# Extracción de Reglas de Negocio de Código Fuente

Estudiantes:
Martín Ale
Pablo Claps
Pablo Pirán

Tutora:
Profesora Ing. Cristina Mayr

Informe de Proyecto de Grado presentado al Tribunal Evaluador como requisito de graduación de la carrera Ingeniería en Computación.

**Montevideo, República Oriental de Uruguay, año 2011.**

# RESUMEN DEL TRABAJO

El BROU (Banco de la República Oriental del Uruguay) cuenta con un sistema informático central desarrollado en EAE (Enterprise Application Environment)/LINC (Logic and Information Network Compiler), una herramienta de cuarta generación propietaria de Unisys Corporation, que cuenta con un ambiente de desarrollo en Windows utilizando un lenguaje denominado LDL y genera código fuente Cobol para distintas plataformas (en el caso particular del banco equipos Sun Sparc con Sistema Operativo Solaris 10, Base de Datos Oracle 10g y Micro Focus Cobol 4.0 Server Express Service Pack 2).

Este sistema denominado SFB (Sistema Financiero Bancario) ha sido utilizado por el banco durante quince años, en los que ha evolucionado junto con los requerimientos del negocio, pero con cambios que no han sido formalmente registrados. Es debido a esto que en la actualidad el banco no posee documentación actualizada del sistema con las reglas de negocio que se aplican en el mismo.

Nuestro trabajo consistió en la selección de un conjunto compuesto por las transacciones mas importantes del sistema, y el desarrollo de un programa que realiza un proceso de ingeniería inversa y a partir del código fuente del sistema devuelve como salida un pseudocódigo con las reglas de negocio aplicadas.
Para la selección de las transacciones nos basamos en la experiencia de los principales analistas de los diferentes módulos del sistema, junto con estadísticas de ejecución en el ambiente de producción.
Durante el proyecto consideramos una mejor solución realizar un desarrollo aplicable a cualquier desarrollo en EAD, y no restringido a un conjunto de transacciones en particular.

Para la gestión de proyecto estudiamos la metodología PMI y la adaptamos a nuestro proyecto, siguiendo las mejores prácticas sugeridas en el PMBOK.

En el análisis utilizamos teoría de compiladores [15], dado que se utilizaron técnicas de diseño de compiladores para el desarrollo de nuestro traductor. También estudiamos herramientas existentes con funcionalidades similares, y el proceso de “Synchronized Refinement”, ya que en el se define parte del fundamento teórico que utilizamos en nuestro trabajo.

En cuanto a la implementación, optamos por un desarrollo JEE en NetBeans con base de datos MySQL, debido a que son herramientas de software libre y las consideramos adecuadas para nuestra implementación.
Nuestro programa carga en una base de datos la información extraída del sistema a analizar, para luego hacer una doble recorrida del código fuente armando un árbol sintáctico-semántico, e invocando métodos que traducen los comandos a expresiones previamente definidas para formar el pseudocódigo que será la salida final.

Además de incluir el material y la documentación generado durante el proyecto, en los anexos se incluye un relevamiento del módulo de Banca Electrónica (transacciones de cajeros automáticos) que incluye el análisis de su rediseño hacia una arquitectura orientada a servicios.

## Palabras Claves y Temáticas Relevantes

Ingeniería Inversa, Reglas de Negocio, Código Fuente, BROU, Unisys, EAE/LINC, Sistema Financiero Bancario, Synchronized Refinement, Pseudocódigo, Teoría de Compiladores, Árbol Semántico-Sintáctico, SOA.

# TABLA DE CONTENIDO

# CAPÍTULO 1: Introducción

## 1.1 - Planteo del problema

El BROU (Banco de la República Oriental del Uruguay) inició en el año 1995 un proyecto de reestructura que incluyó el cambio de sus sistemas centrales. En esa oportunidad adquirió el sistema SFB (Sistema Financiero Bancario) a la empresa Unisys Corporation, desarrollado en ese momento en la herramienta de cuarta generación LINC (Logic and Information Network Compiler). El sistema fue modificado durante los últimos 15 años, en una primera instancia para ser personalizado según los requerimientos del banco, y posteriormente para adaptarse a los constantes cambios que se han dado en el negocio bancario. También surgieron nuevas versiones para la herramienta LINC. Primero fue LDA (LINC Development Assistant) y luego EAE (Enterprise Application Environment), cada una agregando nuevas funcionalidades e independizando el desarrollo y posterior generación del sistema de la plataforma seleccionada para la ejecución del sistema. El BROU actualmente cuenta con equipos Sun con procesadores Sparc, Sistema Operativo Solaris 10 y Base de Datos Oracle 10g. La versión de Cobol es Micro Focus Cobol 4.0 Service Pack 2.
Los sucesivos cambios no fueron registrados de forma apropiada, debido a lo cual el banco no cuenta actualmente con una documentación técnica actualizada que refleje las reglas de negocio que se aplican en el sistema.
La siguiente versión de LINC/EAE de la empresa Unisys es la herramienta ABSuite (Agile Business Suite), que cuenta con un ambiente de desarrollo en Visual Studio, pero el código generado únicamente podrá ser .NET sobre SQL-Server para Windows Servers o código Cobol para la plataforma propietaria Unisys (servidores ClearPath con sistema operativo MCP y base de datos DMSII).
El banco pretende mantenerse en una plataforma Unix con base de datos Oracle, por lo que ha encarado un nuevo proyecto de cambio de su sistema central.
Para tener éxito en este nuevo emprendimiento necesita evaluar correctamente el sistema SFB y contar con información acerca de las reglas de negocio que son aplicadas en el sistema.

## 1.2 - Objetivos planteados

En base a nuestra experiencia en la herramienta EAE y el sistema SFB, el objetivo fue desarrollar un programa que a partir de código fuente escrito en LDL (LINC Definition Language) extraiga las reglas de negocio aplicadas, descriptas en un pseudocódigo que fuera accesible para un usuario que no haya trabajado en EAE pero que cuente con conocimientos en programación y sistemas bancarios. Este proyecto es considerado muy importante para el banco, ya que cualquiera sea el nuevo sistema que adquiera, este trabajo podrá servir de gran ayuda durante el proceso de personalización del mismo.
Nos planteamos como objetivo que nuestro desarrollo cubriera las transacciones más importantes del banco. La selección de transacciones se basó en la opinión de especialistas técnicos en los diferentes módulos del sistema y en estadísticas del uso del sistema en producción.

## 1.3 - Conclusiones

La extracción de reglas de negocio es un proceso complejo, que difícilmente llegue a poderse automatizar totalmente, ya que nunca podrá sustituir el análisis de un técnico experto. Además siempre se depende de la calidad del código escrito, que no siempre es la mejor. Un código que siga estándares, sea prolijo y eficiente será más fácil de interpretar que uno que no cumpla estas condiciones.
Durante el desarrollo del proyecto entendimos que no era apropiado ceñirnos a un conjunto de transacciones, sino que nuestro desarrollo debería ser genérico y cumplir un nivel de abstracción que permitiera su aplicación a cualquier código fuente EAE.
La meta que nos planteamos fue ambiciosa, y consideramos que obtuvimos un resultado correcto, pero enmarcado en un trabajo que puede seguir avanzando dentro de un proceso de mejora continua, ya que se puede ampliar el espectro de patrones de reconocimiento de bloques de código, además de agregar nuevas funcionalidades a la solución.

## 1.4 - Organización general del documento

El capítulo dos tiene un breve resumen del estado del arte del trabajo al momento de comenzar nuestro trabajo.
El capítulo tres describe la parte central de nuestro trabajo, incluyendo los procesos seguidos para la gestión del proyecto.
El capítulo cuatro documenta la implementación realizada.
Las pruebas realizadas y sus resultados se detallan en el capítulo cinco.
Este documento se completa con un glosario y las referencias bibliográficas.
En el documento de anexos se adjuntan los documentos generados durante el proyecto, detalle de desarrollos auxiliares, información estadística del sistema SFB en producción y un análisis del módulo de Banca Electrónica pensando en un diseño orientado a servicios.

# CAPÍTULO 2: Revisión del Estado del Arte del Trabajo

## 2.1 - Conceptos

### 2.1.1 - Ingeniería inversa

El objetivo de la ingeniería inversa es obtener información  a partir de un producto accesible al público, con el fin de determinar de qué está hecho, qué lo hace funcionar y cómo fue fabricado.
Hoy en día los productos más comúnmente sometidos a ingeniería inversa son los programas de computadoras y los componentes electrónicos, pero, en realidad, cualquier producto puede ser objeto de un análisis de Ingeniería Inversa.
El método se denomina así porque avanza en dirección opuesta a las tareas habituales de ingeniería, que consisten en utilizar datos técnicos para elaborar un producto determinado. En general, si el producto u otro material que fue sometido a la ingeniería inversa fue obtenido en forma apropiada, entonces el proceso es legítimo y legal. De la misma forma, pueden fabricarse y distribuirse, legalmente, los productos genéricos creados a partir de la información obtenida de la ingeniería inversa, como es el caso de algunos proyectos de Software libre ampliamente conocidos. [1]

### 2.1.2 - Reglas de negocio

Las reglas del negocio o conjunto de reglas de negocio describe las políticas, normas, operaciones, definiciones y restricciones presentes en una organización y que son de vital importancia para alcanzar los objetivos misionales.
Ejemplos de reglas de negocio: "Un cliente al que facturamos más de 10.000 al año es un cliente de tipo A", "A los clientes de tipo A les aplicamos un descuento del 10% en pedidos superiores a 3.000".
Las organizaciones funcionan siguiendo múltiples reglas de negocio, explícitas o tácitas, que están embebidas en procesos, aplicaciones informáticas, documentos, etc. Pueden residir en la cabeza de algunas personas o en el código fuente de programas informáticos. [2]

## 2.2 - Herramientas investigadas

Existen herramientas que permiten la conversión automática de un lenguaje de programación a otro, diseñadas en el marco de la modernización de aplicaciones, fundamentalmente de sistemas legados.
Un ejemplo es el de la compañía softwaremining que cuenta con un producto que traduce código Cobol en programas Java o C#. [3]
Más concretamente referido a nuestro proyecto, la compañía Art in Soft ofrece una traducción automática de código EAE/LINC a código Java. [4]
De todas formas, en todos los casos no se logra un 100% de efectividad y siempre es necesaria la intervención manual en el nuevo código generado. Además, dadas las diferencias entre los lenguajes, parte del código generado no es amigable y el posterior mantenimiento puede resultar costoso.

En el caso específico del lenguaje EAE/LINC la empresa BTG (Baltic Technology Group) de la ciudad de Riga, capital de Letonia, ofrece herramientas que procesan los archivos mdl para extraer información estadística de los sistemas EAE (cantidad de transacciones, tablas, reportes, etc.). [5]

Por último, Unisys desarrolló la nueva versión de EAE, llamada ABSuite, que traduce el código EAE a código LDL+ (nueva versión del lenguaje) que es editado en Visual Studio y genera código .NET para Windows con SQL-Server o programas Cobol en caso de utilizar hardware y software propietario de Unisys (equipos Clearpath, sistema operativo MCP y base de datos DBMS2). [6]

No encontramos herramientas que extraigan reglas de negocio de código fuente EAE/LINC.

## 2.3 - Antecedentes en la extracción de reglas de negocio

La extracción de reglas de negocio de código fuente es un concepto que se viene manejando desde la década del 90.
El proceso es conocido como “Synchronized Refinement”, que toma como entrada el código fuente, conocimiento del lenguaje de programación y de conceptos básicos del negocio, y tiene como salida las reglas de negocio que se aplican.
Un trabajo muy interesante que utilizamos fue escrito por Spencer Rugaber del Georgia Tech College of Computing. En el mismo describe el procedimiento genérico a utilizar, mas allá del lenguaje de programación original y la salida esperada. [7]
El nuestro es un caso particular en el que la entrada es código EAE y la salida es un pseudocódigo en español comprensible para usuarios con conocimientos informáticos y del negocio bancario de nuestro país.

Otro punto a considerar es la existencia de un estándar SBVR (Semántica de Vocabulario y Reglas de Negocio, por sus siglas en inglés), desarrollado por la organización OMG (Object Management Group). El documento define formalmente la representación de reglas de negocio, para poder ser intercambiada entre diferentes organizaciones o herramientas de software. [8]
Esto escapa a nuestro objetivo, ya que nuestra intención es dar una salida amigable para usuarios finales, y no generar archivos que puedan ser interpretados por otros programas.

## 2.4 - Leyendo y comprendiendo código fuente

### 2.4.1 - Introducción

¿Por qué es difícil entender un programa? Algunas razones son:

- Flujo de control complicado
- Falta de estructuras de control
- Múltiples propósitos
- Reutilización de variables
- Algoritmos complejos
- Técnicas de programación

¿Por qué leer programas?

Para lograr una comprensión general que permita darles soporte a largo plazo. También para realizar tareas más específicas, como ser probar, documentar, modificar, o extraer para reutilizar.

Algunos números:

- 50%-70% del tiempo de desarrollo es dedicado a mantener o mejorar programas ya existentes.
- 50% del tiempo de mantenimiento de un programa se utiliza para comprender el programa.

Por lo tanto leer un programa es esencial para que el desarrollo sea productivo.

Preguntas:

¿Qué hace este programa? Esta respuesta requiere conocimientos de programación.
¿Por qué hace esto el programa? Mientras que esta requiere conocimiento del negocio que es dominio de la aplicación.

### 2.4.2 - Synchronized Refinement (SR)

Synchronized Refinement es una técnica de lectura de código que simultáneamente examina e interpreta código fuente mientras genera una descripción del programa. Debe interpretar cómo el código fuente realiza tareas específicas del dominio en el que se aplica.

### 2.4.3 - Resumen del proceso SR

Entrada:

- Código fuente.
- Conocimiento del lenguaje de programación.
- Conocimiento de reglas de negocio del dominio de la aplicación.

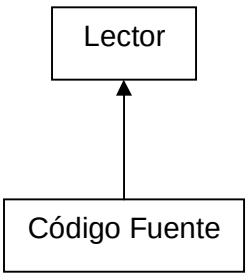
Salida:

- Una lista comentada de pasos y decisiones ejecutadas en función de conceptos específicos del dominio de la aplicación.

Pasos:

- El código puede ser leído sistemáticamente, de principio a fin, en una o más pasadas, buscando respuestas relevantes a preguntas específicas.
- SR relaciona la abstracción de código con conceptos del dominio de la aplicación.

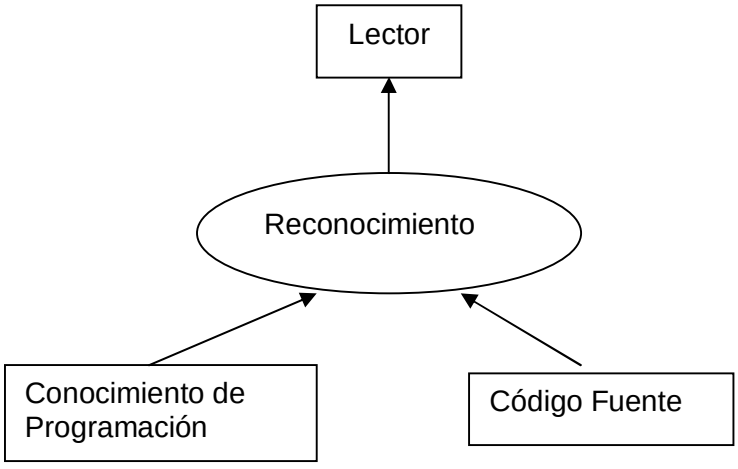
Diagrama de Flujo de información




Para responder la pregunta ¿qué hace este programa? es necesario:

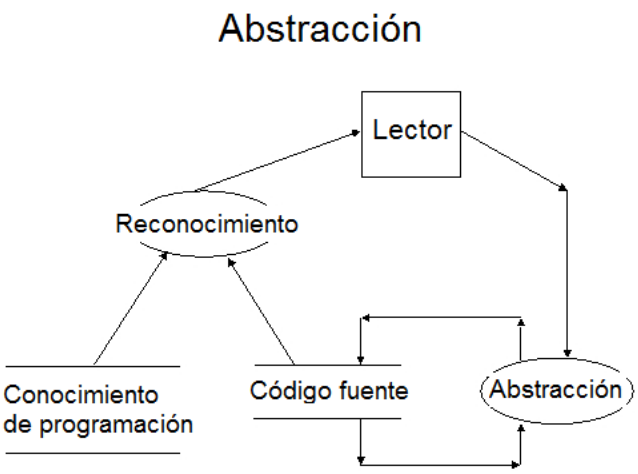
- Reconocer construcciones de código.
- Abstraer el código a una forma resumida.

Para el reconocimiento es necesario tener diferentes conocimientos, como ser: algoritmos, estructuras de datos, arquitectura del hardware, librerías auxiliares, idiomas, semántica y sintaxis del lenguaje de programación.



### 2.4.4 - Abstracción

El reconocimiento permite abstracción, es el proceso de dejar de lado detalles y concentrarse en los conceptos importantes. La abstracción elimina los detalles superfluos, sustituyendo los segmentos removidos por breves indicaciones.

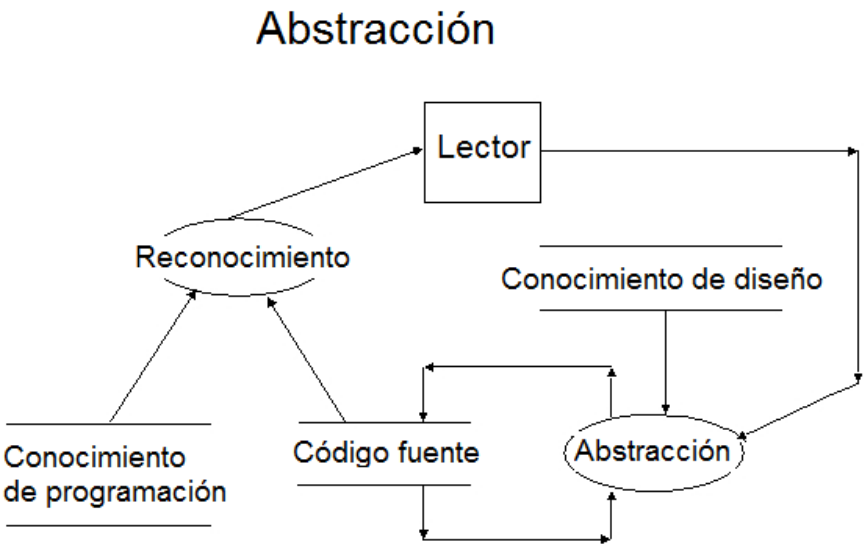



### 2.4.5 - Decisiones de diseño

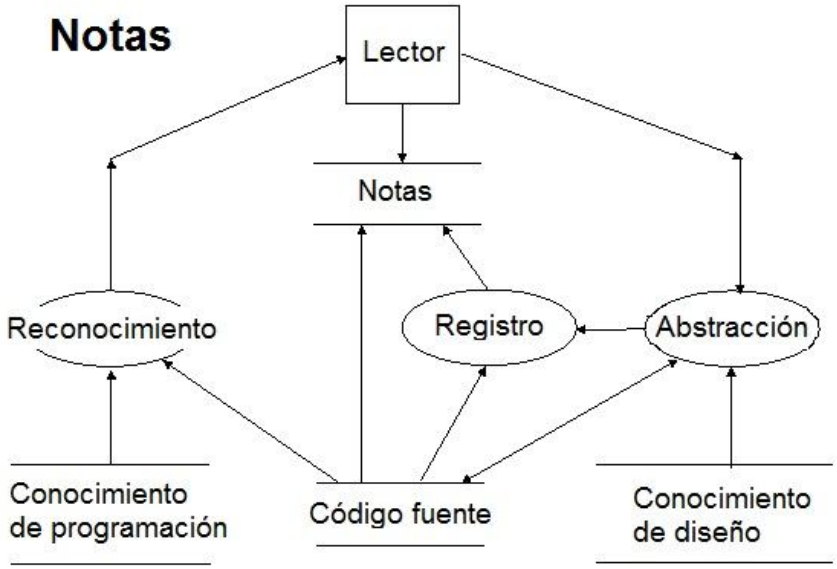
En SR, la abstracción es ejecutada en unidades de decisiones de diseño. Esto es, el lector reconoce que el diseñador/programador decidió expresar una idea usando una construcción de programación en particular.

Las decisiones de diseño se dividen en varias categorías:

- Descomposición (descomponer un código en segmentos más simples).
- Inicialización (salvar los resultados de una operación para su uso posterior, variables en temporales).
- Especialización/Generalización: tratar un problema general como un conjunto de casos particulares.
- Representación: seleccionar una estructura de datos o algoritmo para representar otro.
- Encapsular: dos decisiones de diseño en una sección de código.




El proceso de reconocimiento/abstracción sistemáticamente guía al lector a través del programa, detectando decisiones de diseño y dando una visión global del programa. La detección de cada decisión de diseño encontrada deberá ser registrada como notas o anotaciones para ayudar en la comprensión global del programa. De esta forma se irá guardando el entendimiento logrado.




Las preguntas de ¿Por qué?

El lector quiere saber por qué un segmento de código es incluido en el programa, y la respuesta es en términos de los requerimientos o reglas de negocio que busca cumplir el programa en ese segmento de código. Para eso es necesario contar con conocimiento acerca del dominio de la aplicación.

### 2.4.6 - Conclusiones

Reconocer decisiones de diseño permite identificar *qué* es lo que hace un programa. El conocimiento del dominio de la aplicación permite identificar *por qué* hace algo un programa. El conocimiento recolectado es expresado y registrado en anotaciones (o notas). Estos dos procedimientos de reconocimiento se ejecutan en conjunto en SR para el proceso de lectura del programa.

# CAPÍTULO 3: Parte Central del Trabajo

## 3.1 – Introducción

En primer lugar definimos los procesos a seguir para la gestión del proyecto.
Para ellos nos basamos en la metodología PMI. [16]
Luego de definido el alcance del proyecto analizamos el problema planteado y construimos un modelo de dominio. A partir de dicho modelo se definió el diagrama de clases y se crearon las mismas en el proyecto.
Al usar JPA para la persistencia las estructuras en la base de datos fueron generadas en forma automática. El MER se extrajo utilizando una herramienta de MySQL.
En el siguiente capítulo se detallan la arquitectura, el diseño y la implementación.
Las pruebas se documentan en el capítulo 5.

## 3.2 – Gestión del proyecto

Adaptamos lo que plantea la metodología PMI a nuestro proyecto en particular.

### 3.2.1 – Dirección del proyecto

En el PMBOK define dentro de la dirección del proyecto los siguientes puntos.

#### 3.2.1.1 – Gestión de la integración del proyecto

Se definió realizar el Proyecto de Grado presentado en conjunto por el BROU y UNISYS, según consta en el Acta de Constitución del Proyecto, que fue formalmente aprobada por las distintas partes.
Se definió un alcance preliminar al inicio del proyecto.

#### 3.2.1.2 – Gestión del alcance del proyecto

Se planificó el alcance y se creó la estructura de desglose de trabajo (WBS). Posteriormente se documentó formalmente el alcance del proyecto. La validación del alcance se realizó mediante la aceptación de dicho documento por parte de los usuarios responsables del BROU y Unisys.
Durante el desarrollo del proyecto se realizaron modificaciones al alcance de común acuerdo con los usuarios responsables.

##### 3.2.1.2.1 – WBS (Work Breakdown Structure)

El diagrama WBS nos permitió organizar las diferentes tareas a realizarse durante el proyecto. Nos ayudó a visualizar de mejor forma las distintas actividades involucradas en el proyecto.
Se utilizó para descomponer el proyecto en elementos de complejidad decreciente hasta llegar a un nivel en el cual se pueda comprender con mayor claridad el trabajo a realizar y el esfuerzo requerido para el mismo. A este último nivel de descomposición se le denomina paquete de trabajo.
Además se creó un diccionario para cada paquete de trabajo que contiene la información más importante en una especificación mínima para cada paquete de trabajo. [17]

#### 3.2.1.3 – Gestión del tiempo del proyecto

En base al WBS se identificaron las actividades necesarias para producir los entregables del proyecto. Las actividades tienen una secuencia en el tiempo y precedencias entre sí.
Para cada actividad se estimaron los tipos y cantidades de recursos necesarios para la llevarla a cabo. También se estimó la duración de cada actividad en base a los recursos asignados.

Con esta información se generó un documento Project con las estimaciones, utilizando la herramienta Microsoft Project, que contempla las distintas actividades, junto con sus requerimientos de tiempos y recursos.
Estas estimaciones se fueron ajustando en el transcurso del proyecto, y periódicamente se realizó un seguimiento de los cambios realizados en las estimaciones.

### 3.2.1.4 – Gestión de los costos del proyecto

No aplica, debido a que no se manejan costos en el proyecto, por lo tanto no es necesario estimar costos de los recursos ni preparar presupuestos de costos asociados al proyecto.

### 3.2.1.5 – Gestión de la calidad del proyecto

Para la gestión de la calidad del proyecto se realizó un Plan de Calidad, en el cual se identificaron los requerimientos de la calidad.
Además se realizó un control de la calidad verificando que se satisfagan los requerimientos detectados.
Como parte de la gestión de la calidad y para el manejo de las configuraciones (SCM) utilizamos la herramienta Tortoise SVN, que nos permitió el acceso a un ambiente controlado Subversión (Controlador de Versiones) disponible en un servidor del INCO. [18]

### 3.2.1.6 – Gestión de los recursos humanos del proyecto

Los recursos humanos fuimos los tres estudiantes del proyecto. Las decisiones del proyecto fueron tomadas en consenso, y consultando a la docente tutor del proyecto. En las distintas fases del proyecto cada uno debió asumir distintos roles en base a las actividades que íbamos desarrollando.

### 3.2.1.7 – Gestión de las comunicaciones del proyecto

Para la gestión de las comunicaciones se elaboró un Plan de Comunicaciones.
En el plan se especifican las diferentes vías de comunicación establecidas para cada actividad del proyecto.

### 3.2.1.8 – Gestión de los riesgos del proyecto

Para la gestión de los riesgos del proyecto se elaboró un Plan de Riesgos.
En el documento se muestran los riesgos detectados y el plan de contingencia a utilizar.
Durante el proyecto se activaron varios riesgos los cuales se pudieron manejar de manera adecuada. Mensualmente se realizó una verificación de los riesgos para detectarlos tempranamente. El riesgo de no poder completar el alcance en el mes de diciembre fue detectado lo cual nos permitió solicitar con anticipación una prórroga.

### 3.2.1.9 – Gestión de las adquisiciones del proyecto

No aplica para nuestro proyecto ya que no tuvimos que hacer adquisiciones.

## 3.2.2 – Entregables del proyecto

Se definieron los siguientes entregables como parte de la gestión del proyecto.

- Project
- Diagrama WBS con Diccionario asociado
- Estándar de Documentación
- Plan de Comunicaciones
- Plan de Calidad
- Plan de Riesgos

## 3.3 – Análisis

### 3.3.1 – Casos de uso

El interés del usuario es obtener una traducción de un programa (bloques de código, en este caso ispecs o lógicas globales) en un pseudocódigo predefinido en lenguaje natural. Esta salida se presentará generando archivos html con la traducción obtenida.

Se identificaron tres casos de uso para satisfacer dicho requerimiento.

- Traducir lógica: Dado el nombre de una lógica o ispec genera archivos html con una traducción de la misma en lenguaje natural.
- Consultar ruta salida: Se devuelve la ruta en el sistema en donde se generan los archivos con las traducciones de la lógica.
- Asignar ruta salida: Asigna la ruta en el sistema en donde se generan los archivos con las traducciones de la lógica.

**Caso de uso: Traducir lógica**

**Flujo principal**

1) El usuario ingresa nombre de lógica a traducir.
2) El sistema devuelve descripción de la lógica, seguida de la traducción obtenida, y genera una serie de archivos html con dicha traducción.

**Flujos alternativos**

2A) Nombre de lógica no existe en el sistema.
El sistema devuelve mensaje de error “No existe la lógica”.
2B) No se ingresó nombre de lógica.
El sistema devuelve mensaje de error “Debe ingresar nombre de lógica”.

**Caso de uso: Consultar ruta salida**

**Flujo principal**

1) El sistema devuelve la ruta completa en el sistema en donde se generarán los archivos.

**Caso de uso: Asignar ruta salida**

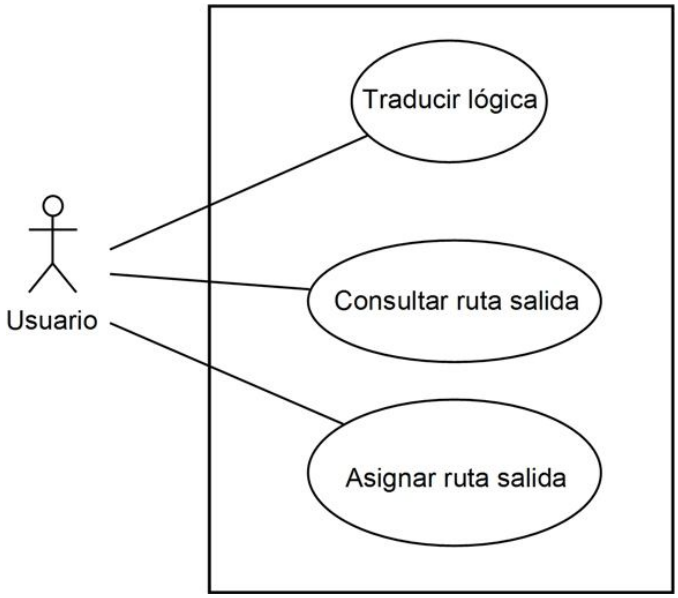
**Flujo principal**

1) El usuario ingresa la ruta completa en el sistema en donde se copiarán los archivos de salida (programas traducidos).

**Flujos alternativos**

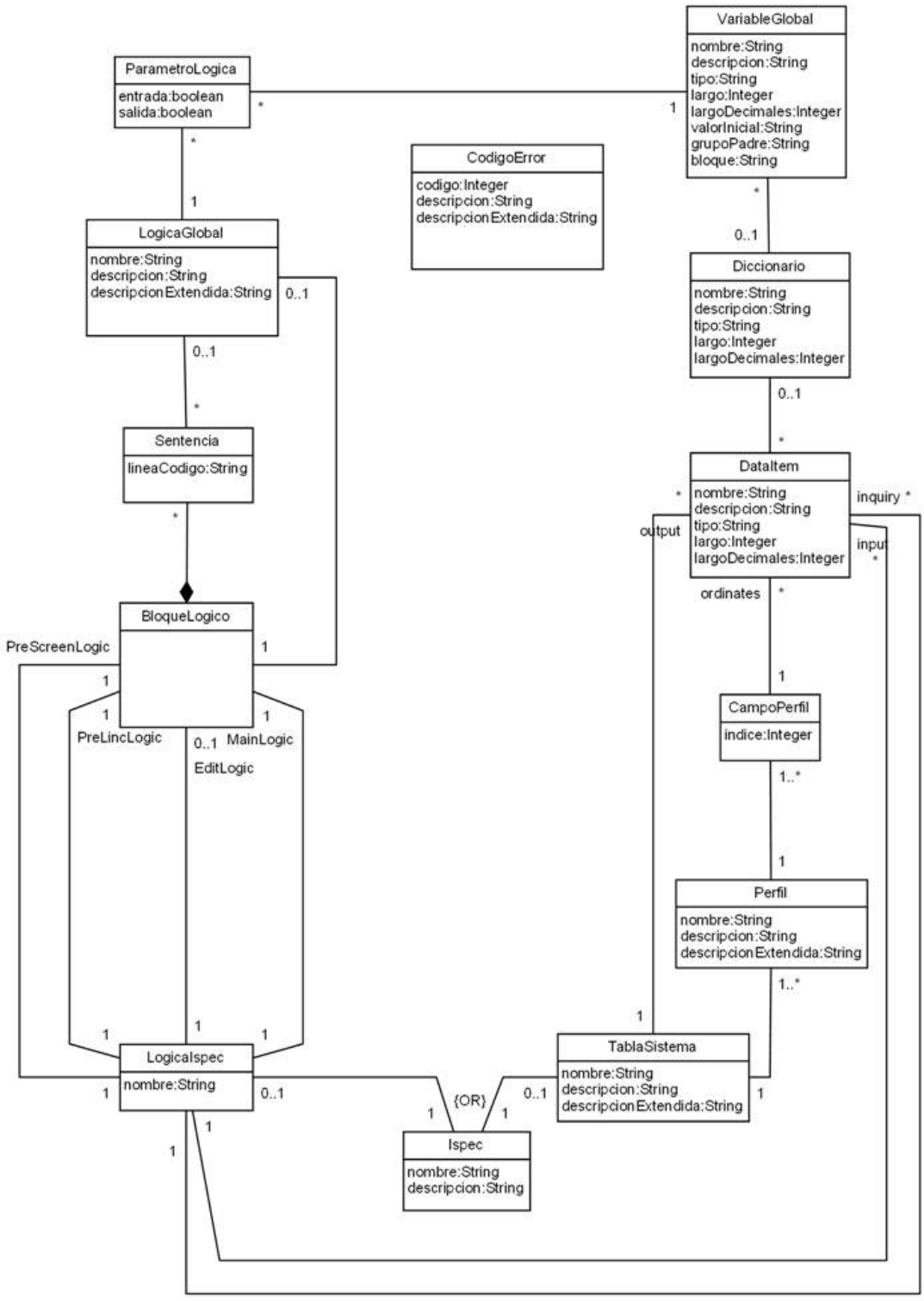
1A) Ruta inválida.
El sistema devuelve mensaje de error “Ruta inválida”.

**Diagrama de casos de uso**



## 3.3.2 – Modelo de Dominio

En base a la realidad del sistema, se creó un modelo que muestra las relaciones entre los componentes de EAE.



### 3.3.2.1 – Descripción de los Elementos del Modelo de Dominio

**Ispec**

Un Ispec es un objeto en EAE que representa un elemento de la realidad, el cual puede definir una Tabla y/ó una Transacción del Sistema.

**LogicaIspec**

La Lógica de un Ispec (corresponde con una transacción en el sistema) tiene asociados varios Bloques Lógicos (“Main Logic”, “PreLincLogic”,”PreScreenLogic”) y opcionalmente (“EditLogic”).
Cada Lógica de un Ispec tiene asociados una lista de DataItems de Input y/ó de DataItems de Inquiry.

**BloqueLogico**

Un Bloque Lógico define un conjunto de sentencias y cada sentencia corresponde a una línea de código.

**LogicaGlobal**

Las Lógicas globales pueden tener asociados varios parámetros.
El pasaje de parámetros a las lógicas globales se realiza a través de Variables globales por eso la asociación entre las entidades LogicaGlobal con ParametroLogica y ParametroLogica con VariableGlobal.

**VariableGlobal**

La definición de la variable global puede obtenerse a partir del Diccionario de Datos, si esto ocurre existe la asociación entre las entidades VariableGlobal y Diccionario.

**TablaSistema**

Cada TablaSistema tiene asociado un Perfil y una lista de DataItems.

**Perfil**

Cada Perfil tiene asociado una lista de CampoPerfil.

**CampoPerfil**

Cada CampoPerfil tiene asociada una lista de DataItems.

**DataItem**

Cada DataItem puede estar o no en el Diccionario.

**CódigoError**

Almacena los códigos de error del sistema.

**Diccionario**

Contiene la definición de DataItem´s y VariablesGlobales.

**Sentencia**

Cada línea de Código es representada como una Sentencia.

**ParámetroLógica**

Tipo de parámetro (entrada o salida) de las lógicas globales.

### 3.3.2.2 – Restricciones al Modelo de Dominio

**Restricción circular**

Tabla Sistema ->Perfil->Campo Perfil ->Lista DataItem
Tabla Sistema ->Lista DataItem (output)

Dada una Tabla de Sistema que tiene una lista de DataItems asociados, todos esos DataItems deben pertenecer a un Campo de un Perfil que pertenezca a un Perfil de la Tabla Sistema.

**Ispec**

El identificador (nombre) de un Ispec es único.

Si un Ispec tiene una relación con una TablaSistema entonces el identificador (nombre) de la TablaSistema debe corresponder con el identificador (nombre) del Ispec.

Si un Ispec tiene una relación con una LógicaIspec entonces el identificador (nombre) del LógicaIspec debe corresponder con el identificador (nombre) del Ispec.

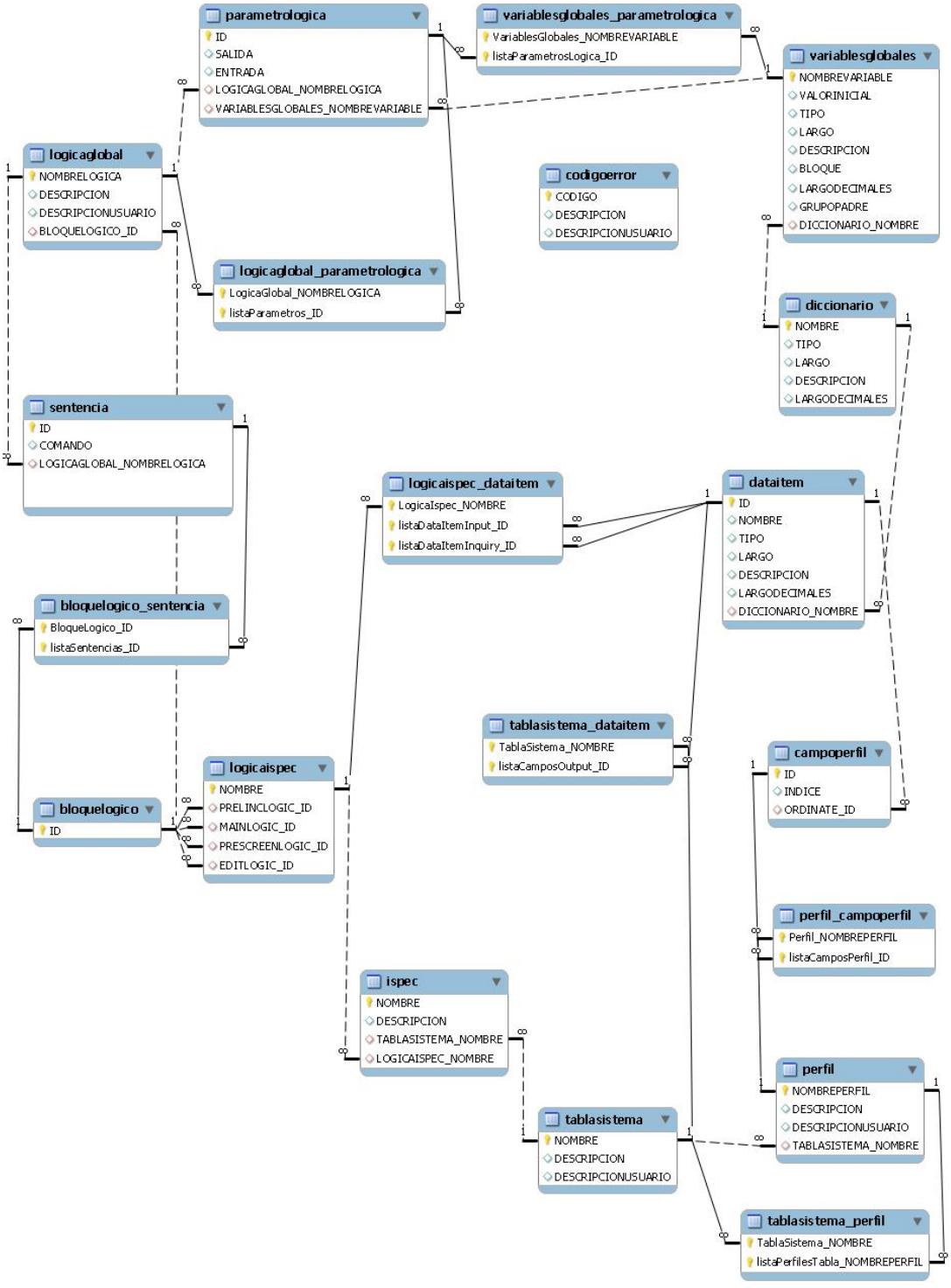
**Diccionario**

El identificador (nombre) de un Diccionario es único.

**DataItem**

Si un DataItem tiene una relación con un Diccionario, entonces todos los atributos del DataItem son idénticos a los del Diccionario.

## 3.3.3 - Modelo Entidad-Relación (MER)




### 3.3.3.1 - Notas sobre el Modelo de Datos

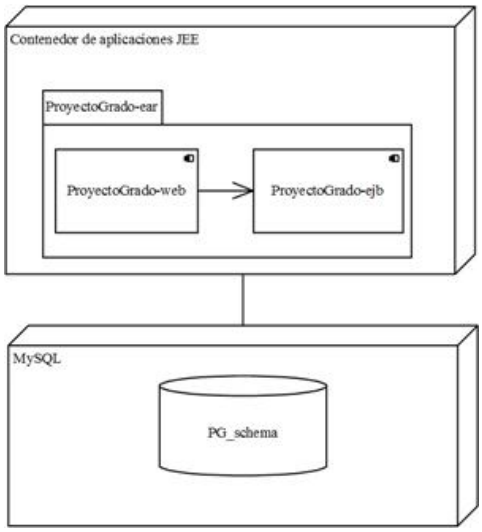
- Las líneas punteadas representan las Foreing keys.
- Las líneas completas representan asociaciones
- El modelo de datos es generado por JPA (Java Persistence Unit)

# CAPÍTULO 4: Diseño e Implementación

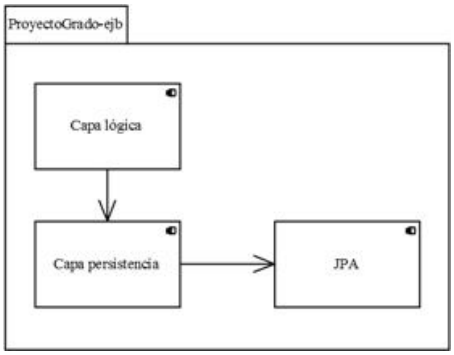
## 4.1 - Arquitectura

### 4.1.1 - Componentes

Se decidió utilizar una arquitectura en tres capas bien definidas. Para ello se creó un proyecto de empresa JEE, el cual se compone de un proyecto web y un proyecto ejb (enterprise java beans). [10]
Este desarrollo se realizó en NetBeans [12], con un servidor de aplicaciones GlassFish [13] y una base de datos MySQL [14].
El proyecto ejb contiene la lógica de negocio y resuelve la persistencia (capas lógica y de persistencia), mientras que el proyecto web solo contiene la interfaz gráfica, separando de esta forma la capa lógica y persistencia de la capa de presentación en proyectos diferentes. Esto independiza la elección de la tecnología para la implementación de la IG.
La capa de persistencia utiliza la Java Persistence API (JPA). [11]



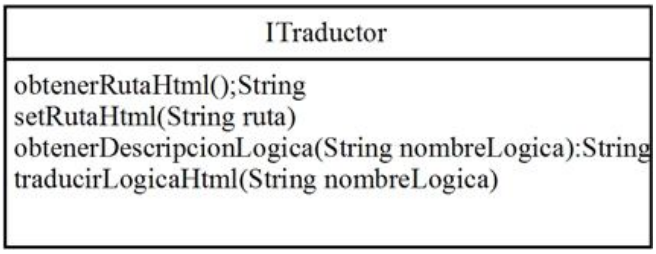

El proyecto ejb contiene una capa lógica y una capa de persistencia, mientras que el proyecto web contiene la capa de presentación.




Luego veremos las interfaces que son provistas tanto para la lógica como para la persistencia.

## 4.2 – Diseño de interfaz

Para satisfacer los requerimientos identificados se diseñó la siguiente interfaz




con las siguientes operaciones:

***obtenerRutaHtml():String***

Esta operación consulta la ruta local sobre la cual se grabarán los archivos html, los cuales contienen las traducciones.

***setRutaHtml(String ruta)***

Esta operación permite cambiar dicha ruta (path local en el pc).

***obtenerDescripcionLogica(String nombreLogica):String***

Cada lógica tiene una descripción asociada, la cual da una idea de lo que hace la lógica.

***traducirLogicaHtml(String nombreLogica)***

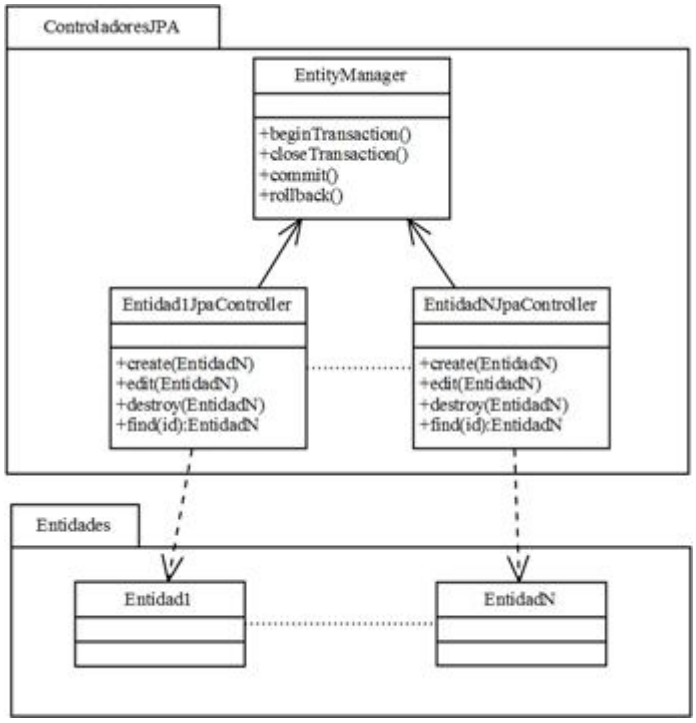
Dado el nombre de una lógica, se realiza la traducción y se graba la salida en un archivo html en la ruta especificada por la operación setLogicaHtml(String ruta).

Las operaciones *obtenerDescripcionLogica()* y *traducirLogicaHtml()* devuelven error si la lógica no existe o no está previamente cargada en el sistema.

## 4.3 - Capa de persistencia

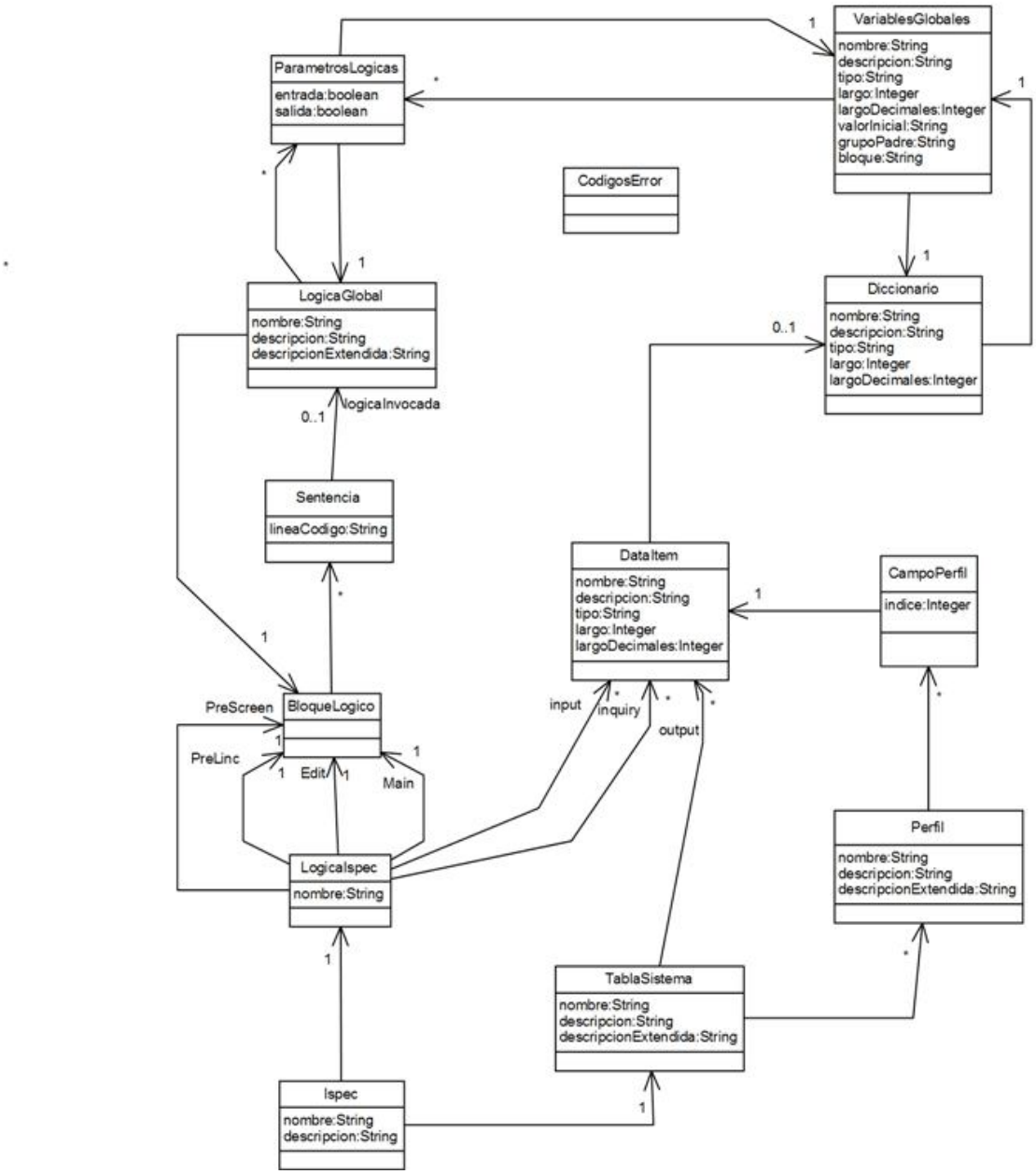
La capa de persistencia está separada de la capa lógica y de la de presentación respetando una arquitectura en tres capas. Para la persistencia se utilizó JPA, por lo que lo único necesario para persistir una clase es agregar anotaciones en la misma y crear un controlador específico que implemente las operaciones básicas (CRUD, o ABMC; alta, baja, modificación y consulta).

### 4.3.1 - Diagrama de clases




Las clases que se decidió persistir son las que modelan el sistema EAE, no las utilizadas para la traducción. Las mismas guardan información del sistema financiero bancario, como ser tablas, programas, módulos, diccionarios, etc.

Las clases que se persisten se muestran en el siguiente diagrama de clases.



### 4.3.2 - Proceso de carga

Se realizó un estudio del entorno EAE y de las entidades que componen el sistema. Como resultado se obtuvo un Modelo de Dominio (ver capítulo 3), y posteriormente se definieron las clases y sus relaciones (diagrama anterior). Para persistir estas clases se utilizó la Java Persistence API (JPA), la cual mapea cada clase con una tabla en la base de datos relacional, y genera otras tablas para las relaciones.

EAE genera archivos de especificación de todos los componentes del sistema. Estos archivos (de extensión mdl) nos detallan cada objeto definido en EAE, sus atributos y relaciones. Se utilizaron estos archivos para cargar la información del modelo, y se implementaron cargadores específicos para cada componente.

Los archivos de especificación generados son los siguientes:

**Diccionario.mdl**

Corresponde a los datos del Diccionario del Sistema EAE.

**LogicasGlobales.mdl**

Lógicas globales del Sistema EAE.

**VariablesGlobales.mdl**

Variables globales del Sistema EAE.

**Ispecs.mdl**

Ispecs del Sistema EAE.

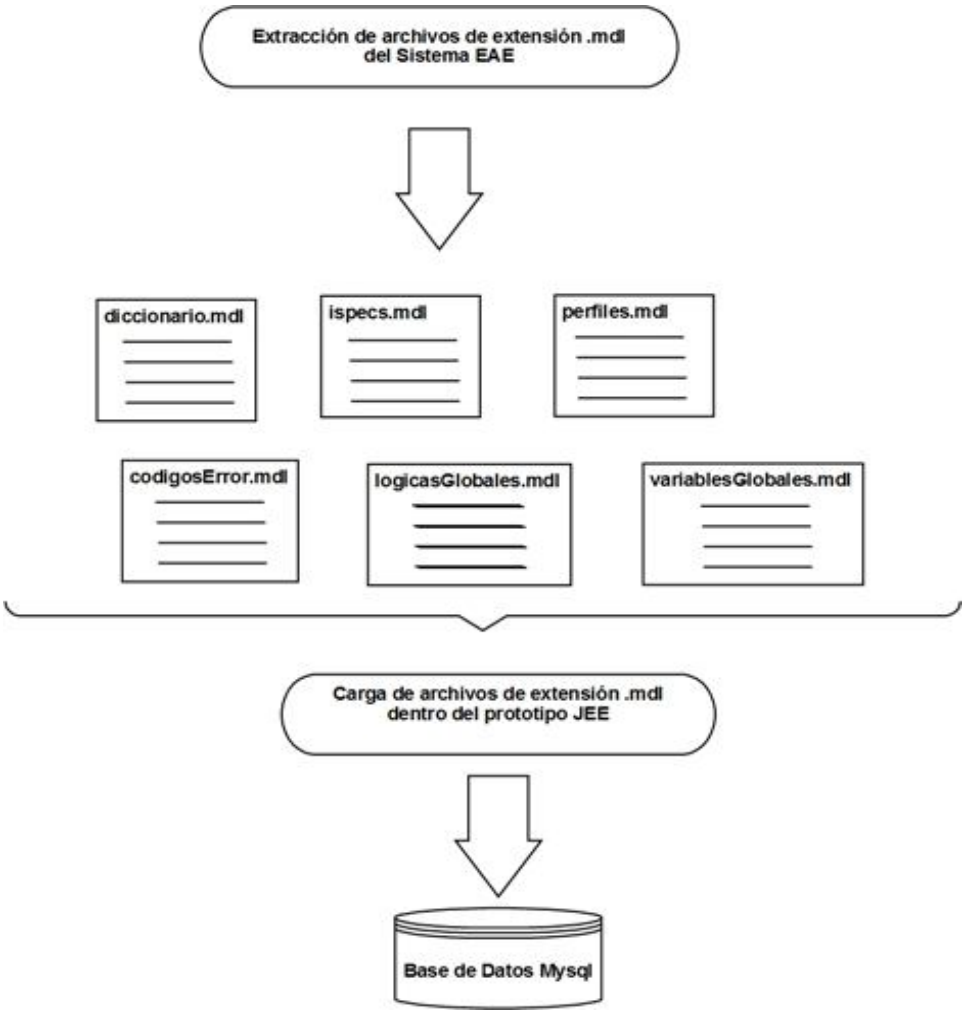
**Perfiles.mdl**

Perfiles del Sistema EAE.

**CodigosError.mdl**

Códigos de error del Sistema EAE.

En el siguiente diagrama se puede ver el proceso de carga de dichos archivos.

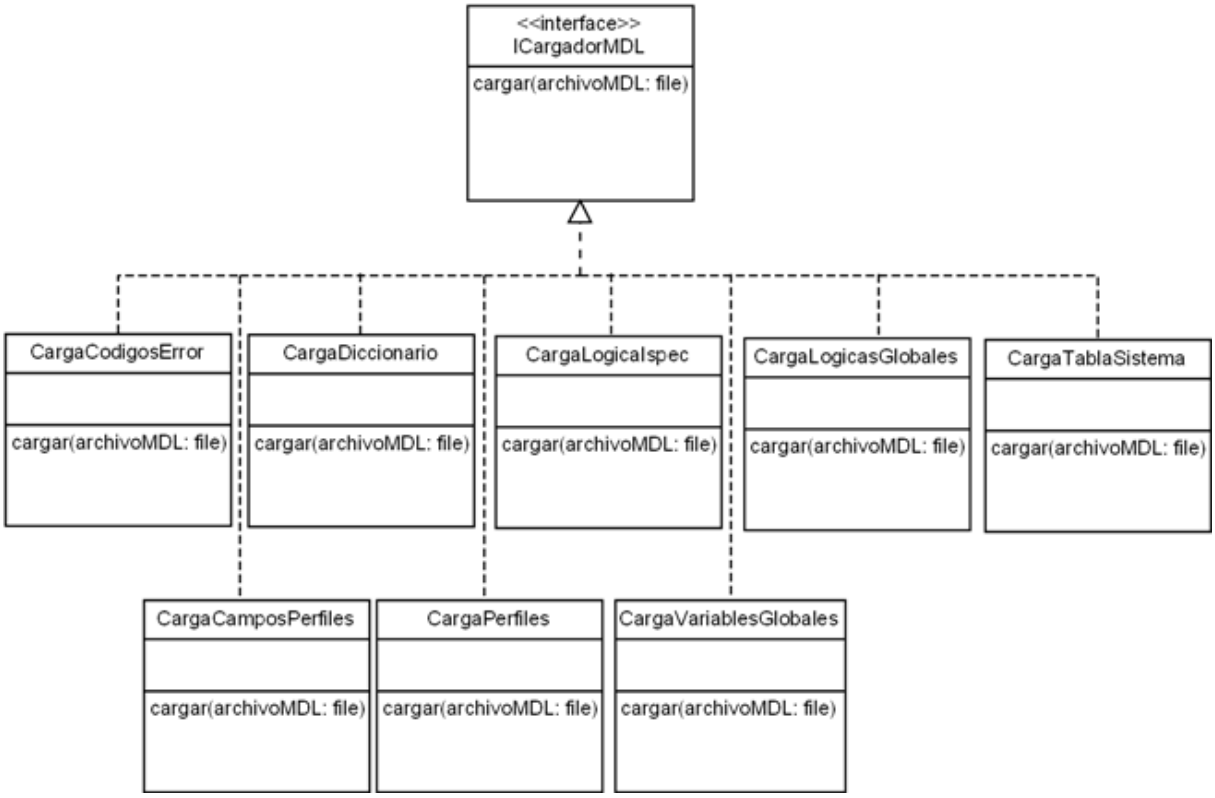



Para la carga de estas clases se diseñaron una serie de cargadores. El objetivo de dichos cargadores es cargar los archivos de extensión .mdl con las clases definidas y persistir esas clases en la base de datos. Para realizar esta funcionalidad se diseñó una interfaz ICargador MDL.


La interfaz tiene la operación

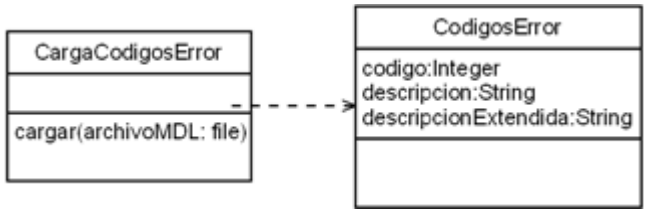
**cargar(archivoMDL:File)**

En donde archivoMDL es la ruta completa en donde se encuentra el archivo de extensión mdl que se quiere cargar. Hay cargadores específicos para cada tipo de archivo mdl, todos ellos implementando la interfaz ICargadorMDL.



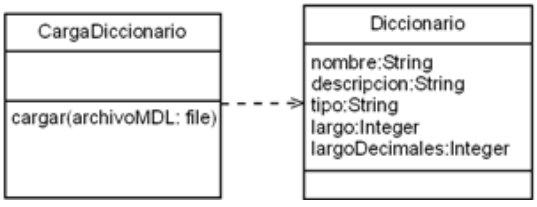

**CargaCodigosError**

Es el cargador responsable de cargar la información en la tabla CodigosError.




**CargaDiccionario**

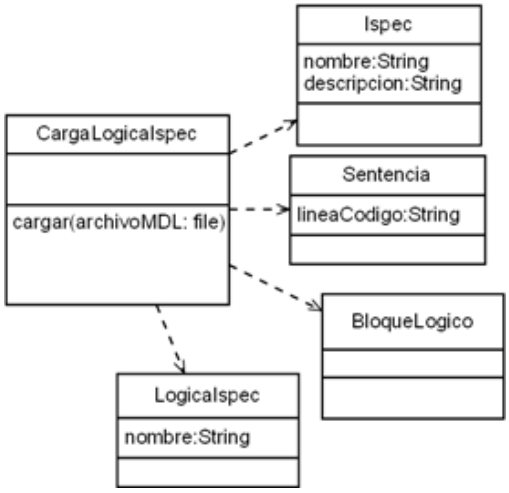
Es el cargador responsable de cargar la información en la tabla Diccionario.



**CargaLogicaIspec**

Es el cargador responsable de cargar la información en las tablas:

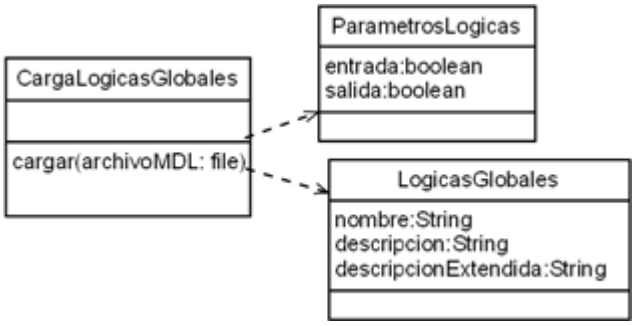
- LógicaIpec
- Ispec
- BloqueLogico
- Sentencia




**CargaLogicaGlobales**

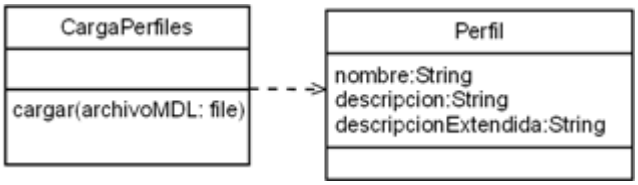
Es el cargador responsable de cargar la información en las tablas:

- LogicasGlobales
- ParametrosLogica



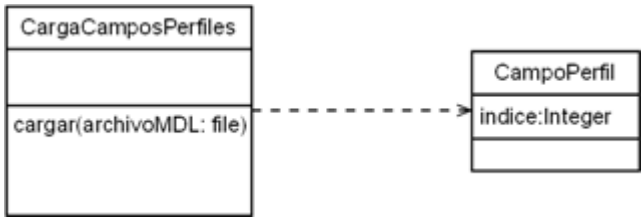

**CargaPerfiles**

Es el cargador responsable de cargar la información en la tabla: Perfil



**CargaCamposPerfiles**

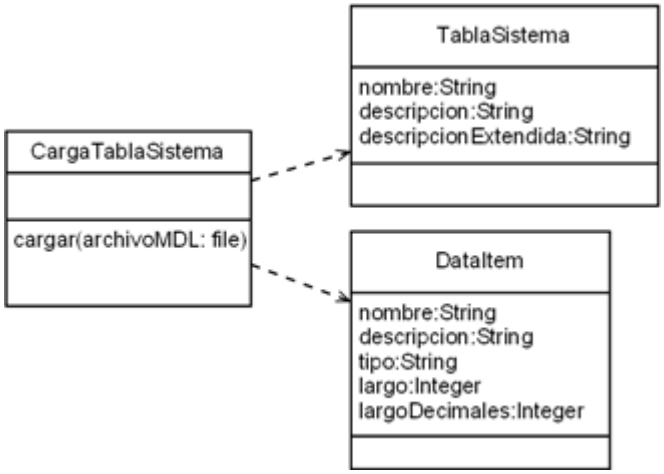
Es el cargador responsable de cargar la información en la tabla CampoPerfil




**CargaTablasSistema**

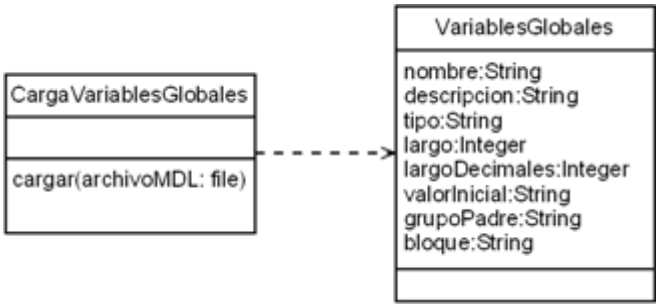
Es el cargador responsable de cargar la información en las tablas:

- TablasSistema
- DataItem




**CargaVariablesGlobales**

Es el cargador responsable de cargar la información en la tabla: VariablesGlobales



Para realizar los cargadores se investigó el formato de los archivos MDL en la Ayuda de la especificación de MDL

Para realizar la carga de los archivos mdl se tomaron en cuenta las dependencias entre los datos de los archivos.

El orden de carga fue el siguiente:

1. Diccionario
2. Variables Globales
3. Códigos de error
4. Lógicas Globales
5. Ispecs
6. Perfiles

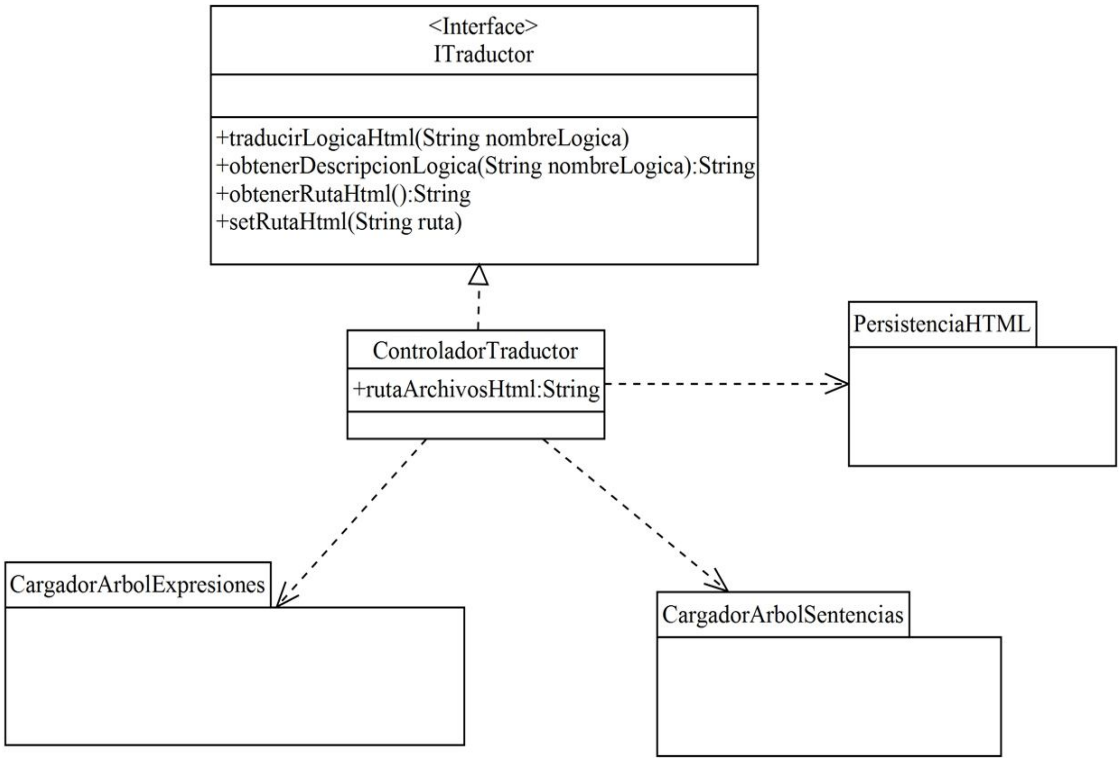
Por ejemplo: no se puede cargar el Perfil de una tabla sin antes tener cargada la Tabla.

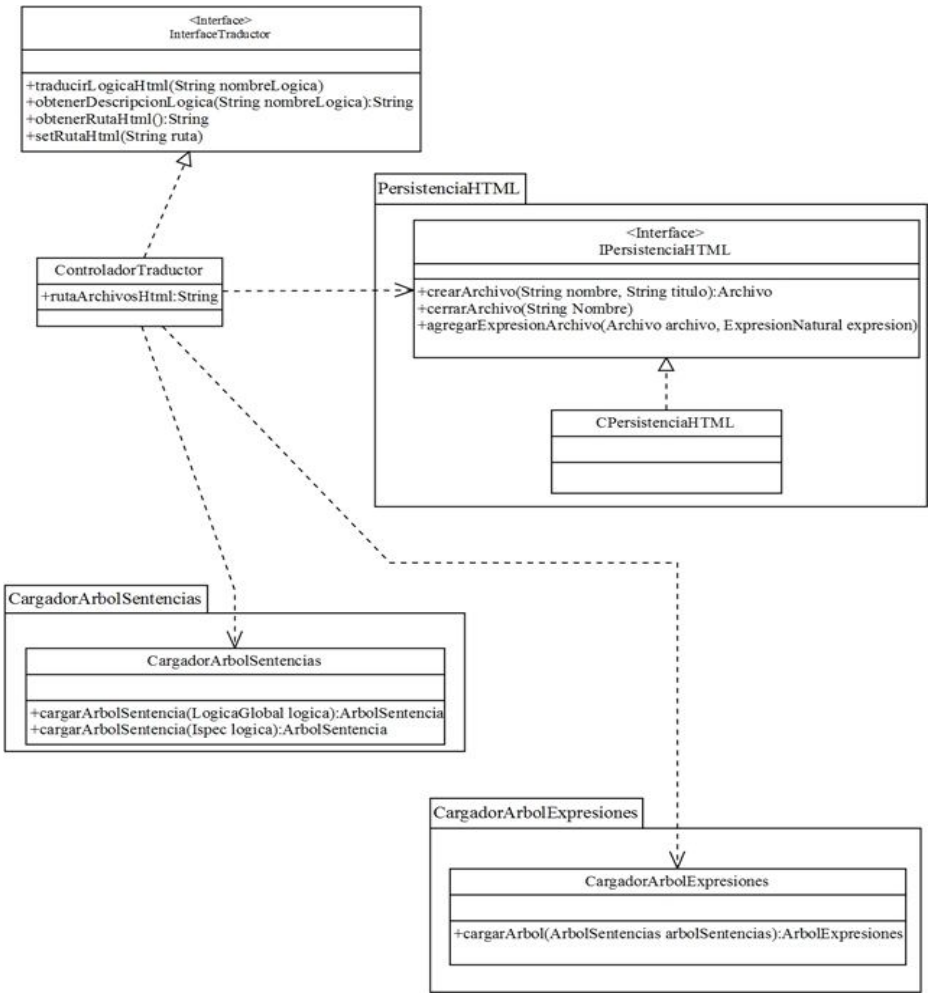
## 4.4 - Capa lógica

En esta capa se resuelve todo lo relacionado con la traducción. Esta capa se comunica con la capa de persistencia accediendo a los programas que serán traducidos, y utilizando información relevante en este sentido. A su vez es consumida por la capa de presentación.

### 4.4.1 - Diagrama de clases

°






El controlador ControladorTraductor implementa la interfaz ITraductor. También mantiene la ruta local en donde se generarán los archivos html de salida.

El manejo de los archivos html se hace a través de la interfaz IPersistenciaHTML la cual provee tres operaciones:

**crearArchivo(nombre:String, titulo:String):Archivo**

Esta operación se invoca para crear el archivo html de salida.

**agregarExpresionArchivo(archivo:Archivo, expresión:ExpresionNatural)**

Imprime la expresión en el archivo. La expresión es del tipo ExpresionNatural, detallada más adelante.

**cerrarArchivo(nombre:String)**

Esta operación cierra el archivo.

La clase controlador CPersistenciaHTML implementa la interfaz.

Para la traducción de la lógica deseada, el controlador ControladorTraductor utiliza dos estructuras auxiliares: ArbolSentencias y ArbolExpresiones. Dichas estructuras son cargadas mediante dos clases: CargadorArbolSentencias y CargadorArbolExpresiones.

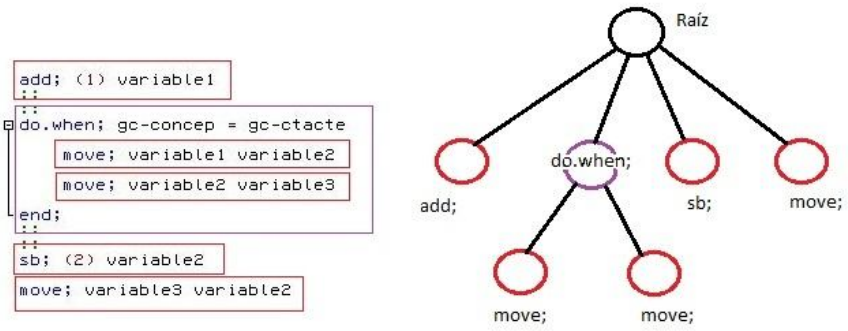
## 4.5 - Árbol de sentencias

El árbol de sentencias es un árbol sintáctico-semántico. El mismo es una representación del código. Es una estructura jerárquica, en donde cada nodo es una sentencia simple del código, y en casos de ciclos (while, for) o sentencias condicionales (if) se genera un subárbol en donde los hijos son todas las sentencias que se ejecutan si se dan las condiciones necesarias en la evaluación.

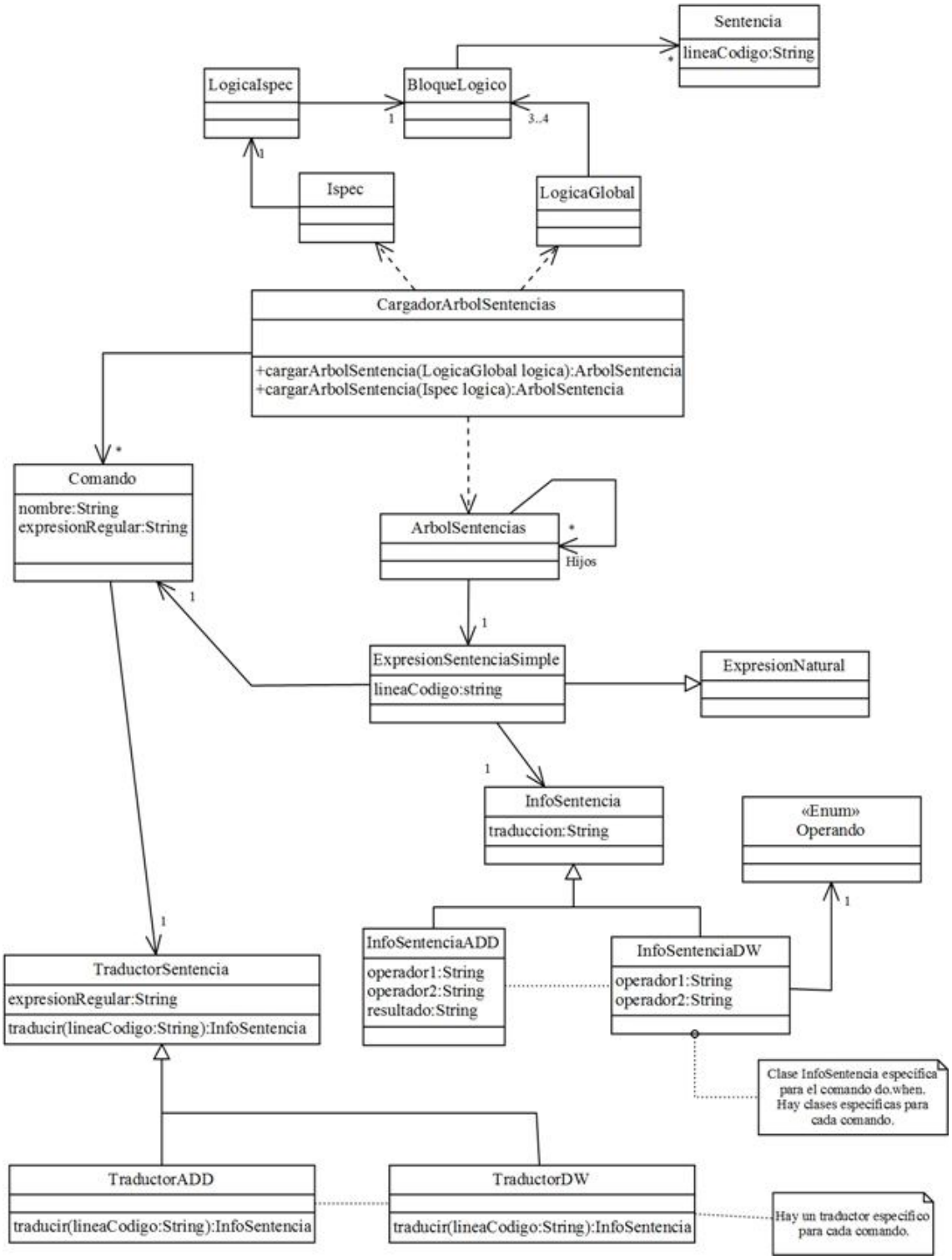
Por ejemplo, dado el siguiente código:

```
add; (1) variable1    // suma
::
::
do.when; gc-concep = gc-ctacte  // sentencia condicional
    move; variable1 variable2   // asignación simple
    move; variable2 variable3   // asignación simple
end;
::
::
sb; (2) variable2     // resta
move; variable3 variable2  // asignación simple
```

Tenemos varias sentencias, siendo una de ellas condicional (do.when). La sentencia condicional genera un subárbol cuyos hijos son todas las sentencias incluidas (dos move`s). Todas las hojas del árbol son sentencias no condicionales, o sentencias que no generan ciclos.
En la siguiente figura se ilustra la correspondencia del código con el árbol.



Cada nodo del árbol se corresponde con una sentencia. Se detalla en el siguiente diagrama de clases los componentes del árbol de sentencias y las estructuras necesarias para cargar el árbol.



### 4.5.1 - CargadorArbolSentencias

El controlador *CargarArbolSentencias* se encarga de cargar el árbol de sentencias a partir de una lista ordenada de sentencias. Estas sentencias son las líneas de código de cada lógica. Estas líneas de código son agrupadas en la clase *BloqueLogico*. Por lo que el controlador accede a todas las líneas de un programa (lógica global o lógica de un ispec) mediante los bloques lógicos.

### 4.5.2 - Comando

El controlador tiene una lista de comandos (pe: move, add, etc), que son los que se van a reconocer entre las líneas de código. Cada línea de código que no sea un espacio o un comentario, tiene un comando asociado. Cada comando tiene una expresión regular, que es la forma en que se puede utilizar el comando, y un traductor asociado.

### 4.5.3 - Traductor

Se debe implementar un traductor específico para cada comando. Cada traductor utiliza la expresión regular del comando para reconocer cada componente de la línea de código. Por ejemplo, en un comando MOVE, el cual es una asignación simple, se reconoce el primer operando y el segundo, almacenando ese valor en una clase que almacena información específica de esta operación, *TraductorMV*, la cual es subclase de la clase *Traductor*. Todas las subclases de *Traductor* tienen que implementar el método *traducir()*.

### 4.5.4 - InfoSentencia

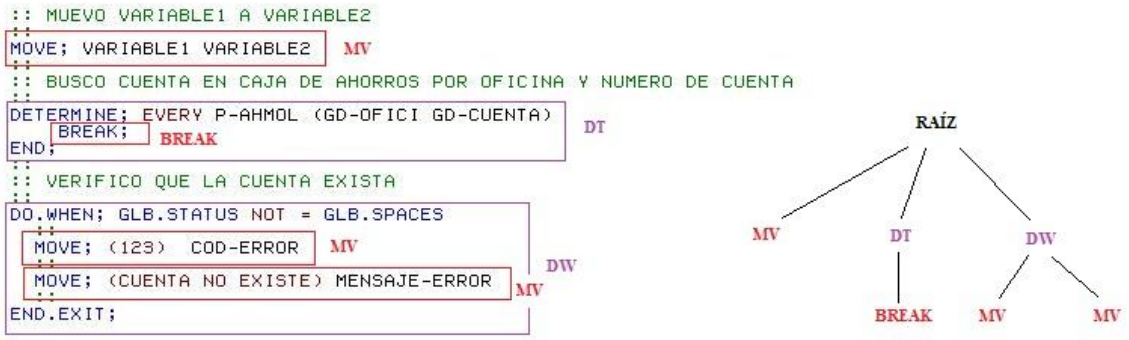
La clase *InfoSentencia* tiene subclases con información específica para cada comando. La clase *InfoSentenciaMV* es específica para el comando MOVE. Tendrá el operador origen (valor de asignación) y el operador destino (variable asignada). La clase *InfoSentenciaInsert* por ejemplo, tendrá información de la lógica invocada por el comando (el comando INSERT es una llamada a una lógica o subrutina).

## 4.6 - Árbol de expresiones

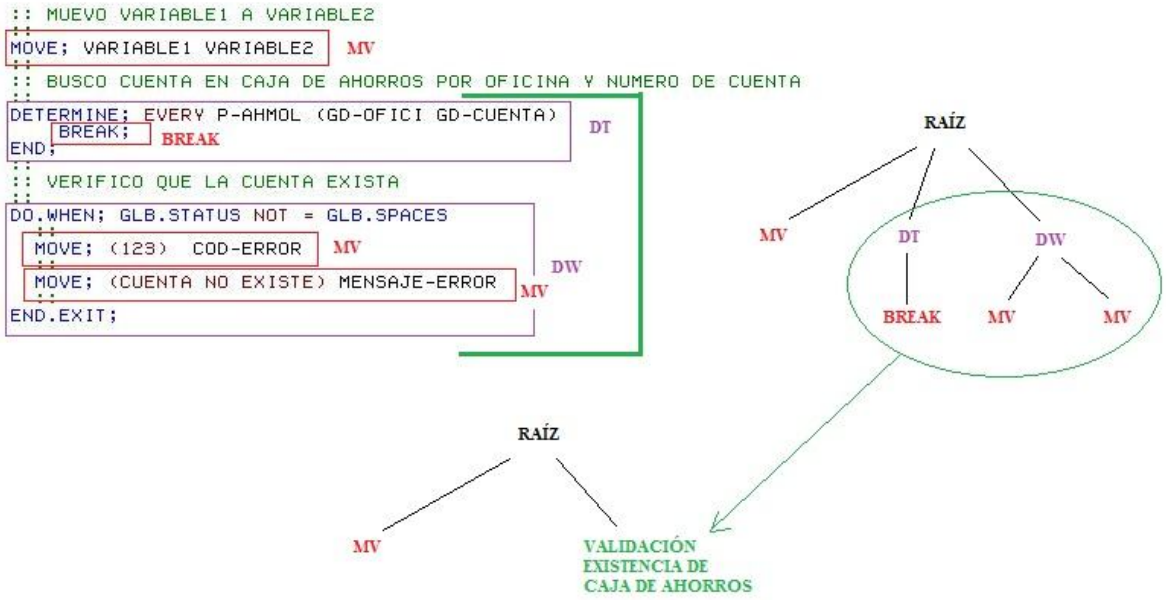
El árbol de expresiones también es un árbol sintáctico-semántico, aunque a diferencia del árbol de sentencias, cada nodo puede contener o bien una sentencia simple (un comando), o bien un conjunto de sentencias que representan una operación en el sistema, por ejemplo una validación de estado de una cuenta.

Por lo que para un conjunto de sentencias de un programa (lógica global, o lógica de Ispec en este caso), primero se carga el árbol de sentencias, y luego reconociendo conjuntos de estas sentencias se carga el árbol de expresiones.

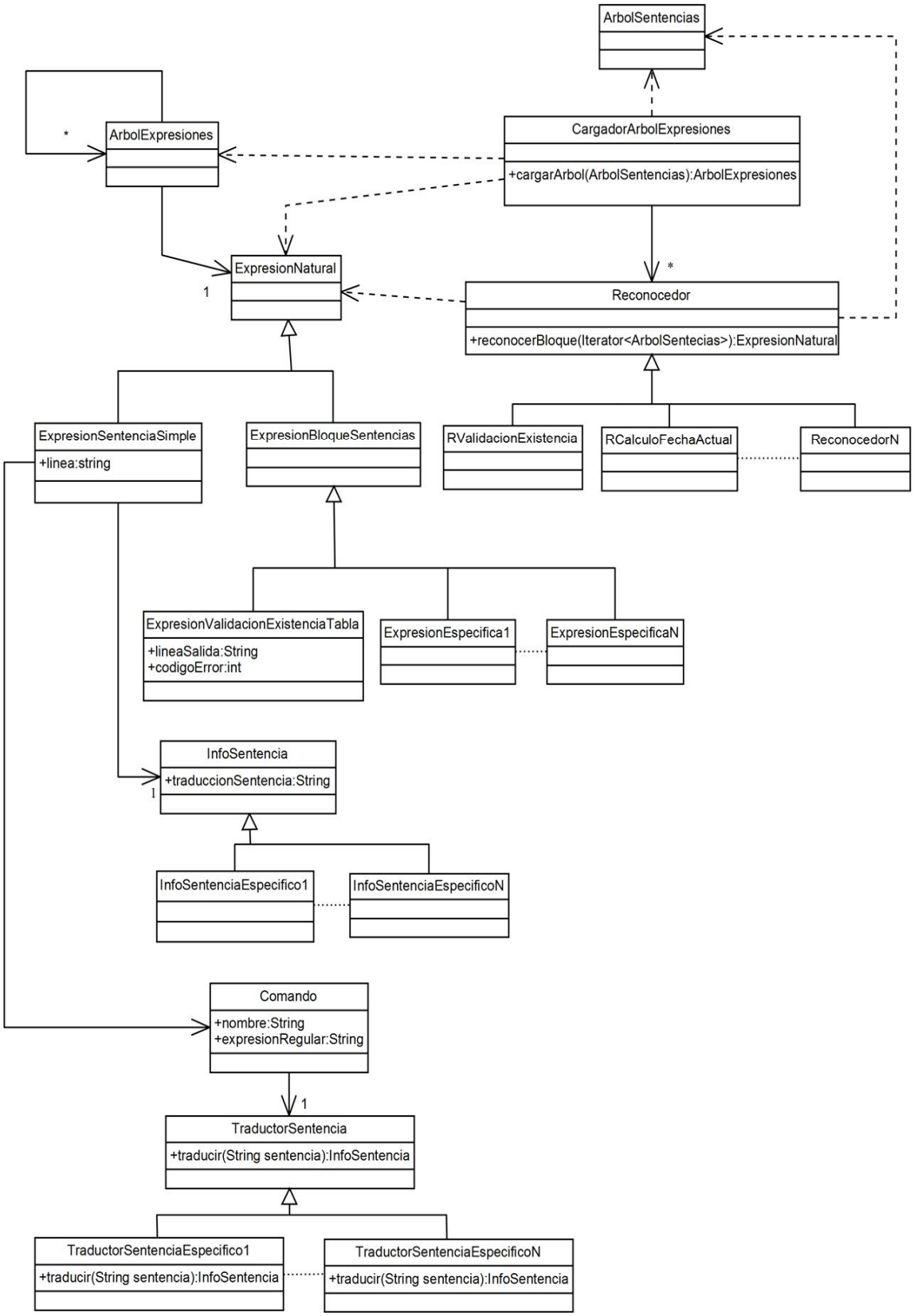
Para visualizarlo mejor, supongamos el siguiente caso. Tenemos una asignación simple, una consulta sobre una caja de ahorros, y la posterior verificación de que la caja de ahorros existe. Este proceso se puede codificar de la siguiente manera, obteniendo el siguiente árbol de sentencias.




Recorriendo el árbol podemos identificar conjuntos de nodos, los cuales representan conjuntos de sentencias simples del programa como operaciones complejas del sistema (por ejemplo la actualización de saldo de una caja de ahorros). El árbol de expresiones para el ejemplo tendría la siguiente forma.



El siguiente diagrama de clases muestra los componentes necesarios para este proceso.



### 4.6.1 - CargadorArbolExpresiones

Este controlador toma un árbol de sentencias simples (*ArbolSentencias*) y genera un árbol de expresiones (*ArbolExpresiones*). Las expresiones puede ser o bien una sentencia simple, o un conjunto de sentencias que determinan una operación más compleja.

### 4.6.2 - ArbolExpresiones

Esta estructura representa el árbol de expresiones. Cada nodo tiene lo que llamamos una expresión natural (clase *ExpresionSentenciaSimple*), la cual puede ser o bien una expresión de una sentencia simple (una asignación, una suma, etc), o un conjunto de operaciones, a lo que llamamos expresión de bloque de sentencias, representado en la clase *ExpresionBloqueSentencias*. Se hizo referencia a las expresiones simples en la sección anterior (CargadorArbolSentencias).

### 4.6.3 - ExpresionBloqueSentencias

Esta clase representa un conjunto de sentencias simples, las cual se pueden corresponder a una sola operación o acción del sistema. Esa clase es una clase abstracta. Para cada acción compleja específica que se reconozca en el código (por ejemplo una validación de estado de tarjeta de débito, una actualización de saldo en cuenta corriente, etc) se debe implementar la clase específica que tendrá información específica de esta acción. Por ejemplo en el caso de una validación de existencia de un registro, se puede almacenar la tabla sobre la cual se está verificando.

### 4.6.4 – Reconocedor

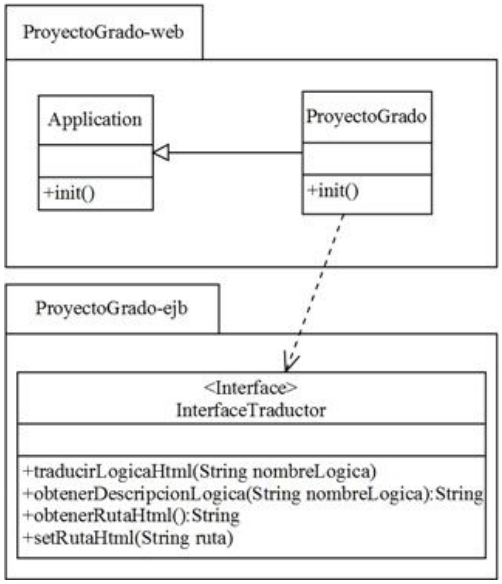
Es una clase abstracta con una operación, reconocerBloque, la cual a partir del iterador de un árbol de sentencias, reconoce una expresión (ExpresionNatural) desde un conjunto de sentencias. Esta expresión natural puede ser una sentencia simple, o un bloque de sentencias que representen una acción compleja. Se tiene que agregar una subclase de Reconocedor para cada tipo de expresión que quiera reconocer. En el diagrama aparece una validación de existencia de un registro en una base de datos, o el cálculo de una fecha (ambas acciones se implementan con varias sentencias en EAE).

## 4.7 - Capa de presentación

Para la interfaz gráfica se utilizó el framework vaadin (ex Mileston). La utilización de dicho framework facilitó el desarrollo de las pantallas, aunque las mismas no son complejas.

El framework utiliza una clase Application, y cada página que se desarrolle tiene que extender a esta clase, e implementar el método init(). Como ventaja, para la utilización de esta tecnología no es necesario el diseño de archivos jsp ni html.

La clase que implementa Application es ProyectoGrado. La misma se comunica con el traductor por intermedio de la interfaz ITraductor. En el siguiente diagrama de clases se ilustran las relaciones.



## 4.8 - Decisiones de diseño

Synchronized Refinement es una técnica de lectura de código que simultáneamente examina e interpreta código fuente generando una descripción del mismo.

Básicamente está técnica trata de responder qué hace un programa, por qué lo hace, y a su vez genera una abstracción del mismo y va registrando este conocimiento del programa en anotaciones (oraciones, sentencias, notas, etc).

Para el proyecto de extracción de reglas de negocio construimos una abstracción del programa a analizar utilizando conocimiento del lenguaje, así como conocimiento del negocio. Se identificaron patrones de diseño utilizados en dicho programa, y se fueron registrando anotaciones (o frases) que expresan una acción en el sistema.

La clase Traductor y sus subclases son las que generan estas anotaciones para cada sentencia, o conjunto de sentencias. Las anotaciones se modelaron utilizando las clases ExpresionNatural y sus subclases. La clase Reconocedor y sus subclases son las encargadas de reconocer cada patrón de diseño del código fuente, a partir del conocimiento de programación y de negocio ya incorporado, generando anotaciones (en este caso genera instancias de la clase ExpresionNatural).

Se puede ver este proceso de transformación como una compilación del código fuente, aunque con algunas diferencias con un compilador que veremos a continuación.

Si bien nuestro traductor no es un compilador, ya que no pasa por todos los procesos que un compilador necesita, tiene en común que toma como entrada una serie de sentencias escritas en un lenguaje, y obtiene como salida otra serie de sentencias en otro lenguaje (pseudocódigo en lenguaje natural). Por lo tanto, se utilizaron técnicas de diseño de compiladores para el desarrollo del traductor.

El proceso de un compilador se divide en análisis y síntesis, generando entre estos procesos una representación intermedia. Nuestro traductor no tiene verificaciones semánticas de qué tipo de datos son utilizados en asignaciones, o sobre qué posiciones se hace referencia en un vector, ya que partimos de código que fue compilado anteriormente por el compilador de EAE. Tampoco se muestran mensajes de error en cada proceso, ni se hace una evaluación de la correctitud del código (ni sintáctica ni semántica). Pero sí se reconocen patrones que son cargados en una estructura intermedia y posteriormente traducidos.

Nuestro proceso de análisis consta de leer el programa fuente línea por línea, reconociendo los diferentes comandos (para ello se utilizan expresiones regulares), e ir cargando una estructura auxiliar (ArbolSentencias). Luego de tener esta primera representación intermedia del código, se carga una segunda estructura (ArbolExpresiones) que agrega información a la primera. Se reconoce en el primer árbol determinados patrones y se mapean estos patrones reconocidos a otro tipo de estructura (ExpresionBloqueSentencia).

Luego de tener esta otra estructura intermedia, empieza el proceso de síntesis, del cual se obtiene el código objeto. En nuestro caso el código objeto consta de oraciones predefinidas en lenguaje natural que describen una acción en el sistema.

En cuanto a la capa de presentación, la misma se comunica con la capa lógica por medio de la interfaz ITraductor. El proyecto que resuelve la interfaz gráfica es diferente

del que resuelve la lógica, por lo que la presentación es totalmente independiente. Si se decidiera utilizar otro lenguaje para la capa de presentación (php, .net, etc), se podrían implementar web services fácilmente que contengan los métodos de la interfaz del traductor.

# CAPÍTULO 5: Pruebas

## 5.1 - Introducción

La planificación de las pruebas es una tarea que debe ser tenida en cuenta desde el inicio del ciclo de desarrollo de software. El objetivo de la verificación del software es descubrir faltas para evitar fallas, y así poder evaluar (y utilizarlo para mejorar) la calidad del producto.

Existen diferentes técnicas de verificación de software, dependiendo de la realidad que se desee probar. Se utilizaron algunas de ellas, las cuales se detallan en este capítulo junto con las razones para la elección de las mismas. [20]

## 5.2 - Estrategia de Pruebas

Cualquier técnica de pruebas tiene que tener en cuenta cuatro puntos: modelo de la realidad, técnica de selección, criterio de cubrimiento, teoría de errores.

**Modelo de la realidad**

El modelo que encontramos más adecuado fue el de conjuntos y clases de equivalencias, ya que encontramos entradas parecidas, las cuales se pueden agrupar en elementos con características similares. Estas entradas dependen de lo que se quiera probar.
Se puede dividir el desarrollo en dos etapas, la primera relacionada con el análisis, en la cual se define el modelo del sistema. Lo que se quiere probar es que este modelo se está cargando correctamente a partir de los archivos mdl que lo describen, por lo que las variables son las diferentes clases a cargar en la base de datos, las cuales representan objetos en el sistema (elementos del diccionario, tablas, programas, etc).
En la segunda parte del desarrollo, en la síntesis, lo que se quiere probar es que se está traduciendo bien cada sentencia, o bloque de sentencias. Por lo que las variables son sentencias o bloques de sentencias a traducir.

**Definición de técnica de selección**

Existen diferentes técnicas de selección de casos de prueba. Aquí aplicamos una técnica de caja blanca: partición en clases de equivalencias. Si bien la elección es arbitraria, y la técnica no tiene por qué se única a lo largo del proceso de verificación del software, se estimó que esta era la más adecuada dada la realidad planteada. Nuevamente dividimos el desarrollo en dos etapas: análisis y síntesis.

Para el análisis las variables son los elementos que componen el sistema, y para la elección de estas variables se eligieron representantes de cada objeto del sistema descriptos en los archivos mdl. Por ejemplo, como tabla del sistema se eligió BSMCL, que es en donde se guarda información de los clientes del banco.

Para la segunda etapa, la síntesis, se tomaron en cuenta todos los comandos (MOVE, DETERMINE, COMPUTE, etc.) y se eligió una sentencia con cada uno de ellos.

**Criterio de cubrimiento**

Una vez realizada la partición del dominio en clases de equivalencia se seleccionan elementos que sean representativos de cada clase. El criterio que se utilizó fue tomar elementos que sean fieles representantes de los de su clase, tanto para el análisis como para la síntesis. Se utilizaron todas las clases de equivalencia.

**Teoría de errores**

Se establece los tipos de error que pueden ocurrir. En este caso el error se da si no se graba correctamente el valor en la base de datos.

La definición de los casos de prueba tuvo la particularidad de que los archivos de extensión mdl que usamos como entrada de nuestras pruebas son generados por la herramienta, por lo que partimos de la base de que tienen un formato correcto. Por lo tanto, verificaciones sobre la correctitud de estos archivos no fueron tenidas en cuenta.

Primero se identifican las variables con sus dominios y sus posibles valores. Luego se definen las clases de equivalencias y se seleccionan los representantes más apropiados; finalmente creamos los casos de prueba.

Los casos de prueba diseñados se centraron en verificar que los datos guardados en la base de datos sean correctos. El oráculo es una entidad que determina los valores esperados en los casos de prueba. En este caso leyendo los archivos mdl y conociendo el modelo de la realidad, se estableció fácilmente el valor esperado en cada prueba.

En el caso de los traductores (etapa de síntesis), se ingresaron sentencias y se ejecutó el traductor comparando la salida (la traducción) contra el valor esperado de dicha traducción.

## 5.3 - Capa de persistencia (etapa análisis)

### 5.3.1 - Criterio de selección

#### CodigosError

Variables identificadas para el cargador

| Variable | Dominio | Entrada Válida | Entrada No válida |
|---|---|---|---|
| Código | Naturales{0,9999} | 1234 | -1,10000,12345 |
| Descripción | Texto [Largo Máximo 60] | “Cliente Invalido” | Texto con largo mayor a 60 caracteres. |

Selección de casos de prueba

| Variable | Clase de Equivalencia | Estado | CP01 | CP02 | CP03 | CP04 |
|---|---|---|---|---|---|---|
| Código | CE11:0=<x<=9999 | Válido | * | * | | |
| | CE12: x<0 | No válido | | | * | * |
| Descripción | CE21: x pertenece al Dominio | Válido | * | | * | |
| | CE22: x No pertenece al Dominio | No válido | | * | | * |

#### Diccionario

Variables identificadas para el cargador

| Variable | Dominio | Entrada Válida | Entrada No válida |
|---|---|---|---|
| Nombre | String[Largo Máximo 10] | “ACO-MONED” | String largo >10, String vacio |
| Tipo | String[Largo Máximo 1] | “A”, “N” | “B”,”” |
| Largo | Naturales positivos | 10, 5 | -1 |
| LargoDecimales | Naturales positivos | 2 | -1 |
| Descripción | Texto [Largo Máximo 30] | “Cliente Invalido” | Texto con largo mayor a 30 caracteres. |

Selección de casos de prueba

| Variable | Clase de Equivalencia | Estado | CP01 | CP02 | CP03 | CP04 | CP05 | CP06 |
|---|---|---|---|---|---|---|---|---|
| Nombre | CE11:0<String <=10 | Válido | * | * | | * | * | * |
| | CE12: String >10 | No válido | | | * | | | |
| Tipo | CE21:String x ="A" ò x="N" | Válido | * | * | * | | * | * |
| | CE22: String x No pertenece al Dominio | No válido | | | | * | | |
| Largo | CE31:Natural x >0 | Válido | * | * | * | * | | * |
| | CE32:Natural x <0 | No válido | | | | | * | |
| LargoDecimales | CE41:Natural x >0 | Válido | * | * | * | * | * | |
| | CE42:Natural x <0 | No válido | | | | | | * |
| Descripción | CE51:0<String <=30 | Válido | * | | * | * | * | * |
| | CE52: String >30 | No válido | | * | | | | |

**CampoPerfil**

Variables identificadas para el cargador

| Variable | Dominio | Entrada Válida | Entrada No válida |
|---|---|---|---|
| Id | Naturales positivos | 123 | -1 |
| Indice | Naturales positivos | 2 | -1 |

Selección de casos de prueba

| Variable | Clase de Equivalencia | Estado | CP01 | CP02 | CP03 | CP04 |
|---|---|---|---|---|---|---|
| Id | CE11:x>=0 | Válido | * | * | | |
| | CE12: x<0 | No válido | | | * | * |
| Indice | CE21:x>=0 | Válido | * | | * | |
| | CE22: x<0 | No válido | | * | | * |

**DataItem**

Variables identificadas para el cargador

| Variable | Dominio | Entrada Válida | Entrada No válida |
|---|---|---|---|
| Id | Naturales positivos | 222 | -1 |
| Nombre | String[Largo Máximo 30] | “ACO-CAJER” | String largo >30, String vacio |
| Tipo | String[Largo Máximo 1] | “A”, “N” | “B”,”” |
| Largo | Naturales positivos | 10, 5 | -1 |
| LargoDecimales | Naturales positivos | 2 | -1 |
| Descripción | Texto [Largo Máximo 30] | “Cliente Invalido” | Texto con largo mayor a 30 caracteres. |

Selección de casos de prueba

| Variable | Clase de Equivalencia | Estado | CP01 | CP02 | CP03 | CP04 | CP05 | CP06 | CP07 |
|---|---|---|---|---|---|---|---|---|---|
| Nombre | CE11:0<String <=10 | Válido | * | * | | * | * | * | * |
| | CE12: String >10 | No válido | | | * | | | | |
| Tipo | CE21:String x ="A" ò x="N" | Válido | * | * | * | | * | * | * |
| | CE22: String x No pertenece al Dominio | No válido | | | | * | | | |
| Largo | CE31:Natural x >0 | Válido | * | * | * | * | | * | * |
| | CE32:Natural x <0 | No válido | | | | | * | | |
| LargoDecimales | CE41:Natural x >0 | Válido | * | * | * | * | * | | * |
| | CE42:Natural x <0 | No válido | | | | | | * | |
| Descripción | CE51:0<String <=30 | Válido | * | * | * | * | * | * | |
| | CE52: String >30 | No válido | | | | | | | * |
| Id | CE61:Natural x >0 | Válido | * | | * | * | * | * | * |
| | CE62:Natural x <0 | No válido | | * | | | | | |

## Ispec

Variables identificadas para el cargador

| Variable | Dominio | Entrada Válida | Entrada No válida |
|---|---|---|---|
| Nombre | String[Largo=5] | “BSMCL” | String largo >5, String largo <5 |
| Descripción | Texto [Largo Máximo 30] | “Maestro de Clientes” | Texto con largo mayor a 30 caracteres. |

Selección de casos de prueba

| Variable | Clase de Equivalencia | Estado | CP01 | CP02 | CP03 | CP04 |
|---|---|---|---|---|---|---|
| Nombre | CE11: String s largo =5 | Válido | * | * | | |
| | CE12: String s largo distinto 5 | No válido | | | * | * |
| Descripción | CE21:0<String <=30 | Válido | * | | * | |
| | CE22: String >30 | No válido | | * | | * |

## LogicaGlobal

Variables identificadas para el cargador

| Variable | Dominio | Entrada Válida | Entrada No válida |
|---|---|---|---|
| Nombre | String[Largo Máximo 17] | “GP-GRABA” | String largo >17 |
| Descripción | Texto [Largo Máximo 30] | “Maestro de Clientes” | Texto con largo mayor a 30 caracteres. |
| DescripciónUsuario | Texto [Largo Máximo 30] | “Maestro de Clientes” | Texto con largo mayor a 30 caracteres. |

Selección de casos de prueba

| Variable | Clase de Equivalencia | Estado | CP01 | CP02 | CP03 | CP04 |
|---|---|---|---|---|---|---|
| Nombre | CE11: String s 0<largo<17 | Válido | * | * | | * |
| | CE12: String s largo >17 | No válido | | | * | |
| Descripción | CE21:0<String <=30 | Válido | * | * | * | |
| | CE22: String >30 | No válido | | | | * |
| Descripción Usuario | CE31:0<String <=30 | Válido | * | | * | * |
| | CE32: String >30 | No válido | | * | | |

**LogicalSpec**

Variables identificadas para el cargador

| Variable | Dominio | Entrada Válida | Entrada No válida |
|---|---|---|---|
| Nombre | String[Largo Máximo 17] | "GP-GRABA" | String largo >17 |

Selección de casos de prueba

| Variable | Clase de Equivalencia | Estado | CP01 | CP02 |
|---|---|---|---|---|
| Nombre | CE11: String s 0<largo<17 | Válido | * | |
| | CE12: String s largo >17 | No válido | | * |

**Sentencia**

Variables identificadas para el cargador

| Variable | Dominio | Entrada Válida | Entrada No válida |
|---|---|---|---|
| Comando | String[Largo Máximo 120] | "DO.WHEN; PPMAP.PCU-MONED NOT = WS-MONDOL" | String largo >120 |

Selección de casos de prueba

| Variable | Clase de Equivalencia | Estado | CP01 | CP02 |
|---|---|---|---|---|
| Comando | CE11: String s 0<largo<120 | Válido | * | |
| | CE12: String s largo >120 | No válido | | * |

**Perfil**

Variables identificadas para el cargador

| Variable | Dominio | Entrada Válida | Entrada No válida |
|---|---|---|---|
| NombrePerfil | String[Largo Máximo 17] | "P-BSMCL" | String largo >17 |
| TablaSistema | String[Largo =5] | BSMCL | BSMCL12 |
| Descripción | Texto [Largo Máximo 30] | "Maestro de Clientes" | Texto con largo mayor a 30 caracteres. |
| DescripciónUsuario | Texto [Largo Máximo 30] | "Maestro de Clientes" | Texto con largo mayor a 30 caracteres. |

Selección de casos de prueba

| Variable | Clase de Equivalencia | Estado | CP01 | CP02 | CP03 | CP04 | CP05 |
|---|---|---|---|---|---|---|---|
| TablaSistema | CE11: String s largo =5<br>CE12: String s largo distinto 5 | Válido<br>No válido | * | * | * | * | * |
| NombrePerfil | CE21:0<String <=17<br>CE22: String >17 | Válido<br>No válido | * | * | * | * | * |
| Descripción Usuario | CE31:0<String <=30<br>CE32: String >30 | Válido<br>No válido | * | * | * | * | * |
| Descripción | CE41:0<String <=10<br>CE42: String >10 | Válido<br>No válido | * | * | * | * | * |

## TablaSistema

Variables identificadas para el cargador

| Variable | Dominio | Entrada Válida | Entrada No válida |
|---|---|---|---|
| Nombre | String[Largo =5] | BSMCL | BSMCL12 |
| Descripción | Texto [Largo Máximo 30] | “Maestro de Clientes” | Texto con largo mayor a 30 caracteres. |
| DescripciónUsuario | Texto [Largo Máximo 30] | “Maestro de Clientes” | Texto con largo mayor a 30 caracteres. |

Selección de casos de prueba

| Variable | Clase de Equivalencia | Estado | CP01 | CP02 | CP03 | CP04 |
|---|---|---|---|---|---|---|
| Nombre | CE11: String s largo =5<br>CE12: String s largo distinto 5 | Válido<br>No válido | * | * | * | * |
| Descripción | CE21:0<String <=30<br>CE22: String >30 | Válido<br>No válido | * | * | * | * |
| Descripción Usuario | CE31:0<String <=30<br>CE32: String >30 | Válido<br>No válido | * | * | * | * |

## VariablesGlobales

Variables identificadas para el cargador

| Variable | Dominio | Entrada Válida | Entrada No válida |
|---|---|---|---|
| NombreVariable | String[Largo Máximo 30] | “GBS-LARGO” | String largo >30, String vacio |
| ValorInicial | String[Largo Máximo 10] | “(132)”, “” | (XXXXXXXXXXXXX) |
| Tipo | String[Largo Máximo 1] | “A”, “N” | “B”,”” |
| Largo | Naturales positivos | 3 | -1 |
| LargoDecimales | Naturales positivos | 0 | -1 |
| GrupoPadre | String[Largo Máximo 10] | “GC-TAREAS” | String largo >10 |
| Bloque | String[Largo Máximo 10] | “GLB-SDS” | String largo >10 |
| DiccionarioNombre | String[Largo Máximo 10] | “ACO-TAREA” | String largo >10 |
| Descripción | Texto [Largo Máximo 30] | “Cliente Invalido” | Texto con largo mayor a 30 caracteres. |

Selección de casos de prueba

| Variable | Clase de Equivalencia | Estado | CP01 | CP02 | CP03 | CP04 | CP05 | CP06 | CP07 | CP08 | CP09 | CP10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| NombreVariable | CE11:0<String <=30 | Válido | * | * | | * | * | * | * | * | * | * |
| | CE12: String >30 | No válido | | | * | | | | | | | |
| ValorInicial | CE11:0<String <=10 | Válido | * | * | * | | * | * | * | * | * | * |
| | CE12: String >10 | No válido | | | | * | | | | | | |
| Tipo | CE21:String x ="A" ò x="N" | Válido | * | * | * | * | | * | * | * | * | * |
| | CE22: String x No pertenece al Dominio | No válido | | | | | * | | | | | |
| Largo | CE31:Natural x >0 | Válido | * | * | * | * | * | | * | * | * | * |
| | CE32:Natural x <0 | No válido | | | | | | * | | | | |
| LargoDecimales | CE41:Natural x >0 | Válido | * | * | * | * | * | * | | * | * | * |
| | CE42:Natural x <0 | No válido | | | | | | | * | | | |
| Descripción | CE51:0<String <=30 | Válido | * | * | * | * | * | * | * | | * | * |
| | CE52: String >30 | No válido | | | | | | | | * | | |
| GrupoPadre | CE11:0<String <=10 | Válido | * | * | * | * | * | * | * | * | | * |
| | CE12: String >10 | No válido | | | | | | | | | * | |
| Bloque | CE11:0<String <=10 | Válido | * | * | * | * | * | * | * | * | * | |
| | CE12: String >10 | No válido | | | | | | | | | | * |
| DiccionarioNombre | CE11:0<String <=10 | Válido | * | | * | * | * | * | * | * | * | * |
| | CE12: String >10 | No válido | | * | | | | | | | | |

La definición de los casos de prueba tuvo la particularidad de que los archivos de extensión mdl que usamos como entrada de nuestras pruebas son generados por la herramienta, por lo que partimos de la base de que tienen un formato correcto. Por lo tanto, verificaciones sobre la correctitud de estos archivos no fueron tenidas en cuenta.

Los casos de prueba diseñados se centraron en verificar que los datos guardados en la base de datos sean correctos. El oráculo es una entidad que determina los valores esperados en los casos de prueba. En este caso leyendo los archivos mdl y conociendo el modelo de la realidad, se estableció fácilmente el valor esperado en cada prueba.

### 5.3.2 - Criterio de cubrimiento

El criterio elegido fue el de usar todas las clases de equivalencia.

### 5.3.3 - Teoría de errores

Se utilizó principalmente la comparación de los resultados obtenidos contra la información de la base de datos.

## 5.4 - Capa lógica (etapa de síntesis)

### 5.4.1 - Criterio de selección

#### TraductorAdd

Variables identificadas

| Variable | Dominio | Entrada Válida | Entrada No válida |
|---|---|---|---|
| Comando | String {ADD;} | ADD; | A |
| Parametro1 | String[Largo Máximo 20 ] | GD-PARUNO | String con largo mayor a 20 caracteres. |
| Parametro2 | String [Largo Máximo 20] | GD-PARDOS | String con largo mayor a 20 caracteres. |

Selección de casos de prueba

| Variable | Clase de Equivalencia | Estado | CP01 | CP02 | CP03 | CP04 |
|---|---|---|---|---|---|---|
| Comando | CE11: String s =ADD; | Válido | * | * | | * |
| | CE12: String s NOT= ADD; | No válido | | | * | |
| Parametro1 | CE21:0<String <=20 | Válido | * | * | * | |
| | CE22: String >20 | No válido | | | | * |
| Parametro2 | CE31:0<String <=20 | Válido | * | | * | * |
| | CE32: String >20 | No válido | | * | | |

#### TraductorSubstract

Variables identificadas

| Variable | Dominio | Entrada Válida | Entrada No válida |
|---|---|---|---|
| Comando | String {SUBSTRACT;,SUB;} | SUB; ó SUBTRACT; | A |
| Parametro1 | String[Largo Máximo 20 ] | GD-PARUNO | String con largo mayor a 20 caracteres. |
| Parametro2 | String [Largo Máximo 20] | GD-PARDOS | String con largo mayor a 20 caracteres. |

Selección de casos de prueba

| Variable | Clase de Equivalencia | Estado | CP01 | CP02 | CP03 | CP04 |
|---|---|---|---|---|---|---|
| Comando | CE11: String s ={SUB; ,SUBSTRACT;} | Válido | * | * | | * |
| | CE12: String s NOT ={SUB; ,SUBSTRACT;} | No válido | | | * | |
| Parametro1 | CE21:0<String <=20 | Válido | * | * | * | |
| | CE22: String >20 | No válido | | | | * |
| Parametro2 | CE31:0<String <=20 | Válido | * | | * | * |
| | CE32: String >20 | No válido | | * | | |

## TraductorDo.When

### Variables identificadas

| Variable | Dominio | Entrada Válida | Entrada No válida |
|---|---|---|---|
| Comando | String {DO.WHEN;} | DO.WHEN; | A |
| Operando1 | String[Largo Máximo 20 ] | GD-PARUNO | String con largo mayor a 20 caracteres. |
| Operando2 | String [Largo Máximo 20] | GD-PARDOS | String con largo mayor a 20 caracteres. |
| Operador | String {<,>,=,NOT=,>=,<=} | = | X |

### Selección de Casos de prueba

| Variable | Clase de Equivalencia | Estado | CP01 | CP02 | CP03 | CP04 | CP05 |
|---|---|---|---|---|---|---|---|
| Comando | CE11: String s =DO.WHEN;<br>CE12: String s NOT =DO.WHEN; | Válido<br>No válido | * | * | <br>* | * | * |
| Operando1 | CE21:0<String <=20<br>CE22: String >20 | Válido<br>No válido | * | * | * | <br>* | * |
| Operando2 | CE31:0<String <=20<br>CE32: String >20 | Válido<br>No válido | * | * | * | * | <br>* |
| Operador | CE41:0<String <=20<br>CE42: String >20 | Válido<br>No Válido | * | <br>* | * | * | * |

## TraductorDetermine

### Variables identificadas

| Variable | Dominio | Entrada Válida | Entrada No válida |
|---|---|---|---|
| Comando | String {DT;,DETERMINE;} | DT; ó DETERMINE; | A |
| TipoAcceso | String { EVERY,ACTUAL,GROUP} | EVERY | CUALQUIERA |
| Perfil | String [Largo Máximo 9] | P-BSMCL | P-BSMCL1234 |
| camposFijos | String {<,>,=,NOT=,>=,<=} | = | X |
| camposDesde | String | SD-PRIAPEDES | XXXXXXXXXXXXXXXXXX |
| camposHasta | String | SD-CODCLI | XXXXXXXXXXXXXXXXXX |
| ListaCampos | Lista{String} | { SD-CODCLI , GLB.HIGH } | {XXXXXXXXXXXXXXXXX, YYYYYYYYYYYYYYYYYY} |
| Secure | SECURE | SECURE | XX |

Selección de casos de prueba

| Variable | Clase de Equivalencia | Estado | CP01 | CP02 | CP03 | CP04 | CP05 | CP06 | CP07 | CP08 | CP09 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Comando | CE11:s String= {DT;,DETERMINE;}<br>CE12: s String NOT={DT;,DETERMINE;} | Válido<br>No válido | * | * | <br>* | * | * | * | * | * | * |
| TipoAcceso | CE21:0 s String = {EVERY,ACTUAL,GROUP}<br>CE22:0 s String NOT = {EVERY,ACTUAL,GROUP} | Válido<br>No válido | * | * | * | <br>* | * | * | * | * | * |
| Perfil | CE31:String s largo <=9<br>CE32: String s largo >9 | Válido<br>No válido | * | * | * | * | <br>* | * | * | * | * |
| camposFijos | CE41:s String= {<,>,=,NOT=,>=,<=}<br>CE42:s StringNOT= {<,>,=,NOT=,>=,<=} | Válido<br>No válido | * | * | * | * | * | <br>* | * | * | * |
| camposDesde | CE51:s 60 >String >0<br>CE52:s 60 >String >0 | Válido<br>No válido | * | * | * | * | * | * | <br>* | * | * |
| camposHasta | CE61:s 60 >String >0<br>CE62:s 60 >String >0 | Válido<br>No válido | * | * | * | * | * | * | * | <br>* | * |
| ListaCampos | CE71:Lista{String}<br>CE72: Lista{String} | Válido<br>No válido | * | * | * | * | * | * | * | * | <br>* |
| Secure | CE81:String =SECURE<br>CE82: String NOT= SECURE | Válido<br>No válido | * | <br>* | * | * | * | * | * | * | <br>* |

**TraductorInsert**

Variables identificadas

| Variable | Dominio | Entrada Válida | Entrada No válida |
|---|---|---|---|
| Comando | String {INSERT;,INS;} | INS; ó INSERT; | A |
| NombreLogica | String[Largo Máximo 20 ] | GI-PANT-BASE | String con largo mayor a 20 caracteres. |

Selección de casos de prueba

| Variable | Clase de Equivalencia | Estado | CP01 | CP02 | CP03 |
|---|---|---|---|---|---|
| Comando | CE11: String s {INSERT;,INS;}<br>CE12: String s NOT {INSERT;,INS;} | Válido<br>No válido | * | * | <br>* |
| NombreLogica | CE21:0<String <=20<br>CE22: String >20 | Válido<br>No válido | * | <br>* | * |

**TraductorMove**

Variables identificadas

| Variable | Dominio | Entrada Válida | Entrada No válida |
|---|---|---|---|
| Comando | String {MOVE,MV} | MV ó MOVE | A |
| Parametro1 | String[Largo Máximo 20 ] | GD-PARUNO | String con largo mayor a 20 caracteres. |
| Parametro2 | String [Largo Máximo 20] | GD-PARDOS | String con largo mayor a 20 caracteres. |

Selección de casos de prueba

| Variable | Clase de Equivalencia | Estado | CP01 | CP02 | CP03 | CP04 |
|---|---|---|---|---|---|---|
| Comando | CE11: String s =ADD;<br>CE12: String s NOT= ADD; | Válido<br>No válido | * | * | <br>* | * |
| Parametro1 | CE21:0<String <=20<br>CE22: String >20 | Válido<br>No válido | * | * | * | <br>* |
| Parametro2 | CE31:0<String <=20<br>CE32: String >20 | Válido<br>No válido | * | <br>* | * | * |

## TraductorFlag

Variables identificadas

| Variable | Dominio | Entrada Válida | Entrada No válida |
|---|---|---|---|
| Comando | String {FLAG;,FL;,FG;} | FL ó FG ó FLAG | A |
| Operador1 | String[Largo Máximo 30 ] | GD-RESTO | String con largo mayor a 30 caracteres. |
| Operador2 | String [Largo Máximo 30] | BSASL.AVA-UTILI | String con largo mayor a 30 caracteres. |

Selección de casos de prueba

| Variable | Clase de Equivalencia | Estado | CP01 | CP02 | CP03 | CP04 |
|---|---|---|---|---|---|---|
| Comando | CE11: String s ={FLAG,FL,FG};<br>CE12: String s NOT= {FLAG,FL,FG}; | Válido<br>No válido | * | * | <br>* | * |
| Operador1 | CE21:0<String <=30<br>CE22: String >30 | Válido<br>No válido | * | * | * | <br>* |
| Operador2 | CE31:0<String <=30<br>CE32: String >30 | Válido<br>No válido | * | <br>* | * | * |

## TraductorAttach

Selección de casos de prueba

| Variable | Dominio | Entrada Válida | Entrada No válida |
|---|---|---|---|
| Comando | String {ATTACH;,ATT;} | ATT; ó ATTACH; | A |
| Parametro1 | String[Largo Máximo 60 ] | GD-PARUNO | String con largo mayor a 60 caracteres. |
| Parametro2 | String [Largo Máximo 60] | GD-PARDOS | String con largo mayor a 60 caracteres. |
| Resultado | String [Largo Máximo 120] | GD-PARUNO GD-PARDOS | String con largo mayor a 120 caracteres. |

Selección de casos de prueba

| Variable | Clase de Equivalencia | Estado | CP01 | CP02 | CP03 | CP04 | CP05 |
|---|---|---|---|---|---|---|---|
| Comando | CE11: String s ={ATTACH;,ATT;}<br>CE12: String s NOT= {ATTACH;,ATT;} | Válido<br>No válido | * | * | <br>* | * | * |
| Operador1 | CE21:0<String <=60<br>CE22: String >60 | Válido<br>No válido | * | * | * | <br>* | * |
| Operador2 | CE31:0<String <=60<br>CE32: String >60 | Válido<br>No válido | * | * | * | * | <br>* |
| Resultado | CE31:0<String <=120<br>CE32: String >120 | Válido<br>No válido | * | <br>* | * | * | * |

## 5.4.2 - Criterio de cubrimiento

El criterio elegido fue el de usar todas las clases de equivalencia.

## 5.4.3 - Teoría de errores

Comparación de las traducciones obtenidas con las esperadas para cada sentencia.

Para la ejecución de estas pruebas se generaron dumps en la base de datos para mantener el estado inicial de la misma, y así poder recuperarlo luego de cada ejecución. También se implementaron servicios web con el objetivo de que las lógicas que se querían probar corran en el servidor de aplicaciones, y para simplificar las pruebas. Estos servicios web tenían diferentes métodos para cada prueba específica.

Una alternativa podría haber sido la implementación de stubs y mocks para algunas funcionalidades, pero dado el proceso que elegimos (iterativo incremental) decidimos ir probando los módulos a medida que se construían, e ir integrando las pruebas de los nuevos módulos con estos ya construidos y probados (por supuesto siempre pueden aparecer nuevos errores con nuevas pruebas). Para ciertas pruebas (traducciones simples, y utilización de expresiones regulares) se utilizó JUnit [19] por su simplicidad e integración con la herramienta de desarrollo (NetBeans[12]).

# CAPÍTULO 6: Conclusiones y Trabajo Futuro

Desarrollamos un prototipo que extrae las reglas de negocio de EAE y tiene como salida un pseudocódigo que puede ser interpretado por usuarios con conocimientos bancarios.
Encontramos el marco teórico para nuestro desarrollo en la Teoría de Compiladores y en el proceso de “Synchronized Refinement”.
Consideramos que es posible seguir trabajando en este proyecto, sumando nuevas funcionalidades.
Como parte de nuestro desarrollo identificamos patrones que cumplen bloques de código que representan una acción u operación (por ejemplo una validación). En esta identificación nos restringimos a las lógicas incluidas en el alcance del proyecto. Como trabajo futuro se deberían identificar otros patrones de bloques de código que puedan escribirse en programas EAE. Incluso sería posible extraer información del flujo de control de un programa, y tener como salida un diagrama de flujo.
Por otro lado es posible agregar una interfaz que permita al usuario realizar la carga de la información a partir de los archivos mdl (los métodos para realizar dicha carga están desarrollados, pero no tienen una interfaz para el usuario sino que se implementó un programa que invocó a cada uno ellos).
También es posible proveer consultas con otros datos que tenemos cargados en la base de datos, como por ejemplo cantidad de transacciones, de tablas, de índices, etc.
Para la información estadística de SFB en producción realizamos un desarrollo auxiliar que captura los logs de transacciones EAE y los carga en una base de datos. Esto puede sumarse a nuestro prototipo y tendríamos no solamente la información con la definición del sistema sino además toda aquella referida a su desempeño en producción.
Si además contamos con acceso a información de la base de datos de SFB podríamos almacenar estadísticas de uso de tablas, tasas de crecimiento, espacio ocupado, etc.
En definitiva podríamos contar en una sola base de datos con toda la información relevante del sistema, y brindar al usuario múltiples consultas que le permitan sacar provecho de la misma.
Por supuesto que este desarrollo no solo es aplicable a SFB, sino también a los otros aplicativos EAE del banco, y en definitiva a cualquier aplicativo EAE.
Siendo más ambiciosos, considerando el diseño elegido en el que los traductores están encapsulados en clases, podemos pensar en ampliar nuestro desarrollo para otros lenguajes, creando nuevos traductores para los comandos de ese lenguaje (un ejemplo de esta aplicación son los programas batch de SFB escritos en pro-Cobol).

# GLOSARIO

**AB Suite (Agile Business Suite)**

Siguiente generación de LINC, que incluye un ambiente de desarrollo integrado en Visual Studio, y genera únicamente código .NET para servidores Windows con SQL-Server o código Cobol en caso de utilizar hardware y software propietario de Unisys.

**Business Segment**

Se denomina de esta manera a una aplicación EAE, compuesta por los diferentes objetos que se utilizan para representar las entidades o actividades que componen el negocio.

**ClearPath**

Servidores propietarios de Unisys Corporation.

**Data Items**

Se denominan así a los campos pertenecientes a una tabla definida en EAD, o a los que definen la interfaz con un usuario en una transacción del sistema.

**DMSII (Data Management System II)**

Base de datos propietaria de Unisys Corporation.

**EA Builder (Enterprise Application Builder)**

Componente de EAE que genera y compila código Cobol para la plataforma seleccionada, a partir del código LINC desarrollado en EAD.

**EA Runtime (Enterprise Application Runtime)**

Módulo de EAE encargado de la infraestructura necesaria para la ejecución de un sistema LINC en la plataforma seleccionada.

**EAD (Enterprise Application Developer)**

Componente de EAE que consiste en un editor del lenguaje de programación LINC en un entorno Windows.

**EAE (Enterprise Application Environment)**

Segunda generación de LINC, compuesta por un conjunto de herramientas que incluyen un editor para un ambiente Windows (Developer), un generador y compilador de código Cobol a partir de código LINC (Builder), y un conjunto de programas que dan la infraestructura necesaria para la ejecución de un sistema LINC en producción (Runtime).

**Global Logics**

Las lógicas globales son segmentos de código que pertenecen a un segmento de negocio, y pueden ser invocadas desde los diferentes objetos pertenecientes a dicho segmento de negocio.

**GSD (Global Setup Data Items)**

Variables globales a las que es posible acceder desde los diferentes componentes de EAD, y se utilizan para el pasaje de parámetros en las lógicas globales.

**Ispec (Interface Specification)**

Objeto EAD que permite modelar una entidad o actividad del negocio, mediante la definición de una interfaz con el usuario, lógica de negocio aplicada, y estructuras de datos asociadas.

**LDA (LINC Development Assistant)**

Editor del lenguaje LINC para un ambiente Windows.

**LDL (LINC Development Language)**

Especificación del lenguaje de programación LINC.

**LDL+ (LINC Development Language extension)**

Lenguaje de programación de ABSuite, extensión de LDL, orientado a objetos.

**LINC (Logic and Information Network Compiler)**

Lenguaje de programación de 4ta generación, propietario de Unisys Corporation.

**MCP (Master Control Program)**

Primer sistema operativo escrito en lenguaje de alto nivel, propietario de Unisys Corporation.

**MDL (Model File)**

Extensión de archivos generados desde EAD, de texto plano y formato particular, que contienen toda la información definida en un repositorio.

**Model**

Elemento principal de un sistema EAE, que consiste en un repositorio que contiene una o más aplicaciones LINC, denominadas Business Segments.

**Ordinates**

Campos de los perfiles que definen los campos de la tabla que componen el índice definido en la base de datos, y su orden dentro del mismo.

**PMI (Project Management Institute)**

Organización internacional que define estándares para la gestión de proyectos.

**PMBOK (Project Management Body of Knowledge)**

Guía desarrollada por PMI que contiene una descripción general de los fundamentos de Gestión de Proyectos reconocidos como buenas prácticas.

**Profiles**

Elemento de EAD que define un índice en una tabla de la base de datos.

**SFB (Sistema Financiero Bancario)**

Aplicación central del BROU, que incluye todas las transacciones administrativas y financieras de la operativa del banco a nivel nacional.

**SOA (Service Oriented Architecture)**

Concepto de Arquitectura de Software que define la utilización de servicios para dar soporte a los requisitos de negocio.

**SR (Synchronized Refinement)**

Técnica de lectura de código para generar la descripción de un programa.

# REFERENCIAS


1- **Ingeniería Inversa**

http://es.wikipedia.org/wiki/Ingenier%C3%ADa_inversa

22/02/2011

2- **Reglas de Negocio**

http://es.wikipedia.org/wiki/Reglas_de_negocio

22/02/2011

3- **Softwaremining**

http://www.softwaremining.com/

22/02/2011

4- **Art in Soft**

http://www.artinsoft.com/

22/02/2011

5- **Baltic Technology Group**

http://www.btg.org.lv/

22/02/2011

6- **Agile Business Suite**

https://ecommunity.unisys.com/ecommunity/templates/programs.aspx?cat=ABSuite-20

22/02/2011

7- **Synchronized Refinement**

http://www.cc.gatech.edu/reverse/tutorial/slides/sld001.htm

22/02/2011

8- **Semantics of Business Vocabulary and Business Rules (SBVR)**

http://www.omg.org/spec/SBVR/1.0/

22/02/2011

**9- Enterprise Application Environment**

https://ecommunity.unisys.com/ecommunity/templates/programs.aspx?cat=EAE
Enterprise Application Environment Release 3.3

22/02/2011

Documentación del producto

**10- Java Enterprise Edition**

http://download.oracle.com/javaee/

22/02/2011

**11- Java Persistence API**

http://es.wikipedia.org/wiki/Java_Persistence_API

http://www.oracle.com/technetwork/articles/javaee/jpa-137156.html

22/02/2011

**12- NetBeans**

http://es.wikipedia.org/wiki/NetBeans

http://netbeans.org/features/index.html

22/02/2011

**13- GlassFish**

http://es.wikipedia.org/wiki/GlassFish

http://glassfish.java.net/

22/02/2011

**14- MySQL**

http://www.mysql.com/

22/02/2011

**15- Teoría de Compiladores**

**Compiladores: Principios, técnicas y herramientas**
Alfred V. Aho – Ravi Sethi – Jeffrey D. Ullman
1990
Pearson Education
Addison Wesley Longman

**16- PMI**

http://www.pmi.org/

22/02/2011

**17- WBS**

http://excelza.blogspot.com/2010/03/uso-practico-de-la-wbs.html

22/02/2011

**18 – Tortoise SVN**

http://tortoisesvn.tigris.org/

20/02/2011

**19 – Junit**

http://www.junit.org/

22/02/2011

**20 – Taller de Verificación de Software**

http://www.fing.edu.uy/inco/cursos/tvs/pmwiki/field.php?n=Curso.Curso

22/02/2011

# Extracción de Reglas de Negocio de Código Fuente

Estudiantes:
Martín Ale
Pablo Claps
Pablo Pirán

Tutora:
Profesora Ing. Cristina Mayr

# Anexos

# TABLA DE CONTENIDO

# 1 – Estándar de Documentación

## 1.1 – Propósito

Establecer un estándar para la elaboración de documentos del proyecto de grado P2010_001 (Extracción de Reglas de Negocio de Código Fuente).

## 1.2 – Alcance

Todos los documentos generados por los procesos y actividades del proyecto.

## 1.3 – Definiciones

Todos los documentos serán elaborados en hoja de tamaño A4, orientación vertical, en una sola columna, con márgenes superior e inferior de 3 centímetros, y márgenes izquierdo y derecho de 2,5 centímetros.
Se utilizará la plantilla Plantilla_PG.dot, que tiene definidos los siguientes estilos:

- Texto Normal, letra Arial de tamaño 11, Normal
- Título 5, letra Arial de tamaño 12, Negrita
- Título 4, letra Arial de tamaño 13, Negrita
- Título 3, letra Arial de tamaño 14, Negrita
- Título 2, letra Arial de tamaño 15, Negrita
- Título 1, letra Arial de tamaño 16, Negrita

# 2 - Definición de Alcance del Proyecto

## 2.1 - Propósito

Establecer el Alcance del proyecto de grado P2010_001 (Extracción de Reglas de Negocio de Código Fuente).

## 2.2 - Alcance

Se definirán todos los entregables de cada etapa del proyecto.

## 2.3 - Entregables

El proyecto tendrá los siguientes entregables:

- Programa para Extraer Reglas de Negocio de Código Fuente EAE.
- Informe Final del Proyecto
- Documentos Anexos
- Presentación Final

## 2.4 - Gestión del proyecto

Durante el desarrollo del proyecto se generó documentación necesaria para la correcta gestión del mismo. Tomando como base la metodología PMI se crearon los siguientes archivos:

- Estándar de Documentación
- Diagrama WBS con Diccionario asociado
- Plan de Comunicaciones
- Plan de Calidad
- Plan de Riesgos
- Project

## 2.5 - Extracción de reglas de negocio de código fuente

Se desarrollará un programa Java con las siguientes especificaciones:
La entrada será un archivo con un extracto del sistema central del banco en formato mdl (EAE brinda la posibilidad de generar este tipo de archivos).
Se definirá un set con las principales transacciones y funciones del sistema, en base a reuniones con los usuarios y estadísticas de ejecución en producción.
La salida será, para las transacciones y funciones seleccionadas, un pseudocódigo en lenguaje natural, detallando los pasos que siguen, de manera que pueda ser entendido por alguien con conocimientos de informática y del negocio, pero que no conozca el lenguaje LDL utilizado en la programación de sistemas EAE.
Ver en sección 2.8 el set de transacciones definido.

## 2.6 - Documentación SOA

Se desarrollará un documento Word conteniendo un conjunto de recomendaciones que incluyan los pasos necesarios para llevar el módulo de Banca Electrónica del BROU hacia una Arquitectura Orientada a Servicios (SOA). Sólo se indicarán los pasos sugeridos, no se realizará ninguna implementación en este sentido.

## 2.7 - Documentación y presentación final

Se generará un documento final que incluye el planteo del problema, la solución propuesta y el marco teórico utilizado, el trabajo realizado, las conclusiones y posibles extensiones futuras del trabajo.
Además se hará una presentación en Power Point que resuma todo lo realizado en el proyecto.

## 2.8 - Conjunto de transacciones seleccionado

Se desarrolló un script para extraer estadísticas del funcionamiento del sistema en producción. La única transacción que se destacó por su mayor cantidad de ejecuciones es la extracción de caja de ahorros por cajero automático (AU400).
Todas las transacciones tienen excelentes tiempos de respuesta, estando el 89% de las ejecuciones por debajo de la décima de segundo, y el 96% por debajo de las dos décimas de segundo. El tiempo promedio de respuesta es de apenas 5 centésimas de segundo.
En base a estos datos basamos la selección de las transacciones en la opinión de los especialistas técnicos de los principales módulos.
Se consideraron por separado los módulos de Activas (Préstamos y Garantías), y los módulos de Pasivas (Captación).

**Activas**

Para el Módulo de Préstamos las transacciones seleccionadas fueron:

PPTSO - Alta de Solicitud de Préstamo
PPTLS - Liquidación de Solicitud de Préstamo
PPTCA - Aprobación de Solicitud de Préstamo

Además se analizarán las siguientes lógicas globales:

GP-CTL-EXCEP - Control de las Condiciones de Excepción para Aprobar/Liquidar
GP-CTL-OBLIG - Control de las Condiciones Obligatorias para Aprobar/Liquidar

Para el Módulo de Garantías la transacción seleccionada fue:

NN010 - Alta de Garantía

La elección de estas transacciones se hizo basándose en que son ejecutadas para todos los créditos otorgados por el banco.

**Pasivas**

Para los módulos de captación se seleccionaron las lógicas globales que se utilizan para validar y procesar los débitos o créditos en los diferentes productos del banco.

GP-VAL-DB-CC - Validación Débito en Cuenta Corriente
GP-VAL-DB-CA - Validación Débito en Caja de Ahorros
GP-VAL-DB-PP - Validación Débito en Préstamo
GP-VAL-DB-PF - Validación Débito en Plazo Fijo
GP-VAL-DB-AQ - Validación Débito en Cuentas Varias
GP-VAL-DB-CHQB - Validación Débito Cheque BROU
GP-VAL-DB-CHQC - Validación Débito Cheque Certificado
GP-PRO-DB-CC - Proceso Débito en Cuenta Corriente
GP-PRO-DB-CA - Proceso Débito en Caja de Ahorros
GP-PRO-DB-PP - Proceso Débito en Préstamo
GP-PRO-DB-PF - Proceso Débito en Plazo Fijo
GP-PRO-DB-AQ - Proceso Débito en Cuentas Varias
GP-PRO-DB-CHQB - Proceso Débito Cheque BROU
GP-PRO-DB-CHQC - Proceso Débito Cheque Certificado
GP-VAL-CR-CC - Validación Crédito en Cuenta Corriente
GP-VAL-CR-CA - Validación Crédito en Caja de Ahorros
GP-VAL-CR-PF - Validación Crédito en Plazo Fijo
GP-VAL-CR-AQ - Validación en Crédito Cuentas Varias
GP-PRO-CR-CC - Proceso Crédito en Cuenta Corriente
GP-PRO-CR-CA - Proceso Crédito en Caja de Ahorros
GP-PRO-CR-PF - Proceso Crédito en Plazo Fijo
GP-PRO-CR-AQ - Proceso en Crédito Cuentas Varias
GP-OP-CMB - Validación y proceso de Operación de Cambio

Se seleccionaron estas lógicas porque encapsulan las sentencias necesarias en los procesos de débito y crédito en todos los conceptos de PASIVAS.

Además, por ser la transacción más ejecutada del sistema se seleccionó:

AU400 - Retiro de Caja de Ahorros por Cajero Automático

## 2.9 - Conclusiones

De esta manera nuestro estudio incluirá las principales transacciones de préstamos, además de la transacción de alta de garantías.
Para los módulos de captación serán estudiadas las lógicas que manejan los débitos y créditos en todos los productos del banco.
Por último, se incluye la transacción más ejecutada del sistema, que es el Retiro de Caja de Ahorros por Cajero Automático.

# 3 – Documento de WBS

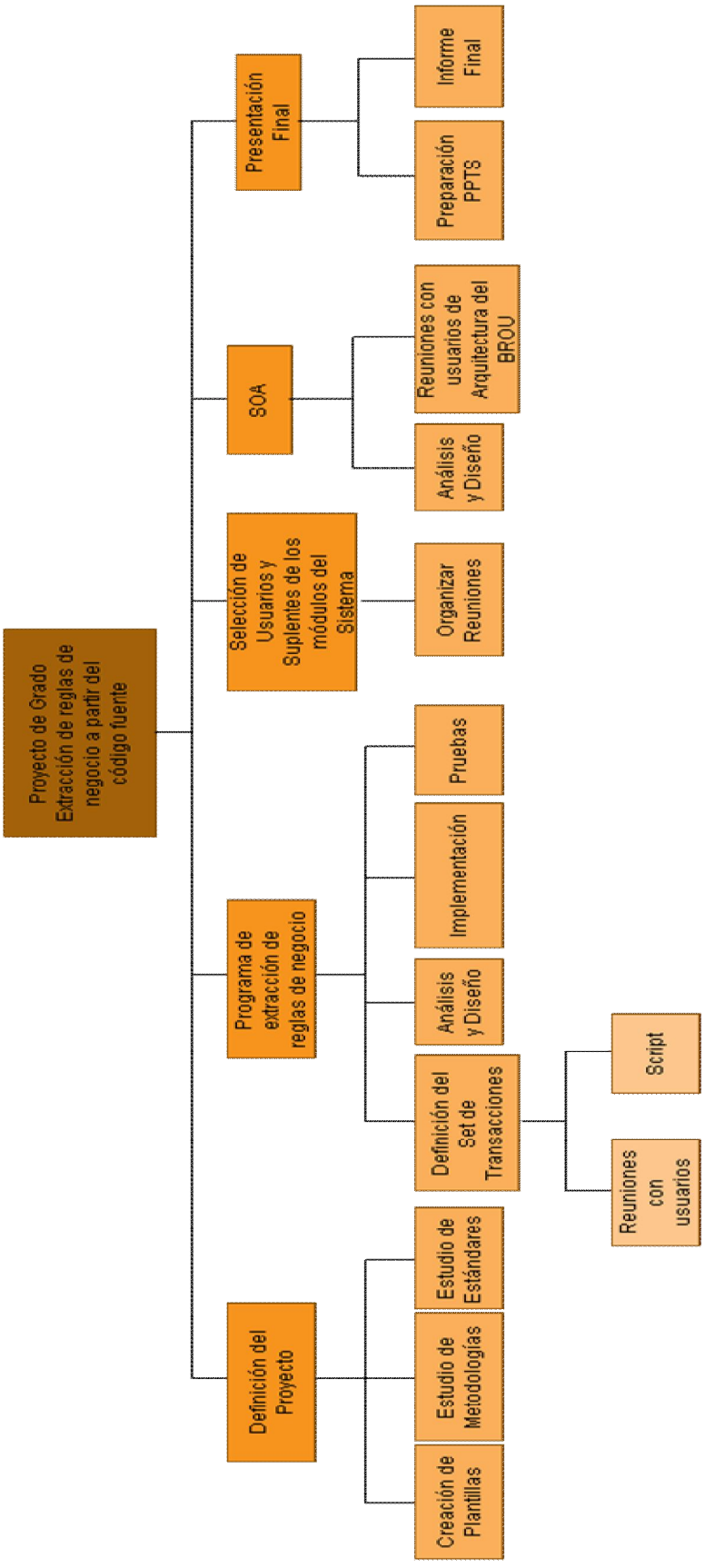
## 3.1 – Propósito

Crear diagrama WBS y definir un diccionario WBS que lo complemente.

## 3.2 – Alcance

Todas las clases de trabajo definidas en el diagrama WBS.

## 3.3 – Diagrama


Proyecto de Grado Extracción de reglas de negocio a partir del código fuente
Definición del Proyecto
Estudio de Estándares
Estudio de Metodologías
Creación de Plantillas
Programa de extracción de reglas de negocio
Pruebas
Implementación
Análisis y Diseño
Definición del Set de Transacciones
Script
Reuniones con usuarios
Selección de Usuarios y Suplentes de los módulos del Sistema
Organizar Reuniones
SOA
Reuniones con usuarios de Arquitectura del BROU
Análisis y Diseño
Presentación Final
Informe Final
Preparación PPTS

## 3.4 – Diccionario

| **Identificación** | **1.1** | Creación de Plantillas | |
|---|---|---|---|
| **Responsable** | Martín Ale, Pablo Claps, Pablo Pirán | **Fecha de inicio** | 15/04/2010 |
| **Participantes** | Martín Ale, Pablo Claps, Pablo Pirán | **Duración estimada** | 10 días |
| **Objetivos** | | | |
| Definir las plantillas de los documentos que se van a utilizar en el Proyecto<br>(Minutas de reunión, seguimiento, documentos de análisis, diseño, riesgos, planificación) | | | |
| **Entregables** | | | |
| **Fecha** | **Descripción** | | **Responsable** |
| | Minutas de Reunión y seguimiento | | Martín Ale |
| | Documento de Riesgos | | Pablo Pirán |
| | Project | | Pablo Claps |
| | WBS | | Pablo Claps |
| | Documento de Análisis | | Pablo Pirán |

| **Identificación** | **1.2** | Estudio de Metodologías | |
|---|---|---|---|
| **Responsable** | Martín Ale, Pablo Claps, Pablo Pirán | **Fecha de inicio** | 15/04/2010 |
| **Participantes** | Martín Ale, Pablo Claps, Pablo Pirán | **Duración estimada** | 9 días |
| **Objetivos** | | | |
| Definir las metodologías que se van a utilizar en el Proyecto<br>(Se tomó como base de estudio PMI y metodologías ágiles) | | | |
| **Entregables** | | | |
| **Fecha** | **Descripción** | | **Responsable** |
| | | | |

| **Identificación** | **1.3** | Estudio de Estándares | |
|---|---|---|---|
| **Responsable** | Martín Ale, Pablo Claps, Pablo Pirán | **Fecha de inicio** | 15/04/2010 |
| **Participantes** | Martín Ale, Pablo Claps, Pablo Pirán | **Duración estimada** | 3 días |
| **Objetivos** | | | |
| Definir los estándares que se van a utilizar en el Proyecto | | | |
| **Entregables** | | | |
| **Fecha** | **Descripción** | | **Responsable** |
| | Estándar de documentación | | Martín Ale |

| **Identificación** | **2.1** | Definición del Set de transacciones | |
|---|---|---|---|
| **Responsable** | Martín Ale | **Fecha de inicio** | 29/04/2010 |
| **Participantes** | Martín Ale, Pablo Claps, Pablo Pirán | **Duración estimada** | 30 días |
| **Objetivos** | | | |
| Crear un script que obtenga los cabezales de las transacciones para luego evaluar las transacciones más usadas | | | |
| **Entregables** | | | |
| **Fecha** | **Descripción** | | **Responsable** |
| | Documento de Análisis y Diseño | | Martín Ale |

| **Identificación** | **2.2** | Análisis y Diseño de Programa | |
|---|---|---|---|
| **Responsable** | Martín Ale, Pablo Claps, Pablo Pirán | **Fecha de inicio** | 17/05/2010 |
| **Participantes** | Martín Ale, Pablo Claps, Pablo Pirán | **Duración estimada** | 22 días |
| **Objetivos** | | | |
| Análisis y Diseño de un programa que extraiga las reglas de negocio en base al código fuente de una aplicación EAE. | | | |
| **Entregables** | | | |
| **Fecha** | **Descripción** | | **Responsable** |
| | Documento de Análisis y Diseño | | Martín Ale, Pablo Pirán, Pablo Claps |
| | Entregable Aceptación Alcance | | Martín Ale, Pablo Pirán, Pablo Claps |

| **Identificación** | **2.3** | Implementación de Programa | |
|---|---|---|---|
| **Responsable** | Martín Ale, Pablo Claps, Pablo Pirán | **Fecha de inicio** | 21/06/2010 |
| **Participantes** | Martín Ale, Pablo Claps, Pablo Pirán | **Duración estimada** | 167 días |
| **Objetivos** | | | |
| Desarrollar un programa que extraiga las reglas de negocio en base al código fuente. Se realizarán entregas incrementales. | | | |
| **Entregables** | | | |
| **Fecha** | **Descripción** | | **Responsable** |
| | Entregable Diseño e Implementación | | Martín Ale |
| | Manual de Usuario | | Pablo Claps |

| **Identificación** | **2.4** | Pruebas | |
|---|---|---|---|
| **Responsable** | **Pablo Claps** | **Fecha de inicio** | **12/07/2010** |
| **Participantes** | Martín Ale, Pablo Claps, Pablo Pirán | **Duración estimada** | 75 días |
| **Objetivos** | | | |
| Documentar el programa desarrollado y mostrar la estrategia de pruebas utilizada. | | | |
| **Entregables** | | | |
| **Fecha** | **Descripción** | | **Responsable** |
| | Documentación de Casos de Pruebas | | Martín Ale |

| **Identificación** | **3.1** | Selección de usuarios y suplentes de cada módulo | |
|---|---|---|---|
| **Responsable** | Pablo Pirán | **Fecha de inicio** | 15/04/2010 |
| **Participantes** | Martín Ale, Pablo Claps, Pablo Pirán | **Duración estimada** | 1 día |
| **Objetivos** | | | |
| Seleccionar los usuarios y suplentes de los módulos del sistema, y una agenda para las reuniones | | | |
| **Entregables** | | | |
| **Fecha** | **Descripción** | | **Responsable** |
| | | | |

| **Identificación** | **4.1** | Análisis y Diseño SOA | |
|---|---|---|---|
| **Responsable** | Martín Ale | **Fecha de inicio** | 26/11/2010 |
| **Participantes** | Martín Ale, Pablo Claps, Pablo Pirán | **Duración estimada** | 30 días |
| **Objetivos** | | | |
| Análisis y Diseño de cómo llevar el módulo de banca electrónica a SOA | | | |
| **Entregables** | | | |
| **Fecha** | **Descripción** | | **Responsable** |
| | Entregable SOA | | Martín Ale |

| **Identificación** | **4.2** | Reuniones con usuarios de Arquitectura del BROU | |
|---|---|---|---|
| **Responsable** | Martín Ale, Pablo Claps, Pablo Pirán | **Fecha de inicio** | 03/09/2010 |
| **Participantes** | Martín Ale, Pablo Claps, Pablo Pirán | **Duración estimada** | 5 días |
| **Objetivos** | | | |
| Plantear a los usuarios el análisis de cómo llevar a cabo la transformación del módulo de banca electrónica a SOA | | | |
| **Entregables** | | | |
| **Fecha** | **Descripción** | | **Responsable** |
| | Minutas de reunión con los usuarios de arquitectura | | Pablo Pirán |

| **Identificación** | **5.1** | **Informe Final** | |
|---|---|---|---|
| **Responsable** | Martín Ale, Pablo Claps, Pablo Pirán | **Fecha de inicio** | 24/01/2010 |
| **Participantes** | Martín Ale, Pablo Claps, Pablo Pirán | **Duración estimada** | 25 días |
| **Objetivos** | | | |
| Creación del informe final | | | |
| **Entregables** | | | |
| **Fecha** | **Descripción** | | **Responsable** |
| | informe final | | Martín Ale, Pablo Claps, Pablo Pirán |

| **Identificación** | **5.2** | Creación de PPT | |
|---|---|---|---|
| **Responsable** | Martín Ale | **Fecha de inicio** | 28/02/2010 |
| **Participantes** | Martín Ale, Pablo Claps, Pablo Pirán | **Duración estimada** | 11 días |
| **Objetivos** | | | |
| Creación de Presentación Power Point. | | | |
| **Entregables** | | | |
| **Fecha** | **Descripción** | | **Responsable** |
| | Presentación Final | | Martín Ale |

# 4 - Plan de Comunicaciones

## 4.1 - Propósito

Establecer el plan para las comunicaciones que deben ser llevadas a cabo por el equipo del proyecto con el objetivo de que los involucrados en alguna actividad del mismo estén debidamente informados de todas las tareas que se van realizando.

## 4.2 - Alcance

Toda la información relevante y que debe ser conocida por todos los participantes del proyecto.

## 4.3 - Descripción del procedimiento

| | Descripción | Responsable | Periodicidad | Destinatarios |
|---|---|---|---|---|
| 1 | Informar al tutor del proyecto acerca del avance del mismo. Se hará en la reunión semanal de seguimiento. | Estudiantes | Semanal | Docente Tutor. |
| 2 | Informar a los usuarios responsables sobre el avance del proyecto. Se enviará vía correo electrónico. | Estudiantes | Mensual | Para: Tutor, Usuarios. Asunto: Informe Mensual. |
| 3 | Informar sobre cambios relevantes para el proyecto. En caso de surgir cambios se elaborará un documento que será enviado por correo electrónico. | Estudiantes | - | Para: Tutor, Usuarios. Asunto: Informe Cambios. |
| 4 | Intercambio de información y opiniones durante la elaboración del documento de alcance del proyecto. Se hará en reuniones y por correo electrónico. | Estudiantes Usuarios | Semanal | Para: Usuarios, Estudiantes. Asunto: Información sobre req's y solución propuesta. |

## 4.4 – Comunicación interna

Se creó un grupo de correo en googlegroups, para facilitar el intercambio de mails, y con la funcionalidad adicional de poder almacenar documentos compartidos.

# 5 - Plan de Riesgos

## 5.1 – Propósito

Definir el plan de riesgos para el proyecto de grado P2010_001 (Extracción de Reglas de Negocio de Código Fuente).

## 5.2 – Alcance

Todos los riesgos que puedan surgir en las diferentes etapas del proyecto.

## 5.3 – Descripción del procedimiento

Se utilizó una planilla Excel que define los diferentes puntos a considerar. Mensualmente se hizo una revisión de la planilla, buscando detectar nuevos riesgos o anticipar la ocurrencia de alguno.

### 5.3.1 – Identificación del riesgo

| Identificación del Riesgo | | | |
|---|---|---|---|
| Identificación del Riesgo | Nombre del Riesgo | Descripción del Riesgo | Fecha de identificación del riesgo |
| 1 | | | |

### 5.3.2 – Evaluación y análisis

| Evaluación y análisis | | |
|---|---|---|
| Consecuencia | Probabilidad | Nivel de riesgo |
| | | |

### 5.3.3 – Controles existentes

| Controles existentes | | |
|---|---|---|
| Acciones actuales para manejar el riesgo | Tarea en WBS | **Riesgo Aceptable** (Si o No) |
| | | |

### 5.3.4 – Otras acciones

| Otras acciones | | | |
|---|---|---|---|
| **Prevención** | **Reducción** | **Mitigación** | **Contingencia** |
| | | | |

### 5.3.5 – Asignación

| **Asignado a** |
|---|
| |

## 5.4 – Métricas utilizadas

### 5.4.1 – Nivel de riesgo

| **Nivel de Riesgo** | |
|---|---|
| MA | Muy Alto |
| A | Alto |
| M | Medio |
| B | Bajo |

## 5.4.2 - Consecuencias

| | |
|---|---|
| **Consecuencias:** | 1=insignificante |
| | 2=menor |
| | 3=moderada |
| | 4=mayor |
| | 5=catastrófica |

## 5.4.3 – Probabilidad

| | | |
|---|---|---|
| **Probabilidad** | **A -** | La consecuencia es casi seguro que se produzca en la mayoría de las circunstancias |
| | **B -** | La consecuencia es probable que se produzcan con frecuencia |
| | **C -** | Es posibles y probable que la consecuencia que se produzca en algún momento |
| | **D -** | La consecuencia es poco probable que ocurra, pero puede suceder |
| | **E -** | La consecuencia puede ocurrir, pero sólo en circunstancias excepcionales |

## 5.5 – Criterio de evaluación de riesgos

| | | | Criterios de consecuencias | | | | |
|---|---|---|---|---|---|---|---|
| | | | 1 – Insignificante | 2 – Menor | 3 – Moderada | 4 – Mayor | 5 – Catastrófica |
| Probabilidad | A - | La consecuencia es casi seguro que se produzca en la mayoría de las circunstancias | Medio (M) | Alto (A) | Alto (A) | Muy Alto (MA) | Muy Alto (MA) |
| | B - | La consecuencia es probable que se produzcan con frecuencia | Medio (M) | Medio (M) | Alto (A) | Alto (A) | Muy Alto (MA) |
| | C - | Es posibles y probable que la consecuencia que se produzca en algún momento | Bajo (B) | Medio (M) | Alto (A) | Alto (A) | Alto (A) |
| | D - | La consecuencia es poco probable que ocurra, pero puede suceder | Bajo (B) | Bajo (B) | Medio (M) | Medio (M) | Alto (A) |
| | E - | La consecuencia puede ocurrir, pero sólo en circunstancias excepcionales | Bajo (B) | Bajo (B) | Medio (M) | Medio (M) | Alto (A) |

| Criterios de consecuencias | Límites entre consecuencias |
|---|---|
| Catastrófica | Extremadamente importante afectación a: 1) Reputación 2) Costos 3) Cronograma 4) Calidad del producto |
| Mayor | Importante Afectación a: 1) Reputación 2) Costos 3) Cronograma 4) Calidad del producto |
| Moderada | Moderada Afectación a: 1) Reputación 2) Costos 3) Cronograma 4) Calidad del producto |
| Menor | Baja Afectación a: 1) Reputación 2) Costos 3) Cronograma 4) Calidad del producto |
| Insignificante | Insignificante Afectación a: 1) Reputación 2) Costos 3) Cronograma 4) Calidad del producto |

## 5.6 – Planilla de riesgos

**Registro de Riesgos y Plan de Acción**

| Identificación del Riesgo | Identificación del Riesgo | | Evaluación y análisis | | | Controles existentes | | Otras acciones | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Nombre del Riesgo | Descripción del Riesgo | Fecha de identificación del riesgo | Consecuencia | Probabilidad | Nivel de riesgo | Acciones actuales para manejar el riesgo | Riesgo Aceptable (Si o No) | Tarea en WBS | Prevención | Reducción | Mitigación | Contingencia | Asignado a |
| 1 | Disponibilidad de usuarios responsables | Retrasos en el proyecto por no contar con disponibilidad de los usuarios responsables | 19/05/2010 | 3 | D | M | Informar e involucrar a los usuarios | SI | | NA | Mantener el interés de los usuarios mediante informes y mostrando avances que les interesen. | Avanzar en temas que no requieran aprobación de los usuarios. | Encontrar usuarios con mayor disponibilidad. | Martin / Pablo |
| | Problemas técnicos | Dificultades técnicas que requieran soporte de difícil acceso. | 19/05/2010 | 3 | C | A | Investigar y documentar los eventuales problemas que | SI | | NA | Capacitación previa en las herramientas a utilizar. | Contar con recursos técnicos que puedan dar soporte. | Consultar recursos técnicos que puedan dar soporte | Martin / Pablo |
| 2 | Disponibilidad de Estudiantes | Ausencia de estudiantes por motivos ajenos al proyecto (laborales, salud) | 20/09/2010 | 4 | C | A | Tener en cuenta el riesgo al momento de realizar estimaciones | SI | | NA | NA | En caso de viajes por trabajo estudiar la posibilidad de trabajar de forma remota para el proyecto. | Reveer la fecha de finalización del Proyecto | Martin / Pablo / Pablo |
| 3 | | | | | | | | | | | | | | |

| Consecuencias: | |
|---|---|
| 1=insignificante | |
| 2=menor | |
| 3=moderada | |
| 4=mayor | |
| 5=catastrófica | |

| | Probabilidad |
|---|---|
| A - | La consecuencia es casi seguro que se produzca en la mayoría de las circunstancias |
| B - | La consecuencia es probable que se produzcan con frecuencia |
| C - | Es posibles y probable que la consecuencia que se produzca en algún momento |
| D - | La consecuencia es poco probable que ocurra, pero puede suceder |
| E - | La consecuencia puede ocurrir, pero sólo en circunstancias excepcionales |

| Nivel de Riesgo | |
|---|---|
| MA | Muy Alto |
| A | Alto |
| M | Medio |
| B | Bajo |

# 6 – Plan de Calidad

## 6.1 – Propósito

Establecer un plan de calidad para el proyecto de grado P2010_001 (Extracción de Reglas de Negocio de Código Fuente).

El aseguramiento de calidad se consigue a través de la planificación del seguimiento de la calidad durante el proyecto indicando para cada una de las actividades cuales son los atributos de calidad a tener en cuenta y quienes son los responsables.

## 6.2 – Alcance

Se cubrirán las siguientes etapas del ciclo de vida:

- Análisis y Requerimientos
- Diseño
- Implementación
- Verificación

- Requerimientos
    - Alcance del Sistema
    - Modelo de Dominio

- Diseño
    - Modelo de Diseño
    - Descripción de la Arquitectura

- Implementación
    - Prototipo JEE

- Verificación
    - Casos de Prueba

- Gestión de Proyecto
    - Project
    - WBS
    - Documento de Riesgos
    - Documento de Casos de Prueba
    - Informe Final de Proyecto

## 6.3 – Requerimientos de calidad

Acá se explican los requerimientos que aseguran la satisfacción del cliente.
Estos requerimientos deberían ser determinados por los usuarios de las distintas áreas de negocio.

Se detectaron los siguientes requerimientos de Calidad:

- Facilidad de Uso del Sistema.
- Obtener una salida de pseudocódigo en lenguaje natural que sea interpretable fácilmente por el usuario.

Facilidad de Uso es el esfuerzo requerido para entender, aprender y usar.
Es la capacidad del software para ser atractivo al usuario.

## 6.4 – Organización y administración de la calidad

Definimos los siguientes roles:

- Administrador de la Calidad del Proyecto
  - Pablo Claps

- Administrador de la Integración
  - Pablo Pirán

- Administrador de Configuración
  - Martín Ale

## 6.5 – Aseguramiento de procesos de calidad

Revisiones internas del Proyecto:

Se revisarán los entregables, en cada cambio de versión y/ó mensualmente.

Revisiones por parte de los Usuarios:

Se revisarán los entregables cuando sea relevante.

## 6.6 – Estándar de documentación

Se definió un estándar de documentación de manera que todos los documentos tengan el mismo formato (ver anexo Estándar de Documentación).

## 6.7 – Estrategia de pruebas

Toda técnica de pruebas debe tener en cuenta los siguientes cuatro puntos:

1. Modelo de la realidad
2. Técnica de selección
3. Criterio de Cubrimiento
4. Teoría de Errores

La planificación de las pruebas es una tarea que debe ser tenida en cuenta desde el inicio del ciclo de desarrollo de software. El objetivo de la verificación del software es descubrir faltas para evitar fallas, y así poder evaluar la calidad del producto.

Existen diferentes técnicas de selección de casos de prueba. Aquí aplicamos una técnica de caja blanca: partición en clases de equivalencias.
Si bien la elección es arbitraria, y la técnica no tiene por qué ser única a lo largo del proceso de verificación del software, se estimó que esta era la más adecuada dada la realidad planteada

Además se realizarán revisiones de a pares a lo largo del desarrollo del prototipo.

## 6.8 – Control de reparación de defectos

Los defectos encontrados en los distintos documentos serán corregidos por el autor del documento.
Luego de ser corregidos los mismos deberán ser controlados por el responsable de la calidad.

## 6.9 – Administración de la configuración

### 6.9.1 – Ambiente controlado

Se definieron dos repositorios uno para documentos y otro para el Código Fuente.
Ambos repositorios fueron creados en un servidor del Inco que brinda un servicio de Ambiente Controlado SubVersion (Controlador de Versiones).
Localmente se utlilizó la herramienta Tortoise SVN, de uso gratuito.

Para el repositorio de Documentos se creó una estructura de carpetas con la siguiente forma:

Documentos
- -Entregas
- -Diagramas
- -Gestión de Proyecto

### 6.9.2 – Línea base

Una línea base se define como un producto que acaba de ser aprobado y que define la “base” de ese producto que para ser modificado deberá pasar por un protocolo de control de cambios. También puede verse como un punto de referencia en la configuración de un proyecto que marca un estado estable en algún producto del proyecto.

Inicialmente se creó el repositorio de Código Fuente, que consistió en la primer versión del Proyecto JEE en NetBeans, el Servidor GlassFish y la base de datos MySQL.

La primer línea base quedó completada al finalizar la Segunda Etapa de Desarrollo (Carga de Modelo de Sistema)

La segunda línea base quedó completada al finalizar la Tercer Etapa de Desarrollo (Construcción de Traductores)

## 6.10 – Administración de la integración

No aplica realizar integración entre módulos, debido a que se realizó una implementación iterativa incremental, donde no fue necesario integrar distintos módulos.

## 6.11 – Manual de usuario

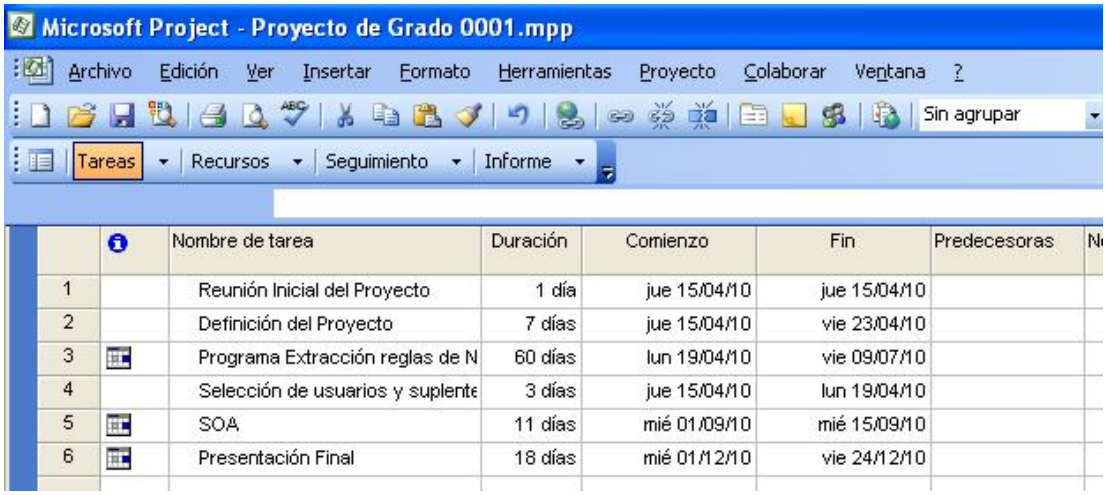
La documentación de usuario debe especificar y describir los datos y entradas de control requeridos, así como la secuencia de entradas, opciones, limitaciones de programa y otros elementos necesarios para la ejecución exitosa del software.

Todos los errores deben ser identificados y las acciones correctivas descritas.

Como resultado del proyecto el cliente obtendrá una documentación para el usuario de acuerdo a los requerimientos específicos del proyecto.

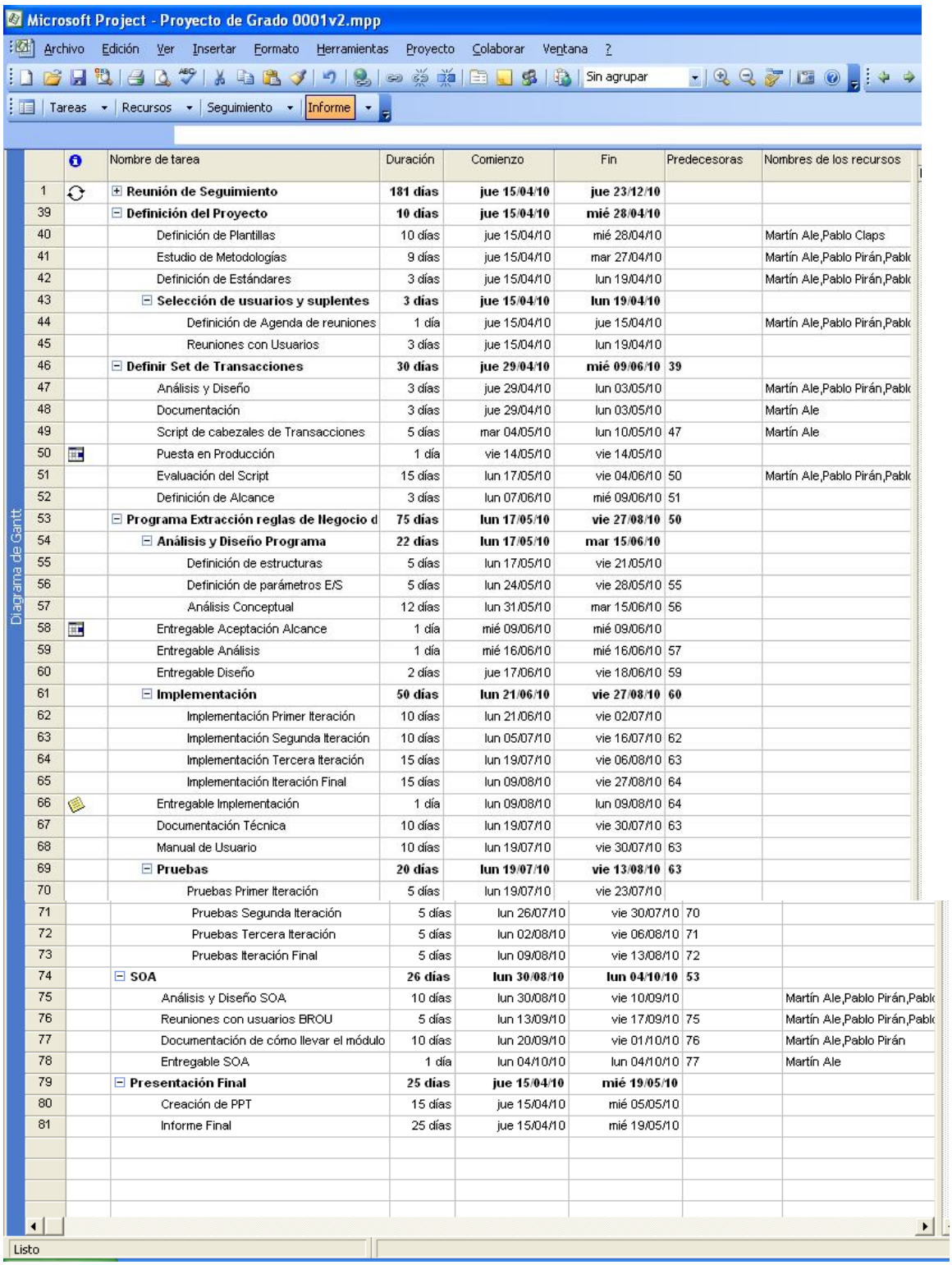
# 7 - Project

## 7.1 – Versión 12/05/2010



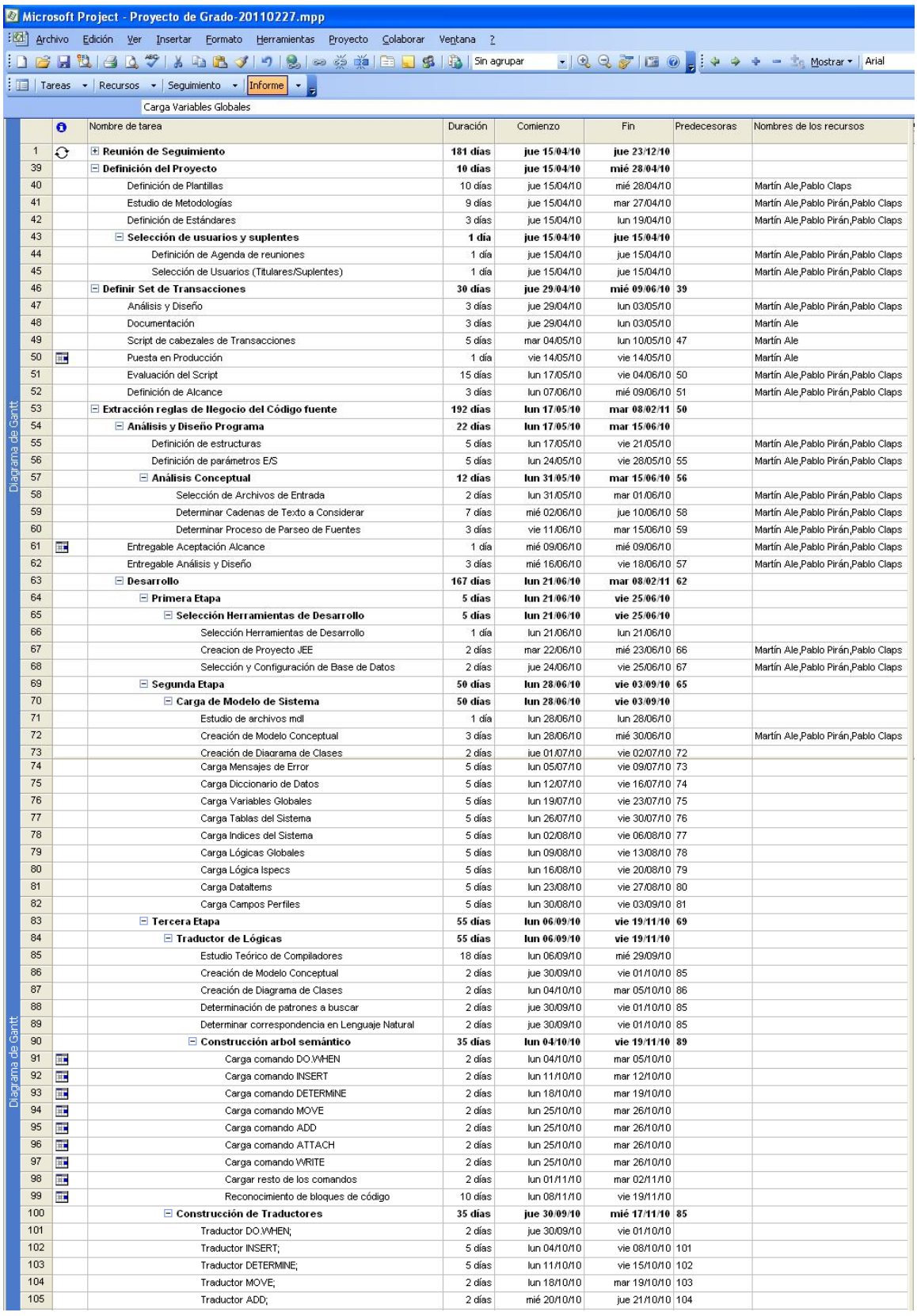

| | Nombre de tarea | Duración | Comienzo | Fin | Predecesoras |
|---|---|---|---|---|---|
| 1 | Reunión Inicial del Proyecto | 1 día | jue 15/04/10 | jue 15/04/10 | |
| 2 | Definición del Proyecto | 7 días | jue 15/04/10 | vie 23/04/10 | |
| 3 | Programa Extracción reglas de N | 60 días | lun 19/04/10 | vie 09/07/10 | |
| 4 | Selección de usuarios y suplente | 3 días | jue 15/04/10 | lun 19/04/10 | |
| 5 | SOA | 11 días | mié 01/09/10 | mié 15/09/10 | |
| 6 | Presentación Final | 18 días | mié 01/12/10 | vie 24/12/10 | |

## 7.2 – Versión 09/06/2010




| | Nombre de tarea | Duración | Comienzo | Fin | Predecesoras | Nombres de los recursos |
|---|---|---|---|---|---|---|
| 1 | **Reunión de Seguimiento** | **181 días** | **jue 15/04/10** | **jue 23/12/10** | | |
| 39 | **Definición del Proyecto** | **10 días** | **jue 15/04/10** | **mié 28/04/10** | | |
| 40 | Definición de Plantillas | 10 días | jue 15/04/10 | mié 28/04/10 | | Martín Ale,Pablo Claps |
| 41 | Estudio de Metodologías | 9 días | jue 15/04/10 | mar 27/04/10 | | Martín Ale,Pablo Pirán,Pabl |
| 42 | Definición de Estándares | 3 días | jue 15/04/10 | lun 19/04/10 | | Martín Ale,Pablo Pirán,Pabl |
| 43 | **Selección de usuarios y suplentes** | **3 días** | **jue 15/04/10** | **lun 19/04/10** | | |
| 44 | Definición de Agenda de reuniones | 1 día | jue 15/04/10 | jue 15/04/10 | | Martín Ale,Pablo Pirán,Pabl |
| 45 | Reuniones con Usuarios | 3 días | jue 15/04/10 | lun 19/04/10 | | |
| 46 | **Definir Set de Transacciones** | **30 días** | **jue 29/04/10** | **mié 09/06/10** | **39** | |
| 47 | Análisis y Diseño | 3 días | jue 29/04/10 | lun 03/05/10 | | Martín Ale,Pablo Pirán,Pabl |
| 48 | Documentación | 3 días | jue 29/04/10 | lun 03/05/10 | | Martín Ale |
| 49 | Script de cabezales de Transacciones | 5 días | mar 04/05/10 | lun 10/05/10 | 47 | Martín Ale |
| 50 | Puesta en Producción | 1 día | vie 14/05/10 | vie 14/05/10 | | |
| 51 | Evaluación del Script | 15 días | lun 17/05/10 | vie 04/06/10 | 50 | Martín Ale,Pablo Pirán,Pabl |
| 52 | Definición de Alcance | 3 días | lun 07/06/10 | mié 09/06/10 | 51 | |
| 53 | **Programa Extracción reglas de Negocio d** | **75 días** | **lun 17/05/10** | **vie 27/08/10** | **50** | |
| 54 | **Análisis y Diseño Programa** | **22 días** | **lun 17/05/10** | **mar 15/06/10** | | |
| 55 | Definición de estructuras | 5 días | lun 17/05/10 | vie 21/05/10 | | |
| 56 | Definición de parámetros E/S | 5 días | lun 24/05/10 | vie 28/05/10 | 55 | |
| 57 | Análisis Conceptual | 12 días | lun 31/05/10 | mar 15/06/10 | 56 | |
| 58 | Entregable Aceptación Alcance | 1 día | mié 09/06/10 | mié 09/06/10 | | |
| 59 | Entregable Análisis | 1 día | mié 16/06/10 | mié 16/06/10 | 57 | |
| 60 | Entregable Diseño | 2 días | jue 17/06/10 | vie 18/06/10 | 59 | |
| 61 | **Implementación** | **50 días** | **lun 21/06/10** | **vie 27/08/10** | **60** | |
| 62 | Implementación Primer Iteración | 10 días | lun 21/06/10 | vie 02/07/10 | | |
| 63 | Implementación Segunda Iteración | 10 días | lun 05/07/10 | vie 16/07/10 | 62 | |
| 64 | Implementación Tercera Iteración | 15 días | lun 19/07/10 | vie 06/08/10 | 63 | |
| 65 | Implementación Iteración Final | 15 días | lun 09/08/10 | vie 27/08/10 | 64 | |
| 66 | Entregable Implementación | 1 día | lun 09/08/10 | lun 09/08/10 | 64 | |
| 67 | Documentación Técnica | 10 días | lun 19/07/10 | vie 30/07/10 | 63 | |
| 68 | Manual de Usuario | 10 días | lun 19/07/10 | vie 30/07/10 | 63 | |
| 69 | **Pruebas** | **20 días** | **lun 19/07/10** | **vie 13/08/10** | **63** | |
| 70 | Pruebas Primer Iteración | 5 días | lun 19/07/10 | vie 23/07/10 | | |
| 71 | Pruebas Segunda Iteración | 5 días | lun 26/07/10 | vie 30/07/10 | 70 | |
| 72 | Pruebas Tercera Iteración | 5 días | lun 02/08/10 | vie 06/08/10 | 71 | |
| 73 | Pruebas Iteración Final | 5 días | lun 09/08/10 | vie 13/08/10 | 72 | |
| 74 | **SOA** | **26 días** | **lun 30/08/10** | **lun 04/10/10** | **53** | |
| 75 | Análisis y Diseño SOA | 10 días | lun 30/08/10 | vie 10/09/10 | | Martín Ale,Pablo Pirán,Pabl |
| 76 | Reuniones con usuarios BROU | 5 días | lun 13/09/10 | vie 17/09/10 | 75 | Martín Ale,Pablo Pirán,Pabl |
| 77 | Documentación de cómo llevar el módulo | 10 días | lun 20/09/10 | vie 01/10/10 | 76 | Martín Ale,Pablo Pirán |
| 78 | Entregable SOA | 1 día | lun 04/10/10 | lun 04/10/10 | 77 | Martín Ale |
| 79 | **Presentación Final** | **25 días** | **jue 15/04/10** | **mié 19/05/10** | | |
| 80 | Creación de PPT | 15 días | jue 15/04/10 | mié 05/05/10 | | |
| 81 | Informe Final | 25 días | jue 15/04/10 | mié 19/05/10 | | |

## 7.3 – Versión 20/02/2011




| | Nombre de tarea | Duración | Comienzo | Fin | Predecesoras | Nombres de los recursos |
|---|---|---|---|---|---|---|
| 1 | **Reunión de Seguimiento** | **181 días** | **jue 15/04/10** | **jue 23/12/10** | | |
| 39 | **Definición del Proyecto** | **10 días** | **jue 15/04/10** | **mié 28/04/10** | | |
| 40 | Definición de Plantillas | 10 días | jue 15/04/10 | mié 28/04/10 | | Martín Ale,Pablo Claps |
| 41 | Estudio de Metodologías | 9 días | jue 15/04/10 | mar 27/04/10 | | Martín Ale,Pablo Pirán,Pablo Claps |
| 42 | Definición de Estándares | 3 días | jue 15/04/10 | lun 19/04/10 | | Martín Ale,Pablo Pirán,Pablo Claps |
| 43 | **Selección de usuarios y suplentes** | **1 día** | **jue 15/04/10** | **jue 15/04/10** | | |
| 44 | Definición de Agenda de reuniones | 1 día | jue 15/04/10 | jue 15/04/10 | | Martín Ale,Pablo Pirán,Pablo Claps |
| 45 | Selección de Usuarios (Titulares/Suplentes) | 1 día | jue 15/04/10 | jue 15/04/10 | | Martín Ale,Pablo Pirán,Pablo Claps |
| 46 | **Definir Set de Transacciones** | **30 días** | **jue 29/04/10** | **mié 09/06/10** | **39** | |
| 47 | Análisis y Diseño | 3 días | jue 29/04/10 | lun 03/05/10 | | Martín Ale,Pablo Pirán,Pablo Claps |
| 48 | Documentación | 3 días | jue 29/04/10 | lun 03/05/10 | | Martín Ale |
| 49 | Script de cabezales de Transacciones | 5 días | mar 04/05/10 | lun 10/05/10 | 47 | Martín Ale |
| 50 | Puesta en Producción | 1 día | vie 14/05/10 | vie 14/05/10 | | Martín Ale |
| 51 | Evaluación del Script | 15 días | lun 17/05/10 | vie 04/06/10 | 50 | Martín Ale,Pablo Pirán,Pablo Claps |
| 52 | Definición de Alcance | 3 días | lun 07/06/10 | mié 09/06/10 | 51 | Martín Ale,Pablo Pirán,Pablo Claps |
| 53 | **Extracción reglas de Negocio del Código fuente** | **192 días** | **lun 17/05/10** | **mar 08/02/11** | **50** | |
| 54 | **Análisis y Diseño Programa** | **22 días** | **lun 17/05/10** | **mar 15/06/10** | | |
| 55 | Definición de estructuras | 5 días | lun 17/05/10 | vie 21/05/10 | | Martín Ale,Pablo Pirán,Pablo Claps |
| 56 | Definición de parámetros E/S | 5 días | lun 24/05/10 | vie 28/05/10 | 55 | Martín Ale,Pablo Pirán,Pablo Claps |
| 57 | **Análisis Conceptual** | **12 días** | **lun 31/05/10** | **mar 15/06/10** | **56** | |
| 58 | Selección de Archivos de Entrada | 2 días | lun 31/05/10 | mar 01/06/10 | | Martín Ale,Pablo Pirán,Pablo Claps |
| 59 | Determinar Cadenas de Texto a Considerar | 7 días | mié 02/06/10 | jue 10/06/10 | 58 | Martín Ale,Pablo Pirán,Pablo Claps |
| 60 | Determinar Proceso de Parseo de Fuentes | 3 días | vie 11/06/10 | mar 15/06/10 | 59 | Martín Ale,Pablo Pirán,Pablo Claps |
| 61 | Entregable Aceptación Alcance | 1 día | mié 09/06/10 | mié 09/06/10 | | Martín Ale,Pablo Pirán,Pablo Claps |
| 62 | Entregable Análisis y Diseño | 3 días | mié 16/06/10 | vie 18/06/10 | 57 | Martín Ale,Pablo Pirán,Pablo Claps |
| 63 | **Desarrollo** | **167 días** | **lun 21/06/10** | **mar 08/02/11** | **62** | |
| 64 | **Primera Etapa** | **5 días** | **lun 21/06/10** | **vie 25/06/10** | | |
| 65 | **Selección Herramientas de Desarrollo** | **5 días** | **lun 21/06/10** | **vie 25/06/10** | | |
| 66 | Selección Herramientas de Desarrollo | 1 día | lun 21/06/10 | lun 21/06/10 | | |
| 67 | Creacion de Proyecto JEE | 2 días | mar 22/06/10 | mié 23/06/10 | 66 | Martín Ale,Pablo Pirán,Pablo Claps |
| 68 | Selección y Configuración de Base de Datos | 2 días | jue 24/06/10 | vie 25/06/10 | 67 | Martín Ale,Pablo Pirán,Pablo Claps |
| 69 | **Segunda Etapa** | **50 días** | **lun 28/06/10** | **vie 03/09/10** | **65** | |
| 70 | **Carga de Modelo de Sistema** | **50 días** | **lun 28/06/10** | **vie 03/09/10** | | |
| 71 | Estudio de archivos mdl | 1 día | lun 28/06/10 | lun 28/06/10 | | |
| 72 | Creación de Modelo Conceptual | 3 días | lun 28/06/10 | mié 30/06/10 | | Martín Ale,Pablo Pirán,Pablo Claps |
| 73 | Creación de Diagrama de Clases | 2 días | jue 01/07/10 | vie 02/07/10 | 72 | |
| 74 | Carga Mensajes de Error | 5 días | lun 05/07/10 | vie 09/07/10 | 73 | |
| 75 | Carga Diccionario de Datos | 5 días | lun 12/07/10 | vie 16/07/10 | 74 | |
| 76 | Carga Variables Globales | 5 días | lun 19/07/10 | vie 23/07/10 | 75 | |
| 77 | Carga Tablas del Sistema | 5 días | lun 26/07/10 | vie 30/07/10 | 76 | |
| 78 | Carga Indices del Sistema | 5 días | lun 02/08/10 | vie 06/08/10 | 77 | |
| 79 | Carga Lógicas Globales | 5 días | lun 09/08/10 | vie 13/08/10 | 78 | |
| 80 | Carga Lógica Ispecs | 5 días | lun 16/08/10 | vie 20/08/10 | 79 | |
| 81 | Carga DataItems | 5 días | lun 23/08/10 | vie 27/08/10 | 80 | |
| 82 | Carga Campos Perfiles | 5 días | lun 30/08/10 | vie 03/09/10 | 81 | |
| 83 | **Tercera Etapa** | **55 días** | **lun 06/09/10** | **vie 19/11/10** | **69** | |
| 84 | **Traductor de Lógicas** | **55 días** | **lun 06/09/10** | **vie 19/11/10** | | |
| 85 | Estudio Teórico de Compiladores | 18 días | lun 06/09/10 | mié 29/09/10 | | |
| 86 | Creación de Modelo Conceptual | 2 días | jue 30/09/10 | vie 01/10/10 | 85 | |
| 87 | Creación de Diagrama de Clases | 2 días | lun 04/10/10 | mar 05/10/10 | 86 | |
| 88 | Determinación de patrones a buscar | 2 días | jue 30/09/10 | vie 01/10/10 | 85 | |
| 89 | Determinar correspondencia en Lenguaje Natural | 2 días | jue 30/09/10 | vie 01/10/10 | 85 | |
| 90 | **Construcción arbol semántico** | **35 días** | **lun 04/10/10** | **vie 19/11/10** | **89** | |
| 91 | Carga comando DO.WHEN | 2 días | lun 04/10/10 | mar 05/10/10 | | |
| 92 | Carga comando INSERT | 2 días | lun 11/10/10 | mar 12/10/10 | | |
| 93 | Carga comando DETERMINE | 2 días | lun 18/10/10 | mar 19/10/10 | | |
| 94 | Carga comando MOVE | 2 días | lun 25/10/10 | mar 26/10/10 | | |
| 95 | Carga comando ADD | 2 días | lun 25/10/10 | mar 26/10/10 | | |
| 96 | Carga comando ATTACH | 2 días | lun 25/10/10 | mar 26/10/10 | | |
| 97 | Carga comando WRITE | 2 días | lun 25/10/10 | mar 26/10/10 | | |
| 98 | Cargar resto de los comandos | 2 días | lun 01/11/10 | mar 02/11/10 | | |
| 99 | Reconocimiento de bloques de código | 10 días | lun 08/11/10 | vie 19/11/10 | | |
| 100 | **Construcción de Traductores** | **35 días** | **jue 30/09/10** | **mié 17/11/10** | **85** | |
| 101 | Traductor DO.WHEN; | 2 días | jue 30/09/10 | vie 01/10/10 | | |
| 102 | Traductor INSERT; | 5 días | lun 04/10/10 | vie 08/10/10 | 101 | |
| 103 | Traductor DETERMINE; | 5 días | lun 11/10/10 | vie 15/10/10 | 102 | |
| 104 | Traductor MOVE; | 2 días | lun 18/10/10 | mar 19/10/10 | 103 | |
| 105 | Traductor ADD; | 2 días | mié 20/10/10 | jue 21/10/10 | 104 | |

Diagrama de Gantt

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 106 | | Traductor ATTACH | 2 días | vie 22/10/10 | lun 25/10/10 | 105 | |
| 107 | | Traductor WRITE; | 2 días | mar 26/10/10 | mié 27/10/10 | 106 | |
| 108 | | Otros Traductores | 5 días | jue 28/10/10 | mié 03/11/10 | 107 | |
| 109 | | Determinar variables de entrada y salida para reconoci | 10 días | jue 04/11/10 | mié 17/11/10 | 108 | |
| 110 | | **Etapa Final** | **22 días** | **lun 10/01/11** | **mar 08/02/11** | **83** | |
| 111 | | **Interfaz Gráfica** | **10 días** | **lun 10/01/11** | **vie 21/01/11** | | |
| 112 | | Definición de IG | 10 días | lun 10/01/11 | vie 21/01/11 | | |
| 113 | | Construcción de Salida en formato Html | 5 días | lun 17/01/11 | vie 21/01/11 | | |
| 114 | | **Entregables** | **12 días** | **lun 24/01/11** | **mar 08/02/11** | | |
| 115 | | Entregable Diseño e Implementación | 9 días | lun 24/01/11 | jue 03/02/11 | | Pablo Claps,Pablo Pirán |
| 116 | | Manual de Usuario | 3 días | vie 04/02/11 | mar 08/02/11 | 115 | Martín Ale |
| 117 | | **Pruebas** | **150 días** | **lun 12/07/10** | **vie 04/02/11** | | |
| 118 | | **Segunda Etapa** | **45 días** | **lun 12/07/10** | **vie 10/09/10** | | |
| 119 | | Prueba Carga Mensajes de Error | 5 días | lun 12/07/10 | vie 16/07/10 | 74 | |
| 120 | | Prueba Carga Diccionario de Datos | 5 días | lun 19/07/10 | vie 23/07/10 | 75 | |
| 121 | | Prueba Carga Variables Globales | 5 días | lun 26/07/10 | vie 30/07/10 | 76 | |
| 122 | | Prueba Carga Tablas del Sistema | 5 días | lun 02/08/10 | vie 06/08/10 | 77 | |
| 123 | | Prueba Carga Indices del Sistema | 5 días | lun 09/08/10 | vie 13/08/10 | 78 | |
| 124 | | Prueba Carga Lógicas Globales | 5 días | lun 16/08/10 | vie 20/08/10 | 79 | |
| 125 | | Prueba Carga Lógica Ispecs | 5 días | lun 23/08/10 | vie 27/08/10 | 80 | |
| 126 | | Prueba Carga DataItems | 5 días | lun 30/08/10 | vie 03/09/10 | 81 | |
| 127 | | Prueba Carga Campos Perfiles | 5 días | lun 06/09/10 | vie 10/09/10 | 82 | |
| 128 | | **Tercer Etapa** | **20 días** | **lun 04/10/10** | **vie 29/10/10** | | |
| 129 | | Prueba Traductor DO.WHEN; | 2 días | lun 04/10/10 | mar 05/10/10 | 101 | |
| 130 | | Prueba Traductor INSERT; | 5 días | lun 11/10/10 | vie 15/10/10 | 102 | |
| 131 | | Prueba Traductor DETERMINE; | 5 días | lun 18/10/10 | vie 22/10/10 | 103 | |
| 132 | | Prueba Traductor MOVE; | 2 días | mié 20/10/10 | jue 21/10/10 | 104 | |
| 133 | | Prueba Traductor ADD; | 2 días | vie 22/10/10 | lun 25/10/10 | 105 | |
| 134 | | Prueba Traductor ATTACH | 2 días | mar 26/10/10 | mié 27/10/10 | 106 | |
| 135 | | Prueba Traductor WRITE; | 2 días | jue 28/10/10 | vie 29/10/10 | 107 | |
| 136 | | **Etapa Final** | **10 días** | **lun 24/01/11** | **vie 04/02/11** | | |
| 137 | | Prueba Salida en formato HTML | 10 días | lun 24/01/11 | vie 04/02/11 | 113 | |
| 138 | | **SOA** | **30 días** | **vie 26/11/10** | **jue 06/01/11** | | |
| 139 | | **Análisis y Diseño SOA** | **4 días** | **vie 26/11/10** | **mié 01/12/10** | | **Martín Ale,Pablo Pirán,Pablo Cl** |
| 140 | | Estudio SOA | 4 días | vie 26/11/10 | mié 01/12/10 | | |
| 141 | | Reuniones con usuarios BROU | 2 días | jue 02/12/10 | vie 03/12/10 | 140 | Martín Ale,Pablo Pirán,Pablo Claps |
| 142 | | Documentación de cómo llevar el módulo de Banca Electrónica a SOA | 8 días | lun 27/12/10 | mié 05/01/11 | | Martín Ale,Pablo Pirán |
| 143 | | Entregable SOA | 1 día | jue 06/01/11 | jue 06/01/11 | 142 | Martín Ale |
| 144 | | **Presentación Final** | **36 días** | **lun 24/01/11** | **lun 14/03/11** | | |
| 145 | | Informe Final | 25 días | lun 24/01/11 | dom 27/02/11 | | Martín Ale,Pablo Pirán,Pablo Claps |
| 146 | | Creación de PPT | 11 días | lun 28/02/11 | lun 14/03/11 | 145 | Martín Ale,Pablo Pirán,Pablo Claps |

# 8 - Captura de Log de Ejecución de Transacciones

## 8.1 - Propósito

Documentar técnicamente el script desarrollado para la captura de los cabezales de las transacciones ejecutadas en el sistema central del BROU (Sistema Financiero Bancario).

## 8.2 - Alcance

Análisis, Diseño e Implementación de la solución.

## 8.3 - Documentación

Se documentará cada etapa por separada.

## 8.4 - Análisis

Los sistemas EAE permiten registrar en un log todos los mensajes de entrada y de salida del sistema. En el BROU esa opción está habilitada, por lo que se cuenta con logs de todas las transacciones ejecutadas.
Cada mensaje de entrada y de salida es particionado en una cantidad de líneas diferente según el largo del mensaje, pero a través de una codificación es posible identificar la primer línea de los mensajes de entrada y de salida, que son las que contienen la información necesaria para elaborar las estadísticas que nos interesan para nuestro proyecto. Dicha información incluye la transacción ejecutada, usuario que la ejecuta y desde qué terminal, fecha/hora de solicitud de la transacción y fecha/hora de la respuesta.

## 8.5 - Diseño

Los logs se guardan en archivos de un tamaño parametrizable, y cuando el archivo actual alcanza dicho tamaño, se ejecuta el script log_change que está incluido en la instalación de EAE y es posible modificarlo. Por defecto, lo que hace dicho script es cerrar el archivo de log actual, y abrir un nuevo archivo, agregando un numerador incremental al nombre del nuevo archivo.
Nuestra solución consiste en invocar dentro del script log_change un awk que extraiga los primeros caracteres de los mensajes de entrada y de salida, que tienen los datos que nos interesan y no contienen ningún tipo de información confidencial.

## 8.6 - Implementación

El awk desarrollado es el siguiente:

```
# gensqa.awk
#
# Este script extrae del log de linc los primeros 112 caracteres de los registros

#
# Variables en uso:
#     ord = orden del registro del log
#           " " el mensaje ocupa un solo registro del log
#           "1", "2" el mensaje ocupa mas de un registro en este orden
#           "E" el mensaje ocupa mas de un registro y este es el ultimo
#     linea = registro leido del log
#     largo = largo de la linea


{
   ord = substr($0,2,1);

   if ( ord == " " || ord == "1" )
      {
       # es el primer registro de un mensaje de entrada o salida
       linea = $0;
       largo = length(linea);

       if ( largo < 112 )
          { # completo la linea para que mida 112 caracteres
            for (j=largo;j<112;j++)
              linea = linea "#";          }
       else
          { linea = substr(linea,1,112);  }

       # imprimo la linea
       print linea;
      }
}
```

La invocación de dicho awk se realiza con la siguiente sentencia:

```
nawk -f gensqa.awk $LOG_FILE > $LOG_FILE.sqa
```

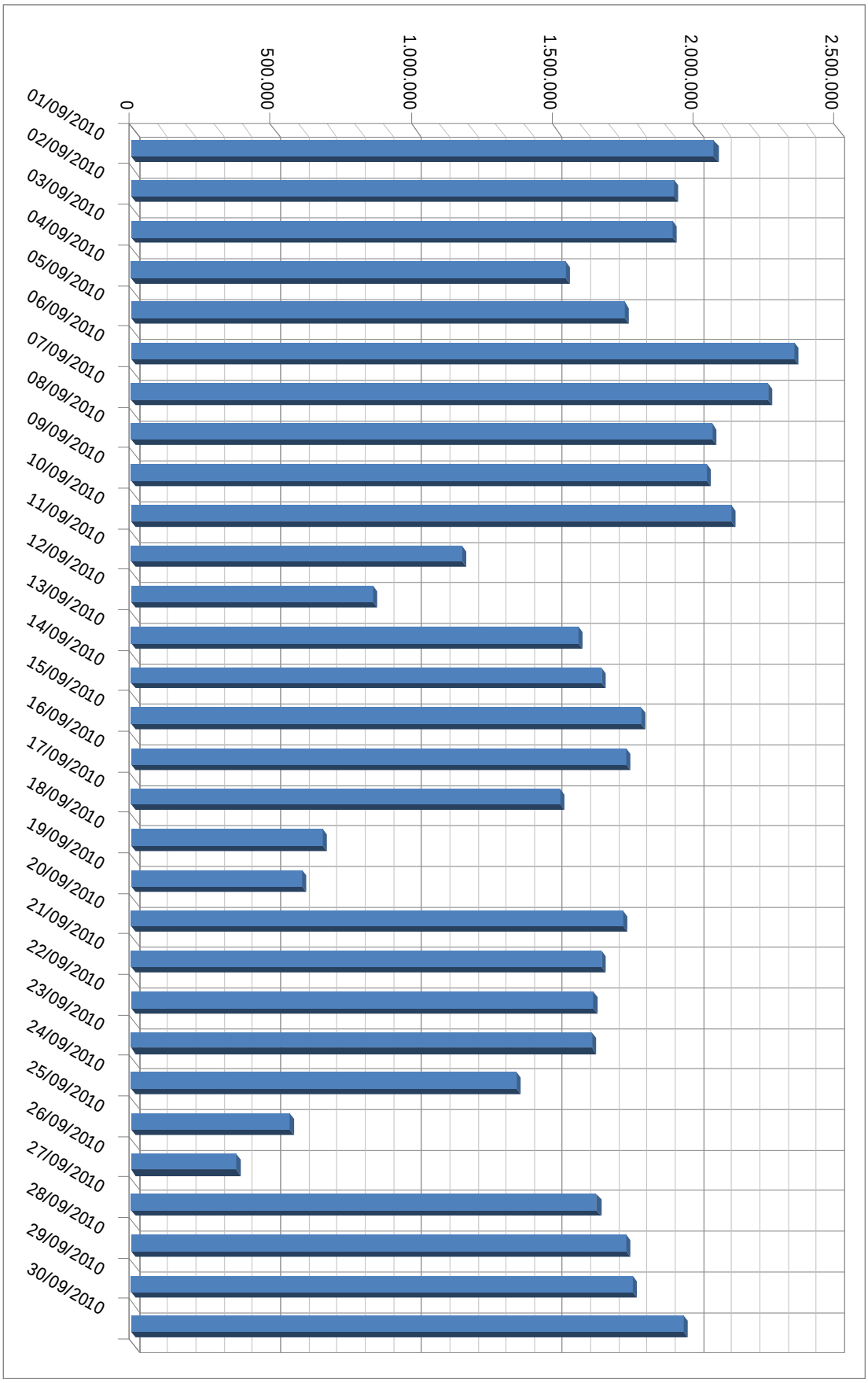
donde $LOG_FILE es el archivo de LOG que se está cerrando, y $LOG_FILE.sqa será el archivo de salida con la información requerida por nosotros.

# 9 - Estadísticas del Sistema en Producción

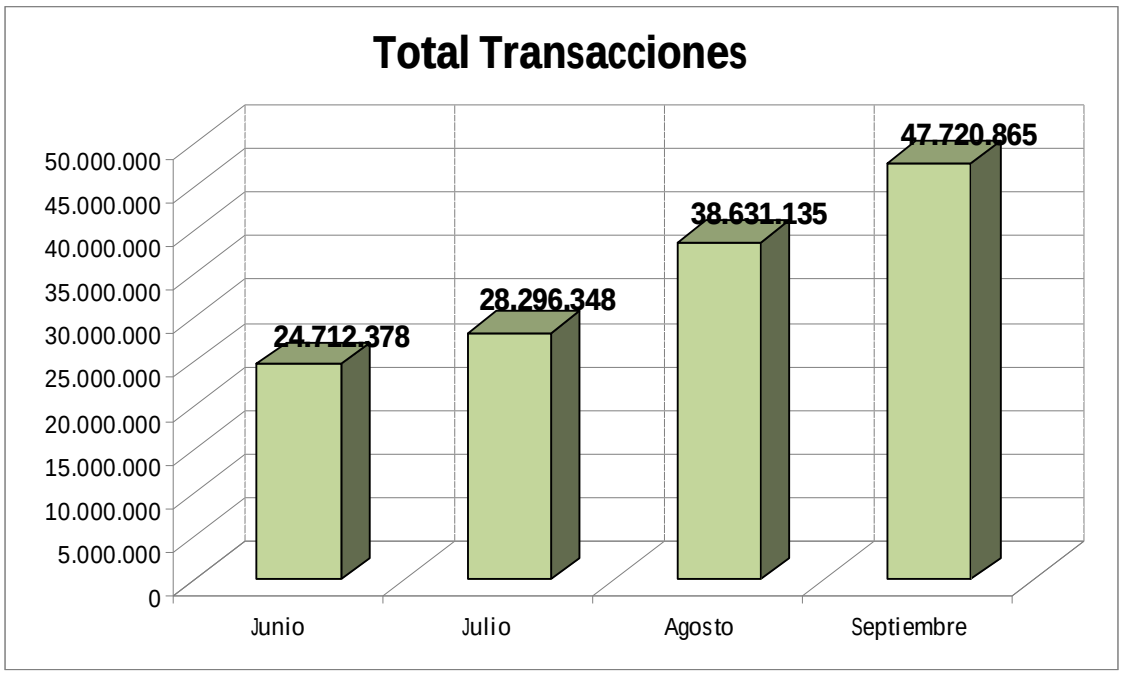
## 9.1 - Transacciones de setiembre del 2010

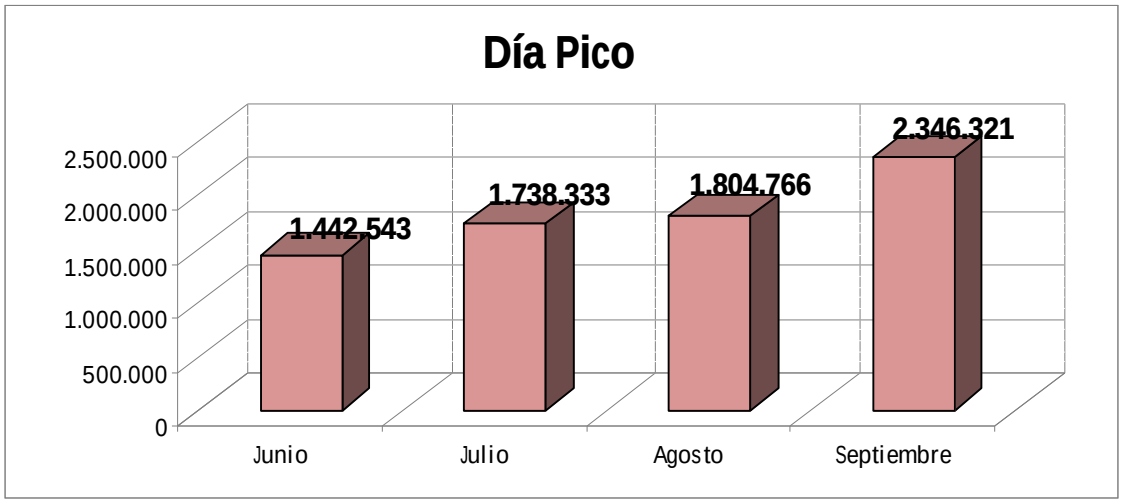
| Fecha | Transacciones |
|---|---|
| 01/09/2010 | 2.063.413 |
| 02/09/2010 | 1.921.287 |
| 03/09/2010 | 1.916.098 |
| 04/09/2010 | 1.538.045 |
| 05/09/2010 | 1.747.392 |
| 06/09/2010 | 2.346.321 |
| 07/09/2010 | 2.254.292 |
| 08/09/2010 | 2.057.651 |
| 09/09/2010 | 2.035.599 |
| 10/09/2010 | 2.122.350 |
| 11/09/2010 | 1.168.662 |
| 12/09/2010 | 854.722 |
| 13/09/2010 | 1.582.027 |
| 14/09/2010 | 1.664.510 |
| 15/09/2010 | 1.803.175 |
| 16/09/2010 | 1.752.805 |
| 17/09/2010 | 1.518.418 |
| 18/09/2010 | 676.697 |
| 19/09/2010 | 605.501 |
| 20/09/2010 | 1.739.334 |
| 21/09/2010 | 1.665.188 |
| 22/09/2010 | 1.634.128 |
| 23/09/2010 | 1.631.677 |
| 24/09/2010 | 1.364.004 |
| 25/09/2010 | 559.024 |
| 26/09/2010 | 372.380 |
| 27/09/2010 | 1.646.995 |
| 28/09/2010 | 1.749.389 |
| 29/09/2010 | 1.776.753 |
| 30/09/2010 | 1.953.028 |
| Total | 47.720.865 |

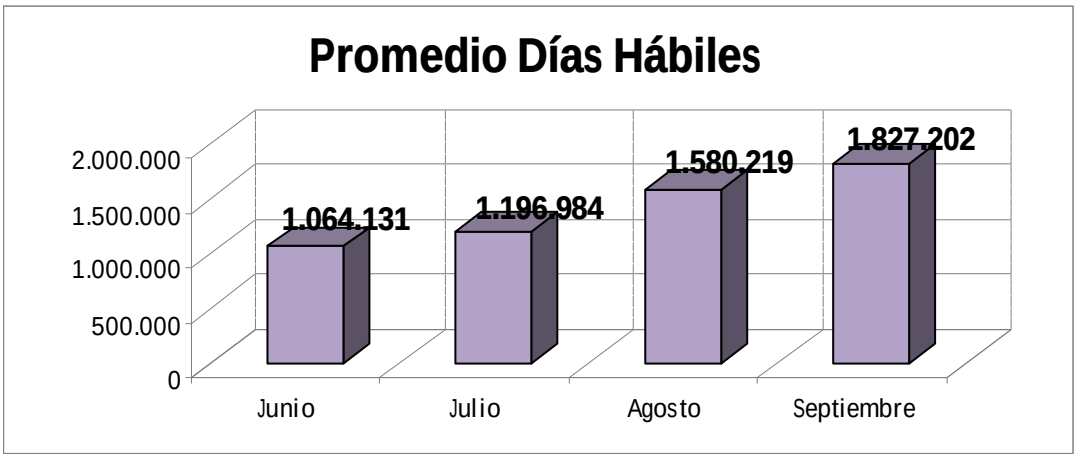
| | |
|---|---|
| Promedio Días Hábiles | 1.827.202 |
| Promedio Fines de Semana | 940.303 |

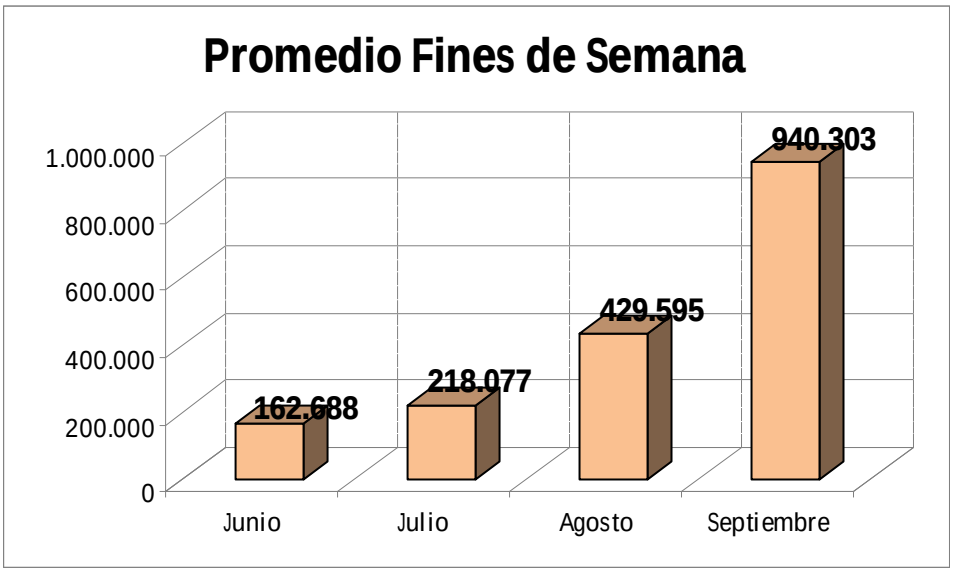

0
500.000
1.000.000
1.500.000
2.000.000
2.500.000
01/09/2010
02/09/2010
03/09/2010
04/09/2010
05/09/2010
06/09/2010
07/09/2010
08/09/2010
09/09/2010
10/09/2010
11/09/2010
12/09/2010
13/09/2010
14/09/2010
15/09/2010
16/09/2010
17/09/2010
18/09/2010
19/09/2010
20/09/2010
21/09/2010
22/09/2010
23/09/2010
24/09/2010
25/09/2010
26/09/2010
27/09/2010
28/09/2010
29/09/2010
30/09/2010

## 9.2 - Progresión mensual

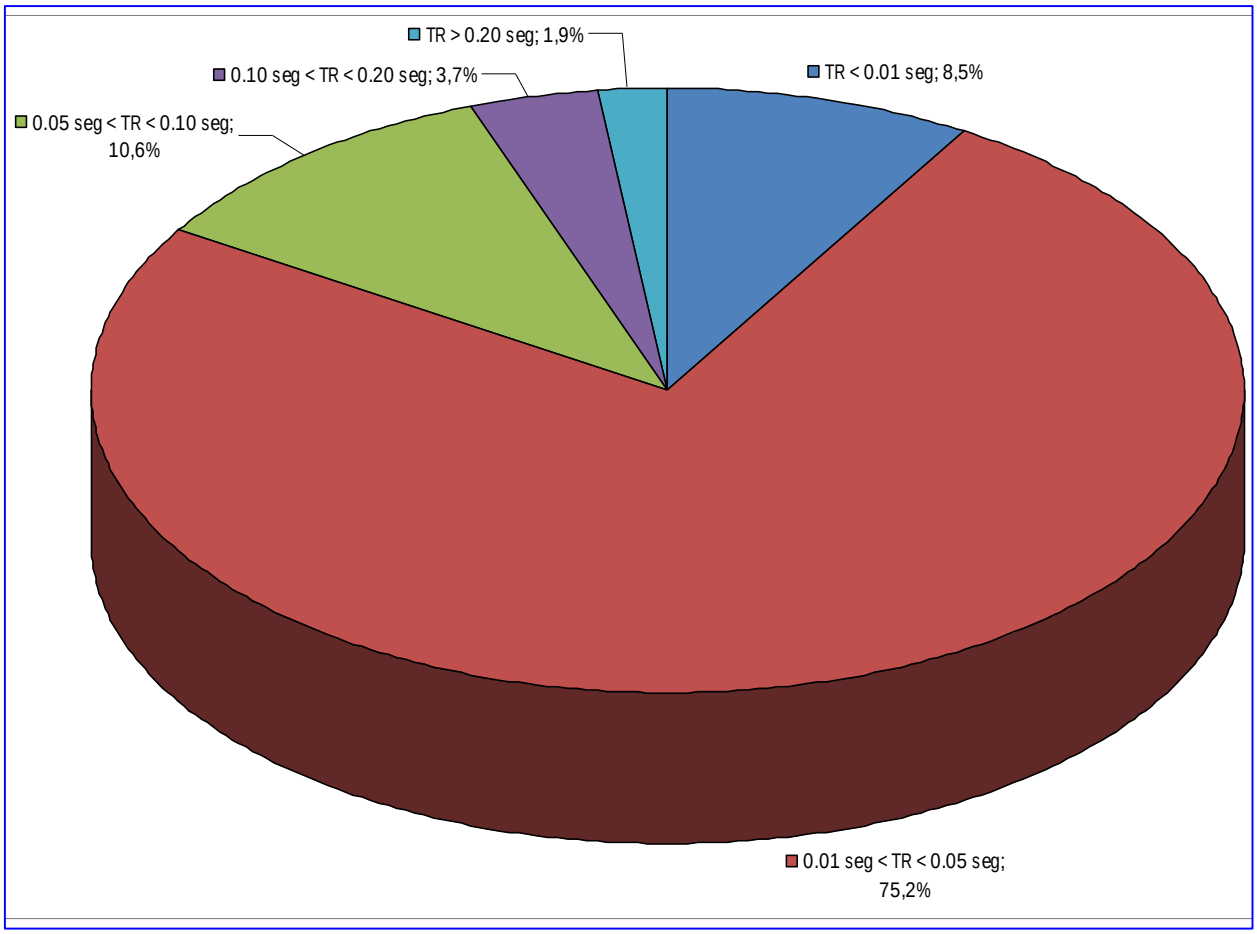



Promedio Fines de Semana
1.000.000
800.000
600.000
400.000
200.000
0
162.688
218.077
429.595
940.303
Junio
Julio
Agosto
Septiembre

## 9.3 - Tiempos de respuesta



# 10 - Introducción a E.A.E.

## 10.1 - Propósito

Dar una breve introducción a EAE, sus principales características y los objetos que lo componen.

## 10.2 - Alcance

Descripción de los objetos y características de EAE que son relevantes para el proyecto de grado.

## 10.3 - Introducción a EAE

### 10.3.1 - Definición

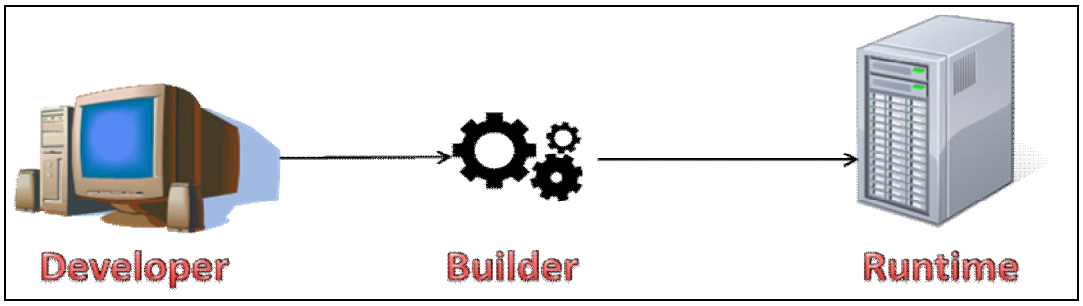
Enterprise Application Environment (EAE) es un conjunto de herramientas que proveen un ambiente de desarrollo, pruebas, generación y ejecución de aplicaciones, propietario de Unisys Corporation. Es un lenguaje de cuarta generación, multiplataforma, que permite generar a partir de código fuente escrito en una ambiente Windows fuentes Cobol para diferentes sistemas operativos y bases de datos.
Componentes

Los componentes básicos de EAE son los siguientes:

**Developer:** es un ambiente de desarrollo para Windows que permite diseñar y definir elementos y sus relaciones, construir y testear sistemas localmente. El lenguaje de desarrollo es conocido como LDL: LINC Development Language (LINC es el nombre de las primeras versiones de EAE).
El editor presenta una estructura de árbol, en la que la raíz es el modelo (que se corresponde con un repositorio), y el primer nivel es ocupado por los segmentos de negocio (que se corresponden con los sistemas EAE).

**Builder:** a partir de la información contenida en un repositorio EAE, se generan fuentes Cobol que son transferidos y compilados en el Host seleccionado, y sentencias SQL que son ejecutadas en la base de datos indicada.

**Runtime:** se encarga de manejar la insfraestructura necesaria para la ejecución del sistema EAE (integridad transaccional, manejo de reportes, intercomunicación entre sistemas, etc.)




### 10.3.2 - Principales elementos

Los elementos que podemos identificar en EAD son los Componentes, los Eventos, las Lógicas Globales, los Perfiles y los Reportes.
En el caso del sistema central del BROU, SFB (por las siglas de Sistema Financiero Bancario), no son utilizados los Eventos, por lo que no serán tenidos en cuenta en este documentos.

#### 10.3.2.1 - Componentes

Los Componentes ISPEC (de Interface SPECifcation) es uno de los bloques básicos a partir de los que se construye una aplicación EAE.
Permiten definir la interfaz con el usuario, la lógica de negocio, y las estructuras de la base de datos de la aplicación.
Hay tres tipos de Componentes según la definición de la propiedad Usage (Uso):
Input: Definen una interfaz con el usuario (pantalla) y ejecutan lógica de negocio.
Output: Se corresponden con una estructura en la base de datos.
Input-Output: Combina las propiedades de ambos tipos.

Los atributos o campos pertenecientes a un Componente se denominan “Data Items”.
También los “Data Items” tienen la propiedad Usage, y debe ser compatible esta propiedad con la del Componente al que pertenece.
Los Usos posibles para los “Data Items” son:
Input: Es un campo de entrada en la pantalla de un Componente.
Output: Se corresponde con un campo en la tabla definida por el Componente al que pertenece.
Input-Output: Es ambas cosas, un campo de entrada en la pantalla y un campo en la base de datos.
Inquiry: En un campo de pantalla no ingresable (se utilizan para desplegar información, pero no pueden ser modificados)

Las posibles combinaciones de Usos de Componentes con los de sus Data Items se resumen en la siguiente tabla:

| Uso Componente | Usos válidos Data Items |
|---|---|
| Input | Input, Inquiry |
| Output | Output |
| Input-Output | Input, Output, Input-Output, Inquiry |

#### 10.3.2.2 - Lógicas globales

Las Lógicas Globales permiten encapsular rutinas y definiciones de interfaz de manera de ser reutilizadas por el resto de los elementos de la aplicación.

#### 10.3.2.3 - Perfiles

Los Perfiles permiten a los desarrolladores definir índices en tablas de la base de datos de la aplicación.

#### 10.3.2.4 - Reportes

Los Reportes son programas EAE que permiten ejecutar procesos en la base de datos. No son utilizados únicamente para procesamiento batch. Permiten extraer, consolidar y listar información de la base de datos de la aplicación.

#### 10.3.2.5 - Otros elementos

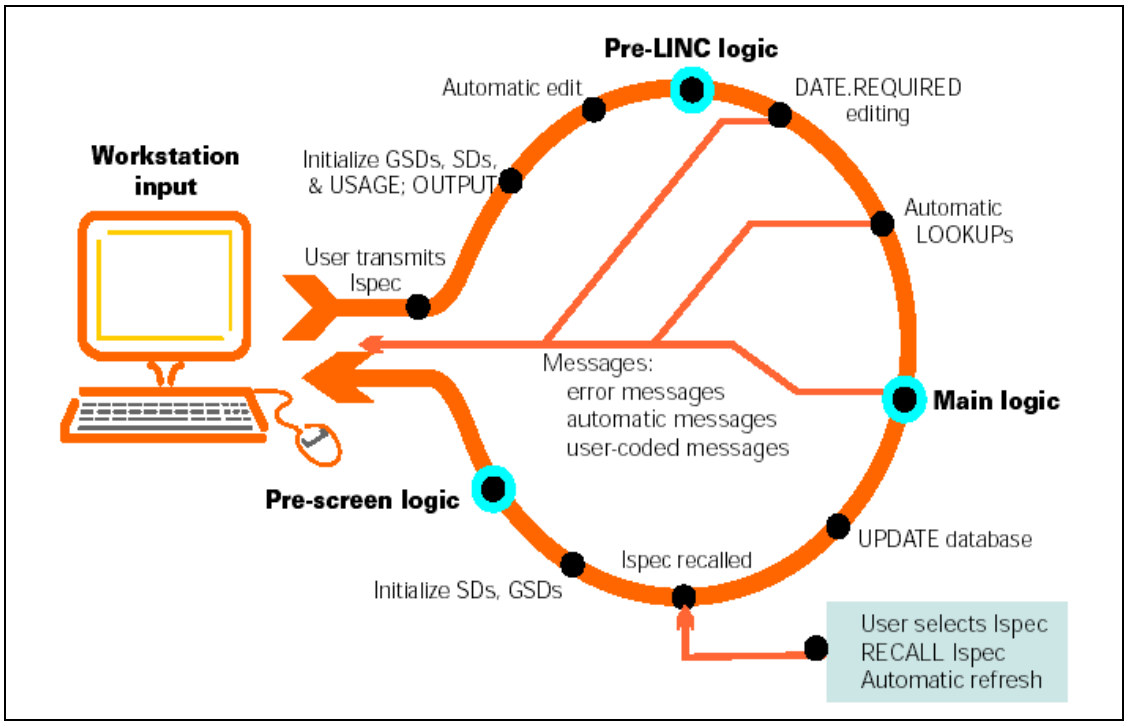
EAE permite definir un diccionario de datos para cada aplicación.
En el caso del sistema SFB del BROU, se utiliza como Diccionario de la Base de Datos. Cada campo de una tabla del sistema está definido en dicho Diccionario.

También se definen en EAE bloques de variables globales. Las variables globales son las que se utilizan para el pasaje de parámetros en las lógicas globales.
Además, si se incluyen en un grupo definido como GLOBAL.WORK, es posible el pasaje de parámetros entre dos transacciones.

### 10.3.3 - Ciclo de ejecución

EAE permite definir diferentes lógicas en cada ISPEC, que se ejecutan según el Ciclo de Ejecución LINC. Los diferentes pasos del ciclo deberán ser tenidos en cuenta para extraer las reglas de negocio a partir del código fuente EAE.

La siguiente figura describe el ciclo LINC que siguen las transacciones EAE que vamos a estudiar como parte del proyecto.

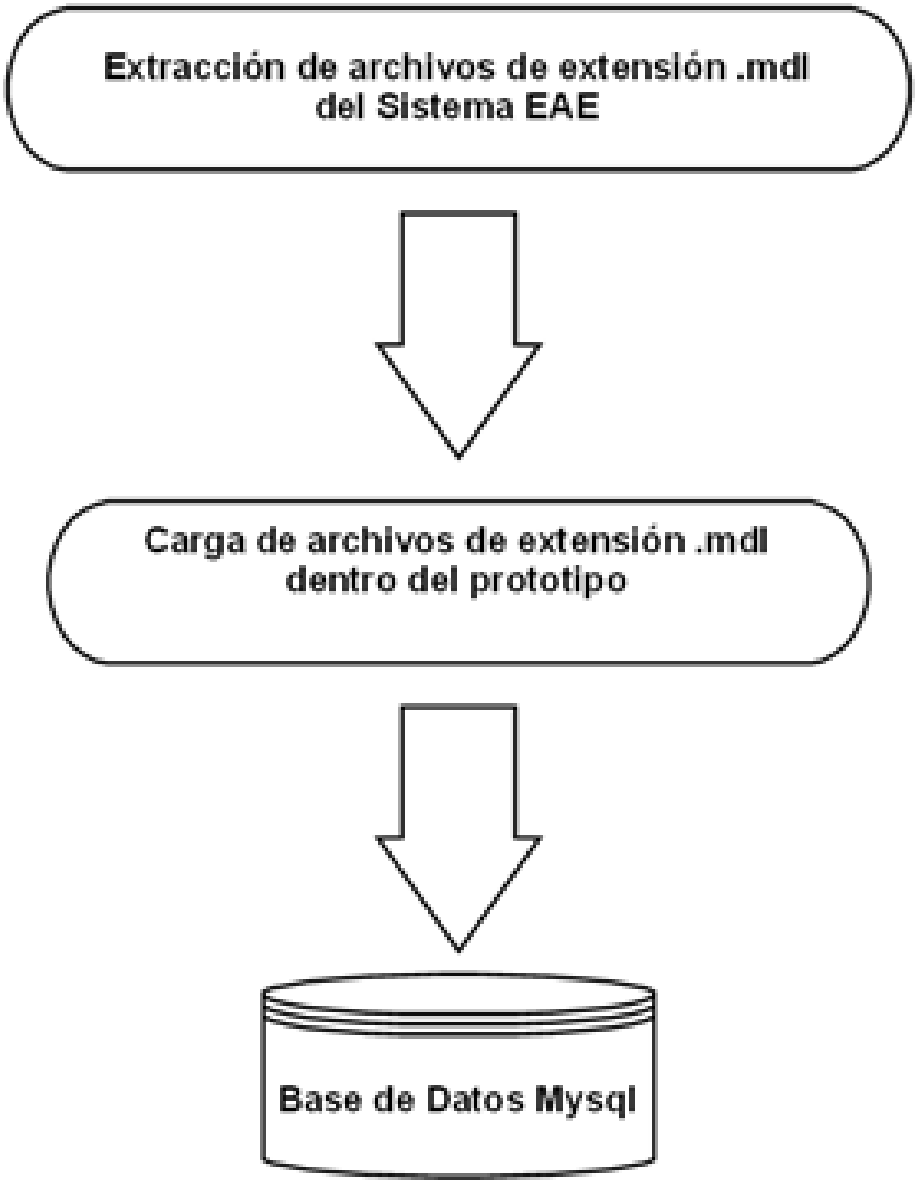



Básicamente, el ciclo se inicia cuando el usuario final ejecuta una acción que envía una solicitud al sistema.

Cada transacción tiene asociados tres segmentos de código: la lógica de Pre-LINC (“Prepare Logic”), la lógica principal (“Main Logic”) y la lógica previa a que se despliega la pantalla (“Pre-Screen Logic”).

El resto del ciclo se completo con inicializaciones, validaciones y procesamientos automáticos que se crean durante la generación del sistema EAE.

## 10.4 - Extracción de información de EAE y carga en la BD



# 11 - Análisis de Código Fuente EAE

## 11.1 - Propósito

Documentar el análisis del Código Fuente EAE que se procesará.

## 11.2 - Alcance

Determinación de patrones a buscar dentro del Código Fuente. Explicar las cadenas de texto a buscar y el pseudocódigo a la que corresponde.

## 11.3 - Análisis de código fuente

### 11.3.1 - Introducción

Consideramos dos secciones dentro del código fuente EAE. Por un lado aquella que corresponde a validaciones, y por otro la que corresponde al procesamiento.

### 11.3.2 - Validación

Cuando se valida una condición de error en EAD se corta el ciclo de ejecución con alguno de los siguientes comandos:
END.EXIT
JUMP.TO; {etiqueta}

El sistema SFB está desarrollado siguiendo un estándar para el manejo de errores. Al encontrar un error, siempre se carga un código de error en la variable global GD-MEN-CODIG.
Podemos encontrar los siguientes casos de validaciones:

**Caso 1:**

Se valida una condición particular.

```
DO.WHEN; {condición}
        MOVE;  (nnnn) GD-MEN-CODIG
        JUMP.TO; {etiqueta}
END;
```

Ejemplo:

```
DO.WHEN;  GD-MONDES  NOT =  GD-MONRES
   MOVE;  (5760)  GD-MEN-CODIG
   JUMP.TO;  FINERRCA
END;
```

**Caso 2:**

Se invoca lógica global que carga la variable GD-MEN-CODIG en caso de error.

```
INSERT; {Lógica Global}
DO.WHEN; GD-MEN-CODIG NOT = GLB.ZEROS
        JUMP.TO; {etiqueta}
END;
```

Ejemplo:

```
INSERT;  GP-LOCK-CTA-AH
DO.WHEN;   GD-MEN-CODIG  NOT = GLB.ZEROS
   JUMP.TO;  FINERRCA
END;
```

**Caso 3:**

Se lee una tabla y se valida el resultado de la lectura. La lectura se realiza con el comando DETERMINE, que carga la variable global GLB.STATUS con espacios en caso de encontrar registros que cumpla la condición, y con "*****" en caso contrario.

```
DETERMINE; {tabla y tipo de lectura}
        …
END;
DO.WHEN; {condición con GLB.STATUS}
        MOVE;  (nnnn) GD-MEN-CODIG
        JUMP.TO; {etiqueta}
END;
```

Ejemplo:

```
DETERMINE; EVERY P-AQAUT (GD-OFIPAR GD-NUMPAR GD-DOCDES) SECURE;
  BREAK;
END;
DO.WHEN;  GLB.STATUS  NOT =  GLB.SPACES
  MOVE;  (4249)  GD-MEN-CODIG
  JUMP.TO; FINERRAQ
END;
```

**Caso 4:**

Se invoca una lógica global que ejecuta una lectura en la base de datos, y devuelve el resultado en la variable global GD-STATUS (se carga con el valor de GLB.STATUS después de ejecutar el comando DETERMINE).

```
INSERT; {Lógica Global}
DO.WHEN; {condición con GD-STATUS}
        MOVE;  (nnnn) GD-MEN-CODIG
        JUMP.TO; {etiqueta}
END;
```

Ejemplo:

```
INSERT;  GP-BUSCA-TRX
DO.WHEN;   GD-STATUS  NOT =  GLB.SPACES
  MOVE;  (0077)       GD-MEN-CODIG
  JUMP.TO;  FINERRAQ
END;
```

En todos los casos después de cargada la variable GD-MEN-CODIG, puede aparecer una llamada a una lógica de manejo de errores y se cierra el comando DO.WHEN con la línea END.EXIT.

### 11.3.3 - Proceso

Durante el proceso lo que debemos buscar son bloques lógicos, compuestos por uno o más comandos EAE.
A continuación se listan los comandos que serán considerados en nuestro proceso, la sintaxis de cada uno, y una breve descripción en caso de ser necesaria.

Suma de dos números:
ADD;
ADD; {item 1} {item 2} [ GIVING; {item 3} ] [ ROUNDED; ]

Ejemplo:

```
ADD;  AMOUNT  SUBTOTAL  GIVING;  TOTAL
```

Si no se usa la opción GIVING el resultado se guarda en {ítem 2}

Concatenación de strings:
ATTACH;
ATTACH; {item 1} {item 2}

ATTACH&SPACE;
ATTACH&SPACE; {item 1} {item 2}

Ejemplo:

```
ATTACH; SURNAME  FIRSTNAME
```

El resultado queda almacenado en el {item 2}
El comando ATTACH&SPACE inserta un blanco entre los dos strings.

Alta de Registro en Tabla
AUTO.ENTRY;
AUTO.ENTRY; {ispec}

Ejemplo:

```
AUTO.ENTRY; AJNL
        AUTO; CUSTOMER  CUSTOMER
        AUTO; AMOUNT    AMOUNT
AUTO;WRITE&CLEAR
```

Bloque Iterativo
BEGIN.LOOP;
BEGIN.LOOP;

El BEGIN.LOOP se cierra con el comando END.

Cortar Iteración
BREAK;
BREAK; [ ALL ]

Los comandos que generan una iteración son BEGIN.LOOP y DETERMINE.

Sentencia CASE
CASE;
BEGIN.CASE;  {item 1}
CASE;  {Data Item 2}  [ {Data Item 3} ...]
OTHERWISE;
END.CASE;

Operación Aritmética
COMPUTE;
COMPUTE; {Item 1} ({item 2} {operator} {item 3} [[{operator} {item 4}] ...[{operator}{item 16}]]) [ ROUNDED; ]

Cambio de signo de un número:
CONTRA;
CONTRA;  {numeric data}

Para que tenga efecto el dato debe ser numérico con signo.

Conversión y Validación de Fechas
DATE.CONVERT;
DATE.CONVERT; {item 1} [ {+ or - } {item 2}] FORMAT; {format} [EDIT.ONLY]

DATE.CONVERT; {item 1} [ {+ or - } {item 2}] FORMAT; {format} [{item 3} FORMAT {format}

Lectura de una tabla:
DETERMINE;
DETERMINE; ACTUAL {profile} [ SERIAL; ] [ SECURE; ] [ MULTI; {numeric data} ]

DETERMINE; EVERY {profile} ( {profile keys} ) [ SERIAL; ] [ SECURE; ] [ MULTI; {numeric data} ]

DETERMINE; FROM {profile} ( {profile keys} ) [ SERIAL; ] [ SECURE; ] [ MULTI; {numeric data} ]

DETERMINE; GROUP {profile} {profile keys} [ SERIAL; ] [ SECURE; ] [ MULTI; {numeric data} ]

El DETERMINE crea una iteración que recorre los registros que cumplen la condición por la que se está leyendo la tabla.
División Aritmética:
DIVIDE;

DIVIDE;  {item1}  {item 2} [ GIVING; {item 3} ]  [ ROUNDED; ]  [ REMAINDER;  {item 4} ]

Si no se usa la opción GIVING el resultado se guarda en {ítem 2}

Control de Condición:
DO.WHEN;
DO.WHEN;  { item 1}  {operator} [ {item 2}]  [ AND | OR ]

Los comandos DO.WHEN pueden combinarse utilizando AND o OR (para evaluar múltiples condiciones). En caso de cumplirse las condiciones se ejecuta el bloque lógico que sigue al DO.WHEN y que se cierra con un END.

Ejemplo:

```
   DO.WHEN;  GD-MTOPEN NOT =  GLB.ZEROS
      ::
      MOVE;  GD-SALPE       GD-TRANS
      ::
      DETERMINE; EVERY P-AHAPE (GD-OFICUE GD-MONED GD-NUMCUE
            WS-RELSI  GD-NUMDOC)  SECURE;
         ::
         DO.WHEN;  AHAPE.ACO-TRANS = GD-TRANS   AND
         DW;       AHAPE.ACO-CAUSA = GD-CAUSA   AND
         DW;       AHAPE.TVA-MOVIM = GD-MTOPEN
            ::
            PURGE;  AHAPE
         END;
      END;
      ::
      MOVE;  GD-INGPE       GD-TRANS
      ::
      DETERMINE; EVERY P-AAACN ( GD-CONCEP WS-RELSI
            GD-NROREV ) SECURE;
         ::
         DO.WHEN;  AAACN.DCO-TRANS = GD-TRANS   AND
         DW;       AAACN.ACO-CAUSA = GD-CAUSA   AND
         DW;       AAACN.ACU-OFICI = GD-OFICUE  AND
         DW;       AAACN.ACUNUMCUE = GD-NUMCUE  AND
         DW;       AAACN.TNUDOCTRA = GD-NUMDOC  AND
         DW;       AAACN.TVA-MOVIM = GD-MTOPEN
            ::
            PURGE;  AAACN
         END;
         ::
      END;
      ::
   END;
```

Cierre de Bloque:
END;
END;

Cierre de Bloque con Error:
END.EXIT;

Las líneas siguientes a un END.EXIT no son ejecutadas.

Modificación de registro de una tabla:
FLAG;
FLAG;  { item 1}  { item 2}

Llamada a una lógica global:
INSERT;
INSERT; {global logic}

Salto:
JUMP.TO;
JUMP.TO;  {label}

Etiqueta:
LABEL;
LABEL;  {label}

Enviar mensaje al usuario:
MESSAGE;
MESSAGE;  {item 1}  {item 2}

MESSAGE; ERROR  {item 2}

MESSAGE; ATTENTION {item 2}

Manipulación de datos:
MOVE;
MOVE;  {item 1} [POSITION; {pos 1}] [LENGTH; {le}] {item 2} [POSITION; {pos 2}

Multiplicación Aritmética:
MULTIPLY;
MULTIPLY;  {item 1}  {item 2} [ GIVING; {item 3} ]  [ ROUNDED; ]

Eliminación de registro de una tabla:
PURGE;
PURGE;  {component}

Invocación a otra transacción:
RECALL;
RECALL;  [ {ispec} ]

Resta Aritmética:
SUBSTRACT;
SUBTRACT;  {item 1}  {item 2}  [ GIVING; {item 3} ]  [ ROUNDED; ]

# 12 - Teoría de Compiladores

## 12.1 – Propósito

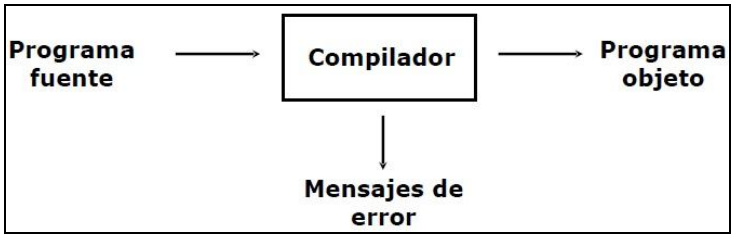
Ofrecer una introducción al marco teórico de diseño de compiladores.

## 12.2 – Alcance

Describir en forma macro el funcionamiento de un compilador, y de qué forma este marco teórico aplica al proyecto de extracción de reglas de negocio.

## 12.3 – Marco teórico de compiladores

Un compilador es un programa que lee un programa escrito en un lenguaje y lo traduce a un programa escrito en otro lenguaje. Parte del proceso es la generación de mensajes de error.

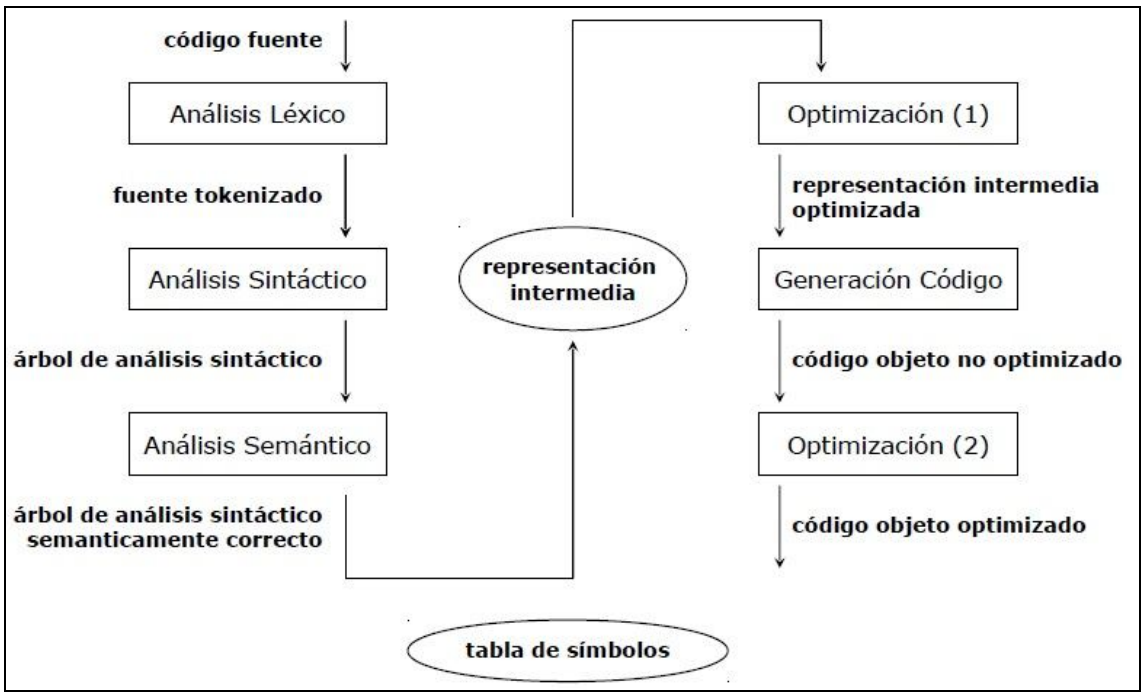


## 12.4 – Proceso de compilación

El proceso de compilación se divide en dos partes, análisis y síntesis. El análisis divide el programa fuente en sus elementos componentes creando una representación estructurada e intermedia del mismo. La síntesis construye el programa objeto desde la representación intermedia realizada por el análisis. Asimismo se puede optimizar el programa generado.

El proceso de compilación es el proceso que nos lleva desde el fuente hasta el objeto, pasando por etapas intermedias específicas y obteniendo subproductos en cada una de ellas. En cada etapa se pueden detectar errores, pero al terminar una etapa se asume que hasta ese punto no existen (por lo menos errores detectables).

En la siguiente figura se ilustra el proceso de compilación y sus etapas.




## 12.5 – Análisis léxico

Se analiza la entrada carácter a carácter de izquierda a derecha y se agrupan en componentes léxicos denominados tokens. Los tokens son secuencias de caracteres con un significado común. Se utilizan expresiones regulares para especificar los mismos. En esta etapa se eliminan comentarios y espacios en blanco.
También se relacionan los errores con la posición en el programa fuente.

Ejemplo:

velocidad = distancia / tiempo * 60 → ID ASIG ID DIV ID PROD NUM

Se utilizan principalmente tres conceptos: componente léxico o token, patrón y lexema. Un conjunto de la entrada concuerda con un token (para el ejemplo: ID, ASIG, DIV, etc), el patrón es la regla que asocia la cadena al token, mientras que un lexema es la

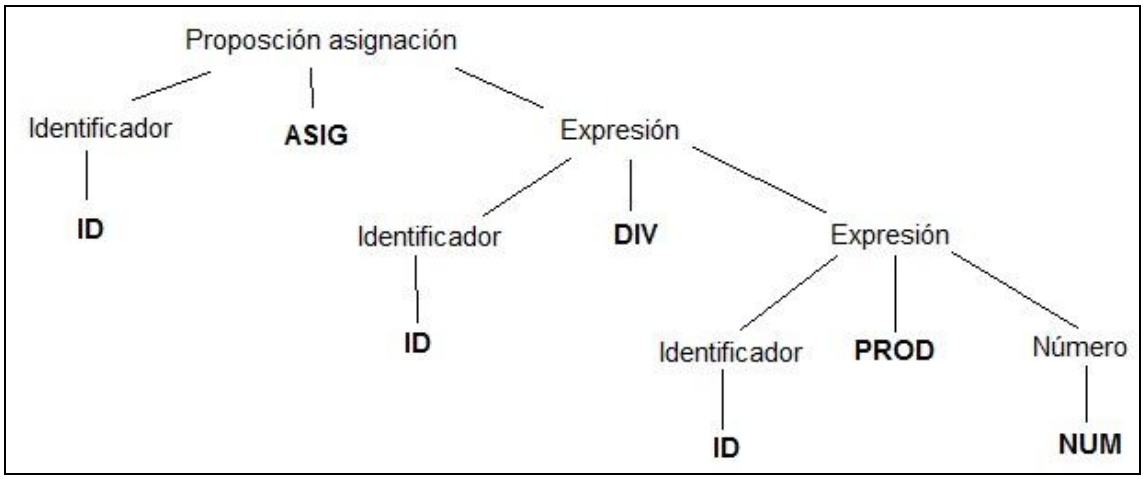
secuencia de caracteres en el fuente con el que concuerda con el patrón para un token. Los patrones se escriben como expresiones regulares.

## 12.6 – Análisis sintáctico

Corresponde a un análisis de los tokens y agruparlos en frases gramaticales utilizadas para la síntesis. Las frases gramaticales se representan con árboles de análisis sintáctico, o árboles sintácticos.

Ejemplo

ID ASIG ID DIV ID PROD NUM →




La división entre el análisis léxico y el sintáctico es arbitraria, y por lo general depende de las características del lenguaje.

Existen dos grandes clases de análisis sintáctico, ascendente y descendente. El análisis por descenso recursivo, o descendente, construye el árbol para una cadena desde la raíz hasta las hojas, mientras que el ascendente lo hace desde las hojas a la raíz.

## 12.7 – Análisis semántico

Busca errores semánticos a partir del resultado de las etapas anteriores. Reúne información de los tipos y verifica entre otras cosas el uso correcto de los operandos en un operador, acceso a rangos dentro de un vector, y conversiones explícitas o implícitas de tipos. Una conversión de tipos se dice implícita cuando es realizada automáticamente por el compilador, mientras que las conversiones explícitas tienen que ser realizadas por el programador (casteos).

Se puede generar una representación intermedia explícita del programa fuente, la cual debe ser fácil de producir y de traducir. Algunos ejemplos de esta representación son el árbol sintáctico y el código de tres direcciones (a lo sumo un operador en el lado derecho de la instrucción).

## 12.8 – Optimización (1)

Se mejora el código intermedio para lograr un código más rápido y compacto. Entre los tipos de optimizaciones está el movimiento de código, eliminación de código inalcanzable, y eliminación de código redundante. Es una de las etapas más largas y complejas del proceso.

## 12.9 – Código final

Se genera el código objeto, el cual suele ser código de máquina reubicable o código ensamblador (assembler). Cada instrucción de este código se traduce en una instrucción de código final.

## 12.10 – Optimización (2)

Análogamente a la primera optimización, aquí se busca optimizar el código objeto generado.

## 12.11 – Tabla de símbolos

La tabla de símbolos está presente a lo largo de todo el proceso. La misma registra identificadores usados a lo largo del programa fuente y almacena información de cada uno de ellos, como ser nombre, tipo y ámbito. Esta información se va completando a medida que avanzan las fases. Debe ser de fácil acceso y permitir una búsqueda rápida a la información de cualquier identificador en cualquier momento.

## 12.12 – Detección de errores

Puede existir en cada fase y debe tratar de detectar la mayor cantidad de errores posible. Un compilador luego de detectar un error no debería detenerse. La mayor cantidad de errores se encuentran en la etapa de análisis, luego de que es construida la representación intermedia es difícil encontrar errores en el fuente.

## 12.13 – FrontEnd vs. BackEnd

Estas etapas se pueden separar en dos divisiones. El frontend está compuesto por las etapas dependientes del lenguaje fuente e independiente del lenguaje objeto. A su vez, el backend está compuesto por las etapas dependientes del lenguaje objeto e independiente del lenguaje fuente. La frontera entre estas divisiones es la representación intermedia. Esta visión ofrece ventajas al momento de organizar el trabajo, aumentando la reutilización del compilador. Por ejemplo, si cambiara solo el lenguaje fuente, o solo el lenguaje objeto, solo cambiaría el frontend, o solo el backend.

## 12.14 – Generación de código

Se toma como entrada la representación de código intermedio del programa fuente y produce su equivalente en código objeto. El generador debe ser eficiente, el código generado debe ser correcto, y se deben aprovechar efectivamente los recursos de la máquina objeto. En la práctica para la generación de código óptimo se utilizan heurísticas, aunque probablemente el código quede no óptimo, sino bueno.

La representación intermedia debe ser lo suficientemente de bajo nivel, con tipos de datos que se pueden mapear a la arquitectura objeto. Se utiliza la tabla de símbolos para determinar las direcciones en tiempo de ejecución de los objetos de datos utilizando los nombres en la representación intermedia.

En este punto asumimos que el chequeo de tipos ya fue realizado, y que no hay errores semánticos obvios. Esta fase se basa en la asunción de que la representación intermedia no tiene errores (aunque en algunos compiladores la generación de código se realiza a la vez que el chequeo). El código objeto puede tomar distintas formas, como ser código de máquina absoluto, código de máquina reubicable, o código ensamblador (assembler).

El código de máquina absoluto tiene como ventaja que puede ser ubicado en un lugar fijo en memoria y ejecutado inmediatamente. El reubicable permite tener módulos que se compilan por separado para su posterior unión y carga (linking). Producir código ensamblador (assembler) como salida facilita el proceso de compilación, también facilita la optimización, aunque agrega una etapa al proceso (etapa final de ensamblado).

# 13 – Herramientas Utilizadas

## 13.1 – Propósito

Se describen las herramientas de Software utilizadas para la creación del prototipo de Extracción de Reglas de Negocio.

## 13.2 – Alcance

Implementación del Prototipo de extracción de reglas de negocio de código fuente.

## 13.3 – Tecnología y herramientas de desarrollo

**Java Platform, Enterprise Edition (Java EE)**

**Java Edición Empresarial**

La tecnología Java es un lenguaje de programación y una plataforma. El lenguaje de programación Java es un lenguaje de alto nivel orientado a objetos que tiene una sintaxis particular y estilo.

Una plataforma Java es un ambiente particular en el cual se ejecutan las aplicaciones programadas en el lenguaje Java.

Todas las plataformas Java consisten en una máquina virtual Java (VM) y una interfaz de programación de aplicaciones (API). La máquina virtual de Java es un programa, para un hardware en particular y la plataforma de software, que ejecuta las aplicaciones de la tecnología Java. Una API es una colección de componentes de software que puede utilizar para crear otros componentes de software o aplicaciones. Cada plataforma Java proporciona una máquina virtual y una API, y esto permite que las aplicaciones escritas para esa plataforma puedan funcionar en cualquier sistema compatible con todas las ventajas del lenguaje de programación Java: la independencia de la plataforma, la potencia, estabilidad, facilidad de desarrollo, y la seguridad.

La plataforma Java EE proporciona un API y entorno de ejecución para el desarrollo y ejecución a gran escala, las aplicaciones de red de varios niveles, escalable, confiable y seguro.

**Java Persistence API (JPA)**

Más conocida por sus siglas JPA, es la API de persistencia desarrollada para la plataforma Java EE. La Java Persistence API, a veces referida como JPA, es un framework del lenguaje de programación Java que maneja datos relacionales en aplicaciones usando la Plataforma Java Empresarial (JEE).

**NetBeans**

NetBeans es un entorno de desarrollo de código abierto, integrado para desarrolladores de software.
NetBeans es un producto libre, gratuito y sin restricciones de uso.

**GlassFish**

GlassFish es un servidor de aplicaciones de software libre desarrollado por Sun Microsystems, compañía adquirida por Oracle Corporation, que implementa las tecnologías definidas en la plataforma Java EE y permite ejecutar aplicaciones que siguen esta especificación.

**MySQL WorkBench**

MySQL Workbench es un software creado por la empresa informática Sun Microsystems, esta herramienta permite modelar diagramas de entidad-relación para bases de datos MySQL. Puede utilizarse para diseñar el esquema de una base de datos nueva, documentar una ya existente o realizar una migración compleja.
La aplicación elabora una representación visual de las tablas, vistas, procedimientos almacenados y claves foráneas de la base de datos.

**MySQL**

MySQL es un sistema de gestión de base de datos relacional, multihilo y multiusuario con más de seis millones de instalaciones.

# 14 - Modelo S.O.A. para Módulo de Banca Electrónica

## 14.1 - Propósito

Describir el uso de una Arquitectura Orientada a Servicios (SOA) para el módulo de Banca Electrónica de SFB.

## 14.2 - Alcance

Todas las transacciones de SFB que pueden ser ejecutadas desde un cajero automático (ATM) o desde una terminal POS.

## 14.3 - Arquitectura Orientada a Servicios

La Arquitectura Orientada a Servicios (en inglés Service Oriented Architecture), es un concepto de arquitectura de software que define la utilización de servicios para dar soporte a los requisitos del negocio.

Permite la creación de sistemas altamente escalables que reflejan el negocio de la organización, a su vez brinda una forma bien definida de exposición e invocación de servicios (comúnmente pero no exclusivamente servicios web), lo cual facilita la interacción entre diferentes sistemas propios o de terceros.

SOA define las siguientes capas de software:

Aplicaciones básicas - Sistemas desarrollados bajo cualquier arquitectura o tecnología, geográficamente dispersos y bajo cualquier figura de propiedad;
De exposición de funcionalidades - Donde las funcionalidades de la capa aplicativa son expuestas en forma de servicios (generalmente como servicios web, pero no necesariamente);
De integración de servicios - Facilitan el intercambio de datos entre elementos de la capa aplicativa orientada a procesos empresariales internos o en colaboración;
De composición de procesos - Que define el proceso en términos del negocio y sus necesidades, y que varía en función del negocio;
De entrega - donde los servicios son desplegados a los usuarios finales.

SOA proporciona una metodología y un marco de trabajo para documentar las capacidades de negocio y puede dar soporte a las actividades de integración y consolidación.

### 14.3.1 - Terminología básica

**Servicio**

Una función sin estado, auto-contenida, que acepta una(s) llamada(s) y devuelve una(s) respuesta(s) mediante una interfaz bien definida. Los servicios pueden también ejecutar unidades discretas de trabajo como serían editar y procesar una transacción.

Los servicios no dependen del estado de otras funciones o procesos. La tecnología concreta utilizada para prestar el servicio no es parte de esta definición. Existen servicios asíncronos en los que una solicitud a un servicio, crea, por ejemplo, un archivo, y en una segunda solicitud se obtiene ese archivo.

**Orquestación**

Secuenciar los servicios y proveer la lógica adicional para procesar datos. No incluye la presentación de los datos. Coordinación.

**Sin estado**

No mantiene ni depende de condición pre-existente alguna. En una SOA los servicios no son dependientes de la condición de ningún otro servicio. Reciben en la llamada toda la información que necesitan para dar una respuesta. Debido a que los servicios son "sin estado", pueden ser secuenciados (orquestados) en numerosas secuencias (algunas veces llamadas tuberías o pipelines) para realizar la lógica del negocio.

**Proveedor**

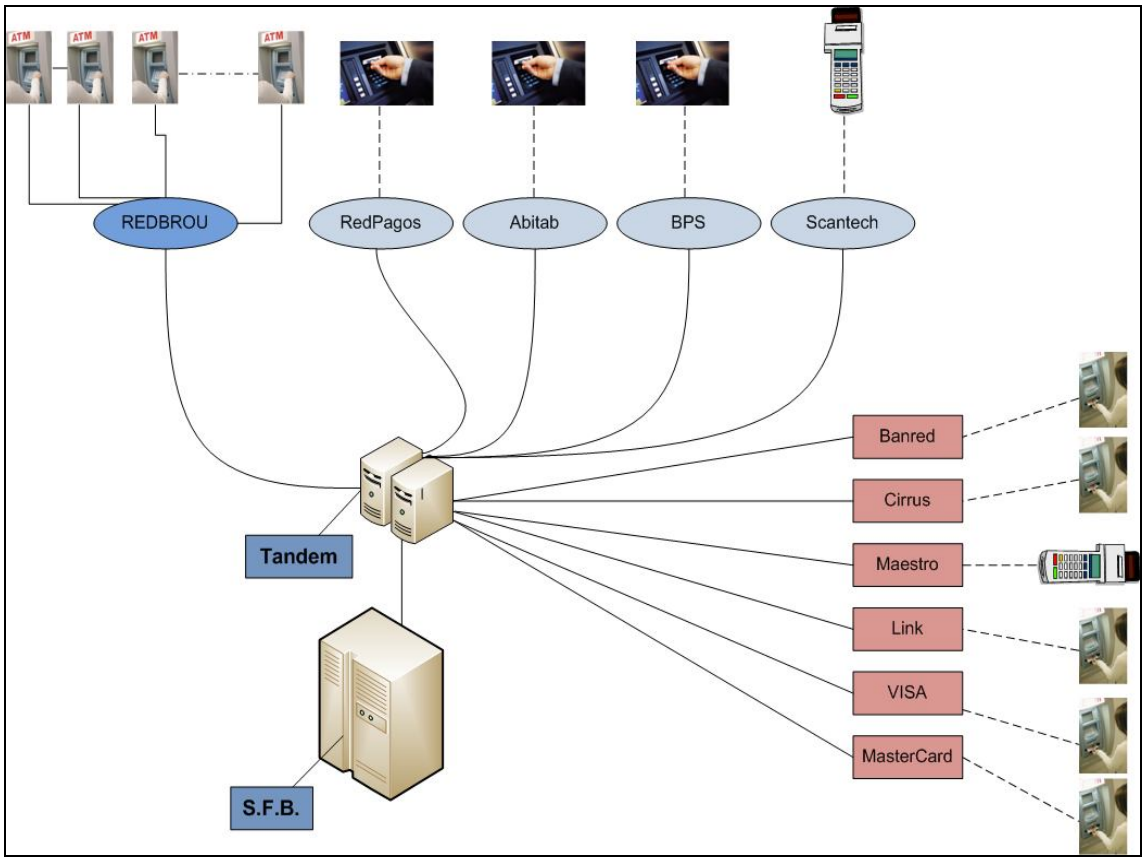
La función que brinda un servicio en respuesta a una llamada o petición desde un consumidor.

**Consumidor**

La función que consume el resultado del servicio provisto por un proveedor

## 14.4 - Módulo de Banca Electrónica de S.F.B.

El módulo de Banca Electrónica comprende a aquellas transacciones que pueden realizarse desde un cajero automático, o desde una terminal POS.
El siguiente diagrama describe la arquitectura del módulo en producción.




El BROU cuenta con una red propia, pero además cuenta con subredes (redes que sólo pueden operar con tarjetas de débito emitidas por el BROU), y está a su vez interconectado con redes locales e internacionales, lo que permite a clientes BROU utilizar otras redes, y a otros clientes utilizar la red de cajeros REDBROU.

Las transacciones que se registran en el sistema pueden dividirse en tres grupos:

Clientes propios que operan en REDBROU

Estas transacciones son registradas por SFB, ya que implican consultas o movimientos en cuentas de sus clientes, además de movimientos de dinero en sus cajeros automáticos.
En este caso, las transacciones habilitadas son Retiro, Depósito, Consulta de Saldo, Transferencia de Fondos y Cambio de PIN.

Clientes propios que operan en otras redes

En este caso SFB registra las consultas o movimientos en las cuentas de sus clientes, pero no hay una salida de dinero para el sistema. Estas transacciones generan una obligación del banco con la otra red.

En este caso solo es posible realizar Retiros y Consultas.

Clientes ajenos que operan en REDBROU

Este último caso es el único en el que SFB no autoriza la ejecución de la transacción, ya que no cuenta con los datos del cliente o la cuenta que está operando. De todas maneras la transacción queda registrada ya que genera un derecho del BROU para con la red a la que pertenece la tarjeta del usuario.
Es análogo al caso anterior, y únicamente es posible realizar Retiros y Consultas.

## 14.5 - Banca Electrónica en una arquitectura SOA

Para llevar el módulo de Banca Electrónica a una Arquitectura SOA, debemos identificar cada uno de los actores.

SFB sería el Proveedor de Servicios.

Los Servicios que publicaría son:

Retiro de Cuenta SFB
Consulta de Saldo de Cuenta SFB
Depósito en Cuenta SFB
Transferencia de Fondos entre Cuentas SFB
Cambio de PIN

(*) Todos estos servicios pueden ser requeridos por Clientes BROU desde un cajero perteneciente a un ATM de RedBROU. El Cambio de PIN es solo a efectos de registrar el cambio de estado, ya que en realidad el PIN es encriptado en el propio ATM, y el código encriptado es almacenado en el Tandem.

(**) Desde un cajero de otra red, un Cliente BROU tiene habilitados únicamente el Retiro y la Consulta de Saldo.

Retiro de Cliente Ajeno
Consulta de Saldo de Cliente Ajeno

(***) Estas transacciones son las realizadas por Clientes Ajenos en un ATM de RedBROU. No es SFB quien las autoriza, pero si deben quedar registradas a los efectos de realizar luego las compensaciones con las otras redes.

El Tandem se encargaría de la Orquestación.

En base al tipo de solicitud que recibe, determinaría que servicio consumir de SFB, ya sea para pedir la autorización de un retiro o transferencia, o consultar un saldo, o informar un cambio de PIN. Para las transacciones de Clientes Ajenos, son autorizadas por la otra red, y consume los servicios de SFB que registran esas operaciones (6 o 7).

Los ATM y las terminales POS son los Consumidores de Servicios, en base a las transacciones realizadas por los usuarios finales.

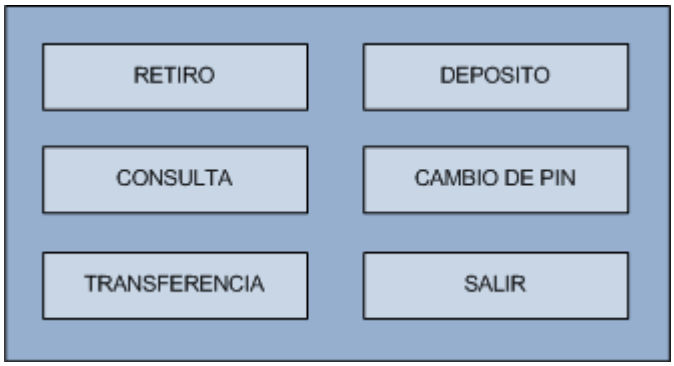
En resumen, este sería el catálogo de servicios disponibles:

| SERVICIO | PARA CLIENTES | DISPONIBLE EN | ACCIÓN EN SFB |
|---|---|---|---|
| Retiro Cuenta | BROU | Todas las Redes | Autoriza |
| Consulta Saldo | BROU | Todas las Redes | Informa |
| Transferencia | BROU | REDBROU | Autoriza |
| Depósito | BROU | REDBROU | Registro |
| Cambio de PIN | BROU | REDBROU | Registro |
| Retiro Cliente Ajeno | Otros Bancos | REDBROU | Registro |
| Consulta Saldo Cliente Ajeno | Otros Bancos | REDBROU | Registro |

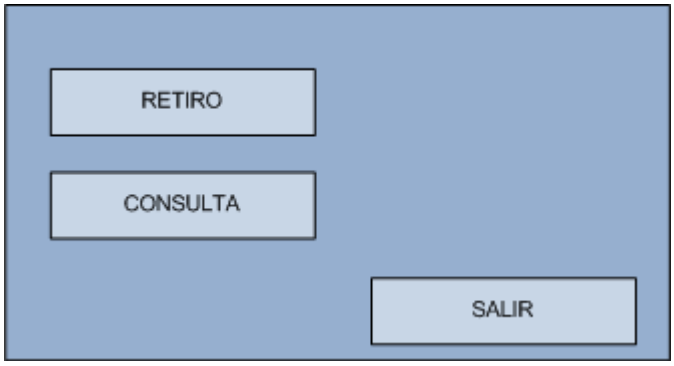
El retiro de cuenta esta disponible para los clientes REDBROU en todas las redes con las que está conectada. SFB es el que autoriza el retiro y lo hace efectivo.
El saldo de las cuentas de clientes REDBROU es informado por SFB.
La transferencia de fondos entre cuentas BROU es autorizada y hecha efectiva por SFB.
El depósito en cuentas SFB solo es registrado en el sistema, pero no impacta en línea, ya que es necesaria la intervención de un usuario que verifique el contenido del sobre.
Las claves personales no son almacenadas en SFB. El cambio de PIN solo se registra en SFB a modo informativo.

Las transacciones de Clientes Ajenos en REDBROU son autorizadas por el otro banco (al que pertenece la tarjeta del cliente), pero son registradas en SFB porque se genera un derecho con la otra RED (por el uso del cajero y por el monto del retiro en caso de una extracción).

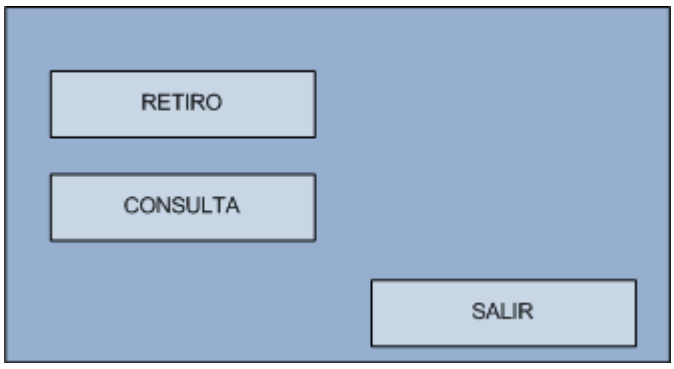
A nivel de los ATM, cuando un cliente BROU opera en un cajero perteneciente a REDBROU estas son las opciones que se le presentan:



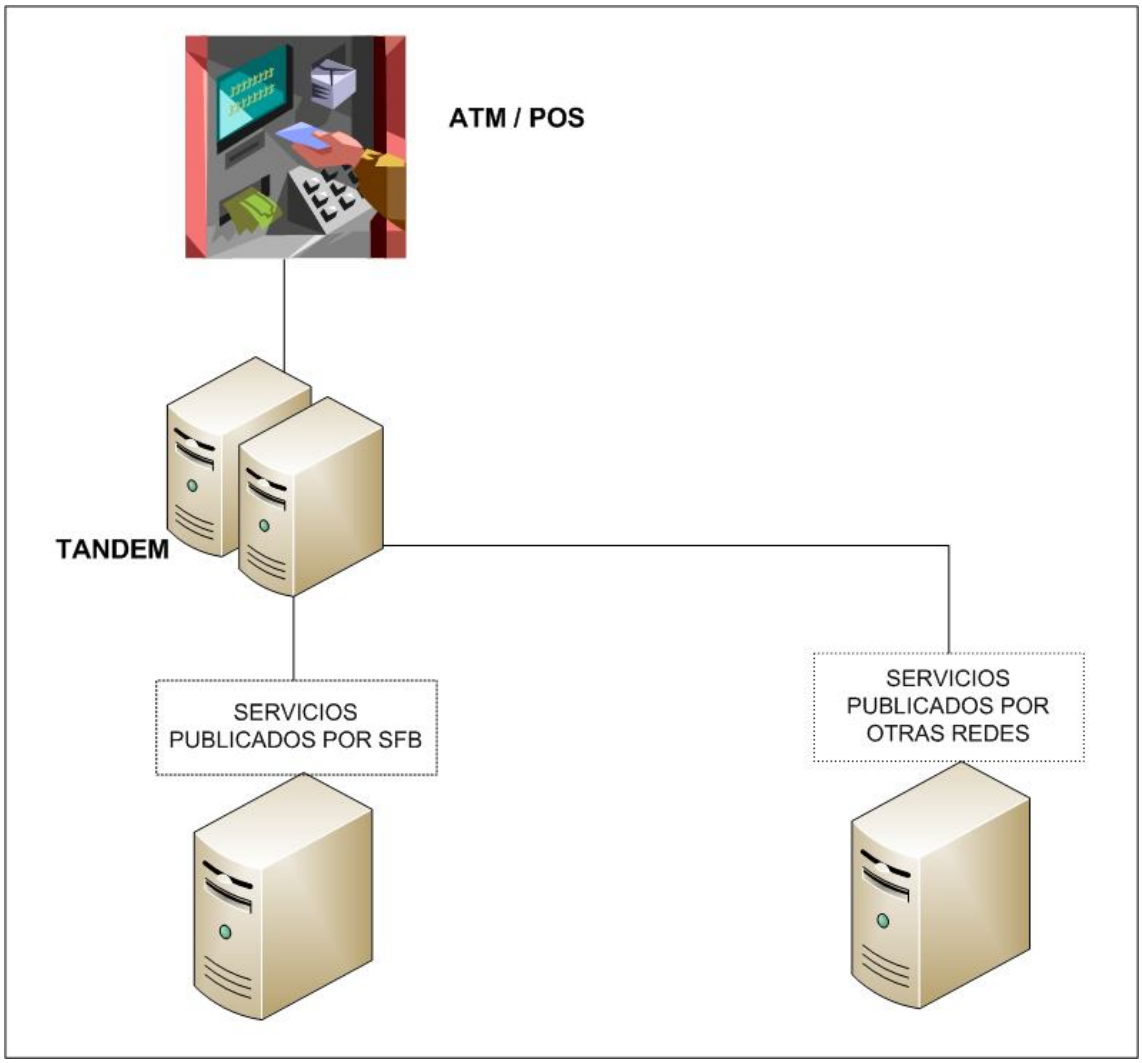

En caso de operar en un cajero de otra red las operaciones habilitadas son las siguientes:




Los clientes ajenos que operan en REDBROU tienen habilitadas las mismas operaciones (aunque consumen servicios diferentes de SFB):



La nueva arquitectura sería como se describe en la siguiente figura:




## 14.6 - Conclusiones

Es posible llevar la actual solución de banca electrónica del BROU hacia una arquitectura orientada a servicios.
Actualmente existe un programa que funciona como intermediario entre el equipo Tandem y el sistema SFB, traduciendo la mensajería propia del Tandem (ISO8583) a mensajería EAE y viceversa.
Al encapsular las operaciones de SFB y publicarlas como servicios pueden ser tanto consumidas por el Tandem como por otras tecnologías disponibles para el soporte de transacciones de ATM y POS, por lo que también se lograría independizarse de la plataforma Tandem.
La publicación de servicios desde SFB ya ha sido implementada para la solución de Internet banking (eBROU) y funciona correctamente para un volumen muy importante de transacciones, sin afectar los tiempos de respuesta.
Además permitiría reducir los costos de mantenimiento, siendo más adaptable a los cambios del negocio y facilitando la escalabilidad e interoperabilidad con otros sistemas.

# 15 – Manual de Usuario

## 15.1 – Propósito

Proveer al usuario de un manual de uso del prototipo implementado.

## 15.2 – Alcance

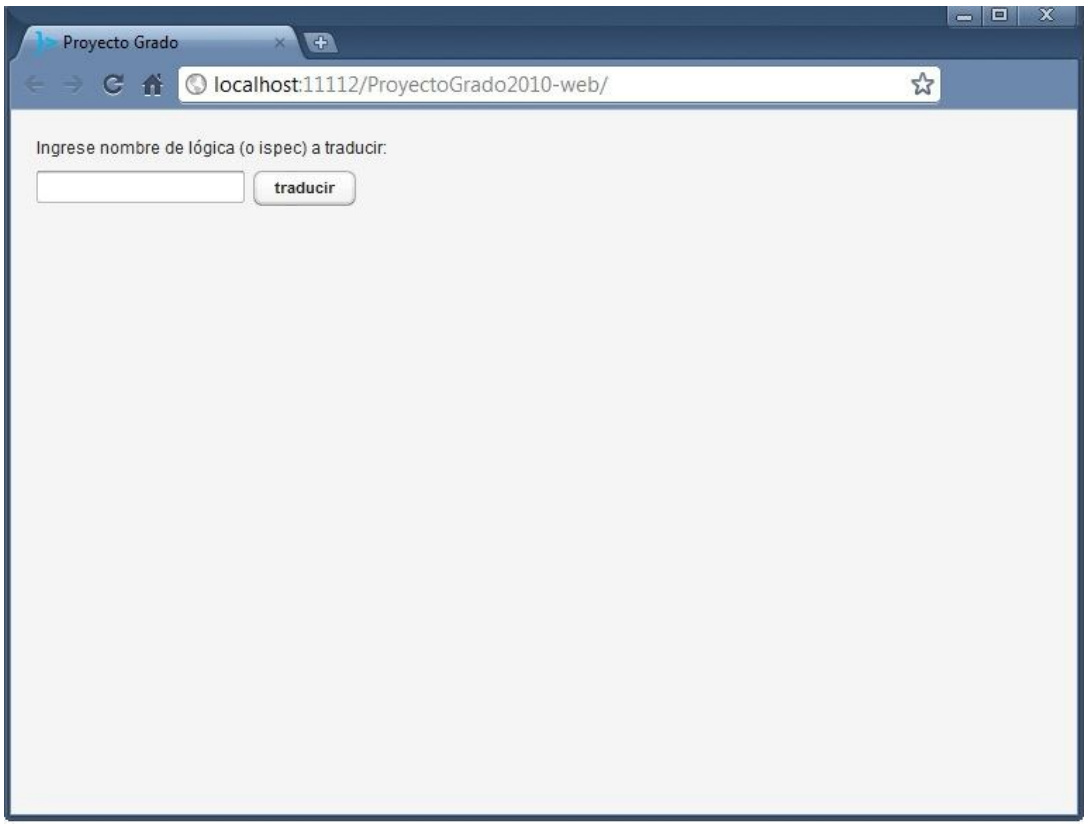
Explicar las funcionalidades provistas del prototipo JEE.

## 15.3 – Funcionalidades

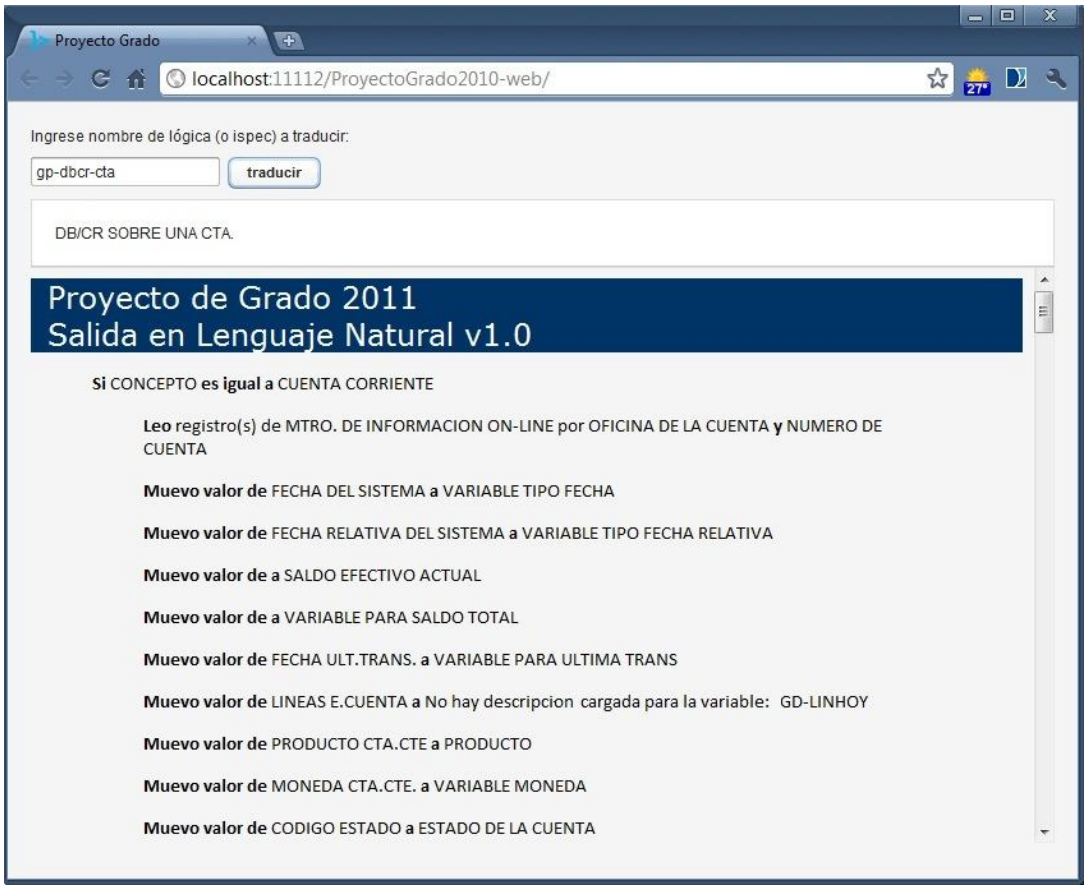
### 15.3.1 – Traductor de código

En cuanto a la implementación del prototipo, la única funcionalidad provista es la de traducir una lógica. Para esto se ingresa como parámetro de entrada el nombre de la lógica que se quiere traducir, y se obtiene como salida una serie de archivos html (desplegados en la misma pantalla) con la descripción de la lógica en lenguaje natural. También se traducen todas las lógicas que son invocadas por la lógica que se traduce, por eso se generan varios archivos.

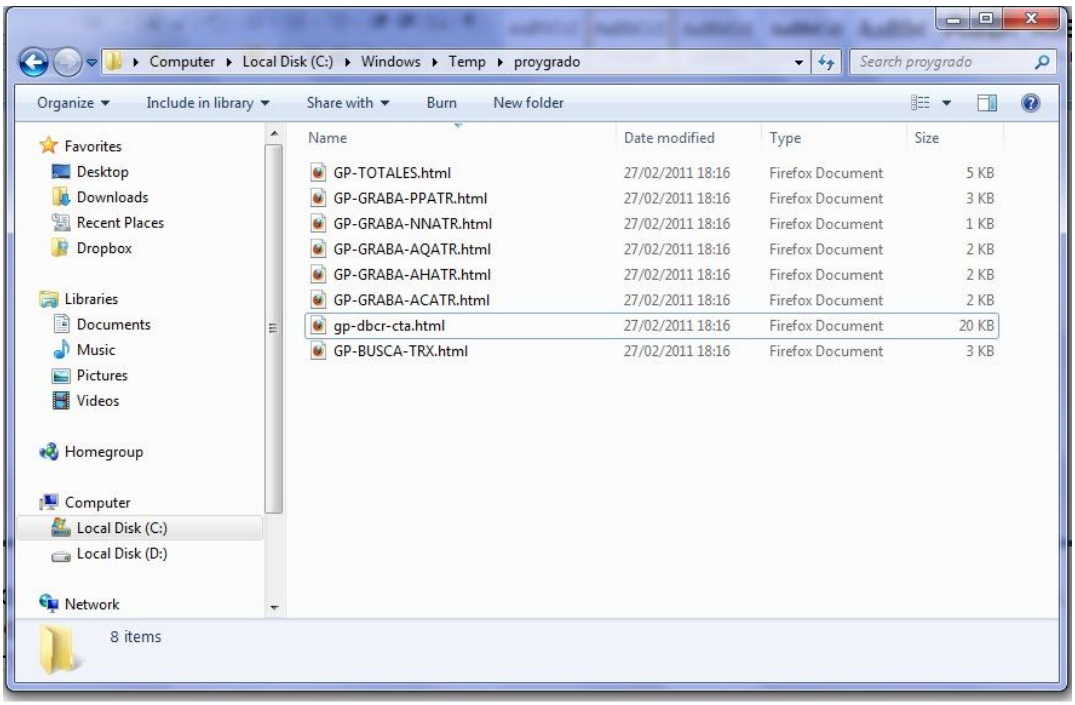
La pantalla es la siguiente:



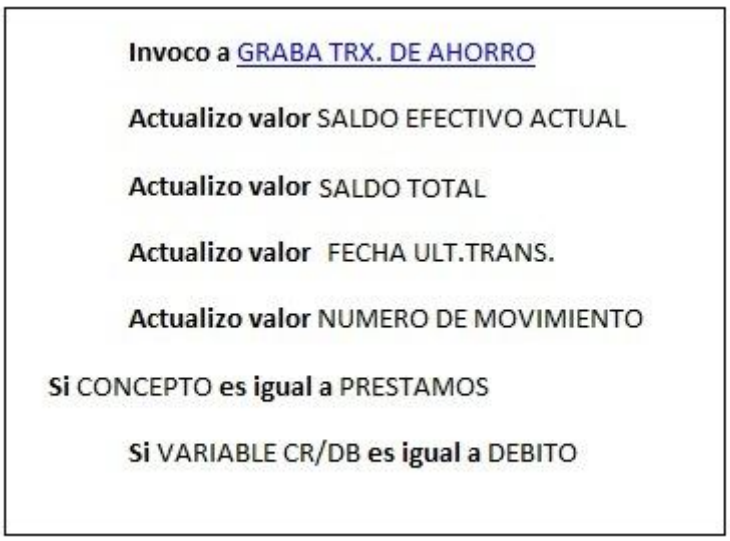
Luego de ingresar el nombre y apretar el botón *traducir*, se muestra la descripción que esté cargada en la lógica y en una ventana interior la traducción.



Lo que se muestra en esta ventana interior es simplemente el archivo html generado. Por defecto el directorio en donde se guardan los archivos generados es *c:\windows\temp\proygrado*\.




Si una lógica (o un ISPEC) invoca a otra lógica, esta otra lógica también se traduce y se genera el archivo html correspondiente. En el archivo origina se muestra un link a la descripción de la nueva lógica traducida, o sea, al archivo html generado para esta lógica.
Para el ejemplo anterior (lógica GP-DBCR-CTA) se invoca la lógica GP-GRABA-AHATR (cuya descripción es: “graba trx. de ahorro”) en el siguiente ejemplo



## 15.4 – Manejo de errores

Si se presiona el botón de traducir sin ingresar un nombre de lógica para traducir se muestra el siguiente mensaje de advertencia.


Mientras que si se ingresa el nombre de una lógica o ispec que no está cargada en el sistema, o que no existe, se muestra el siguiente mensaje de error.