arquitectura, el primero favorece la comunicación entre los diferentes interesados del proyecto (Stakeholders) y mientras que el segundo, favorece el seguimiento y control de las decisiones arquitectónicas a lo largo de todo el proceso de desarrollo de software.

CAPÍTULO 7. NÚCLEO DEL ENTORNO SOFTWARE DE MODELAMIENTO Y SIMULACIÓN DE MODELOS INTEGRADOS

El presente capítulo corresponde al cumplimiento del objetivo:

> *Promover el desarrollo del núcleo del Entorno Software de Modelamiento y Simulación de Modelos Integrados (ESMS-MI), a partir de la herramienta Evolución 4.0.*

7 NÚCLEO DEL ENTORNO SOFTWARE DE MODELAMIENTO Y SIMULACIÓN DE MODELOS INTEGRADOS

7.1 INTRODUCCIÓN

Para dar inicio a las actividades de la comunidad que desarrollará el ESMS-MI, es necesario definir los requerimientos y la arquitectura del nuevo entorno. Igualmente, se debe desarrollar un núcleo a partir de Evolución. Esto permite que la comunidad visione el nuevo desarrollo y pueda verificar la viabilidad del mismo, antes de comprometerse a realizar un sub-proyecto. Es por esto, que el grupo SIMON se ha dado a la tarea de desarrollar el núcleo de la nueva herramienta, a partir de la herramienta software Evolución.

Varios proyectos software desarrollados en comunidad surgen a partir de un desarrollo inicial, es el caso de Mozilla el cual nace a partir de la liberación del código de Navigator por parte de Netscape, en el año 1998. Luego, un grupo de voluntarios se unen alrededor de dicho código para continuar con su desarrollo y mejoramiento (Mozilla Hispano, 2007). Otro ejemplo es Moodle desarrollado inicialmente por Martin Dougiamas y hoy cuenta con una comunidad de más de 200 desarrolladores (Moodle.org).

Este capítulo presenta las condiciones necesarias para promover el desarrollo del núcleo del ESMS-MI y los proyectos de pregrado asociados a la creación de dicho núcleo.

7.2 PROMOCIÓN DEL DESARROLLO DEL NÚCLEO DEL ESMS-MI

Para iniciar el desarrollo del núcleo del nuevo entorno a partir de la herramienta software Evolución, es necesario contar la documentación completa de esta herramienta, principal debilidad encontrada por el proyecto de maestría (Moreno Chaustre, 2006), y en general realizar el mantenimiento de la herramienta a la luz de los errores encontrados durante su evaluación. Igualmente, es necesario obtener los requisitos del sistema como un todo, estos requisitos y su respectivo análisis guiarán el diseño e implementación no sólo del núcleo sino de todo el entorno. Por otra parte, se debe realizar un diseño a alto nivel que permita establecer los componentes mínimos del entorno y las relaciones entre estos. Esta visión general es dada por el diseño de la arquitectura del nuevo entorno. Dadas estas condiciones es posible promover el desarrollo del núcleo.

El desarrollo del núcleo del ESMS-MI comienza con la creación de un evaluador de expresiones matemáticas, componente software que hace parte (o está incluido) del motor de simulación. El motor de simulación, es en esencia el núcleo del nuevo entorno, el cual tendrá a su cargo la integración de otros motores de simulación, con el fin de implementar la integración entre la D.S y otras herramientas matemáticas.

Para promover el desarrollo del núcleo, se ha llevado a cabo una convocatoria al interior de la UIS, de la cual surgen los siguientes proyectos de pregrado:

- Mantenimiento de Evolución
- Evaluador de expresiones
- Motor del ESMS

7.3 MANTENIMIENTO DE EVOLUCIÓN

Proyecto que tiene como objetivo realizar el mantenimiento de Evolución 4.0 a partir de la ejecución de pruebas al sistema, errores reportados por los usuarios y la evaluación realizada al software en la tesis de maestría (Moreno Chaustre, 2006). Este proyecto aporta al ESMS con la documentación y una versión depurada del software Evolución, del cual parte el proyecto del desarrollo del ESMS-MI. También hace un especial aporte en la metodología y artefactos para la realización de pruebas que aseguren la calidad de los desarrollos del ESMS-MI.

7.4 EVALUADOR DE EXPRESIONES

Proyecto que tiene como objetivo desarrollar un componente software que permita interpretar y evaluar expresiones matemáticas, utilizando funciones y operadores empleados en el lenguaje matemático convencional y que incluya la verificación de consistencia dimensional. Este proyecto aporta al desarrollo del ESMS con el producto principal de la tesis, el componente software. Este intérprete a diferencia de los tradicionales (por ejemplo el utilizado por Evolución), permite utilizar

unidades, lo cual posibilita verificar la consistencia dimensional de una expresión matemática. Debido a que uno de sus integrantes no reside actualmente en el país, ha generado nuevas necesidades y aportado sugerencias a la plataforma software de integración de la comunidad.

7.5 MOTOR DEL ESMS

Este proyecto tiene como objetivo desarrollar el motor de simulación para un ESMS-MI medio de D.S, en términos de un componente software reutilizable. El motor debe ser multiplataforma y trabajar integrado al entorno o de forma independiente, recibir como entrada el modelo y entregar los resultados generados a otros componentes del entorno u otras aplicaciones. Al igual que el evaluador de expresiones, este proyecto aporta un componente software que hace uso del evaluador y será el núcleo de simulación del ESMS-MI.

Los tres proyectos han estado bajo la orientación del autor de la presente tesis, quien tiene el rol de codirector. El trabajo de codirección y coordinación de los tres proyectos, se ha realizado haciendo uso de la plataforma software de la comunidad o plataforma de integración. Los proyectos mencionados trabajan de manera distribuida y remotamente, aunque la mayoría de los integrantes radican en Bucaramanga. Cabe resaltar que tanto los proyectos y la tesis de investigación se realimentan entre sí.

7.6 CONCLUSIONES

El trabajo por proyectos de grado de forma distribuida es una buena forma de abordar la realización de proyectos de gran envergadura. Pero se hace necesario realizar un proyecto inicial que analice los aspectos globales de todo el macro-proyecto.

El desarrollo basado en componentes facilita el desarrollo de aplicaciones y herramientas software a gran escala y de gran calidad (Casal Terreros, 2007). Se propone crear en la Escuela de Ingeniería de Sistemas e Informática (EISI) de la UIS una librería de componentes y Frameworks para futuros desarrollos.

CAPÍTULO 8. ORIENTACIONES PARA LA DOCUMENTACIÓN, PRUEBA Y EVALUACIÓN DE LOS DESARROLLOS INDEPENDIENTES DEL ESMS-MI

El presente capítulo corresponde al cumplimiento del objetivo:

> *Formular las especificaciones metodológicas que orienten la realización de la documentación, prueba y evaluación de los desarrollos independientes a cargo de la creación de los componentes del ESMS-MI*[32].

[32] A partir de las especificaciones desarrolladas en la tesis "Diseño de una arquitectura para un entorno de modelamiento-simulación y creación de un proceso para su desarrollo por una comunidad (I+D)".

8 ORIENTACIONES PARA LA DOCUMENTACIÓN, PRUEBA Y EVALUACIÓN DE LOS DESARROLLOS INDEPENDIENTES DEL ESMS-MI

8.1 INTRODUCCIÓN

La documentación y la evaluación de software son dos aspectos claves en todo desarrollo de software, por su relación directa con la calidad del mismo.

En un proceso de desarrollo distribuido la documentación del software se hace aún más importante y necesaria, ya que es un medio que facilita la comunicación y el entendimiento de los desarrollos individuales por parte de los integrantes de la comunidad. Se debe establecer un conjunto de lineamientos que orienten a todos los involucrados en el desarrollo, en cuanto a la realización de una documentación uniforme y sobre los aspectos mínimos para comunicar y mantener el producto software.

Así mismo, hay que definir las políticas y los lineamientos para el desarrollo de la evaluación. Preguntas como: ¿Quién debe realizar la evaluación general del software desarrollado de forma distribuida?, ¿Quién es el responsable de su calidad? ¿Qué pruebas debe realizar cada uno de los sub-proyectos? ¿Cuándo se debe realizar la evaluación y las pruebas? deben ser resueltas al iniciar el macro-proyecto para el desarrollo del ESMS-MI.

Este capítulo trata los temas de la documentación y evaluación de software en el contexto de la metodología de desarrollo Agile-DISOP, generando un marco conceptual, orientaciones y artefactos para la realización de estas actividades durante el desarrollo distribuido del ESMS-MI.

8.2 DOCUMENTACIÓN

La documentación es un aspecto muy importante en el desarrollo de software. La mayoría de las veces la documentación es realizada al finalizar el desarrollo de software, pero los procesos de desarrollo iterativos posibilitan realizar la documentación de forma paralela al desarrollo del software. La documentación se ha convertido en una especie de piedra en el zapato para los desarrolladores de software. No obstante, la mayoría de los sistemas software cuya única documentación es el código, quedan obsoletos rápidamente y es imposible mantenerlos (MITOpenCourseware, 2001).

La documentación de software es uno de los aspectos que ha estado en el ojo del huracán con la aparición de los métodos ágiles. Para muchos el encanto de estas metodologías ágiles es su reacción a la burocracia de las metodologías tradicionales. Estos nuevos métodos buscan el punto medio entre la no existencia de un proceso de desarrollo y un proceso exagerado, proporcionando simplemente el proceso suficiente para que el esfuerzo valga la pena (Fowler, 2000). Los seguidores de estas metodologías, comparten la idea de que es necesaria una alternativa a los métodos consagrados, basados en una gestión burocrática, una documentación exhaustiva y procesos de peso pesado. Igualmente, promueven el modelado pero no con el fin de archivar algún diagrama en un polvoriento repositorio corporativo. Promueven la documentación, pero no cientos de páginas en tomos a los que no se les hace mantenimiento y rara vez son usados; planifican, pero reconocen los límites de la planificación en un entorno turbulento (Beck, y otros, 2001).

El resultado de todo esto es que los métodos ágiles cambian significativamente algunos de los énfasis de los métodos ingenieriles. La diferencia inmediata es que son menos orientados al documento, exigiendo una cantidad más pequeña de documentación para una tarea dada. Son más bien orientados al código: siguiendo un camino que dice que la parte importante de la documentación es el código fuente (Fowler, 2000).

Sin embargo, a menos que los desarrolladores sean infalibles y vivan en un mundo en el que nada cambia, tendrán que volver a consultar el código que ya está escrito, y pondrán en duda decisiones que tomaron durante el desarrollo del mismo. Si no documentan sus decisiones, cometerán los mismos errores y tardarán tiempo en comprender lo que pudo haber descrito fácilmente en una ocasión (MITOpenCourseware, 2001).

Las documentaciones extensas abruman al lector y puede fácilmente acarrear inconvenientes de desactualización. Es esencial documentar sólo los asuntos *correctos*. La documentación no sirve de ayuda si su extensión desanima a la gente a la hora de leerla (MITOpenCourseware, 2001).

Otro asunto es *cuándo* documentar. Aunque algunas veces es conveniente posponer la tarea de la documentación mientras se realizan experimentos, los programadores con experiencia suelen documentar de forma metódica incluso el código provisional, los análisis de un problema inicial y los borradores de un diseño. Ellos creen que esto hace que la experimentación sea más productiva. Además, dado que han tomado la documentación como hábito, les resulta normal documentar a medida que van avanzando (MITOpenCourseware, 2001).

En (Gravendeel, 2009) se definen dos tipos de documentos en el contexto del desarrollo software:

- Documentación que los miembros del equipo necesitan para trabajar en el proyecto.
- Documentación para ser entregada con el producto final.

Dentro de los desafíos planteados por la metodología Agile DISOP relacionados con la documentación del sistema software se puede mencionar:

- ¿Cómo unificar un modelo de desarrollo a través de la comunidad?
- ¿Cómo alcanzar compatibilidad en los diferentes estilos de desarrollo que usa la comunidad?
- ¿Cómo alcanzar compatibilidad total de la documentación?
- ¿Cómo integrar una suite de herramientas de soporte para la comunidad?

Estos desafíos generan unos requisitos a la metodología:

- Unificar un estilo de desarrollo para toda la comunidad.
- Unificar el modelo de documentación para toda la comunidad.
- Unificar las herramientas de soporte y su entorno garantizando su disponibilidad, libre acceso y bajo costo para toda la comunidad.
- Llenar los vacíos dejados por las metodologías de desarrollo horizontales, las cuáles especifican como debe dirigirse el desarrollo de software de forma transversal indicando a veces lo “QUÉ” debe hacerse y omitiendo el “CÓMO” debe hacerse.

A continuación se presenta las decisiones en cuanto a los requisitos mencionados, para el desarrollo del ESMS-MI:

1. Estilo de desarrollo

Un estilo de desarrollo común en toda la comunidad facilita la coherencia y comprensión de la documentación. En este sentido la metodología Agile DISOP promueve el desarrollo Ágil, Iterativo, Incremental y Comunitario. (Moreno Chaustre, 2006). Igualmente, se recomienda la utilización de un estándar de notación para el código fuente (por ejemplo la notación húngara), el cual debe ser seleccionado en consenso por la comunidad. Por último, y no sobra decirlo, comentar su código cuidadosamente y con buen gusto.

2. Modelo de documentación

La metodología Agile DISOP propone dos formas de organizar la documentación, por disciplinas o por Fases (Moreno Chaustre, 2006). Se recomienda hacer uso

de la primera forma (ver **Figura 60**), apoyado con la implementación de un sistema de control de versiones (ver capítulo 9).

Figura 60. Organización de la Documentación por Disciplinas

Fuente (Moreno Chaustre, 2006)

Algunos documentos, por su relevancia a nivel de consulta se deben publicar en un Wiki común para toda la comunidad. Es el caso de la especificación de requisitos, la arquitectura, el diseño y el manual de usuario. Estos Wikis deben estar disponibles en la plataforma de la comunidad.

Con el fin de dar uniformidad a toda la documentación generada por la comunidad, se recomienda la utilización de la estructura mencionada en cada uno de los sub-proyectos, así como la utilización de los artefactos y plantillas definidas por el Agile DISOP (por ejemplo plantillas de casos de uso y casos de prueba) y las orientaciones para la documentación presentadas a continuación.

La estructura de las carpetas, plantillas y los artefactos de Agile Disop, se encuentran disponibles en la plataforma de la comunidad. También pueden ser consultadas en la tesis de maestría (Moreno Chaustre, 2006).

La documentación para el sistema software debe tener la siguiente estructura[33]

a. Requisitos

La sección de requisitos describe el problema que se está solventando junto con la solución. Esta sección de la documentación es de interés tanto para los usuarios como para los implementadores; no debería contener detalles sobre la estrategia de implementación en concreto. Las otras partes de la documentación del sistema no serán de interés para los usuarios, únicamente para los implementadores, los encargados del mantenimiento y demás personal.

[33] Esta estructura ha sido tomada y enriquecida del (MITOpenCourseware, 2001)

b. Diseño

Esta sección proporciona un cuadro de alto nivel de la estrategia de implementación. Para explicar la descomposición y otras decisiones de diseño, manifieste explícitamente cuándo aportan simplicidad, extensibilidad (facilidad para añadir características nuevas), particionalidad (los distintos miembros del equipo pueden trabajar en las diferentes partes del diseño sin necesidad de mantener una comunicación constante) u objetivos similares relativos a la ingeniería de software.

c. Pruebas

Esta sección indica el enfoque que se ha elegido para verificar y validar el sistema[34]. En esta sección no es recomendable comunicar el diseño de su sistema presentando el código o incluso listando las clases. Igualmente, no debe solo enumerar las pruebas realizadas; Por el contrario, se debe explicar cómo se seleccionaron las pruebas, por qué éstas son suficientes, por qué un lector debería creer que no se ha omitido ninguna prueba importante y por qué debería creer que el sistema realmente funcionará como debe ser, cuando se ponga en práctica.

d. Reflexión

La sección de reflexión (post mortem) del documento es donde se puede hacer generalizaciones partiendo de los éxitos o los fallos concretos, hasta llegar a formular reglas que usted u otros puedan utilizar en el futuro desarrollo de software. ¿Qué fue lo que más le sorprendió? ¿Qué hubiera deseado saber cuándo comenzó? ¿Cómo podría haber evitado los problemas que encontró durante el desarrollo?

e. Apéndice

El apéndice contiene detalles de bajo nivel acerca del sistema, que no son necesarios para comprenderlo en un nivel superior, pero que se exigen para usarlo en la práctica o para verificar afirmaciones realizadas en cualquier parte del documento. Esta puede referenciar artefactos ubicados en la organización de la documentación descrita anteriormente.

f. Herramientas de soporte para la documentación

En cuanto a las herramientas de soporte a la gestión de la documentación para el proyecto ESMS-MI, se ha colocado en funcionamiento una plataforma software para la comunicación y gestión de la comunidad, como parte de los resultados de la presente tesis. Esta plataforma es discutida en detalle en el capítulo 9 del presente documento.

[34] Podría incluir tanto las pruebas de usuario para determinar la idoneidad del sistema como la solución al problema descrito en la sección de requisitos, como la ejecución de *suites* de prueba para verificar la corrección algorítmica del código.

8.3 EVALUACIÓN DE SOFTWARE

Con el fin de entregar a los clientes productos satisfactorios, el software debe alcanzar ciertos niveles de calidad. La calidad es “el grado con el que un sistema, componente o proceso cumple los requisitos especificados y las necesidades o expectativas del cliente o usuario” (IEEE Computer Society, 1991).

Para facilitar la comprensión de los procesos de aseguramiento de calidad, esta se ha descompuesto en atributos. A la hora de abordar estos atributos es importante distinguir entre cuatro conceptos muy relacionados pero distintos (IEEE Computer Society, 1990):

- ***Error*:** Acción humana que produce un resultado incorrecto.

- ***Falta*:** Algo que está mal en un producto (modelo, código, documento, etc.).

- ***Fallo*:** La incapacidad de un sistema o de alguno de sus componentes para realizar las funciones requeridas dentro de los requisitos de rendimiento especificados.

- ***Defecto*:** Un defecto en el software como, por ejemplo, un proceso, una definición de datos o un paso de procesamiento incorrectos en un programa.

Controlar y corregir las faltas existentes en un producto software afecta positivamente algunos atributos de calidad. En particular, si se trabaja en detectar y eliminar las faltas y los fallos la funcionalidad y la fiabilidad mejoran.

Durante el desarrollo de software, las distintas técnicas de evaluación son las principales estrategias para detectar faltas y fallos. Por tanto, son métodos de control de la calidad.

En términos generales, se pueden distinguir dos tipos de evaluaciones durante el proceso de desarrollo: Verificaciones y Validaciones. Según la IEEE éstas se definen como (IEEE Computer Society, 1990):

- ***Verificación*:** El proceso de evaluación de un sistema o de uno de sus componentes para determinar si los productos de una fase dada satisfacen las condiciones impuestas al comienzo de dicha fase.

- ***Validación*:** El proceso de evaluación de un sistema o de uno de sus componentes durante o al final del proceso de desarrollo para determinar si satisface los requisitos especificados.

8.4 PROCESO DE EVALUACIÓN DE SOFTWARE

El proceso aquí presentado está basado en la norma ISO/IEC 14598 e ISO/IEC 9126-1. La evaluación de software inicia con una visión cualitativa y deriva en una evaluación cuantitativa. El proceso de evaluación de software debe ser documentado en su totalidad y está conformado por los siguientes pasos:

1. Establecer el estado de desarrollo del Software: Conocimiento del estado del software, estableciendo si se trata de un desarrollo sin terminar o un producto terminado para la entrega al cliente.

2. Identificar el tipo de Software: Especificar el tipo de software a evaluar, si es un sistema operativo, software de seguridad, software de ofimática, lenguaje de programación, base de datos, aplicativo a la medida, herramienta de modelado y simulación, entre otros. Tenga en cuenta que dependiendo del tipo de software se debe personalizar el plan de pruebas, dando prioridad a una u otra prueba según las características del software.

3. Identificar perfiles de los evaluadores: Teniendo como marco conceptual al estándar ISO, se consideran tres perfiles de usuario: usuarios finales, desarrolladores y gerentes (a un alto nivel de abstracción para desarrollo de software).

   El estándar afirma que la relativa importancia de las características de calidad (como usabilidad**,** funcionalidad**,** confiabilidad**,** eficiencia**,** portabilidad, mantenibilidad y calidad en uso) varían dependiendo del punto de vista considerado y de la crítica de los componentes del software a evaluar.

   La visión del **usuario final**, concierne al interés de los mismos en usar el software, en su funcionalidad, su eficiencia, su facilidad de uso, entre otros aspectos. Los usuarios finales no están interesados en características internas o de desarrollo del software (sin embargo, atributos internos contribuyen a la calidad de uso).

   La visión de calidad del **desarrollador** debe considerar no sólo los requerimientos del software para la visión del usuario, sino también, la calidad para los desarrollos intermedios resultantes de las actividades de la fase de desarrollo. Se debe tener en cuenta que los desarrolladores están, principalmente, preocupados por las características de calidad del software así como en la mantenibilidad y la portabilidad.

   La visión de calidad del **gerente** es una visión integradora, que incorpora requerimientos de negocio a las características individuales. Ejemplo, un gerente está interesado en el equilibrio entre la mejora del software, los costos y los tiempos establecidos.

4. Especificar los Objetivos: Conocer los objetivos tanto generales como específicos del software.

5. Aplicar el Modelo de Calidad: Elaborar un instrumento o formato donde aplique el modelo de calidad externo e interno y calidad de uso. Si existe un comité o conjunto de personas encargadas de la evaluación, el instrumento debe ser aprobado por los participantes.

6. Criterios de la Evaluación: Los criterios para evaluar el software se dividen en dos bloques:

    a. Criterios generales: son aplicables a cualquier tipo de software.
    b. Citerior específicos: son adaptables al grupo de software evaluados.

    En este paso se definen los criterios de la evaluación según el tipo de software, para el cual debe conformar un equipo evaluador, este ejercicio ayuda a definir que opciones se deben evaluar con más detalle y valor.

    A continuación, se describe la clasificación de los criterios para la selección de software de simulación en una estructura jerárquica realizada por (Criteria for simulation software evaluation, 1998). El software, el vendedor y los usuarios son los elementos más importantes por lo que forman el nivel más alto de la jerarquía.

    Software

    - Modelo y entrada
    - Ejecución
    - Animación
    - Prueba y eficiencia
    - Salida

    Vendedor

    - Pedigree
    - Documentación
    - Soporte
    - Preventa

    Usuarios

    - Tipo de Simulación
    - Orientación
    - Hardware
    - Dispositivos de Seguridad
    - Sistema Operativo
    - Versión en red

- Económico
- Experiencia requerida
- Clase de software

7. Seleccionar Métricas: La selección de métricas se obtiene a partir de los indicadores especificados en el modelo.

8. Niveles o escalas: Se deben definir los niveles o escala para cada una de las métricas, siguiendo las siguientes orientaciones:
   - A cada métrica seleccionada le asigna un puntaje máximo de referencia.
   - La suma de los puntajes máximos de todas las métricas debe ser igual o aproximado a 100 puntos.
   - El personal que participa en la evaluación debe establecer niveles de calificación cualitativa con base a los puntajes, por ejemplo:
     De 0 a 1 Inaceptable.
     De 2 a 3 mínimo aceptable.
     Más de 3 Aceptable o satisfactorio.

     Otro ejemplo de calificación pero de tipo cualitativo puede ser: Deficiente, Insuficiente, Aceptable, Sobresaliente, Excelente.

   - Se permite usar números enteros o hasta con un decimal de aproximación.
   - Definir por cada métrica, un puntaje mínimo de aprobación, y al final de de la evaluación, dependiendo del puntaje si es mayor o menor a lo propuesto, considerar si el software cumple o no cumple con los objetivos propuestos.

9. Establecer Criterios para la escala: Las persona que participa en el proceso de evaluación debe tener criterios con respecto al indicador que se está analizando y para cada uno de los niveles establecidos, es importante tener en cuenta que el criterio se debe ajustar al tipo de software que se va a evaluar.

10. Tomar Medidas: Para la medición, las métricas seleccionadas se aplican al software. Los resultados son valores expresados en las escalas de las métricas, definidos previamente. Es posible que algunas métricas se puedan realizar de forma automatizada con la ayuda de software especializado.

11. Resultados: El proceso de evaluación genera un cuadro con los resultados por cada uno de los principales indicadores y el total final de resultado.

12. Documentación: El proceso de evaluación se documenta, indicando la fecha, empresa, los cargos, nombres y apellidos, dependencia de las personas que participan en el proceso de evaluación, especificando las etapas en las que participaron.

13. Seguimiento: Si el resultado de la evaluación tiene observaciones o indicadores de calidad bajos, y el personal que lo evalúa permite realizar la corrección, se programa otra evaluación donde se verifique que el proceso mejora, el tiempo que se estime debe influir en los criterios de la próxima evaluación.

8.5 PRUEBAS DE SOFTWARE

La IEEE (IEEE Computer Society) define los conceptos de prueba y caso de prueba como:

Pruebas *(test):* Una actividad en la cual un sistema o uno de sus componentes se ejecuta en circunstancias previamente especificadas, los resultados se observan, se registran y se realiza una evaluación de algún aspecto.

Caso de Prueba *(test case):* Un conjunto de entradas, condiciones de ejecución y resultados esperados desarrollados para un objetivo particular.

Las pruebas deben centrarse en:

Probar si el software no hace lo que debe.

Probar si el software hace lo que no debe, es decir, si provoca efectos secundarios adversos.

TIPOS DE PRUEBAS

Según la norma ISO/IEC 14598 e ISO/IEC 9126-1 los tipos de prueba se pueden clasificar en función de qué se conoce, grado de automatización, lo que se prueba.

En función de qué conocemos:

Pruebas de Caja Negra: Se centra en el estudio de la especificación del software, del análisis de las funciones que debe realizar, de las entradas y de las salidas.

Pruebas de Caja Blanca: conocemos el código (la implementación de éste) que se va a ejecutar y podemos definir las pruebas que cubran todos los posibles caminos del código.

Según el grado de automatización:

Pruebas Manuales: son las que se hacen normalmente al programar o las que ejecuta una persona con la documentación generada durante la codificación (Por ejemplo: comprobar cómo se visualiza el contenido de una página Web en dos navegadores diferentes).

Pruebas Automáticas: se usa un determinado software para sistematizar las pruebas y obtener los resultados de las mismas (Por ejemplo: verificar un algoritmo de ordenación).

En función de qué se prueba:

Prueba de Unidad: Centra el proceso de verificación en la menor unidad del diseño del software: la clase u objeto encapsulado. Está dirigida por las operaciones encapsuladas por la clase y el estado del comportamiento de la clase.

Pruebas de Integración: Pueden realizarse usando estrategias basadas en hilos o basadas en el uso. Las pruebas basadas en hilos integran el conjunto de clases necesario para responder a una entrada o evento del sistema. La prueba basada en usos construye el sistema por capas, comenzando con aquellas clases que no hacen uso de clases servidores.

Pruebas de Validación: Se centra en las acciones visibles del usuario y las salidas del sistema reconocido por éste. Podemos distinguir entre dos pruebas:

> **Pruebas Alfa:** las realiza el usuario en presencia de personal de desarrollo del proyecto haciendo uso de una máquina preparada para tal fin.
>
> **Pruebas Beta**: las realiza el usuario después de que el equipo de desarrollo les entregue una versión casi definitiva del producto.

Prueba del Sistema: Está constituida por una serie de pruebas diferentes, cuyo propósito primordial es ejercitar profundamente el sistema software. Aunque cada prueba tiene un propósito diferente, todas trabajan para verificar que se han integrado adecuadamente todos los elementos del sistema y que realizan las funciones apropiadas.

Otros tipos de pruebas:

Pruebas de Regresión: Es la actividad que ayuda a asegurar que los cambios (debidos a las pruebas o por otros motivos) no introducen un comportamiento no deseado o errores adicionales.

8.5.1 Proceso de pruebas en un entorno software de modelado y simulación ESMS

Durante la fase de transición del proceso de desarrollo (Agile DISOP) es posible iterar varias veces haciendo énfasis en la disciplina de pruebas con el objeto de preparar el producto para su liberación con base en la realimentación de los usuarios. En este punto, la realimentación del usuario ayuda a enfatizar en los ajustes de granularidad fina sobre el producto, su configuración e instalación, así como las dificultades relacionadas con la usabilidad. Al final de la fase de transición, el proyecto debe estar próximo a concluir, sin embargo el final del proyecto puede coincidir con el inicio de otro para la próxima versión del producto.

Considere distintos grados de *granularidad* para las pruebas. Utilice pruebas de unidad para componentes individuales, pruebas de integración para el componente versus su entorno y pruebas globales para el sistema versus todos los componentes recién actualizados o integrados.

Agile-DISOP define cuatro actividades en el proceso de prueba del software, las cuales son descritas en la **Tabla 6. Actividades relacionadas con prueba de software en Agile-DISOP**.

Tabla 6. Actividades relacionadas con prueba de software en Agile-DISOP

Nombre de Actividad	**Pruebas:** Planificar Pruebas y Definir Estrategias
Descripción	Esta actividad tiene como propósito fundamental, la identificación de la información y de los componentes software del proyecto que se eligen durante esta iteración para las pruebas, la enumeración de los requerimientos para realizar las pruebas, recomendar y describir las estrategias de pruebas que se emplearán, identificar los recursos y esfuerzo requeridos para realizar las pruebas y finalmente, enumerar los productos entregables al final de las pruebas. Por otra parte, la estrategia de pruebas define: • Las herramientas y las técnicas que se usarán. • Los criterios de finalización exitosa de las pruebas.
Entradas	▪ Plan de Proyecto. ▪ Plan de Iteración. ▪ Modelo del Análisis – Modelo del Diseño – Modelo de la Implementación.
Salidas	▪ **Plan de Pruebas de la Iteración.**
Roles Responsables	▪ Gestor del Proyecto (RUP 2003)
Nombre de Actividad	**Pruebas:** Ejecutar Pruebas
Descripción	Consiste en implementar las pruebas sobre los componentes de software elegidos para esta iteración, aplicando las estrategias concebidas.
Entradas	▪ Modelo del Análisis – Modelo del Diseño – Modelo de la Implementación. ▪ Plan de Pruebas de la Iteración.
Salidas	▪ **Registro de los Resultados de las Pruebas.**
Roles Responsables	▪ Ingeniero de Pruebas
Nombre de Actividad	**Pruebas:** Evaluar Pruebas
Descripción	Consiste en la consolidación, síntesis e interpretación de los resultados de las pruebas con el objeto de fundamentar la toma de decisiones relacionadas con el rediseño o no de la arquitectura. De la satisfacción de estos resultados depende que el incremento de software sea publicado o no en la plataforma.

Entradas	▪ Registro de los Resultados de las Pruebas.
Salidas	▪ **Interpretación de Resultados.**
Roles Responsables	▪ Analista de Pruebas
Nombre de Actividad	**Pruebas:** Publicar en la Comunidad el Incremento de Software
Descripción	Con base en la interpretación de resultados de las pruebas, se valora la estabilidad y la calidad del producto o incremento software desarrollado para la actual iteración. Si el producto satisface los requerimientos funcionales y no funcionales tomados durante la iteración, entonces se publica a toda la comunidad para ser integrado como un componente software estable, que hace parte del producto final. En caso contrario, se inicia nuevamente desde la sub-disciplina del análisis-diseño-implementación para corregir los defectos descubiertos durante las pruebas.
Entradas	▪ Interpretación de Resultados.
Salidas	▪ **Incremento Software**
Roles Responsables	▪ Equipo (i+d)

Fuente (Moreno Chaustre, 2006)

8.5.2 Artefactos para el desarrollo de pruebas

Agile-DISOP define tres artefactos, los cuales son presentados en la **Tabla 7. Artefactos generados en las pruebas del software en Agile-DISOP**.

Tabla 7. Artefactos generados en las pruebas del software en Agile-DISOP

Artefacto	Descripción
Plan De Pruebas de la Iteración	Determina los recursos, componentes, esfuerzo, personal y dinero que se destinan para realizar las pruebas de software durante una iteración.
Registro de los Resultados de las Pruebas	Organiza los datos que arrojan las pruebas hechas al software.
Interpretación de los Resultados de las Pruebas	Consolida y sintetiza los datos de las pruebas mediante gráficos y tablas.

Fuente (Moreno Chaustre, 2006)

CAPÍTULO 9. PLATAFORMA SOFTWARE PARA LA GESTIÓN Y COMUNICACIÓN DE LA COMUNIDAD DESARROLLADORA DEL ESMS-MI

El presente capítulo corresponde al cumplimiento del objetivo:

> *Instanciar y soportar mediante una plataforma software el proceso de desarrollo en comunidad Agile-DISOP[35], con el propósito de promover la puesta en marcha de la comunidad (I+D) que desarrollará el ESMS-MI. Ésta metodología guiará la creación del ESMS-MI permitiendo el trabajo distribuido entre la comunidad[36] que lo desarrolle.*

[35] Metodología propuesta por la tesis "Diseño de una arquitectura para un entorno de modelamiento-simulación y creación de un proceso para su desarrollo por una comunidad (I+D)" **¡Error! No se encuentra el rigen de la referencia.**

[36] Los integrantes de la comunidad estarán distantes espacialmente.

9 PLATAFORMA SOFTWARE PARA LA GESTIÓN Y COMUNICACIÓN DE LA COMUNIDAD DESARROLLADORA DEL ESMS-MI

9.1 INTRODUCCIÓN

Para la realización de un trabajo colaborativo o trabajo en red, lo más importante es la voluntad, el deseo y la motivación de los participantes para alcanzar el logro de sus objetivos. En estos procesos, la comunicación y la coordinación entre sus integrantes juega un papel fundamental y por ende las herramientas y los medios de comunicación.

Cuando los integrantes de la red se encuentran distribuidos geográficamente, una de las mayores dificultades o retos son la comunicación y la coordinación entre sus estos. Pese a que el medio de comunicación no es lo más importante para el éxito en un trabajo colaborativo, si juega un papel fundamental que puede afectar o no, en la consecución satisfactoria de los objetivos. Con la llegada de Internet se ha generado una gran cantidad de servicios, herramientas software que brindan nuevas y diversas formas de comunicación, que permiten la interacción entre personas separadas por grandes distancias por un bajo costo.

En cuanto al rol que juegan las herramientas en el desarrollo de software distribuido o disperso Scott Ambler dice:

> Las herramientas, su uso y la interoperabilidad entre éstas son críticas para el éxito de su proyecto. La administración de la configuración también es crítica para tener éxito (más aún para un equipo de proyecto disperso) por consiguiente, necesita seleccionar una herramienta de control de versiones tempranamente. También necesitará estar de acuerdo mutuamente en las herramientas de comunicación, incluyendo el software para charlar tal como MNS Messenger o AOL Instant Messenger, así como software colaborativo como NetMeeting de Microsoft. Para usar las herramientas, necesita estándares y pautas simples y concisas (como numerar los esquemas para el sistema de control de versiones) y reglas que dictan cuándo colocar el estado de la charla a ocupado. (Ambler, 2002).

Este capítulo define una plataforma software para la gestión y comunicación de la comunidad desarrolladora del ESMS-MI. Esta plataforma no está constituida por una sola herramienta software, sino por un conjunto de herramientas y servicios que se especializan en diferentes tareas.

9.2 PLATAFORMA SOFTWARE PARA LA GESTIÓN Y LA COMUNICACIÓN DE LA COMUNIDAD

Agile-DISOP es un proceso de desarrollo software distribuido, que debe ser instanciado haciendo uso de un conjunto de herramientas que faciliten la gestión y la comunicación de la comunidad, los cuales son los principales desafíos que afronta todo proceso de desarrollo distribuido.

La plataforma software de la comunidad debe apoyar la realización de actividades como la gestión de la comunidad, comunicación entre integrantes, la publicación, el compartir archivos, debe ofrecer herramientas para el desarrollo, control de versiones, la creación y edición de documentos, y la formación y realimentación a partir de experiencias.

Como plataforma software se ha seleccionado un grupo de herramientas y servicios, que en conjunto, abarcan las necesidades de comunicación, gestión de la comunidad y el desarrollo del proyecto ESMS-MI. Para cada una de estas herramientas se deben analizar aspectos como: la facilidad para el trabajo colaborativo, facilidades de acceso (licencias, curva de aprendizaje, acceso remoto…), prestaciones y funcionalidades.

9.2.1 Herramientas para la Gestión de la comunidad

En (Moreno Chaustre, 2006) se examinan varias opciones de herramientas software y recomendó la utilización de **Moodle** para las labores de integración, promoción, comunicación, coordinación y gestión de la comunidad desarrolladora del entorno. Esta herramienta es OpenSource y permite la distribución de archivos, creación de foros, Chat, Wikis, listas de correos (por medio de foros), entre otros. Es una herramienta web que requiere de un servidor web, base de datos MySQL y PHP.

Durante la realización de la presente investigación, se ha colocado en funcionamiento esta herramienta en la dirección: http://simon.uis.edu.co/comunidadesms. La cual ha sido utilizada por algunos miembros de la comunidad desarrolladora del ESMS. En lo que resta de este documento se hará referencia a ésta como “sitio web de la comunidad”.

9.2.2 Herramientas para la comunicación y la coordinación entre los integrantes de la comunidad

Se debe incentivar la comunicación entre los integrantes de la comunidad, no sólo de forma textual, también por otros medios como imágenes, sonidos y videos. Las comunicaciones pueden ser sincrónicas (chat, mensajería instantánea, llamadas, etc) o asincrónicas (Foros, coreo, etc). La comunicación es un reto en actividades no presenciales como son las reuniones entre integrantes de la comunidad, presentaciones, orientaciones o incluso para las discusiones entre los desarrolladores.

En cuanto a las comunicaciones sincrónicas, se recomienda el uso de la herramienta **Skype**, ya que presenta las siguientes características:

- Facilita la comunicación de diversas formas, en una sola herramienta (llamadas, mensajería instantánea, video llamada y video conferencia) con calidad de transmisión (depende de la conexión a internet).
- Se puede establecer diferentes estados de conexión como conectado, ausente, ocupado, etcétera, para configurar dependiendo la disponibilidad del usuario.
- No es invasivo con mensajes de información (por ejemplo cuando se conecta un contacto) o recibir mensajes en momentos no deseados (según el tipo de estado seleccionado).
- El software es libre y el servicio es gratuito.
- Guarda el historial de las conversaciones de forma automática (el usuario puede eliminar las conversaciones que desee).
- Permite transferir archivos (incluso ejecutables) y en caso de caerse la conexión reanuda la transferencia desde el punto en que se encontraba.
- Fácil de manejar. Su funcionamiento es parecido a la mayoría de software de mensajería instantánea disponibles en el mercado, lo cual facilita su aprendizaje.
- Para acceder al servicio debe estar instalado el software. El historial de conversaciones es almacenado en la respectiva máquina.

También es posible la comunicación sincrónica por medio del Chat ofrecido por el sitio web de la comunidad.

Para comunicaciones asincrónicas se puede hacer uso de foros y Wikis. Estas herramientas pueden crearse al interior del sitio web de la comunidad. Ejemplo de uso: para debatir y discutir la toma de una decisión por toda la comunidad (un foro).

9.2.3 Herramientas para compartir archivos

Para compartir archivos debemos examinar primero la naturaleza y fin de los mismos. Si el propósito es compartir documentación para toda la comunidad, se debe hacer uso del sitio web de la comunidad. Pero si se trata de compartir documentación que no son de interés para toda la comunidad o son versiones no acabadas (borradores) usadas entre grupos o desarrolladores, se puede hacer uso de Skype, Office Workspace (recomendado) o incluso el correo electrónico.

Office Workspace es una herramienta en línea que permite la creación de documentos de Office colaborativamente, cuenta con un control de versiones incorporado y los archivos pueden ser editados en la suite de Office de manera local; al presionar “guardar”, los cambios son almacenados directamente en el servidor. El servicio es gratuito, pero debe contar con la suite de Office instalada (sólo para editar ya que puede ser visualizado en línea). También puede compartir archivos de otros tipos como PDF pero no podrá modificarlos.

Otra opción es utilizar Dropbox (Dropbox, 2009) que permite sincronizar archivos en línea y a través cualquier computador conectado a internet, pero a diferencia de los discos virtuales y de otros sistemas de control de versiones, este permite tener los archivos de manera local en una carpeta creada por el programa y al editar un documento y guardarlo, este automáticamente se actualiza en los demás computadores donde instaló el software. También se puede acceder a sus archivos desde cualquier computador o dispositivo móvil utilizando el sitio web de Dropbox. En caso de querer compartir archivos o carpetas solo se debe colocarlos en su Dropbox, e invitar a otras personas para que accedan a estos. También se puede enviar vínculos de archivos específicos a personas desde su Dropbox

9.2.4 Herramientas de desarrollo

9.2.4.1 Herramientas para la planeación de proyectos

Para la gestión del proyecto y seguimiento de cronogramas y tareas, la tesis de Jorge Moreno (Moreno Chaustre, 2006) recomienda la utilización de DotProjet, herramienta Web, libre y de código abierto, alternativa a las herramientas de su tipo pero que son de código propietario. Esta puede ser consultada en http://simon.uis.edu.co/dotproject. Al ser una herramienta Web permite el seguimiento y control del cronograma de trabajo por parte de toda la comunidad, ya que estos pueden ser accedidos y modificados de forma remota. Para los usuarios de MS-Project se facilitará su aprendizaje a sus similitudes, como la presentación del cronograma por medio del diagrama de Grantt.

9.2.4.2 Herramientas de modelado con UML

La herramienta de modelado seleccionada para la elaboración de modelos debe cumplir las siguientes características:

- Utilizar la notación UML: Es la notación más extendida en nuestro país, por lo tanto disminuye la probabilidad de requerir el aprendizaje de una notación de modelado por parte de los integrantes de la comunidad desarrolladora. Específicamente debe aceptar la notación UML2.0.
- Uso gratuito y manejo del formato XMI (Object Management Group, 2005): El uso de herramientas gratuitas facilita que toda la comunidad desarrolladora pueda tener a disposición la misma herramienta. Sin embargo, en caso de que sea utilizada una herramienta de software propietario esta debe poder exportar e importar el modelo en formato XMI.
- Organización de modelos: Que permita organizar los modelos en carpetas, submodelos y/o paquetes, para facilitar el entendimiento del modelo.
- Patrones de diseño: Debe brindar herramientas para facilitar la utilización (y reutilización) de patrones de diseño

Se recomienda el uso de la misma herramienta para toda la comunidad de desarrollo. En caso de no se llegue a un consenso, se debe hacer uso de herramientas que acepten el estándar XMI. Por todo la anterior, se propone el uso

de **StarUML** herramienta que cumple con las especificaciones anteriores y además, brinda las funcionalidades para la generación de código, ingeniería inversa y generación de documentación (por ejemplo documento de casos de uso en Word). StarUML es de fácil manejo y ofrece las funcionalidades de la mayoría de las herramientas de su tipo. Es un software mono usuario y no ofrece funcionalidades para el trabajo de forma remota.

9.2.4.3 Herramientas para el control de versiones al código fuente

Basados en el estudio realizado en el proyecto de pregrado (Niño Diaz, 2008), en el cual se realizó una comparación diferentes herramientas para el control de versiones, se ha seleccionado la herramienta Subversión para llevar el control de versiones del código fuente durante el desarrollo del ESMS. Esta herramienta permite acceder de forma remota a un repositorio de archivos, el cual es como un servidor de archivos ordinario, excepto porque recuerda todos los cambios hechos a sus archivos y directorios (Collins-Sussman, y otros, 2009). Esto permite recuperar versiones antiguas de los datos, o examinar el historial de cambios de los mismos. Subversion puede acceder al repositorio a través de redes o la Web, lo que le permite ser usado por personas que se encuentran en diferentes lugares facilitando el trabajo colaborativo. Esta herramienta es de uso libre y código abierto.

9.2.5 Herramientas para la creación y edición de documentos

Para la creación y edición de documentos es posible utilizar cualquier software para la edición de documentos, pero estos deben soportar el formato de archivo de Office, el cual se propone como estándar de archivo de la comunidad. Para la publicación de documentos definitivos (para la comunidad usuaria) se debe exportar a formato PDF.

Los documentos de construcción colectiva (o colaborativa) como el manual de usuario o el documento de diseño, se recomienda utilizar Wikis. Rodolfo Pilas define a los Wikis en (Herramientas para el desarrollo distribuido y cooperativo de software, 2005) como una aplicación de informática colaborativa que permite crear colectivamente documentos web usando un simple esquema de etiquetas y marcas, sin que la revisión del contenido sea necesaria antes de su aceptación para ser publicado en el sitio web. En este documento también son presentadas las siguientes ventajas de los Wikis para la creación de documentos colectivos:

- Edición simple, rápida y sencilla.
- Control automático de versiones y cambios.
- Hipertexto simple (crear nuevas páginas)
- Índice, Capítulos (espacios wiki) automáticos
- Inclusión de imágenes, exportación, impresión, etc.

9.2.6 Herramientas para la formación y realimentación a partir de las experiencias.

En todo proceso de trabajo en red o en comunidad se genera conocimiento y experiencias que de no ser formalizado y publicado se perderá y por tanto no podrá ser apropiado por la comunidad. La metodología Agile-DISOP plantea como una de sus disciplinas, el mejoramiento continuo del proceso de desarrollo (Moreno Chaustre, 2006). Para estos propósitos se recomienda el uso de foros, Wikis y Blog como medio o recurso de registro y publicación.

9.2.7 Otras posibles herramientas

Existen otras herramientas y servicios web que pueden ser evaluados y utilizados por la comunidad para diferentes propósitos. Por ejemplo existen diferentes formas de compartir escritorios y aplicaciones como Microsoft SharedView o Microsoft NetMeeting, esto puede facilitar explicaciones o debates y discusiones entre los integrantes de la comunidad. Otra posible herramienta es el uso de software para el reporte de errores y solicitud de nuevos requerimientos (bug tracking system) como es el caso de Bugzilla o Mantis.

La inclusión, cambio o eliminación de herramientas de la plataforma de la comunidad es responsabilidad de la misma y de sus integrantes, basados en criterios de usabilidad, funcionalidad y estandarización.

INFRAESTRUCTURA TECNOLÓGICA DISPONIBLE

Para el desarrollo del entorno se cuenta con el servidor Dell PowerEdge 1800 del grupo SIMON. Este servidor cuenta con las siguientes características:

- Procesador: Intel XEON 2.8 GHz, 800 MHz en bus frontal, 2 MB L2 Caché
- Memoria: 1 GB
- Disco duro: 68 GB
- Sistema operativo: Linux Debian V Lenny
- Servidor Web: Apache 2.0, Tomcat 5.5
- Servidor de BD: MySQL 5.0
- PHP 5.0
- Java 1.5

9.3 CONCLUSIONES

La selección de las herramientas y servicios debe ser concertado por la misma comunidad, el éxito de su implementación y uso radica en la utilidad que sus integrantes le encuentren. No obstante, se hace necesario que esta plataforma de comunicación sea estándar para todos, lo cual se logra a partir de una plataforma inicial unificada y el establecimiento de mecanismos de información y discusión sobre la misma.

La comunidad debe definir pautas y estándares para el desarrollo del proyecto, como estilos de codificación, estilos de modelado, formatos de archivos, etc. Estos estándares y pautas deben ser definidos, y al mismo tiempo exigidos, por la misma comunidad desarrolladora.

Hay gran cantidad de opciones de herramientas y servicios para elegir. Algunos de éstos son de libre uso y otros tienen algún costo y con funcionalidades y prestaciones que varían de acuerdo al proveedor. Es importante definir criterios claros definidos por la comunidad, al momento de elegir entre las opciones disponibles.

CAPÍTULO 10. ESTRUCTURA Y DINÁMICA DE LA COMUNIDAD DESARROLLADORA DEL ESMS-MI

El presente capítulo corresponde al cumplimiento de los objetivos:

Orientar el inicio de las actividades de una comunidad interesada en desarrollar sostenidamente, a partir del núcleo del ESMS-MI implementado, las características previstas en los requerimientos funcionales y el diseño lógico del entorno.

Definir las orientaciones para un desarrollo continuado de la plataforma de modelado, así como para el sostenimiento y consolidación de la comunidad desarrolladora.

10 ESTRUCTURA Y DINÁMICA DE LA COMUNIDAD DESARROLLADORA DEL ESMS-MI

10.1 INTRODUCCIÓN

El Inicio de las actividades para el desarrollo de ESMS-MI es viable, en la medida que se establecen unas condicione mínimas para que la comunidad funcione como una red humana, que trabaja en la consecución de un objetivo en común, e igualmente se defina el rol que juega cada uno de los integrantes de la comunidad.

Dentro de las condiciones mínimas se encuentran el definir los medios de comunicación, contar con una plataforma para la integración de la comunidad, la organización de la comunidad y otras relacionadas con el desarrollo de software como: definir los requerimientos iniciales para el entorno, definir la arquitectura, definir los sub-proyectos de desarrollo y sus responsabilidades y las herramientas software a utilizar.

Este capítulo se centra en las orientaciones para el inicio de la comunidad desarrolladora del ESMS-MI, así como en la definición de las orientaciones para la gestión efectiva de la comunidad de desarrollo que posibilita el sostenimiento de la comunidad.

10.2 FASE DE INICIO DEL PROYECTO ESMS-MI

La primera fase del desarrollo de un software que define el proceso de desarrollo Agile-DISOP es la Fase de inicio[37]. La tesis (Moreno Chaustre, 2006) define que el objetivo fundamental de esta fase consiste en alcanzar la estabilidad de los objetivos del sistema mediante el consenso general de toda la comunidad de desarrollo. Durante su realización, la comunidad de desarrollo debe acordar cuestiones relativas a los alcances, riesgos, requerimientos y objetivos del sistema con el propósito de comprobar la viabilidad del proyecto teniendo en cuenta las condiciones del entorno de desarrollo.

Durante esta fase es preciso determinar lo siguiente:

- **El alcance del proyecto.** Determina las características que estarán en el producto y las que no, mediante la captura de los requerimientos más importantes y las restricciones del sistema. El desarrollo de esta actividad, es cubierto en el Capítulo 5 del presente informe.

- **Proponer una arquitectura candidata.** Los casos que son críticos desde el punto de vista de la arquitectura, ayudarán a seleccionar un estilo arquitectónico. La valoración de cuales componentes se van a desarrollar, reutilizar o comprar ayudará a la estimación de los costos y de los recursos para el proyecto. Se sugiere el desarrollo de un prototipo rápido que ayude a la estimación de la viabilidad del proyecto mismo. El desarrollo de esta actividad, es cubierto en el Capítulo 6 del presente informe.

- **Preparar el entorno para soportar la ejecución del proyecto en comunidad.** Valorando y ajustando el proyecto Vs. la comunidad, se seleccionan las herramientas y se decide que partes del proceso se van a adoptar. El desarrollo de esta actividad, es cubierto en el Capítulo 9 del presente informe.

- **Estimar en términos generales el costo, el cronograma y los riesgos.** Para todo el proyecto haciendo énfasis en la siguiente fase de desarrollo, la fase de "Elaboración". Debido a la naturaleza del desarrollo de este proyecto, el cual será desarrollado por proyectos de pregrado y en el marco de investigaciones en cada uno de los grupos que asuma el desarrollo particular de cada sub-proyecto, no es posible determinar un cronograma y costo general del proyecto. Estas mediciones son posibles de realizar una vez sean asignados la totalidad de los proyectos y de esta forma toda la comunidad definirá unos tiempos y costes a nivel general. Para lo anterior, la plataforma software del proyecto cuenta con una herramienta para la gestión de proyectos (administración de cronograma de actividades y

[37] Las fases del proceso de desarrollo Agile-DISOP se presentan en la sección Marco teórico.

recursos de proyectos) llamada DotProject (ver capítulo 9). Para los proyectos vinculados actualmente a la comunidad, se ha establecido su cronograma, el cual puede ser consultado en el sitio web http://simon.uis.edu.co/dotproject..

10.3 DEFINIR LOS SUB-PROYECTOS DEL ESMS-MI

La metodología de desarrollo AgileDisop plantea que un macro-proyecto debe ser abordado por proyectos más pequeños denominados sub-proyectos, los cuales pueden ejecutarse separados geográficamente (Moreno Chaustre, 2006). Consecuentemente, se presenta a continuación el conjunto de sub-proyectos que conforman el desarrollo del macro-proyecto ESMS-MI:

Tabla 8. Listado de sub-proyectos del ESMS-MI

N°	Nombre del sub-proyecto	Estado	Tamaño relativo
1	Entorno del ESMS-MI	No asignado	Grande
2	Editor	No asignado	Grande
3	Simulador	No asignado	Mediano
4	Motor de simulación del ESMS-MI	En desarrollo	Mediano
5	Evaluador de expresiones matemáticas	En desarrollo	Pequeño
6	Generador de reportes	No asignado	Pequeño
7	Submodelo de MBOR	No asignado	Grande
8	Submodelo de Lógica difusa	No asignado	Pequeño
9	Herramientas del entorno	No asignado	Mediano

Fuente: Autor

Los sub-proyectos listados anteriormente deberán desarrollar un componente o conjunto de estos, según la descripción que aparece a continuación. Estos subsistemas hacen parte de la arquitectura del ESMS-MI, por lo tanto, para una

mayor descripción de los mismos, se debe consultar el documento de descripción arquitectónica del entorno mencionado.

Los proyectos 4 y 5 corresponden al núcleo del entorno el cual se encuentra actualmente en desarrollo por medio de proyectos de pregrado en la EISI de la UIS. Los desarrolladores de estos proyectos son los primeros integrantes de la comunidad, por tanto se encuentran haciendo uso de la plataforma software de integración de la comunidad y han aportado a la selección, implementación y modo de uso de la misma.

10.3.1 Sub-proyecto Entorno del ESMS-MI

- Estado: No asignado
- Tamaño relativo: Grande
- Objetivo: desarrollar el entorno general del ESMS-MI el cual comprende la interfaz gráfica de usuario y los subsistemas: Interfaz dinámica, Manejador de extensiones, Enrutador, Manejador de errores, Coordinadores (de modelado y simulación, simuladores e intérpretes), e Interpretes (Persistencia). Además, este proyecto debe integrar todos los componentes desarrollados en los otros sub-proyectos (ver **Tabla 8. Listado de sub-proyectos del ESMS-MI**).

Interfaz dinámica

La interfaz dinámica permite crear y administrar las Interfaces Gráficas de Usuario (IGU) creadas dinámicamente. Las IGU son creadas en tiempo de ejecución a partir de un archivo o cadena de texto con un script con código XIGU. Un archivo XIGU define una interfaz de usuario independiente de la plataforma, está escrito en el estándar XML y es interpretado por un ParserXIGU, el cual convierte el script en una interfaz real. La interfaz generada automáticamente es acorde a la interfaz de todo el entorno (el mismo estilo de componentes).

Manejador de extensiones

Este subsistema es el encargado de gestionar la instalación de las extensiones, habilitarlas/deshabilitarlas, desinstalarlas y coordinar la ejecución su ejecución.

Intérpretes

El subsistema de persistencia está conformado por Agentes encargados de gestionar los diferentes tipos de archivos soportados por el entorno.

10.3.2 Sub-proyecto Editor

- Estado: No asignado

- Tamaño relativo: Grande

Objetivo: Desarrollar el editor del entorno que permitirá introducir de forma gráfica un modelo. Este proyecto incluye componentes como el editor de modelos estructurados y todos los componentes necesarios para la transformar modelos en diferentes representaciones.

Editor

El subsistema Editor permite al usuario realizar de forma gráfica e interactiva un modelo de simulación. Este subsistema contiene un conjunto de herramientas con los cuales el usuario pueda realizar un modelo de forma gráfica. El editor debe permitir crear un modelo por el método de arrastrar-soltar o seleccionar-colocar y por medio de comandos de consola.

Responsabilidades

- Proveer interfaz gráfica de interacción sistema-usuario.
- Permite hacer modelos en los diferentes lenguajes de la D.S.
- Coordinar y sincronizar los diferentes modelos y diagramas.
- Generar un modelo a partir del diagrama introducido por el usuario.
- Administración de escenarios.

10.3.3 Sub-proyecto Simulador

- Estado: No asignado
- Tamaño relativo: Mediano
- Objetivo: Desarrollar un componente software reutilizable para un ESMS-MI por medio de la D.S. El componente debe ser multiplataforma y trabajar integrado al entorno o de forma independiente, recibir como entrada el modelo, presentar los resultados de forma gráfica y animada a través de diferentes controles de presentación de resultados y permitirle al usuario interactuar con la simulación (poder controlar la simulación y modificar los valores que obtienen las variables durante la simulación). Este proyecto hace uso del componente motor generado por el sub-proyecto “Motor de simulación”.

El subsistema Simulador permite realizar simulaciones a partir de un modelo suministrado. Este sub-sistema tiene diversas formas de presentación de los resultados, ya sea de forma tabular, gráfica o por medio de animaciones parametrizadas, es decir, la animación está regida por los valores que se obtienen de la simulación. Tiene una interfaz gráfica que le permite al usuario interactuar con la simulación a través de controles. Algunos de estos permiten controlar la

simulación y otros permiten modificar los valores de las variables del modelo. El simulador encapsula los diferentes motores de simulación y las diferentes interfaces o presentadores de resultados.

Responsabilidades

- Realizar simulación de modelos integrados y otros tipos de modelos según los servidores externos implementados.
- Presentar por medio de interfaces gráficas y en diferentes plataformas Software-Hardware, los resultados de la simulación.
- Permitir a los usuarios interactuar con la simulación (cambiar valores de las variables durante la simulación y/o controlar la simulación) a través de controles.
- Interactuar con otros simuladores, local y remotamente.
- Permitir interactuar con otras aplicaciones (enviar y recibir datos durante la simulación).

10.3.4 Sub-proyecto Motor de simulación

- Estado: En desarrollo
- Tamaño relativo: Mediano
- Objetivo: Desarrollar el motor de simulación para un ESMS-MI por medio de la D-S, en términos de un componente software reutilizable. El motor debe ser multiplataforma, trabajar tanto integrado al entorno como de forma independiente. Recibe como entrada el modelo y entregar los resultados generados a otros componentes del entorno u otras aplicaciones.

10.3.5 Sub-proyecto Evaluador de expresiones matemáticas

- Estado: En desarrollo
- Tamaño relativo: Pequeño
- Objetivo: Desarrollar un componente software basado en un modelo matemático que permita interpretar y evaluar expresiones matemáticas, utilizando funciones y operadores empleados en el lenguaje matemático convencional y que incluya la verificación de la consistencia dimensional.

10.3.6 Sub-proyecto Generador de reportes

- Estado: No asignado

- Tamaño relativo: Pequeño
- Objetivo: El subsistema Generador de reportes permite generar un documento que contiene datos sobre el proceso de modelado y simulación, tales como imágenes del modelo, ecuaciones, descripciones de elementos, resultados de una simulación, gráficas de resultados, etcétera. Este reporte es personalizable, permitiendo al usuario seleccionar las partes que desea que aparezcan en éste. También, soporta el uso de plantillas predefinidas para la elaboración de reportes. El documento generado se debe poder exportar a diferentes tipos de archivos como: Open Document, MS-Word y Html.

Responsabilidades

- Generación de reportes en diferentes formatos.
- Configuración de reportes.
- Administrar plantillas de reportes

10.3.7 Sub-proyecto Sub-modelo MBOR

- Estado: No asignado
- Tamaño relativo: Grande
- Objetivo: Desarrollar el conjunto de componentes (motor de simulación y simulador de HOMOS) relacionados con el MBOR. Estos componentes hacen parte tanto del software ESMS-MI como de la segunda versión del software HOMOS.

10.3.8 Sub-proyecto Sub-modelo LD

- Estado: No asignado
- Tamaño relativo: Pequeño (puede ser abordador por el sub-proyecto Motor o Simulador)
- Objetivo: Desarrollar el conjunto de componentes (motor de simulación LD y simulador de LD) relacionados con el modelado y la simulación con LD. Estos componentes hacen parte tanto del software ESMS-MI como de la Herramienta software para el desarrollo de controladores difusos mencionada al principio de la Suite de modelado y simulación mencionada en la sección 6.4.3 Framework de modelado y simulación.

10.3.9 Sub-proyecto Herramientas del entorno

- Estado: No asignado

- Tamaño relativo: Mediano
- Objetivo: Desarrollar componentes para la realización de análisis de sensibilidad de modelos, para generación de árboles de relaciones entre variables y la calibración de escenarios haciendo por medio de algoritmos genéticos para el ESMS-MI. Estos componentes deben ser implementados como extensiones (add_on) para el entorno software.

10.4 CONVOCATORIA DE PROYECTOS

Por el conjunto de áreas y de tecnología especializada involucradas, el desarrollo del ESMS-MI requiere de un equipo que incluya especialistas en cada una de éstas, lo cual no es común encontrar en un sólo grupo de investigación o incluso una universidad. Por lo tanto, se definió en (Moreno Chaustre, 2006) que el desarrollo del entorno esté a cargo de una comunidad conformada por grupos Investigación y desarrollo (I+D) de diferentes universidades del país.

Para la realización de estos proyectos, varios grupos de investigación han mostrado interés de participar en la realización del ESMS.MI. Entre estos se encuentra el grupo de Investigación en Tecnologías de la Información – GTI de la universidad del Cauca al cual está vinculado el Ing. Jorge Jair Moreno autor de la tesis (Moreno Chaustre, 2006).

Para la realización de la convocatoria se ha creado un conjunto de documentos como son:

- Presentación con diapositivas del macro-proyecto ESMS-MI, la cual incluye la presentación de cada uno de los sub-proyectos. (ver **Anexo D**).
- Formato para la Solicitud de participación de cada uno de los integrantes de la comunidad. (ver **Anexo E**).
- Orientaciones de uso de la plataforma Moodle para el desarrollo del ESMS. (ver **Anexo F**).

10.5 ASPECTOS PARA LA VINCULACIÓN DE VARIAS UNIVERSIDADES AL PROYECTO ESMS-MI

Pese a que las condiciones técnicas, metodológicas y organizativas están dadas, para el inicio del desarrollo del ESMS-MI por parte de una comunidad geográficamente distribuida[38], y existe el interés de grupos de investigación de otras universidades en participar, se hace necesario subscribir un convenio que

[38] De hecho la comunidad ya está en funcionamiento, iniciando actividades con el desarrollo del núcleo del entorno.

apruebe la cooperación entre los grupos de investigación asociados a cada una de las universidades.

Cabe aclarar que los objetivos de la presente tesis están enfocados en brindar las condiciones necesarias para el inicio de las actividades de la comunidad desarrolladora, por lo que la gestión de dicho convenio rebasa los objetivos de la presente investigación.

10.6 GESTIÓN DE LA COMUNIDAD DE AGILE-DISOP

El propósito fundamental de una de las disciplinas de Agile-DISOP[39] consiste en asegurar la gestión efectiva de la comunidad de desarrollo, mediante la puesta en marcha de mecanismos adecuados para favorecer la sociabilidad de los equipos (I+D) bajo un ambiente distribuido. Algunos de estos mecanismos estarán soportados por servicios informáticos publicados en la plataforma de software integración de la comunidad descrita en el capítulo 9. La sociabilidad de una comunidad de desarrollo enfatiza en la interacción social. Las comunidades con una buena sociabilidad establecen políticas que dan soporte al propósito mismo de la comunidad, son entendidas y aceptadas por todos los miembros y finalmente, son lo suficientemente prácticas como para ser implementadas con facilidad. La sociabilidad evoluciona con la comunidad, pero siempre enfatizará en el favorecimiento de su propósito. (Moreno Chaustre, 2006).

En la medida que evoluciona la comunidad, esta debe irse configurando bajo los lineamientos establecidos por la metodología de desarrollo. Se deben establecer los roles de los integrantes, como el **R.I.C.P**[40] o comité organizador y la configuración de cada equipo de desarrollo (ver la sección 3.3.5 en el capítulo FUNDAMENTOS TEÓRICOS). Inicialmente, el R.I.C.P está conformado por:

- Profesor Hugo Hernando Andrade Sosa
- Profesor Jorge Jair Moreno Chaustre
- Ingeniero Emiliano Lince Mercado

Y la configuración escogida para los grupos de desarrollo es “Programador Jefe”. El equipo con programador jefe resulta adecuado para proyectos creativos, en los que tener un cerebro al frente ayudará a proteger la integridad conceptual del sistema. También resulta adecuado para proyectos de ejecución táctica, en los que el programador jefe puede ser una especie de dictador que diseñe los medios más expeditivos para lograr la culminación del proyecto (Moreno Chaustre, 2006).

[39] Denominada Gestión de la comunidad. Las disciplinas de Agile-DISOP pueden ser consultadas en el marco teórico de la presente tesis.
[40] Acrónimo de: *“Responsable de la Integridad Conceptual del Producto”*

Es posible agregar a la comunidad grupos de desarrollo con otros tipos de configuración[41], en la medida que estos sean necesarios y la evolución misma de la comunidad.

[41] Otros tipos de configuración pueden ser consultados en(Moreno Chaustre, 2006).

10.7 CONCLUSIONES

La presente investigación ha conseguido brindar las condiciones necesarias para el desarrollo del proyecto ESMS-MI por una comunidad geográficamente distribuida. Pero para viabilizar la realización del proyecto por parte de varios grupos de investigación en diferentes universidades, es necesario subscribir un convenio entre las universidades implicadas.

No siempre cada sub-proyecto corresponde a un módulo de la arquitectura software, algunos de estos abarcan un conjunto de módulos. Además, se hace necesario la realización de un sub-proyecto encargado de realizar la integración de todos los módulos.

Se deben definir compromisos claros con la ejecución de cada uno de los sub-proyectos y compromisos en cuanto al cumplimiento de sus respectivos cronogramas de trabajo e igualmente compromisos de comunicación y participación permanente con la comunidad.

El desarrollo continuado del ESMS-MI lo garantiza la relación entre la comunidad desarrolladora con la comunidad usuaria. Igualmente, ciertos criterios en la homogeneidad del desarrollo que faciliten que nuevos desarrolladores retomen el desarrollo de nuevas módulos y versiones. La gestión de los requerimientos y la gestión de los errores reportado por los usuarios, la promoción de nuevos desarrollos y la reflexión de los investigadores y desarrolladores sobre el uso de la D.S

CAPÍTULO 11. CONCLUSIONES Y RECOMENDACIONES GENERALES

- El desarrollo de este nuevo entorno ofrecerá mecanismos técnicos que posibilitan la integración de herramientas matemáticas como la LD y las RN, entre otras, a través de la D.S, que por sus características y naturaleza, son trabajadas de forma independiente. Así, se crean nuevas posibilidades para el modelado y la simulación, sus usos y las aplicaciones en un nuevo contexto: la integración de modelos expresados en diferentes representaciones matemáticas.

- Pese a que el desarrollo distribuido ofrece beneficios en cuanto a los recursos con los que se cuenta (principalmente el recurso humano), también trae consigo algunos retos relacionados con la comunicación y la coordinación de sus integrantes. Por lo tanto, para el buen funcionamiento y el sostenimiento de la comunidad, se debe fomentar la comunicación entre sus integrantes y el uso de la plataforma software de integración. Igualmente, se deben definir estándares para la documentación, prueba, diseño y codificación, basados en las propuestas aquí establecidas y concertados por la comunidad, la cual debe evaluarlos de forma permanentemente a lo largo de todo el proceso de desarrollo.

- Se recomienda continuar con el desarrollo de más ciclos del proceso de investigación-acción, en esta ocasión realizando un mayor énfasis en el desarrollo de software, posibilitando la implementación de propuestas de cambio en el uso y aplicación de la D.S, producto de las mismas investigaciones. Asimismo, se recomienda la vinculación de la mayor cantidad de proyectos en paralelo, con el fin de abordar la complejidad del entorno por medio del trabajo colaborativo, creando de esta manera una comunidad de proyectos no sólo de ingeniería de software, sino también de otras áreas como por ejemplo de diseño industrial para la creación del material de publicidad y el diseño de interfaces de usuario.

Finalmente y en cuanto a las preguntas de investigación formuladas, se concluye:

- P1 ¿Cómo debe ser la arquitectura de una plataforma software para el modelado estructural y simulación, centrada en D.S, que facilite su desarrollo distribuido?

  La arquitectura diseñada tiene algunas características que permiten dar respuesta a este interrogante, como por ejemplo: es una arquitectura basada en componentes claramente definidos, su arquitectura del sistema es PAC-Heterogenea, en la cual los agentes PAC son componentes complejos cada uno con su propia arquitectura. Esta arquitectura posee los siguientes beneficios: Separación de conceptos semánticos del dominio de la aplicación, soporta el cambio y la extensión y soporta la multitarea.

- P2 ¿Cuáles deben ser los requerimientos y desarrollos básicos (análisis, diseño, implementación herramientas de soporte, etc.) para garantizar el inicio de un desarrollo distribuido por una comunidad dispersa geográficamente?

  La metodología para el desarrollo en comunidad debe encontrar el balance entre diferentes aspectos del desarrollo de software. Por ejemplo, no es recomendable realizar un súper diseño al inicio del proyecto, pero es igualmente poco recomendable sólo hacer el diseño para cada iteración. En el desarrollo geográficamente distribuido se debe contar con un diseño base que permita unificar criterios, y los candidatos ideales para esto son la arquitectura del sistema y los bocetos del sistema. Estos dos instrumentos permiten ver los aspectos globales del sistema y se convierten en un mecanismo de coordinación y comunicación entre los diferentes integrantes. En consecuencia, es necesaria una obtención de un conjunto de requisitos inicial que permita a la comunidad desarrolladora visionar las dimensiones de todo el sistema e identificar posibles riesgos de manera temprana. Hay que tener en cuenta que los diversos estilos de arquitecturas software brindan diversos niveles de satisfacción de requerimientos y de calidad del software.

  Es común encontrar que los desarrollos distribuidos dan inicio con un desarrollo base, en términos de un núcleo o plataforma que se convierte en la columna vertebral del desarrollo, lo cual brinda un clima de confianza y un incentivo a la comunidad desarrolladora. Pese a que los aspectos técnicos de la plataforma software de integración no son (ni deben ser) los más importantes en un desarrollo en comunidad, pueden dificultar o facilitar la comunicación entre los integrantes de la misma, por lo que no deben ser evidentes ni mucho menos entorpecer la interacción de los usuario de los servicios que provee la plataforma software de integración de la comunidad.

- P3 ¿Cuáles deben ser las orientaciones de diseño para el desarrollo de software de usuario final basados en la arquitectura definida?

  El diseño arquitectónico propuesto por esta tesis, está pensado y estructurado para favorecer la interoperabilidad y el uso de sus componentes[42] por otras aplicaciones para usuario final, como lo son algunas de las aplicaciones y herramientas software de la suite de modelado y simulación planteada en el presente documento.

[42] Denominados “Agentes” en la arquitectura PAC.

BIBLIOGRAFÍA

A standard simulation environment: a review of preliminary requirements. **Haigh, Peter L, y otros. 1994.** 1994. The winter simulation conference. págs. 664- 672.

Alfonso, Angélica y Calderón, Jacqueline. 2003. *Análisis de Sensibilidad y Diagramas causales para la herramienta de Modelado y Simulación, EVOLUCION.* Universidad Industrial de Santander. Bucaramanga : s.n., 2003. Tesis pregrado. Nº inv: 111069.

Ambler, Scott. 2002. Bridging the Distance. *Dr. Dobb's - Architecture & Design.* [En línea] 12 de Agosto de 2002. [Citado el: 5 de Abril de 2009.] http://www.ddj.com/architect/184414899.

An examination of environment design and the applicability of software engineering requirements to the standard simulation environment. **Arthur, James D. 1994.** 1994. The winter simulation conference.

Anacleto, Valerio Adrián. 2009. El rol de la arquitectura de software en las metodologías ágiles. Lineamientos para su implementación. *sitio web Epidata Consulting.* [En línea] 2009. www.epidataconsulting.com.

Andrade Sosa, Hugo Hernando y Gómez Flórez, Luis Carlos. 1990. *Dinámica de Sistemas aplicada a algunos fenómenos de transporte.* Universidad Industrial de Santander. Bucaramanga (Colombia) : s.n., 1990. Tesis de maestría. Nº clasificación: IN04412.

—. 2009. *Tegnología Informática en la escuela.* Cuarta edición. Bucaramanga : Computadores para educar, 2009. ISBN: 978-958-44-0833-4.

Andrade Sosa, Hugo Hernando, y otros. 2001. *Pensamiento Sistémico: Diversidad En Búsqueda De Unidad.* Bucaramanga : Ediciones Universidad Industrial De Santander, 2001. ISBN 958-9318-78-9.

Ardila Arango, Carlos Humberto y Duran Sanchez, Pedro Enrique. 1995. *Herramienta software para construir y analizar modelos mediante Dinámica de Sistemas: "Evolución 2.0".* Universidad Industrial de Santander. Bucaramanga : s.n., 1995. Tesis de pregrado. Nº Inv: S 05999.

Ardila Arango, Maria Isabel y Moreno Suarez, William. 2000. *Evolución 3.0. Herramienta Software para el Modelamiento Y Simulación con Dinámica de Sistemas.* Universidad Industrial de Santander. Bucaramanga : s.n., 2000. Tesis pregrado. Nº inv: 99056.

Arias Blanco, Alberto Julio y Ortiz Fonseca, Rafael. 1995. *Herramienta software para la representación solución y análisis de sistemas dinámicos en la*

forma espacio-estado. Universidad Industrial de Santander. Bucaramanga : s.n., 1995. Tesis de pregrado.

Bass, Len y Coutaz, Joëlle. 1991. *Developing software for the user interface.* s.l. : Addison-Wesley Publishing Company, 1991.

Bass, Len, Clements, Paul y Kazman, Rick. 2007. *Software architecture in practice.* Segunda edición. Boston : Addison-Wesley Professional, 2007. pág. 560. 978-0321154958.

Beck, Kent, y otros. 2001. Manifesto for Agile Software Development. *Sitio Web The Agile Manifesto.* [En línea] 2001. http://agilemanifesto.org/.

Buschmann, Frank, y otros. 1996. *Pattern-Oriented Software Architecture Volume 1: A System of Patterns.* West Sussex : John Wiley & Sons Ltd, 1996. pág. 476. Vol. 1. ISBN: 978-0471958697.

Cabarcas Alvarez, Amaury y Diaz Marino, Nestor M. 2000. *Micromundo para el desarrollo del pensamiento sistémico soportado en el modelamiento basado en objetos y reglas.* Universidad Industrial de Santander. Bucaramanga : s.n., 2000. Nº Inv: S 10099.

Cala Amaya, Jenny Milena y Tasco Baron, Jairo. 2008. *Ambiente software apoyado en el modelado y la simulación para el aprendizaje de las ciencias de la naturaleza en la educación básica secundaria y media vocacional. Un enfoque Dinámico- Sistémico.* Universidad Industrial de Santander. Bucaramanga : s.n., 2008. Tesis pregrado. Nº clasificación: S 18095.

Casal Terreros, Julio. 2007. Desarrollo de Software basado en Componentes. *Microsoft Developer Network en Español.* [En línea] 2007. http://www.microsoft.com/spanish/msdn/comunidad/mtj.net/voices/MTJ_3985.asp.

Collins-Sussman, Ben, Fitzpatrick, Brian W. y Pilato, C. Michael. 2009. Control de versiones con Subversion. [En línea] 2009. http://svnbook.red-bean.com/nightly/es/index.html.

Consejo Superior de Investigaciones Científicas. Publicaciones individuales. *Sitio Web del Instituto de filosofía-Consejo Superior de Investigaciones Científicas.* [En línea] http://www.ifs.csic.es/Personal/mosterin.htm.

Criteria for simulation software evaluation. **Nikoukaran, Jalal, Hlupic, Vlatka y Paul, Ray J. 1998.** 1998. 1998 Winter Simulation Conference. págs. 399-406.

Cuellar Yeneris, Mario y Lince Mercado, Emiliano. 2003. *Evolución 3.5 Herramienta Software para el modelamiento y simulación con Dinámica de sistemas.* Universidad Industrial de Santander. Bucaramanga : s.n., 2003. Tesis de pregrado.

Dropbox. 2009. Sitio Web Dropbox. [En línea] 2009. http://www.getdropbox.com/.

Duarte Mogotocoro, Carmen Elena y Lozano Avellaneda, Oscar Armando. 1998. *HOMOS 1.0 herramienta software para el modelamiento y simulación basado en objetos y reglas.* Universidad Industrial de Santander. Bucaramanga : s.n., 1998. Tesis de pregrado. Nº clasificación: S 07944.

Dueñas Amaris, Ivonne Zulay y Rojas Mendoza, Luisa Mireya. 2000. *Software Educativo de apoyo para el Aprendizaje del área de Ciencias en los grados octavo y noveno basado en el Modelo Educativo Construtivista Centrado en los Procesos de Aprendizaje.* Universidad Industrial de Santander. Bucaramanga : s.n., 2000. Teis de pregrado.

Dyner, Isaac y Perez, Ana Lucia. 2004. *Boletín Informativo del Capítulo Latinoamericano de la Sociedad de Dinámica de Sistemas.* [Electrónico] Noviembre de 2004. SisTemas.

El modelado con Dinámica de Sistemas, en su integración con otras herramientas matemáticas. **Andrade Sosa, Hugo Hernando, y otros. 2005.** Cartagena : s.n., 2005. Tercer congreso latinoamericano de Dinámica de Sistemas.

Fernández de Valdenebro, Maria Verónica. 2005. *Aplicación de ATAM para una evaluación arquitectónica de la herramienta para entornos de modelado y simulación Evolucion 3.5.* Universidad del Cauca. Popayán : s.n., 2005. Tesis pregrado.

Foundations for the study of software architecture. **Perry, Dewayne E y Wolf, Alexander L. 1992.** 4, New York : ACM, Octubre de 1992, ACM SIGSOFT Software Engineering Notes, Vol. 17, págs. 40-52. ISSN:0163-5948.

Fowler, Martin. 2000. The New Methodology. *Sitio Web de Martin Fowler.* [En línea] Julio de 2000. [Citado el: 23 de Febrero de 2009.] http://martinfowler.com/articles/newMethodology.html.

Framework para el desarrollo de ambientes software de aprendizaje y toma de decisiones con modelos en Dinámica de Sistemas. **Andrade Sosa, Hugo Hernando, Lince Mercado, Emiliano y Gómez Prada, Urbano Eliecer. 2009.** Habana (Cuba) : s.n., 2009. XIII Congreso internacional de informática educativa "InforEdu 2009".

Garlan, David y Shaw, Mary. 1994. *An Introduction to Software Architecture.* Pittsburgh : s.n., 1994. Reporte tecnico. CS-94-166.

Gelvez Pinto, Lilia Nayibe y Muskus Morales, Zandy Cecilia. 1994. *Construcción y simulación de un modelo para el control de la Leishmaniasis bajo un enfoque sistemático.* Universidad Industrial de Santander. Bucaramanga : s.n., 1994. Tesis de pregrado. Nº inv: S 05549.

Gómez Prada, Urbano Eliecer y Barragán Tarazona, Omar Augusto. 2002. *Modelo de simulación de sistemas de producción de ganadería bovina para la investigación integral.* Universidad Industrial de Santander. Bucaramanga (Colombia) : s.n., 2002. Tesis pregrado.

Gravendeel, Eelco. 2009. What documents to write in an Agile environment. *Xebia Blog.* [En línea] 9 de 8 de 2009. http://blog.xebia.com/2009/08/09/agile-documentation-the-good-the-bad-and-the-ugly/.

Hernandez Cuadrado, Alexander Elias y Monsalve Quintero, Adriana Judith. 2009. *Mantenimiento del software Evolución 4.0.* Universidad Industrial de Santander. Bucaramanga : s.n., 2009. Plan de proyecto aprobado.

Herramientas para el desarrollo distribuido y cooperativo de software. **Pilas, Rodolfo. 2005.** Iquique, Chile : s.n., 2005. Sexto encuentro nacional Linux 2005.

Hussey, Andrew y Carrington, David. 1996. *Comparing two user-interface architectures: MVC and PAC.* Queensland : s.n., 1996. Reporte técnico.

IEEE Computer Society. 1991. *Computer Dictionary - Compilation of IEEE Standard Computer Glossaries.* New York : IEEE, 1991. IEEE Std. 610-1991.

—. 2000. *Recommended Practice for Architectural Description of Software-Intensive Systems.* New York : IEEE, 2000. pág. 34, Estandar. IEEE Std 1471-2000 / ISO/IEC 42010:2007(E) / ISBN 0-7381-2519-9.

—. 1998. *Recommended Practice for Software Requirements Specifications.* Software Engineering Standards Committee, IEEE, inc. New York : IEEE, 1998. pág. 38, Estandar. IEEE 830-1998/ISBN 0-7381-0448-5, SS94654.

—. 1990. *Standard Glosary of Software Engineering Terminology.* New York : IEEE, 1990. IEEE Std. 610.12-1990.

Ingenieria de sistemas -realidad virtual y aprendizaje-. **Andrade Sosa, Hugo Hernando y Navas Garnica, Ximena Marcela. 2002.** 1, Bucaramanga, Colombia : Universidad Industrial de Santander, 2002, Revista UIS Ingenierías, Vol. 1. ISSN 1657 - 4583.

Inquiry-based requirements analysis. **Potts, Collin, Takahashi, Kenji y Anton, Annie I. 1994.** 2, Atlanta : IEEE, Marzo de 1994, IEEE Software, Vol. 11, págs. 21-32. ISSN: 0740-7459.

Integración de la dinámica de sistemas y la lógica difusa. Evolución con FIS. **Andrade Sosa, Hugo Hernando, Machado Mendoza, Gesman David y González Pérez, Cesar Eduardo. 2007.** Medellin (Colombia) : s.n., 2007. 5º Encuentro Colombiano de Dinámica de Sistemas.

Integración entre la Dinámica de Sistemas y otras matemáticas para la representación del conocimiento. **Rivera, Cesar Augusto, Sarmiento Villamizar, Freddy y Andrade Sosa, Hugo Hernando. 2004.** Talca (Chile) : s.n., 2004. Segundo congreso latinoamericano de Dinámica de Sistemas.

Integrando Dinámica de Sistemas y lógica Fuzzy, en tiempo de modelado y de simulación, un ejercicio de clase. **Alcazar Rivas, Julio, y otros. 2004.** Santa Marta (Colombia) : s.n., 2004. Segundo Encuentro Colombiano de Dinámica de Sistemas.

Introduction to modeling and simulation. **Carson II, John S. 2005.** 2005, Marietta : Winter Simulation Conference, 2005, Winter Simulation Conference.

Introduction to simulation. **Gogg, Thomas J. y Mott, Jack R. A. 1993.** 1993, Palos Verdes : Winter Simulation Conference, 1993, Winter Simulation Conference.

—*.* **Seila, Andrew F. 1995.** 1995. The 1995 Winter Simulation Conference. págs. 7-15.

—*.* **Shannon, Robert E. 1992.** Texas : s.n., 1992, Winter Simulation Conference, págs. 65-73.

Luque Y Guzman Saenz, Guillermo Gustavo y Pradilla Valbuena, Vladimir Eduardo. 2003. *Modelo conceptual generalizado de un sistema para soportar la toma de decisiones. Enfoque integrador entre la Dinámica de Sistemas y la Inteligencia Artificial.* Universidad Industrial de Santander. Bucaramanga (Colombia) : s.n., 2003. Tesis pregrado.

Machado Mendoza, Gesman David y González Pérez, Cesar Eduardo. 2006. *Componente de sistema de inferencia difusa (FIS) para Evolución 3.5.* Universidad Industrial de Santander. Bucaramanga : s.n., 2006. Tesis de pregrado. Nº Inv: S15913.

Malan, Ruth y Bredemeyer, Dana. 2004. The Visual Architecting Process. [En línea] 19 de Abril de 2004. [Citado el: 22 de Enero de 2008.] Libro digital. http://www.ruthmalan.com.

Martínez Beltrán, Rosa Cecilia y Vanegas Vega, Jorge Heriberto. 1997. *Diseño y elaboración de la interfase grafica y material de marketing para la herramienta software EVOLUCION 3.0.* Universidad Industrial de Santander. Bucaramanga : s.n., 1997. Tesis pregrado. Nº inv: DI07518.

MITOpenCourseware. 2001. Curso práctico en Ingeniería de Software. Otoño de 2001. *Boletín S7: Cómo documentar un sistema de software.* [En línea] 2001. http://mit.ocw.universia.net/6.170/6.170/f01/related-resources/documentation.html.

Modelling styles and their support in the CASM enviroment. **Balmer, David W. 1987.** 1987, 1987, Winter Simulation Conference, págs. 478-485.

Moodle.org. Developer. *Sitio Web de Moodle.org.* [En línea] http://moodle.org.

Moreno Chaustre, Jorge Jair. 2006. *Diseño de una arquitectura para un entorno de modelamiento- simulación y creación de un proceso para su desarrollo por una comunidad (I+D).* Universidad Industrial de Santander. Bucaramanga : s.n., 2006. Tesis Maestría.

Mosterín, Jesús. 1987. *Conceptos y teorías en la ciencia.* Segunda edición. Madrid : Alianza Editorial, 1987. ISBN: 84-206-2394-6.

Mozilla Hispano. 2007. Preguntas frecuentes sobre la organización Mozilla. *Sitio Web de Mozilla Hispano.* [En línea] 8 de Agosto de 2007. http://www.mozilla-hispano.org/documentacion/Preguntas_frecuentes_sobre_la_organizaci%C3%B3n_Mozilla.

Multifacetted, multiparadigm modelling perspectives: Tools for the 90's. **Zeigler, Bernard P. y Oren, Tuncer I. 1986.** 1986. 1986 Winter Simulation Conference. págs. 708-712.

Navas Garnica, Ximena Marcela. 2006. *Propuesta informática para la educación en el cambio, basada en ambientes de Modelado y Simulación. Un enfoque Sistémico.* Universidad Industrial de Santander. Bucaramanga (Colombia) : s.n., 2006. Tesis de maestría.

Navas Garnica, Ximena Marcela y Benitez Gonzalez, Fabian. 2000. *Micromundo de simulación para el aprendizaje de ciencias de la naturaleza para los grados sexto y séptimo, un enfoque sistémico.* Universidad Industrial de Santander. Bucaramanga : s.n., 2000. Tesis de pregrado. Nº Inv: S 10100.

Niño Diaz, Angélica María. 2008. *Prototipo Web para control de versiones software y documentación en línea, aplicado al servidor de la Escuela de Ingeniería de Sistemas – UIS, Cormoran.* Universidad Industrial de Santander. Bucaramanga : s.n., 2008. pág. 156, Tesis de pregrado.

Nuseibeh, Bashar y Easterbrook, Steve. 2000. *Requirements Engineering: A Roadmap.* Limerick, Ireland : International Conference on Software Engineering, 2000. págs. 35 - 46.

Object Management Group. 2005. *XML Metadata Interchange (XMI®).* OMG. Needham : s.n., 2005. Especificación. ISO/IEC 19503.

Ortiz Ramos, John Fredy y Suarez Celis, Diana Carolina. 2008. *MICHRO 2.0, micromundo para el desarrollo del pensamiento sistémico soportado en el modelamiento basado en objetos y reglas.* Universidad Industrial de Santander. Bucaramanga : s.n., 2008. Tesis de pregrado. Nº clasificación: S 18639.

Ospino Reales, Merilin y Guillermo Prada, Carlos. 2006. *HCAIAD: Herramienta para la creación de Ambientes Educativos informáticos con aprendizaje dinámicos.*

Universidad Industrial de Santander. Bucaramanga : s.n., 2006. Tesis pregrado. Nº clasificación: S 15467.

Pineda Gomez, Amolfi Hernando y Rueda Nuñez, Blanca Ines. 2000. *Interfaz grafica para Evolución 2.0 Y SIMUIS 1.0.* Universidad Industrial de Santander. Bucaramanga : s.n., 2000. pág. 64, Tesis de pregrado. Nº clasificación: S 09596.

Problems for the future of system dynamics. **Richardson, George P. 1998.** 2, Albany (USA) : s.n., 12 de 1998, System Dynamics Review, Vol. 12, págs. 141-157.

Purcell, Edwin J y Varberg, Dale. 1992. *Cálculo con geometría analítica.* [ed.] José Tomás Pérez Bonilla. Sexta edición. Naucalpan de Juárez : Prentice Hall Hispanoamerica, SA, 1992. pág. 924. ISBN 968-880-338-3.

Rational Software Corporation. 2003. RUP. Rational Unified Process. 2003. Version 2003.06.00.65.

Real Academia Española. 2009. Diccionario de la lengua española. [En línea] 2009. http://www.rae.es.

Reynoso, Carlos Billy. 2004. *Sitio Web Willydev.* [En línea] Marzo de 2004. http://www.willydev.net/descargas/prev/IntroArq.pdf.

Rivera, Cesar Augusto y Sarmiento Villamizar, Freddy. 2005. *Aplicaciones de un Modelo Conceptual generalizado de un sistema para soportar la toma de decisiones.* Universidad Industrial de Santander. Bucaramanga : s.n., 2005. Tesis pregrado.

Santamaria Pabon, Oscar Enrique y Mendez Forero, Andres Eduardo. 2004. *Herramienta software para el estudio de fenómenos ambientales mediante el modelado y la simulación con Dinámica de Sistemas.* Universidad Industrial de Santander. Bucaramanga (Colombia) : s.n., 2004. Tesis pregrado. S 13581.

SEI. 2009. Software Enginering Institute. *Definitions of Architecture.* [En línea] 2009. http://www.sei.cmu.edu/architecture/definitions.html.

SIMON, Grupo. Grupo SIMON. [En línea] http://simon.uis.edu.co.

Sotaquirá, Ricardo. 1994. *Incidencia de la corrosión sobre la economía nacional. Aplicación de la Dinámica de Sistemas.* Universidad Industrial de Santander. Bucaramanga : s.n., 1994. Tesis de pregrado.

Sterman, John D. 2000. *Business Dynamics: Systems Thinking and Modeling for a Complex World.* s.l. : Irwin McGraw-Hill, 2000. pág. 1008. ISBN: 978-0072389159.

Systems and software consortium. 2009. Software Productivity Consortium. *Systems and software consortium.* [En línea] 09 de 2009. http://www.software.org.

The CASM enviroment revisited. **Paul, Ray J. y Hlupic, Vlatka. 1994.** 1994, s.l. : Winter Simulation Conference, 1994, Winter Simulation Conference, págs. 641-648.

The construction of user interfaces and the object paradigm. **Coutaz, Joëlle. 1987.** Paris : Springer-Verlag, 1987. European conference on object-oriented programming on ECOOP '87. págs. 121-130. ISBN:0-387-18353-1.

Torres Cantillo, Dilia Ester y Solorzano Dangond, Gerardo. 2000. *Análisis y diseño de Evolución 32, herramienta software para la simulación con Dinámica de Sistemas.* Universidad Industrial de Santander. Bucaramanga : s.n., 2000. Tesis pregrado. Nº inv: S 09550.

Ulloa, Ana y Ulloa, Carmen. 1994. *Dinámica de Sistemas: aplicada al análisis, modelamiento y simulación al proceso de polimerización en cadena.* Universidad Industrial de Santander. Bucaramanga : s.n., 1994. Tesis de pregrado.

Ventana Systems. 2007. Vensim® Ventana® Simulation Environment. *Vensim documentation.* [En línea] 4 de Julio de 2007. http://www.vensim.com/ffiles/VensimUsersGuide.zip.

Vera, Cristián Eduardo y Anaya Ayala, Ricardo. 2006. *Ambiente Software para apoyar el Aprendizaje de las Ciencias de la Naturaleza en la Educación, Básica primaria. Un Enfoque Dinámico-Sistémico.* Universidad Industrial de Santander. Bucaramanga : s.n., 2006. Tesis pregrado.

Villa Abaunza, Alfredo Enrique y Zafra Gerena, Carlos Alfonso. 2000. *MACMedia, micromundo para el aprendizaje de ciencias en la educación media, basado en el modelo educativo centrado en los procesos de pensamiento.* Universidad Industrial de Santander. Bucaramanga : s.n., 2000. Tesis pregrado. Nº Inv: S 09414.

Zave, Pamela. 1997. *Classification of research efforts in requirements engineering.* 4. New York : ACM Computing Surveys (CSUR), 1997. págs. 315 - 321. Vol. 29.

ANEXOS

(Espacio intencional para conservar el formato)

Anexo A. Especificación de requisitos para el software ESMS-MI

(Espacio intencional para conservar el formato)

Versión: 2.0
Tipo: Especificación de requisitos

Especificación de requisitos para el software ESMS-MI

Elaborado por:
Grupo SIMON Universidad Industrial de Santander[43]

Resumen:

Palabras claves:

Fecha de creación: 1 de noviembre de 2008
Última modificación: 5 de noviembre de 2008

[43] Grupo SIMON Universidad Industrial de Santander Cll.27 Cr.9. Edf. Fisicomecanicas Of. 337. Bucaramanga.
Telefax: 57-(7)6343377. Teléfono: 57-(7)6344000 ext. 2681

Fecha	Versión	Descripción	Autor
16/Jun/09	1.0	Versión inicial	Emiliano Lince Mercado
05/Oct/09	2.0	Adición de nuevos requerimientos	Emiliano Lince Mercado

Contenido

INTRODUCCIÓN

Este documento contiene el conjunto de requisitos funcionales y no funcionales del entorno software de modelado y simulación de modelos integrados (ESMS-MI).

El ESMS-MI es un entorno software de modelado y simulación centrado en Dinámica de Sistemas (D.S), que permite la integración de modelos realizados con otras herramientas matemáticas.

El desarrollo de esta herramienta está a cargo por una comunidad conformada por diferentes grupos I+D, de varias universidades (geográficamente dispersa) y es una iniciativa del grupo SIMON de la Universidad Industrial de Santander.

Como antecedentes al ESMS-MI podemos encontrar al software Evolución desarrollado por el grupo SIMON, el cual es una herramienta para el modelado y la simulación con dinámica de sistemas. En el mercado existen herramientas similares a Evolución, entre las que se destacan Powersim, Vensim, Stella/ IThink y PROFESSIONAL DYNAMO.

Este documento está acorde a las orientaciones de la IEEE para la definición de requisitos software (1).

PROPÓSITO

Esta especificación de requisitos reúne los requisitos funcionales y no funcionales del sistema. Este documento es una fuente de consulta permanente durante todo el proceso de desarrollo del entorno y es un insumo para el desarrollo del diseño y la realización de pruebas.

ALCANCE

Desarrollar una herramienta que permita modelar y simular modelos realizados con diferentes herramientas matemáticas, integrados a través de la dinámica de sistemas. El software debe tener un conjunto de herramientas que apoyen al modelador en todas las fases del modelado y de la simulación, así como brindar herramientas de soporte para realizar análisis de modelos y análisis de comportamiento.

DEFINICIONES, SIGLAS, Y ABREVIATURAS

- ***Entorno software de modelado y simulación: Herramienta software que combina*** varios aspectos del proceso relacionados con el desarrollo y simulación de modelos, en una poderosa, completa y armonizada colección de herramientas integrada.

- ***Modelo:*** herramienta conceptual para observar un sistema complejo desde un punto de vista particular. Abstracción de una realidad más compleja.

- ***Modelo integrado:*** es un macro modelo que integra modelos realizados con diferentes herramientas matemáticas (D.S, lógica difusa, Redes

Neuronales, Agentes, Análisis Estadístico, Investigación Operacional, Representación Espacio Estado, entre otras).

- ***D.S:*** Dinámica de sistemas.
- ***ESMS-MI:*** Entorno software de modelado y simulación de modelos integrados.
- ***Grupo I+D:*** Grupo de Investigación y desarrollo.
- ***MyS:*** Modelado y simulación.
- ***Grupo SIMON: Grupo de investigaciones en modelamiento y simulación, adscrito a la escuela de ingeniería de sistemas e informática de la Universidad Industrial de Santander*** (2)***.***
- ***Evolución:*** herramienta software para el modelado y simulación con dinámica de sistemas, desarrollada en el grupo de investigación SIMON de la Universidad Industrial de Santander, actualmente se encuentra en su versión 4.0 y puede descargarse en el sitio web del grupo SIMON (2).
- ***UIS:*** Universidad Industrial de Santander.

DESCRIPCIÓN GENERAL

Esta especificación de requisitos está conformada por una descripción general acerca del producto software, la cual presenta detalles del mismo como son: la perspectiva del producto, sus funciones y los interesados en el desarrollo del proyecto incluyendo las características de los usuarios. Luego continúa con la especificación formal de los requisitos o requerimientos del software tanto funcionales, clasificados según las funciones principales del software. También se especifican los requisitos o requerimientos no funcionales (o de calidad) y se presentan en orden de prioridad (el primero es la característica de mayor importancia). Por último, el documento finaliza con las referencias e índice.

DESCRIPCIÓN GLOBAL

PERSPECTIVA DEL PRODUCTO

El ESMS-MI es una herramienta para el modelado y simulación de modelos integrados, herramienta que será desarrollada por una comunidad de investigación y desarrollo dispersa geográficamente. Esta herramienta pretende ser utilizada por la comunidad latinoamericana de dinámica de sistemas, además por diversas universidades y grupos de investigación en los cuales se estudian y discuten temáticas relacionadas al pensamiento sistémico, dinámica de sistemas, modelado y simulación de fenómenos complejos y comunidades e instituciones educativas que utilizan el MyS con D.S como herramienta para crear y recrear fenómenos a estudiar.

FUNCIONES DEL PRODUCTO
Esta una herramienta software para apoyar el proceso de modelado y simulación de modelos realizados con diferentes herramientas matemáticas, integrados por medio de la dinámica de sistemas.

INTERESADOS EN EL PROYECTO (STAKEHOLDERS)
Los interesados en el proyecto son las personas que de directa o indirectamente tienen relación con el proyecto para el desarrollo del software. Entre los interesados (Stakeholders en inglés) podemos clasificarlos en: el cliente, los usuarios, el equipo desarrollador[44] y el equipo de mantenimiento.

Cliente
Persona(s) u organización(es) que encarga la realización del software. Es la persona u organización que adquiere el software, en ocasiones el cliente no corresponde al usuario del sistema. El cliente es equivalente al "Acquirer" definido en el estándar de la IEEE 1471-2000 (3).

En el desarrollo del ESMS-MI el cliente corresponde a los diferentes grupos I+D encabezados por el grupo SIMON de la UIS.

Usuarios
Son las personas u organizaciones que harán uso del software. También son denominados usuarios finales.

Características del usuario
Los usuarios del ESMS-MI se clasifican en avanzados, Intermedios, principiantes y potenciales. Las características de estos grupos de usuarios se describen a continuación:

- Usuarios avanzados: se consideran usuarios avanzados a modeladores que han sido usuarios frecuentes de la herramienta Evolución en sus versiones 3.5 o superior. Entre estos usuarios se encuentran modeladores del grupo de investigación SIMON encabezados por su director Hugo Hernando Andrade Sosa. Igualmente, existen usuarios avanzados dentro de la comunidad latinoamericana de dinámica de sistemas.

- Usuarios intermedios: son usuarios que no son considerados modeladores expertos pero han utilizado la sintaxis propia de evolución. Estos usuarios están en la capacidad de leer y modificar un modelo realizado en la herramienta Evolución, además son usuarios frecuentes u ocasionales de la misma. En este grupo se encuentran estudiantes universitarios de cursos de modelado y simulación con DS, que utilizan la herramienta Evolución.

[44] Comúnmente se utiliza el término desarrollador indistintamente con el de programador. Aquí el término se refiere a los integrantes del equipo que desarrollará el software, no solo los programadores.

Igualmente, algunos profesores de instituciones educativas de básica y media que ejecutan proyectos de inclusión del MyS en la educación[45].

- Usuarios principiantes: son usuarios que conocen poco o nada la sintaxis utilizada por la herramienta Evolución en alguna de sus versiones, igualmente el uso de la herramienta es ocasional o nulo. Normalmente son lectores de modelos. En este grupo se encuentran profesores de instituciones educativas de básica y media que ejecutan proyectos de inclusión del MyS en la educación[46].

- Usuarios potenciales: son usuarios de herramientas similares a Evolución, es decir modeladores o lectores de modelos en dinámica de sistemas, que hacen uso de herramientas como Stella, Powersim, Vensim, entre otras. En este grupo se encuentran algunos de la comunidad latinoamericana de dinámica de sistemas. También son usuarios potenciales usuarios de otras herramientas en idiomas diferentes al español.

Desarrolladores

Son todas las personas u organizaciones que realizan o desarrollan el software. Los roles que estas desempeñan son: arquitectos, analistas, diseñadores, programadores, director del proyecto, evaluadores y soporte de infraestructura, entre otros. Eso depende de los requerimientos y tamaño del sistema a desarrollar. El desarrollo del ESMS-MI estará a cargo de una comunidad conformada por grupos de investigación y desarrollo distribuidos geográficamente, los cuales tendrán a su cargo diferentes módulos del ESMI-MI. Los grupos I+D su vez desarrollarán los diferentes módulos por medio de proyectos de pregrado. Inicialmente, han manifestado el interés de ser parte de esta comunidad los siguientes grupos:

Nombre del grupo	Universidad	URL
SIMON	Universidad Industrial de Santander	http://simon.uis.edu.co
GTI	Universidad del Cauca	http://gti.unicauca.edu.co

Además de los desarrolladores de los componentes del software, también se encuentran en esta clasificación los desarrolladores de aplicaciones que hacen uso de los componentes, es decir otros software (educativo, para toma de

[45] Profesores que han sido beneficiados por el programa Computadores Para Educar, en convenio con la Universidad Industrial de Santander y han sido acompañados por el Grupo SIMON para el diseño y ejecución de proyectos de inclusión de TIC en sus instituciones educativas y hacen parte Reddinamica.

[46] Profesores que han sido beneficiados por el programa Computadores Para Educar, en convenio con la Universidad Industrial de Santander y han sido acompañados por el Grupo SIMON para el diseño y ejecución de proyectos de inclusión de TIC en sus instituciones educativas.

decisiones, de MyS de propósito específico) que utilizarán los componentes del ESMS-MI al igual que el framework de Evolución(4).

Encargados del mantenimiento y soporte
Corresponde a todo persona u organización encargada de dar soporte a los usuarios y mantenimiento al software luego de ser terminado y entregado. En el caso del ESMS-MI se pretende que la comunidad de grupos I+D conformada para su desarrollo se mantenga, pudiendo así estar a cargo de las actividades de mantenimiento y soporte a los usuarios del software. Esta comunidad será auto gestionada y permitiendo la vinculación de nuevos grupos I+D.

REQUISITOS ESPECÍFICOS

REQUISITOS DE LAS INTERFACES EXTERNAS

Interfaz del usuario
Se debe contar con una interfaz gráfica que permita la realización de modelos de forma visual (gráfica). Debe permitir la creación de modelos en diferentes lenguajes (diferentes diagramas) como son el diagramas de influencias, flujo nivel, sectores, entre otros; también debe permitir agregar crear y editar el modelo por medio de un editor de formulas y por medio de comando de consola.

A continuación se enumeran cada uno de los requisitos que corresponden a la interfaz de usuario:

- Uniformidad en cada uno de las ventanas así como en los mensajes.
- Las barras de herramientas deben ser personalizables, al igual que los botones que ellas contienen.
- Debe permitir el acceso a todas las herramientas a través del teclado
- Permitir ingreso de elementos por líneas de comando (parecido a Autocad)
- El dispositivo de entrada principal es el ratón
- Todas las funciones deben poderse acceder utilizando touch para dispositivos de pantalla táctil como las Tablet PC
- Se debe permitir imprimir las diferentes vistas del modelo así como los resultados de la simulación, reportes, ecuaciones, etc.

Interfaz del hardware
El entorno debe permitir comunicación entre dos instancias de la aplicación a través de los puertos TCP/IP del equipo como en los software de juegos. Se debe poder configurar por medio de que puerto se va a realizar la conexión.

Debe tener una interfaz para comunicación con el puerto de la impresora (preferiblemente con otros puertos como el USB, o serial ya que el LTP no se

encuentran en la mayoría de portátiles) para que el modelo tome/entregue valores a dispositivos hardware conectados al PC.

Interfaz del software

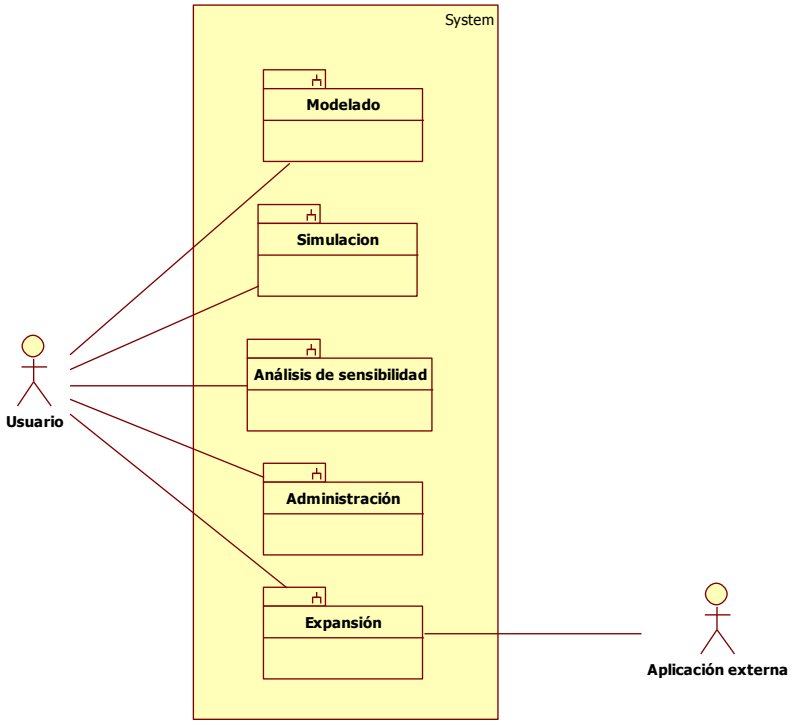
- Permitir agregar .dll de diferentes
- Permitir parámetros por línea de comandos por ej: ESMS c:\miArchivo.ext /Play
- API para interoperabilidad con otros software

Interfaz de comunicaciones

- Permitir comunicación por puertos TCP/IP

CARACTERÍSTICAS DEL SISTEMA

Las características del sistema son Modelado, Simulación, análisis de sensibilidad, Administración y expansión las cuales pueden verse en la siguiente figura:



Modelado

Introducción/Propósito de la característica

El ESMS-MI, como su nombre lo indica, es un entorno de modelado de modelos integrados, por lo tanto debe brindarle al usuario de todas las herramientas necesarias para abordar esta tarea. El modelado en el ESMS-MI es centrado en dinámica de sistemas, por lo tanto, debe permitir realizar modelos en todos los tipos de diagrama que componen a la D.S. los cuales son:

- **Diagrama en prosa:** Expresar el modelo en lenguaje natural (texto). Por ejemplo:

  "Al pensar sobre cómo se ha comportado una población de conejos, pensamos en cuánta población ha existido en cada momento y si ha venido aumentando, disminuyendo o ha permanecido estable. Además, si nos preguntamos qué la ha hecho cambiar, mínimo pensaremos en los nacimientos y en las muertes naturales. La anterior reflexión sobre el fenómeno identifica tres elementos fundamentales: Población (número de conejos en cualquier instante, por ejemplo día), Nacimientos (conejos que nacen por unidad de tiempo) y muertes (conejos que mueren por unidad de tiempo)."

- **Diagrama de influencias:** en este diagrama se presentan los elementos que hacen parte de la estructura del modelo y las relaciones que existe entre ellos. Su representación grafica son flechas unidireccionales y existen dos clases de información y de material, las cuales se diferencian por su color. Las relaciones pueden ser positivas, negativas y positivas/negativas lo cual se indica explícitamente con un signo al final de la flecha. En caso de que las relaciones formen ciclos, se les identifica con un símbolo, el cual puede ser positivo, negativo o positivo/negativo. Un ejemplo de este diagrama es el que muestra en la **Figura 61**.

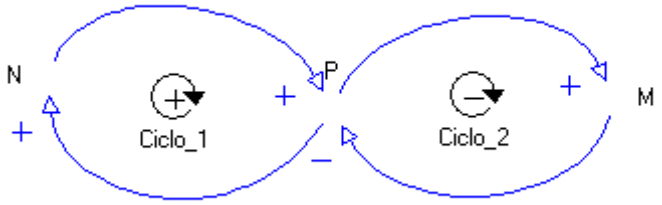
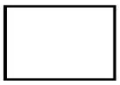
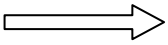


Figura 61: Diagrama de influencia.

Diagrama Flujo-nivel: También conocido como diagrama de Forrester. En este diagrama se específica el tipo de elemento y las relaciones según sus características y comportamiento en el modelo. Los elementos que conforman este diagrama se presentan en la **Tabla 9**. Cada herramienta de MyS con D.S. (ver

INTRODUCCIÓN) tienen representaciones diferentes de los elementos de este diagrama. En este diagrama se definen también los escenarios y las ecuaciones de algunas de las relaciones (excepto las de los niveles que son ecuaciones diferenciales que el software puede determinar a partir del diagrama flujo-nivel).

Tabla 9: Elemento del diagrama Flujo-Nivel.

	Nube: Representa una fuente o pozo. Se interpreta como un nivel inagotable y que no tiene interés.
	Nivel: Es la variable de estado; representa una acumulación de flujos.
	Flujo: Es la variación de un nivel; representa un cambio en el estado del sistema
	Canal de Material: Es la transmisión de una magnitud física que se conserva.
	Canal de Información: Es la transmisión de información que no necesita conservarse.
	Variable Auxiliar: Cantidad con cierto significado para el modelador (que no siempre tiene un significado físico en la vida real) y con un tiempo de respuesta inmediato.
	Parámetro: Es un elemento del modelo independiente del sistema o una constante de éste que no varía durante una corrida de simulación.
	Variable Exógena: Variable cuya evolución es independiente de las del resto del sistema. Representa una acción del medio sobre el sistema.
	Retardo: Es un elemento que simula retrasos en la transmisión de información o de material entre los elementos del sistema.
	No Linealidad: Representa una relación de no linealidad entre dos variables.
	FIS (componente de lógica difusa) permite agregar variables cualitativas por medio de un submodelo lógico-difuso.

	Red neuronal
	Elemento no disponible (Elemento con un tipo desconocido por el sistema)
	Elemento sin tipo (elemento al cual no se la ha definido un tipo específico)

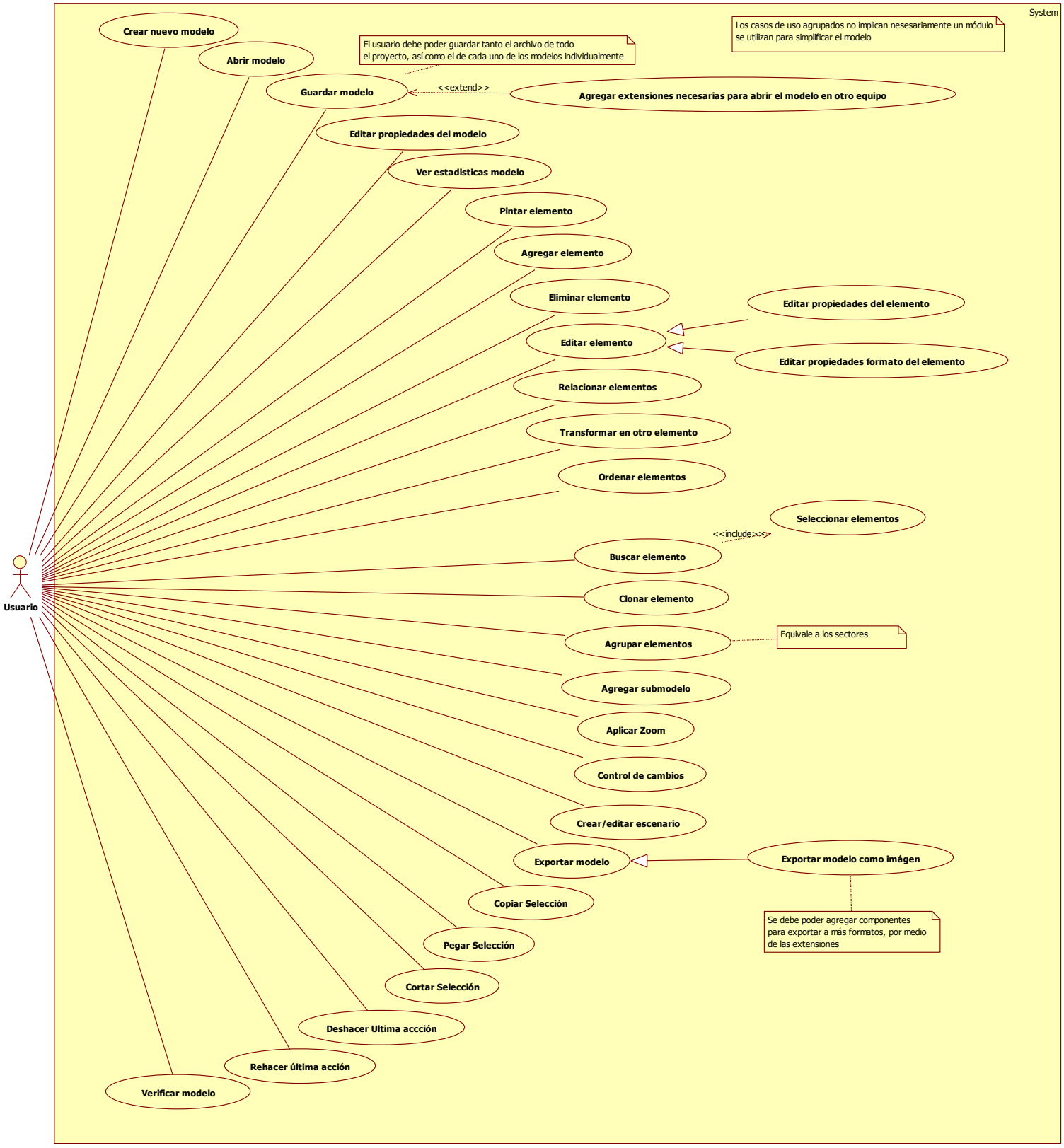
- **Diagrama de ecuaciones:** en este diagrama se presentan las ecuaciones que el software genera a partir del diagrama de Flujo-nivel y que relacionan a los elementos del diagrama. El sistema de ecuaciones debe ser resuelto numéricamente para realizar la simulación.

Estos diagramas deben ser extensibles, permitiendo agregar nuevos elementos a cada uno de los diagramas. Es así como por ejemplo es posible agregar un componente FIS (Sistema de inferencia difusa), pero a diferencia de Evolución, este debe poderse agregar al editor como lo hacen los plug-in en un navegador (por ejemplo un visualizador de flash) por lo que es posible abrir un modelo que tenga un elemento que no es posible de visualizar, y el modelo no se podrá simular hasta no contar con el plug-in correspondiente.

Los plug-in utilizados en el modelo pueden ser incluidos en el mismo archivo del modelo (si el usuario así lo desea). En la medida que se desarrollen nuevos plug-in y se conviertan en estables, serán agregados a una librería de plug-in que se instalan con el software por defecto. Es por esto que los plug-in deben definir una versión mínima del entorno en la cual se encuentra disponible (si no está disponible para ninguna versión también se debe especificar).

Secuencia de estimulo/Respuesta

Requisitos funcionales asociados


System
Los casos de uso agrupados no implican nesesariamente un módulo
se utilizan para simplificar el modelo
Crear nuevo modelo
Abrir modelo
El usuario debe poder guardar tanto el archivo de todo
el proyecto, así como el de cada uno de los modelos individualmente
Guardar modelo
<<extend>>
Agregar extensiones necesarias para abrir el modelo en otro equipo
Editar propiedades del modelo
Ver estadisticas modelo
Pintar elemento
Agregar elemento
Eliminar elemento
Editar propiedades del elemento
Editar elemento
Editar propiedades formato del elemento
Relacionar elementos
Transformar en otro elemento
Ordenar elementos
Seleccionar elementos
<<include>>
Buscar elemento
Clonar elemento
Agrupar elementos
Equivale a los sectores
Agregar submodelo
Aplicar Zoom
Control de cambios
Crear/editar escenario
Exportar modelo
Exportar modelo como imágen
Copiar Selección
Se debe poder agregar componentes
para exportar a más formatos, por medio
de las extensiones
Pegar Selección
Cortar Selección
Deshacer Ultima accción
Rehacer última acción
Verificar modelo
Usuario

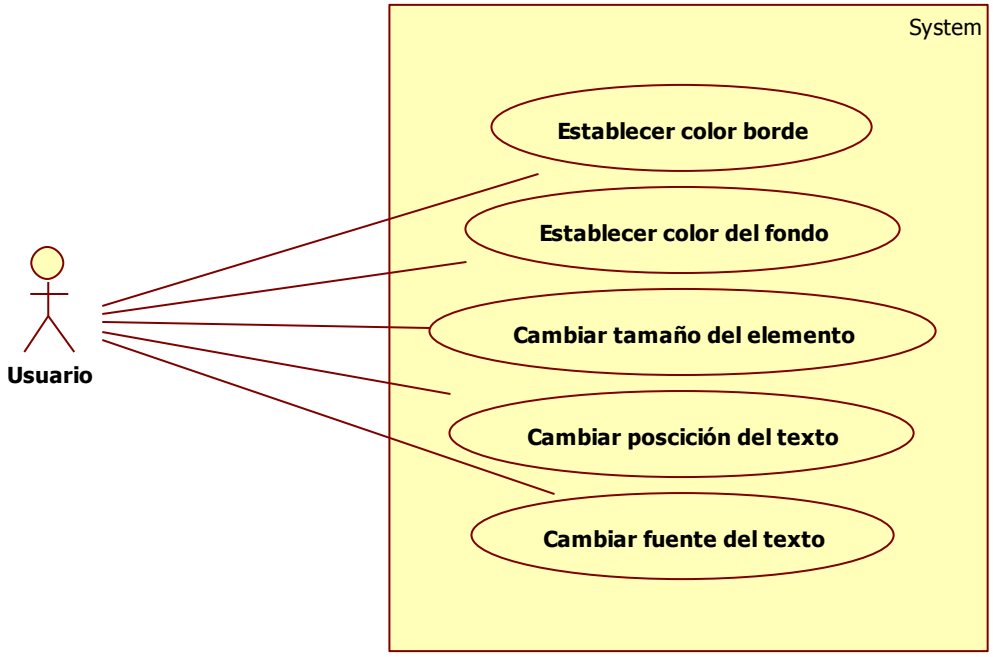


Figure 1 – Diagrama de caso de uso de Editar propiedades gráficas

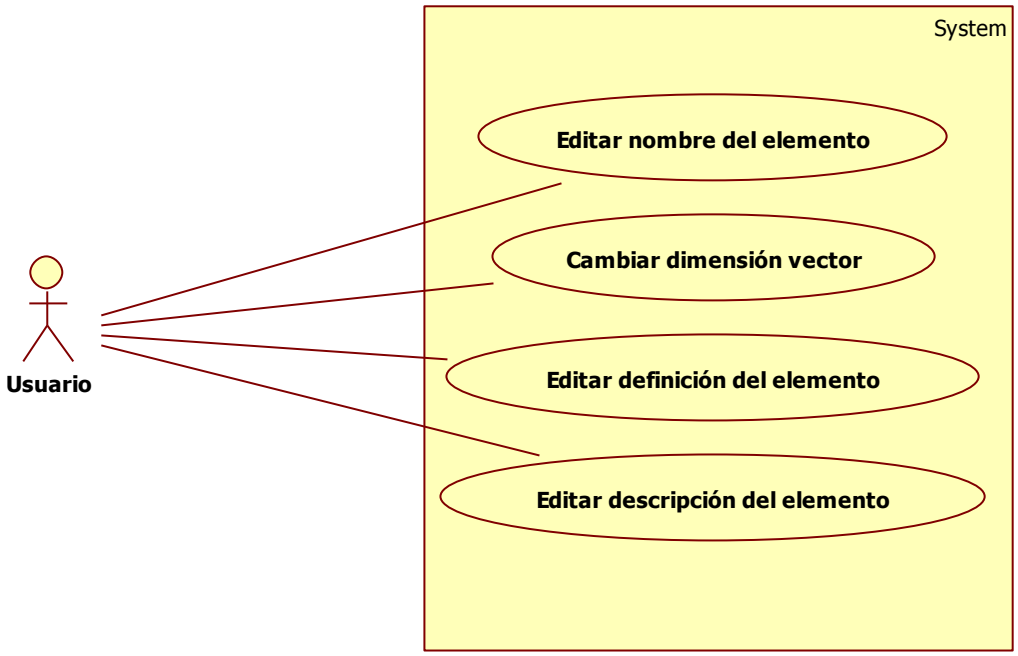


Figure 2 – Diagrama de caso de uso de Editar propiedades lógicas

Figure 3 – Diagrama de caso de uso de Ordenar elementos

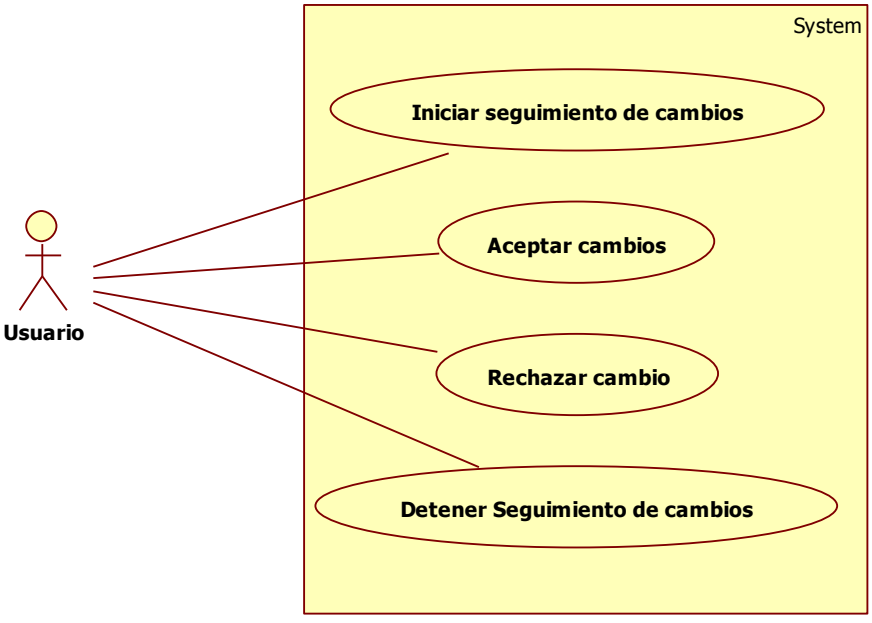


Figure 4 – Diagrama de caso de uso de Control de cambios

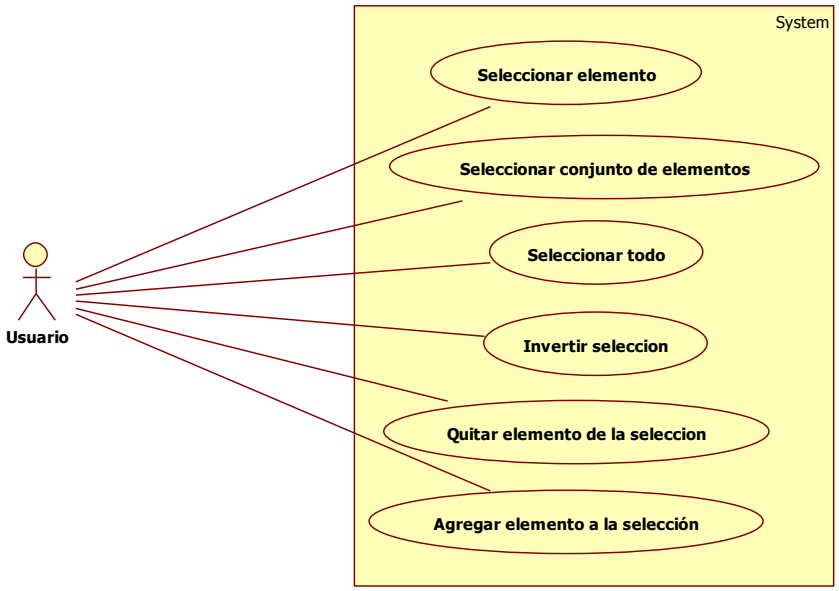


Figure 5 – Diagrama de caso de uso de Seleccionar elementos

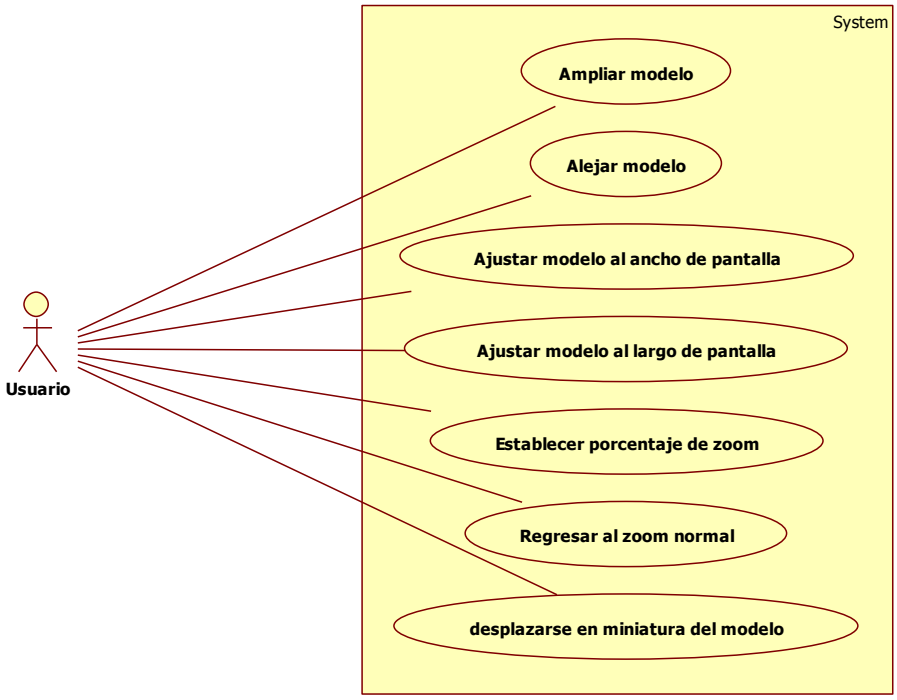


Figure 6 – Diagrama de caso de uso de Aplicar zoom

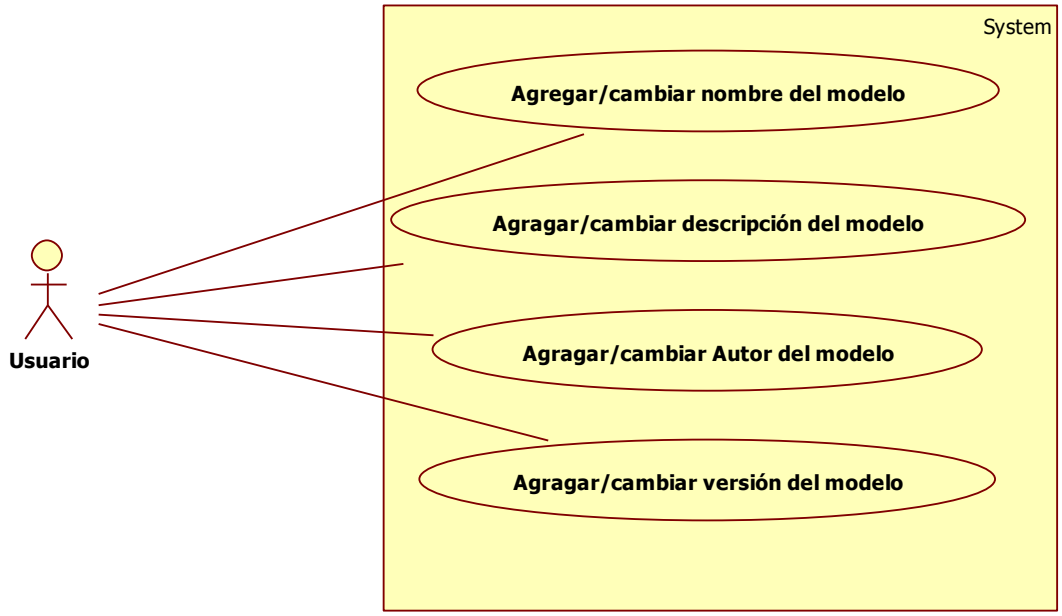


Figure 7 – Diagrama de caso de uso de Editar propiedades del modelo

Requisitos funcionales asociados a los diagramas Flujo-Nivel

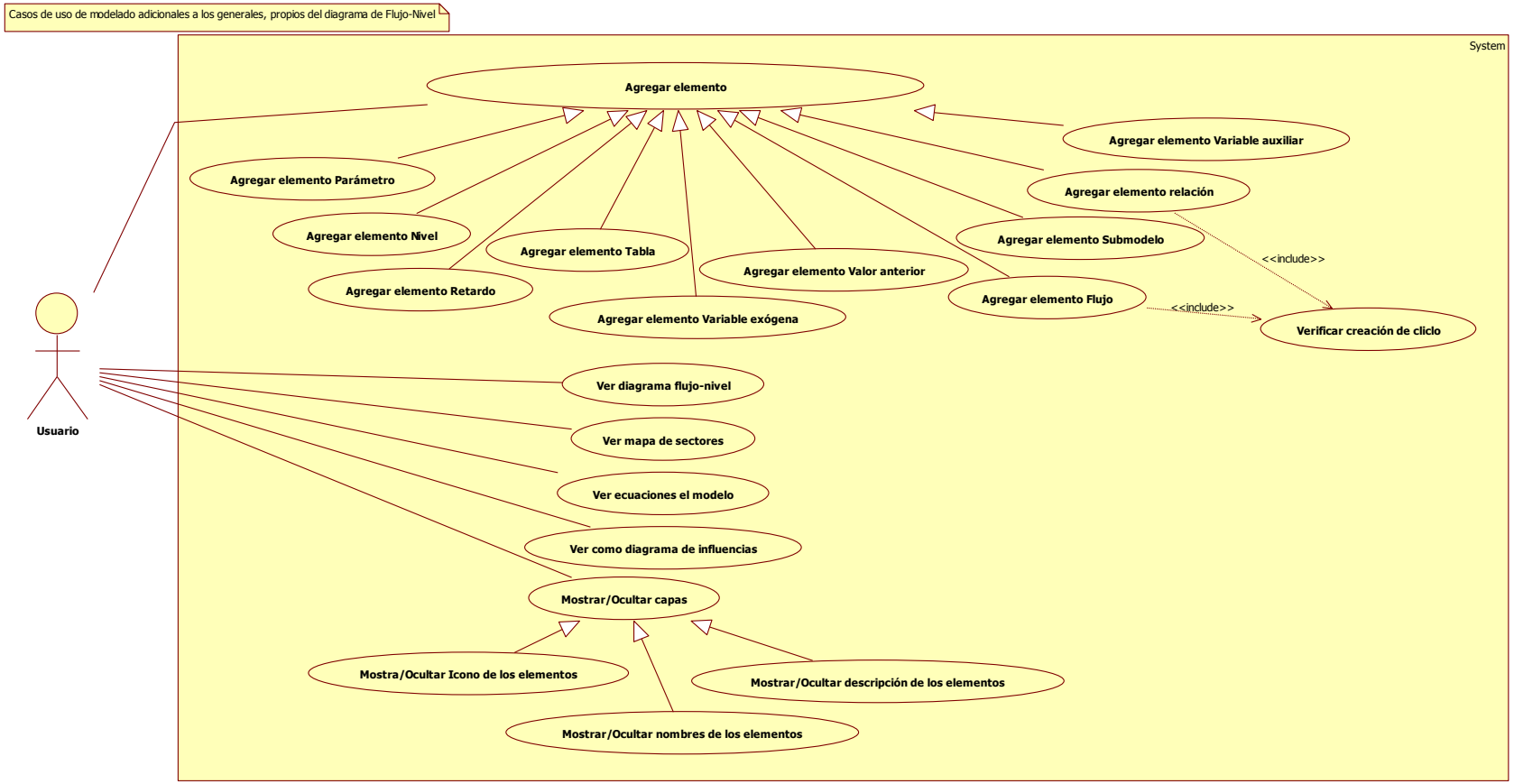
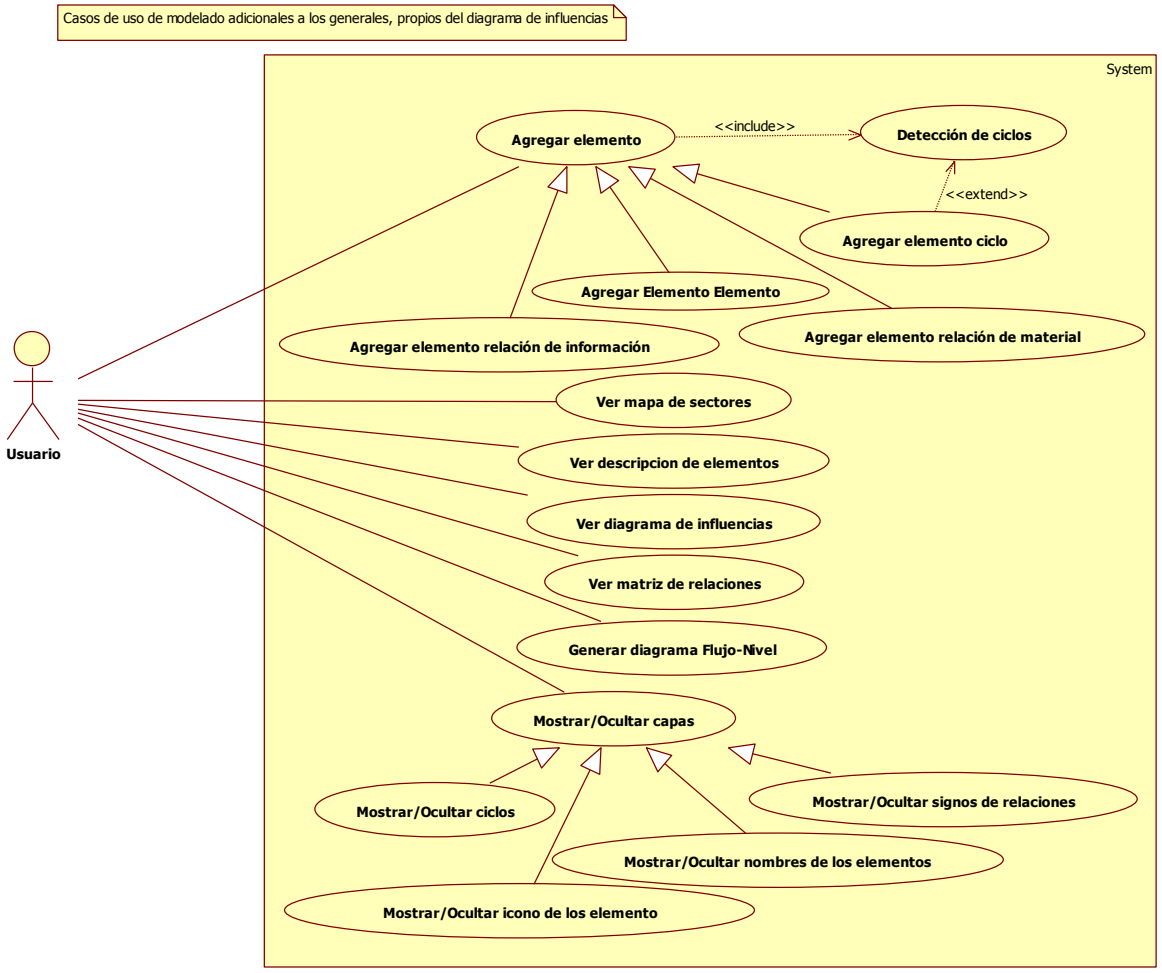


Figure 8 – Caso de uso de Principal

Requisitos funcionales asociados a los diagramas de influencias



Simulación

Introducción/Propósito de la característica
Una vez se cuenta con un modelo se puede dar inicio a la simulación del mismo, lo cual inicia con definir un escenario para la simulación. La simulación consta de dos pasos principalmente: resolver el modelo matemático haciendo uso de algún algoritmo, el siguiente paso es la presentación de los resultados ya sea de forma tabular o por medio de gráficos o animaciones.

Los procesos de simulación para el ESMS debe ser interactivo, de esta forma el usuario puede cambiar un valor de algún elemento durante la ejecución de la simulación, lo cual debe tener efecto inmediato en las siguientes iteraciones de la simulación.

Igualmente, para el ESMS se debe poder realizar formularios con diferentes formas de representar los resultados obtenidos como son gráficas, tablas, botones de control, etc...

También es posible contar con más de un presentador de resultados funcionando como clientes conectados a otro que hace las veces de servidor.

Secuencia de estimulo/Respuesta

Requisitos funcionales asociados

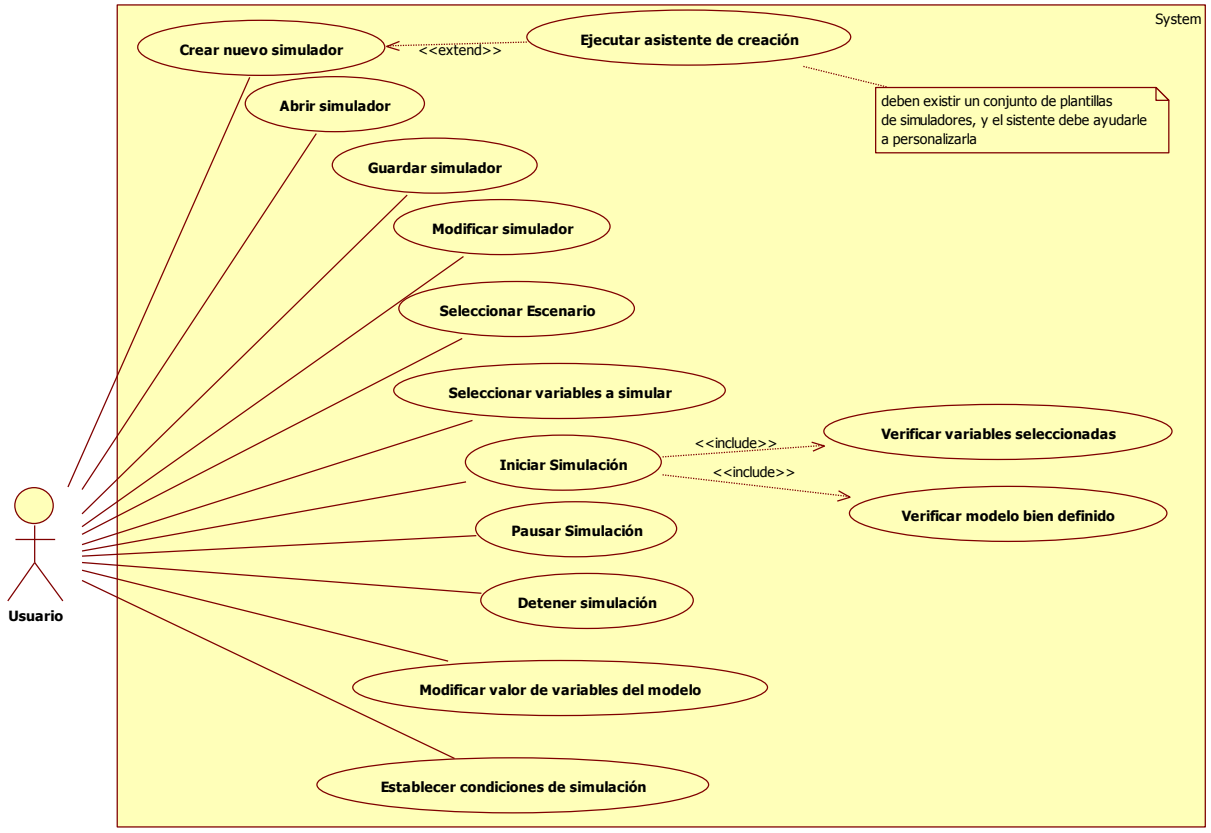


Figure 9 – Caso de uso de Principal

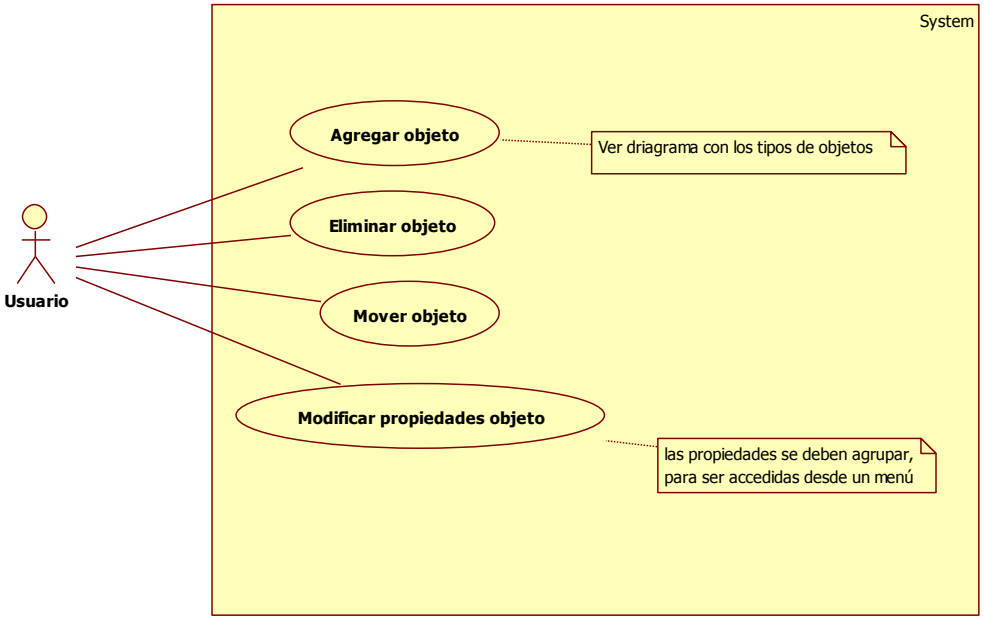


Figure 10 – Caso de uso de Modificar simulador

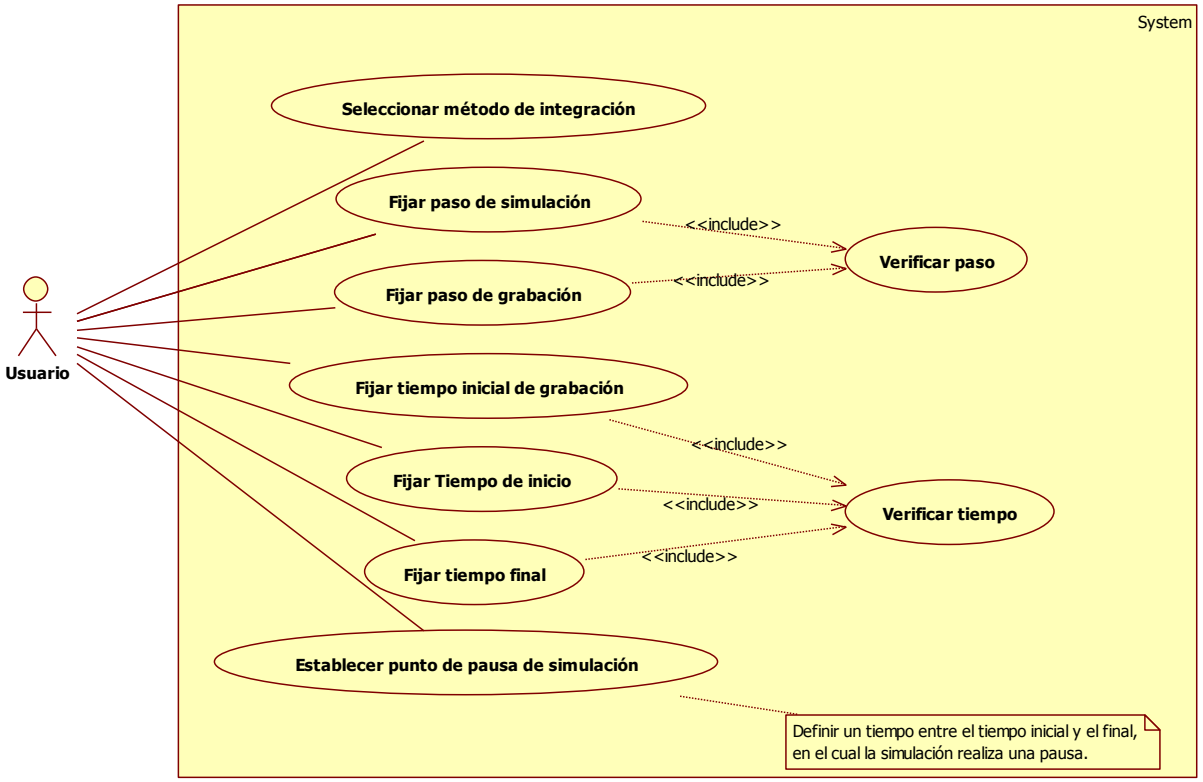


Figure 11 – Caso de uso de Establecer condiciones de simulación

Administración

Introducción/Propósito de la característica
El entorno debe ofrecer mecanismos para la administración de las diferentes herramientas que componen en entorno, así como ofrecer características que son generales a todo el proceso de modelado y simulación.

Entre estas características encontramos por ejemplo: el generar un reporte de todo el proceso de modelado y simulación o crear un proyecto con varios modelos y varios presentadores de resultados, asignar una descripción a todo el proyecto, Manejo de errores (generación y envío de reporte), entre otros.

Secuencia de estimulo/Respuesta

Requisitos funcionales asociados

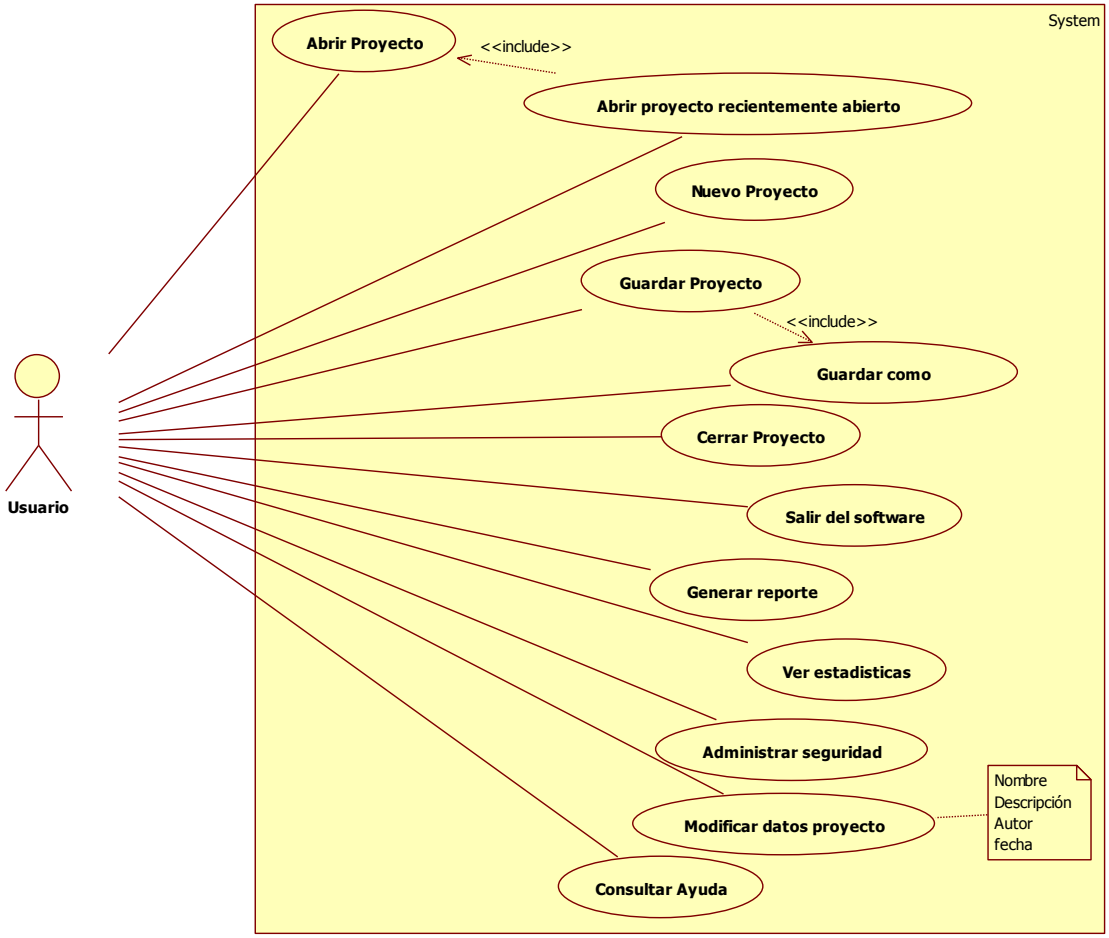


Figure 12 – Caso de uso de Principal

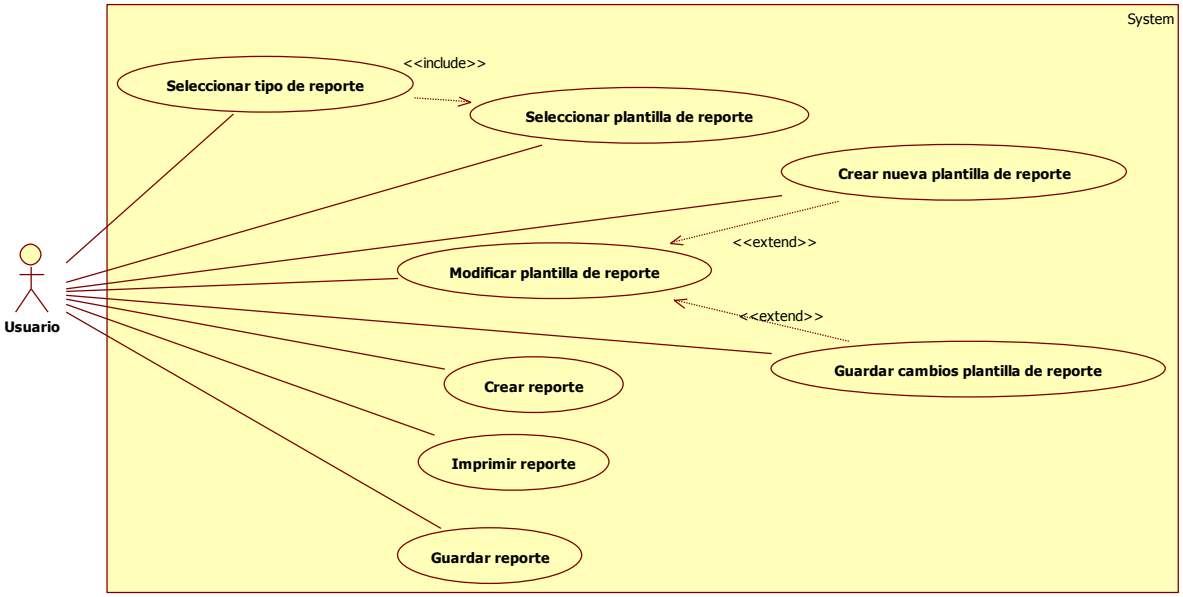


Figure 13 – Caso de uso de Generar reporte

Expansión

Introducción/Propósito de la característica
El entorno debe facilitar características que permitan agregar nuevos módulos y herramientas al entorno. Igualmente ofrecer funcionalidades para la comunicación con otras aplicaciones, brindando acceso a sus funcionalidades.

Secuencia de estimulo/Respuesta

Requisitos funcionales asociados

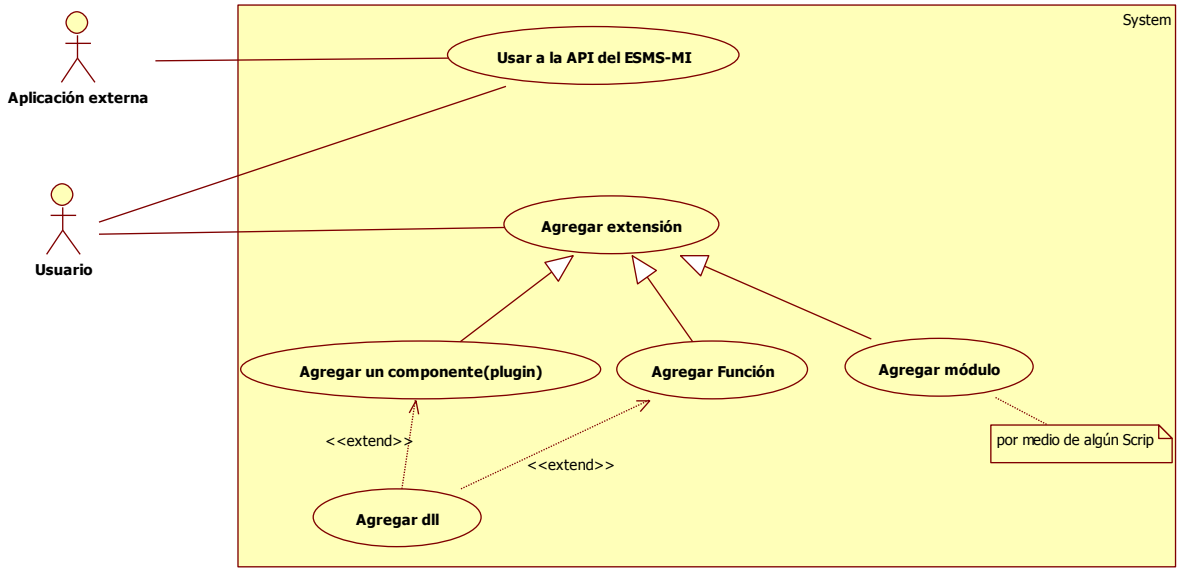


Figure 14 – Caso de uso de Principal

ATRIBUTOS DE SISTEMA DE SOFTWARE (REQUISITOS NO FUNCIONALES)
Existe una gran cantidad de taxonomías y definiciones para los atributos del sistema software (Bass, y otros, 2007), también denominados atributos de calidad o requisitos no funcionales. En este documento se hace uso de un conjunto de escenarios de atributos de calidad para ayudar a la obtención de los requisitos no funcionales, esta técnica es propuesta en (Bass, y otros, 2007).

Los escenarios de atributos de calidad están conformados por las siguientes partes (Ver **Figura 11**):

Fuente del estimulo: es alguna entidad (un humano, un sistema de computadora o cualquier otro actor) que genera el estimulo.

Estimulo: es el requisito que necesita ser considerado cuando este llega a un sistema.

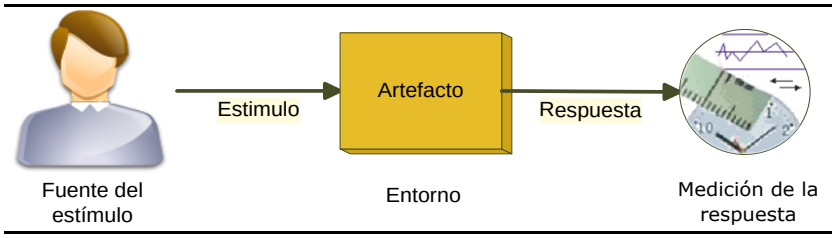
Entorno: el estimulo ocurre con ciertas condiciones. El sistema puede estar en una condición de sobrecarga o puede estar ejecutándose cuando el estimulo ocurre, o algún otro requisito o condición que pueda ser cierta. Esta parte contextualiza el requisito

Artefacto: el artefacto estimulado. Este puede ser todo el sistema o alguna parte de éste.

Respuesta: es la actividad realizada o emprendida después de la llegada del estímulo.

Medición de la respuesta: cuando la respuesta es generada, esta puede ser medida de alguna forma, de modo que los requisitos pueden ser probados.

Figura 62. Escenario de atributo de calidad

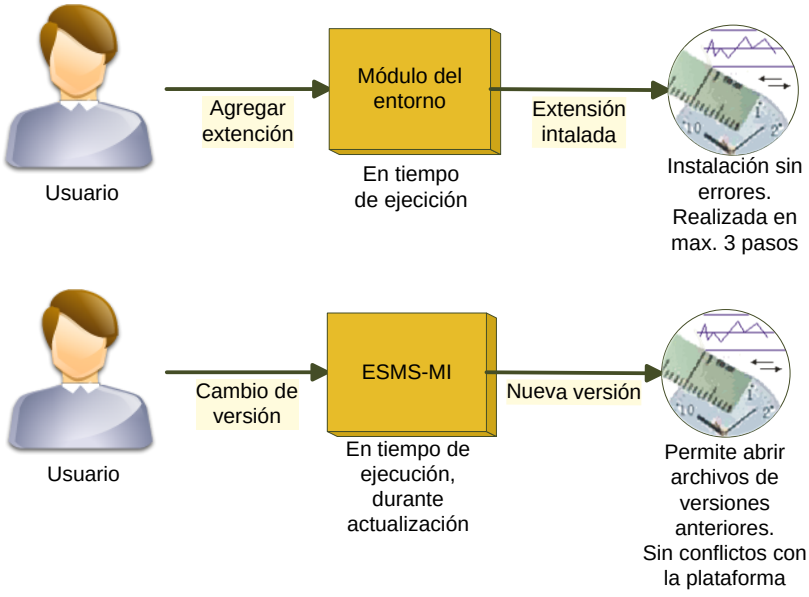


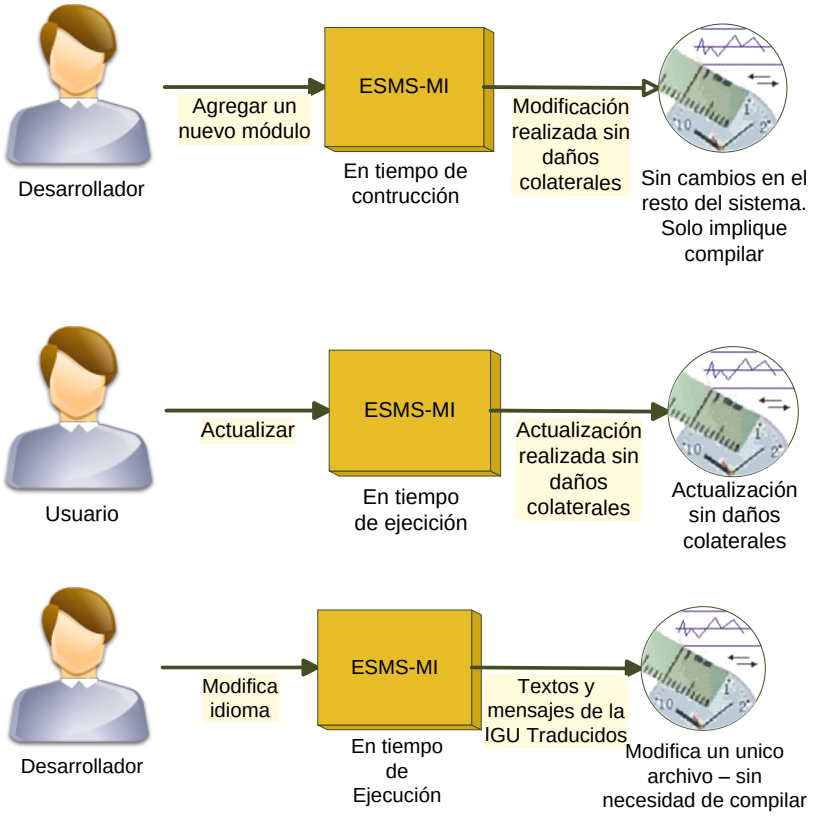
Dentro de los atributos de calidad más relevantes para el ESMS-MI se pueden enumerar los siguientes (en orden de importancia):

Modificable (Modifiability)
Facilidad para el cambio y para realizar mantenimiento. Esfuerzo necesario para localizar y arreglar un error en un programa o realizarle cambios.

Dentro de este atributo de calidad encontramos los siguientes escenarios:

Figura 63. Escenarios del atributo modificable

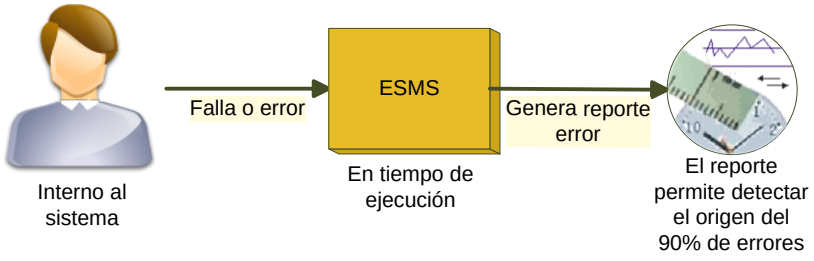


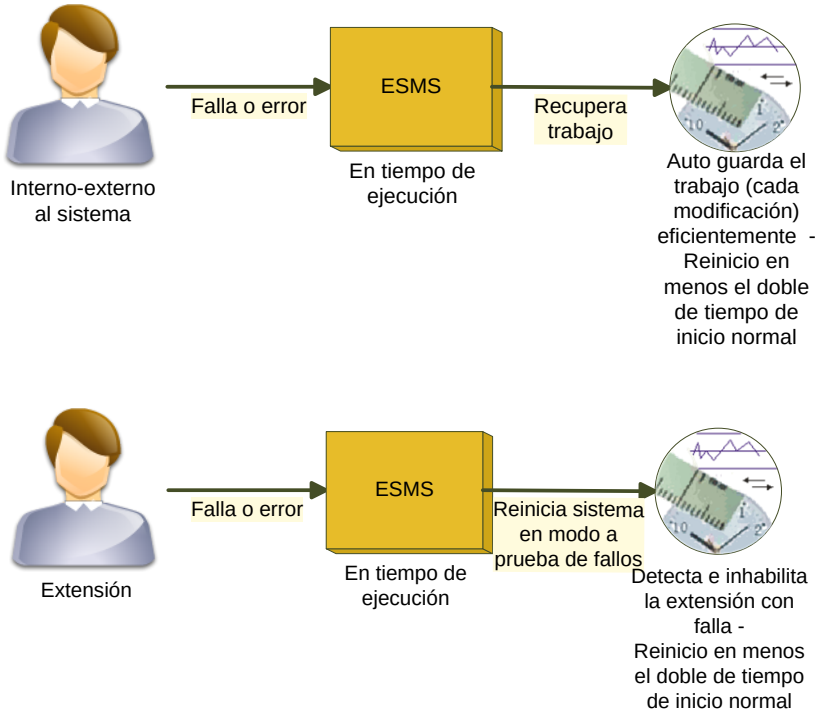


Fiabilidad (Reliability)

Se evalúa midiendo la frecuencia y la gravedad de los fallos, la exactitud de las salidas (resultados), el tiempo medio de fallos (TMDF), la capacidad de recuperación de un fallo.

Figura 64. Escenarios de Fiabilidad

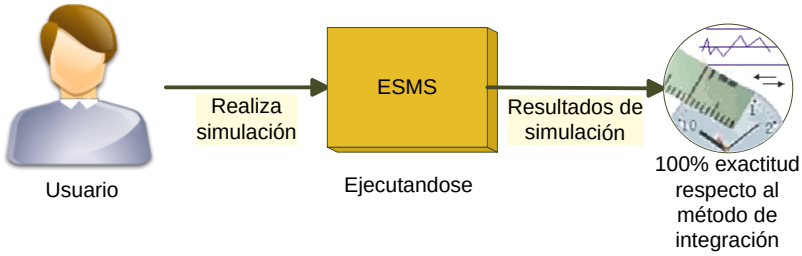




Desempeño (Performance)

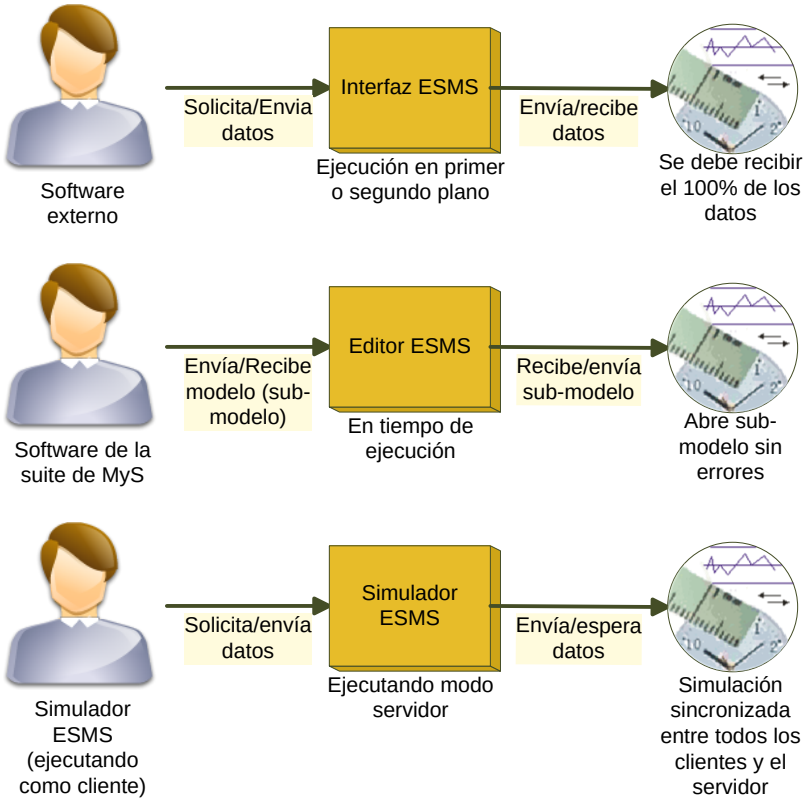
Es el grado en el cual un sistema o componente cumple con sus funciones designadas, dentro de ciertas restricciones dadas, como velocidad, exactitud o uso de memoria. El desempeño de un sistema se refiere a aspectos temporales del comportamiento del mismo. Se refiere a la capacidad de respuesta, ya sea el tiempo requerido para responder a aspectos específicos o el número de eventos procesados en un intervalo de tiempo. Además, se refiere a la cantidad de comunicación e interacción existente entre los componentes del sistema. El rendimiento se mide por la velocidad de procesamiento, el tiempo de respuesta, el consumo de recursos, el rendimiento efectivo total y la eficacia.

Figura 65. Escenarios de desempeño



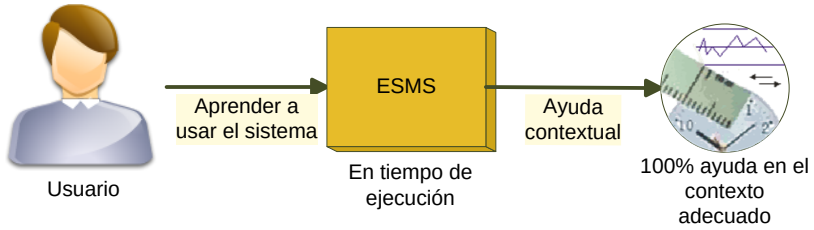
Interoperabilidad (Interoperability)
Capacidad de operar en conjunto con otros software

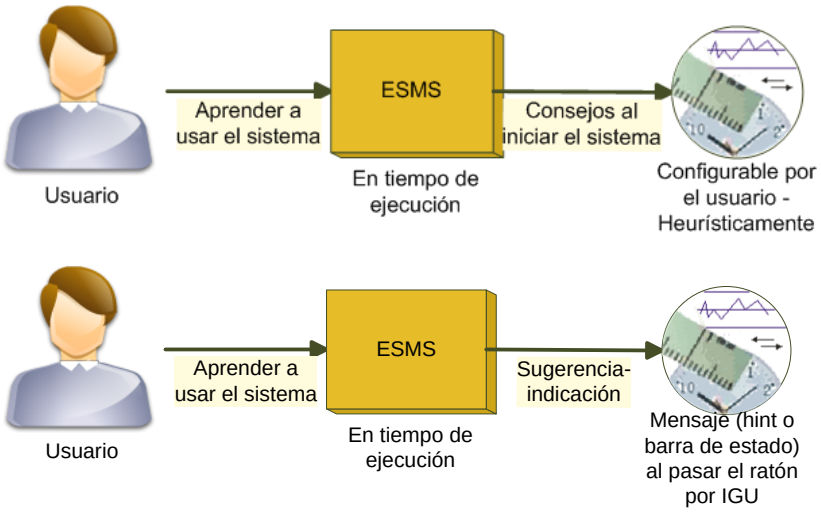
Figura 66. Escenarios de interoperabilidad



Usabilidad
Relacionada con qué tan fácil es que el usuario realiza una tarea deseada y el tipo de soporte a usuario que provee el sistema.

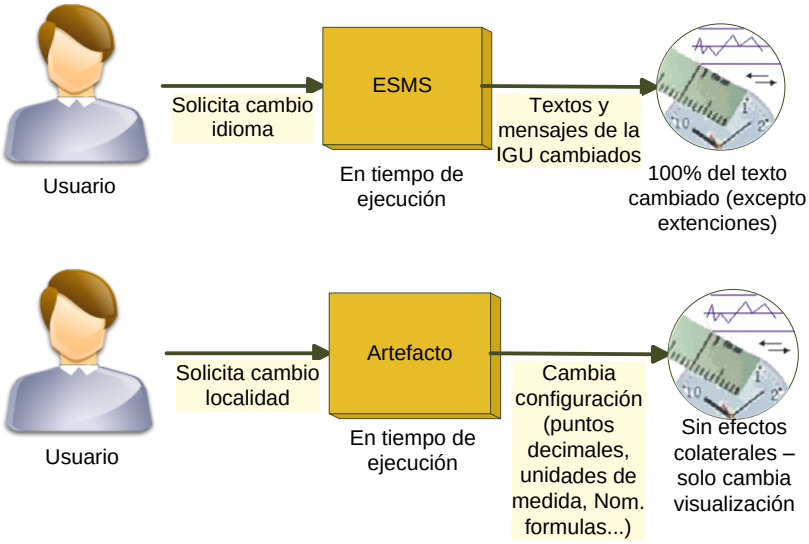
Figura 67. Escenarios de usabilidad





Internacionalización - Regionalización (localibity)
Disponibilidad para adaptarse a otras regiones (idioma, formatos de número y moneda).

Figura 68. Escenarios de internacionalización



REFERENCIAS

1. **IEEE Computer Society.** *Recommended Practice for Software Requirements Specifications.* Software Engineering Standards Committee, IEEE, inc. New York : IEEE, 1998. pág. 38, Estandar. IEEE 830-1998/ISBN 0-7381-0448-5, SS94654.

2. **Grupo SIMON.** Pagína principal: grupo SIMON de la UIS. *Sitio Web del Grupo SIMON de la UIS.* [En línea] 2009. http://simon.uis.edu.co.

3. **IEEE Computer Society.** *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems.* Software Engineering Standards Committee, IEEE. New York : IEEE, 2000. pág. 34, Estandar. ISO/IEC 42010:2007(E)/ IEEE Std 1471-2000; PDF/ ISBN 0-7381-2519-9 SS94869.

4. **Andrade Sosa, Hugo Hernando, Lince Mercado, Emiliano y Gómez Prada, Urbano Eliecer.** FRAMEWORK PARA EL DESARROLLO DE AMBIENTES SOFTWARE DE APRENDIZAJE Y TOMA DE DECISIONES CON MODELOS EN DINÁMICA DE SISTEMAS. *Memorias XIII Congreso de Informática en la Educación.* Habana, Cuba : s.n., 2009.

5. **Bass, Len, Clements, Paul y Kazman, Rick.** *Software architecture in practice.* Segunda edición. Boston : Addison-Wesley Professional, 2007. pág. 560. 978-0321154958.

6. **Nuseibeh, Bashar y Easterbrook, Steve.** *Requirements Engineering: A Roadmap.* Limerick, Ireland : International Conference on Software Engineering, 2000. págs. 35 - 46.

7. **Zave, Pamela.** *Classification of research efforts in requirements engineering.* 4. New York : ACM Computing Surveys (CSUR), 1997. págs. 315 - 321. Vol. 29.

8. **Cuellar Yeneris, Mario y Lince Mercado, Emiliano.** *Evolución 3.5 Herramienta Software para el modelamiento y simulación con Dinámica de sistemas.* Universidad Industrial de Santander. Bucaramanga : s.n., 2003. Tesis de pregrado.

9. **Moreno Chaustre, Jorge Jair.** *Diseño de una arquitectura para un entorno de modelamiento- simulación y creación de un proceso para su desarrollo por una comunidad (I+D).* Universidad Industrial de Santander. Bucaramanga : s.n., 2006. Tesis Maestría.

10. *Inquiry-based requirements analysis.* **Potts, Collin, Takahashi, Kenji y Anton, Annie I.** 2, Atlanta : IEEE, Marzo de 1994, IEEE Software, Vol. 11, págs. 21-32. ISSN: 0740-7459.

11. **Collins-Sussman, Ben, Fitzpatrick, Brian W. y Pilato, C. Michael.** Control de versiones con Subversion. [En línea] http://svnbook.red-bean.com/nightly/es/index.html.

ÍNDICE

Anexo B. Documento de descripción de la arquitectura

(Espacio intencional para conservar el formato)

Entorno Software de Modelado y Simulación de Modelos Integrados Comunidad ESMS-MI

Versión <2.0>

Historial de revisiones

Fecha	Versión	Descripción	Autor
09/Jul/2009	1.0	Versión inicial	Emiliano Lince Mercado
05/Oct/2009	2.0	Detalle de algunas relaciones	Emiliano Lince Mercado

Contenido

INTRODUCCIÓN

Este documento presenta la arquitectura del Entorno software de Modelado y simulación de Modelos Integrados (ESMS-MI) y la descripción de cada uno de sus componentes. El ESMS-MI es desarrollado y soportado por una comunidad conformada por grupos de investigación y desarrollo (I+D) de varias universidades.

META ARQUITECTURA

La meta arquitectura es un conjunto de decisiones de alto nivel que influirán fuertemente a la estructura del sistema, pero no es en sí misma la estructura del sistema (Malan, y otros, 2004).

La arquitectura del ESMS-MI corresponde a una arquitectura heterogénea (Garlan, y otros, 1994) que combina varios estilos arquitectónicos. En principio, la arquitectura del entorno sigue el patrón arquitectónico PAC (Presentación-Abstracción-Control). Una arquitectura PAC define la estructura de un sistema software como una jerarquía de agentes PAC que cooperan entre sí. Cada agente está conformado por tres componentes: Presentación, Abstracción y Control. Estos componentes permiten separar los aspectos relacionados con la interfaz humano-computador de los aspectos de la lógica funcional y la comunicación con otros agentes (Buschmann, y otros, 1996).

La arquitectura PAC es una arquitectura para sistemas interactivos. Esta interacción con el usuario muchas veces es realizada por medio de interfaces gráficas de usuario (IGU). Cuando se diseña sistemas interactivos es deseable separar la lógica de la aplicación o núcleo funcional de la interfaz con el usuario. Entre los patrones arquitectónicos para sistemas interactivos sobresalen dos: Modelo-Vista-Control (MVC) y Presentación-Abstracción-Control (PAC), los cuales son descritos en (Buschmann, y otros, 1996).

El ESMS-MI es un entorno software que proveer diversos mecanismos de interacción con el usuario con el propósito de ofrecer una gran cantidad de herramientas para el modelado visual y la simulación interactiva con mecanismos avanzados de presentación gráfica y animada de resultados.

Entre los patrones arquitectónicos para sistemas interactivos, se ha seleccionado el patrón PAC. Este patrón define el sistema en función de agentes que cooperan entre sí. En nuestro sistema se logra identificar claramente agentes para el modelado, simulación, generador de reportes, entre otros. Otro requisito de nuestro sistema es la reutilización y los agentes son candidatos ideales para facilitar el cumplimiento de este requisito, permitiendo a los desarrolladores de otras aplicaciones hacer uso de los agentes para el propósito específico de su aplicación

Es de aclarar, que en este contexto un agente es un componente para procesar la información que incluye el envío y recepción de eventos, estructuras de datos para mantener el estado, un procesador que maneja los eventos recibidos, actualización de su estado y que pueda producir nuevos eventos. Los agentes pueden ser tan pequeños como un objeto, pero también tan complejos como un sistema software completo (Buschmann, y otros, 1996).

Esta arquitectura PAC presenta los siguientes beneficios y debilidades[47] (ver **Tabla 5**).

Tabla 10. Beneficios y debilidades de la arquitectura PAC

Beneficios	***Debilidades***
• Separación de intereses o conceptos. • Soporta el cambio (mantenimiento) y la extensión. • Soporta la multitarea (cada agente en un hilo o los componentes abstracción y presentación en hilos independientes). • Desacopla Interfaz de usuario del núcleo funcional del sistema	• Incrementa la complejidad del sistema. • Componentes "control" complejos. • Eficiencia. • Aplicabilidad.

A continuación, se presenta algunas conclusiones de la comparación entre dos arquitecturas orientadas a sistemas interactivos, el patrón PAC con el patrón MVC:

Para sistemas grandes (como lo en este caso) la estructura introducida por el modelo PAC provee mayor facilidad de mantenimiento que el provisto por el modelo MVC (Hussey, y otros, 1996).

En la arquitectura MVC la noción de control está diluida a través de los tres objetos relacionados (modelo, vista y controlador), mientras que en la arquitectura PAC el control es centralizado explícitamente en el componente *Control*. (The construction of user interfaces and the object paradigm, 1987).

El componente "presentación" de PAC es en esencia toda la interfaz de usuario tal como el usuario la percibe. En el modelo MVC, la interfaz de usuario está

[47] Las debilidades pueden ser fortalecidas por medio de la combinación de patrones arquitectónicos o aplicación de patrones de diseño

compuesta tanto por el componente “vista” como el componente “control”(Bass, y otros, 1991).

El modelo MVC es más simple que el modelo PAC y resulta en implementaciones y especificaciones menos complejas. (Hussey, y otros, 1996).

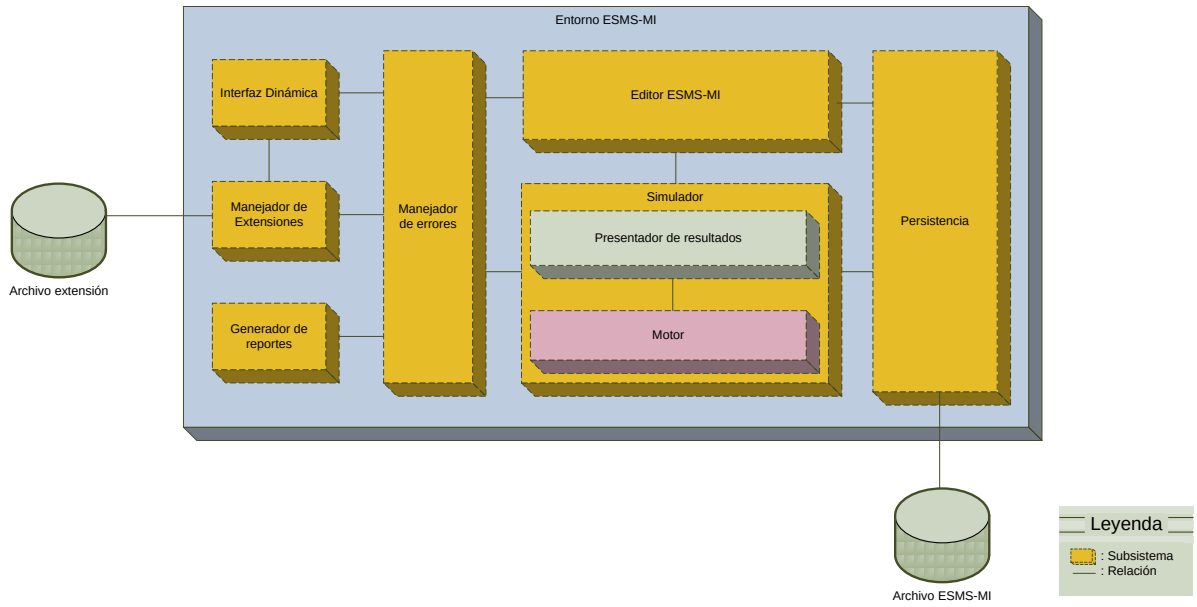
ARQUITECTURA CONCEPTUAL

En la arquitectura conceptual nos concentramos en una descomposición apropiada del sistema, antes de ahondar en los detalles de especificación de interfaces y tipos de información (Malan, y otros, 2004).

ARQUITECTURA CONCEPTUAL DE ALTO NIVEL

La arquitectura conceptual de alto nivel contiene los principales subsistema del entorno y sus relaciones. En la **Figura 52** se presenta el diagrama de la arquitectura del entorno y sus elementos son descritos en los numerales siguientes numerales.

Figura 69. Arquitectura conceptual de alto nivel del ESMS-MI

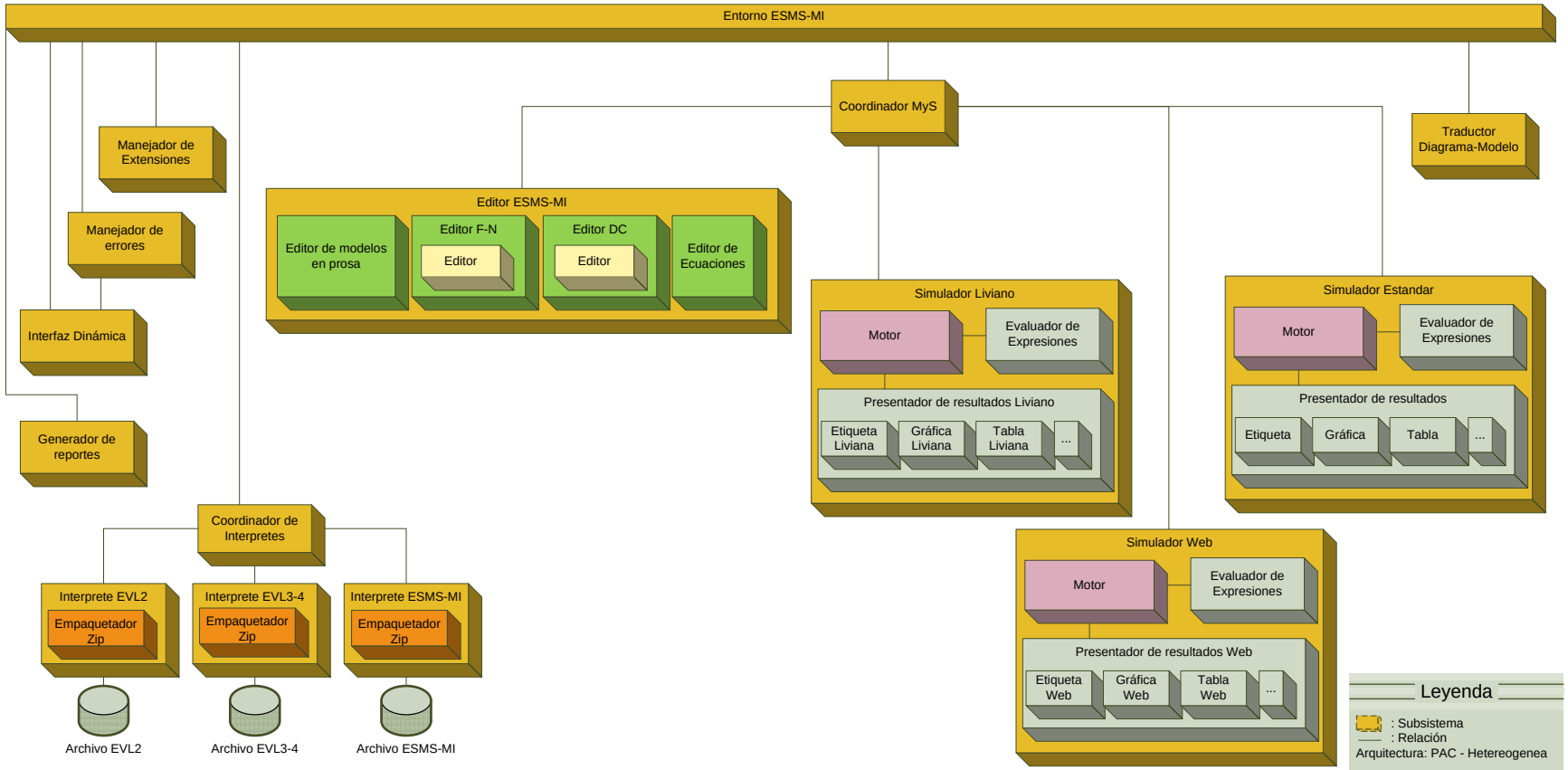


ESTILO ARQUITECTÓNICO

La arquitectura del entorno es una arquitectura basada en componentes, lo cual facilita su desarrollo por equipos independientes y separados geográficamente. El patrón arquitectónico es en esencia Presentación – Abstracción – Control (PAC), pero algunos de los agentes son subsistemas complejos con otros

patrones arquitectónicos, es decir, su estructura interna no está compuesta por componentes PAC (Ver **Figura 53**)

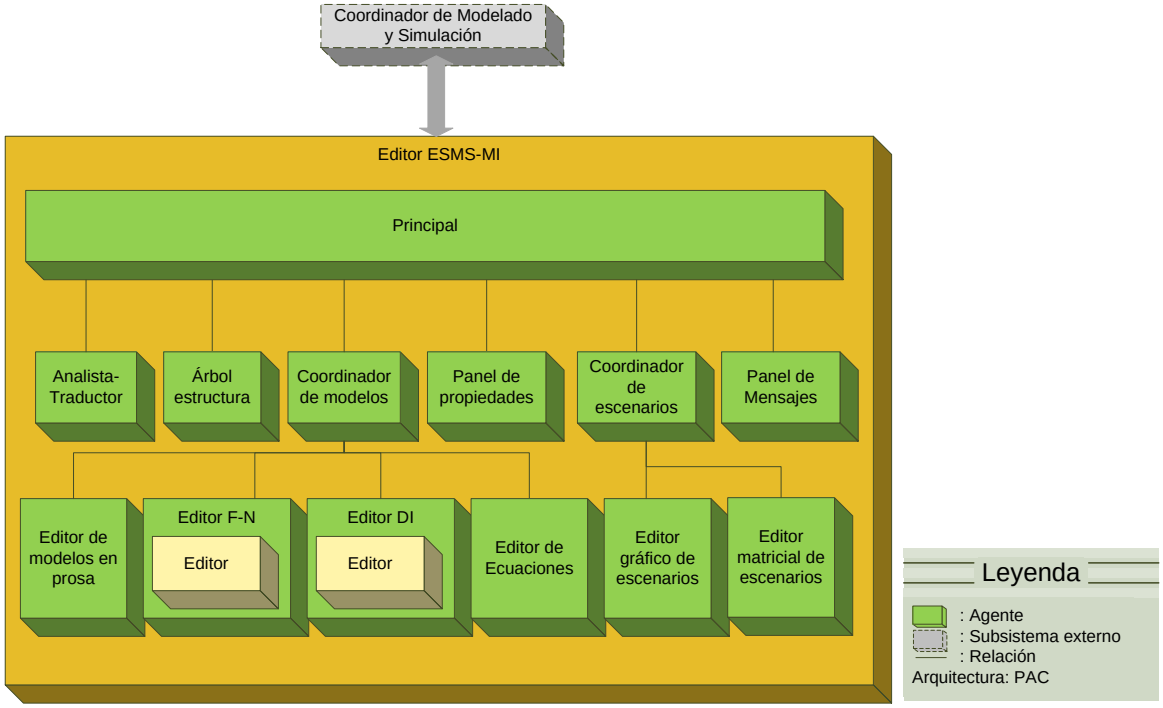
Figura 70. Arquitectura del ESMS-MI



EDITOR ESMS-MI

El subsistema Editor (ver **Figura 54**) permite al usuario realizar de forma gráfica e interactiva un modelo de simulación. Este subsistema contiene un conjunto de herramientas con los cuales el usuario puede realizar un modelo de forma gráfica. El editor debe permitir crear un modelo por el método de arrastrar-soltar o seleccionar-colocar y por medio de comandos de consola.

Figura 71. Editor del ESMS-MI



Estilo arquitectónico

El estilo arquitectónico del Editor es PAC. Tanto como el editor de flujo-nivel (F-N) como el editor de diagramas de influencias (DI) hacen uso del editor Editor de modelos visuales.

Estructura

El editor está conformado por siguientes agentes:

- Principal
- Analista-Traductor

- Coordinador de modelos
- Árbol estructura
- Panel de propiedades
- Coordinador de escenarios
- Panel de mensajes
- Editor de modelos en prosa
- Editor Flujo-Nivel
- Editor Diagrama de Influencias
- Editor de ecuaciones
- Editor gráfico de escenarios
- Editor matricial de escenarios

Responsabilidades

- Proveer interfaz gráfica de interacción sistema-usuario.
- Permite hacer modelos en los diferentes lenguajes de la D.S.
- Coordinar y sincronizar los diferentes modelos y diagramas.
- Generar un modelo a partir del diagrama introducido por el usuario.
- Administración de escenarios.

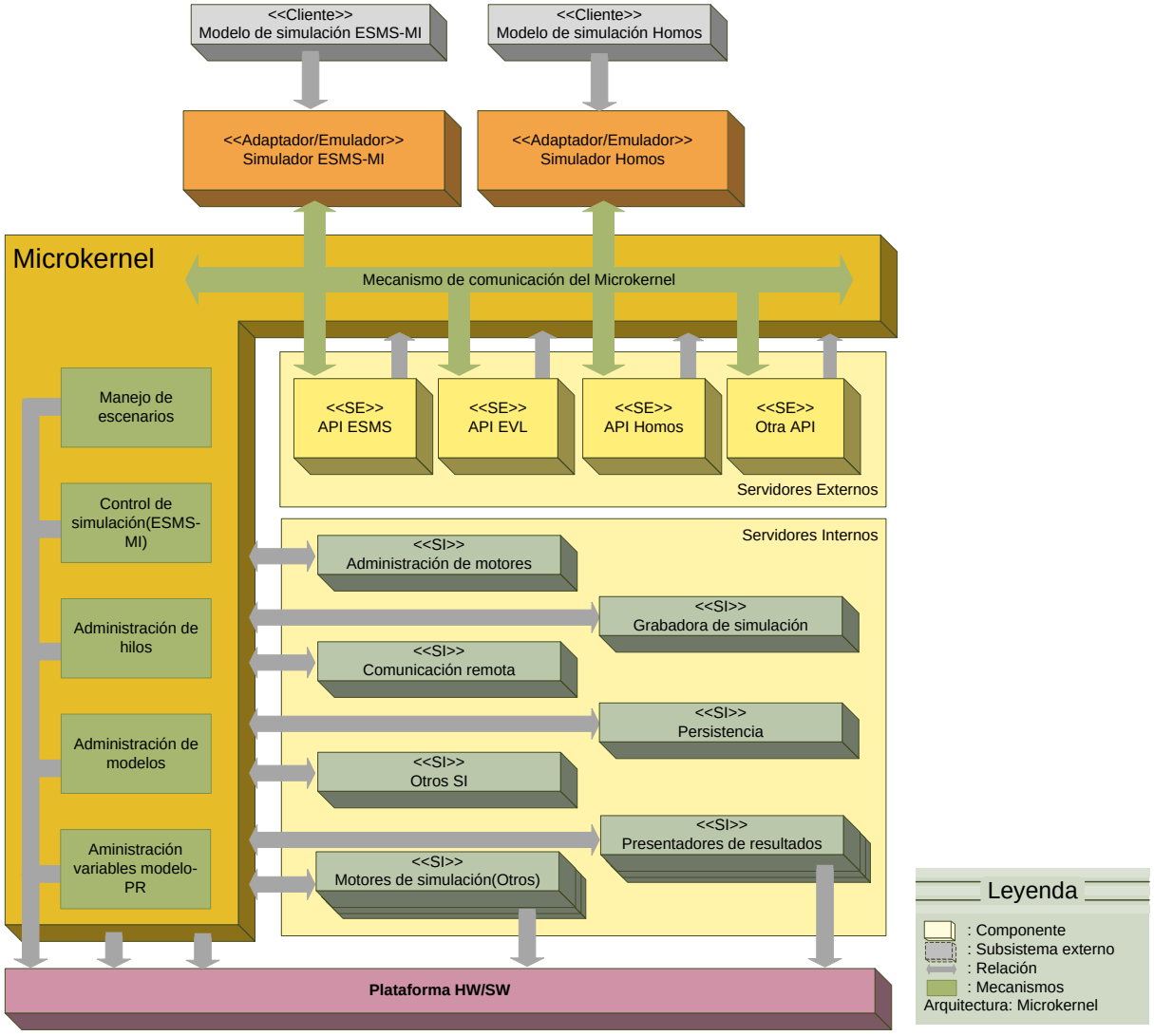
Relaciones y flujo de datos.

El Editor ESMS-MI se relaciona con el Coordinador de Modelado y Simulación (MyS), al cual le suministra los diagramas generados por el usuario. El diagrama correspondiente al modelo es convertido en un modelo entendible (incluyendo los escenarios) por el simulador y suministrado al coordinador de MyS. Igualmente, a través del coordinador envía los mensajes de error al manejador de errores. Por último, también por este medio envía los diferentes diagramas al subsistema de persistencia.

SIMULADOR

El subsistema simulador (ver **Figura 55**) permite realizar simulaciones a partir de un modelo de simulación suministrado. El simulador tiene diversas formas de presentación de los resultados, ya sea de forma tabular, gráfica o por medio de animaciones parametrizadas, es decir, la animación está regida por los valores que se obtienen de la simulación del modelo. El simulador presenta una interfaz gráfica que le permite al usuario interactuar con la simulación a través de algunos controles. Unos controles permiten controlar la simulación y otros permiten modificar los valores de las variables del modelo. El simulador encapsula los diferentes motores de simulación y las diferentes interfaces o presentadores de resultados.

Figura 72. Simulador



Estilo arquitectónico

El estilo arquitectónico del Simulador es Microkernel que es ideal para sistemas cambiantes. El componente principal es el Microkernel, el cual ofrece servicios atómicos denominados mecanismos, estos corresponden a la funcionalidad ofrecida por el núcleo del simulador. La funcionalidad de éste es extendida con los servicios ofrecidos por sus servidores internos. Los servidores externos colocan a disposición servicios más elaborados, creando plataformas de aplicación. Los clientes se comunican con los servidores externos por medio de las facilidades de comunicación ofrecidas por el Microkernel y que son encapsuladas por los adaptadores o emuladores. Los adaptadores también pueden implementar el patrón de diseño Proxy para la comunicación con los servidores externos remotos.

Estructura

El Simulador está conformado por los siguientes componentes:

- Microkernel
- Servidores internos
- Administración de motores
- Grabadora de simulación
- Comunicación remota
- Persistencia
- Presentadores de resultados
- Presentador de resultados liviano
- Presentadores de resultados estándar
- Presentadores de resultados Web
- Motores de simulación
- Motor de simulación de HOMOS
- Motor de simulación de LD
- Servidores externos
- API ESMS-MI
- API EVL
- API HOMOS

El motor del ESMS-MI hace parte del componente Microkernel. Tanto los presentadores de resultados como los motores de simulación son subsistemas complejos. Se recomienda que el componente motor se diseñe haciendo uso del patrón arquitectónico Tuberías y Filtros y los presentadores de resultado el patrón arquitectónico PAC.

Responsabilidades

- Realizar simulación de modelos integrados y otros tipos de modelos según los servidores externos implementados.

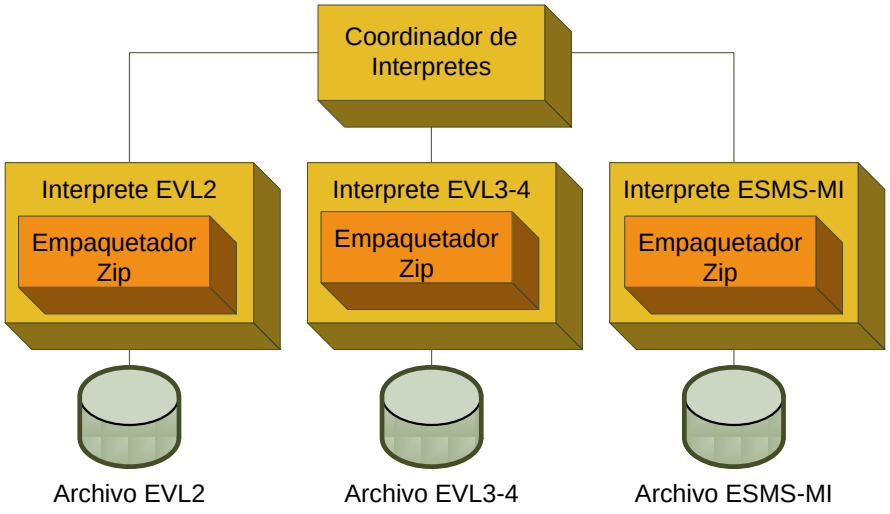
- Presentar por medio de interfaces gráficas y en diferentes plataformas Software-Hardware, los resultados de la simulación.
- Permitir a los usuarios interactuar con la simulación (cambiar valores de las variables durante la simulación y/o controlar la simulación) a través de controles.
- Interactuar con otros simuladores local y remotamente.
- Permitir interactuar con otras aplicaciones (enviar y recibir datos durante la simulación).

Relaciones y flujo de datos.
El simulador se relaciona con el Coordinador de Modelado y Simulación (MyS), el cual suministra la información necesaria para su funcionamiento como son el modelo y los escenarios de simulación.

PERSISTENCIA

El subsistema de persistencia (ver **Figura 56**) está conformado por Agentes encargados de gestionar los diferentes tipos de archivos soportados por el entorno. En la **Figura 56** se observa la estructura de este subsistema y los agentes encargados de abrir tres de los posibles tipos de archivos.

Figura 73. Subsistema de persistencia



Estilo arquitectónico
Este subsistema continúa con el patrón arquitectónico del entorno, por lo que tiene un agente encargado de la coordinación, denominado Coordinador de intérpretes.

Además, se cuenta con agentes denominados Intérpretes para los diferentes tipos de archivos soportados. Al menos se debe implementar intérpretes para los archivos de Evolución 2, 3.5-4 y el del entorno.

Estructura
El subsistema de persistencia está conformado por siguientes agentes:

- Coordinador de intérpretes.
- Intérprete EVL 2.
- Intérprete EVL 3-4.
- Intérprete ESMS-MI.

Responsabilidades

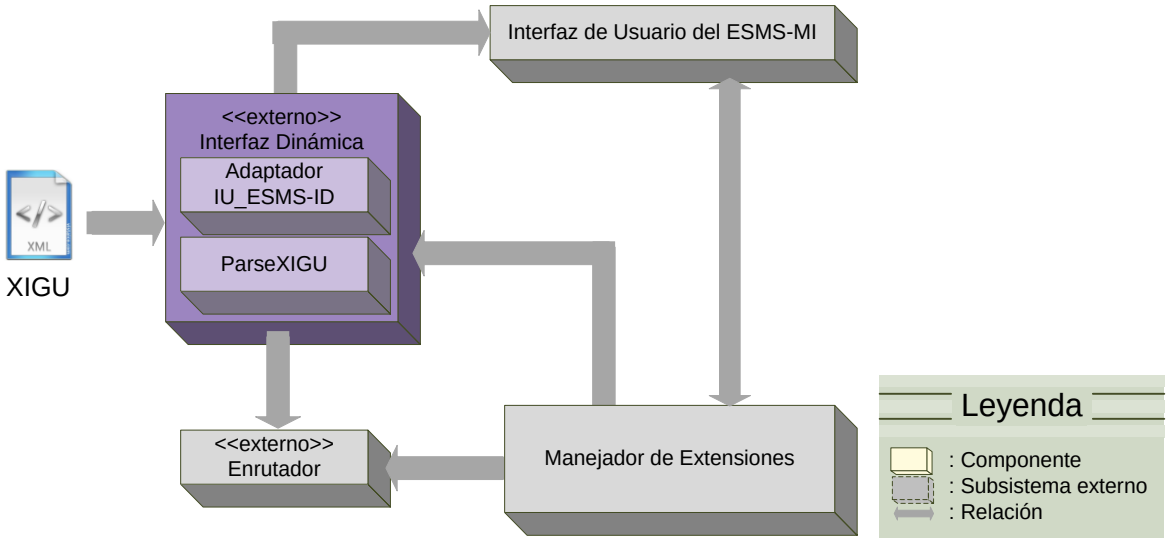
- Abrir y guardar archivos soportados por el entorno, suministrar y recibir los datos al entorno según corresponda.
- Seleccionar el procedimiento adecuado para poder abrir/guardar en los diferentes formatos.
- Compactar los datos (archivo de los diagramas, simuladores, modelo, etcétera) en un archivo comprimido según sea el caso.

Relaciones y flujo de datos.
Toda la comunicación se debe realizar por medio del componente Control del entorno. Este subsistema recibe por parte del entorno del ESMS-MI los datos a guardar en el archivo, igualmente entrega a éste los datos del archivo que se le solicite abrir.

INTERFAZ DINÁMICA

La interfaz dinámica (ver **Figura 57**) permite crear y administrar las Interfaz Gráfica de Usuario (IGU) creadas dinámicamente. Las IGU son creadas en tiempo de ejecución a partir de un archivo o cadena de texto con un script con código XIGU (Interfaz de Grafica de Usuario XML). Un archivo XIGU define una interfaz de usuario independiente de la plataforma, está escrito en el estándar XML y es interpretado por un ParserXIGU, el cual convierte el script en una interfaz real. La interfaz generada es acorde a la interfaz de todo el entorno.

Figura 74. Interfaz dinámica



Estilo arquitectónico
Se recomienda utilizar el patrón arquitectónico Capas. La API XIGU debe cumplir con el estándar XML.

Estructura
El subsistema Interfaz dinámica está conformada por:

- ParserXIGU
- Adaptador IU_ESMS-ID

Responsabilidades

- Interpretar archivos XIGU.
- Crear GUI a partir del script XIGU.
- Gestionar las IGU creadas dinámicamente.

Relaciones y flujo de datos.

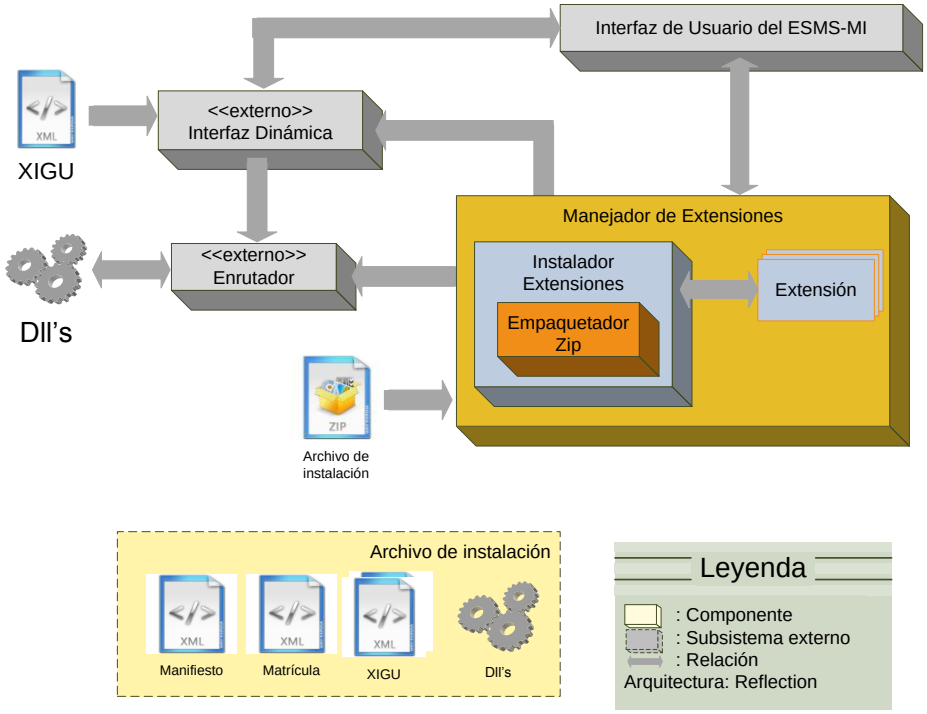
- El entorno

- Subsistema Manejador de extensiones
- Componente enrutador

Manejador de extensiones

Este subsistema es el encargado de gestionar la instalación de las extensiones, habilitarlas/deshabilitarlas, desinstalarlas y coordinar la ejecución su ejecución (ver **Figura 58**).

Figura 75. Manejador de extensiones



Estilo arquitectónico

El estilo arquitectónico para el manejador de extensiones es patrón Reflection.

Estructura

El subsistema Manejador de extensiones está conformado por los siguientes componentes:

- Instalador de extensiones.
- Extensiones

Responsabilidades

- Instalación/desinstalación de extensiones
- Provee acceso a las librerías y códigos de extensiones
- Proveer a las extensiones de mecanismos para la creación a la interfaz de usuario.
- Desempaquetar archivos de instalación.
- Verificar consistencia de los paquetes de instalación.
- Habiliatar/deshabilitar extensiones instaladas.
- Administración de las versiones de extensiones (verificar versión, comprobar actualizaciones, verificar compatibilidad, entre otras).

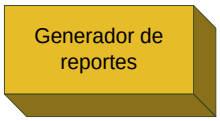
Relaciones y flujo de datos.

- Interfaz dinámica
- Enrutador

GENERADOR DE REPORTES

El subsistema Generador de reportes (ver **Figura 59**) permite generar un documento que contiene datos sobre el proceso de modelado y simulación, tales como imágenes del modelo, ecuaciones, descripciones de elementos, resultados de una simulación, gráficas de resultados, etcétera. Este reporte es personalizable, permitiendo al usuario seleccionar las partes que desea que aparezcan en este. También, soporta el uso de plantillas predefinidas para la elaboración de reportes. El documento generado se debe poder exportar a diferentes tipos de archivos como: Open Document, MS-Word y Html.

Figura 76. Generador de reportes



Estilo arquitectónico
Se recomienda utilizar el patrón arquitectónico Capas.

Responsabilidades

- Generación de reportes en diferentes formatos.
- Configuración de reportes.
- Administrar plantillas de reportes

Relaciones y flujo de datos.
Se relaciona con el Entorno. El generador de reportes toma los datos para armar el reporte, de un archivo XML que genera el ESMS-MI, el cual solicita información a cada uno de los componentes o subsistemas.

ARQUITECTURA DETALLADA

La arquitectura detallada o arquitectura lógica define de forma precisa los mecanismos de conexión y las interfaces entre los componentes y los protocolos.

A continuación se detallan las interfaces de los principales componentes del entorno.

INTERFAZ SIMULADOR

Información <<interfase>>
Nombre: string read FNombre write SetNombre;
Descripcion:TDescripcionAnimador //el usuario agrega la descripción del animador
FechaCreacion:TDate read FFechaCreacion; //se guarda la fecha en que se creó el animador
Autor:
Persistencia <<interfase>>
Cargar(S: TDatosEnMemoria):boolean;virtual; xml
Guardar(S: TDatosEnMemoria):boolean;virtual;
ScriptSimulación <<interfase>>
CargarScript
EjecutarScript
EjecutarScriptPasoAPaso
DetenerScript
ControlSimulación <<interfase>>
AumentarVelocidad
DisminuirVelocidad
ControlSimulacion <<interfase>>
RegresarAlEscenarioInicial: boolean
Iniciar;virtual;
Detener;virtual;
Pausa;virtual;
PasoDeSimulacion;virtual;
EstadoSimulacion:TEstadoSimulacion read FEstadoSimulacion write SetEstadoSimulacion;
Animar(lista de variables modelo, lista variables sistema)
VentanaSimulacion <<interfase>>
CrearVentana(CS:TCondicionesDeSimulacion):entero //retorna el código de la ventana
EliminarVentana(idVentana:entero):booleano //retorna falso si hay error
ModificarVentana(idVentana:entero; nomVariable: texto; Valor:texto) :booleano //retorna falso si hay error

TPropiedadesVentana <<tipo>>
TiempoInicial
TiempoFinal
Delta
Velocidad

CondicionesDeSimulacion <<interfase>>
CargarCondicionesSimulacion
EstablecerTiempoInicial
EstablecerTiempoFinal
EstablecerAlgoritmoSimulacion(texto) <<en DS corresponde al método de integración>>
EstablecerDeltaSimulacion

RolBack
SnapShoot(unPaso:integer)

INTERFAZ MOTOR

Modelo
DefinirEscenario(TObject) read getEscenario write SetEscenario;
ControlSimulación
    AumentarVelocidad
    DisminuirVelocidad
ControlSimulacion
RegresarAlEscenarioInicial: boolean
PuedeIniciar:boolean;virtual;
    Iniciar;virtual;
    Detener;virtual;
    Pausa;virtual;
    PasoDeSimulacion;virtual;
EstadoSimulacion:TEstadoSimulacion read FEstadoSimulacion write SetEstadoSimulacion;
Grabacion
ActivarGrabarSimulacion
DetenerGrabarSimulacion
IrAPasoAnterior
Persistencia
CargarModelo(S: TDatosEnMemoria): boolean;virtual;
    GuardarModelo(S: TDatosEnMemoria): boolean;virtual;
Consulta
    TiempoDeSimaulacion

Motor.iniciar
Motor.pausar
Motor.pasoSimulacion
Motor.detener
Motor.cargarSnapShoot(SnapShoot)

INTERFAZ EDITOR DE MODELOS VISUALES (DIAGRAMAS)

Persistencia
AbrirModeloDeArchivo(Archivo:string):boolean;
GuardarModeloEnArchivo(Archivo:string):boolean;
AbrirDiagrama(EspacioMemoria)
GuardarDiagrama(EspacioMemoria)
Metamodelo
AgregarMetamodelo
ListaDeMetamodelos
EliminarMetamodelo
Edicion
AgregarElemento(tipo, nombre: string; posX, PosY: integer):true;
EliminarElemento
SeleccionarElemento
UbicarElemento
AgregarElementoASeleccion
EliminarElementoDeSeleccion
ModificarPropiedadElemento(elemento: string; propiedad:string; valor:string)
ElementosSeleccionados:Lista
CopiarSelección
CortarSelección
DeshacerUltimaAccion
Consulta
ExisteUnElementoSeleccionado: boolean
NumeroPropiedadesElementoSeleccionado: integer
Exportar
ExportarModeloGrafico(En_donde:string; Filtro:integer);
ExportarModeloTexto(En_donde:string; Filtro:integer);
Importar

INTERFAZ DEL ENTORNO

Administracion/Seguridad
ProtegerModelo(contrasena)

CambiarContrasena(contrasenaAnterior; contrasenaNueva)
EditarElementoSeleccionado;
EditarEscenarios;
EnviarMensajeSector(ObligarAUno:boolean=false);
EnviarMensajeOrdenes;
ImprimirDiagramaForresterEnCanvas(Cual_canvas: TCanvas;
ImprimirDescripcionElementosForrester(Cual_Canvas: TCanvas;
HacerReporteDiagramaForresterEnCanvas(Documento : TDocumentoWord;
HacerReporteDescripcionElementosForrester(Documento:TDocumentoWord;
HacerReporteSoloDescripcionYAutor(Documento : TDocumentoWord;
KeyDown(var Key: Word; Shift: TShiftState); override;
MouseWheelHandler(var Message: TMessage); override;
Pegar;
ReHacer;
VerAdministracion;
ZoomMas;
ZoomMenos;
AcercaDe;
Guardado:boolean read GetGuardado write SetGuardado;
DatosProyecto:TDatosProyecto
ElementoSeleccionado:TTipoElemento
EscenarioActual:string
ZoomValor:single read GetZoomValor write SetZoomValor;
TipoPlumaSector: TPenStyle
AnchoPlumaSector: Integer
FormaSectorSector: TDrawingTool;
ColorSector: TColor;
ColoresPersonalizadosDelFormato: TStringList;
ManejadorElementos: TManejadorElementos;
MantenerTipoElemento:Boolean;
OnMensaje:TEventoMensaje;
OnNuevaPagina:TNuevaHojaEvent;
Visible: Boolean;
end;

Referencias

Bass, Len y Coutaz, Joëlle. 1991. *Developing software for the user interface.* s.l. : Addison-Wesley Publishing Company, 1991.

Buschmann, Frank, y otros. 1996. *Pattern-Oriented Software Architecture Volume 1: A System of Patterns.* West Sussex : John Wiley & Sons Ltd, 1996. pág. 476. Vol. 1. ISBN: 978-0471958697.

Garlan, David y Shaw, Mary. 1994. *An Introduction to Software Architecture.* Pittsburgh : s.n., 1994. Reporte tecnico. CS-94-166.

Hussey, Andrew y Carrington, David. 1996. *Comparing two user-interface architectures: MVC and PAC.* Queensland : s.n., 1996. Reporte técnico.

Malan, Ruth y Bredemeyer, Dana. 2004. The Visual Architecting Process. [En línea] 19 de Abril de 2004. [Citado el: 22 de Enero de 2008.] Libro digital. http://www.ruthmalan.com.

The construction of user interfaces and the object paradigm. **Coutaz, Joëlle. 1987.** Paris : Springer-Verlag, 1987. European conference on object-oriented programming on ECOOP '87. págs. 121-130. ISBN:0-387-18353-1.

Anexo C. Ficha técnica de la arquitectura

(En CD)

Anexo D. Presentación con diapositivas del macro-proyecto ESMS-MI

(En CD)

Anexo E. Formato para la Solicitud de participación de cada uno de los integrantes de la comunidad

(Espacio intencional para conservar el formato)

Señores:

Comunidad I+D del ESMS-MI

Colombia

La presente tiene como objetivo, solicitar formalmente mi participación en el proyecto para el desarrollo en comunidad del Entorno Software de Modelado y Simulación de Modelos Integrados (ESMS-MI)[48]. Igualmente, solicitar la vinculación de mi grupo de Investigación y/o desarrollo, a la comunidad ESMS-MI, que tiene a su cargo el desarrollo de dicho entorno. A continuación suministro los datos requeridos para tal efecto:

Fecha de solicitud: Haga clic aquí para escribir texto.
Grupo de Investigación: Haga clic aquí para escribir texto.
Director: Haga clic aquí para escribir texto.
Responsable ante la comunidad[49]: Haga clic aquí para escribir texto.
Solicitante[50]: Haga clic aquí para escribir texto.
Rol[51]: Haga clic aquí para escribir texto.

Una vez aceptada la presente solicitud, me comprometo a cumplir con las normas establecidas por la comunidad.

Firma del solicitante
C.C:

Firma Responsable
C.C:

[48] *Comunidad conformada por grupos de investigación y/o desarrollo geográficamente dispersos. Esta comunidad es la encargada del desarrollo del proyecto ESMS-MI.*

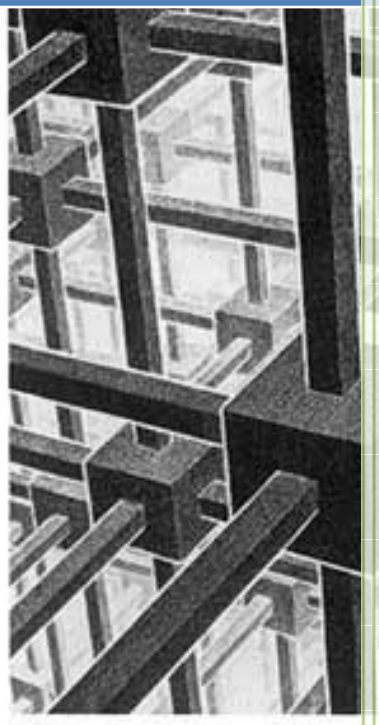
[49] *Integrante del grupo de investigación que responde por los sub proyectos del ESMS-MI a cargo del grupo. Dicho integrante representa al grupo ante la comunidad ESMS-MI.*

[50] *Integrante del grupo que solicita la vinculación a la comunidad.*

[51] *Representante, Director de proyecto(o codirector), Desarrollador.*

Anexo F. Orientaciones de uso de la plataforma Moodle para el desarrollo del ESMS

(Espacio intencional para conservar el formato)



INTRODUCCIÓN

La plataforma software de la comunidad debe apoyar la realización de actividades como la gestión de la comunidad, comunicación entre integrantes, la publicación, el compartir archivos, debe ofrecer herramientas para el desarrollo, control de versiones, la creación y edición de documentos, y la formación y realimentación a partir de experiencias.

Como plataforma software se ha seleccionado un grupo de herramientas y servicios, que en conjunto, abarcan las necesidades de comunicación, gestión de la comunidad y el desarrollo del proyecto ESMS-MI. Para cada una de estas herramientas se deben analizar aspectos como: la facilidad para el trabajo colaborativo, facilidades de acceso (licencias, curva de aprendizaje, acceso remoto…), prestaciones y funcionalidades.

HERRAMIENTAS PARA LA GESTIÓN DE LA COMUNIDAD

La principal herramienta de la plataforma es el software para la gestión e integración de la comunidad, para estos propósitos se ha seleccionado el software Moodle (software para la gestión de cursos y contenidos), este ha sido adaptado para su funcionamiento de acuerdo a la estructura de la comunidad desarrolladora. Esta herramienta dará soporte para las labores de integración, promoción, comunicación, coordinación y gestión de la comunidad desarrolladora del entorno. Esta herramienta es OpenSource y permite la distribución de archivos, creación de foros, Chat, Wikis, listas de correos (por medio de foros), entre otros. Es una herramienta web que requiere de un servidor web, base de datos MySQL y PHP.

Esta plataforma se encuentra en funcionamiento en la dirección: http://simon.uis.edu.co/comunidadesms.

HERRAMIENTAS PARA LA COMUNICACIÓN Y LA COORDINACIÓN ENTRE LOS INTEGRANTES DE LA COMUNIDAD

La comunicación entre los integrantes de la comunidad no debe ser solo de forma textual, sino con el uso de imágenes, sonidos y videos. Las comunicaciones pueden ser sincrónicas haciendo uso de chat, mensajería instantánea, llamadas, etc o asincrónicas con el uso de Foros, coreo, etc. La comunicación es un reto en actividades no presenciales como son las reuniones entre integrantes de la

comunidad, presentaciones, orientaciones o incluso para las discusiones y debates entre personas.

En lo que respecta a las comunicaciones sincrónicas, se recomienda el uso de la herramienta Skype, ya que presenta las siguientes características:

- Facilita la comunicación de diversas formas, en una sola herramienta (llamadas, mensajería instantánea, video llamada y video conferencia) con calidad de transmisión (depende de la conexión a internet).
- Se puede establecer diferentes estados de conexión como conectado, ausente, ocupado, etcétera, para configurar dependiendo la disponibilidad del usuario.
- No es invasivo con mensajes de información (por ejemplo cuando se conecta un contacto) o recibir mensajes en momentos no deseados (según el tipo de estado seleccionado).
- El software es libre y el servicio es gratuito.
- Guarda el historial de las conversaciones de forma automática (el usuario puede eliminar las conversaciones que desee).
- Permite transferir archivos (incluso ejecutables) y en caso de caerse la conexión reanuda la transferencia desde el punto en que se encontraba.
- Fácil de manejar. Su funcionamiento es parecido a la mayoría de software de mensajería instantánea disponibles en el mercado, lo cual facilita su aprendizaje.
- Para acceder al servicio debe estar instalado el software. El historial de conversaciones es almacenado en la respectiva máquina.

También es posible la comunicación sincrónica por medio del Chat ofrecido por el sitio web de la comunidad.

Para comunicaciones asincrónicas se puede hacer uso de foros y Wikis. Estas herramientas pueden crearse al interior del sitio web de la comunidad. Ejemplo de uso: para debatir y discutir la toma de una decisión por toda la comunidad (un foro).

HERRAMIENTAS PARA COMPARTIR ARCHIVOS

Para compartir archivos debemos examinar primero la naturaleza y fin de los mismos. Si el propósito es compartir documentación para toda la comunidad, se debe hacer uso del sitio web de la comunidad. Pero si se trata de compartir documentación que no son de interés para toda la comunidad o son versiones no

acabadas (borradores) usadas entre grupos o desarrolladores, se puede hacer uso de Skype, Dropbox o Office Workspace o incluso el correo electrónico.

HERRAMIENTAS DE DESARROLLO

HERRAMIENTAS PARA LA PLANEACIÓN DE PROYECTOS

Para la gestión del proyecto y seguimiento de cronogramas se hace uso de DotProjet, herramienta Web, libre y de código abierto, alternativa a las herramientas de su tipo que son de código propietario. Al ser una herramienta Web permite el seguimiento y control del cronograma de trabajo por parte de toda la comunidad, ya que estos pueden ser accedidos y modificados de forma remota. Para los usuarios de MS-Project se facilitará su aprendizaje debido a sus similitudes, como la presentación del cronograma por medio del diagrama de Grantt.

Dotproject se encuentra en funcionamiento y a disposición de la comunidad en la dirección http://simon.uis.edu.co/dotproject.

HERRAMIENTAS DE MODELADO CON UML

La herramienta de modelado seleccionada para la elaboración de modelos de sistemas software debe cumplir las siguientes características:

- Utilizar la notación UML: Es la notación más extendida en nuestro país, por lo tanto disminuye la probabilidad de requerir el aprendizaje de una notación de modelado por parte de los integrantes de la comunidad desarrolladora. Específicamente debe aceptar la notación UML2.0.
- Uso gratuito y manejo del formato XMI (Object Management Group, 2005): El uso de herramientas gratuitas facilita que toda la comunidad desarrolladora pueda tener a disposición la misma herramienta. Sin embargo, en caso de que sea utilizada una herramienta de software propietario esta debe poder exportar e importar el modelo en formato XMI.
- Organización de modelos: Que permita organizar los modelos en carpetas, submodelos y/o paquetes, para facilitar el entendimiento del modelo.
- Patrones de diseño: Debe brindar herramientas para facilitar la utilización (y reutilización) de patrones de diseño

Se recomienda el uso de la misma herramienta para toda la comunidad de desarrollo. En caso de no se llegue a un consenso, se debe hacer uso de herramientas que acepten el estándar XMI. Por todo la anterior, se propone el uso

de StarUML herramienta que cumple con las especificaciones anteriores y además, brinda las funcionalidades para la generación de código, ingeniería inversa y generación de documentación (por ejemplo documento de casos de uso en Word). StarUML es de fácil manejo y ofrece las funcionalidades de la mayoría de las herramientas de su tipo. Es un software mono usuario y no ofrece funcionalidades para el trabajo de forma remota.

HERRAMIENTAS PARA EL CONTROL DE VERSIONES AL CÓDIGO FUENTE

En cuanto a las herramientas para el control de versiones se ha seleccionado Subversion. Esta herramienta permite llevar el control de versiones del código fuente durante el desarrollo del ESMS. Subversion permite acceder de forma remota a un repositorio de archivos, el cual es como un servidor de archivos ordinario, excepto porque recuerda todos los cambios hechos a sus archivos y directorios (Collins-Sussman, y otros, 2009). Esto permite recuperar versiones antiguas de los datos, o examinar el historial de cambios de los mismos. Subversion puede acceder al repositorio de archivos a través de redes o la Web, lo que le permite ser usado por personas que se encuentran en diferentes lugares facilitando el trabajo colaborativo. Esta herramienta es de uso libre y código abierto.

HERRAMIENTAS PARA LA CREACIÓN Y EDICIÓN DE DOCUMENTOS

Para la creación y edición de documentos es posible utilizar cualquier software para la edición de documentos, pero estos deben soportar el formato de archivo de Office, el cual se propone como estándar de archivo de la comunidad. Para la publicación de documentos definitivos (para la comunidad usuaria) se debe exportar a formato PDF.

Los documentos de construcción colectiva (o colaborativa) como el manual de usuario o el documento de diseño, se recomienda utilizar Wikis. Rodolfo Pilas define a los Wikis en (Herramientas para el desarrollo distribuido y cooperativo de software, 2005) como una aplicación de informática colaborativa que permite crear colectivamente documentos web usando un simple esquema de etiquetas y marcas, sin que la revisión del contenido sea necesaria antes de su aceptación para ser publicado en el sitio web. En este documento también son presentadas las siguientes ventajas de los Wikis para la creación de documentos colectivos:

- Edición simple, rápida y sencilla.
- Control automático de versiones y cambios.

- Hipertexto simple (crear nuevas páginas)
- Índice, Capítulos (espacios wiki) automáticos
- Inclusión de imágenes, exportación, impresión, etc.

HERRAMIENTAS PARA LA FORMACIÓN Y REALIMENTACIÓN A PARTIR DE LAS EXPERIENCIAS.

En todo proceso de trabajo en red o en comunidad se genera conocimiento y experiencias que se debe formalizar y publicar, así como la documentación referente al proceso de desarrollo, resultante de su mejoramiento continuo por parte de la comunidad. Para estos propósitos se recomienda el uso de foros, Wikis y Blog como medio o recurso de registro y publicación.

Haga clic aquí para acceder a los foros de la comunidad.

Haga clic aquí para acceder a los Wikis de la comunidad.

OTRAS POSIBLES HERRAMIENTAS

Existen otras herramientas y servicios web que pueden ser evaluados y utilizados por la comunidad para diferentes propósitos. Por ejemplo existen diferentes formas de compartir escritorios y aplicaciones como Microsoft SharedView o Microsoft NetMeeting, esto puede facilitar explicaciones o debates y discusiones entre los integrantes de la comunidad. Otra posible herramienta es el uso de software para el reporte de errores y solicitud de nuevos requerimientos (bug tracking system) como es el caso de Bugzilla o Mantis.

La inclusión, cambio o eliminación de herramientas de la plataforma de la comunidad es responsabilidad de la misma y de sus integrantes, basados en criterios de usabilidad, funcionalidad y estandarización.

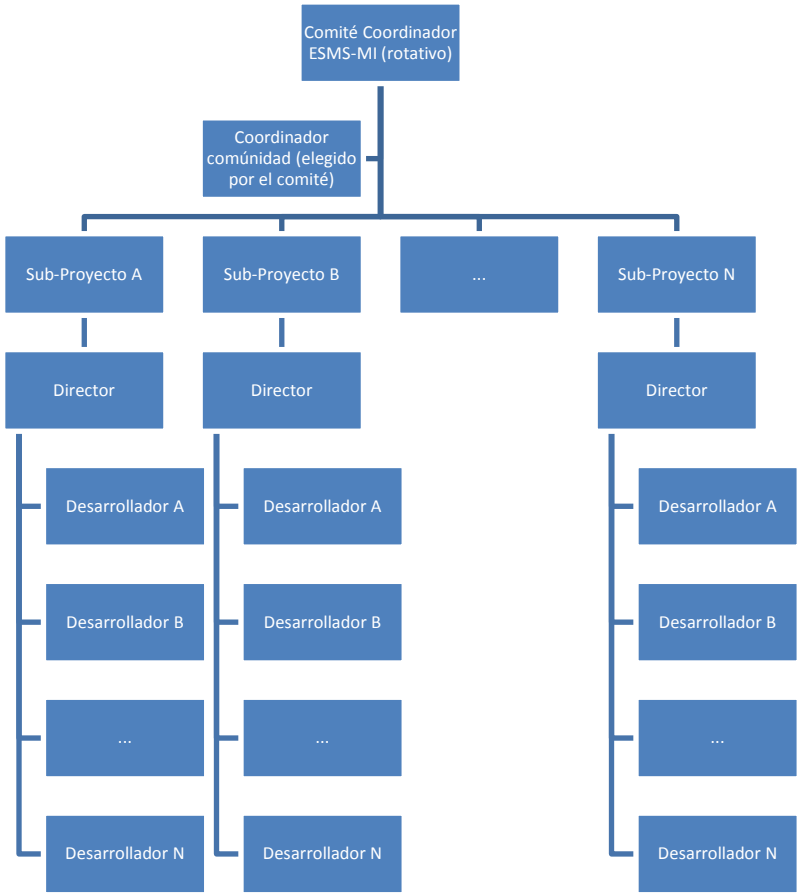
DESCRIPCIÓN DE HERRAMIENTAS

MOODLE

Moodle ha sido seleccionada como sitio web para la integración de la comunidad por las siguientes características: es una herramienta OpenSource que permite la distribución de archivos, creación de foros, Chat, Wikis, listas de correos (por medio de foros), entre otros. Moodle requiere de un servidor web, base de datos MySQL y PHP.

La estructura organizacional de la comunidad desarrolladora del Entorno Software de Modelado y Simulación de Modelos Integrados (ESMS-MI) es presentada en la **Figura 77**, esta es coherente con la organización que presenta el sitio Moodle, al cual se puede acceder en la dirección http://simon.uis.edu.co/comunidadesms. A este sitio se le llamará "sitio web de la comunidad".

Figura 77. Estructura organizacional de la comunidad (Organigrama)



La estructura de la comunidad se ve reflejada en sitio web de la comunidad, de la siguiente manera:

Una categoría corresponde a un grupo de investigación y desarrollo que participan de la elaboración del ESMS-MI. En cada grupo de investigación existe un responsable ante la comunidad de desarrollo de uno o varios sub-proyecto. Cada sub-proyecto se desarrolla normalmente por un proyecto de pregrado (aunque podría realizarse por más de un proyecto de pregrado o de postgrado) el cual desarrolla un módulo (o varios relacionados); en caso de que un módulo sea abordado por más de un proyecto de grado, sus integrantes deben ser miembros del mismo curso (sus integrantes deben coordinar y compartir entre sí el curso). Cada uno de los cursos tiene una estructura de carpetas que corresponde a la estructura de Agile-Disop, así:

ESTRUCTURA DE CARPETAS DE AGILEDISOP

Visión_del_Proyecto [.doc] [abrir]

02 Plan_Proyecto (F:\AgileDisop\02 Elaboracion\Iteracion\03_Gestion_Proyecto\02 Plan_Proyecto)

Plantilla_Plan_Proyecto_SW [.doc] [abrir]

03 Requisitos (F:\AgileDisop\02 Elaboracion\Iteracion\03_Gestion_Proyecto\03 Requisitos)

Plantilla_Elicitacion_Requisitos [.doc] [abrir]

04 Plan_Iteracion (F:\AgileDisop\02 Elaboracion\Iteracion \03_Gestion_Proyecto\04 Plan_Iteracion)

Plantilla_Plan_Iteracion [.dot] [abrir]

05 Riesgos (F:\AgileDisop\02 Elaboracion\Iteracion\03_Gestion_Proyecto\05 Riesgos)

Anexo10_Modelo_Gestion_de_Riesgos [.xls] [abrir]

04_Desarrollo (F:\AgileDisop\02 Elaboracion\Iteracion\04_Desarrollo)

01 Requisitos-Analisis (F:\AgileDisop\02 Elaboracion\Iteracion\04_Desarrollo\01 Requisitos-Analisis)

1_DCUSO (F:\AgileDisop\02 Elaboracion\Iteracion\04_Desarrollo\01 Requisitos-Analisis\1_DCUSO)

Actividad_DCasosUso [.doc] [abrir]

Plantilla_DCasosUso [.doc] [abrir]

2_CUSO (F:\AgileDisop\02 Elaboracion\Iteracion\04_Desarrollo\01 Requisitos-Analisis\2_CUSO)

Actividad_CUso [.doc] [abrir]

especificacionCasoUso [.dot] [abrir]

Plantilla_CUso_Alto_Nivel [.doc] [abrir]

Plantilla_CUso_Expandido [.doc] [abrir]

3_DSecuencia (F:\AgileDisop\02 Elaboracion\Iteracion\04_Desarrollo\01 Requisitos-Analisis\3_DSecuencia)

Actividad_DSecuencia [.doc] [abrir]

Plantilla_DSecuencia [.doc] [abrir]

4_Contratos (F:\AgileDisop\02 Elaboracion\Iteracion\04_Desarrollo\01 Requisitos-Analisis\4_Contratos)

Actividad_Contratos [.doc] [abrir]

Plantilla_Contrato [.doc] [abrir]

5_MConceptual (F:\AgileDisop\02 Elaboracion\Iteracion\04_Desarrollo\01 Requisitos-Analisis\5_MConceptual)

Actividad_MConceptual [.doc] [abrir]

Plantilla_ModConceptual [.doc] [abrir]

6_Glosario (F:\AgileDisop\02 Elaboracion\Iteracion\04_Desarrollo\01 Requisitos-Analisis\6_Glosario)

Actividad_Glosario [.doc] [abrir]

Plantilla_Glosario [.doc] [abrir]

02 Diseño (F:\AgileDisop\02 Elaboracion\Iteracion\04_Desarrollo\02 Diseño)

1_IGU (F:\AgileDisop\02 Elaboracion\Iteracion\04_Desarrollo\02 Diseño\1_IGU)

Actividad_Diseño_IGU [.doc] [abrir]

2_CRealesUso (F:\AgileDisop\02 Elaboracion\Iteracion\04_Desarrollo\02 Diseño\2_CRealesUso)

Actividad_CUso_Real [.doc] [abrir]

IO [.jpg] [abrir]

Plantilla_CUso_Real [.doc] [abrir]

3_DPaquetes (F:\AgileDisop\02 Elaboracion\Iteracion\04_Desarrollo\02 Diseño\3_DPaquetes)

Actividad_Arquitectura [.doc] [abrir]

Plantilla_DPaquetes [.doc] [abrir]

4_DColaboracion (F:\AgileDisop\02 Elaboracion\Iteracion\04_Desarrollo\02 Diseño\4_DColaboracion)

Actividad_DInteración [.doc] [abrir]

Plantilla_DInteraccion [.doc] [abrir]

5_DClases (F:\AgileDisop\02 Elaboracion\Iteracion\04_Desarrollo\02 Diseño\5_DClases)

Actividad_DClases [.doc] [abrir]

Plantilla_DClases [.doc] [abrir]

6_EsquemaBDatos (F:\AgileDisop\02 Elaboracion\Iteracion\04_Desarrollo\02 Diseño\6_EsquemaBDatos)

Actividad_EsquemasBD [.doc] [abrir]

- Plantilla_EsquemasBD [.doc] [abrir]
- **7_Reportes** (F:\AgileDisop\02 Elaboracion\Iteracion\04_Desarrollo\02 Diseño\7_Reportes)
  - Actividad_Reportes [.doc] [abrir]
  - Plantilla_Reportes [.doc] [abrir]

03 Implementacion (F:\AgileDisop\02 Elaboracion\Iteracion\04_Desarrollo\03 Implementacion)

- **1_IGU** (F:\AgileDisop\02 Elaboracion\Iteracion\04_Desarrollo\03 Implementacion\1_IGU)
  - Actividad_Implementar_IGU [.doc] [abrir]
- **2_Paquetes_Clases** (F:\AgileDisop\02 Elaboracion\Iteracion\04_Desarrollo\03 Implementacion\2_Paquetes_Clases)
  - Actividad_DefClases [.doc] [abrir]
  - Actividad_DefPaquetes [.doc] [abrir]
  - Plantilla_DefClase [.doc] [abrir]
- **3_Base_Datos** (F:\AgileDisop\02 Elaboracion\Iteracion\04_Desarrollo\03 Implementacion\3_Base_Datos)
  - Actividad_ImpBDs [.doc] [abrir]
- **4_Reportes** (F:\AgileDisop\02 Elaboracion\Iteracion\04_Desarrollo\03 Implementacion\4_Reportes)
  - Actividad_ImpReportes [.doc] [abrir]
  - Plantilla_Reporte [.doc] [abrir]

04 Pruebas (F:\AgileDisop\02 Elaboracion\Iteracion\04_Desarrollo\04 Pruebas)

- **1_Plan** (F:\AgileDisop\02 Elaboracion\Iteracion\04_Desarrollo\04 Pruebas\1_Plan)
  - planPruebas [.dot] [abrir]
- **2_Ejecucion** (F:\AgileDisop\02 Elaboracion\Iteracion\04_Desarrollo\04 Pruebas\2_ Ejecucion)
  - **01_Casos_Procs** (F:\AgileDisop\02 Elaboracion\Iteracion \04_Desarrollo\04 Pruebas\2_Ejecucion\01_Casos_Procs)
    - Actividad_Caso_Prueba [.doc] [abrir]
    - Actividad_Procedimiento [.doc] [abrir]
    - Plantilla_Caso [.doc] [abrir]


Plantilla_Procedimiento [.doc] [abrir]
02_Evl_Arquitectonica (F:\AgileDisop\02 Elaboracion\Iteracion \04_Desarrollo\04 Pruebas\2_Ejecucion\02_Evl_Arquitectonica)
Evl_Arquitectura [.doc] [abrir]
Graficos_Evolucion [.xls] [abrir]
Metricas_Evolucion [.xls] [abrir]
4_Evaluacion (F:\AgileDisop\02 Elaboracion\Iteracion\04_Desarrollo\04 Pruebas\4_Evaluacion)
EvaluacionPruebas [.dot] [abrir]
05 Mejora_Proceso (F:\AgileDisop\02 Elaboracion\Iteracion\05 Mejora_Proceso)
Areas_Estrategias_Impactos_de_Mejora [.doc] [abrir]
Formalizacion_Estrategias_Mejora [.doc] [abrir]
06 Gestion_Cambio (F:\AgileDisop\02 Elaboracion\Iteracion\06 Gestion_Cambio)
Solicitudes_de_Cambio [.doc] [abrir]
Agile_DISOP_Iteracion [.ppt] [abrir]
03 Construccion (F:\AgileDisop\03 Construccion)
Hito (F:\AgileDisop\03 Construccion\Hito)
04 Transicion (F:\AgileDisop\04 Transicion)
1_Documentacion_Tecnica (F:\AgileDisop\04 Transicion\1_Documentacion_Tecnica)
2_Documentacion_Usuario (F:\AgileDisop\04 Transicion\2_Documentacion_Usuario)
3_Pruebas del Sistema (F:\AgileDisop\04 Transicion\3_Pruebas del Sistema)
4_Pruebas_Aceptacion (F:\AgileDisop\04 Transicion\4_Pruebas_Aceptacion)
5_Pruebas_Documentacion (F:\AgileDisop\04 Transicion\5_Pruebas_Documentacion)
6_Capacitacion (F:\AgileDisop\04 Transicion\6_Capacitacion)
7_Ejecuciones (F:\AgileDisop\04 Transicion\7_Ejecuciones)
8_Soporte (F:\AgileDisop\04 Transicion\8_Soporte)
9_Instalacion (F:\AgileDisop\04 Transicion\9_Instalacion)
Hito (F:\AgileDisop\04 Transicion\Hito)

Cada sub-proyecto debe decidir cuándo debe publicar archivos a toda la comunidad (liberaciones). Al menos debe colocar a disposición códigos fuentes y ejecutables o dll.

Pasos :

1. Crear una categoría para el grupo de investigación (si no está creado)
2. Crear un curso por cada uno de los proyectos de pregrado
    a. Seleccionar la categoría según su grupo de investigación
    b. Coloque el Nombre completo del sub-proyecto
    c. Coloque un Nombre corto para el sub-proyecto
    d. Seleccione el Formato “Temas”
    e. Establezca el Numero de temas a 5 (luego se puede modificar encaso de necesitarse un número mayor)
    f. Renombrar roles
        i. Profesor → Director de proyecto
        ii. Estudiante → Desarrollador
3. Asignar los directores (directores y codirectores del proyecto) y desarrolladores (estudiantes) al curso
4. Crear la estructura de carpetas de Agile-Disop (para este efecto se colocará un archivo Zip del cual permite crear dicha estructura en Moodle)

DOTPROJECT

Para la gestión del proyecto y seguimiento de cronogramas y tareas, se ha seleccionado DotProject, la cual es una herramienta de gestión de proyectos de software libre.

En la siguiente dirección http://tutoriales.innox.com.mx/vhost/tutoriales/completo/index.html puede encontrar un conjunto de video-tutoriales sobre el uso de Dotproject. Se sugiere que el nombre de usuario para cada uno de los integrantes, sea el mismo que el utilizado en Skype.

Cada uno de los subproyectos para el desarrollo ESMS-MI, creará un proyecto de dotProject con el nombre del sub-proyecto. En este proyecto sew debe colocar el cronograma con las actividades para el desarrollo de los módulos correspondiente, y especificando fechas de entrega de versiones a la comunidad desarrolladora del entorno.

SKYPE

En lo que respecta a las comunicaciones sincrónicas entre los integrantes de la comunidad, se ha seleccionado el uso de Skype, servicio de mensajería instantánea gratuito, adicionalmente ofrece servicios para videoconferencias, video-llamadas, llamadas de voz.

Cada uno de los integrantes de la comunidad debe crear una cuenta de Skype, la cual será pública para todos los integrantes de la comunidad.

DROPBOX Y OFFICE WORSPACE

Para compartir borradores de archivos y la edición de los mismos de forma colaborativa por parte de los integrantes de los grupos de desarrollo, se recomienda hacer uso de dos servicios Dropbox y OfficeWorkspace según las necesidades y gustos particulares de los integrantes:

Office Workspace es una herramienta en línea que permite la creación de documentos de Office colaborativamente, cuenta con un control de versiones incorporado y los archivos pueden ser editados en la suite de Office de manera local; al presionar “guardar”, los cambios son almacenados directamente en el servidor. El servicio es gratuito, pero debe contar con la suite de Office instalada (sólo para editar ya que puede ser visualizado en línea). También puede compartir archivos de otros tipos como PDF pero no podrá modificarlos.

Otra opción es utilizar Dropbox que permite sincronizar archivos en línea y a través cualquier computador conectado a internet, pero a diferencia de los discos virtuales y de otros sistemas de control de versiones, este permite tener los

archivos de manera local en una carpeta creada por el programa y al editar un documento y guardarlo, este automáticamente se actualiza en los demás computadores donde instaló el software. También se puede acceder a sus archivos desde cualquier computador o dispositivo móvil utilizando el sitio web de Dropbox. En caso de querer compartir archivos o carpetas solo se debe colocarlos en su Dropbox, e invitar a otras personas para que accedan a estos. También se puede enviar vínculos de archivos específicos a personas desde su Dropbox

A continuación, se presenta una comparación entre las principales características de estos dos servicios:

Dropbox	**Office Workspace**
Copia de los archivos en el servidor	Copia de los archivos en el servidor
Copia local de los archivos	Sin copia local
Sincroniza archivos en varios computadores	
Edición de archivo en el programa respectivo	Edición de archivo en el programa respectivo (solo Word, Excel y PowerPoint)
Gratuito	Gratuito
Compartir archivos entre varios usuarios	Compartir archivos entre varios usuarios (dos tipos de permisos: edición y visualización)
Espacio de 2GB (Gratis)	Espacio de 5GB (Gratis)
Requiere instalación de un programa	Requiere la instalación de un complemento en versiones anteriores a Office 2007
Los archivos pueden ser descargados del sitio web	Los archivos pueden ser descargados y visualizados desde el sitio web

STARUML

StarUML es una herramienta para la creación de modelos UML2.0, tiene capacidades para la generación de código, documentación e ingeniería inversa. Esta desarrollada en Delphi y es de código abierto. Actualmente, tanto la documentación de Evolución como la descripción de casos de uso del nuevo entorno están disponibles en el formato de archivo de esta herramienta. También es posible importar y exportar archivos en el formato XMI.

Los archivos de los modelos (archivos fuente) deben ser organizados y almacenados en la estructura de directorios de Subversion. La publicación de los modelos debe realizarse por medio de los Wikis de la plataforma web de la comunidad, destinados para ese propósito.

SUBVERSION

Subversión es una herramienta para el control de versiones, que es de opensource y plataforma web. Para el desarrollo del ESMS-MI se ha implementado el sistema en la dirección /var/lib/svn/ del servidor del grupo web (próximamente se tendrá acceso vía web). Este será utilizado para llevar el control de las versiones del código fuente, para los archivos fuente de los modelos UML y fuentes (.doc, .ppt, etc…) de documentos formales que se publiquen como a la comunidad desarrolladora y usuaria en formato pdf. Se puede obtener mayor información sobre subversión en el enlace http://svnbook.red-bean.com/nightly/es/index.html.