UNIVERSIDAD TECNOLÓGICA DE LA MIXTECA

"DESARROLLO DE COMPONENTES DE SOFTWARE LOCAL Y DISTRIBUIDO BAJO LA PLATAFORMA COM-SOAP, QUE ENCAPSULE LAS OPERACIONES MATRICIALES DE MATLAB"

T E S I S

PARA OBTENER EL TÍTULO DE

INGENIERO EN COMPUTACIÓN

P R E S E N T A

FELIPE DE JESÚS GARCÍA PÉREZ

DIRECTOR DE TESIS

M.C. DAVID MARTÍNEZ TORRES

HUAJUAPAN DE LEÓN, OAX., DICIEMBRE 2005

Tesis presentada el 16 de Diciembre de 2005 ante los siguientes sinodales:

M.C. Wendy Yaneth García Martínez
M.C. Everth Haydeé Rocha Trejo
M.C. Irma Guadalupe Solís Genesta

Director de Tesis:
M.C. David Martínez Torres

Dedicatoria

A mis padres, **Porfirio** e **Inés**, por haber confiado ciegamente en mí, por darme las armas para salir adelante y avanzar paso a paso, por la dicha de ver en sus ojos las esperanzas puestas en mí y saber que no los he defraudado, porque son mi mayor ejemplo de lucha y tenacidad, porque simplemente…los amo.

Los llevo siempre en mi mente y en mi corazón.

A la memoria de mis abuelos **Aniceto** y **Sabino**, porque sé, que desde la estrella en donde se encuentran estan conmigo en todo momento. Y a mi abuelas **Antonia** e **Irene**, a quienes aún las puedo abrazar y decirles cuánto las quiero y lo agradecido que estoy con ellas.

Los amo viejos lindos.

A mis hermanos, **Eduardo** y **Adriana**, porque sé que a pesar de la distacia siempre estan junto a mí, por los buenos momentos y por las lecciones que me han dado.

Gracias hermanos.

A mi sobrino **Antonio**, porque lo quiero mucho, porque me recuerda mucho mi niñez y porque estoy seguro, que a pesar de que la vida no ha sido del todo benevola con él, tiene las agallas suficientes para salir adelante.

No te imaginas cuanto te quiero Toño.

Agradezco especialmente a:

Primeramente a **Dios**, por todas las bendiciones que me ha dado, por darme una familia como la mía, y por permitirme llegar a este momento, gracias Dios, mil gracias.

A mis tios, en especial a **Martha**, **Marina**, **Octavio**, **Oscar**, **Rolando**, **Mercedes**, **Teresa**, **Armando** e **Irma**, quienes siempre se preocuparon por mí, de los que siempre tuve una palabra de aliento, un consejo y muchas, muchas muestras de cariño.

A toda mi familia, por el apoyo que me brindaron, espero no defraudarlos nunca.

A **Josefina**, por haberte encontrado, por estar conmigo, porque hemos reído y llorado juntos y porque desde hace mucho tiempo has sido una compañera excepcional, gracias Chepis.

Agradecimientos

David, gracias por ayudarme tanto y tanto para que alcanzaramos este objetivo, por darme tu confianza, por regalarme tus conocimientos, fuiste más que un asesor, eres y seguirás siendo mi amigo, gracias por permitirme conocer a un profesionista como tu. Te estoy eternamente agradecido.

Al **Dr. Emilio García Roselló**, por su invaluable aportación durante el desarrollo de esta tesis, aunque te encuentres del otro lado del océano sé que este mensaje te va a llegar. Gracias Emilio.

A mis revisores y sinodales, **Everth**, **Irma** y **Wendy**, gracias por su tiempo, su cooperación y sus valiosos comentarios.

A mis **amigos** de la Universidad, si los nombro no acabo, gracias por compartir conmigo las noches de desvelo, los buenos ratos juntos, quizá a muchos no los vuelva a ver pero tienen un lugar privilegiado dentro de mis recuerdos.

A la **Universidad Tecnológica de la Mixteca**, de la que aprendí que el trabajo, la tenacidad y la disciplina son el vestido de gala para asistir a la fiesta de la vida.

Nuevamente a **Dios**, por permitirme crecer entre todos ustedes, excelentes seres humanos, de nuevo gracias.

Índice

Lista de figuras

Lista de tablas

Prólogo

Las ciencias computacionales han venido a revolucionar el mundo actual, un mundo totalmente conectado entre sí, en el que, cualquier persona, en cualquier lugar del planeta está a tan solo un clic de distancia. Asimismo, en la actualidad, millones de personas de todas partes del mundo colaboran en este crecimiento continuo, desarrollando, manteniendo y optimizando todo tipo de sistema de cómputo, por tal motivo, el desarrollo de software se ha tornado en un pilar importante para el desarrollo tecnológico en el mundo entero.

Ante esta situación, es necesario crear herramientas, modelos y métodos capaces de facilitar el desarrollo de aplicaciones para disminuir los tiempos y complejidades propias del desarrollo de software, es decir, disminuir los costos de desarrollo de software. Por tanto, comienzan a aparecer nuevos paradigmas de programación, como pueden ser la coordinación, la programación orientada a objetos, la programación orientada a componentes, o la movilidad, que persiguen una mejora en los procesos de construcción de aplicaciones de software [42] .

Por todo lo anterior, surge la idea de crear una librería de componentes que encapsule las operaciones de un entorno propietario poderoso en su dominio, estamos hablando de MATLAB. Encapsular las operaciones de MATLAB en una librería de componentes de software reutilizable significaría ofrecer a los desarrolladores de software todo el poder que MATLAB ofrece, y a su vez, ocultar la complejidad que tiene. De esta manera, se tiene pensado crear una librería de componentes representada por una arquitectura cliente-servidor la cual contenga componentes de software dedicados a encapsular y distribuir las principales operaciones de MATLAB. Dentro de este proyecto, ésta propuesta apoya la creación de la librería de componentes, permitiendo desarrollar los componentes de software necesarios para encapsular y distribuir las operaciones matriciales que MATLAB ofrece, de esta manera, se crearán componentes dedicados a resolver operaciones matriciales y a su vez, podrán ser parte de una librería con otros componentes desarrollados en otros proyectos (en principio el módulo de redes de neuronas y cálculo integral/diferencial). Hay que señalar, que para el desarrollo de la librería de componentes, esta tesis trabaja en coordinación con otros proyectos de tesis que tienen como objetivo desarrollar componentes dedicados a distintas funcionalidades de MATLAB, por tanto, es importante mencionar que a pesar de que este proyecto forma parte de un proyecto más grande, no impide que se pueda trabajar de manera individual, de tal manera que el desarrollo de esta tesis no se encuentre supeditado al desarrollo de otras tesis y al final, los componentes que encapsulen las operaciones matriciales puedan ser probados en una aplicación educativa donde se demuestre el impacto de dichos componentes en el desarrollo de software.

De esta manera, el objetivo principal de esta tesis es desarrollar un par de componentes de software local y distribuido respectivamente, bajo la tecnología COM (Component Object Model) y SOAP (Simple Object Access Protocol), que encapsule las operaciones de matrices que MATLAB proporciona, de tal forma que puedan ser utilizados por cualquier entorno de desarrollo que soporte la tecnología COM o la tecnología SOAP. Con esto se gana reutilización en ambos sentidos: reutilización del componente software y reutilización de lo que brinda MATLAB. Para ello, se aplica la metodología orientada a objetos y la programación orientada a componentes, con la misión primordial de lograr, a través de la reutilización de dichos componentes, disminuir los costos de desarrollo de cualquier aplicación que involucre el manejo de operaciones matriciales.

Como objetivos específicos se plantean los siguientes:

- Investigación del estado del arte y de la práctica de la evolución del software, centrándonos principalmente en la Tecnología de componentes software.
- Realizar el análisis y diseño de los componentes de software con la metodología Orientada a Objetos, utilizando herramientas Rational.
- Desarrollar los componentes software orientado a objetos utilizando Delphi 7.0 como ambiente de desarrollo.
- Probar que el componente COM a desarrollar trabaje bien de forma local. Es decir, que interactúe correctamente con MATLAB.
- El componente SOAP a desarrollar trabajará en forma distribuida, de tal manera que cualquier computadora con acceso a un servidor que ofrezca los servicios de MATLAB pueda obtener dichos servicios sin la necesidad de tener instalado MATLAB en la computadora local, todo esto a través de un servicio Web.
- Desarrollar una aplicación, la cual involucrará el uso de los componentes desarrollados, de tal manera que se podrá comprobar las ventajas que el componente proporcionará a los desarrolladores que necesiten cálculos matemáticos de matrices.
- Finalmente, a lo largo del desarrollo de cada objetivo se realizará la documentación necesaria del software.

Con estos componentes se pretende apoyar a todos los desarrolladores de software que necesiten reutilizar algunos aspectos como: el análisis de datos o alguna operación matricial, sin tener una herramienta que pueda realizar dichas operaciones. Ya que hasta ahora, los propios desarrolladores tienen que codificar las operaciones matriciales que sus sistemas necesiten.

Un punto significativo es mencionar la importancia de que el componente SOAP sea distribuido. Esta característica incrementa la flexibilidad del componente, ya que no es necesario que cada computadora tenga instalado MATLAB para poder ejecutar una aplicación que ocupe algún componente, esto es, para poder acceder a los recursos de MATLAB basta con tener un servidor el cual proporcione dichos servicios y tener a su vez, una conexión hacia el servidor. Con esto se imprime un grado más de importancia al desarrollo de esta tesis.

El desarrollo de esta tesis esta supervisado y respaldado por el cuerpo académico de ingeniería de software (CASI) perteneciente a la Universidad Tecnológica de la Mixteca (UTM). El CASI tiene como objetivos la investigación y el desarrollo de las diversas discipli-

nas que conforman la ingeniería de software. La búsqueda en la realización de estos objetivos se hace a través de investigaciones realizadas en tres líneas de investigación[41] :

- Metodologías orientadas a objetos.
- Interacción humano computadora.
- Desarrollo de software educativo.

Como se ha mencionado, el desarrollo de los componentes está soportado por la metodología orientada a objetos, así como por el desarrollo de componentes de software, por lo que se enmarca principalmente dentro de la primera línea de investigación. Aunque cabe mencionar que toma un poco de cada línea de investigación.

El CASI proporciona todos los recursos que el desarrollo de esta tesis requiere, tales como: bibliografía, asesorías técnicas, software, hardware, etc.

A continuación, se muestra una estructura general de esta tesis, la distribución de los capítulos y una breve semblanza de lo que trata cada capítulo.

Estructura de la tesis

Para tener un panorama general del desarrollo de este trabajo de tesis, el documento se ha estructurado de la siguiente manera:

La tesis cuenta con un total de siete capítulos: en el primer capítulo, se presenta una introducción general de lo que persigue la tesis, la importancia de la misma y una descripción de las herramientas utilizadas.

El capítulo número dos muestra el estado del arte, se estudian las principales técnicas de análisis y diseño que van a ser la base para alcanzar un componente abierto y reutilizable así como una aplicación de calidad. En el mismo capítulo se muestran las características principales de la Tecnología de componentes, la programación orientada a objetos, la programación orientada a componentes, sus ventajas, diferencias y desventajas, es decir, este capítulo describe las bases teóricas que soportan este trabajo.

La aplicación de los conceptos teóricos se muestra en el capítulo tres y en el capítulo cuatro, en los cuales se describen el análisis y diseño tanto de los componentes como de la aplicación respectivamente, se muestra la arquitectura de software, la definición de los objetos y los modelos del sistema, sin dejar de lado la especificación de las interfaces.

En el capítulo cinco se muestra el proceso de implementación de los componentes, mientras que en el capítulo seis se muestran la implementación de la aplicación, así como también se presentan los resultados de las pruebas realizadas a los mismos.

Las conclusiones, recomendaciones y trabajos futuros se encuentran en el capítulo 7.

Por último se presentan las referencias bibliográficas utilizadas para el desarrollo de esta tesis, así como los anexos correspondientes.

Con esto, se tiene una estructura del presente trabajo mediante la cual se puede observar de manera modular el desarrollo de este proyecto, desde el análisis, diseño, implementación, pruebas y conclusiones.

1. Introducción

Desde finales de los 60's ya se impulsaba por el desarrollo masivo de componentes, como una solución a la crisis del software, pero no es hasta finales de la pasada década, con el desarrollo de sistemas abiertos, donde la tecnología orientada a objetos no ha sido suficiente para hacer frente a los requerimientos cambiantes de las aplicaciones actuales, cuando verdaderamente se ha tomado en serio la idea de desarrollar componentes software reutilizables y por consiguiente aplicaciones basadas en componentes de software reutilizables.

El paradigma basado en componentes es una alternativa poderosa para lograr un desarrollo de aplicaciones efectivo en tiempo y en recursos, por desgracia, la mayoría de los candidatos a componentes son aplicaciones propietarias de difícil integración debido a la dependencia de una interfaz de usuario y las limitaciones con las API ofrecidas[23] . Esto significa que el mercado de componentes de software es limitado, por tanto, es necesario crear componentes de software abiertos, que separen la interfaz de la funcionalidad y desarrollados bajo tecnologías estándares, de tal manera que favorezcan la integración en diferentes aplicaciones de software.

Por tal motivo, como se ha citado anteriormente, en esta tesis desarrolla componentes de software reutilizables, eligiendo en este caso a MATLAB como un entorno propietario poderoso en su dominio. Considerando la gran cantidad de *toolboxes*[1] que tiene esta herramienta, podrían desarrollarse un mayor número de componentes de este dominio, ofreciendo de esta manera una amplia reutilización y por consiguiente abarcar una mayor área de desarrollo de aplicaciones.

La capacidad de integrar este tipo de herramientas propietarias como componentes reutilizables en otras aplicaciones de software posibilitaría aprovechar su amplia funcionalidad ocultando al mismo tiempo su complejidad. El no estar ligados a utilizar una interfaz de usuario predeterminada, abre además la puerta al desarrollo de aplicaciones que integren diferentes componentes de software educativo bajo una interfaz de usuario diseñada a la medida de necesidades y dominios concretos[23] .

[1]Las toolboxes en MATLAB, son librerías de funciones dedicadas a operaciones matemáticas en un campo específico.

1.1. Conceptos previos

El definir conceptos previos tiene la finalidad de ayudar a entender de manera amplia los temas tratados en esta tesis.

El principal concepto a definir, debido a la naturaleza de esta tesis, es el de componente de software. Samuel Garrido define componente de software como bloques reutilizables de construcción de sistemas de software. Encapsulan aplicaciones o servicios técnicos con sentido semántico. Difieren de otros tipos de módulos reutilizables en que pueden modificarse, al tiempo de diseño, en sus ejecutables binarios, mientras los demás lo hacen en su nivel de programa fuente. Los componentes restringen acceso vía una o más interfaces públicas que definen propiedades, métodos y eventos que permiten comunicación[33] [39] [32] .

Definir otros conceptos tales como sistemas de cómputo o software resultaría redundante, ya que actualmente dichos conceptos son bien conocidos y tratados a menudo, por lo que a continuación se muestra un panorama general de esta tesis.

1.2. MATLAB y las operaciones matriciales

MATLAB nace como una solución a la necesidad de mejores y más poderosas herramientas de cálculo para resolver problemas de cálculo complejos en los que es necesario aprovechar las amplias capacidades de proceso de datos de grandes computadoras[26] .

MATLAB es el nombre abreviado de “MATrix LABoratory”, MATLAB es un software propietario, de carácter científico, orientado especialmente al ámbito de la ingeniería que se ha convertido desde su aparición en un estándar “de facto” en esta disciplina. Además de su uso profesional, tiene una elevada presencia en el ámbito académico, colocándose como una herramienta ampliamente difundida a nivel educativo en materias y carreras relacionadas con disciplinas tecnológicas. Buena prueba de ello es su presencia en foros científicos relacionados con esta temática[20] [43]

Se decidió utilizar MATLAB debido a su amplia capacidad para efectuar cálculos matriciales, los cuales son desaprovechados por los desarrolladores que desconocen las ventajas que ofrece, por otro lado, las operaciones matriciales vienen incluidas dentro de la MATLAB C Math Library, la construcción y desarrollo de aplicaciones utilizando esta librería es un proceso de amplias perspectivas una vez que se tiene un dominio adecuado del mismo sistema. El producto está dividido en dos categorías (como librerías objeto): la librería built-in library, contiene versiones de las funciones de MATLAB en lenguaje C del tipo numérico, lógico y utilidades. Por otra parte la librería de toolboxes (toolbox library), contiene versiones compiladas de la mayoría de archivos M[2] de MATLAB para cálculo numérico, análisis de datos y funciones de acceso a archivos y matrices[40] . La decisión de orientar el componente hacia operaciones matriciales se obtuvo gracias al apoyo de los profesores del Instituto de Física y Matemáticas de la UTM, los cuales indicaron, en su mayoría, que las operaciones matriciales son ocupadas en muchos algoritmos para resolver problemas matemáticos comunes.

A continuación, se especifican las herramientas que se utilizaron en el desarrollo de esta tesis.

[2] Los archivos M son archivos tipo texto que contienen programas en código MATLAB.

1.3. Herramientas a utilizar en el desarrollo de esta tesis

Como se ha visto, una herramienta fundamental para este proyecto es MATLAB, sin embargo, no es la única herramienta involucrada en esta tesis.

Por principio de cuentas, se debe especificar que MATLAB únicamente es el ejecutor de las operaciones que el desarrollador de software invoque a través del componente de software, es decir, MATLAB va a trabajar en segundo plano, sin que el usuario detecte la existencia de MATLAB en la aplicación. Por tal motivo se necesita un lenguaje de programación en el cual se desarrolle el componente de software, y para ello se eligió Delphi en su versión 7.0 como entorno de desarrollo tanto del componente como de la aplicación. Delphi es un producto de Borland, Delphi es un ambiente de desarrollo rápido de aplicaciones (RAD) para el lenguaje Object Pascal. Los sistemas RAD están orientados a facilitar la productividad en el desarrollo de software.

Todo lo anterior se refiere a las herramientas que se utilizan durante la codificación del componente y de la aplicación. Sin embargo, al tratarse de un componente y de una aplicación orientada a objetos es necesario tener herramientas para el modelado del sistema. Por tal motivo se utiliza la suite de Rational Rose para modelar tanto el componente como la aplicación. Esto con la finalidad de sacarle el máximo provecho a la reutilización, es decir, no solamente reutilizar código, sino ir mas allá, buscar reutilizar el análisis, diseño, implementación y documentación. Por último para probar que realmente el componente funciona de manera distribuida, se utiliza el Internet Information Server (IIS), el cual es el servidor Web que Windows proporciona, aunque podría funcionar con cualquier servidor Web.

En el siguiente capítulo, se va a profundizar en los fundamentos teóricos que regirán el desarrollo de esta tesis, tales como tecnologías de componentes, arquitecturas de software, patrones de diseño, análisis y diseño orientado a objetos, etc.

2. Marco Teórico

Tener bases sólidas para realizar cualquier actividad cotidiana es el precedente del éxito o fracaso para dicha actividad. Por ejemplo, en la construcción de un edificio, la edificación de los cimientos es una parte esencial, ya que de ellos depende en gran medida la magnitud del edificio, los materiales que conforman el edificio y la posible modificación del mismo. Por tal motivo, existen ciencias especializadas en este ámbito. Algo muy similar ocurre en el desarrollo de software, en donde, la costeabilidad del software depende en gran medida del análisis y el diseño, y que a su vez, estén soportados por alguna tecnología probada, de tal manera que los sistemas sean robustos y de fácil integración y manutención. Prueba de esto, son la cantidad de sistemas desarrollados anteriormente, aquellos sistemas en los que el análisis y el diseño eran no más que una tarea sin sentido y considerada como pérdida de tiempo, aquellos sistemas en los que la documentación del proceso no era considerada. Estos procesos de desarrollo de software originaron sistemas limitados, con un ciclo de vida muy corto y que generaban más problemas de los que podían resolver, generando la conocida "crisis del software". Ante este panorama, fue necesario crear una nueva disciplina que pudiera brindar una guía a los desarrolladores de software, nace la ingeniería de software.

La ingeniería de software es relativamente joven, la noción de esta ingeniería fue propuesta inicialmente en 1968 en una conferencia de la Organización del Tratado del Atlántico Norte (OTAN), en la cual se discutió la problemática de la crisis del software[36] , dado que en ese periodo, la evolución del hardware permitió la construcción de una nueva generación de computadoras, permitiendo a su vez, desarrollar software que hasta esa fecha parecía imposible, y como consecuencia, los nuevos sistemas de cómputo fueron cada vez más complejos y poderosos, requiriendo nuevas técnicas y métodos para poder controlar la complejidad inherente en los sistemas de cómputo.

La ingeniería de software según varios autores, es una disciplina o área de la informática o Ciencias de la Computación, que ofrece métodos, técnicas, herramientas y documentación, para desarrollar y mantener software de calidad que resuelvan problemas de todo tipo, y que además sea costeable[29] [36] [7] [4] .

Actualmente, la ingeniería de software tiene bases sólidas mediante las cuales puede regir el desarrollo de sistemas de cómputo, y se torna cada vez más importante por el hecho de que los programas de software están omnipresentes y los usuarios de software lo ven como un hecho tecnológico de la vida. En muchos ejemplos, las personas dejan su trabajo, bienestar, seguridad, entretenimiento, decisiones y su propio quehacer diario, en manos de software informático [29] .

En este capítulo, abordaremos técnicas importantes del desarrollo de software, que apoyarán el futuro análisis, diseño e implementación de los componentes y la aplicación software.

2.1. La tecnología orientada a objetos

El paradigma orientado a objetos, es uno de los más completos, y actualmente más utilizado que la programación estructurada. Este paradigma hace énfasis en el análisis y el diseño con el principal objetivo de lograr en mayor medida, reutilización tanto de código como del proceso de desarrollo.

La tecnología orientada a objetos permite de manera eficaz la reutilización, y la reutilización de componentes de software, lleva a un desarrollo de software más rápido y a programas de mejor calidad. El software orientado a objetos es más fácil de mantener debido a que su estructura es inherentemente poco acoplada, esto lleva a menores efectos colaterales cuando es necesario hacer cambios, además los sistemas orientados a objetos son más fáciles de adaptar y son más escalables [29] .

El paradigma orientado a objeto se basa en el concepto de objeto. Por principio de cuentas, se tiene que definir qué es un objeto. Un objeto es aquello que tiene un estado (propiedades más valores), comportamiento (acciones y reacción a mensajes) e identidad (propiedad que los distingue de los demás objetos). Deitel [16] menciona que los objetos, son en esencia, componentes de software reutilizables que simulan elementos reales, y que mediante los objetos se pueden crear sistemas de mejor calidad, más fáciles de mantener y de ampliar ya que el ser humano por naturaleza piensa en términos de objetos, dado que tiene una enorme capacidad de abstracción. Por su parte, McKim [28] menciona que un objeto es la implementación de un tipo abstracto de datos que encapsula las operaciones y sus datos.

Los beneficios de la tecnología orientada a objetos se fortalecen si se usa antes y durante el proceso de ingeniería de software. Considerar esta tecnología debe hacer sentir su impacto en todo el proceso de ingeniería de software. Los ingenieros del software y sus directores deben considerar tales elementos como: el análisis de requisitos orientado a objetos (AROO), el diseño orientado a objeto (DOO), el análisis del dominio orientado a objetos (ADOO), sistemas de gestión de bases de datos orientadas a objetos (SGBDO) y la ingeniería del software orientada a objetos asistida por computadora (ISOOAC) [5] .

Según Pressman [29] , los principales beneficios que ofrece una arquitectura orientada a objetos son los siguientes:

- Los detalles de implementación interna de datos y procedimientos están ocultos al mundo exterior. Esto reduce la propagación de los efectos colaterales cuando ocurren cambios.
- La estructura de datos y las operaciones que las manipulan están mezcladas en una entidad sencilla: la clase. Esto facilita la reutilización de componentes.
- Las interfaces entre objetos encapsulados están simplificadas. Un objeto que envía un mensaje no tiene que preocuparse de los detalles de las estructuras de datos internas en el objeto receptor, lo que simplifica la interacción y hace que el acoplamiento del sistema tienda a reducirse.

Sommerville [36] considera que el proceso de desarrollo orientado a objetos consiste en lo siguiente:

- **El análisis orientado a objetos** comprende el desarrollo de un modelo orientado a objetos del dominio de aplicación. Los objetos identificados reflejan las entidades y operaciones que se asocian con el problema a resolver.
- **El diseño orientado a objetos** comprende el desarrollo de un modelo orientado a objetos de un sistema de software para implementar los requerimientos identificados. Los objetos en un diseño orientado a objetos están relacionados con la solución del problema por resolver. El diseño tiene que agregar nuevos objetos para transformar los objetos del problema e implementar la solución.
- **La programación orientada a objetos** se refiere a llevar a cabo el diseño de software utilizando un lenguaje de programación orientado a objetos, el cual permite la implementación directa de los objetos y suministra los recursos para definir las clases de objetos.

De esta manera se debe revisar el proceso de análisis y diseño orientado a objetos, basado en algún método orientado a objetos, para tener una visión general de todo lo que involucra utilizar un paradigma orientado a objetos.

2.1.1. La programación orientada a objetos

La programación orientada a objetos, es una técnica para programar, un paradigma para escribir "buenos" programas para un conjunto de problemas [37] . Para Greiff [19] la programación orientada a objetos es una forma de pensar, una filosofía, de la cual surge una nueva cultura que incorpora técnicas y metodologías diferentes, la programación orientada a objetos como paradigma es una postura ontológica: el universo computacional está poblado por objetos, cada uno responsabilizándose por sí mismo, y comunicándose con los demás por medio de mensajes. Desde el punto de vista computacional, la programación orientada a objetos es un método de implementación en el cuál los programas son organizados como grupos cooperativos de objetos, cada uno de los cuales representa una instancia de alguna clase, y estas clases son miembros de una jerarquía de clases unidas vía relaciones de herencia.

Existen algunas características principales en la programación orientada a objetos [30] :

- *Abstracción:* La abstracción es un mecanismo que permite al diseñador concentrarse en los detalles esenciales de un componente de programa (ya sean datos o procesos), prestando poca atención a los detalles de bajo nivel.
- *Encapsulación:* Para los sistemas orientados a objetos, la encapsulación engloba las responsabilidades de una clase, incluyendo sus atributos y operaciones, y los estados de la clase, definidos por los valores de atributos específicos.
- *Polimorfismo:* El polimorfismo, es la propiedad que poseen algunas operaciones de tener un comportamiento diferente dependiendo del objeto sobre que se aplica, por ejemplo, en el caso del operador +, realiza la suma de dos números, mediante el polimorfismo se puede lograr que el mismo operador realice la suma de cadenas.
- *Herencia:* La herencia es un mecanismo que habilita las responsabilidades de un objeto para propagarse a otros objetos. La herencia ocurre a todos los niveles de una jerarquía de clases.

Considerando lo anterior, se puede decir que los principales conceptos de la programación orientada a objetos son el objeto, la clase, los atributos, las operaciones, la abstracción, el

encapsulamiento, el polimorfismo y la herencia. Donde el objetivo del análisis orientado a objetos es descubrir las clases y los objetos, mientras que en el diseño orientado a objeto se plantean las abstracciones y mecanismos que proporcionan el comportamiento.

2.1.2. UML

Para el desarrollo de software OO, es necesario aplicar métodos orientados a objetos, que faciliten el análisis y diseño del sistema. Uno de los métodos más completos y robustos es UML (Unified Model Languaje), el cual viene siendo la unificación de tres métodos pioneros en esta área: el método Booch [8] del Dr. Grady Booch, el método OMT [31] (Object-Oriented Modeling Technique) del Dr. James Rumbaugh, y el método OOSE [27] (Object Oriented Software Engineering) del Dr. Ivar Jacobson. UML es la más reciente y poderosa notación o metodología orientada a objetos para el modelado integral de sistemas de software, por esta razón, será utilizado para el desarrollo de este trabajo de tesis.

UML es un lenguaje que permite modelar, construir y documentar los elementos que forman un sistema software orientado a objetos [9] [19] [24] . En la Figura 2.1 se puede ver cuál ha sido la evolución de UML hasta la creación de UML1.1.

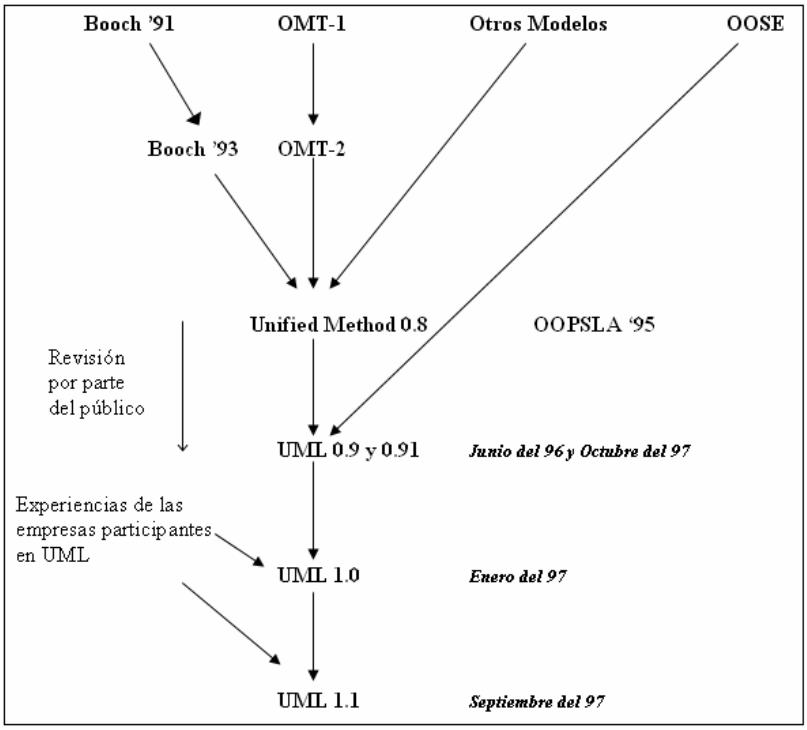


Figura 2.1. Historia de UML[24] .

Con la creación de UML, se pretende alcanzar los siguientes objetivos [18] :

- Representar sistemas completos con conceptos de objetos.
- Establecer una relación explícita entre los conceptos y los "artefactos" ejecutables.
- Tener en cuenta los factores de escala inherentes a los sistemas complejos y críticos.

- Crear un lenguaje de modelado utilizable tanto por humanos como por máquinas.

Se ha visto, que en el transcurso del tiempo, se ha tratado de estandarizar los métodos de análisis y diseño de software en pro de un mejor desarrollo del mismo. En este proceso, la arquitectura de software es la encargada de lograr dicha estandarización, por lo tanto, es importante destacar el trabajo en esta área.

2.2. Arquitectura de software

Desde que el primer programa fue dividido en módulos, los sistemas de software han tenido arquitectura, es decir, las arquitecturas de software han estado implícitas.

La importancia de la arquitectura de software es muy simple, la arquitectura de software de un programa de cómputo es la estructura de las estructuras del sistema, la cual comprende los componentes del software, las propiedades de esos componentes visibles externamente y las relaciones entre ellos.

La mayoría de autores [42] [3] [35] [13] concluyen de manera general que, una arquitectura de software es una herramienta de alto nivel para cubrir distintos objetivos, entre los que se destacan:

- Comprender y manejar las estructuras de las aplicaciones complejas.
- Reutilizar dicha estructura (o partes de ella) para resolver problemas similares.
- Planificar la evolución de la aplicación, identificando las partes mutables e inmutables de la misma, así como los costes de los posibles cambios.
- Analizar la corrección de la aplicación y su grado de cumplimiento respecto a los requerimientos iniciales (prestaciones o fiabilidad).
- Permitir el estudio de alguna propiedad especifica del dominio.

Por último, Shaw [35] destaca los principales estilos arquitecturales definidos hasta el momento:

- Tuberías y filtros.
- Orientada a objetos.
- Basada en eventos.
- De dominio específico.
- Basada en capas.
- Repositorios.
- Basada en reglas.
- Control de procesos.
- Distribuida.
- Basada en estados de transición.
- Heterogénea.

En esta tesis, la principal arquitectura que se ocupa es la arquitectura basada en objetos, en la que los datos son representados mediante objetos y sus asociaciones primitivas son encapsuladas en tipos abstractos de datos. Dos importantes aspectos de este estilo son: 1. Que los

objetos son responsables de perseverar la integridad de la representación. 2. Que la representación no es visible a otros objetos.

Cabe recordar que el objetivo principal de esta tesis, es desarrollar dos componentes de software, es lógico pensar, que al tratarse del desarrollo de componentes de software, se debería ocupar una arquitectura orientada a componentes. Por desgracia, hasta el momento el desarrollo orientado a componentes, a diferencia del desarrollo orientado a objetos, solo tiene como base la programación orientada a componentes, es decir, dentro de este paradigma, todavía no hay métodos de análisis y diseño orientado a componentes, por lo que generalmente, al desarrollar componentes de software, es necesario combinar el paradigma orientado a objetos con la programación orientada a componentes.

2.3. Patrones de diseño

El diseño de software orientado a objetos es difícil, y el diseño de software reusable orientado a objetos lo es aún más. Los patrones de diseño son básicamente una descripción del problema y la esencia de su solución, de tal manera que los patrones de diseño puedan ser reutilizados en diferentes casos.

Los patrones de diseño se derivaron de las ideas de Christopher Alexander, quien sugirió que existían ciertos patrones del diseño de edificios que eran comunes e inherentemente interesante y efectivos[36] .

Un grupo de expertos en el área (Erich Gamma, Richard Helm, Ralph Johnson y John Vlissdes), proponen el diseño de patrones como un mecanismo para expresar el diseño orientado a objetos. Esto es, mediante los patrones de diseño se pueden identificar, nombrar y abstraer los temas más comunes del diseño orientado a objetos. Los patrones de diseño pueden a su vez, jugar diversos roles dentro del proceso de desarrollo orientado a objetos ya que proveen un vocabulario común para el diseño, reducen la complejidad del sistema nombrando y definiendo abstracciones, fomentan las bases para construir software reusable, es decir, los patrones de diseño se pueden considerar micro arquitecturas que contribuyen en una arquitectura completa de un sistema.

Gamma y sus colaboradores [22] mencionan que los patrones de diseño consisten de tres partes esenciales:

- Una descripción abstracta de la clase o el objeto y su estructura. La descripción es abstracta porque concierne a un diseño abstracto, no a un diseño en particular.
- El tópico en el diseño del sistema diseccionado por la estructura abstracta. Esto determina las circunstancias en el que el diseño de patrones es aplicable.
- Un registro de las consecuencias de aplicar la estructura abstracta a una arquitectura de sistema. Esto se utiliza para ayudar a los diseñadores a comprender cuándo se puede o no aplicar en una situación en particular.

De la misma manera, Gamma [21] menciona que los patrones de diseño tienen 4 elementos esenciales.

- El *nombre del patrón*, es el manejador que se va a usar para describir un problema de diseño. Nombrar los patrones automáticamente incrementa el vocabulario de diseño.

- El *problema* describe cuando aplicar el patrón. Este explica el problema y su contexto.
- La *solución* describe los elementos que conforman el diseño, sus relaciones, responsabilidades y colaboraciones. La solución no describe un diseño concreto o particular, debido a que los patrones de diseño son plantillas que pueden ser ocupadas en diversas situaciones.
- Las *consecuencias* son el resultado y cambios de aplicar los patrones de diseño. Las consecuencias de los patrones de diseño incluyen el impacto en la flexibilidad del sistema, la extensibilidad o la portabilidad.

Zimmer [45] menciona que la idea principal detrás de los patrones de diseño es apoyar la reutilización de la información de diseño, permitiendo a los desarrolladores comunicarse de manera más efectiva, por lo que nuevos patrones de diseño están siendo descubiertos, descritos y aplicados en diversos grupos de investigación.

En esta tesis, se ha optado por ocupar el patrón de diseño llamado Modelo Vista Controlador (MVC), este patrón suele ser utilizado para el desarrollo de software con interfaz hombre-máquina, generalmente en aplicaciones basadas en ventanas (Windows y XWindows) y aplicaciones Web.

Esta arquitectura consta de tres componentes, correspondientes a las siglas que le dan nombre. La primer componente, el Modelo, incluye los datos propios de la aplicación a los que ésta accede y que modifica mediante sus algoritmos internos o mediante el proceso de interacción, así como los algoritmos citados. La segunda componente, la Vista, contiene los datos y algoritmos correspondientes a la interfaz en sí, necesarios para mostrar y actualizar la componente gráfica de la aplicación. La tercera componente, el Controlador, contiene la información y los mecanismos necesarios para asociar una acción a cada evento [1] .

Para esta investigación, desde el punto de vista de MVC, las librerías forman el modelo lógico, los componentes los elementos de vista y los botones y eventos del ratón los controladores.

En base a lo anterior, se puede concluir que los patrones de diseño son una herramienta útil dentro del diseño orientado a objetos, lo que se busca es utilizar los patrones de diseño para encontrar una mejor solución, en calidad y tiempo, al diseño orientado a objetos.

Hasta el momento, se ha revisado el desarrollo de software, principalmente el paradigma orientado a objetos, el cual ha aumentado el grado de reutilización con respecto a los anteriores paradigmas. Siendo la reutilización la mayor promesa del desarrollo de software, es necesario definir los diferentes métodos y técnicas de reutilización, para conocer y distinguir las cualidades y defectos de las mismas.

2.4. Reutilización de software

A través del tiempo se ha buscado constantemente la manera de reducir los costos de desarrollo y mantenimiento, así como aumentar la adaptabilidad y flexibilidad del software, de esta manera se ha aumentado el nivel de abstracción en cuanto a programación se refiere, pasando de lenguaje binario a ensamblador, para luego pasar a los lenguajes de alto nivel. Sin embargo, conforme pasaba el tiempo, los desarrolladores se dieron cuenta de que el ir aumentando el nivel de abstracción de la programación no era la solución a todos los problemas relacionados con el software, se dieron cuenta de que existía un problema mayor al esperado: el

mantenimiento del software. Este problema resultaba enormemente costoso, debido a que al ir incrementando la complejidad y el número de sistemas activos, resultaba imposible detectar el origen de las fallas, debido a que los códigos eran difíciles de entender. Ante esto se crearon paradigmas para tratar de resolver este monstruo que acechaba con terminar con una naciente industria, y vinieron paradigmas tales como la programación modular, la cual trataba de estructurar el software. Sin embargo, los resultados fueron visibles hasta la aparición de la programación orientada a objetos. Esta metodología influyó, principalmente, en el análisis y diseño de software, el cual hasta ese momento pasaba a segundo término. Con la aparición de este paradigma apareció a su vez un concepto muy deseado: la reutilización. El concepto de objeto prometía proporcionar la reusabilidad deseada. Por desgracia, la programación orientada a objetos no fue lo suficientemente eficaz como se pensaba, y ante esto surgió un nuevo paradigma: la programación orientada a componentes. Frente a las clases o prototipos de objetos, los componentes han de ofrecer ciertos requisitos, tales como la posibilidad de utilizarlos con independencia de otros elementos [25] .

La reutilización por sí sola posee un conjunto de ventajas:

- Productividad: hay que desarrollar menos elementos, por tanto se aumenta la capacidad productiva del proceso.
- Calidad: cuando un elemento es reutilizado sucesivas veces, es más probable que se detecten y corrijan los defectos y deficiencias que pueda tener.
- Rendimiento: incluso en ausencia de defectos, es más probable que se siga trabajando sobre un elemento que sea muy reutilizado a fin de mejorarlo y hacerlo más eficiente.
- Fiabilidad: al reducirse los errores propios de los componentes utilizados, aumenta la fiabilidad de los sistemas construidos a partir de ellos.
- Economía: la reutilización supone una reducción en el esfuerzo de desarrollo y por tanto en el tiempo. Se reduce la necesidad de documentación, los costes de mantenimiento, los de aprendizaje y el tamaño de los equipos de desarrollo.

La palabra reutilización nació con la crisis del software, diversos autores [32] , todos coinciden en que el reuso es el proceso de crear sistemas de software desde software existente, en vez de construir el software desde la nada.

Con todo esto, se puede observar que la reutilización permite reducir los costos de desarrollo y mantenimiento de los sistemas informáticos, ofreciendo grandes ventajas, aunque, dicho sea de paso, se agrega el trabajo de tener un buen análisis y diseño que garantice componentes de calidad capaces de ser reutilizados.

2.4.1. Técnicas de reutilización

La idea es que el reuso de software llegue a estandarizarse, por tal motivo, se han creado diversas técnicas o aproximaciones que pueden ser usadas en el reuso de software. En este apartado se van a mencionar dos importantes técnicas de reuso: reuso composicional y reuso generativo. Antes de explicar estas técnicas de reuso, cabe mencionar que la abstracción es esencial en cualquier técnica de reuso, sin la abstracción, los desarrolladores se pueden ver forzados a investigar a través de una colección de artefactos reusables, tratando de encontrar qué hace cada artefacto, cuándo puede ser reusado y cómo puede ser reusado. Todas las abstracciones de software tienen dos niveles: nivel de especificación (qué es lo que se va a hacer con la abstracción) y el nivel de realización (cómo se hace).

- **Reuso composicional:** El reuso composicional está basado en la idea de componentes reusables que, idealmente, permanezcan inmodificables en su reuso. Los componentes de alto nivel, o más complejos, se componen por la combinación de componentes de bajo nivel o componentes simples. El reuso composicional está basado en depósitos de componentes o en principios de organización y composición como la arquitectura de tuberías o a la construcción de sistemas de software orientado a objetos.
- **Reuso generativo:** El reuso generativo está basado en el reuso de una generación de procesos en vez de un reuso de componentes [6] . Los componentes que son reusados no son concretos y no están contenidos propiamente como en el reuso composicional. En generadores de aplicaciones el reuso generativo se centra en reutilizar aplicaciones capaces a su vez de generar otras aplicaciones en base a un patrón preestablecido, llamado patrón generativo[32] [6] . Ejemplos típicos de este reuso son los analizadores léxicos, analizadores sintácticos y compiladores, generadores de sistemas expertos etc.

De acuerdo a lo anterior, en este trabajo se utilizó el reuso composicional, el cual es apoyado en gran medida por el paradigma orientado a componentes, razón por la cual se presenta brevemente un estudio sobre tal paradigma.

2.5. Programación Orientada a Componentes

El surgimiento de la programación orientada a objetos nació debido a la necesidad de crear más y mejores sistemas de cómputo, y para ello, se consideró que la piedra angular del desarrollo de software era la reutilización del mismo. El paradigma orientado a objetos prometía ser el paradigma mediante el cual, la reutilización pasara de ser un proceso complejo a ser toda una realidad. Por desgracia, la programación orientada a objetos no fue lo que se esperaba, dio muy buenos resultados y fue un avance muy importante para el desarrollo de software, sin embargo, en cuanto al poder de reutilización, dejó mucho que desear, los objetos no son tan reutilizables como se pensaba. Por tal motivo, se buscó un nuevo paradigma de desarrollo, saltando a la vista que el desarrollo de componentes podía ser la clave para el desarrollo de software basado en la reutilización.

La programación basada en componentes es aquella que se basa en la implementación de sistemas utilizando componentes previamente programados y probados. Este paradigma de programación aparece como una variante natural de la programación orientada a objetos para los sistemas abiertos, en donde la programación orientada a objetos presenta algunas limitaciones; por ejemplo, la orientada a objetos no permite expresar claramente la distinción entre los aspectos computacionales y meramente composicionales de la aplicación, no define una unidad concreta de composición independiente de las aplicaciones (los objetos no lo son, claramente), y define interfaces de muy bajo nivel como para que sirvan de contratos entre las distintas partes que deseen reutilizar objetos [36] .

Ante el avance de la programación orientada a componentes, nace la ingeniería de software basada en componentes, esta ha sido reconocida como una subdisciplina de la ingeniería de software, teniendo tres objetivos principales [14] :

- Desarrollar sistemas a partir de componentes ya construidos.
- Desarrollar componentes como entidades reusables.

- Mantener y evolucionar el sistema a partir de la personalización y el reemplazo de los componentes.

Por su parte, Coulson et al [15] definen que los principales objetivos de la programación orientada a componentes son: *i*) promover un alto nivel de abstracción en el diseño, implementación, desempeño y administración de software, *ii*) facilitar la configuración flexible (y potencialmente, la reconfiguración en tiempo de ejecución) y *iii*) fortalecer la comunidad que fomenta la reutilización de software.

La ingeniería de software basada en componentes, es uno de los paradigmas de desarrollo más populares en el presente. Prueba de ello, es la expansión de las plataformas como Enterprise Java Beans, Component Object Model o Corba Component Model. Mediante la programación orientada a componentes, se busca que el desarrollo de sistemas, se vuelva un proceso de ensamblar módulos de software independientes y reusables, llamados componentes. Un componente es una unidad de software que especifica un conjunto de interfaces y un conjunto de requerimientos. El uso de componentes no solo reduce los costos de software, sino también promueve la flexibilidad, fiabilidad y reusabilidad de aplicaciones, por medio de la reutilización de componentes que han sido verificados y validados [14] .

Sametinger [32] menciona que, al ser los componentes artefactos los podemos identificar claramente en los sistemas de software, la intensión primaria en la reutilización de componentes, es que se puedan tomar componentes e integrarse dentro de sistemas de software, lo que, involucraría la disminución en los costos de desarrollo.

Dentro de este paradigma, generalmente es más conveniente tratar a los componentes como cajas negras (black-box components), es decir, como entidades encapsuladas que tienen una interfaz de servicios bien definida y una implementación oculta, a la que los clientes de los citados servicios no deben tener acceso, porque se considera generalmente que la reutilización de código es menos ventajosa que la reutilización de componentes [39] .

Se puede concluir, que la programación orientada a componentes es un paradigma naciente, el cual muestra signos de un pronto avance, y que quizá adolece de métodos formales de análisis y diseño, pero que sin duda, es hasta el momento la mejor opción para el fortalecimiento de la reutilización de software.

2.6. Tecnología de componentes

Definir la tecnología es un proceso vital en el desarrollo de los sistemas, la tecnología que se elija va a determinar muchos de los alcances y limitaciones de nuestro desarrollo.

Este apartado está dedicado a un breve análisis de los principales modelos de componentes que se encuentran actualmente en el mercado computacional. Se discutirán cinco modelos de componentes: CORBA de OMG, SOM de IBM, JavaBeans de Sun Microsystems y se hará especial énfasis sobre los modelos de componentes COM de Microsoft y SOAP.

2.6.1. CORBA

El consorcio denominado OMG (*Object Management Group*) se creó en 1989 con el propósito de definir estándares que permitieran la interoperabilidad y portabilidad de aplicaciones orientadas a objetos y distribuidas. El OMG no produce software, solo especificaciones, estas especificaciones nacen a través de los miembros del OMG, estos miembros pertenecen principalmente a las principales empresas de software lo que permite la gran riqueza de

ideas y puntos de vista así como la generación de estándares previamente acordada por varias partes. Esto a su vez tiene algunos inconvenientes, principalmente porque los estándares tardan mucho en fraguarse y al tratar de encontrar un consenso entre un amplio espectro de empresas se producen estándares que no suelen satisfacer a todo el mundo y pocas son las implementaciones con éxito.

El OMG ha definido la OMA (*Object Management Architecture*), un conjunto de estándares que definen, a alto nivel, las facilidades necesarias para realizar aplicaciones distribuidas, el ORB (*Object Request Broker*), un mecanismo que permite comunicarse a los objetos haciendo transparentes los problemas de heterogeneidad, dispersión y activación de los sistemas abiertos y distribuidos. De esta manera, la especificación de CORBA (*Common Object Request Broker Architecture*), es básicamente una descripción concreta de las interfaces y servicios que deben proporcionar los desarrolladores de ORBs. Las principales implementaciones de CORBA son: Orbix, Object Broker, Visibroker y Component Broker [42] [34] .

2.6.2. SOM

El modelo de objetos que propone IBM se denomina SOM (*System Object Model*), define un estándar binario de componentes, y por tanto, independiente del lenguaje en el que estén implementados. SOM implementa un super-conjunto de la funcionalidad de CORBA y ofrece un enfoque más homogéneo, coherente y completo que CORBA para el desarrollo de aplicaciones distribuidas desde el punto de vista de la tecnología de objetos. De esta forma, ofrece a los programadores una colección de servicios muy interesantes que forman parte del modelo de manera natural, y por tanto implementados muy eficientemente y perfectamente integrados[42] .

2.6.3. JavaBeans

JavaBeans es un API estándar sobre Java que define el modelo de componentes de Sun. A los componentes se les denomina *Beans*, definidos como “componentes software reutilizables que pueden ser manipulados de forma visual por herramientas de desarrollo de aplicaciones”. Las herramientas a las que hace referencia esa definición pueden ser desde generadores de páginas Web, editores de documentos compuestos o aplicaciones visuales, hasta entornos de desarrollo para aplicaciones de cálculo distribuido. Las principales características de los beans son[34] [42] :

- La interfaz de cualquier bean está compuesta por una serie de atributos, una serie de métodos y una serie de eventos.
- Todo bean soporta inspección.
- Todo bean soporta particularización.
- Los beans soportan persistencia.

2.7. COM y SOAP

La elección de ocupar la tecnología COM y la tecnología SOAP no son arbitrarias, lo que se busca es que los componentes desarrollados puedan ser ocupados desde diversos lenguajes de programación y desde cualquier plataforma. Con base en lo anterior, se puede decir que COM y SOAP nos permiten llegar a ese objetivo, ya que la mayoría de lenguajes de pro-

gramación actuales permiten la utilización de componentes COM, además, en caso que no lo permitiesen, queda la opción de que permitan la ejecución de servicios Web, es decir, la baraja de posibilidades resulta muy amplia.

2.7.1. Component Object Model

En los apartados anteriores, se ha visto que el desarrollo de aplicaciones se ha orientado hacia el desarrollo basado en componentes, de esta manera, Microsoft propuso a principios de 1990 el desarrollo de una nueva tecnología, Component Object Model, y que esta nueva tecnología fuera la base de su nuevo sistema operativo, Windows 95, ya que en la versión anterior (Windows 3.0), el término de componentes se observaba como simples funciones.

COM es una arquitectura de componentes que permite a las aplicaciones y sistemas ser construidos a partir de componentes proporcionados por distintos vendedores de software. COM es una arquitectura base para los servicios software de más alto nivel, estos servicios proporcionan claramente diferentes funcionalidades al usuario, de cualquier manera, requieren de un mecanismo que permita, a componentes binarios, de distintos vendedores, conectarse y comunicarse de una manera bien definida. Este mecanismo es aportado por COM, una arquitectura de componentes que [44] :

- Define un estándar binario para interoperabilidad de componentes.
- Es independiente del lenguaje de programación.
- Está disponible en múltiples plataformas (Windows, Macintosh, Unix).
- Proporciona una evolución robusta para aplicaciones y sistemas basados en componentes.
- Es extensible.

Además, COM proporciona mecanismos para:

- Comunicación entre componentes, incluso entre procesos y distintas redes.
- Gestión de memoria compartida entre componentes.
- Informe de errores y status.
- Carga dinámica de componentes.

Esto es, los componentes desarrollados mediante COM, proveen una manera mediante la cual, dos objetos en diferentes espacios de objetos o redes, puedan comunicar sus métodos entre sí. COM forza al sistema operativo a actuar como un registro central para los objetos. El sistema operativo toma la responsabilidad de crear objetos cuando son requeridos, eliminándolos en caso contrario y manejando las comunicaciones entre ellos [38] .

Una de las grandes ventajas de la tecnología COM es su capacidad de adaptarse al cambio, ya que, si un objeto COM cambia a una nueva versión, la aplicación que use el objeto no necesita volver a ser compilada.

Como se ha observado, la tecnología COM hasta hoy día es muy utilizada, ofrece amplias ventajas y está disponible para el desarrollo en diversos lenguajes (C, C++, Java, J++, Visual Basic, Delphi, etc.), lo que lo coloca en una posición excepcional para que el componente COM desarrollado en esta tesis pueda ser ampliamente utilizado. Para el caso de la distribución (que COM no la realiza), se cuenta con un protocolo igualmente estandarizado como lo es SOAP.

2.7.2. Simple Object Access Protocol

La función de SOAP en este proyecto es sencilla, lograr lo que COM no puede, es decir, distribuir el componente COM. Al igual que COM define un estándar de compatibilidad binario, SOAP define un estándar de compatibilidad entre objetos basados en XML, con las ventajas que ello conlleva: independencia del lenguaje e independencia de la plataforma.

SOAP no define por si solo una semántica de aplicación como un modelo de programación o implementación de una semántica específica, más bien, define un simple mecanismo para expresar aplicaciones semánticas suministrando un modelo empaquetado, modular y codificando mecanismos para codificar datos sin los módulos [10] .

SOAP consiste de tres partes:

- El *SOAP Envelope*, todo mensaje debe ir contenido en un sobre o envoltura que define un *framework* para expresar qué hay en un mensaje, qué estructura debe tener y como procesarlo.
- El *SOAP Encoding Rules* define el mecanismo que puede ser usado para intercambiar instancias de tipos de datos definidos en la aplicación.
- El *SOAP RPC* define una convención que puede ser usada para representar llamadas y respuestas a procedimientos remotos.

El estándar SOAP no hace ninguna consideración acerca del medio de transmisión de los mensajes. La idea es que SOAP sea independiente del protocolo, siendo el desarrollador el encargado de tomar la decisión sobre qué medio empleará para publicar sus servicios y cómo serán consumidos. Esto hace posible la comunicación de objetos SOAP por medio de http, ftp o smtp, cabe señalar que los ejemplos de servicios Web publicados a través de otros protocolos que no sean http son escasos.

El protocolo SOAP tiene diversas ventajas sobre otras maneras de llamar a funciones de manera remota como DCOM, CORBA o directamente en TCP/IP:

- Es sencillo de implementar, probar y usar.
- Es un estándar de la industria adoptado por W3C[3] y por varias empresas.
- Utiliza los mismos estándares de la Web para casi todo. La comunicación se hace mediante HTTP con paquetes virtualmente idénticos; los protocolos de autentificación y encriptación son los mismos.
- Atraviesa firewalls y routers que toman los mensajes SOAP como una comunicación HTTP.
- Tanto los datos como las funciones se describen en XML, lo que permite que el protocolo no sólo sea más fácil de utilizar sino que también sea muy sólido.
- Es independiente del sistema operativo y procesador.

En conclusión, se puede decir que SOAP, es un estándar en servicios WEB, y sobre todo, que al ser independiente del lenguaje y de la plataforma permite acceder a varios lenguajes y entornos de desarrollo que admitan servicios WEB.

Se han sentado las bases teóricas que rigen este desarrollo, en el siguiente capítulo se mostrará el análisis y diseño de los componentes COM y SOAP. Como se explicó, un buen análisis y diseño van a permitir un par de componentes robustos, extensibles y adaptables.

[3] Más información en el siguiente link: http://www.w3.org/TR/SOAP/

3. Análisis y diseño de los componentes de software

En la actualidad, la mayoría de las aplicaciones de software que se encuentran desarrollando no son estáticas, sino que crecen y se desarrollan a medida que cambian las necesidades del mercado y de los consumidores. Por tal motivo, es necesario tener la capacidad de analizar esos cambios y traducirlos en nuevos modelos de software, es aquí, en donde el análisis y diseño orientado a objetos juega un papel primordial, siendo la piedra angular para que la evolución de los sistemas de cómputo sea una realidad.

3.1. Análisis de los componentes de software

La principal meta del análisis de los componentes es lograr componentes que satisfagan las necesidades de los usuarios y lograr que estos componentes puedan tener un mantenimiento muy sencillo. A continuación se presenta el documento de requisitos de usuario y el de especificación de requerimientos del software siguiendo las directrices dadas por el estándar "IEEE Recommended Practice for Software Requirements Specification ANSI/IEEE 830 1998".

3.1.1. Documento de requisitos de usuario

Introducción

En este documento se presenta el resultado de un trabajo previo de captura de requisitos, respecto a los componentes de software que se desean construir.

Componentes COM-SOAP para operaciones matriciales

Básicamente, se construirá un par de componentes de software, uno trabajando bajo la tecnología COM y el otro bajo la tecnología SOAP, estos componentes trabajarán de manera local y remota, respectivamente.

Requisitos

Con el desarrollo de este par de componentes de software se pretende solucionar el alto costo de desarrollo de software que actualmente existen para las aplicaciones que requieran operar matrices numéricas, para esto se pretende brindar lo siguiente:

1. Crear un par de componentes de software, cada uno de ellos estará orientado a conexiones locales y remotas respectivamente, con esto se pretende cubrir un amplio

rango de aplicación respecto a los lenguajes de programación que pueden aplicar estos componentes.
2. Cada componente podrá generar todas las matrices que MATLAB soporta, con el fin de proporcionar una amplia gama de posibilidades a los desarrolladores al momento de generar matrices.
3. Además de soportar las matrices que MATLAB genera, cada componente podrá soportar la generación de matrices con los valores que el desarrollador requiera.
4. Soportar todas las operaciones matriciales que MATLAB ofrece para tener componentes robustos.
5. Soporte para el manejo de excepciones, de esta manera además de tener un par de componentes robustos, tendremos componentes estables.

3.1.2. Especificación de Requerimientos de Software (ERS)

Introducción

Este apartado trata sobre la especificación de los requerimientos que los componentes COM y SOAP deben cumplir.

1. Propósito

El propósito de la especificación de requerimientos es delimitar claramente el dominio de los componentes, de tal manera que de antemano se conozcan las funcionalidades y las restricciones de los mismos. A su vez, este documento pretende sea la base para el diseño de los componentes de software. Este documento servirá al desarrollador de los componentes de software para limitar y comprender de manera clara la etapa de diseño e implementación de los componentes, a su vez, este documento servirá a los revisores de esta tesis para corroborar que los componentes cumplan con todas las características con las que fueron planeados.

2. Ámbito del sistema

La principal razón para el desarrollo de estos componentes es la falta de opciones al momento de desarrollar aplicaciones en las cuales se encuentren implicadas operaciones matriciales.

Estos componentes de software, orientados a los estudiantes universitarios serán de gran ayuda, especialmente a los estudiantes de ingenierías o licenciatura en matemáticas aplicadas, debido a que en el transcurso de la licenciatura es recurrente el uso de aplicaciones que utilicen matrices para la solución de problemas, entonces, ante la falta de herramientas que ayuden con estas operaciones, y el poco conocimiento que se tiene de las mismas, los componentes pueden ser una ayuda excelente al utilizarse en diversos lenguajes de programación, siendo los más utilizados dentro de la UTM: el lenguaje C, Pascal, Java, Visual Basic, Php, Power Builder, entre otros.

De la misma manera, estos componentes serán de gran ayuda a los profesores-investigadores de los diversos institutos y cuerpos académicos con los que cuenta la UTM, para este caso es muy importante el componente SOAP, debido a que la infraestructura tecnológica con la que cuenta la UTM brinda la facilidad de tener instalado en cualquier servidor con plataforma Windows tanto el componente SOAP como MATLAB, de esta manera cada profesor-investigador podrá acceder a los recursos del componente de manera sencilla siempre y cuando esté conectado a la red.

Estos componentes de software se pretende sean parte de una biblioteca de componentes, la cual sería una biblioteca robusta de componentes que integraría la mayoría de operaciones que MATLAB soporta, de esta manera se lograría el principal objetivo de la programación orientada a componentes, la reutilización de los mismos.

3. Definiciones, Acrónimos y Abreviaturas

3.1. Definiciones

Tabla 3.1. Definiciones para ERS de los componentes de software

Usuarios	Son todas aquellas personas que desean desarrollar sistemas de software con ayuda de los componentes COM y/o SOAP desarrollados

3.2. Acrónimos

ERS Especificación de Requisitos de Software

DRU Documento de Requerimientos de Usuario

3.3. Abreviaturas

Tabla 3.2. Abreviaturas para ERS de los componentes de software

COM	*Component Object Model*, tecnología de componentes desarrollada por Microsoft, también conocida como *Active X*.
SOAP	*Simple Object Access Protocol*, es un protocolo para intercambiar mensajes entre sistemas de cómputo, principalmente los que están formados por componentes de software.
MATLAB	*Matrix Laboratory*, es un sistema orientado a matemáticas creado por *The Math Works* en 1984. Es un sistema de software muy usado en universidades, centros de investigación y actividades de ingeniería.
UTM	Universidad Tecnológica de la Mixteca

4. Referencias del Documento

- IEEE Recommended Practice for Software Requirements Specification. ANSI/IEEE std. 830, 1998
- Sommerville, Ian. Ingeniería de software. 6a. edición. Addison Wesley

5. Visión general del documento

Esta sección (ERS) está dividido en tres partes: la introducción que fue descrita con anterioridad, la descripción general y por último los requisitos específicos.

En la introducción se muestra una visión general del documento, muestra la estructura básica para que el lector pueda ubicarse dentro del mismo. En la descripción general se muestra la funcionalidad básica de los componentes de software, sus limitantes, supuestos y dependencias que se encuentran relacionados con el diseño e implementación de los componentes. Por último, la descripción de los requisitos se encuentra en la sección final del documento.

Descripción general

En esta sección se presenta una descripción de alto nivel de los componentes de software. Se presentarán las principales funciones que los componentes deben realizar, la información utilizada, las restricciones y otros factores que afecten el desarrollo del mismo.

6. Perspectivas del producto

Estos componentes serán parte de la librería de componentes que se tiene planeado desarrollar, para su uso es necesario que interactúen con MATLAB en su versión 6.5 o superior.

7. Funciones de los componentes de software

Los componentes de software a desarrollar tendrán dos tareas fundamentales:

- Métodos para la creación de las diversas matrices que MATLAB ofrece, así como métodos para la creación de matrices que el usuario decida, proporcionando a los usuarios una interfaz amplia que ofrezca flexibilidad para la utilización de estos métodos.
- Métodos para la ejecución de las operaciones matriciales que MATLAB ofrece, proporcionando a los usuarios una interfaz amplia y amigable que ofrezca flexibilidad para la utilización de los métodos respectivos.

A continuación se muestra una descripción más detallada de estas operaciones.

7.1. Creación de Matrices

En lo que respecta a la creación de matrices, el usuario tendrá acceso a todas las matrices numéricas que MATLAB ofrece, las cuales son:

Tabla 3.3. Matrices que ofrece Matlab.

• Matriz Hadamard	• Matriz Hilbert
• Matriz Hilbert Inversa	• Matriz Identidad
• Matriz Mágica	• Matriz Pascal
• Matriz Wilkinson	• Matriz aleatoria uniforme (con valores entre 0 y 1)
• Matriz aleatoria (con valores entre $-\infty$ y ∞)	• Matriz con valores iniciales de 1 en cada casilla.
• Matriz con valores iniciales de 0 en cada casilla	• Matriz Rosser

De la misma manera, el usuario tendrá la posibilidad de crear matrices que él mismo podrá definir. Los componentes, tanto el COM como el SOAP, tendrán diversos métodos para que la creación de estas matrices tenga diversas variantes, de esta forma la aplicación de las mismas será flexible en diversos casos.

7.2. Operaciones Matriciales

En el apartado anterior se mostraron todas las matrices que soportarán los componentes COM y SOAP, estos componentes a su vez, soportarán todas las operaciones que sobre estas matrices se pueden realizar en MATLAB. De esta manera, las funciones que los componentes soportarán son las siguientes:

Tabla 3.4. Funciones soportadas por los componentes.

• Balance de la matriz.	• Concatenación Horizontal de Matrices.
• Concatenación vertical de matrices.	• Condición recíproca de la matriz.
• Descomposición en valor singular de la matriz.	• Determinante de la matriz.
• Devolver los índices de la matriz.	• Diagonal de la matriz.
• División.	• Exponencial.
• Factorización Cholesky de la matriz.	• Factorización LU de la matriz.
• Factorización QR de la matriz.	• Factorización Schur de la matriz.
• Inversa de la matriz.	• Matriz Hankel de la matriz.
• Multiplicación.	• Norma de la matriz.
• Ordenar los elementos de la matriz.	• Ortogonal de la matriz.
• Parte alta de la matriz.	• Parte baja de la matriz.
• Producto acumulativo.	• Producto Hessenberg de la matriz.
• Producto Kronecker de la matriz.	• Pseudo inversa de la matriz.
• Raíz cuadrada de los elementos de la matriz.	• Rango de la matriz.
• Resta.	• Rotar la matriz 90 grados.
• Suma.	• Suma acumulativa.
• Suma de la diagonal de la matriz.	• Tamaño de la matriz.
• Transpuesta de la matriz.	• Transpuesta no conjugada de la matriz.
• Valores máximos de la matriz.	• Valores medios de la matriz.
• Valores mínimos de la matriz.	• Verificar si la matriz está vacía.
• Voltear la matriz de arriba hacia abajo.	• Voltear la matriz de izquierda a derecha.

• Voltear la matriz N dimensiones.	

De la misma manera que con la creación de las matrices, estas funciones se soportarán de diversas maneras, de tal suerte que los desarrolladores que incluyan en sus proyectos algunos de los componentes puedan tener mayor flexibilidad a la hora de manipular las matrices.

8. Características de los usuarios

Los usuarios son aquellas personas que deseen utilizar los componentes COM y/o SOAP para desarrollo de software. El número de usuarios que los componentes aceptan son:

- Desde COM, uno por equipo de cómputo.
- Desde SOAP, tantos como puedan acceder al servidor donde se encuentran publicados los servicios Web.

Los usuarios deberán tener experiencia de mínima a media en el desarrollo de software.

9. Restricciones

Los componentes de software solo estarán orientados a la creación y manipulación de matrices soportadas por MATLAB, no se podrán realizar operaciones matemáticas diferentes a las anteriormente mencionadas.

Por otro lado, no existen restricciones en cuanto al alcance de las operaciones, es decir, todos los desarrolladores tendrán la misma cantidad de métodos en los componentes para poder ocuparlos.

La restricción para el uso de los componentes es únicamente:

- Para el uso del componente COM es necesario un lenguaje de programación que soporte la tecnología COM, actualmente la gran mayoría de los lenguajes de programación para aplicaciones de escritorio aceptan esta tecnología.

10. Suposiciones y dependencias

10.1. Suposiciones

Se asume que los requisitos descritos en este documento son estables una vez que es aprobado como tema de tesis. Cualquier petición de cambio en las especificaciones del proyecto será propuesta como trabajo futuro.

10.2. Dependencias

Ambos componentes de software son dependientes totalmente de MATLAB, en el caso del componente SOAP, se suma la dependencia de una máquina que ofrezca los servicios de MATLAB y que se encuentre conectada a la red.

Requisitos específicos

En este apartado se muestran los requisitos funcionales que deben cumplir los componentes de software para poder ser utilizados de la manera en que fueron planeados. Todos los requisitos aquí expuestos son ***esenciales***, no se aceptará que este proyecto sea válido si no cumple con ***todos*** los requisitos aquí mostrados.

11. Requisitos funcionales

11.1. Creación de matrices

Req (1) Cada vez que el usuario desee crear una matriz, deberá tener una sesión de MATLAB abierta, obtener el identificador de la sesión en la que quiere trabajar y apoderarse de dicha sesión, para lo cual tendrá que manejar el identificador de sesión.

Req 1.1 Crear una sesión de MATLAB y obtener el identificador de la sesión, en caso que la sesión ya esté abierta simplemente se obtiene el identificador de sesión.

Req 1.2 Crear el objeto matriz.

Req 1.3 Asignarle a la nueva matriz creada la sesión de MATLAB mediante el identificador de sesión4.

11.2. Manipulación de matrices

Req (2) Cada vez que el usuario desee manipular una matriz, deberá tener una sesión de MATLAB abierta, obtener el identificador de la sesión en la que quiere trabajar y apoderarse de dicha sesión, para lo cual tendrá que manejar el identificador de sesión, además, necesitará tener creada una matriz (Ver Req 1), para poder trabajar sobre esa matriz.

Req 2.1 Crear una sesión de MATLAB y obtener el identificador de la sesión, en caso que la sesión ya esté abierta simplemente se obtiene el identificador de sesión.

Req 2.2 Crear el objeto matriz.

Req 2.3 Asignarle a la nueva matriz creada la sesión de MATLAB mediante el identificador de sesión.

Req 2.4 El objeto podrá acceder a todas las operaciones matriciales que el componente soporte

12. Interfaces Externos

Al ser este proyecto el desarrollo de componentes de software, estos componentes deben tener interfaces de programación de aplicaciones, conocidas como API, estas API ofrecen una serie de métodos de uso general lo cual ayuda a los desarrolladores que hacen uso de la funcionalidad de los componentes evitándose programar todo desde el principio.

Por otro lado, para el componente SOAP, que es un componente orientado a aplicaciones WEB, es necesario tener una interfaz de comunicación, que permita la interacción entre la computadora que ofrezca los servicios de MATLAB y los clientes que ocupen dichos servicios.

12.1. Interfaces Software

Para ambos componentes se tienen dos interfaces de software, una de ellas está orientada a usuarios que tengan conocimientos de MATLAB y sobre todo de la creación y manipulación de operaciones matriciales, en esta interfaz se podrán crear matrices y realizar operaciones de manera más directa, pero para usuarios que no tengan conocimientos de MATLAB y de las operaciones matriciales puede resultar un poco confuso.

De la misma manera, existe otra interfaz de software que está orientada a usuarios poco o nulamente experimentados en MATLAB, esta interfaz está totalmente orientada a objetos y

[4] Este proceso se debe realizar para cada tipo de matriz que el componente soporte.

tiene un mayor número de métodos para poder ofrecer al usuario mayor flexibilidad a la hora de desarrollar los sistemas.

De esta manera, cada interfaz deberá tener los siguientes métodos:

12.1.1 Interfaz para usuarios con experiencia en MATLAB

En esta interfaz, el usuario ejecutará las operaciones básicas tal y como se encuentran en MATLAB, los parámetros de entrada serán similares a los que MATLAB maneja, estas operaciones no tienen parámetros de salida, ya que no es propiamente orientado a objetos. Las funciones de esta interfaz tiene un nombre en el idioma Inglés, ya que es el idioma estándar en el desarrollo de software, además el nombre de las funciones será similar al nombre de las mismas funciones que tiene MATLAB.

Tabla 3.5. Funciones orientadas a los usuarios con experiencia en MATLAB

• Add	• Chol
• CumSum	• Det
• Diag	• Divide
• Eye	• Exponential
• Find	• FlipDim
• FlipLR	• FlipUD
• Hadamard	• Hankel
• Hilbert	• HorzCat
• IdentityNotSquare	• Inv
• InvHilb	• IsEmpty
• Kron	• Lu
• Magic	• Max
• Median	• Min
• Mult	• Norm
• OnesMultiDim	• OnesXYMatrix
• OnesXYZ	• Pascal
• Pinv	• ProdSum
• Qr	• RandMultiDim
• RandNMultidim	• RandXY
• RandXYZ	• RandnXY
• RandNXYZ	• Rank
• RCond	• Rosser

• Rot90	• Schur
• Size	• Sort
• SqrtM	• Substract
• SVD	• Transpose
• TransposeNonConjugate	• TriL
• TriU	• VertCat
• Wilkinson	• ZerosMultiDim
• ZerosXYZ	• ZerosXYMatrix

Las operaciones que muestran en la Tabla 3.5 están soportadas tanto por el componente COM como por el componente SOAP.

12.1.2 Interfaz orientada a objetos (usuarios sin experiencia en MATLAB)

Esta interfaz, está basada en la anterior, de tal manera que se puedan crear más funciones a partir de las funciones soportadas en la interfaz anterior. Esta interfaz ofrece una mayor flexibilidad ya que el desarrollador podrá elegir qué tipos de parámetros de entrada y de salida utiliza para ciertas funciones, por ejemplo, en las funciones que puedan devolver valores dobles (determinante, rango, suma de diagonal, etc.) el desarrollador podrá elegir entre ocupar funciones que devuelvan esos valores como valores dobles, o como matrices o como variables. Entonces se pretende cubrir todas las funciones que puedan ser variadas para aumentar la flexibilidad de la interfaz.

Por todo lo anterior, esta interfaz cuenta con los siguientes métodos:

Tabla 3.6. Funciones orientadas a usuarios sin experiencia en MATLAB.

• AddMatrix	• AddVar
• Balance	• CholeskyFactor
• CumulativeAdd	• CumulativeProd
• DiagonalOfMatrix	• DivideIntoMatrix
• DivideIntoVar	• DivideMatrix
• DivideVar	• Exponential
• Find	• FlipLeftToRight
• FlipNDimensions	• FlipUpToDown
• GenerateIdentityMatrix	• GenerateHadamardMatrix
• GenerateHadamardMatrixAsVar	• GenerateHilbertMatrix
• GenerateHilbertMatrixAsVar	• GenerateIdentityMatrixAsVar

• GenerateIdentityNotSquareMatrix	• GenerateIdentityNotSquareMatrixAsVar
• GenerateInverseHilbertMatrix	• GenerateInverseHilbertMatrixAsVar
• GenerateMagicMatrix	• GenerateMagicMatrixAsVar
• GenerateOnesMultiDimMatrix	• GenerateOnesMultiDimMatrixAsVar
• GenerateOnesXYMatrix	• GenerateOnesXYMatrixAsVar
• GenerateOnesXYZMatrixAsVar	• GenerateOnesXYZMatrix
• GeneratePascalMatrix	• GeneratePascalMatrixAsVar
• GenerateRandomMultiDimMatrix	• GenerateRandomMultiDimMatrixAsVar
• GenerateRandomXYMatrix	• GenerateRandomMatrixXYAsVar
• GenerateRandomXYZMatrix	• GenerateRandomXYZMatrixAsVar
• GenerateRosserMatrix	• GenerateRosserMatrixAsVar
• GenerateUniformRandomMultiDimMatrix	• GenerateUniformRandomMultiDimMatrixAsVar
• GenerateUniformRandomXYMatrix	• GenerateUniformRandomXYMatrixAsVar
• GenerateUniformRandomXYZMatrix	• GenerateUniformRandomXYZMatrixAsVar
• GenerateWilkinsonMatrix	• GenerateWilkinsonMatrixAsVar
• GenerateZerosMultiDimMatrix	• GenerateZerosMultiDimMatrixAsVar
• GenerateZerosXYMatrix	• GenerateZerosXYZMatrixAsVar
• GenerateZerosXYMatrixAsVar	• GenerateZerosXYZMatrix
• GetDeterminant	• GetDeterminantAsMatrix
• GetDeterminantAsVar	• GetMatrixSize
• GetNorm	• GetNormAsMatrix
• GetNormAsVar	• GetRank
• GetRankAsMatrix	• GetRankAsVar
• GetRcond	• GetRconAsMatrix

• GetRcondAsVar	• GetTrace
• GetTraceAsMatrix	• GetTraceAsVar
• HankelMatrix	• HankelMatrixAsVar
• Hessenberg	• HorizontalConcat
• HorzcatFromMatrix	• HorizontalCatVar
• HorizontalCatFromVar	• Inverse
• IsMatrixEmpty	• Kronecker
• KronIntoMatrix	• KronMatrix
• KronVar	• LuFactor
• Maximum	• Median
• Minimum	• MultMatrix
• MultIntoMatrix	• MultVar
• Orthogonal	• PseudoInverse
• QrFactor	• Rotate90Degrees
• SchurDescomposition	• SingularValueDescomposition
• SortMatrix	• SqrtMatrix
• SubstractFromMatrix	• SubstractFromVar
• SubstractMatrix	• SubstractVar
• Transpose	• TransposeNonConjugate
• TriangularLowPart	• TriangularUpPart
• VerticalConcat	• VerticalCatFromVar
• VertcatFromMatrix	• VerticalCatVar

Las operaciones que se muestra en la Tabla 3.6 están soportadas tanto por el componente COM como por el componente SOAP.

12.2. Interfaces de comunicación

La conexión a la red se establece por medio de una conexión a la red Ethernet de la Universidad Tecnológica de la Mixteca, ya que en esta red se encuentra la computadora que funciona como servidor de servicios de MATLAB.

13. Requisitos de rendimiento

Al ser MATLAB un sistema bastante estable, se prevé que no existan problemas en cuanto al número de conexiones tanto locales como remotas.

14. Requisitos de desarrollo

Indudablemente, al tratarse del desarrollo de un par de componentes, como se citó en el capítulo dos, el desarrollo de este proyecto estará basado en el paradigma orientado a componentes y el paradigma orientado a objetos, teniendo un ciclo de vida espiral.

15. Requisitos tecnológicos

Para utilizar el componente de manera local (COM), se necesitará una PC con la siguiente configuración mínima:

Procesador: Pentium III 750 Mhz.

Memoria: 128 Mb.

Espacio libre en disco: 1 Gb. (para la instalación de MATLAB).

Sistema Operativo: Microsoft Windows 98, Me, XP Profesional (recomendado).

Para utilizar el componente de manera remota (SOAP), se necesitará una PC con una configuración mínima de:

Procesador: Pentium III 750 Mhz.

Memoria: 128 Mb(mínimo), 256(recomendado).

Espacio libre en disco: 10 Mb.

Tarjeta Ethernet o Módem.

Acceso a Internet.

Sistema operativo: cualquiera que soporte los lenguajes de programación orientados a WEB y que soporten la tecnología SOAP.

3.2. Diseño de los componentes de software

El diseño de los componentes de software viene precedido, en gran medida, por la declaración de los requisitos funcionales y no funcionales del apartado anterior.

En el apartado anterior, se menciona que se crearon dos interfaces de software para el componente COM y SOAP respectivamente, como ya se explicó, cada una de estas interfaces tiene un objetivo específico, a su vez, el nombre de dichas interfaces será el siguiente:

- *IMOCOMMatlabMatrix*: Interfaz orientada a objetos, trabajará bajo tecnología COM.
- *IMOCOMMatlabMatrixOperationLib*: Interfaz dirigida a usuarios expertos en MATLAB, la cual trabajará bajo la tecnología COM.
- *ISOAPMatlabMatrix*: Interfaz análoga a la interfaz IMOCOMMatlabMatrix, con la diferencia de que esta interfaz trabajará bajo la tecnología SOAP.
- *ISOAPMatlabMatrixOperationLib*: Interfaz análoga a la interfaz IMOCOMMatlabMatrixOperationLib, con la diferencia de que esta trabajará bajo la tecnología SOAP.

Dentro del diseño de estos componentes se han considerado algunos puntos significativos que es importante aclarar.

Como se mencionó con anterioridad, la tecnología SOAP es un protocolo para el intercambio de mensajes entre aplicaciones de software y principalmente entre componentes de software, por lo tanto, el componente SOAP es el medio de comunicación entre las aplicaciones Web y el componente COM, es decir, el componente SOAP depende de las operaciones que realice el componente COM, teniendo (el componente SOAP) como función principal mandar peticiones al componente COM de las operaciones requeridas y recibir los resultados de dichas operaciones.

Para el desarrollo de los componentes se utilizan dos interfaces externas: IMOMatlabSession e IMOMatlabNumericArray, estas interfaces forman parte del desarrollo de la tesis "*Desarrollo de componentes software COM y SOAP, distribuidos y heterogéneos para encapsular la funcionalidad básica y sus tipos de datos de MATLAB, y soportar la integración de nuevos componentes COM-SOAP*".

La IMOCOMMatlabMatrixOperationLib ocupa la interfaz IMOMatlabSession, la función de esta interfaz es ejecutar el comando que reciba de la interfaz IMOCOMMAtlabMatrixOperationLib, de la misma manera, la interfaz IMOCOMMatlabMatrix hereda de la interfaz IMOMatlabNumericArray, cuya función es crear un arreglo numérico en MATLAB, mediante esta función se podrán manipular los arreglos para convertirlos en matrices y podrán ser devueltas al usuario.

Para tener una idea mas clara de lo anterior se presenta la Figura 3.1 en la que se presenta de manera gráfica el diseño general de los componentes de software.

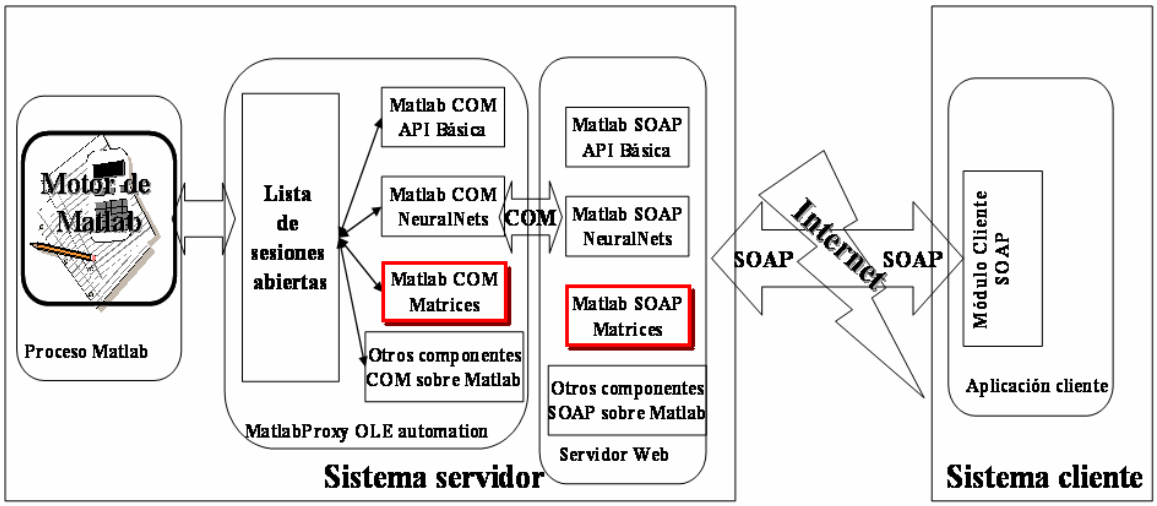


Figura 3.1. Diseño general de los componentes.

Ya que tenemos una idea general acerca del diseño de los componentes, es necesario avanzar al siguiente punto, es decir, identificar alguna arquitectura de software. En este caso, se optó por descomponer los componentes en una arquitectura por capas. Estas capas van a permitir que a su vez se descompongan en tareas más pequeñas llamados paquetes en un lenguaje UML. En un principio, la descomposición más general es la que se muestra en la Figura 3.2.

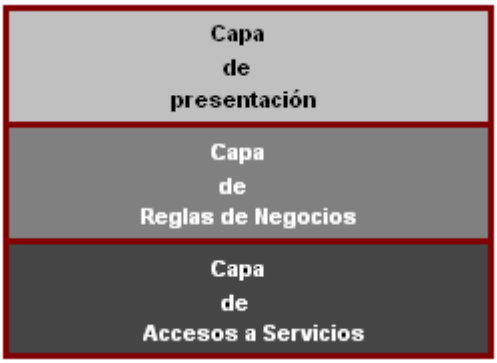


Figura 3.2. Diseño general de capas.

Estas capas se descomponen en tareas esenciales que a su vez se mapean y se convierten en componentes de software, servicios y procesos los cuales mostrarán la manera en que se va a ejecutar la solución.

La Figura 3.3 muestra una descripción gráfica más detallada de las capas de los componentes.

Figura 3.3. Vista lógica de las capas de los componentes

3.2.1. Diseño orientado a objetos

Como se mencionó en capítulos anteriores, el diseño de los componentes se realizó apoyado en el diseño orientado a objetos, siguiendo la notación del Lenguaje Unificado de Modelado (UML), en el marco del método general de Diseño Orientado a Objetos [36] que se compone del contexto y uso del sistema, diseño arquitectónico, identificación de objetos y modelos de diseño.

3.2.1.1. Contexto y uso del sistema

En este apartado se presenta el diseño de paquetes, lo cual va a ayudar a dividir los componentes en subsistemas, permitiendo la repartición de tareas y responsabilidades a los usuarios. Este modelo de paquetes se presenta en la Figura 3.4.

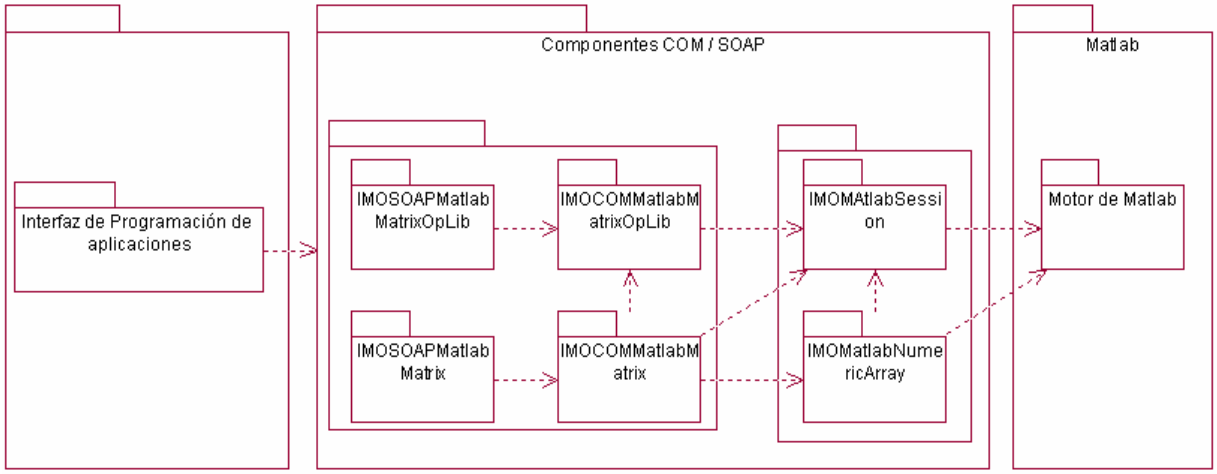


Figura 3.4. Contexto de los componentes de software COM y SOAP.

En este diagrama se puede observar que los componentes son dependientes de dos interfaces de software, estas interfaces administrarán las sesiones en MATLAB (IMOMatlabSession), además de dar pauta para crear arreglos y a través de ellos crear matrices (IMOMatlabNumericArray). De la misma forma, los componentes SOAP (ISOAPMatlabMatrixOperationLib e ISOAPMatlabMatrix) tienen dependencias de sus componentes análogos COM (IMOCOMMatlabMatrixOperationLib e IMOCOMMatlabMatrix) respectivamente.

El siguiente paso es mostrar el diagrama general de casos de uso, este diagrama tiene la finalidad de dar una visión mas amplia del entorno entre los componentes de software, sus operaciones y el papel del usuario de los componentes de software. Este diagrama se presenta en la Figura 3.5.

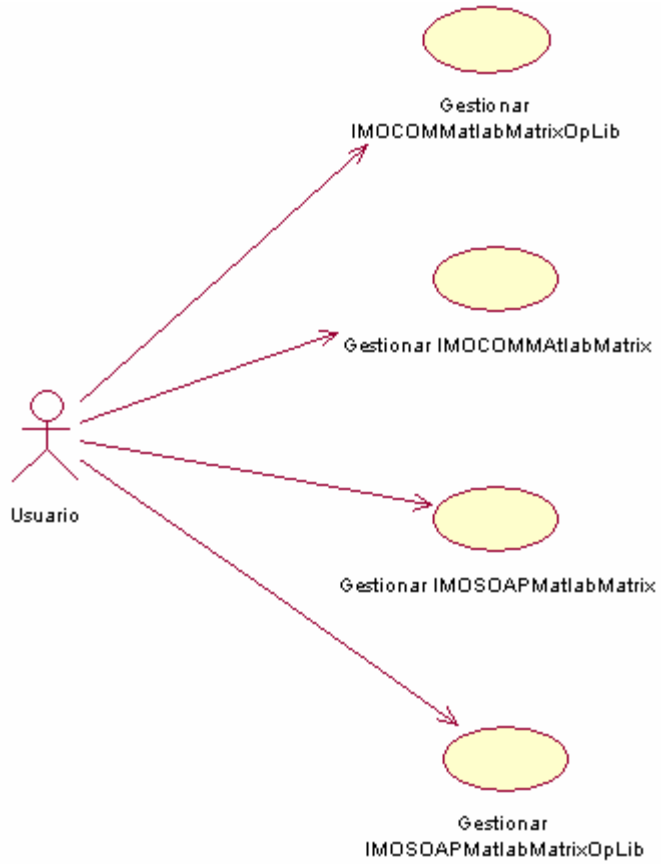


Figura 3.5. Diagrama general de casos de uso para los componentes de software COM y SOAP.

El diagrama presentado en la Figura 3.5 muestra cuatro casos de uso considerados primarios, los cuales son: Gestionar IMOCOMMatlabmatrixOperationLib, Gestionar IMOCOMMatlabMatrix, Gestionar ISOAPMatlabMatrixOperationLib y Gestionar ISOAPMatlabMatrix.

Para profundizar a detalle en estos casos de uso se utiliza la metodología del Proceso Unificado Rational (RUP), utilizando la herramienta *Rational Rose* para la especificación de casos de uso, la especificación completa de los casos de uso se muestra en el Anexo 3.

3.2.1.1.1. Especificación de Casos de Uso (ECU)

Un *caso de uso* es un documento narrativo que describe la secuencia de eventos de un actor (agente externo) que utiliza un sistema para completar un proceso.

A continuación se describen dos casos de uso siguiendo la tecnología RUP, la descripción de los casos de uso va a ser de manera expandida. Un caso de uso expandido muestra más detalles que un caso de uso de alto nivel. Los casos de uso expandidos son útiles para alcanzar un conocimiento más profundo de los procesos y los requerimientos.

Especificación de caso de uso: Gestionar IMOCOMMatlabMatrix

Versión 1.2

Historial de revisiones

Fecha	Versión	Descripción	Autor
25/Octubre/2004	1.0	Creación del Documento	Felipe de Jesús García Pérez
04/Noviembre/2004	1.1	Introducción de primeros diagramas	Felipe de Jesús García Pérez
09/Noviembre/2004	1.2	Documento completado	Felipe de Jesús García Pérez

Escenario Obtener Determinante como Valor Doble.

Objetivo

Obtener el determinante de una matriz cuadrada como un valor doble.

Breve descripción

El usuario podrá gestionar la interfaz IMOCOMMatlabMatrix para poder obtener el determinante de una matriz cuadrada con el formato de valor doble.

Actores

Usuario

Flujo de eventos

Flujo básico

Actor	*Sistema*
Este caso de uso comienza cuando el usuario decide instanciar la interfaz IMOCOMMatlabMatrix, y crea una matriz cuadrada.	
El usuario utiliza el objeto el cual contiene la matriz cuadrada y ejecuta la función “GetDeterminantAsDouble”.	
	El sistema ejecuta la operación y devuelve el resultado en un valor doble.
Finaliza el caso de uso.	

Excepción

Si la interfaz no pudo ser instanciada se notificará mediante un mensaje de error.
Si la operación devolver el determinante no pudo ser ejecutada se notificará mediante un mensaje de error.

Precondiciones

El usuario deberá haber iniciado una sesión en MATLAB y debe tener una matriz cuadrada creada.

Diagrama de actividades

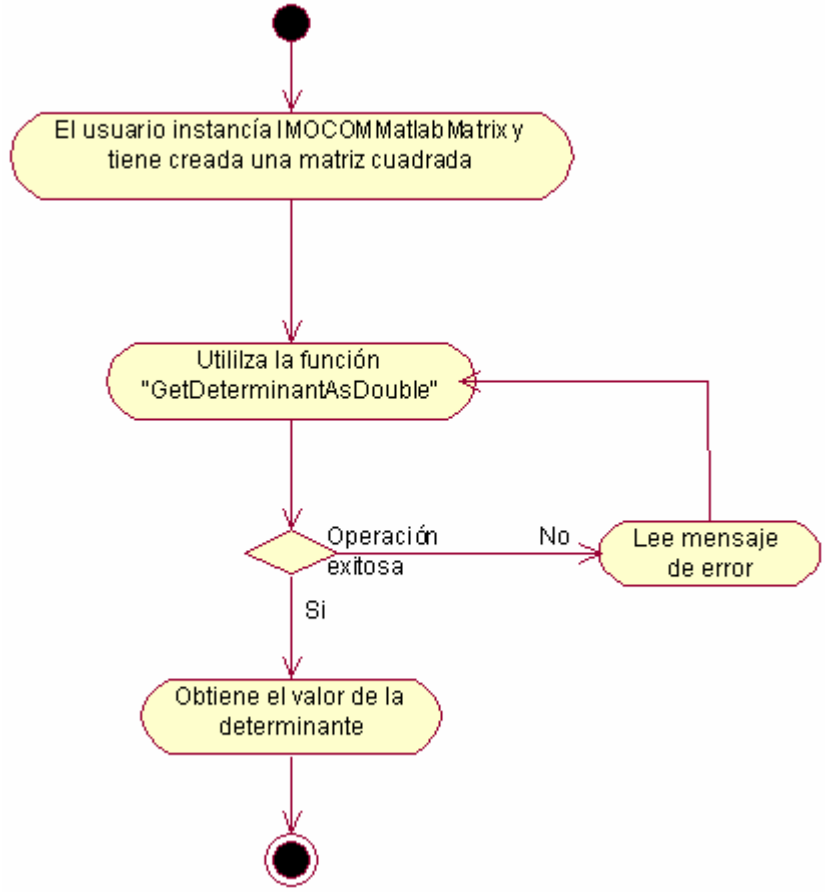


Diagrama de secuencia

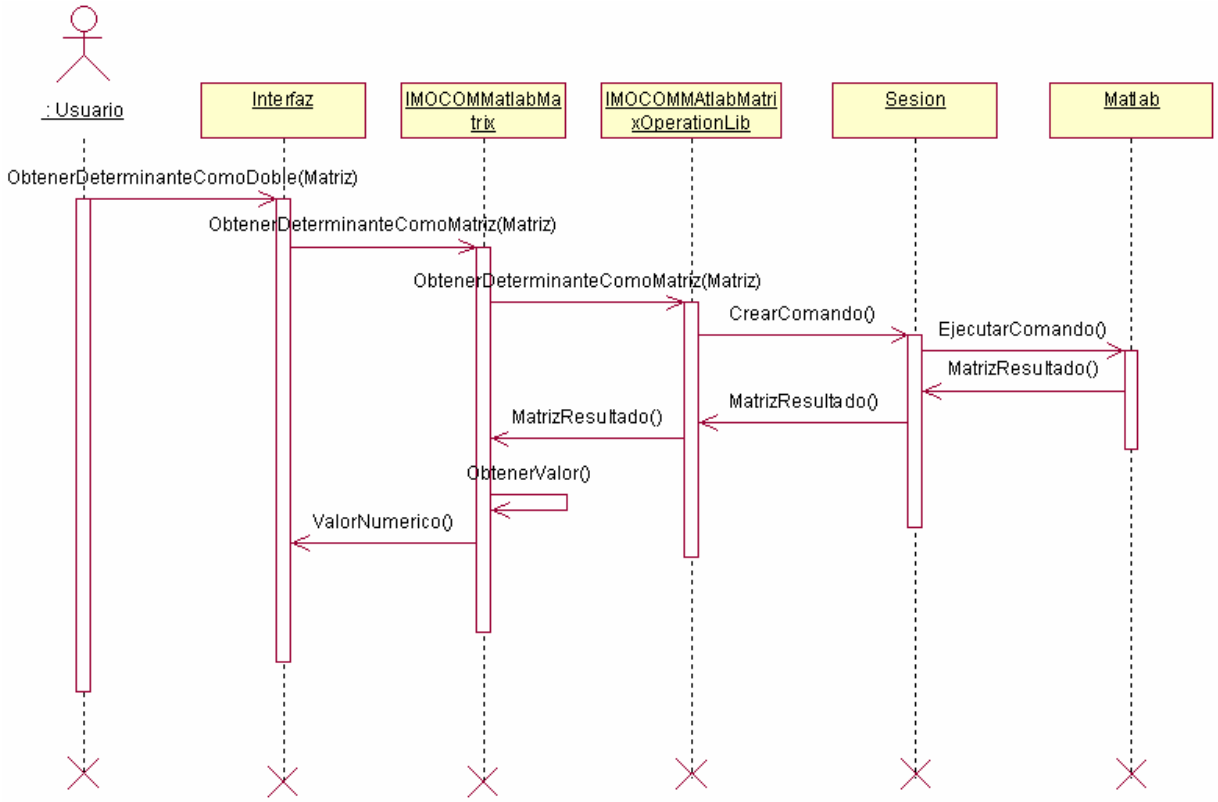


Diagrama de colaboración

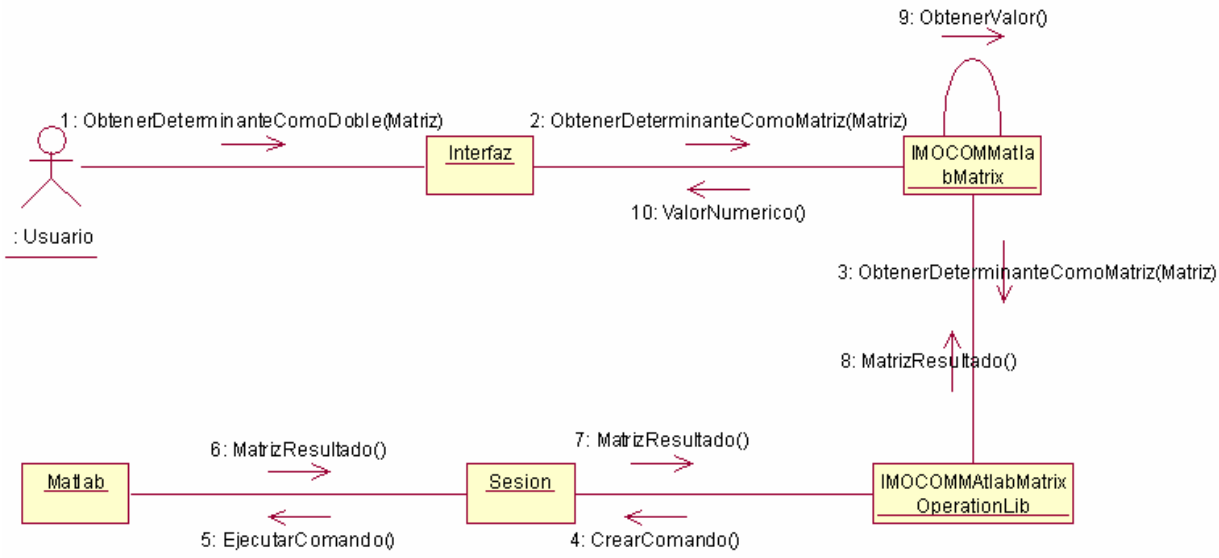


Diagrama de estados

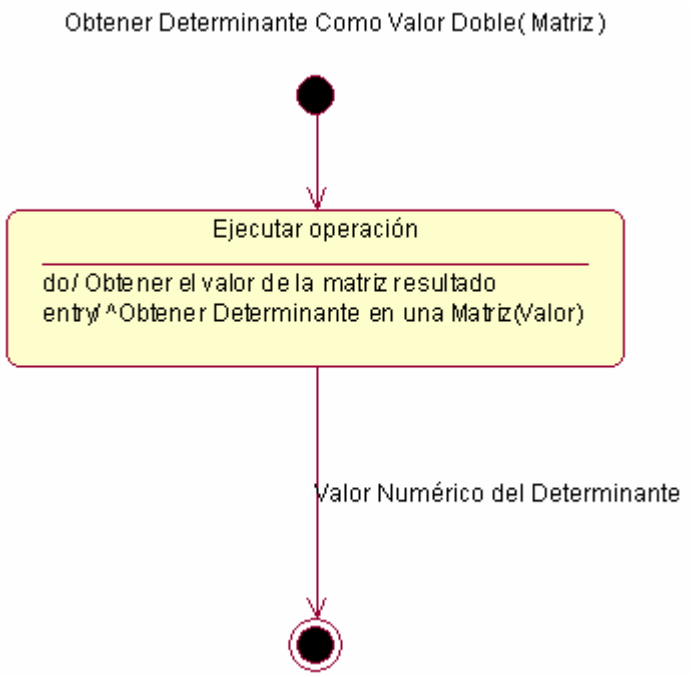
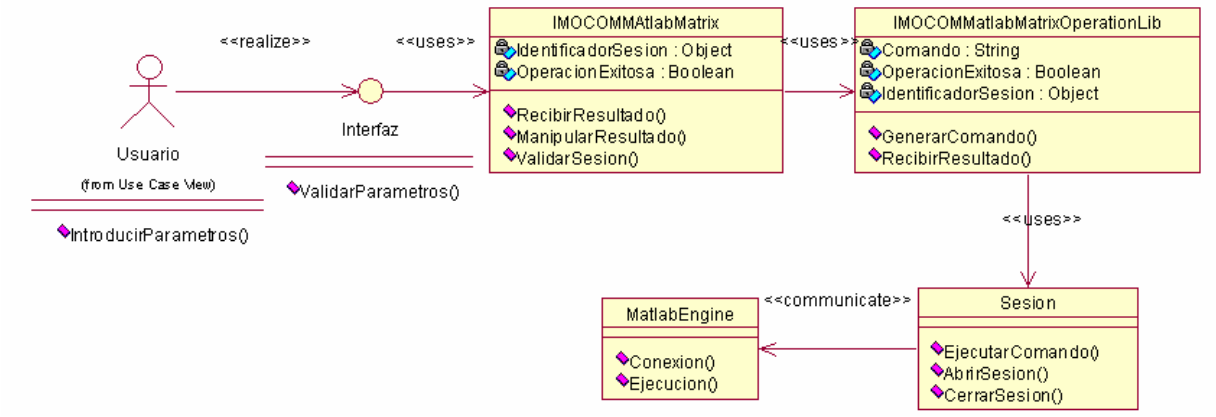


Diagrama de clases



Especificación de caso de uso: GestionarIMOCOMMatlabMatrixOperationLib
Versión 1.2

Historial de revisiones

Fecha	Versión	Descripción	Autor
25/Octubre/2004	1.0	Creación del Documento	Felipe de Jesús García Pérez
04/Noviembre/2004	1.1	Introducción de primeros diagramas	Felipe de Jesús García Pérez
09/Noviembre/2004	1.2	Documento completado	Felipe de Jesús García Pérez

Escenario Crear Matriz Pascal.

Objetivo

Gestionar la interfaz IMOCOMMatlabMatrixOperationLib para crear una matriz del tipo Pascal.

Breve descripción

El usuario podrá gestionar la interfaz IMOCOMMatlabMatrixOperationLib para poder crear una matriz del tipo Pascal la cual se encuentra definida en MATLAB.

Actores

Usuario

Flujo de eventos

Flujo básico

Actor	*Sistema*
Este caso de uso comienza cuando el usuario tiene instanciada la interfaz IMOCOMMatlabMatrixOperationLib y el usuario desea crear una matriz Pascal.	
El usuario ejecuta la función “Pascal” con los parámetros para definir las dimensiones de la matriz y el nombre de la matriz en donde se va a depositar el resultado.	
	El sistema crea la matriz, permaneciendo el resultado de la operación dentro de MATLAB.
Finaliza el caso de uso.	

Excepción

Si la interfaz no pudo ser instanciada, se notificará mediante un mensaje de error.
Si la matriz no pudo ser creada se notificará mediante un mensaje de error.

Precondiciones

El usuario deberá haber iniciado una sesión en MATLAB.

Diagrama de actividades

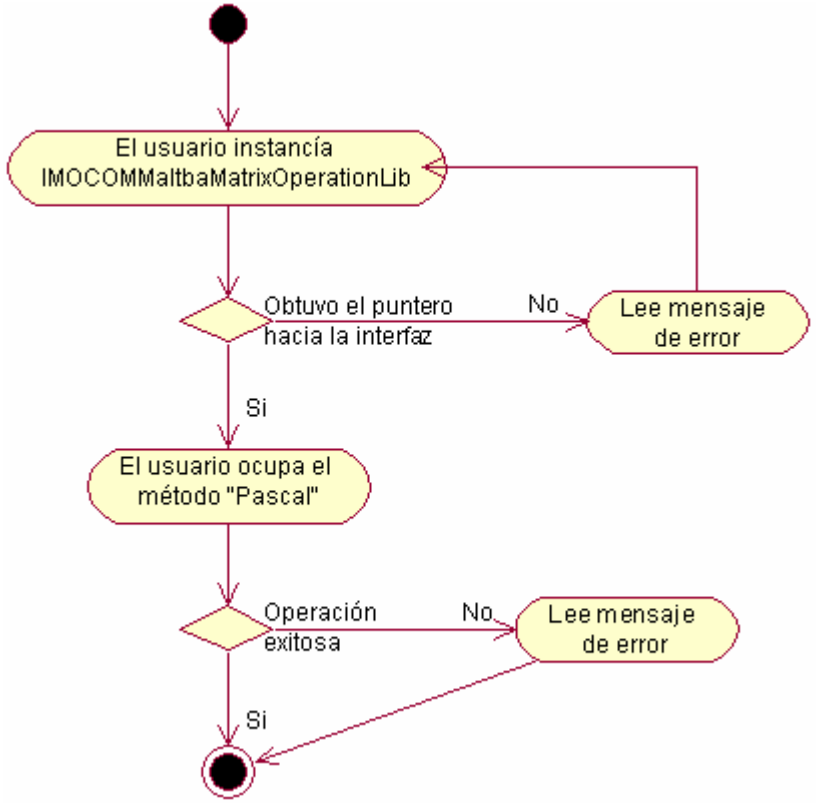


Diagrama de secuencia

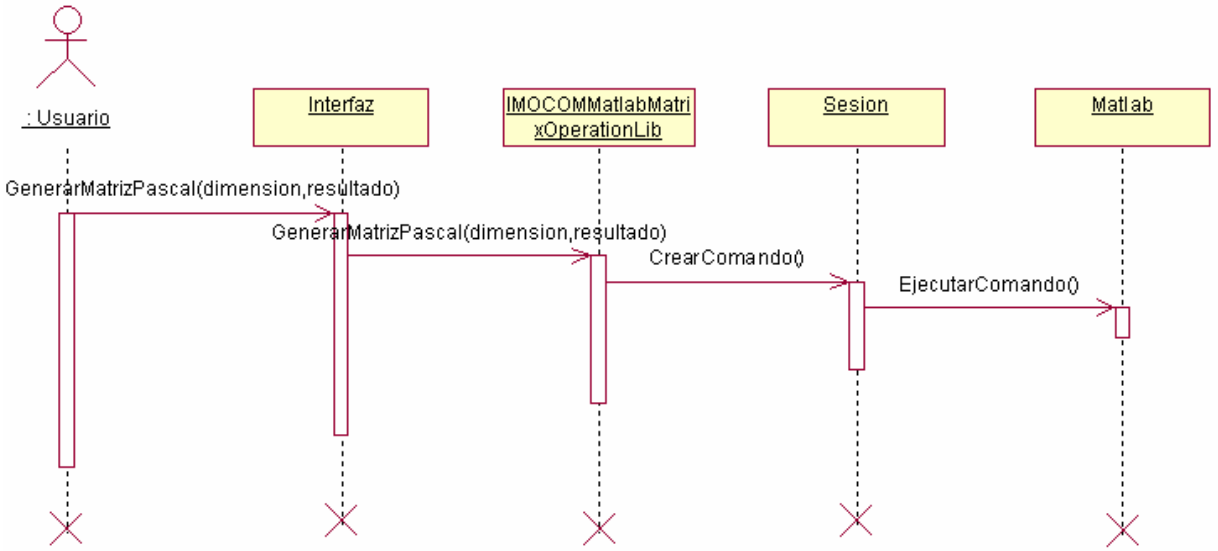


Diagrama de colaboración

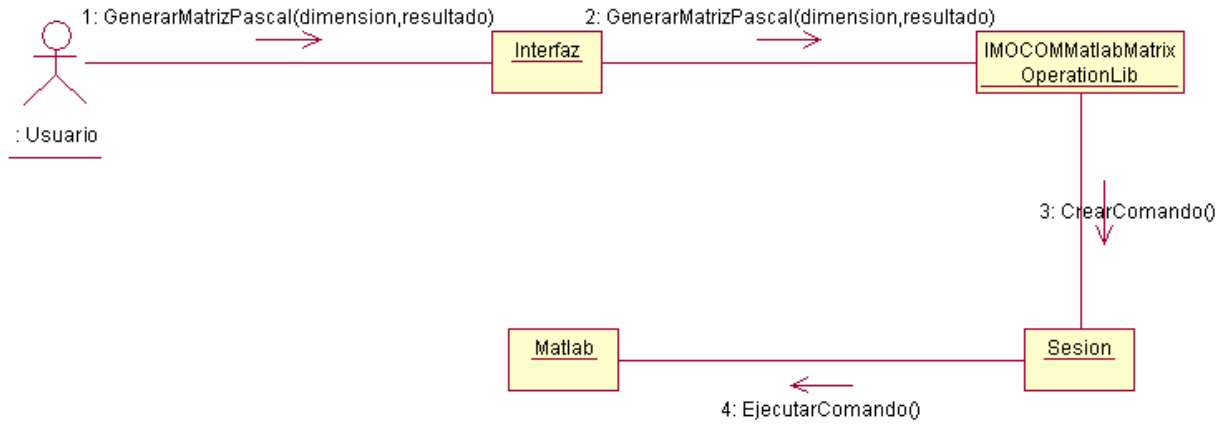


Diagrama de estados

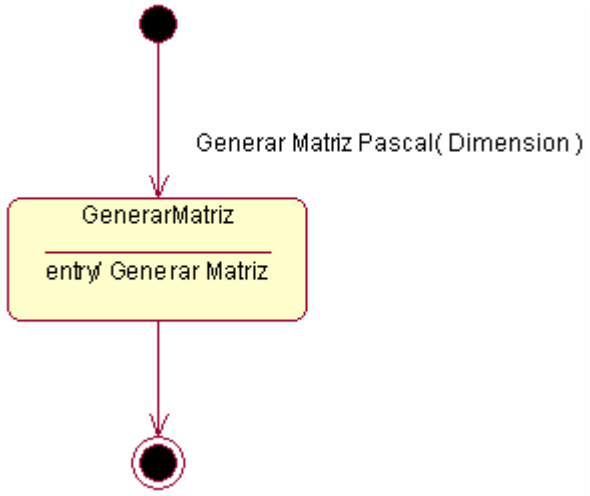
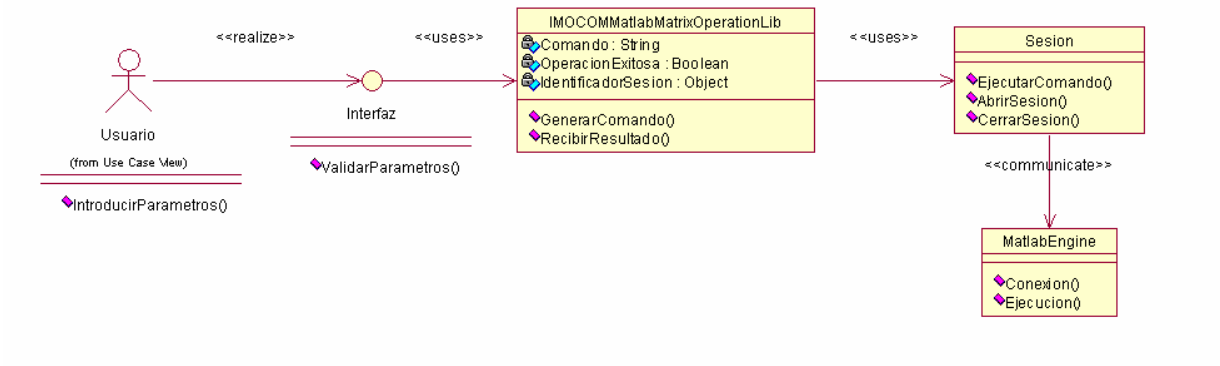


Diagrama de clases



Después de definir la manera en que se diseña el sistema, a continuación se muestran los pasos restantes que este proceso conlleva.

3.2.1.2. Diseño Arquitectónico

Se concluye, por todo lo anterior, que el diseño arquitectónico estuvo basado en una arquitectura de capas (ver Figura 3.3) y dentro de estas capas se tiene una arquitectura orientada a objetos.

3.2.1.3. Identificación de objetos

La identificación de objetos es una etapa importante del diseño, las siguientes clases son las más significativas dentro de los componentes de software.

- Clase ***IMOCOMMatlabMatrix***: permite a los usuarios crear y manipular matrices de manera local en un ambiente orientado a objetos.
- Clase ***IMOCOMMatlabMatrixOperationLib***: permite a los usuarios con cierta experiencia en MATLAB crear y manipular matrices de manera local.

- Clase ***ISOAPMatlabMatrix***: permite a los usuarios crear y manipular matrices de manera remota.
- Clase ***ISOAPMatlabMatrixOperationLib***: permite a los usuarios con cierta experiencia en MATLAB crear y manipular matrices de manera remota.
- Clase ***Sesion***: permite llevar el control de las sesiones sobre las cuales trabaja el usuario.
- Clase ***Interfaz***: provee la interfaz de programación para los usuarios de los componentes.
- Clase ***MATLAB***: es provisto por MATLAB.

3.2.1.4. Modelos de diseño

Para lo modelos de diseño se ocupó el Lenguaje Unificado de Modelado, apoyado por el Proceso Unificado Rational, esto con ayuda de la herramienta Rational Rose con la que se generaron los modelos que rigen a los componentes de software, como son casos de uso, diagramas de colaboración, de secuencia, de actividades y de clases.

El resto de la especificación de casos de uso para el diseño de los componentes de software, se encuentran en el Anexo 3.

4. Análisis y diseño de la aplicación de software

4.1. Análisis del sistema

De la misma manera en que los componentes de software tuvieron un proceso de análisis y diseño, la aplicación llevará un proceso similar, se presentará el documento de requisitos de usuario y la especificación de requerimientos de software.

4.1.1. Documento de requisitos de usuario

1. Introducción

En este documento se presenta el resultado de un trabajo previo de captura de requisitos respectos al sistema que se pretende desarrollar.

2. Aplicación de prueba "Manipulador de Matrices"

Se pretende construir un sistema de software el cual permita mostrar el aporte de los componentes de software local y distribuido en el desarrollo de software.

15.1. Requisitos

Con el desarrollo de este sistema de software, se pretende mostrar la utilidad que representa ocupar los componentes de software local y distribuido en aplicaciones donde se requiera el uso de matrices numéricas, para tal efecto se pretende que la aplicación brinde lo siguiente:

1. Crear un sistema de software en el que incluya el par de componentes, el local y el distribuido.
2. El sistema debe de tener la misma funcionalidad tanto para la aplicación local como para la aplicación distribuida[5].
3. El sistema debe soportar la generación de diversos tipos de matrices que MATLAB soporta.
4. El sistema debe soportar la ejecución de diversas operaciones empotradas en los componentes de software.

[5] Para esto se usará el patrón de diseño Modelo Vista Controlador.

5. El sistema debe tener una interfaz amigable sobre todo porque inicialmente va orientado al área educativa, aunque bien podrá ser utilizado por cualquier usuario que así lo requiera.

4.1.2. Especificación de Requerimientos de Software (ERS)

Introducción

Este apartado trata sobre la especificación de los requerimientos que el sistema de software *Manipulador de Matrices* tiene.

1. Propósito

El propósito de la especificación de requerimientos es delimitar claramente el dominio del sistema, de tal manera que se conozca de antemano las funcionalidades y las restricciones de los mismos. A su vez, este documento pretende sea la base para el diseño del sistema *Manipulador de Matrices*. Este documento servirá al desarrollador del sistema para limitar y comprender de manera clara la etapa de diseño y la aplicación del *Manipulador de Matrices*, a su vez, este documento servirá a los revisores de esta tesis para corroborar que los componentes cumplan con todas las características con las que fueron planeadas.

2. Ámbito del sistema

El desarrollo de este sistema nace de la necesidad de mostrar el impacto de estos componentes de software en el desarrollo de aplicaciones, se pretende dar a conocer, mediante el *Manipulador de Matrices*, la facilidad con la que se pueden crear matrices y manipularlas.

El sistema a desarrollar inicialmente va orientado a los maestros y estudiantes de licenciatura de la UTM, como ayuda para estudiar las matrices y sus operaciones, pudiendo ser utilizado también en otros niveles de la educación.

3. Definiciones, Acrónimos y Abreviaturas

3.1. Definiciones

Tabla 4.1. Definiciones para ERS de la aplicación.

Usuarios	Personas que harán uso del sistema *Manipulador de Matrices*, mediante la cual podrán crear diversos tipos de matrices y realizar operaciones sobre dichas matrices.

3.2. Acrónimos

ERS Especificación de Requisitos de Software

DRU Documento de Requerimientos de Usuario

3.3. Abreviaturas

Tabla 4.2. Abreviaturas para ERS de la aplicación.

Manipulador de Matrices	Sistema para la creación y manipulación de matrices numéricas
COM	*Component Object Model*, tecnología de componentes desarrollada por Microsoft, también conocida como *Active X*.
SOAP	*Simple Object Access Protocol*, es un protocolo para intercambiar mensajes entre sistemas de cómputo, principalmente los que están formados por componentes de software.
MATLAB	*Matrix Laboratory*, es un sistema orientado a matemáticas creado por *The Math Works* en 1984. Es un sistema de software muy usado en universidades, centros de investigación y actividades de ingeniería.

4. Referencias del Documento

- IEEE Recommended Practice for Software Requirements Specification. ANSI/IEEE std. 830, 1998
- Sommerville, Ian. Ingeniería de software. 6a. edición. Addison Wesley

5. Visión general del documento

Esta sección (ERS) está dividido en tres secciones: la introducción, la descripción general y por último los requisitos específicos.

En la introducción se muestra una visión general del documento, muestra la estructura básica para que el lector pueda ubicarse dentro del mismo. En la descripción general se muestra la funcionalidad básica de los componentes de software, sus limitantes, supuestos y dependencias que se encuentran relacionados con el diseño y aplicación del componente. Por último, la descripción de los requisitos se encuentra en la sección final del documento.

Descripción general

En esta sección se presenta una descripción de alto nivel del Manipulador de Matrices. Se presentarán las principales funciones que el sistema debe realizar, la información utilizada, las restricciones y otros factores que afecten el desarrollo del mismo.

6. Perspectivas del producto

El Manipulador de Matrices interactúa con MATLAB mediante los componentes de software ya mencionados y explicados, en una segunda versión podrá interactuar con otros componentes de software cuando estos sean liberados para la librería de componentes.

7. Funciones de los componentes de software

El Manipulador de Matrices tiene diversas tareas, las principales son las siguientes:

- Creación de matrices.
- Realización de operaciones sobre matrices.
- Conexión local.
- Conexión remota

A continuación se muestra una descripción más detallada de estas operaciones.

7.1. Creación de Matrices

En lo que respecta a la creación de matrices, el usuario tendrá la posibilidad de crear las siguientes matrices, en las cuales se puede determinar las dimensiones y el nombre de las mismas, las matrices que podrá crear son las siguientes:

- Matriz Hadamard
- Matriz Hilbert
- Matriz Hilbert Inversa
- Matriz Identidad
- Matriz Identidad no cuadrada
- Matriz Mágica
- Matriz Pascal
- Matriz Wilkinson
- Matriz aleatoria uniforme (con valores entre 0 y 1)
- Matriz aleatoria (con valores entre $-\infty$ y ∞)
- Matriz con valores iniciales de 1 en cada casilla.
- Matriz con valores iniciales de 0 en cada casilla
- Matriz Rosser
- Matriz definida por el usuario.

7.2. Operaciones Matriciales

Para la manipulación de matrices, es necesario mostrar en el sistema la flexibilidad que los componentes de software ofrecen, por tal motivo, es conveniente aplicar las variantes posibles a las operaciones matriciales, dando como resultado la división de estas operaciones en seis secciones, las cuales son ofrecidas por dicho sistema. Tales secciones y sus operaciones se enlistan a continuación:

- Operaciones que devuelven valores dobles.
  - Obtener condición recíproca.
  - Obtener determinante.
  - Obtener norma.
  - Obtener rango.
  - Obtener suma de la diagonal de la matriz.
- Operaciones que no ocupan parámetros de entrada.
  - Balance.
  - Descomposición en valor singular.
  - Descomposición Schur.
  - Diagonal de la matriz.
  - Factor Cholesky.
  - Factor LU.
  - Factor Qr.
  - Generar matriz Hankel.
  - Hessenberg.
  - Índices de la matriz.
  - Inversa.
  - Obtener condición recíproca como matriz.
  - Obtener determinante como matriz.
  - Obtener norma como matriz.
  - Obtener rango como matriz.
  - Obtener suma de la diagonal como matriz.
  - Obtener tamaño de la matriz.
  - Ortogonal.
  - Parte triangular alta de la matriz.
  - Parte triangular baja de la matriz.
  - Pseudo inversas.
  - Raíz cuadrada de la matriz.
  - Rotar 90 grados la matriz.
  - Sort.
  - Transpuesta.
  - Transpuesta no conjugada.

  - Vector con valores máximos de la matriz.
  - Vector con valores medios de la matriz.
  - Vector con valores mínimos de la matriz.
  - Voltear de arriba hacia abajo la matriz.
  - Voltear de izquierda a derecha la matriz.
- Operaciones que requieren otra matriz
  - Concatenación horizontal (destino-origen).
  - Concatenación horizontal (origen-destino).
  - Concatenación vertical (destino-origen).
  - Concatenación vertical (origen-destino).
  - Dividir matriz (destino-origen).
  - Dividir matriz (origen-destino).
  - Multiplicar matriz (destino-origen).
  - Multiplicar matriz (origen-destino).
  - Producto Kronecker (destino-origen).
  - Producto Kronecker (origen-destino).
  - Restar matriz (destino-origen).
  - Restar matriz (origen-destino).
  - Sumar matriz.
- Operaciones que requieren un parámetro entero.
  - Exponencial.
  - Producto acumulativo.
  - Suma acumulativa.
  - Voltear la matriz N dimensiones.
- Operaciones que devuelven una variable.
  - Devolver condición recíproca en una variable.
  - Devolver determinante en una variable.
  - Devolver norma en una variable.
  - Devolver rango en una variable.
  - Devolver suma de la diagonal de la matriz en una variable.
- Operaciones que requieren como parámetro una variable.
  - Dividir matriz entre variable (matriz / variable).
  - Multiplicar variable por matriz.
  - Producto Kronecker (matriz, variable).
  - Restar una variable (variable - matriz).
  - Restar a la matriz una variable (matriz - variable).
  - Sumar una variable a la matriz.

Con la aplicación de estas operaciones se pretende mostrar la utilidad de los componentes en cuanto a las operaciones que soportan.

7.3. Conexión local

El sistema ofrece la opción de elegir una conexión local con MATLAB, es decir, una conexión en la que se tenga MATLAB instalado en la misma computadora. Esto para probar el funcionamiento del componente COM.

7.4. Conexión remota

El sistema también ofrece la opción de elegir una conexión remota con MATLAB, es decir, una conexión en la que no se tenga MATLAB instalado en la misma computadora. Esto para probar el funcionamiento del componente SOAP.

8. Características de los usuarios

Los usuarios son aquellas personas interesadas en ocupar el sistema para generar matrices y poder trabajar sobre ellas.

El Manipulador de Matrices ofrecerá una interfaz intuitiva para que los usuarios novatos en sistemas de cómputo puedan ocupar dicho sistema.

9. Restricciones

El sistema solo está orientado a la creación y manipulación de matrices, no se pueden realizar operaciones matemáticas diferentes a las anteriormente mencionadas.

La principal restricción del sistema es que, en su primera versión, el sistema únicamente trabaja con matrices de dos dimensiones, esto por la complejidad del sistema, pero sobre todo porque rebasan los objetivos de la tesis. En versiones futuras se podrá incluir el manejo de matrices multidimensionales.

10. Suposiciones y dependencias

10.1. Suposiciones

Se asume que los requisitos descritos en este documento son estables una vez que es aprobado como tema de tesis. Cualquier petición de cambio en las especificaciones del proyecto será propuesta como trabajo futuro.

10.2. Dependencias

El sistema es dependiente totalmente de MATLAB. En el caso de una conexión remota[6] se suma la dependencia de una computadora que ofrezca los servicios de MATLAB y que se encuentre conectada a la red.

11. Requisitos específicos

En este apartado se muestran los requisitos funcionales que debe cumplir el sistema de software para que pueda ser utilizado de la manera en que fueron planeados. Todos los requisitos aquí expuestos son ***esenciales***, no se aceptará que este proyecto sea válido si no cumple con ***todos*** los requisitos aquí mostrados.

12. Requisitos funcionales

12.1. Conexiones locales y remotas

Req (1) El usuario podrá crear conexiones locales del sistema con MATLAB.

Req 1.1 El usuario selecciona y elige la conexión local a realizar.

Req 1.2 El sistema realiza la conexión.

Req 1.3 Si la conexión no tuvo éxito, el sistema mandará un mensaje de error correspondiente.

Req (2) El usuario podrá crear conexiones remotas del sistema con MATLAB.

Req 2.1 El usuario selecciona la conexión remota a realizar.

Req 2.2 Especifica el nombre o dirección IP del servidor de servicios de MATLAB y elige realizar la conexión.

Req 2.3 El sistema realiza la conexión.

Req 2.4 Si la conexión no tuvo éxito, el sistema mandará un mensaje de error correspondiente.

12.2. Creación de matrices

Req (3) El usuario podrá crear las matrices que vienen descritas en el apartado anterior, de tal manera que pueda definir las dimensiones de las matrices, así como el nombre que las mismas van a tener.

[6] Uso del componente SOAP desarrollado en este trabajo de tesis.

Req 3.1 Seleccionar la matriz a crear.

Req 3.2 Seleccionar las dimensiones de la matriz.

Req 3.3 Asignarle a la matriz un nombre, en caso de que no se le asigne un nombre, el sistema le asignará uno aleatorio, en caso que el nombre que el usuario le haya asignado ya esté siendo usado, el sistema mandará un mensaje de error.

Req 3.4 Crear la matriz y mostrar el nombre de la matriz en un historial desde donde podrá ser consultado.

12.3. Manipulación de matrices

Req (4) El usuario podrá realizar todas las operaciones sobre las matrices descritas en el apartado anterior, de esta manera, el sistema mostrará todas las posibles operaciones que soporte y el usuario decidirá qué operación realizar.

Req 4.1 Seleccionar y visualizar la matriz sobre la que se quiera hacer la operación.

Req 4.2 Elegir el tipo de operación que se desea realizar.

Req 4.3 El sistema realiza la operación y devuelve el tipo de dato correcto, ya sea matriz, variable o valor numérico simple.

Req 4.4 En caso de que la operación no se pueda realizar, el sistema mostrará el mensaje de error correspondiente.

12.4. Configuración de la interfaz de usuario

Req (5) El usuario podrá modificar el aspecto de la interfaz de usuario, de tal manera que pueda visualizar los valores de las matrices de una manera adecuada.

Req 5.1 Seleccionar el valor dentro de la interfaz de usuario que se va a modificar.

Req 5.2 Modificar el valor seleccionado.

Req 5.3 En caso de que el valor no pueda ser modificado, el sistema mandará un mensaje de error correspondiente.

12.5. Consulta de resultados

Req (6) El usuario podrá consultar todas las matrices que han sido generadas en la sesión en la cuál se encuentra trabajando.

Req 6.1 Seleccionar la matriz a consultar dentro del historial de matrices existentes.

Req 6.2 En caso de que la matriz no pueda visualizarse, el sistema mostrará el mensaje de error correspondiente.

13. Interfaces Externos

13.1. Interfaz de usuario

La interfaz de usuario va a ser la estándar para aplicaciones de escritorio. El manejo del programa se va a realizar a través del teclado y del ratón.

13.2. Interfaces de hardware

No se han definido.

13.3. Interfaces de software

Las interfaces de software del sistema van a ser las que ofrecen los componentes de software que se ocupan en este sistema.

13.4. Interfaces de comunicación

En la parte en la que se ocupan los componentes SOAP, es decir, una conexión remota, se tiene planeado como interfaz de comunicación la red Ethernet de la Universidad Tecnológica de la Mixteca.

14. Requisitos de rendimiento

Al ser MATLAB un sistema bastante estable, se prevé que no existan problemas en cuanto al número de conexiones tanto locales como remotas.

15. Requisitos de desarrollo

El desarrollo de este proyecto estará basado en la tecnología orientada a objetos, siguiendo todas las guías de análisis, diseño e implementación orientada a objetos

16. Requisitos tecnológicos

Para utilizar el sistema, usando una conexión remota, se necesitará una PC con una configuración mínima de:

Procesador: Pentium III 750 Mhz.

Memoria: 128 Mb.

Espacio libre en disco: 10 Mb.

Tarjeta Ethernet o Módem.

Acceso a Internet.

Sistema operativo: Windows 98, 2000, XP.

Para utilizar el sistema usando una conexión local, se necesitará una PC con una configuración mínima de:

Procesador: Pentium III 750 Mhz.

Memoria: 128 Mb.

Espacio libre en disco: 1 Gb. (para la instalación de MATLAB).

Sistema Operativo: Microsoft Windows 98, Me, XP Profesional (recomendado).

4.2. Diseño del sistema

La parte del diseño del sistema, es una consecuencia del apartado anterior, el análisis del sistema nos sirve para que tengamos una idea clara de lo que se va a desarrollar, de la misma manera, mediante el diseño del sistema permite conocer el cómo se va a desarrollar el mismo.

Por principio de cuentas, se va a realizar la primera descomposición del sistema, esta descomposición va a ser un modelo de capas, la descomposición en capas es similar a la descomposición que se muestra en la Figura 3.2, del capítulo anterior.

Estas capas a su vez van a ser divididas en otras subcapas, y de esta manera se podrá obtener servicios, procesos, etc. que van a gobernar el sistema.

La Figura 4.1 muestra una descripción gráfica más detallada de las capas de los componentes.

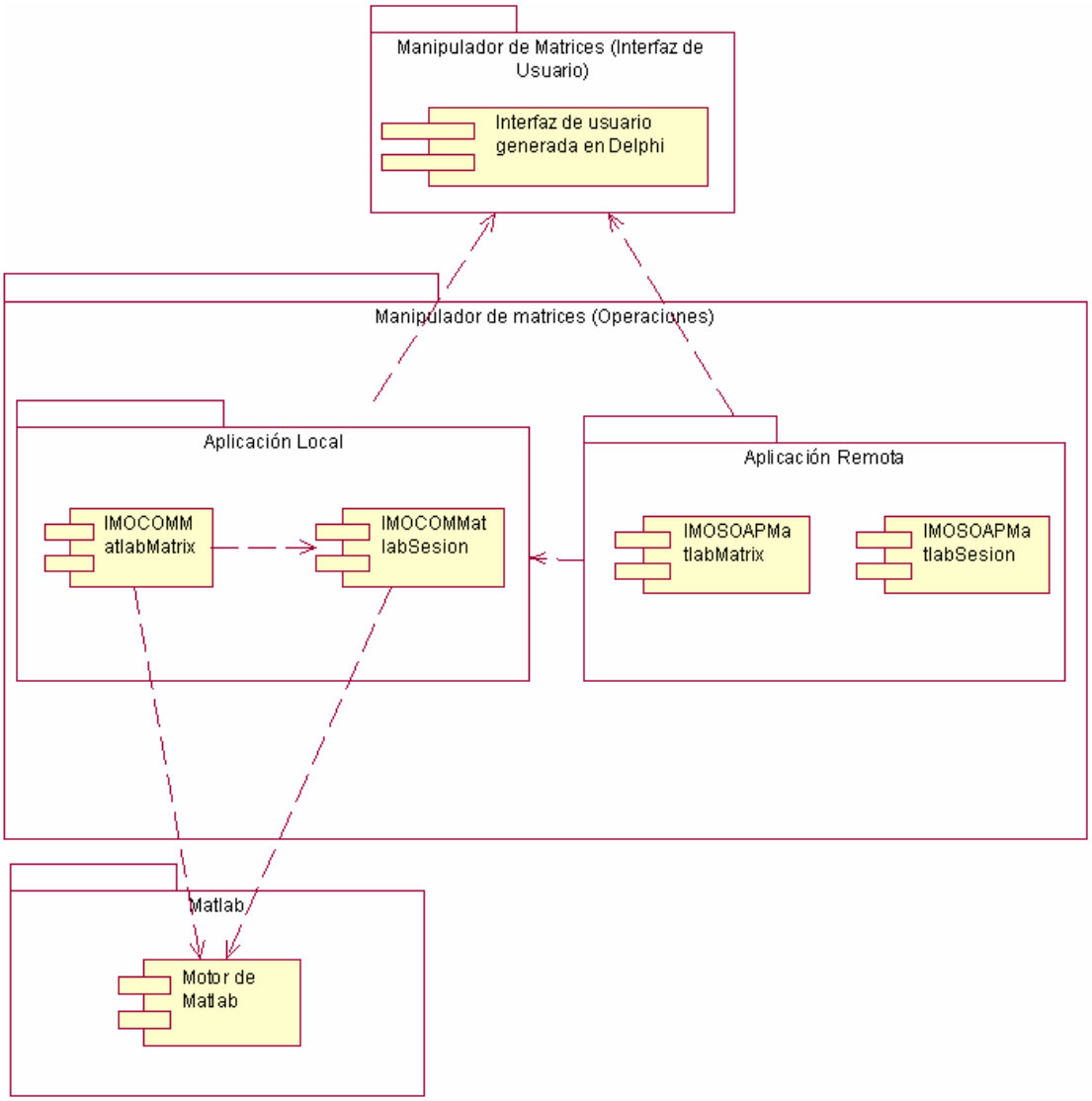


Figura 4.1. Vista física de las capas del sistema

4.2.1. Diseño orientado a objetos

Este sistema, al igual que los componentes de software, está diseñado por la metodología orientada a objetos que incluye el contexto y uso del sistema, el diseño arquitectónico, la identificación de objetos y los modelos de diseño [36] . Estos pasos se llevarán a cabo siguiendo la notación de UML y apoyándose en la metodología RUP.

4.2.1.1. Contexto y casos de uso del sistema

En el modelo de paquetes, que se visualiza en la Figura 4.2, se fragmenta el sistema en sistemas más pequeños, es decir, ayuda a dividir el software en subsistemas y mostrar las dependencias.

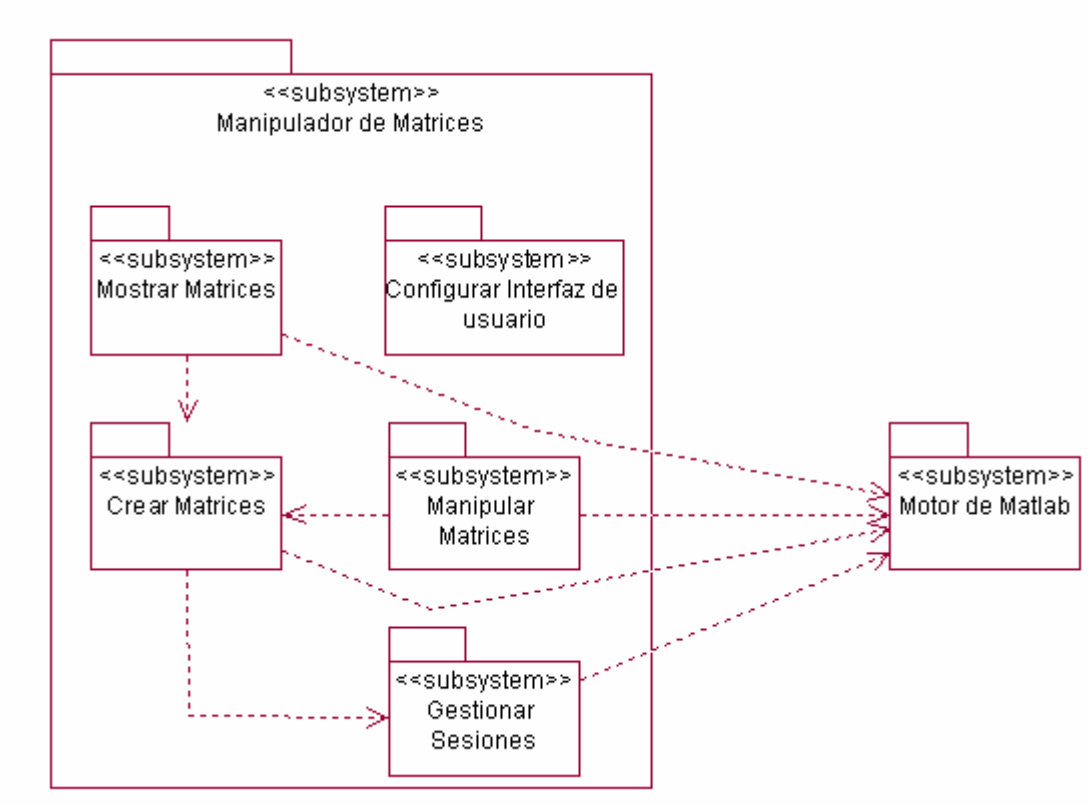


Figura 4.2. Contexto del Manipulador de Matrices.

En este modelo se muestra la dependencia del sistema con MATLAB, a su vez, se muestran las dependencias entre los subsistemas.

Por otro lado, para observar las interacciones del sistema con su entorno se utilizan los casos de uso, en la Figura 4.3 se muestra el diagrama de casos de uso del sistema.

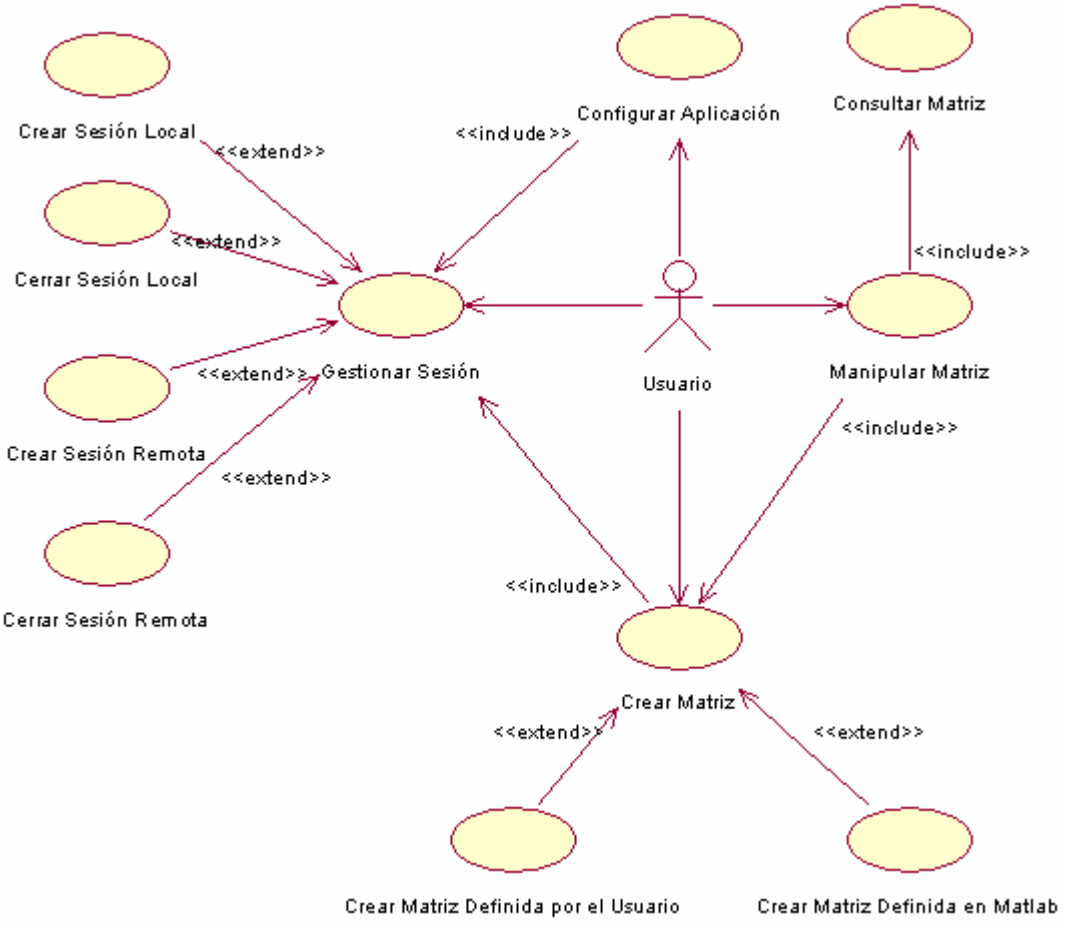


Figura 4.3. Modelo de Utilización del Manipulador de Matrices.

En este modelo, los casos de uso más importante son la gestión de sesión, la creación de matrices y la manipulación de las mismas.

4.2.1.1.1. Especificación de casos de uso (ECU)

Como en el diseño de los componentes del capítulo anterior, en este apartado también solo se muestran dos casos de uso, el total de ellos se pueden encontrar en el archivo *extension de casos de uso.pdf* en el CD-ROM adjunto en este trabajo de tesis.

Especificación de caso de uso: Crear Matriz definida por el Usuario.
Versión 1.2

Historial de revisiones

Fecha	Versión	Descripción	Autor
9/Marzo/2005	1.0	Creación del Documento	Felipe de Jesús García Pérez
15/Marzo/2005	1.1	Introducción de primeros diagramas	Felipe de Jesús García Pérez
21/Marzo/2005	1.2	Documento completado	Felipe de Jesús García Pérez

Objetivo
Crear una matriz en el sistema donde el usuario pueda definir los valores de la misma.

Breve descripción
El usuario podrá crear matrices en donde él defina los valores que la matriz tendrá, el nombre de la matriz se va a almacenar en un historial donde podrá ser consultada.

Actores
Usuario

Flujo de eventos
Flujo básico

Actor	*Sistema*
Este caso de uso comienza cuando el usuario ingresa al sistema y abre una sesión.	
	El sistema realiza el caso de uso Gestionar sesión.
	El sistema abre una ventana para trabajar sobre la sesión recién abierta.
El usuario escoge la opción crear una matriz definida por el usuario	
	El sistema muestra una nueva pantalla y solicita la introducción de las dimensiones de la matriz y el nombre de la matriz
El usuario introduce las dimensiones y el nombre de la matriz	
	El sistema muestra una nueva pantalla para la introducción de datos de la matriz.

El usuario introduce los valores que debe contener la matriz.	
	Si no hay errores, el sistema crea la matriz en MATLAB y manda el nombre de la misma al historial de matrices creadas.
Finaliza el caso de uso.	

Excepciones

Si la sesión no se puede abrir, se notificará al usuario mediante un mensaje de error.
Si la matriz no pudo ser creada se notificará mediante un mensaje de error.

Precondiciones

El usuario deberá haber ejecutado el sistema.

Diagrama de actividades

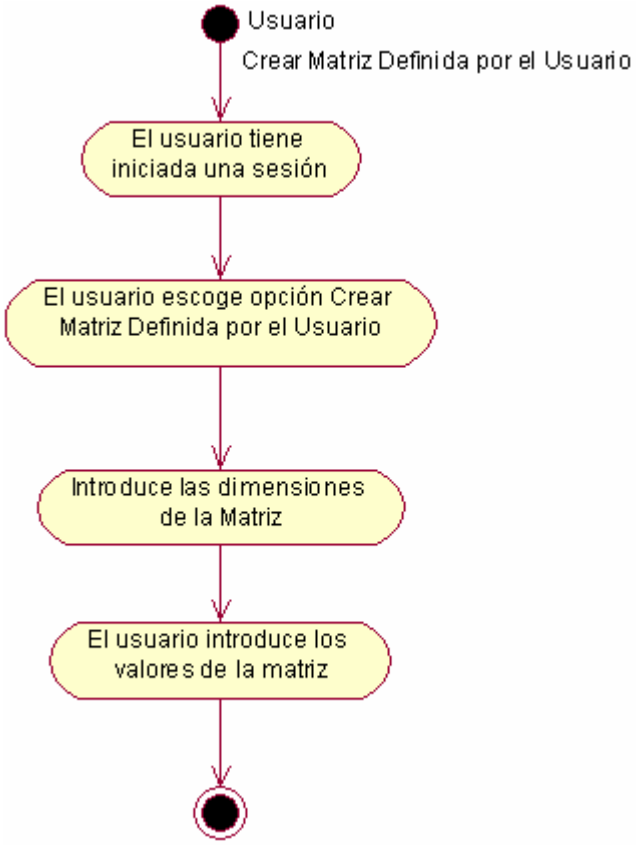


Diagrama de secuencia

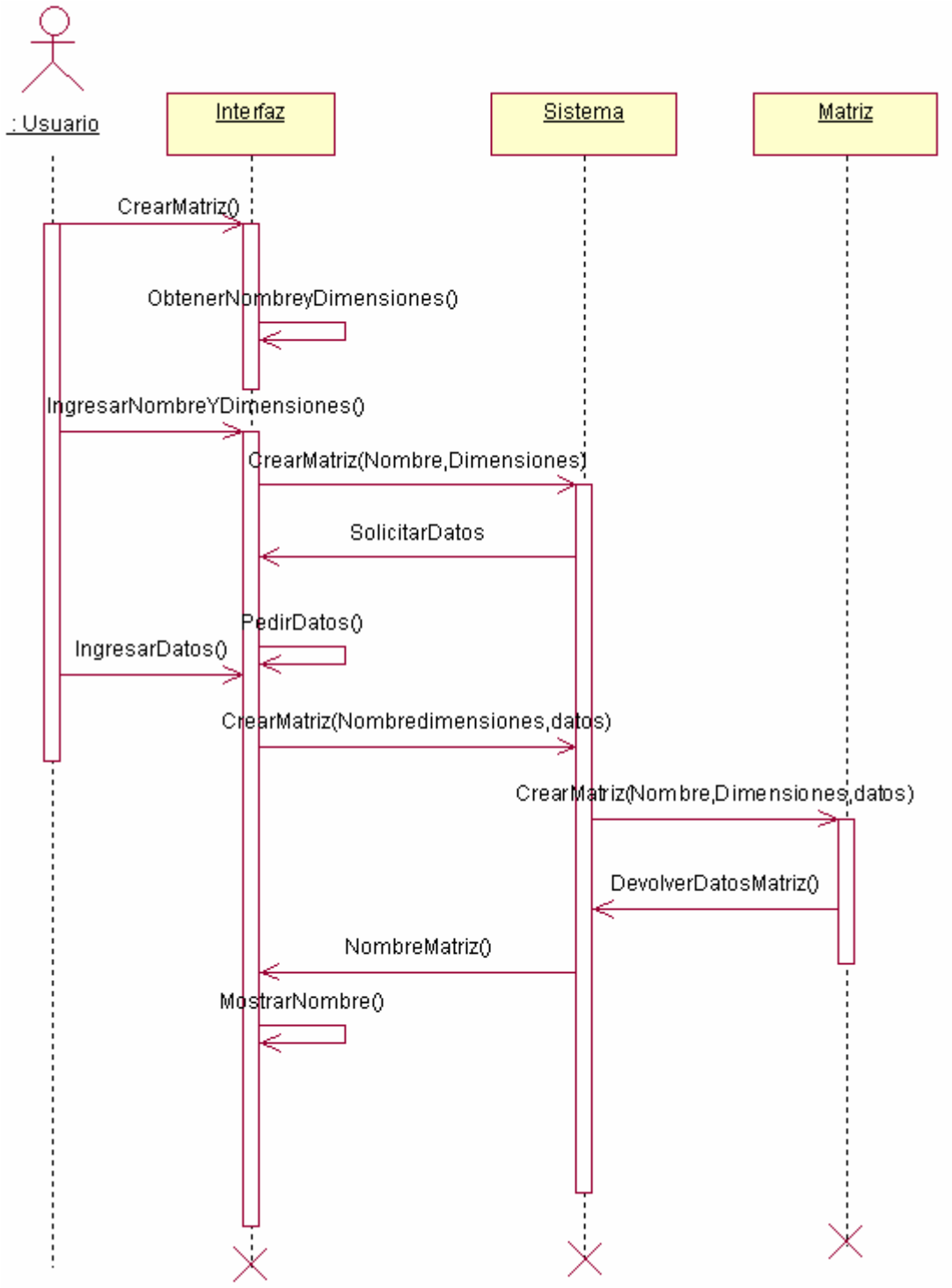


Diagrama de colaboración

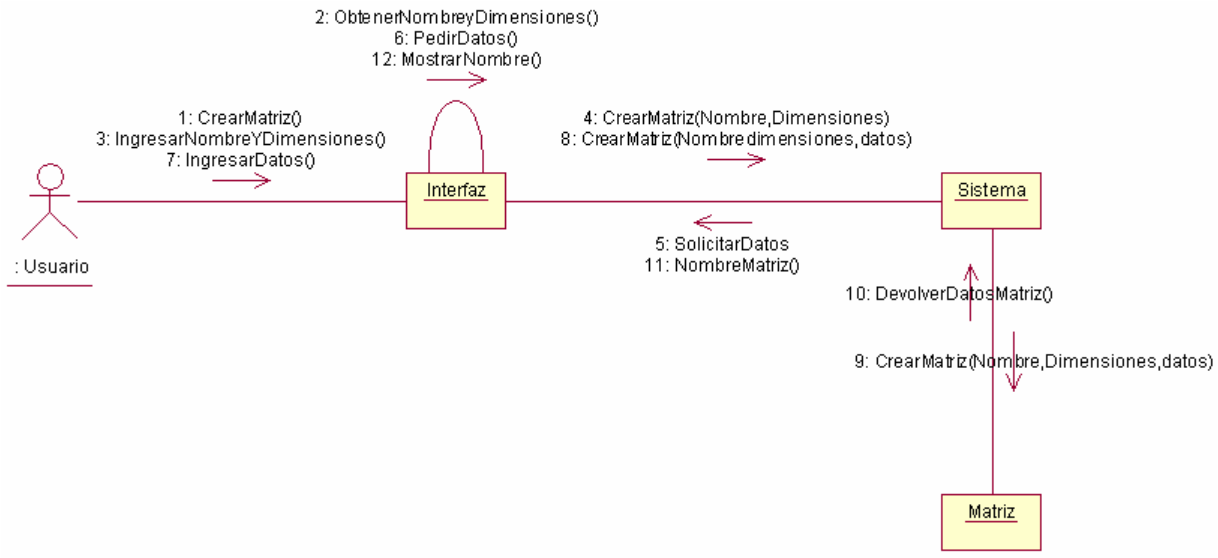


Diagrama de estados

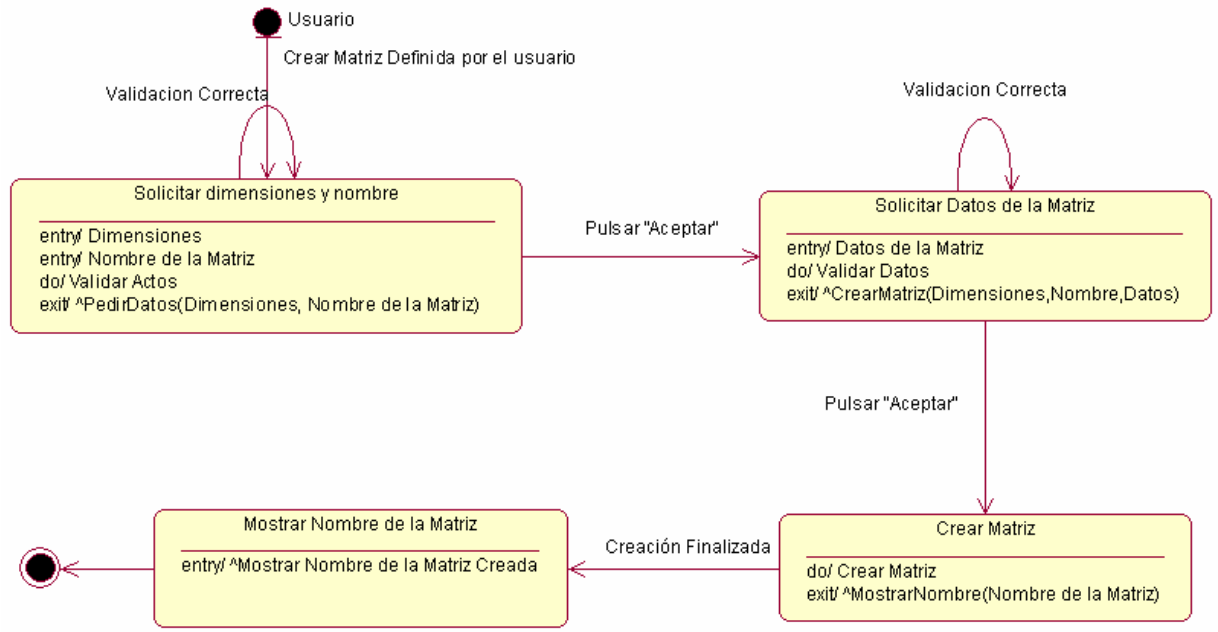
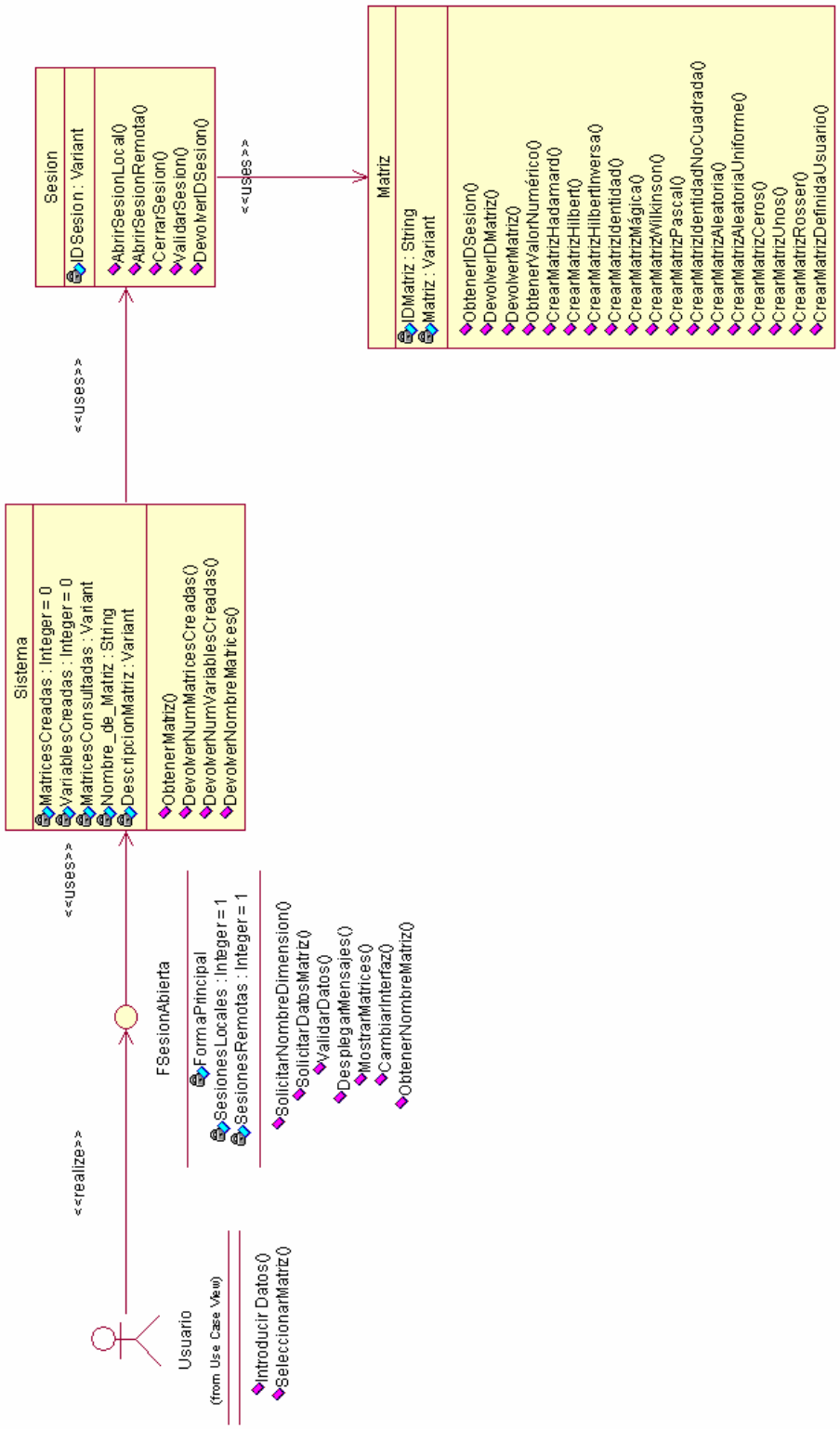


Diagrama de clases



Especificación de caso de uso: Manipular matriz
Versión 1.2

Historial de revisiones

Fecha	Versión	Descripción	Autor
9/Marzo/2005	1.0	Creación del Documento	Felipe de Jesús García Pérez
15/Marzo/2005	1.1	Introducción de primeros diagramas	Felipe de Jesús García Pérez
21/Marzo/2005	1.2	Documento completado	Felipe de Jesús García Pérez

Objetivo

Realizar operaciones sobre matrices.

Breve descripción

El usuario podrá manipular matrices para realizar operaciones sobre las mismas, el resultado de estas operaciones puede tener tres formatos: valores numéricos, matrices y variables.

Actores

Usuario

Flujo de eventos

Flujo básico

Actor	*Sistema*
Este caso de uso inicia cuando el usuario ha ejecutado el sistema y ha creado al menos una matriz.	
El usuario selecciona la matriz a manipular.	
	El sistema realiza el caso de uso **consultar Matriz**.
El usuario solicita la lista de operaciones matriciales existentes.	
	El sistema despliega la lista de operaciones posibles a realizar
El usuario escoge la operación requerida.	
	El sistema realiza la operación y devuelve el resultado requerido.
Finaliza el caso de uso.	

Excepción

Si la sesión no se puede abrir, se notificará al usuario mediante un mensaje de error.
Si la matriz no pudo ser creada se notificará mediante un mensaje de error.

Si la operación no pudo ser realizada se notificará al usuario mediante un mensaje de error.

Precondiciones

El usuario deberá haber ejecutado el sistema.

Diagrama de actividades

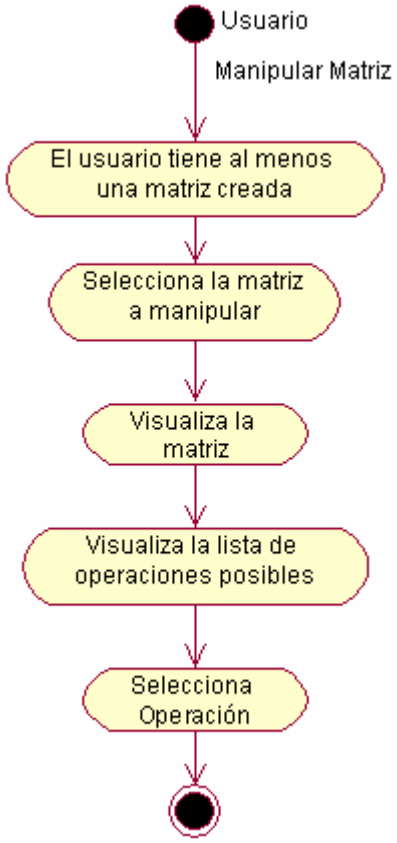


Diagrama de secuencia

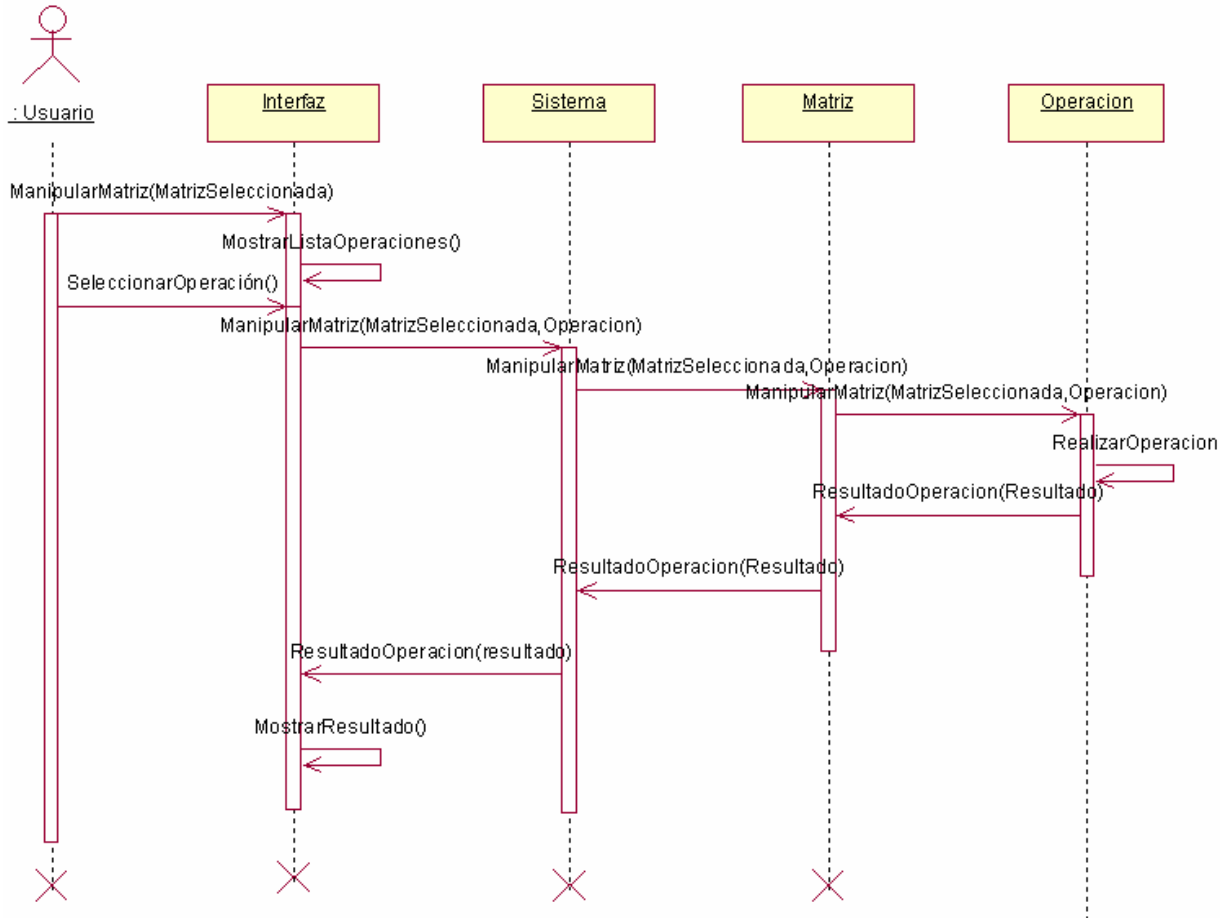


Diagrama de colaboración

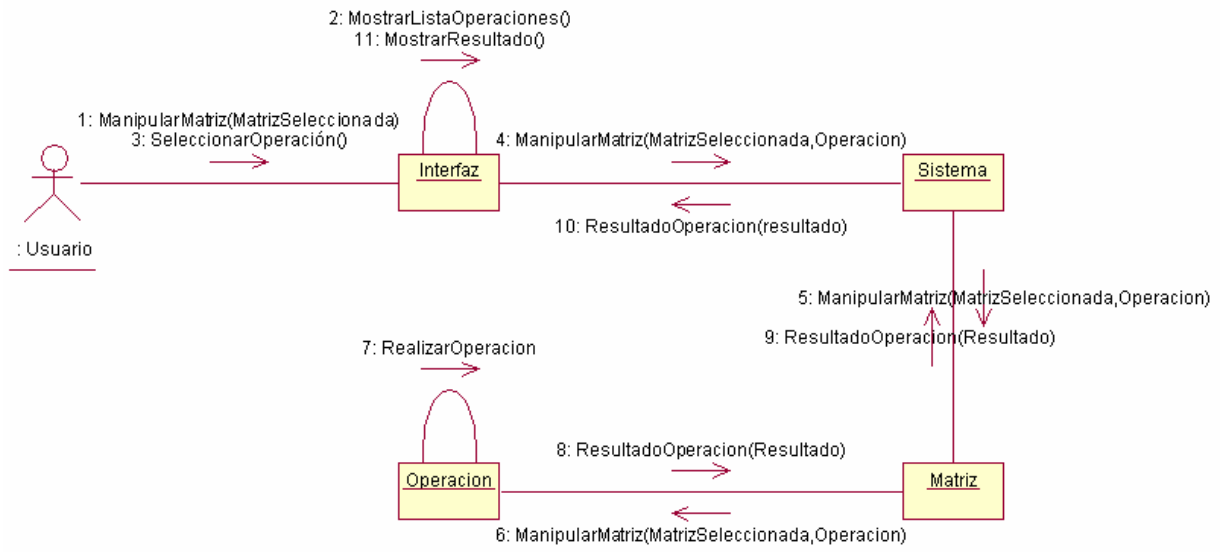


Diagrama de estados

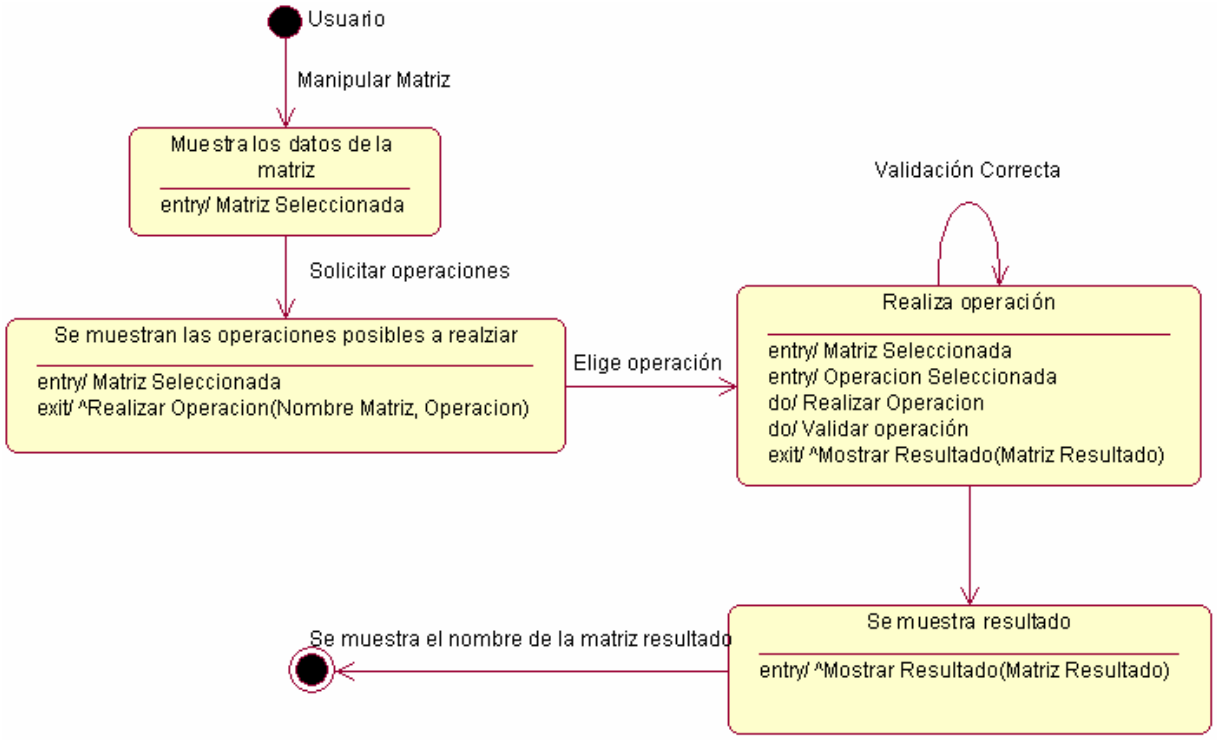
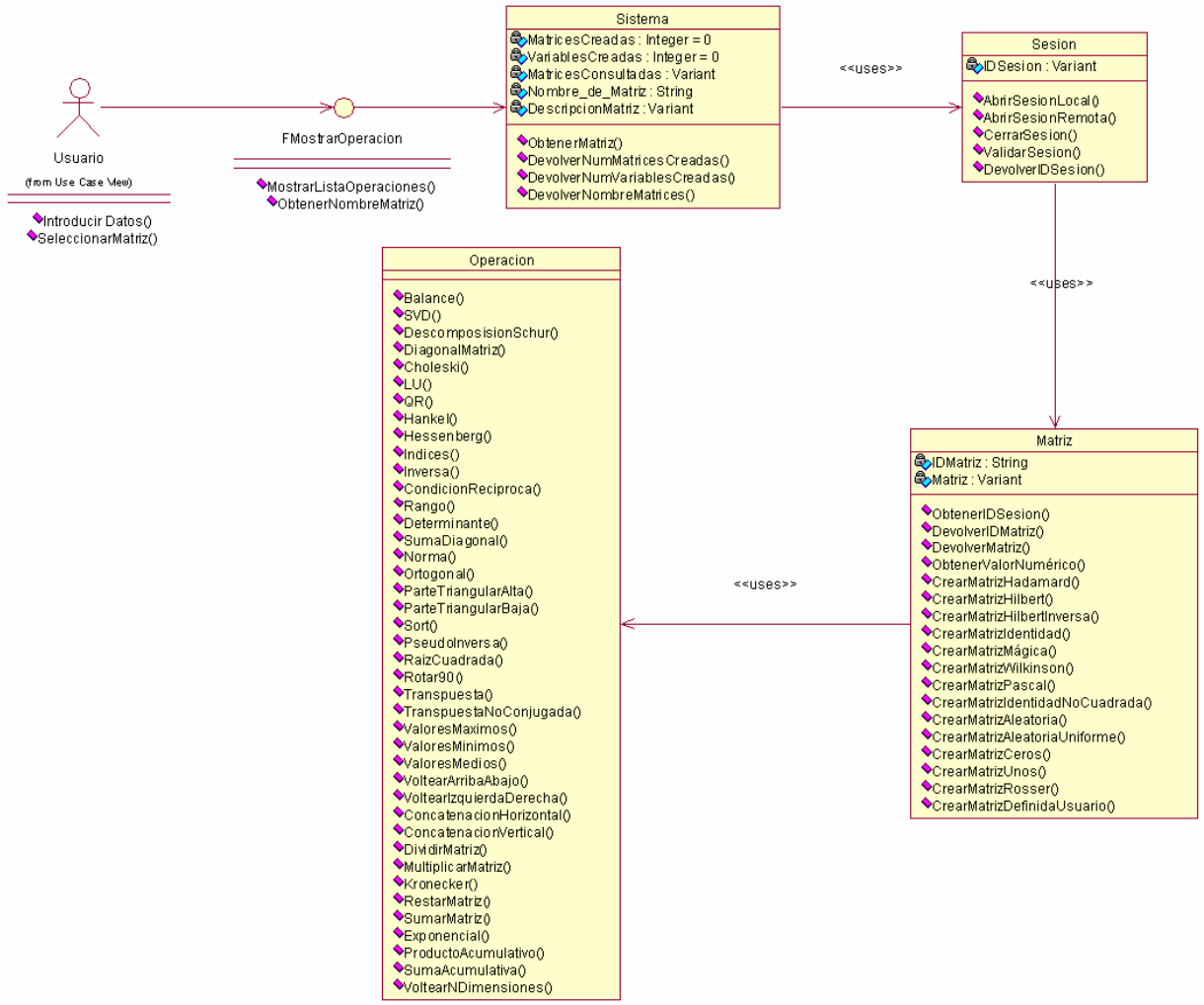


Diagrama de clases



A continuación, se prosigue el proceso de diseño con los apartados de Diseño Arquitectónico, Identificación de Objetos y por último Modelos de Diseño.

4.2.1.2. Diseño Arquitectónico.

Al igual que en el diseño de los componentes de software, esta etapa del diseño del sistema se ocupa un modelo arquitectónico de capas (ver Figura 4.1) y dentro de estas capas se utilizó una arquitectura orientada a objetos.

4.2.1.3. Identificación de Objetos.

Después del proceso anterior, se pueden definir una serie de objetos que van a regir el sistema, los objetos más importantes son:

- ***Interfaz***: Este objeto va a depender del entorno de desarrollo del sistema, en este caso las Formas que el entorno (Delphi 7.0) nos ofrece van a ser los objetos de este tipo.

- ***Matriz***: Este objeto nos va a permitir la creación de matrices.
- ***Operacion***: Este objeto va a permitir la manipulación de las diversas matrices creadas en el sistema
- ***Sesion***: Abre las sesiones locales y remotas de MATLAB.
- ***Sistema***: Provee la interfaz y operaciones para llevar el control de las matrices y sesiones creadas.

4.2.1.4. Modelos de diseño.

Los modelos de diseño muestran las clases y los objetos del sistema, en este caso, se ocupó la notación del Lenguaje Unificado de Modelado, apoyados por la herramienta Rational Rose. Se incluyeron todos los tipos de diagramas como son: casos de usos, diagramas de colaboración, diagramas de secuencia, diagrama de actividades y diagramas de clases.

5. Implementación y pruebas de los componentes de software

Después de tratar en los capítulos anteriores el análisis y el diseño de los componentes, en este capítulo se aborda la implementación y pruebas de los mismos.

5.1. Implementación de los componentes de software

La implementación de los componentes COM y SOAP se desarrolló mediante el lenguaje de programación Object Pascal utilizando el ambiente de desarrollo Delphi en su versión 7.0, todo esto en base a lo expuesto en capítulos anteriores de esta tesis.

5.1.1. Implementación de las interfaces de software

Como se mencionó con anterioridad, tanto el componente COM como el SOAP están conformados por un par de interfaces de software.

A continuación, se muestran todos los métodos, junto con sus parámetros de entrada y de salida de las interfaces que conforman el par de componentes.

En la Tabla 5.1 se presentan todos los métodos que conforman la interfaz IMOMatlabMatrix.

Tabla 5.1. Métodos de la interfaz IMOMatlabMatrix

IMOMatlabMatrix			
Nombre del método	***Parámetros de entrada***	***Tipo de parámetro***	***Salida***
AddMatrix	Matrix	IMOMatlabMatrix	IMOMatlabMatrix
	CreateNew	Boolean	
AddVar	VarName	String	String
	CreateNew	Boolean	
	NewMatrixName	String	
Balance	CreateNew	Boolean	IMOMatlabMatrix
CholeskyFactor	CreateNew	Boolean	IMOMatlabMatrix
CumulativeAdd	Dimension	Integer	IMOMatlabMatrix

	CreateNew	Boolean	
CumulativeProd	Dimension	Integer	IMOMatlabMatrix
	CreateNew	Boolean	
DiagonalOfMatrix	CreateNew	Boolean	IMOMatlabMatrix
DivideIntoMatrix	Matrix	IMOMatlabMatrix	IMOMatlabMatrix
	CreateNew	Boolean	
DivideIntoVar	VarName	String	String
	CreateNew	Boolean	
	NewMatrixName	String	
DivideMatrix	Matrix	IMOMatlabMatrix	IMOMatlabMatrix
	CreateNew	Boolean	
DivideVar	VarName	String	String
	CreateNew	Boolean	
	NewMatrixName	String	
Exponential	Power	Integer	IMOMatlabMatrix
	CreateNew	Boolean	
Find	CreateNew	Boolean	IMOMatlabMatrix
FlipLeftToRight	CreateNew	Boolean	IMOMatlabMatrix
FlipNDimensions	Dimension	Integer	IMOMatlabMatrix
	CreateNew	Boolean	
FlipUpToDown	CreateNew	Boolean	IMOMatlabMatrix
GenerateIdentityMatrix	Dimension	Integer	IMOMatlabMatrix
	CreateNew	Boolean	
GenerateHadamardMatrix	Dimension	Integer	IMOMatlabMatrix
	CreateNew	Boolean	
GenerateHadamardMatrixAsVar	Dimension	Integer	String
	CreateNew	Boolean	
	NewVarName	String	
GenerateHilbertMatrix	Dimension	Integer	IMOMatlabMatrix
	CreateNew	Boolean	
GenerateHilbertMatrixAsVar	Dimension	Integer	String
	CreateNew	Boolean	
	NewVarName	String	
GenerateIdentityMatrixAsVar	Dimension	Integer	String
	CreateNew	Boolean	
	NewVarName	String	
GenerateIdentityNotSquareMatrix	X	Integer	IMOMatlabMatrix
	Y	Integer	

	CreateNew	Bolean	
GenerateIdentityNotSquareMatrixAsVar	X	Integer	String
	Y	Integer	
	CreateNew	Boolean	
	NewVarName	String	
GenerateInverseHilbertMatrix	Dimension	Integer	IMOMatlabMatrix
	CreateNew	Boolean	
GenerateInverseHilbertMatrixAsVar	Dimension	Integer	String
	CreateNew	Bolean	
	NewVarName	String	
GenerateMagicMatrix	Dimension	Integer	IMOMatlabMatrix
	CreateNew	Boolean	
GenerateMagicMatrixAsVar	Dimension	Integer	String
	CreateNew	Boolean	
	NewVarName	String	
GenerateOnesMultiDimMatrix	Dims	Variant	IMOMatlabMatrix
	CreateNew	Boolean	
GenerateOnesMultiDimMatrixAsVar	Dims	Variant	String
	NewVarName	Boolean	
GenerateOnesXYMatrix	X	Integer	IMOMatlabMatrix
	Y	Integer	
	CreateNew	Bolean	
GenerateOnesXYMatrixAsVar	X	Integer	String
	Y	Integer	
	NewVarName	String	
GenerateOnesXYZMatrixAsVar	X	Integer	String
	Y	Integer	
	Z	Integer	
	NewVarName	String	
GenerateOnesXYZMatrix	X	Integer	IMOMatlabMatrix
	Y	Integer	
	Z	Integer	
	CreateNew	Bolean	
GeneratePascalMatrix	Dimension	Integer	IMOMatlabMatrix
	CreateNew	Boolean	
GeneratePascalMatrixAsVar	Dimension	Integer	String
	NewVarName	String	
GenerateRandomMulti-	Dims	Variant	IMOMatlabMatrix

DimMatrix	CreateNew	Boolean	
GenerateRandomMulti-DimMatrixAsVar	Dims	Variant	String
	NewVarName	Boolean	
GenerateRandomXYMatrix	X	Integer	IMOMatlabMatrix
	Y	Integer	
	CreateNew	Bolean	
GenerateRandomMatrixXYAsVar	X	Integer	String
	Y	Integer	
	NewVarName	String	
GenerateRandomXYZMatrix	X	Integer	IMOMatlabMatrix
	Y	Integer	
	Z	Integer	
	CreateNew	Bolean	
GenerateRandomXYZMatrixAsVar	X	Integer	String
	Y	Integer	
	Z	Integer	
	NewVarName	String	
GenerateRosserMatrix	CreateNew	Bolean	IMOMatlabMatrix
GenerateRosserMatrixAsVar	NewVarName	String	String
GenerateUniformRandomMultiDimMatrix	Dims	Variant	IMOMatlabMatrix
	CreateNew	Boolean	
GenerateUniformRandomMultiDimMatrixAsVar	Dims	Variant	String
	NewVarName	Boolean	
GenerateUniformRandomXYMatrix	X	Integer	IMOMatlabMatrix
	Y	Integer	
	CreateNew	Bolean	
GenerateUniformRandomXYMatrixAsVar	X	Integer	String
	Y	Integer	
	NewVarName	String	
GenerateUniformRandomXYZMatrix	X	Integer	IMOMatlabMatrix
	Y	Integer	
	Z	Integer	
	CreateNew	Bolean	
GenerateUniformRandomXYZMatrixAsVar	X	Integer	String
	Y	Integer	
	Z	Integer	
	NewVarName	String	

GenerateWilkinsonMatrix	Dimension	Integer	IMOMatlabMatrix
	CreateNew	Boolean	
GenerateWilkinsonMatrixAsVar	Dimension	Integer	String
	NewVarName	String	
GenerateZerosMultiDimMatrix	Dims	Variant	IMOMatlabMatrix
	CreateNew	Boolean	
GenerateZerosMultiDimMatrixAsVar	Dims	Variant	String
	NewVarName	Boolean	
GenerateZerosXYMatrix	X	Integer	IMOMatlabMatrix
	Y	Integer	
	CreateNew	Bolean	
GenerateZerosXYZMatrixAsVar	X	Integer	String
	Y	Integer	
	Z	Integer	
	NewVarName	String	
GenerateZerosXYMatrixAsVar	X	Integer	String
	Y	Integer	
	NewVarName	String	
GenerateZerosXYZMatrix	X	Integer	String
	Y	Integer	
	Z	Integer	
	CreateNew	Bolean	
GetDeterminant	-	-	Double
GetDeterminantAsMatrix	CreateNew	Bolean	IMOMatlabMatrix
GetDeterminantAsVar	VarName	String	IMOMatlabVar
GetMatrixSize	CreateNew	Bolean	IMOMatlabMatrix
GetNorm	-	-	Double
GetNormAsMatrix	CreateNew	Bolean	IMOMatlabMatrix
GetNormAsVar	VarName	String	IMOMatlabVar
GetRank	-	-	Double
GetRankAsMatrix	CreateNew	Bolean	IMOMatlabMatrix
GetRankAsVar	VarName	String	IMOMatlabVar
GetRcond	-	-	Double
GetRconAsMatrix	CreateNew	Bolean	IMOMatlabMatrix
GetRcondAsVar	VarName	String	IMOMatlabVar
GetTrace	-	-	Double
GetTraceAsMatrix	CreateNew	Bolean	IMOMatlabMatrix
GetTraceAsVar	VarName	String	IMOMatlabVar

HankelMatrix	CreateNew	Bolean	IMOMatlabMatrix
HankelMatrixAsVar	NewVarName	String	IMOMatlabVar
Hessenberg	CreateNew	Bolean	IMOMatlabMatrix
HorizontalConcat	Matrix	IMOMatlabMatrix	IMOMatlabMatrix
	CreateNew	Boolean	
HorzcatFromMatrix	Matrix	IMOMatlabMatrix	IMOMatlabMatrix
	CreateNew	Boolean	
HorizontalCatVar	VarName	String	String
	CreateNew	Boolean	
	NewVarName	String	
HorizontalCatFromVar	VarName	String	String
	CreateNew	Boolean	
	NewVarName	String	
Inverse	CreateNew	Bolean	IMOMatlabMatrix
IsMatrixEmpty	-	-	Boolean
Kronecker	Matrix	IMOMatlabMatrix	IMOMatlabMatrix
	CreateNew	Boolean	
KronIntoMatrix	MatrixName	String	String
	CreateNew	Boolean	
	NewVarName	String	
KronMatrix	MatrixName	String	IMOMatlabMatrix
	CreateNew	Boolean	
	NewVarName	String	
KronVar	VarName	String	String
	CreateNew	Boolean	
	NewVarName	String	
LuFactor	CreateNew	Bolean	IMOMatlabMatrix
Maximum	CreateNew	Bolean	IMOMatlabMatrix
Median	CreateNew	Bolean	IMOMatlabMatrix
Minimum	CreateNew	Bolean	IMOMatlabMatrix
MultMatrix	Matrix	IMOMatlabMatrix	IMOMatlabMatrix
	CreateNew	Boolean	
MultIntoMatrix	Matrix	IMOMatlabMatrix	IMOMatlabMatrix
	NewMatrixName	String	
MultVar	VarName	String	String
	CreateNew	Boolean	
	NewVarName	String	
Orthogonal	CreateNew	Bolean	IMOMatlabMatrix

PseudoInverse	CreateNew	Bolean	IMOMatlabMatrix
QrFactor	CreateNew	Bolean	IMOMatlabMatrix
Rotate90Degrees	CreateNew	Bolean	IMOMatlabMatrix
SchurDescomposition	CreateNew	Bolean	IMOMatlabMatrix
SingularValueDescomposition	CreateNew	Bolean	IMOMatlabMatrix
SortMatrix	CreateNew	Bolean	IMOMatlabMatrix
SqrtMatrix	CreateNew	Bolean	IMOMatlabMatrix
SubstractFromMatrix	Matrix	IMOMatlabMatrix	IMOMatlabMatrix
	CreateNew	Boolean	
SubstractFromVar	VarName	String	String
	CreateNew	Boolean	
	NewVarName	String	
SubstractMatrix	Matrix	IMOMatlabMatrix	IMOMatlabMatrix
	CreateNew	Boolean	
SubstractVar	VarName	String	String
	CreateNew	Boolean	
	NewVarName	String	
Transpose	CreateNew	Bolean	IMOMatlabMatrix
TransposeNonConjugate	CreateNew	Bolean	IMOMatlabMatrix
TriangularLowPart	CreateNew	Bolean	IMOMatlabMatrix
TriangularUpPart	CreateNew	Bolean	IMOMatlabMatrix
VerticalConcat	Matrix	IMOMatlabMatrix	IMOMatlabMatrix
	CreateNew	Boolean	
VerticalCatFromVar	VarName	String	String
	CreateNew	Boolean	
	NewVarName	String	
VertcatFromMatrix	Matrix	IMOMatlabMatrix	IMOMatlabMatrix
	CreateNew	Boolean	
VerticalCatVar	VarName	String	String
	CreateNew	Boolean	
	NewVarName	String	

En el Anexo 4 se encuentra el resto de métodos que conforman las interfaces de los componentes.

5.2. Pruebas de los componentes de software

Una vez finalizada la implementación de los componentes de software, es necesario realizar las pruebas correspondientes a los componentes. Las pruebas que se realizaron, fueron

pruebas de caja negra. Estas pruebas consistieron en probar cada uno de los métodos y verificar que en MATLAB se crearan las variables correspondientes.

5.2.1. Pruebas de caja negra

Estas pruebas de caja negra se realizaron en Delphi 7.0, teniendo como base el archivo de ayuda de los componentes de software que se distribuye junto con este documento. El proceso de pruebas consiste en probar método a método el componente dentro de un proyecto de software, compilar el proyecto y verificar que la operación se haya realizado y se hayan creado las variables dentro de MATLAB que son devueltas al componente. Se realizaron pruebas a todos los métodos de todas las interfaces del componente pero por cuestiones de espacio solo se incluyen algunas en este documento, de las cuales se presentan una prueba de cada interfaz tanto local como remota.

Prueba 1: Prueba Interfaz IMOMatlabMatrix

El objetivo de esta prueba es crear una matriz mágica de 5x5 a través de la interfaz IMOMatlabMatrix.

Para esta prueba, se agregaron los componentes al proyecto, se crea una sesión y una matriz(en una variable) al momento de crearse la forma, y al pulsar un botón se genera la matriz mágica utilizando el método ***GenerateMagicMatrixAsVar*** que devuelve el nombre de la matriz mágica resultante (ver Tabla 5.2).

procedure TForm1.FormCreate(Sender: TObject);
begin
  Sesion:=COIMOCOMMatlabSession.Create;
  Sesion.OpenSession;
  Matrix:=COIMOMatlabMatrix.Create;
  Matrix.SetSessionOwner(Sesion);
end;
	procedure TForm1.Button1Click(Sender: TObject);
	begin
	  ShowMessage(Matrix.GenerateMagicMatrixAsVar(5,''));
	end;

Tabla 5.2. Operaciones realizadas al crearse la forma y al pulsar el botón en Prueba 1.

Al compilar el proyecto y presionar el botón en cuestión, la respuesta del sistema debe ser el mensaje con el nombre de la matriz mágica, para esta prueba el sistema arrojó el resultado mostrado en la Figura 5.1

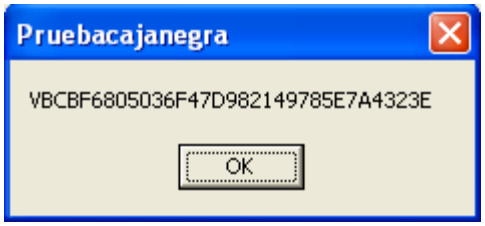


Figura 5.1. Resultado de la creación de la matriz mágica

El sistema imprime el nombre de la matriz resultante, por lo tanto en MATLAB debe haber una matriz con el mismo nombre, y la cual debe contener una matriz mágica. En la Figura 5.2 se muestra la matriz creada en MATLAB y su contenido.

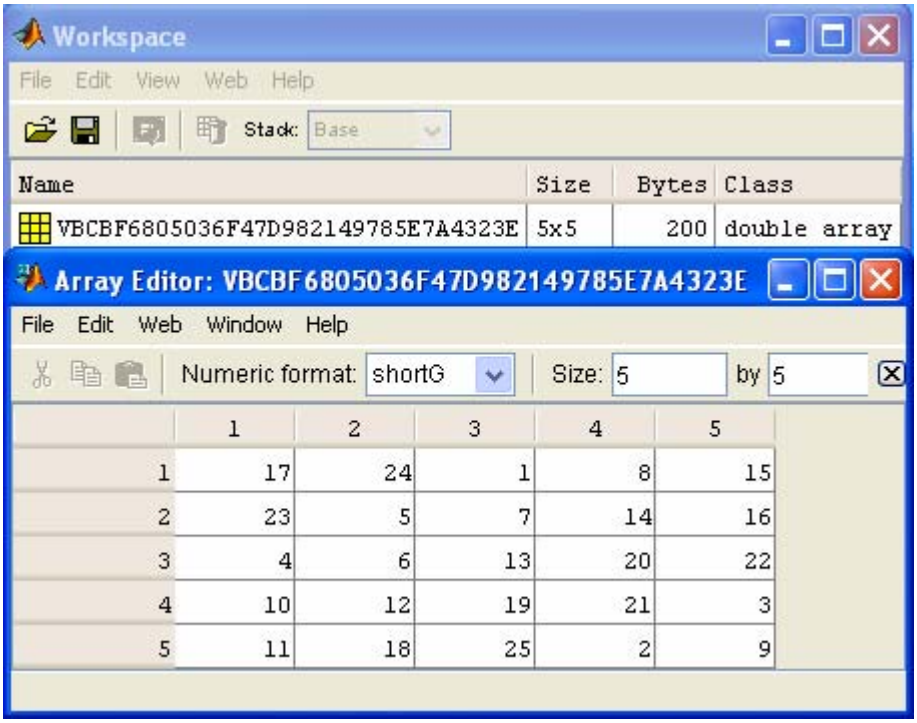


Figura 5.2. Matriz almacenada en MATLAB en Prueba 1.

Si comparamos los resultados de la prueba se puede observar que los resultados son los esperados, con esto comprobamos que el método ***GenerateMagicMatrixAsVar*** de la interfaz IMOMatlabMatrix devuelve los resultados correctos.

Prueba 2: Prueba Interfaz IMOMatlabMatrixOperationLib

El objetivo de esta prueba es crear una matriz Wilkinson de 5x5 para después realizar el determinante de dicha matriz mediante la interfaz IMOMatlabMatrixOperationLib.

Para esta prueba, se agregaron los componentes al proyecto, se crea una sesión y una matriz al momento de crearse la forma, y al pulsar un botón se genera la matriz Wilkinson utilizando el método ***Wilkinson,*** después se obtiene el determinante de la misma matriz a través del método ***Det*** (ver Tabla 5.3).

Tabla 5.3. Operaciones realizadas al crearse la forma y al pulsar el botón en Prueba 2.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Sesion:=COIMOCOMMAtlabSession.Create;
  Sesion.OpenSession;
  Matrix:=COIMOMatlabMatrixOperationLib.Create;
  Matrix.SetSessionOwner(Sesion);
end;
```

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    Matrix.Wilkinson(5,'Wilkin');
  Matrix.Det('Wilkin','Det');
end;
```

Al compilar el proyecto y presionar el botón, se deben crear en MATLAB dos variables, la primera en donde se tenga la matriz Wilkinson de 5x5 llamada “Wilkin” y la segunda

una variable donde se deposite el resultado de sacar la determinante de la matriz Wilkinson, esta variable debe tener por nombre "Det".

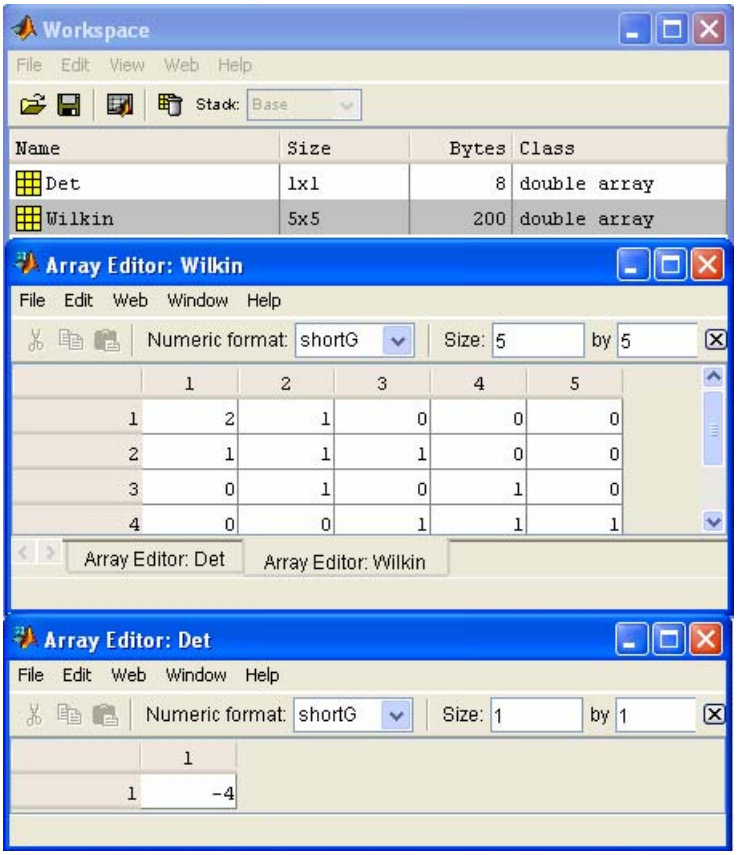


Figura 5.3. Matrices almacenadas en MATLAB en Prueba 2.

En la Figura 5.3 se pueden observar los resultados obtenidos en MATLAB.

Si observamos los resultados de la prueba se puede notar que son los esperados, con esto comprobamos que los métodos ***Wilkinson*** y ***Det*** de la interfaz IMOMatlabMatrixOperationLib devuelven los resultados correctos.

Prueba 3: Prueba Interfaz ISOAPMatlabMatrix

El objetivo de esta prueba es crear una matriz Hilbert de 4x4 a través de la interfaz ISOAPMatlabMatrix.

Para esta prueba, se agregaron los componentes al proyecto, se crea una sesión y una matriz al momento de crearse la forma, y al pulsar un botón se genera la matriz Hilbert utilizando el método ***GenerateHilbertMatrix*** que devuelve el identificador de la matriz resultante (ver Tabla 5.4).

```
procedure TForm1.FormCreate(Sender: TObject);
begin


  Se-
sion:=GetISOAPMatlabSession(false,'http://localhost/IMO_SOAP_for_MATLAB_CGI.exe/soap/ISOAPMatlabSession'
);
  sesionid:=Sesion.OpenSession;
  Matrix:=GetISOAPMatlabMatrix(false,'http://localhost/IMOSOAPMatlabMatrix.exe/soap/ISOAPMatlabMatrix');
  matrixID:=Matrix.CreateMatrix;
  Matrix.SetSessionOwner(matrixID,sesionid);
end;
```

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  matrix.GenerateHilbertMatrix(matrixID,4,Mat_false);
  ShowMessage(matrix.GetVarName(matrixID));
End;
```

Tabla 5.4. Operaciones realizadas al crearse la forma y al pulsar el botón en Prueba 3.

Al compilar el proyecto y presionar el botón en cuestión, el sistema debe crear la matriz Hilbert y después imprimir la variable que contiene la matriz resultante (ver Figura 5.4).

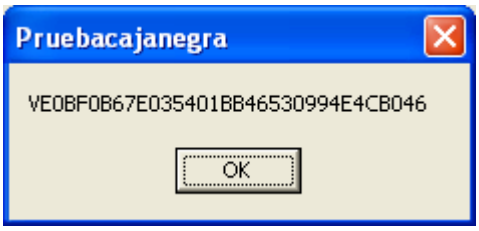


Figura 5.4. Resultado de la creación de la matriz Hilbert.

A su vez, en MATLAB debe existir una matriz con el mismo nombre la cual contenga una matriz Hilbert de 4x4. Esto se muestra en la Figura 5.5

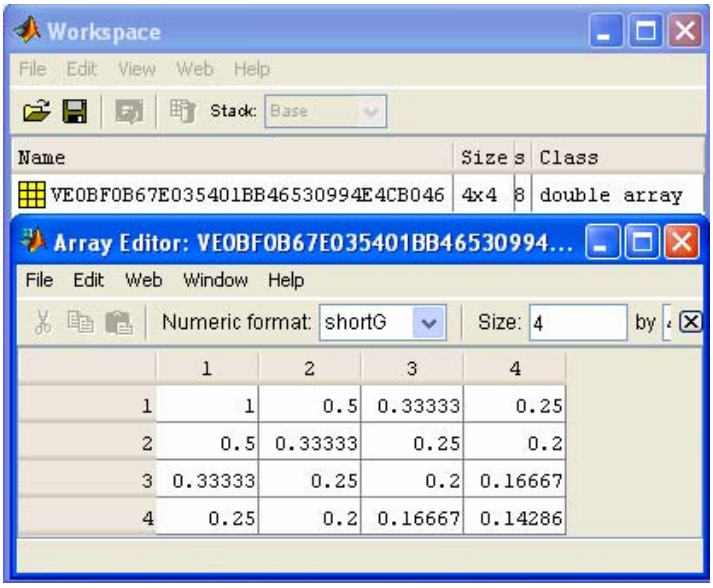


Figura 5.5. Matriz almacenada en MATLAB en Prueba 3.

Si comparamos los resultados de la prueba se puede observar que los resultados son los esperados, con esto comprobamos que el método ***GenerateHilbertMatrix*** de la interfaz ISOAPMatlabMatrix devuelve los resultados correctos.

Prueba 4: Prueba Interfaz ISOAPMatlabMatrixOperationLib

El objetivo de esta prueba es crear una matriz Hadamard de 4x4 para después obtener la norma de dicha matriz mediante la interfaz ISOAPMatlabMatrixOperationLib.

Para esta prueba, se agregaron los componentes al proyecto, se crea una sesión y una matriz al momento de crearse la forma, y al pulsar un botón se genera la matriz Hadamard utilizando el método ***Hadamard,*** posteriormente se obtiene la norma de la misma matriz a través del método ***Norm*** (ver Tabla 5.5).

Tabla 5.5. Operaciones realizadas al crearse la forma y al pulsar el botón en Prueba 4.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Sesion:=GetISOAPMatlabSession(false,'http://localhost/IMO_SOAP_for_MATLAB_CGI.exe/soap/ISOAPMatlabSession');
  sesionid:=Sesion.OpenSession;
  Matrix1:=GetISOAPMatlabMatrixOperationLib(false,'http://localhost/IMOSOAPMatLabMatrix.exe/soap/ISOAPMatlabMatrixOperationLib');
  matrixID1:=Matrix1.CreateMatrix;
  Matrix1.SetSessionOwner(matrixID1,sesionid);
End;
```

```
procedure TForm1.Button2Click(Sender: TObject);
begin
 Matrix1.Hadamard(matrixID1,4,'Hada');
```

```
 Matrix1.Norm(matrixID1,'Hada','Norma');
end;
```

Al compilar el proyecto y presionar el botón, se deben crear en MATLAB 2 variables, la primera en donde se tenga la matriz Hadamard de 4x4 llamada “Hada” y la segunda donde se deposite el resultado de sacar la norma de la matriz Hadamard, esta variable debe tener por nombre “Norma”.

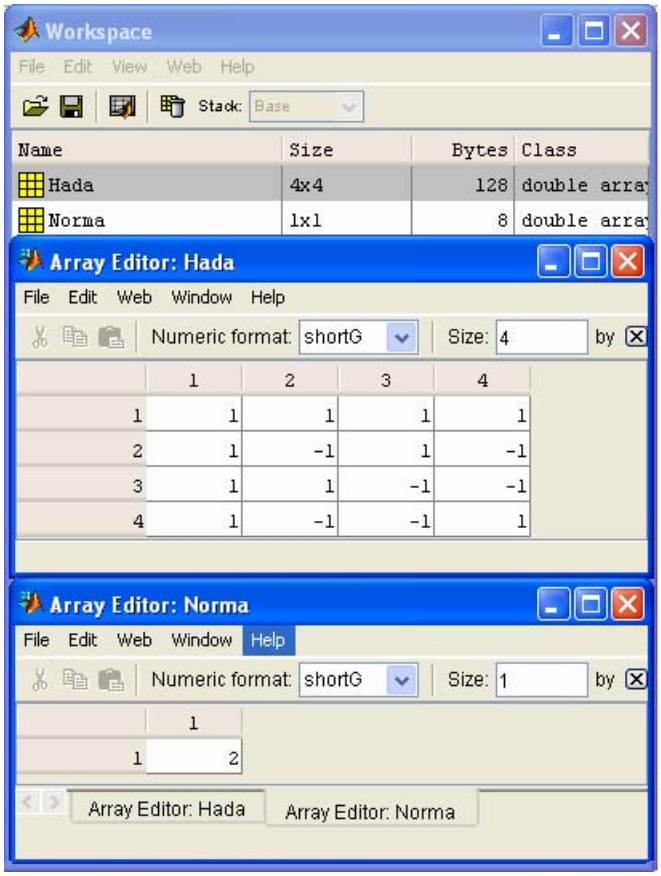


Figura 5.6. Matrices almacenadas en MATLAB en Prueba 4.

En la Figura 5.6 se pueden observar los resultados obtenidos en MATLAB. Si comparamos los resultados de la prueba se puede observar que los resultados son los esperados, con esto comprobamos que los métodos ***Hadamard*** y ***Norm*** de la interfaz ISOAPMatlabMatrixOperationLib devuelven los resultados correctos.

Como resultado de estas pruebas aplicadas a todos los métodos que conforman las interfaces de los componentes, se menciona que no se encontraron errores en los mismos, por lo que se prosigue a utilizarlos en la implementación del Manipulador de Matrices.

6. Implementación y pruebas de la aplicación de software

Este capítulo trata acerca de la implementación de la aplicación de software sobre la cual se ha venido trabajando en el desarrollo de esta tesis. La implementación es la consecuencia de la etapa anterior del análisis y el diseño del sistema.

6.1. Implementación

Como se trató en los capítulos anteriores, para la implementación de esta aplicación se utilizó Object Pascal como lenguaje de programación, para lo cual se usó el ambiente de desarrollo Delphi en su versión 7.

6.1.1. Distribución física de la aplicación

Esta parte de la implementación se refiere a la distribución física de las formas dentro de la aplicación. En la Figura 6.1 se aprecia el árbol de distribución de las principales formas en el sistema.

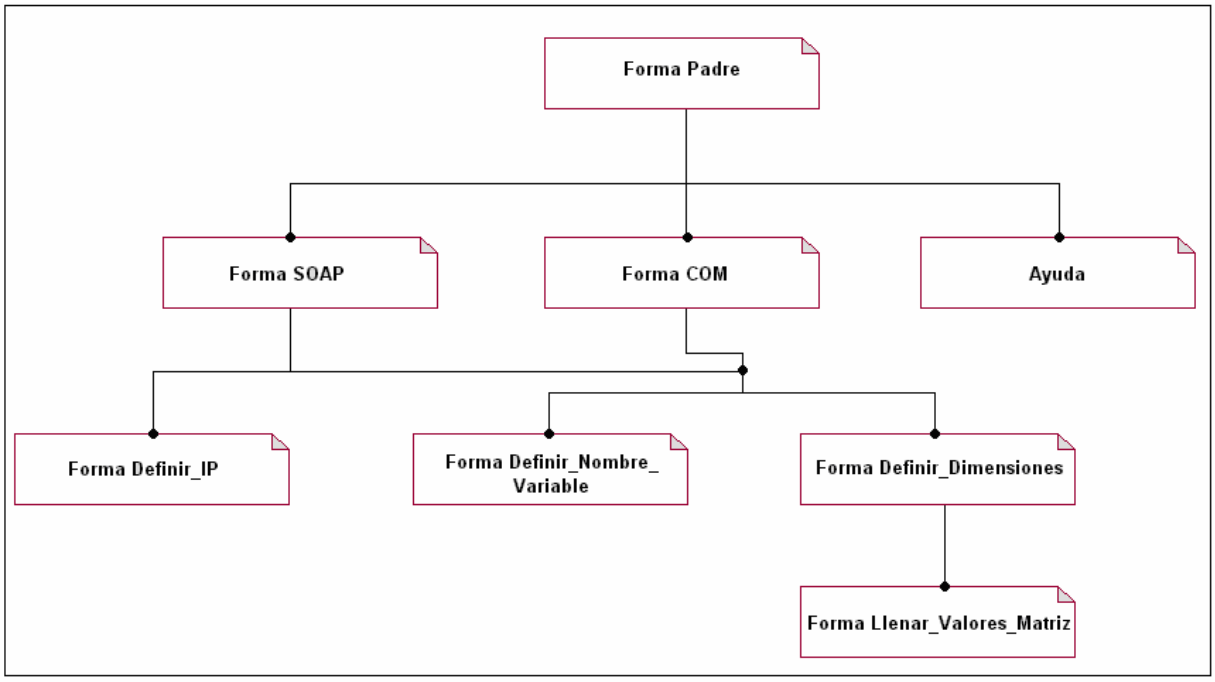


Figura 6.1. Árbol de distribución de las principales formas de la aplicación.

La descripción de las formas que se muestran en la Figura 6.1 se detallan en la Tabla 6.1.

Tabla 6.1. Descripción de las principales formas del sistema.

Nombre de la forma	Descripción
Padre	Esta forma es la primera que se presenta al usuario después de haber sido cargado el sistema, en esta forma trabajarán el resto de formas del sistema. En esta forma existirán opciones para abrir y cerrar sesiones locales y remotas respectivamente; existirá también la opción de ayuda del sistema y mostrará el número de sesiones que el usuario tiene abiertas actualmente.
SOAP	Esta forma aparecerá después de que el usuario haya escogido la opción de abrir una sesión remota y haya introducido la dirección IP o bien el dominio del servidor. En esta forma el usuario podrá crear todos los tipos de matrices antes mencionados y realizar diversas operaciones sobre las matrices creadas. Además, esta forma presentará opciones para la configuración visual de la misma.
COM	Esta forma aparece después de que el usuario haya escogido la opción de abrir una sesión local. La funcionalidad de la misma es similar a la forma SOAP.
Ayuda	Esta forma va a mostrar la ayuda para todas las funciones del sistema.
Definir_IP	Cuando el usuario escoja la opción de abrir una sesión remota, se abrirá una forma en donde el usuario deberá introducir la dirección IP o el nombre del servidor.
Definir_Nombre_Variable	Esta forma aparece cuando el usuario ejecuta una operación sobre una matriz que devuelva una variable, de tal manera que en esta forma se defina el nombre de la variable de salida.
Definir_Dimensiones	Esta forma aparece cada que el usuario elige crear una matriz, en esta se introducen las dimensiones de la matriz y el nombre de la misma.
Llenar_Valores_Matriz	Cuando el usuario decide crear una matriz definida por él mismo, el sistema muestra esta forma en la cual se proporcionarán los valores de cada celda de la matriz.

Aunada a las formas anteriores, en el sistema existen algunas otras formas auxiliares, las cuales son poco relevantes para el sistema por lo que no se han descrito.

6.1.2. Diseño e implementación de interfaces

Una vez que se mostró la distribución de las formas a través del sistema, es necesario diseñar estas formas de tal manera que ayuden a los usuarios a tener una idea clara del funcionamiento del sistema.

En este apartado se mostrarán cada una de las interfaces que muestra el sistema. A través de los procesos del mismo. La primera pantalla que se muestra es cuando el usuario ejecuta el sistema, durante el proceso de carga del sistema en memoria, se mostrará al usuario una pantalla de entrada al sistema, dicha pantalla se muestra en la Figura 6.2.

Figura 6.2. Pantalla inicial del sistema (carga del sistema).

La pantalla anterior se cierra de manera automática al cargarse el sistema, inmediatamente se mostrará la interfaz principal del sistema.

Pantalla principal del sistema

Después de haberse cargado el sistema, se mostrará la interfaz principal del mismo (Figura 6.3), en esta interfaz podremos acceder mediante el menú Archivo a la apertura de sesiones tanto locales como remotas, así como a la opción para salir del sistema, mientras que en el menú Ayuda se puede acceder a la ayuda del sistema.

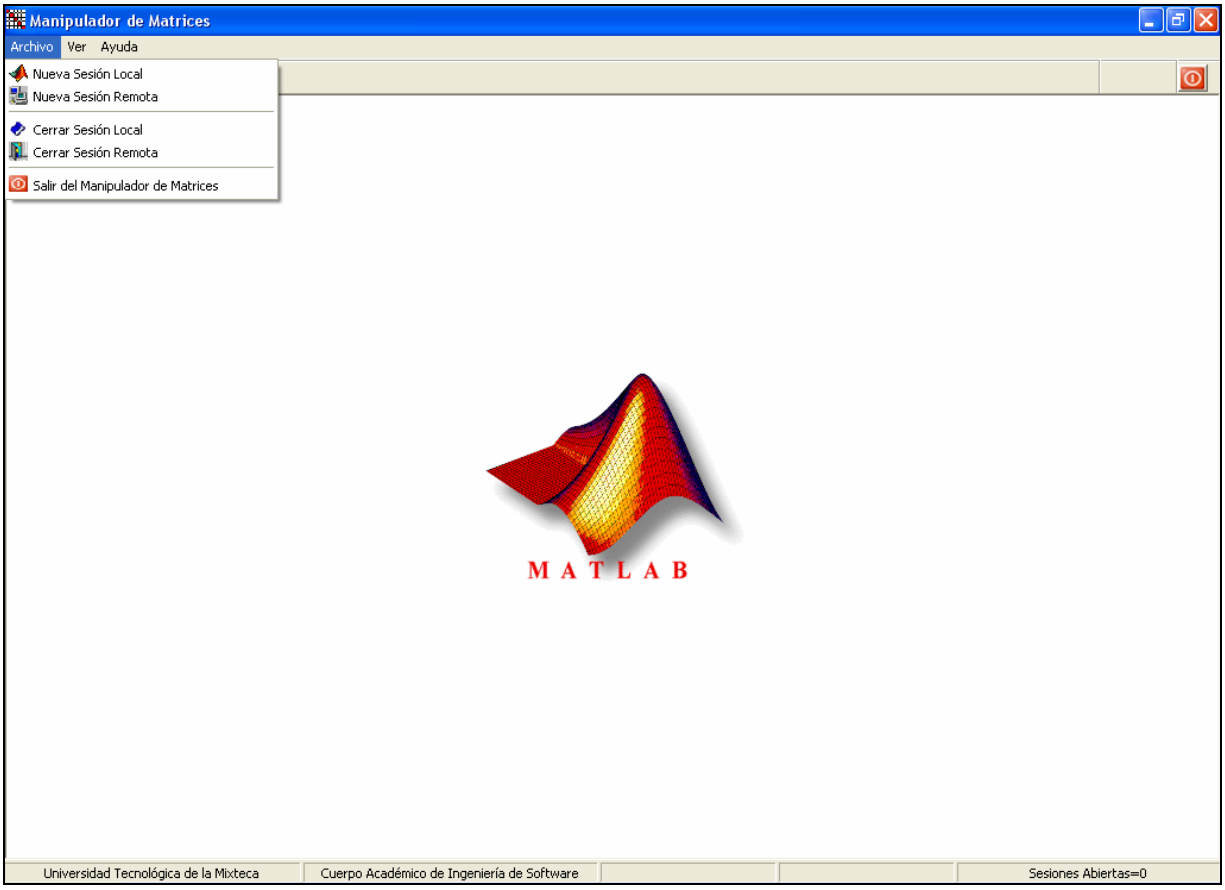


Figura 6.3. Interfaz principal del sistema

En esta misma interfaz encontramos una barra de botones mediante los cuales se pueden acceder a las mismas operaciones que se encuentran en el menú Archivo.

En la parte central se encuentra el panel de trabajo sobre el que se visualizarán las demás interfaces del sistema. Por último, en la parte inferior tenemos una barra de estado en la que se mostrará el número de sesiones abiertas en el sistema, así como el número de matrices y variables creadas dentro de cada sesión abierta.

Crear una sesión local o remota

Para crear una sesión local o remota se debe escoger la opción de creación de sesión en el menú Archivo o en la barra de botones de la interfaz principal del sistema (ver Figura 6.3).

En caso de escoger la opción para crear una sesión remota, se presentará la interfaz que se muestra en la Figura 6.4, en esta interfaz se solicita al usuario que introduzca la dirección IP o el nombre del servidor.

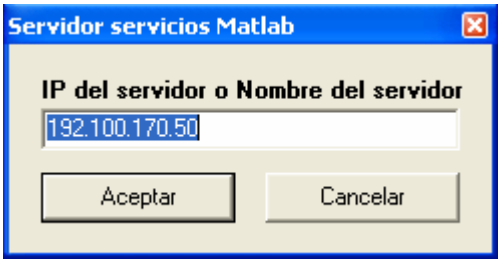


Figura 6.4. Interfaz para introducir la dirección del servidor.

Esta interfaz tendrá como parámetro predefinido la dirección 192.100.170.50 que es la dirección IP en la que se encuentra alojado el software MATLAB y el componente SOAP, por lo tanto funciona como servidor. Después de dar clic en el botón Aceptar el sistema se conectará al servidor y mostrará la interfaz de la Figura 6.5. En caso de que el usuario haya decidido crear una sesión local, el sistema no mostrará la Figura 6.4 y de inmediato pasará a mostrar una interfaz muy similar a la que se observa en la Figura 6.5.

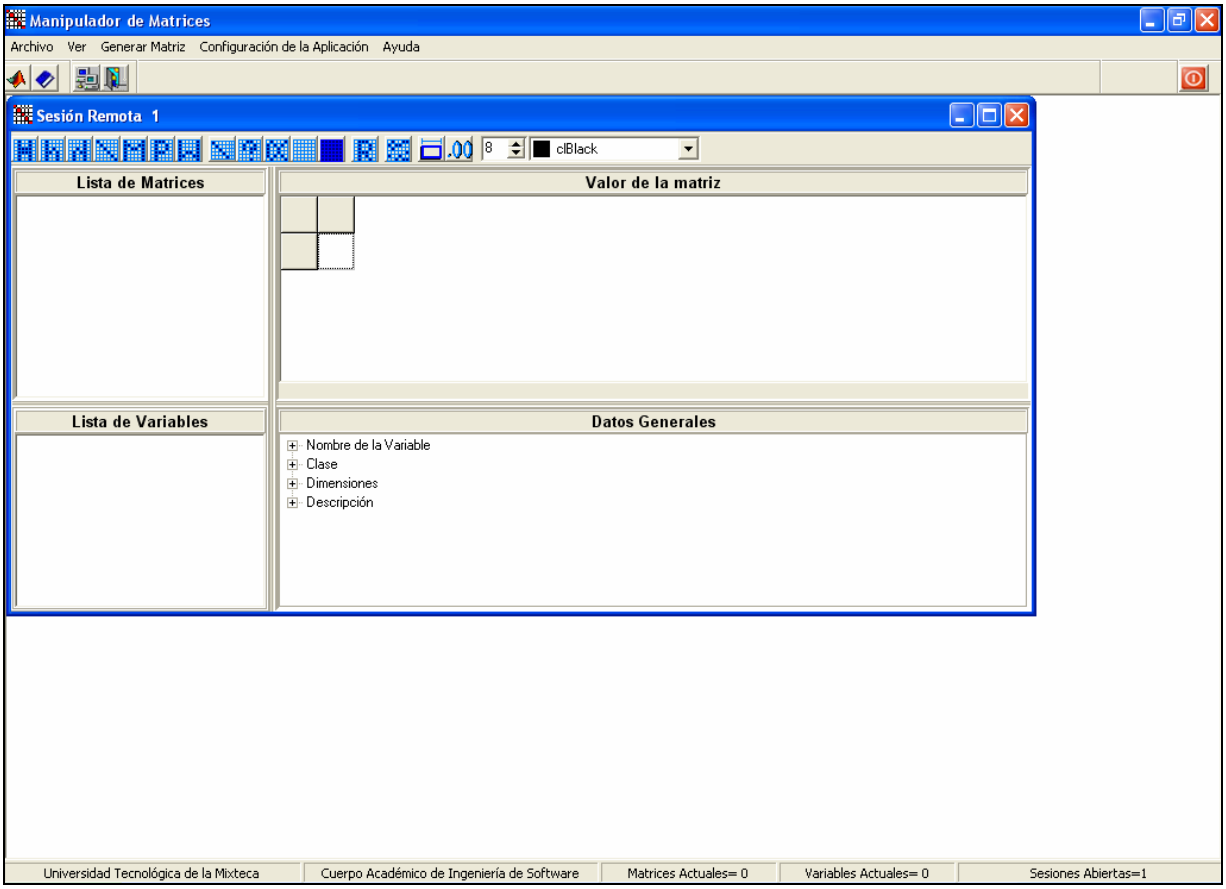


Figura 6.5. Interfaz al abrir una sesión remota.

En esta interfaz existe un menú para la creación de matrices y para la configuración de la aplicación, las opciones del menú también aparecen en la barra de botones que se aprecia en la parte superior de la interfaz. Básicamente, la interfaz se divide en cuatro paneles: en la esquina superior izquierda va a aparecer la lista de los nombres de las matrices creadas en la sesión, en esta sección mediante un menú contextual el usuario podrá acceder a todas la operaciones que se puedan hacer sobre las matrices creadas en el sistema; en la parte inferior izquierda aparece la lista de los nombres de las variables creadas en la sesión; en la esquina superior derecha aparece la sección donde se muestra el contenido de las matrices y variables que el usuario haya seleccionado en los paneles correspondientes; mientras que en la esquina inferior derecha aparece una sección donde se muestran los datos generales de las variables y matrices seleccionadas.

Cabe señalar que se pueden crear múltiples sesiones tanto locales como remotas, y el sistema abrirá una nueva interfaz por cada sesión abierta (ver Figura 6.6).

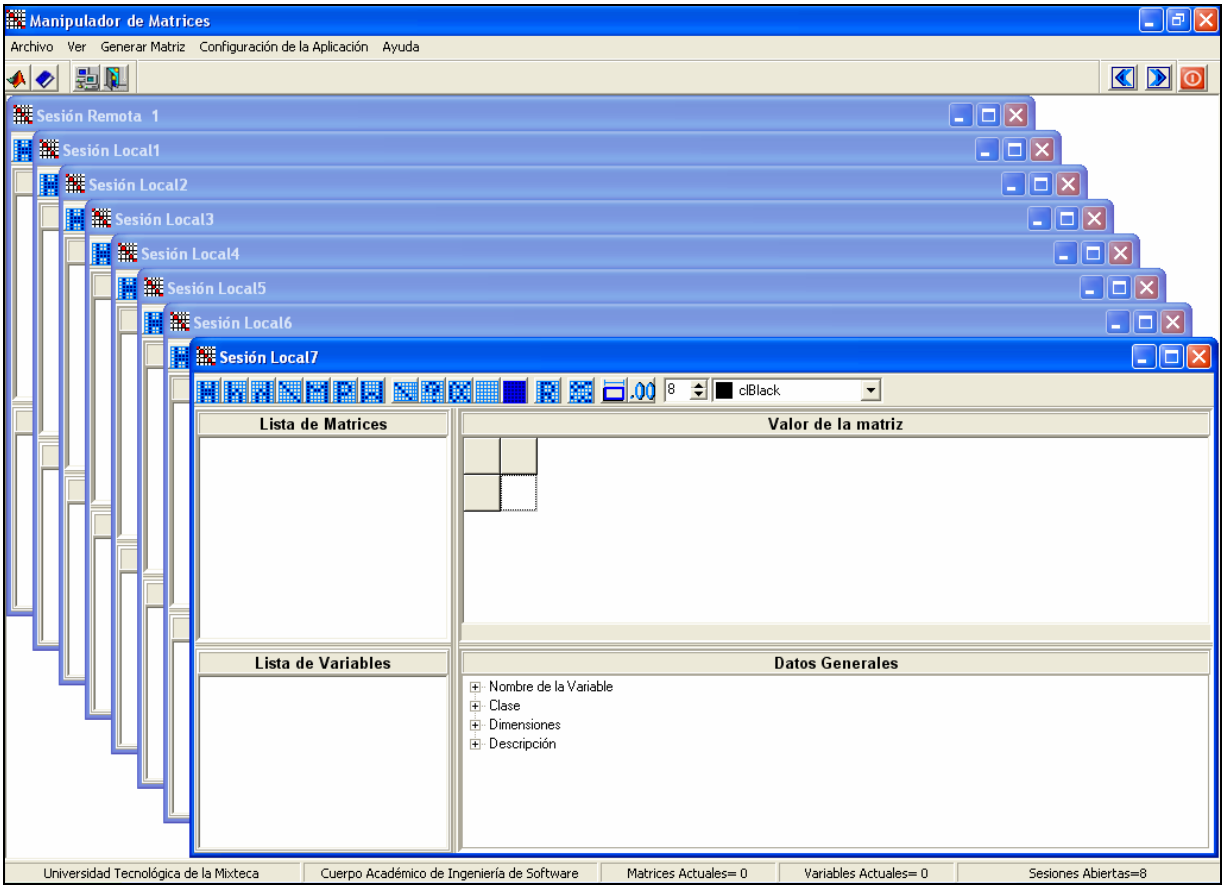


Figura 6.6. Múltiples sesiones abiertas en el sistema.

De la misma manera en que el usuario puede abrir múltiples sesiones las puede cerrar sin importar el orden en las que fueron abiertas.

Creación de matrices

Para la creación de matrices el usuario lo podrá realizar utilizando el menú Generar Matriz o directamente con los botones que se encuentran en la barra de botones. Por ejemplo, si el usuario desea crear una matriz definida por el mismo, deberá escoger dicha opción y a continuación le aparecerá la interfaz en la cual deberá introducir las dimensiones de la matriz, esta interfaz podrá variar dependiendo si la matriz a crear es una matriz cuadrada (ver Figura 6.7) o una matriz no cuadrada (ver Figura 6.8). En este caso, la matriz puede ser no cuadrada y se ocupará la interfaz para dicho tipo de matriz.

Figura 6.7. Interfaz para introducir dimensiones de matriz cuadrada.

Figura 6.8. Interfaz para introducir dimensiones de matriz no cuadrada.

Continuando con el ejemplo, el usuario introduce valores para obtener una matriz de 5x4 y le asigna el nombre "Prueba" a la variable. Después de dar clic en el botón Aceptar, el sistema mostrará la siguiente interfaz, en la cual el usuario deberá introducir los valores de la matriz que desea crear (ver Figura 6.9).

Figura 6.9. Interfaz para introducir valores a la matriz.

Una vez introducidos los valores de la matriz, basta con presionar el botón Aceptar para que la matriz quede registrada en el sistema.

Interfaces auxiliares

Mediante estas interfaces se pretende alertar al usuario sobre alguna operación no válida que quiera realizar sobre el sistema o simplemente alguna confirmación a algún evento importante dentro del mismo.

Por ejemplo, cuando se quiera realizar una operación no válida sobre cierto tipo de matrices, el sistema alertará al usuario mediante un mensaje, tal es el caso al tratar de realizar una operación de determinante sobre una matriz no cuadrada, el mensaje se puede apreciar en la Figura 6.10.

Figura 6.10. Interfaz para mostrar una operación invalida sobre una matriz.

Por otro lado, cuando el usuario realice un evento trascendental, el sistema pedirá una confirmación de dicho evento, tal es el caso del cierre del sistema (ver Figura 6.11).

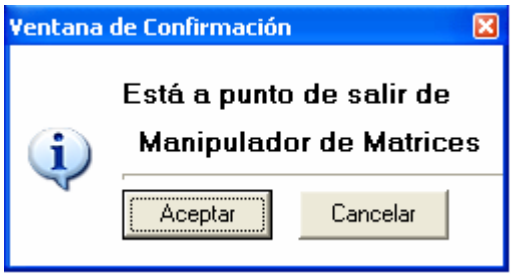


Figura 6.11. Interfaz para confirmar la salida del sistema.

Con la aplicación de estas interfaces auxiliares, el sistema le indica al usuario cualquier estado anormal de alguna operación, haciendo más comprensible el uso del propio sistema.

Una vez que se han mostrado las interfaces del sistema, es necesario probar el mismo para asegurar que se libera un producto confiable.

6.2. Pruebas de la aplicación de software

Después de haber terminado la implementación del sistema, corresponde realizar las respectivas pruebas para asegurar que todas las funcionalidades del sistema operan de manera correcta. Al igual que en las pruebas de los componentes de software, las pruebas que aquí se aplican son pruebas de caja negra las cuales consistieron en probar todas las operaciones del sistema y corroborar que los resultados obtenidos fueran iguales a los resultados que MATLAB arroja al realizar directamente las operaciones en la consola.

6.2.1.1. Pruebas de caja negra

Estas pruebas de caja negra están basadas en el manual de usuario que se distribuye junto con este documento. Se realizaron pruebas a todas las operaciones que la aplicación ofrece, pero por cuestiones de espacio, solo se mostrarán algunas.

Prueba 1*: *Generar matriz Pascal

Para generar una matriz Pascal el usuario deberá iniciar el sistema y tener abierta al menos una sesión local o remota.

En la sesión que tenga abierta deberá escoger la opción “Generar Matriz Pascal” (ver Figura 6.12).

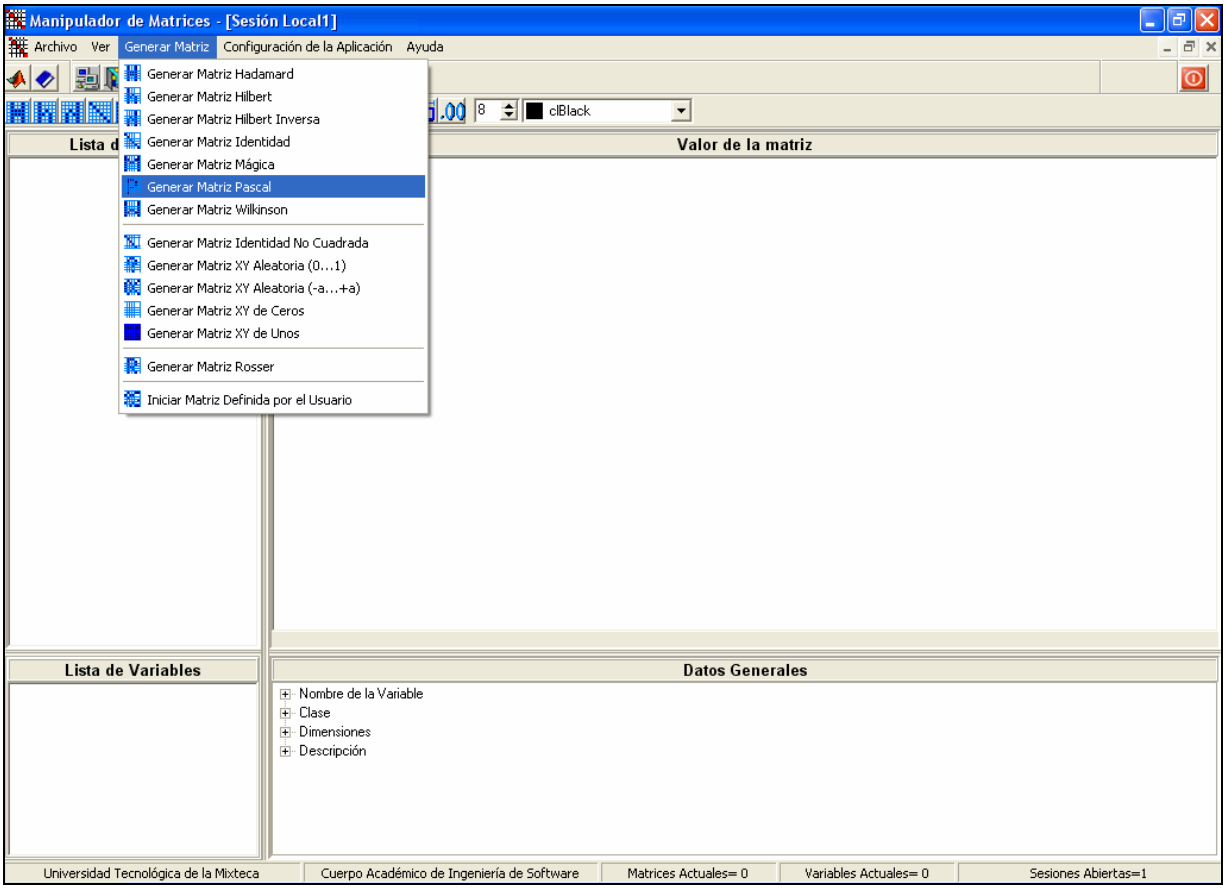


Figura 6.12. El usuario escoge la opción "Generar Matriz Pascal".

Después de esto, el sistema debe mostrar una pantalla para introducir la dimensión y el nombre de la variable de la matriz. El usuario introduce como dimensión 10 y como nombre de matriz Prueba (ver Figura 6.13).

Figura 6.13. Pantalla para introducir la dimensión y el nombre de la matriz

El sistema debe crear una matriz Pascal y poner el nombre de la matriz en el panel superior derecho en donde podrá ser consultado por el usuario.

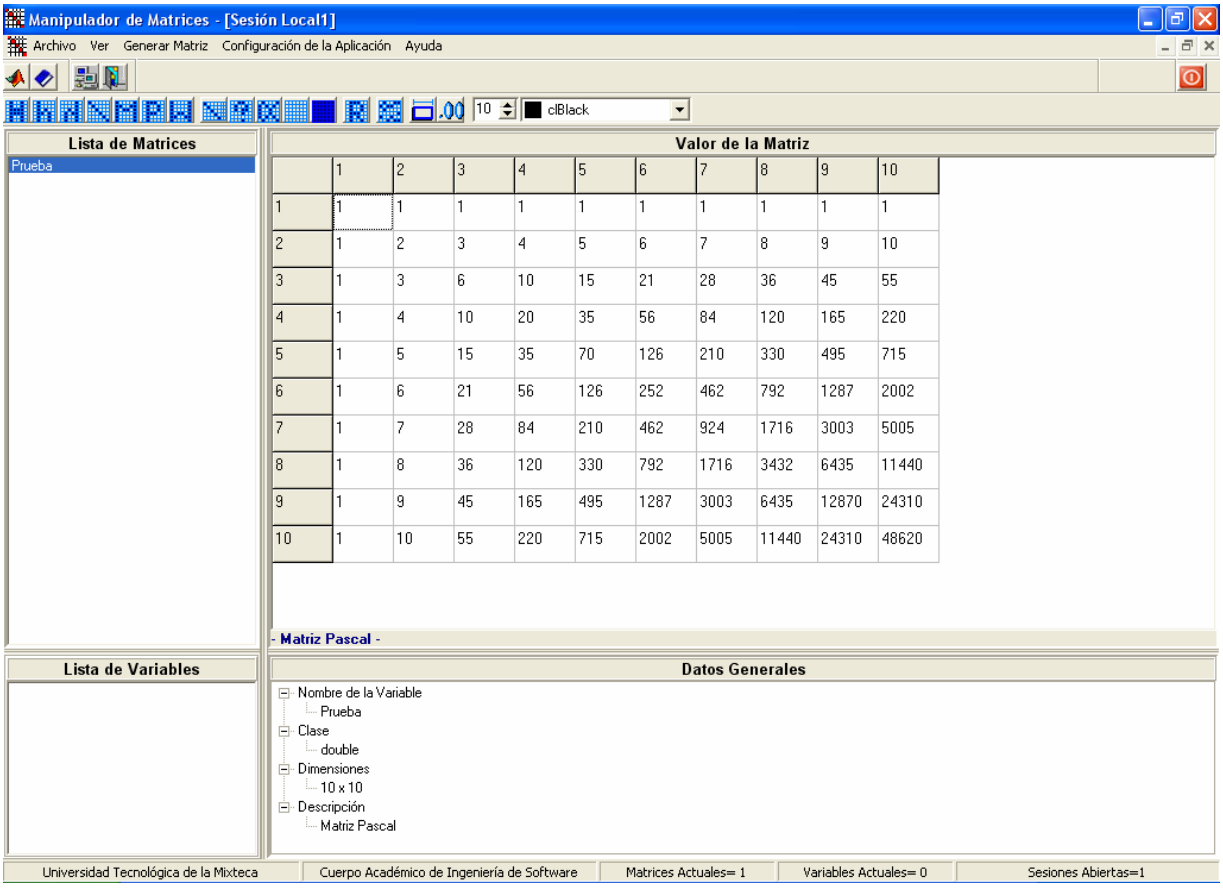
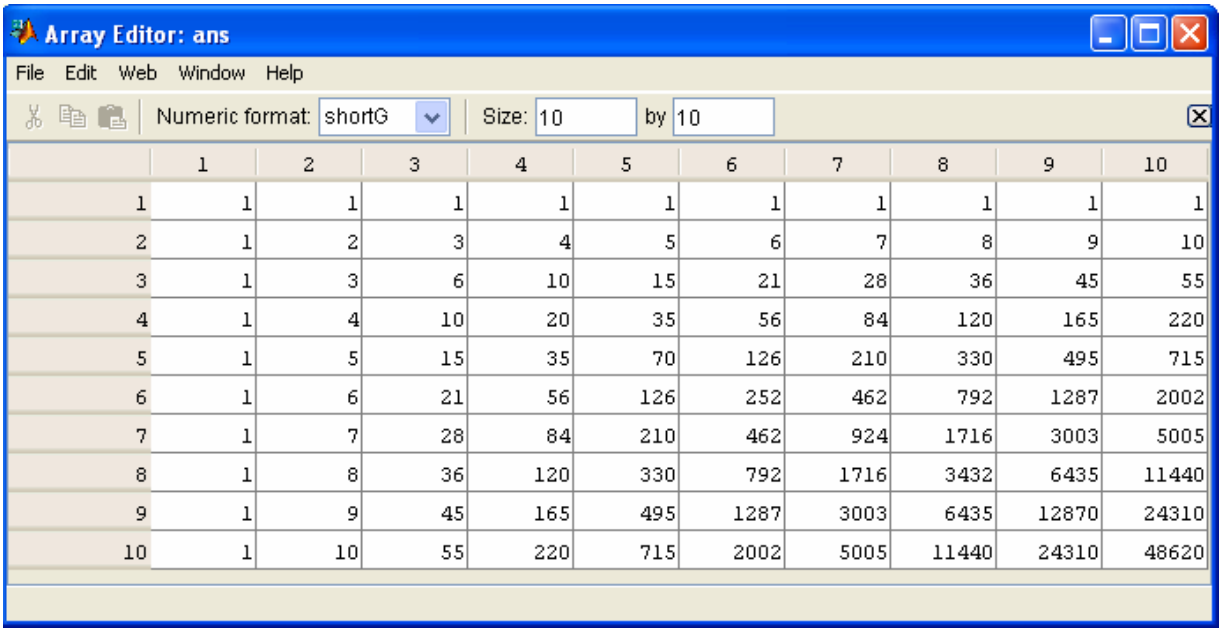


Figura 6.14. Resultado de la operación "Generar Matriz Pascal".

Para comprobar que el resultado es el correcto, abrimos una sesión en MATLAB y ejecutamos el comando para crear una matriz Pascal de las mismas dimensiones.



	1	2	3	4	5	6	7	8	9	10
1	1	1	1	1	1	1	1	1	1	1
2	1	2	3	4	5	6	7	8	9	10
3	1	3	6	10	15	21	28	36	45	55
4	1	4	10	20	35	56	84	120	165	220
5	1	5	15	35	70	126	210	330	495	715
6	1	6	21	56	126	252	462	792	1287	2002
7	1	7	28	84	210	462	924	1716	3003	5005
8	1	8	36	120	330	792	1716	3432	6435	11440
9	1	9	45	165	495	1287	3003	6435	12870	24310
10	1	10	55	220	715	2002	5005	11440	24310	48620

Figura 6.15. Matriz Pascal generada en MATLAB.

Si comparamos los resultados del manipulador de matrices (Figura 6.14) y de la matriz generada en MATLAB (Figura 6.15), se puede observar que corresponden a los mismos valores casilla a casilla, con esto comprobamos que la operación "Generar Matriz Pascal" del Manipulador de Matrices devuelve los resultados correctos.

En las siguientes pruebas, por cuestiones de espacio solo se mostrará la imagen que muestre el resultado de la operación tanto del manipulador de matrices como de MATLAB.

Prueba 2: Generar Matriz Identidad No Cuadrada

Para generar una matriz identidad no cuadrada el usuario deberá haber iniciado el sistema y tener al menos una sesión abierta. En esta sesión deberá escoger la opción "Generar Matriz Identidad No Cuadrada".

El sistema debe mostrar una pantalla donde debe ingresar las dimensiones de la matriz así como el nombre de la misma, en esta prueba se escoge como dimensiones para las filas 5 y para columnas 7 y como nombre de la matriz "Identidad", el resultado de la operación se puede observar en la Figura 6.16

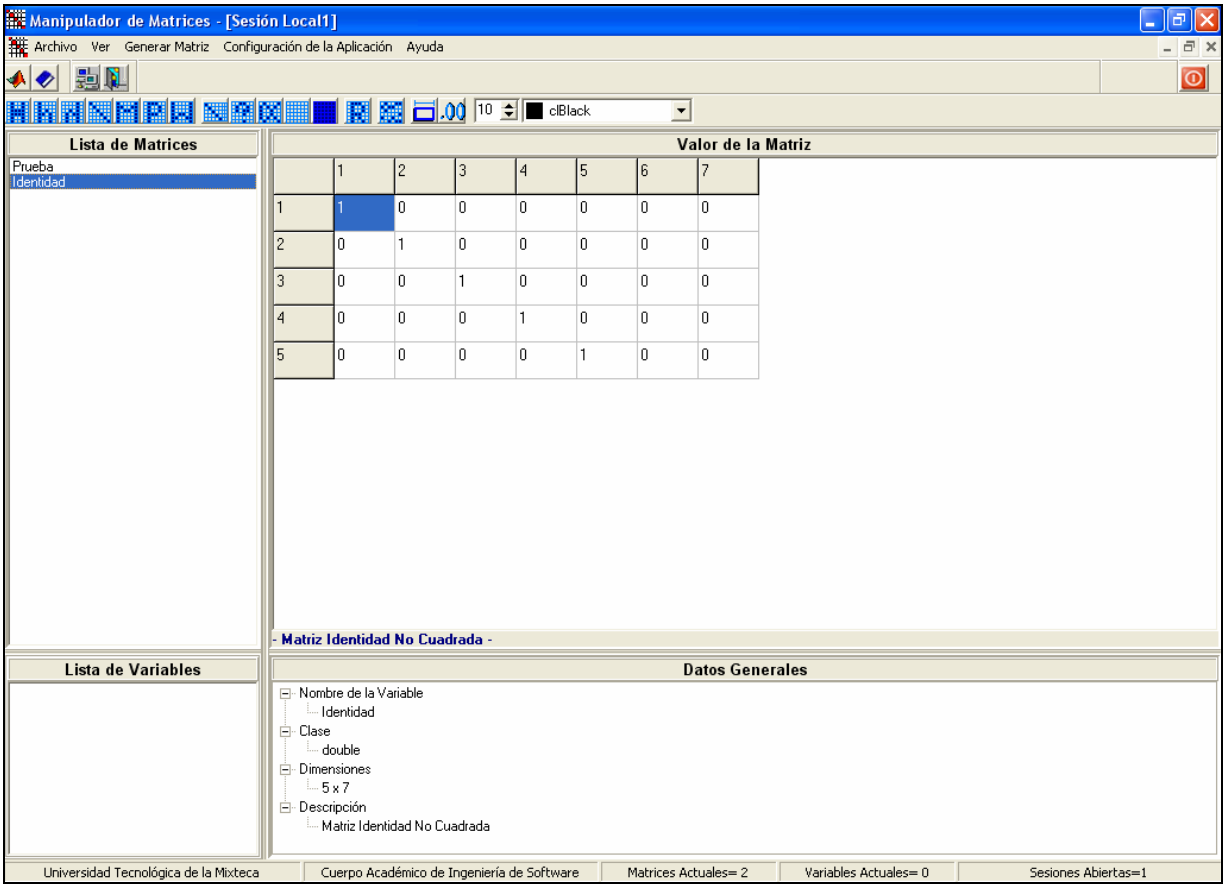
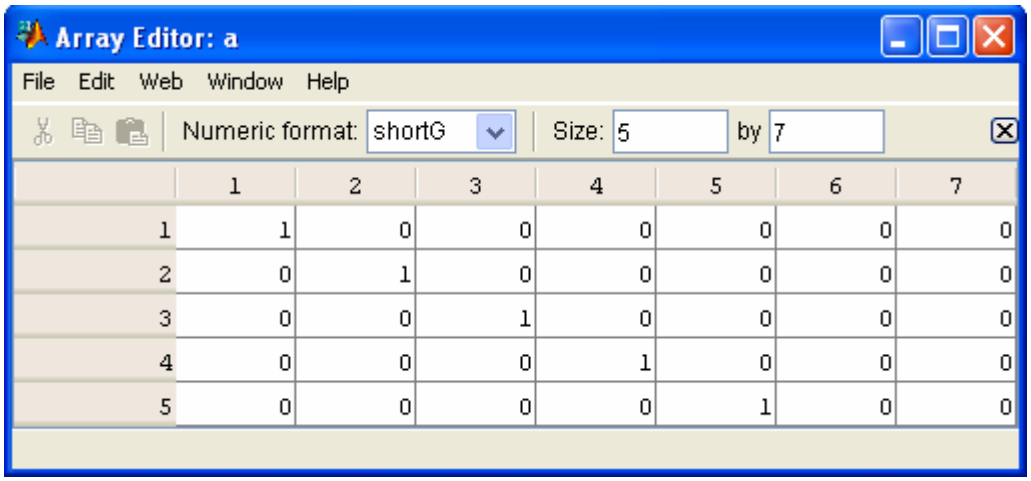


Figura 6.16. Matriz Identidad no Cuadrada

Ahora, ejecutando la misma operación en MATLAB, nos ofrece el resultado que se muestra en la Figura 6.17



	1	2	3	4	5	6	7
1	1	0	0	0	0	0	0
2	0	1	0	0	0	0	0
3	0	0	1	0	0	0	0
4	0	0	0	1	0	0	0
5	0	0	0	0	1	0	0

Figura 6.17. Matriz identidad no cuadrada de 5x7 generada en MATLAB.

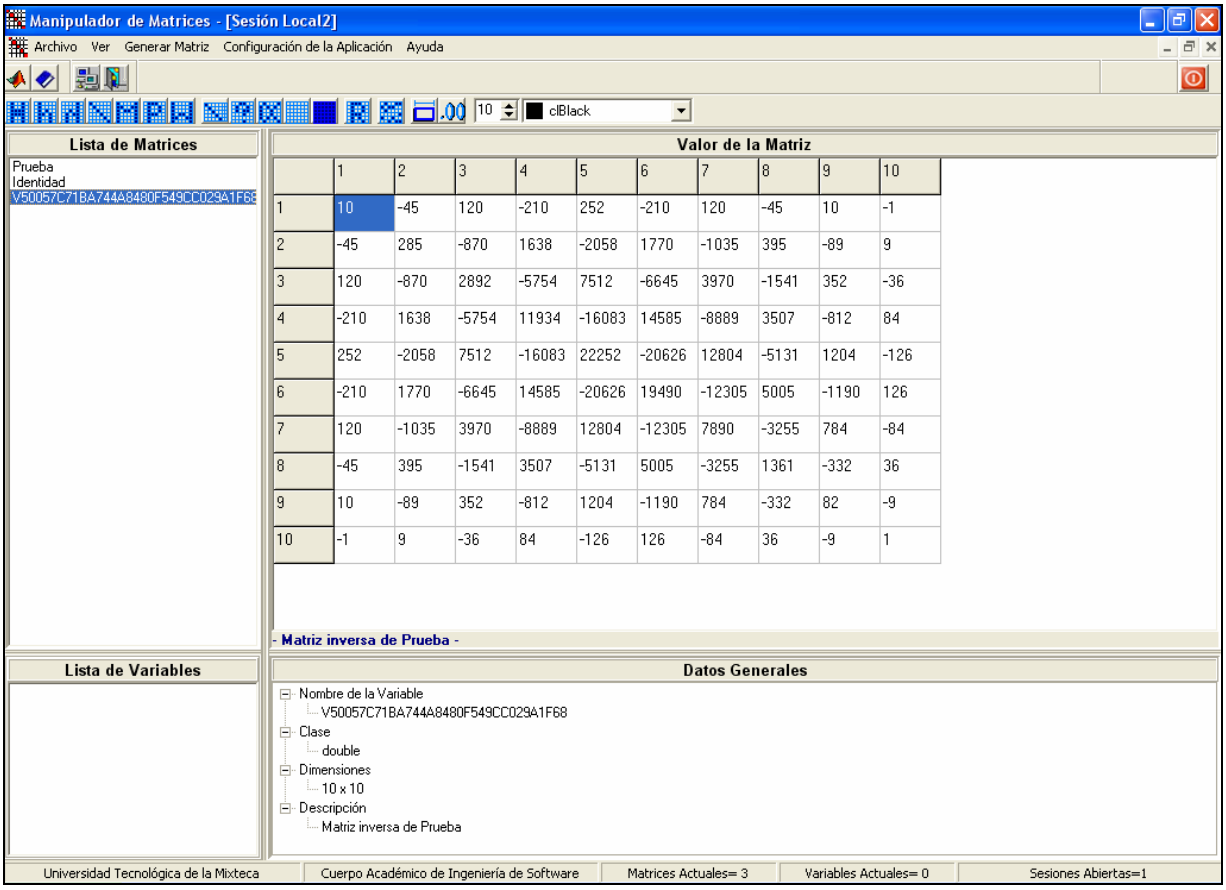
Comparando los resultados del manipulador de matrices (Figura 6.16) y de la matriz generada en MATLAB (Figura 6.17), se puede observar que corresponden a los mismos valores casilla a casilla, con esto comprobamos que la operación "Generar Matriz Identidad no Cuadrada" del Manipulador de Matrices devuelve los resultados correctos

Prueba 3: Obtener Matriz Inversa

Para obtener la matriz inversa de otra matriz, el usuario deberá haber iniciado el sistema, tener al menos una sesión abierta, y tener creada al menos una matriz cuadrada.

Para esta prueba escogemos crear la matriz a partir de la matriz Pascal que se generó en la Prueba 1. Para obtener la matriz Inversa de la matriz Pascal debemos seleccionar la matriz que tiene por nombre "Prueba", dar clic derecho sobre la misma y escoger en el menú emergente el submenú "Operaciones que no ocupan parámetros de entrada" y escoger la opción "Inversa".

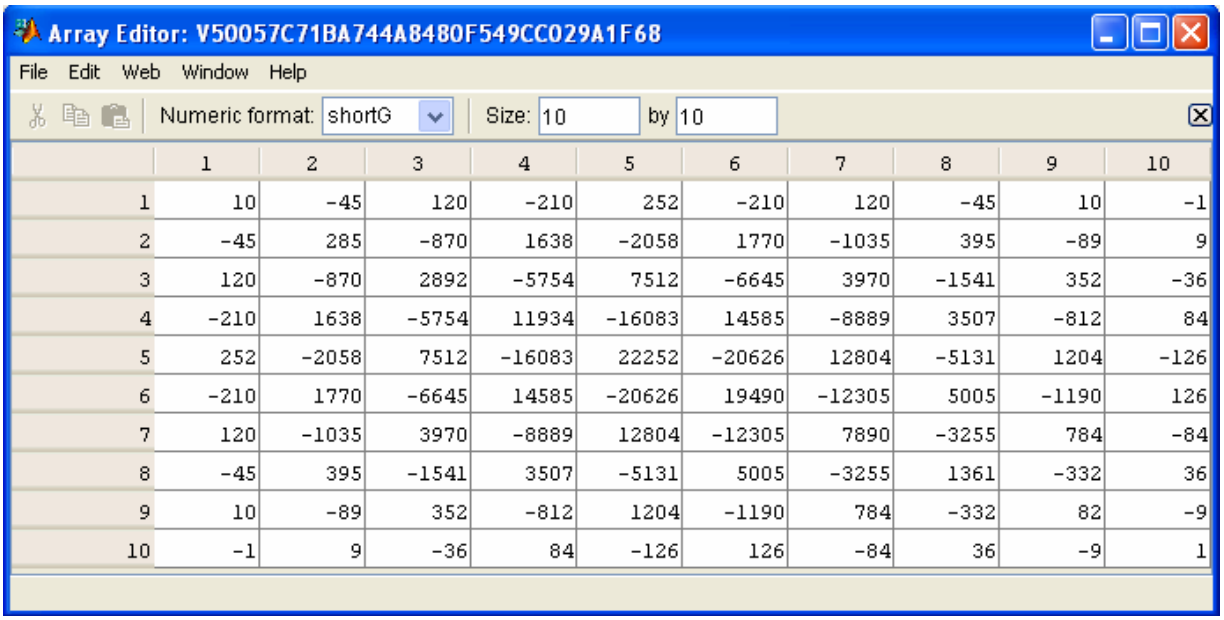
El sistema realizará la operación y generará un nombre aleatorio para la nueva matriz, dicho nombre se alojará en el panel superior derecho desde donde podrá ser consultado (ver Figura 6.18).



	1	2	3	4	5	6	7	8	9	10
1	10	-45	120	-210	252	-210	120	-45	10	-1
2	-45	285	-870	1638	-2058	1770	-1035	395	-89	9
3	120	-870	2892	-5754	7512	-6645	3970	-1541	352	-36
4	-210	1638	-5754	11934	-16083	14585	-8889	3507	-812	84
5	252	-2058	7512	-16083	22252	-20626	12804	-5131	1204	-126
6	-210	1770	-6645	14585	-20626	19490	-12305	5005	-1190	126
7	120	-1035	3970	-8889	12804	-12305	7890	-3255	784	-84
8	-45	395	-1541	3507	-5131	5005	-3255	1361	-332	36
9	10	-89	352	-812	1204	-1190	784	-332	82	-9
10	-1	9	-36	84	-126	126	-84	36	-9	1

Figura 6.18. Matriz Inversa de matriz Pascal.

La misma operación realizada en MATLAB nos arroja el resultado que se muestra en la Figura 6.19



	1	2	3	4	5	6	7	8	9	10
1	10	-45	120	-210	252	-210	120	-45	10	-1
2	-45	285	-870	1638	-2058	1770	-1035	395	-89	9
3	120	-870	2892	-5754	7512	-6645	3970	-1541	352	-36
4	-210	1638	-5754	11934	-16083	14585	-8889	3507	-812	84
5	252	-2058	7512	-16083	22252	-20626	12804	-5131	1204	-126
6	-210	1770	-6645	14585	-20626	19490	-12305	5005	-1190	126
7	120	-1035	3970	-8889	12804	-12305	7890	-3255	784	-84
8	-45	395	-1541	3507	-5131	5005	-3255	1361	-332	36
9	10	-89	352	-812	1204	-1190	784	-332	82	-9
10	-1	9	-36	84	-126	126	-84	36	-9	1

Figura 6.19. Matriz Inversa de matriz Pascal generada en MATLAB

De la misma forma, comparando los resultados del manipulador de matrices (Figura 6.18) y de la matriz generada en MATLAB (Figura 6.19) se puede observar que corresponden a los mismos valores casilla a casilla, con esto comprobamos que la operación “Generar Matriz Identidad no Cuadrada” del Manipulador de Matrices devuelve los resultados correctos

Como se mencionó al principio de este apartado, este tipo de pruebas se aplicaron a todas las operaciones que ofrece el Manipulador de Matrices, todas las operaciones presentaron el comportamiento esperado.

A su vez, como complemento de las pruebas de caja negra, el sistema fue probado con éxito por los profesores: F.M. Hilario Flores y M. Miguel Tirso, ambos profesores pertenecen al instituto de física y matemáticas de la UTM y dentro de sus actividades académicas se encuentra el uso recurrente de MATLAB.

En base a los resultados anteriores se libera la versión final del Manipulador de Matrices.

7. Conclusiones y expectativas

El uso de sistemas de cómputo en todos los ámbitos, es día a día más imperioso. Esta creciente necesidad conlleva a que el proceso de desarrollo de dichos sistemas sufra modificaciones en pro de depreciar los costos de desarrollo. Ante esto, la tecnología de objetos y de componentes supone un cambio radical a los anteriores paradigmas, ofreciendo un nuevo concepto aplicado a los sistemas de software: la reutilización.

En el proceso de reutilización, el desarrollador de los componentes reutilizables juega un papel muy importante, ya que los componentes que desarrolle serán la base de sistemas futuros, por lo que es importante crear componentes de software robustos y confiables.

En base a lo anterior, surge la motivación de desarrollar componentes de software orientados a las operaciones matriciales comúnmente usadas en el área académica, de investigación y en la industria, y a su vez, desarrollar una aplicación en la que fueran aplicados dichos componentes con la finalidad de mostrar el impacto del uso de los componentes en el desarrollo de software. En este capítulo, se presentan las conclusiones, contribuciones y limitaciones que se encontraron durante el desarrollo de esta tesis.

7.1. Conclusiones finales

Los componentes de software desarrollados en esta tesis cumplen su cometido, el cual es brindar a los desarrolladores de software una interfaz mediante la cual puedan desarrollar de manera más sencilla aplicaciones, en las que se encuentren involucradas operaciones matriciales. La característica principal de estos componentes es que son independientes del lenguaje de programación, y en el caso del componente SOAP es independiente también de la plataforma de desarrollo. Otra característica importante a señalar es que estos componentes de software están dirigidos a un amplio sector, ya que podrán ser ocupados no solo por los estudiantes y profesores-investigadores de la UTM, sino que debido a la naturaleza propia de los componentes de software, estarán disponibles para que cualquier persona pueda ocuparlos para el desarrollo de aplicaciones de software. Por lo anterior, se espera que estos componentes de software sean en un futuro, altamente utilizados como una herramienta en el desarrollo de sistemas de cómputo.

Por otro lado, la aplicación de software que de la misma manera se desarrolló en esta tesis, cumplió con las expectativas iniciales, las cuales consistían en comprobar que el uso de los componentes de software era un factor que disminuía los costos de desarrollo de software, ya que mediante una interfaz de usuario amigable, la aplicación abarca una gran cantidad de

operaciones que sin la ayuda de los componentes de software orientados a matrices hubiera sido en extremo difícil haber podido realizar. La principal ventaja de esta aplicación, radica en que ofrece la misma funcionalidad si se trabaja con una conexión local a MATLAB o con una conexión remota, lo que potencializa el uso de la misma aplicación.

El proceso de prueba realizado a los componentes y la aplicación de software en donde se vieron involucrados tanto el tesista, el director de la tesis y usuarios conocedores de MATLAB, da la pauta para considerar que se liberan productos de software estables y robustos.

En base a lo anterior, se puede concluir que se han cumplido con éxito los objetivos planteados inicialmente, sin embargo, tanto el sistema como los componentes de software tienen ciertas limitaciones, por lo que quedan abiertas líneas de investigación para que en trabajos futuros sean estudiadas y se trabaje sobre las mismas. A continuación se detallan aspectos sobre las aportaciones realizadas, las limitantes y las líneas de investigación que esta tesis propone.

7.2. Aportaciones realizadas

Las principales aportaciones de esta tesis son las siguientes:

- Una de las aportaciones más importantes es el desarrollo e implementación de los propios componentes de software, los cuales se espera sean utilizados primeramente por la comunidad universitaria de la UTM, en apoyo al desarrollo del software académico y científico que realizan los profesores-investigadores, así como el alumnado y, conforme se vaya difundiendo la importancia de los componentes, puedan ser utilizados por diversos grupos de desarrollo que así lo requieran.
- Otra aportación importante es el desarrollo e implementación del Manipulador de Matrices, herramienta mediante la cual los usuarios pueden crear diversos tipos de matrices y realizar diversas operaciones sobre las mismas. De la misma manera se muestra el impacto que el uso de los componentes de software tienen en los sistemas de cómputo.
- El valor agregado para la UTM que este trabajo conlleva al liberar componentes de software y un sistema de cómputo desarrollados a través de un proyecto propio y que podrá ser utilizado en gran medida por otros cuerpos académicos y grupos de desarrollo fuera de la Universidad, esto aumenta el prestigio de la propia Universidad y muestra que se encuentra al nivel de las principales instituciones de educación superior del país.
- Finalmente y no menos importante, este trabajo da pauta a varias líneas de investigación para la realización de artículos, tesis e investigaciones tecnológicas propias de la UTM, pudiendo abarcar las líneas de investigación que esta tesis comprende o creando nuevas que no hayan sido tratadas en las investigaciones y desarrollos realizados hasta el momento.

7.3. Limitaciones y líneas de investigación abiertas

Como se ha venido mencionando, esta tesis consistió en desarrollar un par de componentes de software así como una aplicación en la cual se probaran dichos componentes que cumpliesen con los requerimientos de software marcados en los respectivos. En este apartado, se reconocen algunas limitaciones que los componentes y la aplicación tienen, así como las líneas de investigación que directa e indirectamente se abren con la finalidad de atacar estas limitaciones y mejorar estos productos de software.

7.3.1. Limitaciones

Los componentes de software y la aplicación funcionan de manera correcta, como se ha demostrado mediante las pruebas de caja negra que se realizaron, sin embargo, cuentan con ciertas limitaciones propias de la primera implementación las cuales se listan a continuación

- **Tipos de datos soportados por el componente**: en esta primera implementación, los componentes está diseñados para trabajar sobre valores reales, lo que se considera una limitante ya que existen otros tipos de datos que se pueden tratar como son los valores imaginarios o los alfanuméricos.
- **Dimensiones de las matrices en la aplicación**: debido a la complejidad de esta tesis, esta primera versión del Manipulador de Matrices se diseñó para soportar solo matrices de dos dimensiones, lo cual limita de cierta manera al usuario que quiere manipular matrices multidimensionales.

7.3.2. Líneas de investigación abiertas

Con las limitantes que se mencionaron en el apartado anterior, este proyecto de tesis deja abiertas líneas de investigación para temas de tesis, artículos en la UTM, o cualquier forma de investigación que se pueda tratar.

La línea de investigación para la primera limitante se encuentra en la encapsulación de diversos tipos de datos de MATLAB usando la tecnología COM y SOAP, esta diversidad de tipos de datos se pueden aplicar en componentes orientados a diversos tipos de operaciones matemáticas[7].

Otra línea de investigación que continúa abierta es acerca del desarrollo de componentes de software basados en tecnologías como son COM y SOAP, se ha demostrado que la investigación y desarrollo de estas tecnologías puede resultar en productos de software flexibles capaces de interactuar en múltiples sistemas y plataformas de cómputo.

7.4. Expectativas futuras

Con los resultados obtenidos, se pretende motivar a la comunidad universitaria a que siga desarrollando componentes de software bajo tecnologías como las de la esta tesis.

[7] Actualmente se encuentra en desarrollo un par de tesis dentro del CASI que abarcan tanto la encapsulación de diversos tipos de datos soportados por MATLAB así como el desarrollo de componentes de software orientados al cálculo diferencial e integral.

Debido a que este desarrollo se encuentra debidamente documentado en procesos como análisis, diseño e implementación, se pretende que los siguientes puntos sean considerados como desarrollos futuros:

- Aumentar los tipos de datos soportados por los componentes, como son el soporte números imaginarios y valores alfanuméricos.
- Adicionar en el Manipulador de Matrices el soporte para matrices multidimensionales y matrices que soporten números imaginarios y valores alfanuméricos.
- Crear otros componentes orientados a otras ramas de las matemáticas como el cálculo, algebra, etc., de tal manera que interactúen entre si creando una amplia librería de componentes orientado a las matemáticas.
- Migrar de la dependencia de un sistema propietario como MATLAB, por un sistema de igual o mayor potencia y que sea de uso libre.

Estos puntos pueden presentar cierta dificultad técnica, pero deben ser tratados como desarrollos futuros y como una extensión al presente trabajo, ya que esta tesis cubre con éxito todos sus objetivos planteados originalmente.

Bibliografía

[1] Abascal J., Moriyón R (2002), "Tendencias en Interacción Persona Computador", Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial, No. 16, pp 9-24

[2] Basili V., Dieter H., (1988), "Towards a comprehensive framework for reuse: A reuse-enabling software evolution enviroment", University of Maryland.

[3] Bass L., Clements P., Kazman R., (2003), "Software Architecture in Practice", Addisson-Wesley.

[4] Bauer F. L., (1972), "Software Engineering", Information processing, 71, North Holland Publishing Co.

[5] Berard E., (1993), "Essays on Object-Oriented Software Engineering" Ed. Addison-Wesley, Unites Status of America.

[6] Biggerstaff, Ted J.(1998) "A Perspective of Generative Reuse". Annals of Software Engineering, Vol. 5, 1998, pp. 169-226.

[7] Bohem W., "Software Engineering", IEEE Transaction on Computers, C-25, num 12.

[8] Booch Grady, (1994), "Análisis y diseño orientado a objetos con aplicaciones" Segunda Edición, Addison-Wesley/Diaz de Santos.

[9] Booch, Grady, Rumbaugh, James & Jacobson, Ivar (2000) "El Lenguaje Unificado de Modelado". Addison Wesley Iberoamericana, Madrid, España.

[10] Box D., Ehnebuske D., Kakivaya G., Layman A., Mendelsohn N., Nielsen H., Thate S., Winer D., (2000), "Simple Access Object Protocol (SOAP) 1.1", Artìculo, W3C Note.

[11] Braun C., (), "Reuse". Marciniak pp. 1055, 1069.G

[12] Bredemeyer Dana & Malan Ruth, (2001) "Architecture Definitions". Artículo, Bredemeyer Consulting Págs. 3, 4.

[13] Bredenmeyer Dana, "Introduction to Software Architecture", Artículo, Bredenmeyer Consulting, pp 2, 15, 26.

[14] Clemente P., Sánchez F., Pérez M., (2002), "Modeling With UML Component-Based And Aspect Oriented Programming System", Quercus Software Engineering Group, Extremadura University, Spain.

[15] Coulson G., Gordon B., Grace P., Joolia A., Lee K., Ueyama J., (2002), "A Component Model For Building Systems Software", Computing Department, Lancaster University, UK.

[16] Deitel H. M., Deitel P. J. (1999), "C++ Cómo Programar", Prentice Hall Hispanoamericana, México, Segunda Edición, pp. 12-13.

[17] Diccionario de la Real Academia Española, (2003)"Real Academia Española", Diccionario en línea, Disponible en: http://buscon.rae.es/diccionario/drae.htm

[18] Ericsson H. E., Penker M., (1998), "UML Toolkit", Wiley.

[19] Ferré G., Sánchez S., "Desarrollo Orientado a Objetos con UML", Facultad de Informática, Universidad Politécnica de Madrid.

[20] Frontiers in Education Clearing House (2003). http://fie.engrng.pitt.edu

[21] Gamma E., Helm R., Johnson R., Vlssides J., "Design Patterns, Elements of Reusable Orient-Object Software", Addison Wesley Professional Computing Series.

[22] Gamma E., Helm R., Johnson R., Vlssides J., "Desingn Patterns: Abstraction and Reuse of Object Oriented Desing ", ECOOP '93 Conference Proccedings, Springer-Verlag Lecture Notes in Computer Sciences.

[23] García E., Gonzáles J., Pino D., Baltasar J., Martínez D., Pérez M., "Software educativo basado en componentes reutilizables: una propuesta para la integración de entornos propietarios". In proceedings of the 10mo. Congreso Internacional de Investigación en Ciencias Computacionales. Octubre 20-22, 2003 — Cuernavaca, México.

[24] García M., Moreira A., Rossi G., (2004), "UML: El Leguaje Estándar Para el Modelado del Software", Revista Novática/Upgrade, Número 168, pp. 4-5.

[25] Gonzáles A., (2002), "Integración de un modelo de proceso sistemático en el desarrollo de software educativo" Tesis de Doctorado, Universidad de Vigo, Área de Lenguajes y Sistemas informáticos, Málaga, España. Pag 147-153.

[26] Isaksson P., Part-Enander E., Sjoberg A., Melin B., (1996), "The MATLAB Handbook" , Addison-Wesley, New York.

[27] Jacobson, I., Christerson, M., and Jonsson, P. (1992) "Object-oriented software engineering; a Use Case driven approach". ACM Press, New York, 1992.

[28] McKim C, James, (2000), "Object Oriented Concepts", Artículo, Disponible en : http://www.rh.edu/~jcm/concepts.pdf

[29] Pressman R., (1998), "Ingeniería de Software un enfonque práctico", Ed. Mc Graw Hill/Interamericana de España, España.

[30] Pressman R., (2002) "Ingeniería de Software un enfoque práctico", Ed. Mc Graw Hill/Interamericana de España, España. Págs. 343, 348, 349, 362, 423

[31] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W. (1991) "Object Oriented Modeling and Design". Prentice Hall, 1991.

[32] Sametinger J., (1997), "Software Engineering with Reutilizable Components", Springer.

[33] Samuel Garrido, D. (2004), "Componentes", Disponible en: http://computacion.cs.cinvestav.mx/~sgarrido/cursos/ing_soft/Componentes/node16.html

[34] Schneider J. (1999), "Components, Scripts, and Glue: A conceptual framework for software composition", Inauguraldissertation der Philosophisch-naturwissenschaftlichen Fakult¨at der Universit¨at Bern

[35] Shaw M., Garlan D., (1996), "Software Architecture, Perspectivas On An Emerging Discipline", Prentice Hall.

[36] Sommerville Ian, (2002), "Ingeniería de Software", ed. Sexta, Ed. Pearson Educación, México. Pág. 230, 231, 261, 322, 323.

[37] Stroustrup Bjarne, (1991), "What is 'Object Oriented Programming'?", AT&T Bell Laboratorios, Murray Hill New Jersey.

[38] Suresh G., (1998), "The Component Object Model", disponible en http://my.execpc.com/~gopalan/com/com_ravings.html

[39] Szyperski, C., (1998), "Component Software – Beyond Object-Oriented Programming.", ACM Press / Addison-Wesley

[40] The MathWorks, (2002), "Help", Ayuda en línea de MATLAB 6.5.

[41] Universidad Tecnológica de la Mixteca (2004), "Cuerpo Académico de Ingeniería de Software", Disponible en: http://mixtli.utm.mx/~casi/

[42] Vallecillo M., A. (1999) "Un modelo de componentes para el desarrollo de aplicaciones distribuidas." Tesis de Doctorado, Universidad de Vigo, Área de Lenguajes y Sistemas informáticos, Málaga, España Pág.13, 16, 17.

[43] Wiley Periodicals Inc. (2002) Computer Applications in Engineering Education. ISSN: 1061-3773

[44] Williams S., Kindel C., (1994), "The component Object Model: A Technical Overview", Microsoft Corporation.

[45] Zimmer W., "Relationships between Desing Patterns", Forschungszentrum Informatik, Bereich Programmstrukturen, Alemania.

Sitios de Internet

[URL 1]. http://www.webopedia.com "Webopedia: Online Computer Dictionary for Computer and Internet Terms and Definitions" Internet.com, Jupitermedia Corporation 2004.

[URL 2]. http://www.microsoft.com/com/default.mspx "COM: Component Object Model Technologies" Microsoft Corporation 2005.

[URL 3]. http://www.w3.org/TR/2000/NOTE-SOAP-20000508/ "Simple Object Access Protocol (SOAP) 1.1" W3C Note 2000.

[URL 4]. http://www.xml.com/pub/a/98/10/guide0.html?page=2#AEN58 "What is XML" O'Reilly Media, Inc.

Anexo 1. Glosario

- **ActiveX**, conjunto de tecnologías desarrolladas por Microsoft para compartir información entre diferentes aplicaciones [URL 1].
- **Application Program Interface (API)**, es un conjunto de rutinas, protocolos, y herramientas para la construcción de aplicaciones software. Un buen API hace que sea fácil desarrollar un programa que provea todos los bloques de construcción [URL 1].
- **Component Object Model (COM)**, tecnología de la familia de sistemas operativos Microsoft Windows que permite comunicar los componentes de software. COM es usado por los desarrolladores para crear componentes de software reusables, ligar componentes para construir aplicaciones y tomar las ventajas de los servicios de Windows [URL 2].
- **Delphi**, un sistema RAD desarrollado por Borland International Inc. Similar a Visual Basic pero basado en Pascal [URL 1].
- **Simple Object Access Protocol (SOAP)**, es un protocolo basado en XML para cambiar información en un ambiente descentralizado y distribuido. SOAP puede ser usado en combinación con otra variedad de protocolos [URL 3].
- **Rapid Application Development (RAD)**, sistema de programación que permite a los desarrolladores la construcción rápida de programas. Los sistemas RAD proveen herramientas para ayudar a construir interfaces gráficas que normalmente tomarían un largo esfuerzo de desarrollo [URL 1].
- **XML**, es un lenguaje de marcado para documentos que contienen información estructurada. La especificación XML define un estándar para agregar etiquetas a documentos [URL 4].

Anexo 2. Acrónimos

Acrónimo	Definición
API	Application Program Interface (Interfaz de Programación de Aplicaciones)
CASI	Cuerpo Académico de Ingeniería de Software
COM	Component Object Model
COP	Component Oriented Programming (Programación Orientada a Componentes)
ERS	Especificación de Requerimientos de Software
IEEE	Institute of Electrical and Electronics Engineers
MATLAB	Matrix Laboratory
MOO	Métodos Orientados a Objetos
MVC	Modelo Vista Controlador
OMG	Object Management Group
OMT	Object Modeling Technique
OO	Object Oriented
OOSE	Object-Oriented Software Engineering
RUP	Rational Unified Process
SOAP	Simple Access Object Protocol
UML	Unified Modeling Language
URD	Documento de Requerimientos de Usuario
UTM	Universidad Tecnológica de la Mixteca
XML	Extensible Markup Language

Anexo 3. Extensión de especificación de casos de uso para los componentes de software

Especificación de caso de uso: Gestionar IMOCOMMatlabMatrixOperationLib

Versión 1.2

Historial de revisiones

Fecha	Versión	Descripción	Autor
26/Octubre/2004	1.0	Creación del Documento	Felipe de Jesús García Pérez
05/Noviembre/2004	1.1	Introducción de primeros diagramas	Felipe de Jesús García Pérez
10/Noviembre/2004	1.2	Documento completado	Felipe de Jesús García Pérez

Especificación de casos de uso: Gestionar ICOMMatlabMatrixOpLib.

Escenario: Obtener la matriz transpuesta.

Objetivo

Gestionar la interfaz IMOCOMMatlabMatrixOperationLib obtener la matriz transpuesta de una matriz.

Breve descripción

El usuario podrá obtener la matriz transpuesta de una matriz numérica gestionando la interfaz ISOAPMatlabMatrixOperationLib y ejecutando la función "Transpose".

Actores

Usuario

Flujo de eventos

Flujo básico

Actor	*Sistema*
Este caso de uso comienza cuando el usuario tiene instanciada la interfaz IMO-COMMatlabMatrixOperationLib, tiene creada una matriz cuadrada y el usuario desea obtener la matriz transpuesta de dicha matriz.	
El usuario ejecuta la función "Transpose" con los parámetros para definir el nombre de la matriz a la cual se va a realizar la operación y el nombre de la matriz en donde se va a depositar el resultado.	
	El sistema crea la matriz permaneciendo el resultad de la operación dentro de MATLAB en una matriz con el nombre del último parámetro de entrada.
Finaliza el caso de uso.	

Excepción

Si la interfaz no pudo ser instanciada se notificará mediante un mensaje de error.

Si la operación no pudo ser realizada se notificará mediante un mensaje de error.

Precondiciones

El usuario deberá haber iniciado una sesión en MATLAB.

Diagrama de actividades

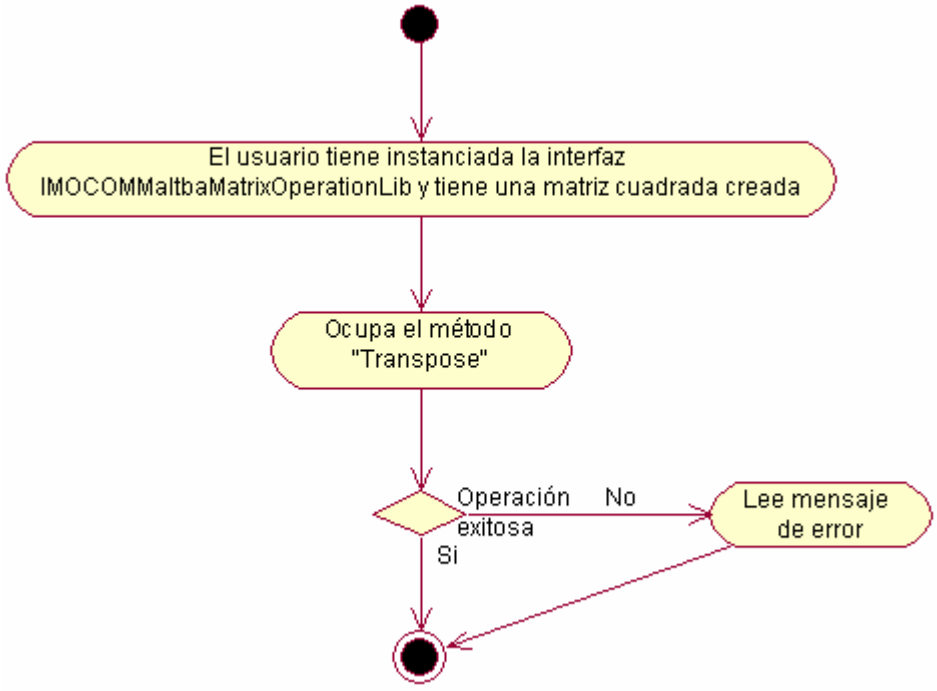


Diagrama de secuencia

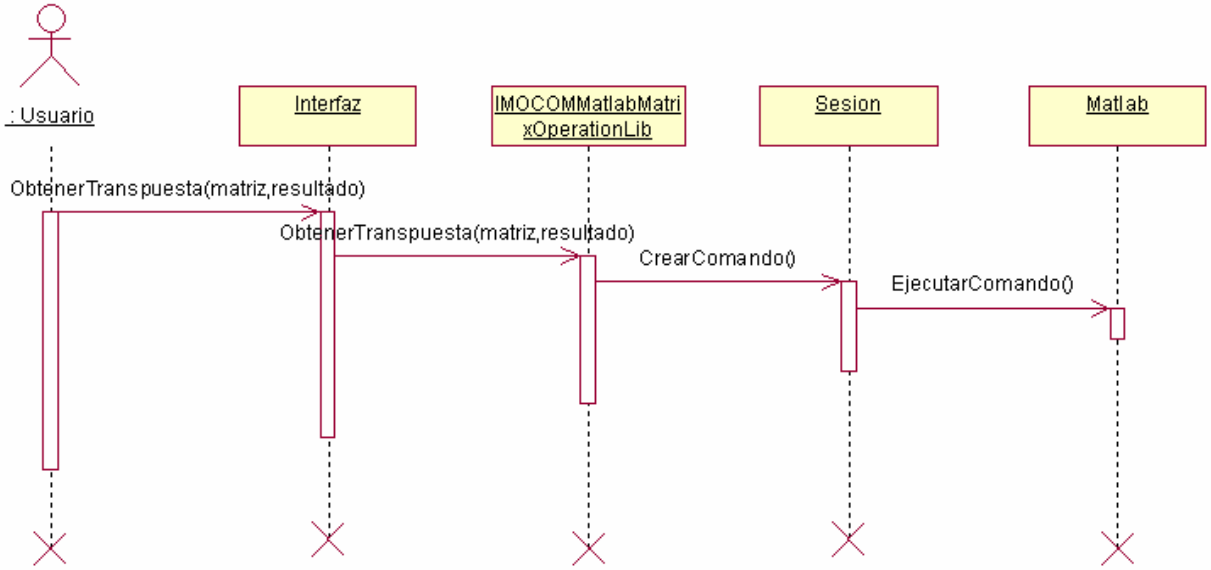


Diagrama de colaboración

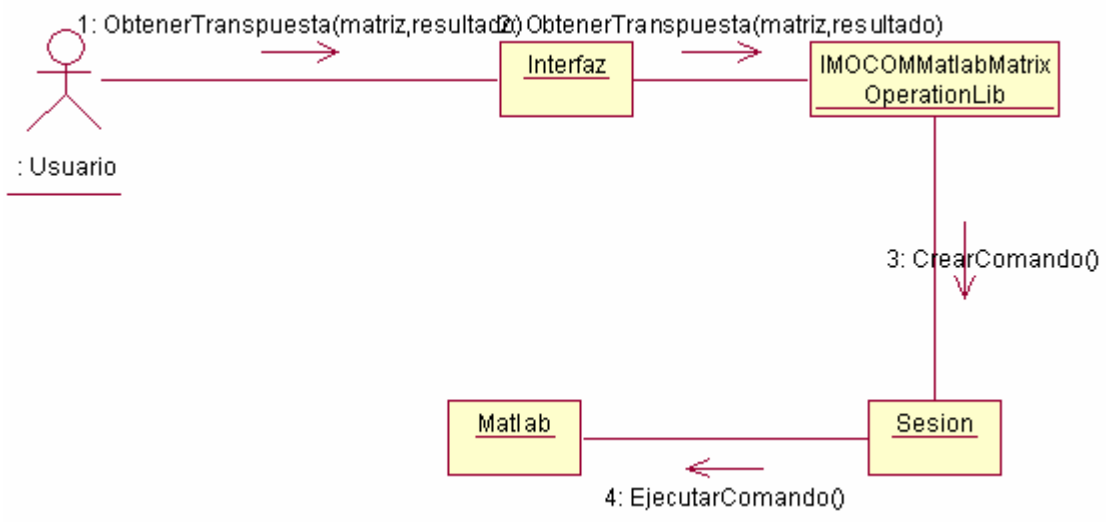


Diagrama de estados

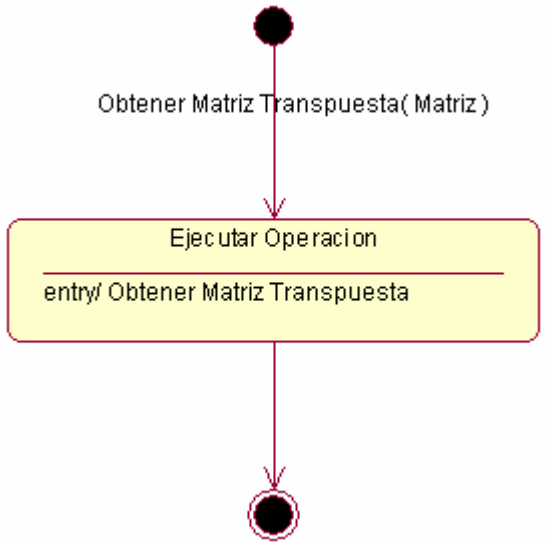
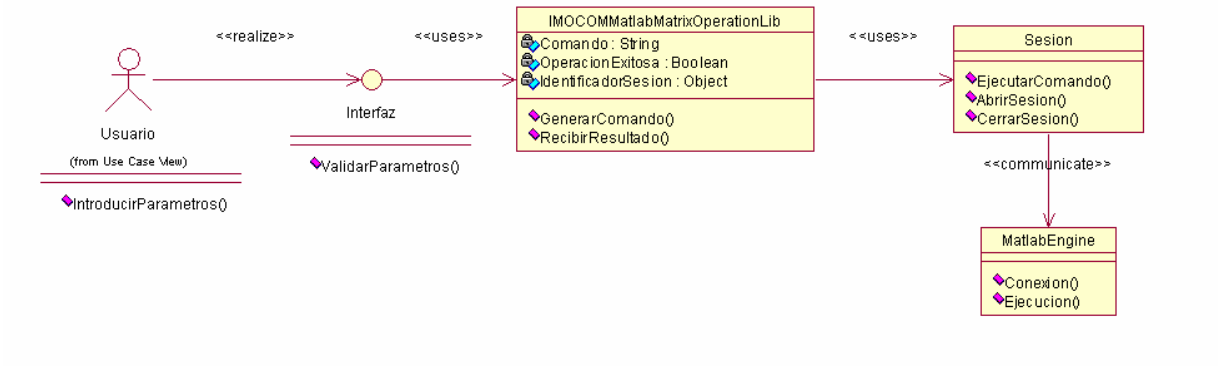


Diagrama de Clases.



Especificación de caso de uso: Gestionar IMOCOMMatlabMatrix
Versión 1.2

Especificación de casos de uso: Gestionar IMOCOMMatlabMatrix.

Escenario: Obtener determinante como matriz.

Objetivo

Gestionar la interfaz IMOCOMMatlabMatrix obtener el determinante de una matriz y depositar el resultado en una matriz.

Breve descripción

El usuario podrá obtener en una matriz el resultado del determinante de una matriz numérica gestionando la interfaz ISOAPMatlabMatrix y ejecutando la función "GetDeterminantAsMatrix".

Actores

Usuario

Flujo de eventos

Flujo básico

Actor	*Sistema*
Este caso de uso comienza cuando el usuario tiene instanciada la interfaz IMOCOMMatlabMatrix, tiene creada una matriz cuadrada y el usuario desea obtener el determinante de dicha matriz y depositar el resultado en una matriz.	
El usuario ejecuta la función "GetDeterminantAsMatrix" con los parámetros para definir si el resultado de la operación se va a depositar en la misma matriz o en una matriz diferente.	
	El sistema realiza la operación, si el parámetro de entrada indica que el resultado se deposite en una nueva matriz el sistema crea una nueva matriz y deposita el resultado en dicha matriz, en caso contrario deposita el resultado en la matriz

	actual.
	El sistema devuelve un apuntador hacia la matriz donde depositó el resultado.
Finaliza el caso de uso.	

Excepción

Si la interfaz no pudo ser instanciada se notificará mediante un mensaje de error.

Si la operación no pudo ser realizada se notificará mediante un mensaje de error.

Precondiciones

El usuario deberá haber iniciado una sesión en MATLAB.

Diagrama de actividades

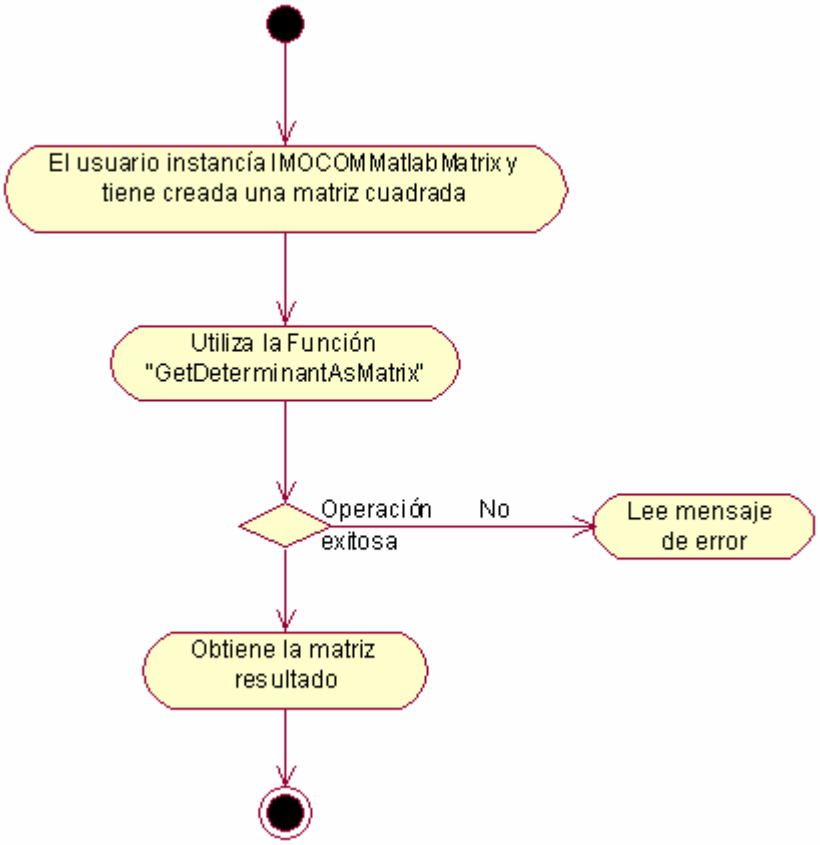


Diagrama de secuencia

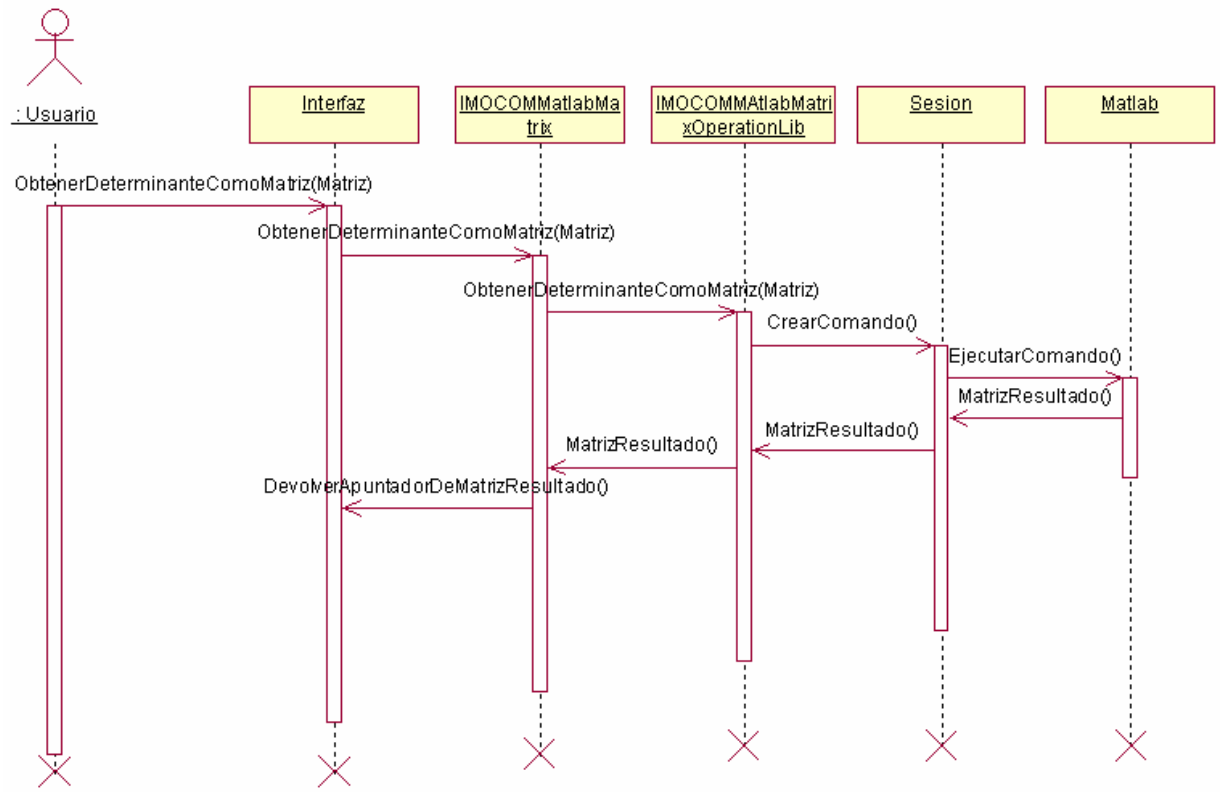


Diagrama de colaboración

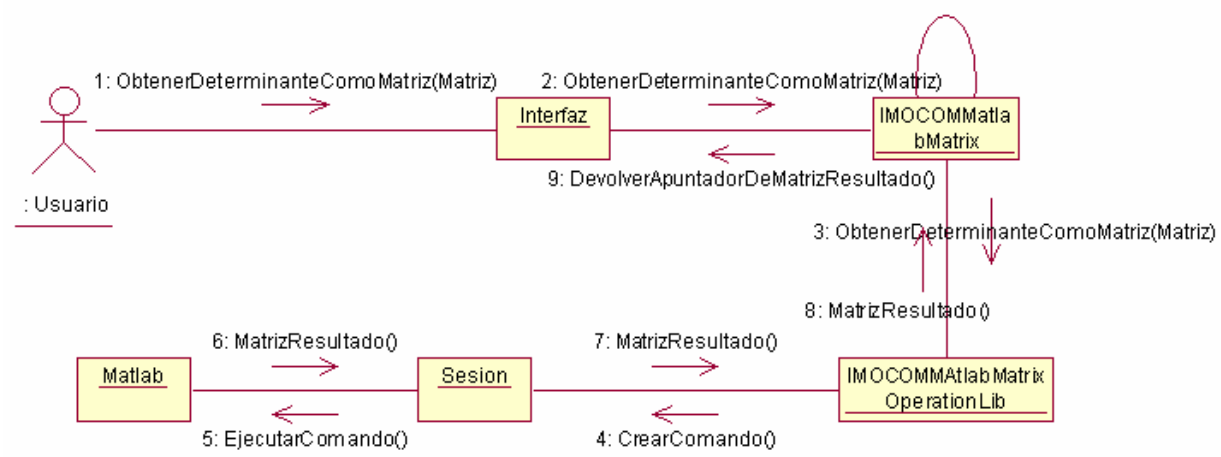


Diagrama de estados

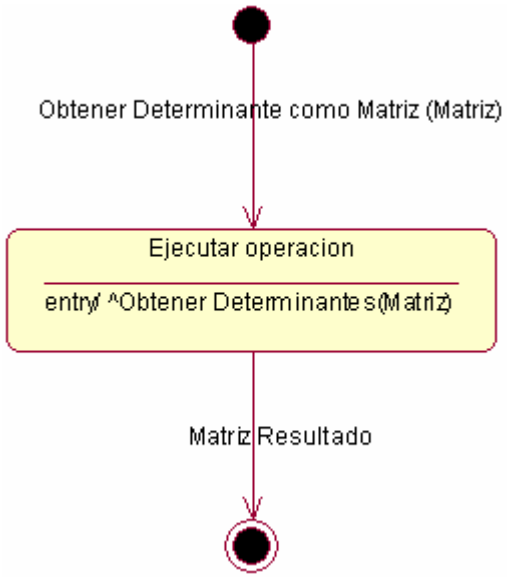
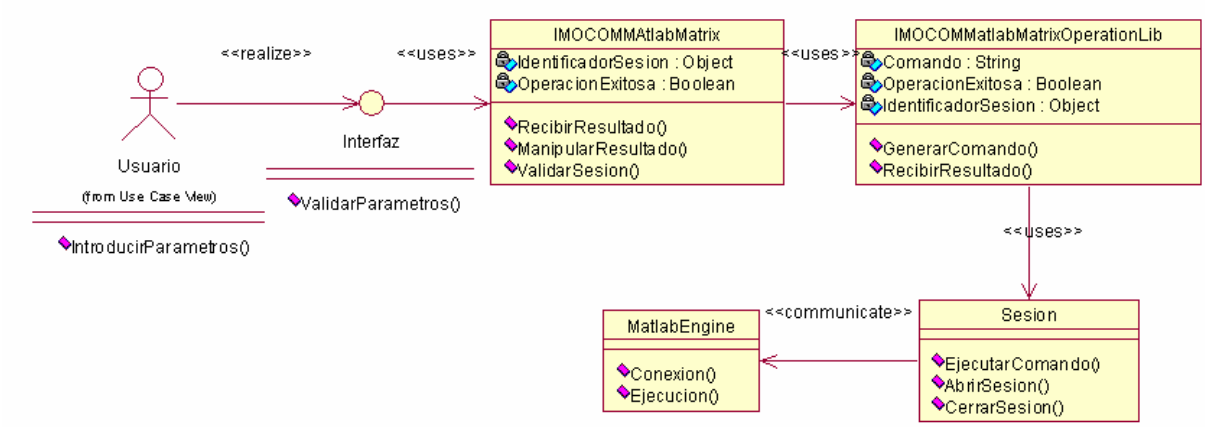


Diagrama de Clases.



El resto de extensión de casos de uso tanto de los componentes de software como de la aplicación “Manipulador de Matrices”, se encuentran en formato digital en el archivo *extensión de casos de uso.pdf* en el CD-ROM adjunto a este trabajo de tesis.

Anexo 4. Extensión de la definición de interfaces del capítulo 5

En la siguiente tabla se muestran todos los métodos de la interfaz IMOMatlabMatrixOperationLib

Tabla 5.6. Métodos de la interfaz IMOMatlabMatrixOperationLib

IMOMatlabMatrixOperationLib			
Nombre del método	***Parámetros de entrada***	***Tipo de parámetro***	***Salida***
Add	Matriz1	String	-
	Matriz2	String	
	ResultName	String	
Chol	Matriz1	String	-
	ResultName	String	
CumSum	Matriz1	String	-
	Dimension	Integer	
	ResultName	String	
Det	Matriz1	String	-
	ResultName	String	
Diag	Matriz1	String	-
	ResultName	String	
Divide	Matriz1	String	-
	Matriz2	String	
	ResultName	String	
Eye	Number	Integer	-
	ResultName	String	
Exponential	Matriz1	String	-
	Power	Integer	
	ResultName	String	
Find	Matriz1	String	-
	ResultName	String	

FlipDim	Matriz1	String	-
	Dimension	Integer	
	ResultName	String	
FlipLR	Matriz1	String	-
	ResultName	String	
FlipUD	Matriz1	String	-
	ResultName	String	
Hadamard	Dimension	Integer	-
	ResultName	String	
Hankel	Matriz1	String	-
	ResultName	String	
Hilbert	Dimension	Integer	-
	ResultName	String	
HorzCat	Matriz1	String	-
	Matriz2	String	
	ResultName	String	
IdentityNotSquare	X	Integer	-
	Y	Integer	
	ResultName	String	
Inv	Matriz1	String	-
	ResultName	String	
InvHilb	Matriz1	String	-
	ResultName	String	
IsEmpty	Matriz1	String	-
	ResultName	String	
Kron	Matriz1	String	-
	Matriz2	String	
	ResultName	String	
Lu	Matriz1	String	-
	ResultName	String	
Magic	Dimension	Integer	-
	ResultName	String	
Max	Matriz1	String	-
	ResultName	String	
Median	Matriz1	String	-
	ResultName	String	
Min	Matriz1	String	-
	ResultName	String	

Mult	Matriz1	String	-
	Matriz2	String	
	ResultName	String	
Norm	Matriz1	String	-
	ResultName	String	
OnesMultiDim	Dims	Variant	-
	ResultName	String	
OnesXYMatrix	X	Integer	-
	Y	Integer	
	ResultName	String	
OnesXYZ	X	Integer	-
	Y	Integer	
	Z	Integer	
	ResultName	String	
Pascal	Dimension	Integer	-
	ResultName	String	
PInv	Matriz1	String	-
	ResultName	String	
ProdSum	Matriz1	String	-
	Dimension	Integer	
	ResultName	String	
Qr	Matriz1	String	-
	ResultName	String	
RandMultiDim	Dims	Variant	-
	ResultName	String	
RandNMultidim	Dims	Variant	-
	ResultName	String	
RandXY	X	Integer	-
	Y	Integer	
	ResultName	String	
RandXYZ	X	Integer	-
	Y	Integer	
	Z	Integer	
	ResultName	String	
RandnXY	X	Integer	-
	Y	Integer	
	ResultName	String	
RandNXYZ	X	Integer	-

	Y	Integer	
	Z	Integer	
	ResultName	String	
Rank	Matriz1	String	-
	ResultName	String	
Rcond	Matriz1	String	-
	ResultName	String	
Rosser	ResultName	String	-
Rot90	Matriz1	String	-
	ResultName	String	
Schur	Matriz1	String	-
	ResultName	String	
Size	Matriz1	String	-
	ResultName	String	
Sort	Matriz1	String	-
	ResultName	String	
SqrtM	Matriz1	String	-
	ResultName	String	
Substract	Matriz1	String	-
	Matriz2	String	
	ResultName	String	
SVD	Matriz1	String	-
	ResultName	String	
Transpose	Matriz1	String	-
	ResultName	String	
TransposeNonConjugate	Matriz1	String	-
	ResultName	String	
TriL	Matriz1	String	-
	ResultName	String	
TriU	Matriz1	String	-
	ResultName	String	
VertCat	Matriz1	String	-
	Matriz2	String	
	ResultName	String	
Wilkinson	Dimension	Integer	-
	ResultName	String	
ZerosMultiDim	Dims	Variant	-
	ResultName	String	

ZerosXYZ	X	Integer	-
	Y	Integer	
	Z	Integer	
	ResultName	String	
ZerosXYMatrix	X	Integer	-
	Y	Integer	
	ResultName	String	

En la siguiente tabla se muestran todos los métodos de la interfaz ISOAP-MatlabMatrix

Tabla 5.7. Métodos de la interfaz ISOAPMatlabMatrix

ISOAPMatlabMatrix			
Nombre del método	***Parámetros de entrada***	***Tipo de parámetro***	***Salida***
AddMatrix	IDSesion	String	String
	Matrix	String	
	CreateNew	Boolean	
AddVar	IDSesion	String	String
	VarName	String	
	CreateNew	Boolean	
	NewMatrixName	String	
Balance	IDSesion	String	String
	CreateNew	Boolean	
CholeskyFactor	IDSesion	String	String
	CreateNew	Boolean	
CumulativeAdd	IDSesion	String	String
	Dimension	Integer	
	CreateNew	Boolean	
CumulativeProd	IDSesion	String	String
	Dimension	Integer	
	CreateNew	Boolean	
DiagonalOfMatrix	IDSesion	String	String
	CreateNew	Boolean	
DivideIntoMatrix	IDSesion	String	String
	Matrix	String	
	CreateNew	Boolean	
DivideIntoVar	IDSesion	String	String
	VarName	String	
	CreateNew	Boolean	
	NewMatrixName	String	
DivideMatrix	IDSesion	String	String
	Matrix	String	
	CreateNew	Boolean	
DivideVar	IDSesion	String	String
	VarName	String	
	CreateNew	Boolean	
	NewMatrixName	String	

Exponential	IDSesion	String	String
	Power	Integer	
	CreateNew	Boolean	
Find	IDSesion	String	String
	CreateNew	Boolean	
FlipLeftToRight	IDSesion	String	String
	CreateNew	Boolean	
FlipNDimensions	IDSesion	String	String
	Dimension	Integer	
	CreateNew	Boolean	
FlipUpToDown	IDSesion	String	String
	CreateNew	Boolean	
GenerateIdentityMatrix	IDSesion	String	String
	Dimension	Integer	
	CreateNew	Boolean	
GenerateHadamardMatrix	IDSesion	String	String
	Dimension	Integer	
	CreateNew	Boolean	
GenerateHadamardMatrixAsVar	IDSesion	String	String
	Dimension	Integer	
	CreateNew	Boolean	
	NewVarName	String	
GenerateHilbertMatrix	IDSesion	String	String
	Dimension	Integer	
	CreateNew	Boolean	
GenerateHilbertMatrixAsVar	IDSesion	String	String
	Dimension	Integer	
	CreateNew	Boolean	
	NewVarName	String	
GenerateIdentityMatrixAsVar	IDSesion	String	String
	Dimension	Integer	
	CreateNew	Boolean	
	NewVarName	String	
GenerateIdentityNotSquareMatrix	IDSesion	String	String
	X	Integer	
	Y	Integer	
	CreateNew	Bolean	
GenerateIdentityNotS-	IDSesion	String	String

quareMatrixAsVar	X	Integer	
	Y	Integer	
	CreateNew	Boolean	
	NewVarName	String	
GenerateInverseHilbert-Matrix	IDSesion	String	String
	Dimension	Integer	
	CreateNew	Boolean	
GenerateInverseHilbert-MatrixAsVar	IDSesion	String	String
	Dimension	Integer	
	CreateNew	Bolean	
	NewVarName	String	
GenerateMagicMatrix	IDSesion	String	String
	Dimension	Integer	
	CreateNew	Boolean	
GenerateMagicMa-trixAsVar	IDSesion	String	String
	Dimension	Integer	
	CreateNew	Boolean	
	NewVarName	String	
GenerateOnesMultiDim-Matrix	IDSesion	String	String
	Dims	TRemotableMultiDim	
	CreateNew	Boolean	
GenerateOnesMultiDim-MatrixAsVar	IDSesion	String	String
	Dims	TRemotableMultiDim	
	NewVarName	Boolean	
GenerateOnesXYMatrix	IDSesion	String	String
	X	Integer	
	Y	Integer	
	CreateNew	Bolean	
GenerateOnesXYMa-trixAsVar	IDSesion	String	String
	X	Integer	
	Y	Integer	
	NewVarName	String	
GenerateOnesXYZMa-trixAsVar	IDSesion	String	String
	X	Integer	
	Y	Integer	
	Z	Integer	
	NewVarName	String	
GenerateOnesXYZMatrix	IDSesion	String	String

	X	Integer	
	Y	Integer	
	Z	Integer	
	CreateNew	Bolean	
GeneratePascalMatrix	IDSesion	String	String
	Dimension	Integer	
	CreateNew	Boolean	
GeneratePascalMatrixAsVar	IDSesion	String	String
	Dimension	Integer	
	NewVarName	String	
GenerateRandomMultiDimMatrix	IDSesion	String	String
	Dims	TRemotableMultiDim	
	CreateNew	Boolean	
GenerateRandomMultiDimMatrixAsVar	IDSesion	String	String
	Dims	TRemotableMultiDim	
	NewVarName	Boolean	
GenerateRandomXYMatrix	IDSesion	String	String
	X	Integer	
	Y	Integer	
	CreateNew	Bolean	
GenerateRandomMatrixXYAsVar	IDSesion	String	String
	X	Integer	
	Y	Integer	
	NewVarName	String	
GenerateRandomXYZMatrix	IDSesion	String	String
	X	Integer	
	Y	Integer	
	Z	Integer	
	CreateNew	Bolean	
GenerateRandomXYZMatrixAsVar	IDSesion	String	String
	X	Integer	
	Y	Integer	
	Z	Integer	
	NewVarName	String	
GenerateRosserMatrix	IDSesion	String	String
	CreateNew	Bolean	
GenerateRosserMatrixAsVar	IDSesion	String	String
	NewVarName	String	

GenerateUniformRandomMultiDimMatrix	IDSesion	String	String
	Dims	TRemotableMultiDim	
	CreateNew	Boolean	
GenerateUniformRandomMultiDimMatrixAsVar	IDSesion	String	String
	Dims	TRemotableMultiDim	
	NewVarName	Boolean	
GenerateUniformRandomXYMatrix	IDSesion	String	String
	X	Integer	
	Y	Integer	
	CreateNew	Bolean	
GenerateUniformRandomXYMatrixAsVar	IDSesion	String	String
	X	Integer	
	Y	Integer	
	NewVarName	String	
GenerateUniformRandomXYZMatrix	IDSesion	String	String
	X	Integer	
	Y	Integer	
	Z	Integer	
	CreateNew	Bolean	
GenerateUniformRandomXYZMatrixAsVar	IDSesion	String	String
	X	Integer	
	Y	Integer	
	Z	Integer	
	NewVarName	String	
GenerateWilkinsonMatrix	IDSesion	String	String
	Dimension	Integer	
	CreateNew	Boolean	
GenerateWilkinsonMatrixAsVar	IDSesion	String	String
	Dimension	Integer	
	NewVarName	String	
GenerateZerosMultiDimMatrix	IDSesion	String	String
	Dims	TRemotableMultiDim	
	CreateNew	Boolean	
GenerateZerosMultiDimMatrixAsVar	IDSesion	String	String
	Dims	TRemotableMultiDim	
	NewVarName	Boolean	
GenerateZerosXYMatrix	IDSesion	String	String
	X	Integer	

	Y	Integer	
	CreateNew	Bolean	
GenerateZerosXYZMatrixAsVar	IDSesion	String	String
	X	Integer	
	Y	Integer	
	Z	Integer	
	NewVarName	String	
GenerateZerosXYMatrixAsVar	IDSesion	String	String
	X	Integer	
	Y	Integer	
	NewVarName	String	
GenerateZerosXYZMatrix	IDSesion	String	String
	X	Integer	
	Y	Integer	
	Z	Integer	
	CreateNew	Bolean	
GetDeterminant	IDSesion	String	Double
GetDeterminantAsMatrix	IDSesion	String	String
	CreateNew	Bolean	
GetDeterminantAsVar	IDSesion	String	String
	VarName	String	
GetMatrixSize	IDSesion	String	String
	CreateNew	Bolean	
GetNorm	IDSesion	String	Double
GetNormAsMatrix	IDSesion	String	String
	CreateNew	Bolean	
GetNormAsVar	IDSesion	String	String
	VarName	String	
GetRank	IDSesion	String	Double
GetRankAsMatrix	IDSesion	String	String
	CreateNew	Bolean	
GetRankAsVar	IDSesion	String	String
	VarName	String	
GetRcond	IDSesion	String	Double
GetRconAsMatrix	IDSesion	String	String
	CreateNew	Bolean	
GetRcondAsVar	IDSesion	String	String
	VarName	String	

GetTrace	IDSesion	String	Double
	IDSesion	String	
GetTraceAsMatrix	IDSesion	String	String
	CreateNew	Bolean	
GetTraceAsVar	IDSesion	String	String
	VarName	String	
HankelMatrix	IDSesion	String	String
	CreateNew	Bolean	
HankelMatrixAsVar	IDSesion	String	String
	NewVarName	String	
Hessenberg	IDSesion	String	String
	CreateNew	Bolean	
HorizontalConcat	IDSesion	String	String
	Matrix	String	
	CreateNew	Boolean	
HorzcatFromMatrix	IDSesion	String	String
	Matrix	String	
	CreateNew	Boolean	
HorizontalCatVar	IDSesion	String	String
	VarName	String	
	CreateNew	Boolean	
	NewVarName	String	
HorizontalCatFromVar	IDSesion	String	String
	VarName	String	
	CreateNew	Boolean	
	NewVarName	String	
Inverse	IDSesion	String	String
	CreateNew	Bolean	
IsMatrixEmpty	IDSesion	String	Boolean
Kronecker	IDSesion	String	String
	Matrix	String	
	CreateNew	Boolean	
KronIntoMatrix	IDSesion	String	String
	MatrixName	String	
	CreateNew	Boolean	
	NewVarName	String	
KronMatrix	IDSesion	String	String
	MatrixName	String	

	CreateNew	Boolean	
	NewVarName	String	
KronVar	IDSesion	String	String
	VarName	String	
	CreateNew	Boolean	
	NewVarName	String	
LuFactor	IDSesion	String	String
	CreateNew	Bolean	
Maximum	IDSesion	String	String
	CreateNew	Bolean	
Median	IDSesion	String	String
	CreateNew	Bolean	
Minimum	IDSesion	String	String
	CreateNew	Bolean	
MultMatrix	IDSesion	String	String
	Matrix	String	
	CreateNew	Boolean	
MultIntoMatrix	IDSesion	String	String
	Matrix	String	
	NewMatrixName	String	
MultVar	IDSesion	String	String
	VarName	String	
	CreateNew	Boolean	
	NewVarName	String	
Orthogonal	IDSesion	String	String
	CreateNew	Bolean	
PseudoInverse	IDSesion	String	String
	CreateNew	Bolean	
QrFactor	IDSesion	String	String
	CreateNew	Bolean	
Rotate90Degrees	IDSesion	String	String
	CreateNew	Bolean	
SchurDescomposition	IDSesion	String	String
	CreateNew	Bolean	
SingularValueDescompo-sition	IDSesion	String	String
	CreateNew	Bolean	
SortMatrix	IDSesion	String	String
	CreateNew	Bolean	

SqrtMatrix	IDSesion	String	String
	CreateNew	Bolean	
SubstractFromMatrix	IDSesion	String	String
	Matrix	String	
	CreateNew	Boolean	
SubstractFromVar	IDSesion	String	String
	VarName	String	
	CreateNew	Boolean	
	NewVarName	String	
SubstractMatrix	IDSesion	String	String
	Matrix	String	
	CreateNew	Boolean	
SubstractVar	IDSesion	String	String
	VarName	String	
	CreateNew	Boolean	
	NewVarName	String	
Transpose	IDSesion	String	String
	CreateNew	Boolean	
TransposeNonConjugate	IDSesion	String	String
	CreateNew	Boolean	
TriangularLowPart	IDSesion	String	String
	CreateNew	Boolean	
TriangularUpPart	IDSesion	String	String
	CreateNew	Boolean	
VerticalConcat	IDSesion	String	String
	Matrix	String	
	CreateNew	Boolean	
VerticalCatFromVar	IDSesion	String	String
	VarName	String	
	CreateNew	Boolean	
	NewVarName	String	
VertcatFromMatrix	IDSesion	String	String
	Matrix	String	
	CreateNew	Boolean	
VerticalCatVar	IDSesion	String	String
	VarName	String	
	CreateNew	Boolean	
	NewVarName	String	

En la siguiente tabla se muestran todos los métodos de la interfaz ISOAPMatlabMatrixOperationLib

Tabla 5.8. Métodos de la interfaz ISOAPMatlabMatrixOperationLib

ISOAPMatlabMatrixOperationLib			
Nombre del método	***Parámetros de entrada***	***Tipo de parámetro***	***Salida***
Add	IDSesion	String	-
	Matriz1	String	
	Matriz2	String	
	ResultName	String	
Chol	IDSesion	String	-
	Matriz1	String	
	ResultName	String	
CumSum	IDSesion	String	-
	Matriz1	String	
	Dimension	Integer	
	ResultName	String	
Det	IDSesion	String	-
	Matriz1	String	
	ResultName	String	
Diag	IDSesion	String	-
	Matriz1	String	
	ResultName	String	
Divide	IDSesion	String	-
	Matriz1	String	
	Matriz2	String	
	ResultName	String	
Eye	IDSesion	String	-
	Number	Integer	
	ResultName	String	
Exponential	IDSesion	String	-
	Matriz1	String	
	Power	Integer	
	ResultName	String	
Find	IDSesion	String	-
	Matriz1	String	
	ResultName	String	
FlipDim	IDSesion	String	-

	Matriz1	String	
	Dimension	Integer	
	ResultName	String	
FlipLR	IDSesion	String	-
	Matriz1	String	
	ResultName	String	
FlipUD	IDSesion	String	-
	Matriz1	String	
	ResultName	String	
Hadamard	IDSesion	String	-
	Dimension	Integer	
	ResultName	String	
Hankel	IDSesion	String	-
	Matriz1	String	
	ResultName	String	
Hilbert	IDSesion	String	-
	Dimension	Integer	
	ResultName	String	
HorzCat	IDSesion	String	-
	Matriz1	String	
	Matriz2	String	
	ResultName	String	
IdentityNotSquare	IDSesion	String	-
	X	Integer	
	Y	Integer	
	ResultName	String	
Inv	IDSesion	String	-
	Matriz1	String	
	ResultName	String	
InvHilb	IDSesion	String	-
	Matriz1	String	
	ResultName	String	
IsEmpty	IDSesion	String	-
	Matriz1	String	
	ResultName	String	
Kron	IDSesion	String	-
	Matriz1	String	
	Matriz2	String	

	ResultName	String	
Lu	IDSesion	String	-
	Matriz1	String	
	ResultName	String	
Magic	IDSesion	String	-
	Dimension	Integer	
	ResultName	String	
Max	IDSesion	String	-
	Matriz1	String	
	ResultName	String	
Median	IDSesion	String	-
	Matriz1	String	
	ResultName	String	
Min	IDSesion	String	-
	Matriz1	String	
	ResultName	String	
Mult	IDSesion	String	-
	Matriz1	String	
	Matriz2	String	
	ResultName	String	
Norm	IDSesion	String	-
	Matriz1	String	
	ResultName	String	
OnesMultiDim	IDSesion	String	-
	Dims	TRemotableMultiDim	
	ResultName	String	
OnesXYMatrix	IDSesion	String	-
	X	Integer	
	Y	Integer	
	ResultName	String	
OnesXYZ	IDSesion	String	-
	X	Integer	
	Y	Integer	
	Z	Integer	
	ResultName	String	
Pascal	IDSesion	String	-
	Dimension	Integer	
	ResultName	String	

PInv	IDSesion	String	-
	Matriz1	String	
	ResultName	String	
ProdSum	IDSesion	String	-
	Matriz1	String	
	Dimension	Integer	
	ResultName	String	
Qr	IDSesion	String	-
	Matriz1	String	
	ResultName	String	
RandMultiDim	IDSesion	String	-
	Dims	TRemotableMultiDim	
	ResultName	String	
RandNMultidim	IDSesion	String	-
	Dims	TRemotableMultiDim	
	ResultName	String	
RandXY	IDSesion	String	-
	X	Integer	
	Y	Integer	
	ResultName	String	
RandXYZ	IDSesion	String	-
	X	Integer	
	Y	Integer	
	Z	Integer	
	ResultName	String	
RandnXY	IDSesion	String	-
	X	Integer	
	Y	Integer	
	ResultName	String	
RandNXYZ	IDSesion	String	-
	X	Integer	
	Y	Integer	
	Z	Integer	
	ResultName	String	
Rank	IDSesion	String	-
	Matriz1	String	
	ResultName	String	
Rcond	IDSesion	String	-

	Matriz1	String	
	ResultName	String	
Rosser	IDSesion	String	-
	ResultName	String	
Rot90	IDSesion	String	-
	Matriz1	String	
	ResultName	String	
Schur	IDSesion	String	-
	Matriz1	String	
	ResultName	String	
Size	IDSesion	String	-
	Matriz1	String	
	ResultName	String	
Sort	IDSesion	String	-
	Matriz1	String	
	ResultName	String	
SqrtM	IDSesion	String	-
	Matriz1	String	
	ResultName	String	
Substract	IDSesion	String	-
	Matriz1	String	
	Matriz2	String	
	ResultName	String	
SVD	IDSesion	String	-
	Matriz1	String	
	ResultName	String	
Transpose	IDSesion	String	-
	Matriz1	String	
	ResultName	String	
TransposeNonConjugate	IDSesion	String	-
	Matriz1	String	
	ResultName	String	
TriL	IDSesion	String	-
	Matriz1	String	
	ResultName	String	
TriU	IDSesion	String	-
	Matriz1	String	
	ResultName	String	

VertCat	IDSesion	String	-
	Matriz1	String	
	Matriz2	String	
	ResultName	String	
Wilkinson	IDSesion	String	-
	Dimension	Integer	
	ResultName	String	
ZerosMultiDim	IDSesion	String	-
	Dims	TRemotableMultiDim	
	ResultName	String	
ZerosXYZ	IDSesion	String	-
	X	Integer	
	Y	Integer	
	Z	Integer	
	ResultName	String	
ZerosXYMatrix	IDSesion	String	-
	X	Integer	
	Y	Integer	
	ResultName	String	