Universidad Central “Marta Abreu” de Las Villas.
Facultad Matemática, Física y Computación
Licenciatura en Ciencia de la Computación

TRABAJO DE DIPLOMA

“Aplicación para Reglas de Restricción en Negocios”.

AUTOR: Alain Pérez Alonso.

TUTORES: MSc. Martha Beatriz Boggiano Castillo.
Dr. Ramiro Pérez Vázquez.

CONSULTANTE: Dra. Luisa González González.

“Año 50 de la Revolución”

3 Julio del 2008.

Dictamen.

Hago constar que el presente trabajo fue realizado en la Universidad Central “Marta Abreu” de Las Villas como parte de la culminación de los estudios de la especialidad de Ciencia de la Computación, autorizando a que el mismo sea utilizado por la institución, para los fines que estime conveniente, tanto de forma parcial como total y que además no podrá ser presentado en eventos ni publicado sin la autorización de la Universidad.

Firma del autor

Los abajo firmantes, certificamos que el presente trabajo ha sido realizado según acuerdos de la dirección de nuestro centro y el mismo cumple con los requisitos que debe tener un trabajo de esta envergadura referido a la temática señalada.

Firma del tutor

Firma del jefe del Laboratorio

Dedicatoria.

A mis abuelos,

A mis padres,

A mi novia,

A mi tío y tías,

A mi hermano,

A mis sobrinos,

A mis primos y primas,

Y a todos mis amigos.

Agradecimientos.

A mis tutores MSc. Martha Beatriz Boggiano Castillo y
Dr. Ramiro Pérez Vázquez por su asesoría y orientación en este trabajo.
A mi profesora consultante Dra. Luisa González González.
A los profesores del grupo de Base de Datos.
A todos los profesores que me formaron y enseñaron.
A todos los que de una u otra forma me han ayudado y han contribuido a la terminación de esta tesis.

Pensamiento.

Recibid mi enseñanza y no plata;
Y ciencia antes que el oro escogido.
Porque mejor es la sabiduría que las piedras preciosas;
Y todo cuanto se puede desear no es de compararse con ella.

Proverbios 8: 10 y 11.

Resumen.

Los problemas en la concepción de Sistemas de Información, debido al tratamiento informal de los requerimientos, han permitido considerar como factor de esencial importancia la forma en que se manejan los requisitos, enfocados actualmente como reglas de negocio.

Por tanto el propósito de este trabajo es automatizar las reglas de restricción, ganando con ello agilidad, transparencia y reducción del costo en un negocio. Esto implica crear un patrón para reglas, analizar los recursos que posibiliten su implementación y determinar su forma de almacenamiento.

Se modifica un patrón adaptándolo a las nuevas exigencias de las reglas, son elegidos los recursos de bases de datos para implementarlas, siendo necesario un compilador que genere este código, y se utiliza un lenguaje de marca para almacenar y validar las reglas.

Se concluye con la realización de un editor para reglas en lenguaje técnico, destinado a los analistas y productivo al negocio por brindar oportunidades. Este editor es capaz de abarcar todos los aspectos tratados en la investigación de forma sencilla y eficaz, resultando una primera versión modificable a las recomendaciones propuestas.

Abstract.

The problems in the conception of Information Systems, due to informal treatment of the requirements, have permitted to consider as an important essential factor the way these requirements are managed, focused at present as business rules.

Therefore our purpose in this work is to automate the restriction rules, getting agility, transparency and reduction of the cost in a business. This implicate the creation of a rules pattern, to analyze those resources that make possible its implementation and to determine their storage form.

A pattern is modified adapting it to the new rules demands, the resources of databases are chosen to implement them, being necessary a compiler that generates this code and markup language are used to store and validate the rules.

It is concluded with the realization of rules editor to be used in technical language, assigned to analysts and productive to business to offer opportunities. This editor is able to embrace all the aspects have been dealing with through the investigation in a simple and effective way, being a first amendable version to the proposed recommendations.

Contenido.

Ilustraciones.

Tablas.

Introducción.

Cuando un proyecto de software falla, el fracaso está determinado principalmente por la forma en que se desarrollan los sistemas. El centro de este problema es resultado del modo casual de tratar los "requisitos" (las declaraciones que dicen lo que un sistema de información hace). Estas declaraciones son capturadas típicamente de una manera rudimentaria, estructuradas pobremente, unidas al software únicamente por las ideas de analistas y diseñadores, además no están abiertas a exámenes o comprobaciones posteriores.

Puede considerarse que el inicio de la identificación de los requisitos son las reglas de negocio. Estas pueden ser consideradas como sentencias que permiten a los usuarios expertos definir políticas, condiciones y conocimientos del negocio en unidades pequeñas y aisladas.

La primera vez que aparece el término de Reglas de Negocio fue antes de 1984, cuando Daniel Appelton escribió un artículo llamado “The Missing Link”. En la actualidad existen diversos sistemas que gestionan reglas de negocios como Atenas y Rule Autor.

Los requisitos, enfocados hoy como reglas de negocio, tradicionalmente han sido realizados “a mano” por el programador en el código fuente del programa, en los objetos de bases de datos de la aplicación, o en ambos.

Adicionando la automatización a este proceso se evitan fallas y, de este modo, se logra importantes reducciones de tiempo y costo. Para automatizar estos “requisitos” sería necesario un sistema encargado de darles funcionalidad, logrando implementarlos en un lenguaje y mantenerlos almacenados para su manipulación.

Estudios actuales tienden a almacenar las reglas en un repositorio, siendo manejadas por un motor de reglas independiente al Sistema de Información del negocio. Por tanto las restricciones no se codifican en el sistema y este sólo puede realizar accesos necesarios al repositorio. Las reglas son tratadas fuera de la base de datos del negocio.

La idea de las reglas de negocio, como definición de la política deseada, podría proporcionar una solución por su posible tratamiento en bases de datos y aplicaciones flexibles, fácilmente modificables. Específicamente, las reglas de tipo restricción, diseñadas para evitar violaciones y frecuentemente delimitar la forma y el valor que puede tomar la información en el negocio.

Este trabajo propone considerar una solución intermedia mediante un editor que permite insertar automáticamente las reglas de restricción en la base de datos del negocio, utilizando un repositorio para almacenarlas y un compilador para generarlas.

Problema de investigación:

¿Cómo automatizar la generación de reglas de negocio, tipo restricción en una base de datos?

Objetivo general:

Automatizar la generación de reglas de negocio, tipo restricción, en una base de datos.

Objetivos específicos:

- Definir un patrón de reglas de negocio, tipo restricción.
- Analizar los recursos en base de datos que posibiliten la implementación de reglas de negocio, tipo restricción.
- Crear una aplicación que permita editar reglas de negocio, tipo restricción, en lenguaje técnico para su generación automática.

Preguntas de investigación:

- ¿Cuál patrón de reglas de negocio, tipo restricción, utilizar?
- ¿Qué recursos en base de datos posibilitan la implementación de reglas de negocio, tipo restricción?
- ¿De qué forma almacenar reglas de negocio, tipo restricción?
- ¿Cómo lograr la generación automática de reglas de negocio, tipo restricción?

Hipótesis de investigación:

- Se puede, mediante recursos de bases de datos, implementar reglas de negocio, tipo restricción y almacenarlas eficazmente a través de un Lenguaje de Marca.

Este estudio tributa a un trabajo de equipo en el departamento de Base de Datos del Centro de Estudios Informáticos (CEI), en la Universidad Central de las Villas, que persigue la producción de un sistema gerenciador de reglas de negocio para transplante renal. Sin embargo la idea de esta investigación se extiende al ser aplicable a cualquier sistema.

Para generar automáticamente reglas de restricciones en la base de datos del negocio, se considera que previamente deben estar almacenadas en un repositorio, el cual constituye la salida de un proceso de edición en lenguaje natural (informal) y transformado a un lenguaje intermedio (técnico) que es compilado para generar el script de la regla (formal) en la base de datos.

Por esto, se hace una propuesta del repositorio para reglas de restricción y un editor de estas en lenguaje técnico que suple la ausencia del editor de reglas en lenguaje informal para el experto del negocio.

El Capítulo I recoge aspectos generales sobre las reglas de negocios, se exploran los diferentes tipos de reglas y sus disímiles formas de implementación, y se examina un formato para el almacenamiento de estas.

El Capítulo II recoge los puntos más relevantes sobre la implementación de las reglas de negocio, tipo restricción. Se analizan las mejores variantes para la representación y generación de estas reglas, así como consideraciones para su edición.

El Capítulo III trata del desarrollo del editor para reglas de negocio, tipo restricción. Consta de tres secciones, una primera en la cual se aborda el proceso de compilación del lenguaje. La segunda sección presenta el esquema con el cual se valida el repositorio. La tercera y última sección discurre sobre la aplicación en sí, sus casos de uso, su estructura y funcionamiento

Capítulo I:

CONSIDERACIONES GENERALES SOBRE REGLAS DE NEGOCIO.

I. CONSIDERACIONES GENERALES SOBRE REGLAS DE NEGOCIO.

Este capítulo recoge aspectos generales sobre las reglas de negocios, así como lo necesario a considerar para su automatización en determinado negocio. Se exploran los diferentes tipos de reglas y sus disímiles formas de implementación, especialmente, profundizando sobre los recursos de Base de Datos y se examina, igualmente, un formato para el almacenamiento sencillo y eficaz de dichas reglas.

Se muestra cómo se insertan las reglas de negocios en un sistema y las diferentes vías para ello, siempre conscientes de las ventajas potenciales de su utilización.

I.1. ¿Qué son las Reglas de Negocio?

Una regla de negocio, básicamente, es una declaración compacta sobre un aspecto del negocio, usando un lenguaje simple, inequívoco, accesible a todas las partes interesadas: el dueño del negocio, el analista, el arquitecto técnico, y así sucesivamente (Morgan, 2002). Este autor enumera características deseables en las declaraciones de reglas. Estas características universales se aplican a cualquier lenguaje o dominio de aplicación.

- Atómico: no pueden ser descompuestas sin que se pierda información.
- No ambiguo: tienen solamente una obvia interpretación
- Compacta: típicamente son frases cortas.
- Consistente: juntas, ellas proporcionan una única y coherente descripción.
- Compatible: usan las mismas condiciones en el resto del modelo de negocio.

Nombre	Regla
R1:	“El salario mínimo de una enfermera es $357”.
R2:	“Un cirujano no debe atender más de 3 pacientes”.

Tabla I.1 Ejemplos de Reglas de Negocios.

I.2. Beneficios de usar Reglas de Negocio.

Varios son los beneficios que pueden derivarse del uso de Reglas de Negocio. De todos, según Lowenthal (Lowenthal, 2005) los tres más importantes son:

- Agilidad: respuesta simple y rápida a los requisitos dinámicos.
- Reducción del Costo: bajo costo para crear o actualizar las partes de aplicaciones que implementan las políticas del negocio.
- Transparencia: las reglas permiten fácilmente la auditoría que los servicios de software llevan a cabo en sus políticas de negocios correspondientes.

I.3. Formas de expresión de las Reglas de Negocio.

De acuerdo con Morgan (Morgan, 2002) las reglas de negocio pueden expresarse de diversas formas, principalmente, según su depuración a lo largo del sistema o incluso en la manera en que se introduzcan a este. Es necesario señalar que existen varios modos a considerar, pero fundamentalmente, se definen tres niveles:

- Informal: este nivel proporciona una sentencia en lenguaje natural, sin un rango limitado de parámetro, tal y como el cliente del negocio desee.
- Técnico: este nivel combina referencias a datos estructurados, operadores y restricciones con el lenguaje natural, nivel intermedio entre la entrada de la regla y su implementación.
- Formal: este nivel proporciona sentencias conforme a una sintaxis definida, más cercana a propiedades matemáticas específicas. Este nivel proporciona la funcionalidad automática de la regla.

I.4. Tipos de Reglas de Negocio.

Existen muchos tipos de reglas de negocio. Debido a su diversidad y complejidad los autores tienden a agruparlas y clasificarlas siguiendo diferentes puntos de vista, pero con un objetivo común; por tanto un estudio de estas clasificaciones arroja que en realidad son similares y se complementan unas a otras.

I.4.1. Clasificación según Solivares.

Una buena clasificación de reglas de negocios es la que ofrece Solivares (P. A. Solivares; citados en (Besembel and Chacón)). De esta se muestran los cinco grupos más interesantes.

Reglas del Modelo de Datos:

El primer grupo de reglas de negocios, engloba todas aquellas encargadas de controlar que la información básica almacenada para cada atributo o propiedad de una entidad u objeto sea válida.

Reglas de Relación:

Otro grupo importante de reglas incluye todas aquellas que controlan las relaciones entre los datos. Estas reglas especifican, por ejemplo, que todo pedido debe ser realizado por un cliente, y el mismo debe ser atendido. Además, una vez que un cliente haya hecho algún pedido, se deberá garantizar no eliminarlo, a menos que previamente se eliminen todos sus pedidos.

Reglas de Restricción:

Otro grupo es el compuesto por las reglas que restringen los datos contenidos en el sistema. Nótese que este grupo se solapa en cierto modo con las reglas del modelo de datos, pues aquellas también impiden la introducción de datos erróneos. La diferencia estriba en que las reglas de restricción condicionan el valor de los atributos o propiedades de una entidad más allá de las restricciones básicas sobre las mismas existentes.

Intervalos Temporales:

Ocurren entre dos eventos específicos, por ejemplo: el cliente llama entre el envío de la oferta y 30 días después.

Periódicos:

Son aquellos que ocurren en intervalos fijos de tiempo, por ejemplo: cada quince días o cada cinco órdenes de compra.

I.4.2. Clasificación según Lowenthal.

Según Lowenthal (Lowenthal, 2005) se crea la regla de negocio usando una sentencia de declaración que al menos debe tener un sujeto, un verbo y un objeto; las palabras usadas deben tener un único significado dentro del dominio del negocio. Si son nombres, deben aparecer en el modelo de datos del negocio como entidades o atributos. Si hay verbos, estos también deben aparecer en el modelo de datos como parte de las interrelaciones entre entidades. Si estas palabras no aparecen en el modelo, entonces deben presentarse en el glosario.

Los posibles tipos de reglas de negocio son:

- Restricciones: estas reglas previenen de posibles violaciones, principalmente en su forma o valor.
- Cómputo: reglas que calculan un valor aritmético o booleano basado en una fórmula.
- Acciones: (al ocurrir un evento o satisfacer una condición)
    - Habilitar o deshabilitar un proceso u otra regla.
    - Ejecutar un proceso.
    - Crear, modificar o eliminar datos.

I.4.3. Clasificación según Morgan.

Por el pensamiento de Tony Morgan (Morgan, 2002), la forma más conveniente de crear sentencias de regla es seleccionar patrones adecuados desde una pequeña lista disponible. Por ejemplo: <sentencia>:= <sujeto> debe <restricción>

Agrega que no existe un estándar de cómo hacer reglas de negocio pero sí es posible hacer algunas recomendaciones. Los patrones pueden reflejar las formas o tipos de problemas que un sistema automatizado pretende negociar. Morgan relaciona varios como los de Restricción, Clasificación, Cálculo y Enumeración. De estos son de interés en este trabajo los que pertenecen a los tipos de restricciones, aunque el patrón de Clasificación puede ser útil.

Los convenios utilizados para los patrones son los siguientes:

- Paréntesis ( ) Encierran un grupo de ítems.
- Corchetes [ ] Encierran elementos opcionales.
- Barra Vertical | Separa términos alternativos.
- Corchetes Angulares <> encierran términos especiales como los mostrados en la siguiente tabla.

Elemento	Significado
<determinante>	Es el determinante para cada sujeto, por ejemplo: Una, Uno, El, La, Cada, Todos.
<sujeto>	Es una entidad reconocible del negocio, tal como objetos, un nombre de Rol o una propiedad de un objeto. La entidad puede ser cualificada por otros elementos descriptores, tales como la existencia en un estado particular, relacionada con una aplicabilidad específica de la regla.
<característica>	El comportamiento del negocio a tomar lugar o interrelación que debe ser establecida.
<hecho>	Interrelación entre términos identificables en el modelo de hechos. Esta puede ser cualificada por otro elemento descriptor relativo a identificar la aplicabilidad de la regla precisada.
<lista de hechos>	Una lista de <hechos> elementos.
<m>,<n>	Son parámetros numéricos.

Tabla I.2 Elementos Variables de los Patrones.

Patrones de Reglas:

- Restricción básica: es el más común entre los patrones para reglas de negocio, establece una restricción sobre el sujeto de una regla.
  <determinante> <sujeto> [no] (debe | tiene) <característica>
  [(si | a menos que) <hecho>].

- Lista de Restricción: este patrón también restringe al sujeto, delimitando la cantidad de hechos verdaderos tomados de una lista.
  <determinante> <sujeto> [no] (debe | tiene) <característica>
  (si | a menos que) como mínimo <m> [no más de <n>] de las siguientes es verdadera: <lista de hechos>.

- Clasificación: establece la definición para un término del negocio.
  <determinante> <sujeto> [no] es definido como <clasificación>
  [(si | a menos que) <hecho>].

I.5. Consideraciones sobre la Notación Punto.

Muchos autores exigen que las reglas de negocio deban declararse en algún lenguaje formalizado (C. J., 2000; Demuth and Hussmann, 1999; Ross, 1998; Ross, 1997; citados en Zimbrão). Se han establecido diagramas y otras herramientas proporcionadas por Unified Modeling Language (UML) como estándares para la modelación de sistemas de software. Una de las herramientas sugerida por UML es el Object Constraint Language (OCL). OCL es el lenguaje UML para la especificación de restricciones. Este es más conciso que SQL, principalmente, debido al estilo del punto de navegación. Es una herramienta a usar en fases tempranas de análisis del sistema.

Según expone Morgan (Morgan, 2002) habrá que decidir cómo referirse a los atributos de objetos. Por ejemplo, el objeto Paciente puede tener un atributo llamado Edad, un estilo es confiar en las estructuras normales disponible en español o cualquier idioma que se emplee: “la Edad del Paciente”, “del Paciente su Edad” o algo similar.

La otra opción principal es usar la notación punto, produciendo expresiones como Paciente.Edad. Además se considera el uso de esta notación, para especificar las relaciones entre los objetos del negocio.

Esta alternativa es mucho más precisa y puede usar los nombres del diagrama de clases, pero no es aceptable en el lenguaje informal del negocio, se usa para las expresiones en lenguaje técnico.

I.6. Implementaciones de Reglas de Negocio.

Las reglas se pueden implementar de formas disímiles e incluso pueden existir diferentes técnicas para una misma regla. Morgan (Morgan, 2002) muestra algunas de las posibles a utilizar.

Lenguajes de Programación:

Las reglas incorporadas en el código del programa, probablemente, son la vía de aplicación más común. Es posible usar las sentencias normales de un lenguaje de programación para implementar una regla. El requisito mínimo es tener una manera de seleccionar, entre las ramas del código, una alternativa basada en una condición dada. Esto es bastante fácil de satisfacer en cualquier lenguaje de programación, aunque los detalles pueden variar de uno a otro.

Ejemplo en Object Pascal:

```
    Procedure R10 (paciente: TPaciente);
     Begin
      ...
      If paciente.edad > 18 Then
       //Código si la restricción es satisfecha
      Else
       //Código si la restricción no es satisfecha
      ...
End.
```

Scripts:

La principal ventaja de los scripts es que al encontrarse en la aplicación cliente son muy veloces, aunque sufren de algunos puntos negativos. De estos el fundamental es la dificultad en su manejo y modificación.

Ejemplo en Java Script:

```
<script language="JScript">
 function chequear(obj) {
  if (obj.Value < 357) {
   alert("El salario mínimo de una enfermera es $357");}}
</script>
```

Bases de Datos:

Es probable que las reglas de negocio encajen más naturalmente dentro de la base de datos, donde pueden tener un contacto más directo con los datos. Los tipos de tecnología más usados son Oracle, SQL Server y Postgres. Otros productos de base de datos correlativos tienen ampliamente capacidades similares, por lo que no es nada complejo extender esta investigación a otros lenguajes. En el caso del Servidor SQL, puede programarse en un dialecto llamado Transact-SQL.

Lo más habitual es querer usar las reglas de negocio respecto a palabras funcionales que estén alrededor de lo básico (CREATE, READ, UPDATE y DELETE). Estos mecanismos son comunes a todos los servicios de datos.

I.7. Recursos del SQL.

El lenguaje SQL estándar (SQL 99) y particularmente la implementación para SQL Server 2000 (Transact-SQL) ofrecen varios recursos de manejo de datos que se analizan en esta sección para seleccionar a la postre los más convenientes en la implementación de reglas de negocio, tipo restricción.

Restricciones (Constraints) CHECK:

Las restricciones CHECK exigen la integridad del dominio mediante la limitación de los valores que puede aceptar una columna. Además, determinan los valores válidos a partir de una expresión lógica que no se basa en datos de otra columna. Se puede crear una restricción CHECK con cualquier expresión lógica (booleana) que devuelva TRUE (verdadero) o FALSE (falso) basándose en operadores lógicos.

Es posible aplicar varias restricciones CHECK a una sola columna. Éstas se evalúan en el orden que fueron creadas. También es permitido aplicar una sola restricción CHECK a varias columnas si se crea al nivel de la tabla.
(SQLServer2000, 2004).

Desencadenadores (Triggers):

Los desencadenadores de Microsoft SQL Server 2000 son una clase especial de procedimiento almacenado definido para la ejecución automática al emitirse una instrucción UPDATE, INSERT o DELETE en una tabla o una vista. Son una herramienta eficaz que pueden utilizar los sitios para exigir automáticamente las reglas comerciales cuando se modifican los datos. Amplían la lógica de comprobación de integridad, valores predeterminados y reglas de SQL Server, aunque se deben utilizar las restricciones y los valores predeterminados siempre que estos aporten toda la funcionalidad necesaria.

Los desencadenadores pueden consultar otras tablas e incluir instrucciones SQL complejas. Son especialmente útiles para exigir reglas o requisitos complejos. También son útiles para exigir la integridad referencial, que conserva las relaciones definidas entre tablas (SQLServer2000, 2004).

Vistas:

Las vistas suelen utilizarse para centrar, simplificar y personalizar la percepción de la base de datos para cada usuario. Pueden emplearse como mecanismos de seguridad, que permiten a los usuarios obtener acceso a los datos por medio de ella, pero no les conceden el permiso de obtener acceso directo a sus tablas subyacentes.

Las vistas también se pueden utilizar para realizar particiones de datos y para mejorar el rendimiento cuando estos se copian. Además, permiten a los usuarios centrarse en datos de su interés y en tareas específicas de las que son responsables. Los innecesarios pueden quedar fuera de la vista; de ese modo, también es mayor la seguridad de los datos, dado que los usuarios sólo pueden ver los definidos en la vista y no los que hay en la tabla subyacente (SQLServer2000, 2004).

Funciones definidas por el usuario:

Microsoft SQL Server 2000 (SQLServer2000, 2004) permite crear funciones definidas por el usuario. Al igual que cualquier función, son rutinas que devuelven valores. Dependiendo del tipo de valor que devuelven, estas pueden entrar en una de las tres categorías siguientes:

- Funciones que devuelve una tabla de datos actualizable: si una función definida por el usuario contiene una única instrucción SELECT y dicha instrucción es actualizable, el resultado tabular devuelto por la función también es actualizable.
- Funciones que devuelve una tabla de datos no actualizable: si una función definida por el usuario contiene varias instrucciones SELECT, o una instrucción SELECT no actualizable, el resultado tabular devuelto por dicha función no es actualizable.
- Funciones que devuelve un valor escalar: las funciones definidas por el usuario pueden devolver valores escalares.

Procedimientos Almacenados:

Los procedimientos almacenados pueden facilitar en gran medida la administración de la base de datos y la visualización de información sobre esta y sus usuarios. Son una colección precompilada de instrucciones SQL e instrucciones de control de flujo opcionales, almacenadas bajo un solo nombre y procesadas como una unidad. Estos se guardan en una base de datos, se pueden ejecutar desde una aplicación y permiten variables declaradas por el usuario, ejecución condicional y otras funciones eficaces de programación.

Los procedimientos almacenados pueden contener flujo de programas, lógica y consultas a la base de datos; aceptar parámetros y proporcionar sus resultados, devolver conjuntos de resultados individuales o múltiples y devolver valores.

Se pueden utilizar procedimientos almacenados para cualquier finalidad que requiera la utilización de instrucciones SQL, con estas ventajas:

- Ejecución de una serie de instrucciones SQL en un único procedimiento almacenado.
- Referenciar a otros procedimientos almacenados desde el propio procedimiento almacenado, con lo que se puede simplificar una serie de instrucciones complejas.
- El procedimiento almacenado se compila en el servidor cuando se crea, por tanto, se ejecuta con mayor rapidez que las instrucciones SQL individuales.

(SQLServer2000, 2004).

Assertions:

Las assertions son un tipo de restricción especial que se puede especificar en SQL sin que deba estar asociada a una tabla en particular, como es el caso de los constraints (Mota, 2005). Generalmente son utilizados para describir restricciones que afectan a más de una tabla. Como los constraints sólo pueden establecerse sobre tuplas de una tabla, las assertions son útiles cuando es necesario especificar una condición general de la base de datos que no se puede asociar a una tabla específica. Por esta característica de poder especificar una assertion para toda la base de datos y no para una tabla en particular, se le llama restricción autónoma. Esto tiene grandes ventajas a nivel conceptual, pero las hace difíciles de implementar.

I.8. Repositorio para Reglas de Negocio.

Para la conservación de las reglas de negocio resulta vital un repositorio de reglas (Morgan, 2002) pues es necesario conocer de manera eficiente y sencilla la información sobre el estado del negocio y tener un acercamiento formal para responder interrogantes como:

- ¿Qué reglas se han definido, junto con su estado actual?
- ¿Qué reglas están contenidas en una regla o relacionadas con esta de otra manera?
- ¿Cuál es el autor original de una regla y el individuo que hizo el último cambio?
- ¿Qué reglas hacen uso de condiciones particulares?

Incluso tener otras informaciones adicionales sobre la realización de las reglas como:

- Encontrar todas las reglas que tienen implementaciones múltiples (y dónde).
- Identificar todas las reglas que tienen un tipo particular de implementación.
- Localizar el lugar donde se realizó particularmente una regla.

Estas funciones no significan ser exhaustivo, pero es obvio que se necesita un recurso de información estructurado para catalogar las reglas. Esto es fundamental pues el repositorio viene siendo como el organizador del sistema, donde se extraen e insertan las reglas en sus diversos formatos y es punto clave para el mantenimiento de la consistencia.

I.9. Los Lenguajes de Marcas.

Es vital descubrir un lenguaje capaz de conservar las reglas de negocio, un lenguaje con características deseables, que permita elaborar un repositorio para dar respuesta a las necesidades del sistema. Los lenguajes de marcas constituyen una buena opción.

I.9.1. Documentos XML (EXTENSIBLE MARKUP LANGUAGE).

En 1996, el World Wide Web Consortium (W3C) comenzó a desarrollar un nuevo lenguaje de marcado estándar más sencillo de utilizar que Standard Generalized Markup Language (SGML) pero con una estructura más rígida que la de HTML. El W3C estableció el Grupo de Trabajo de XML (XWG -XML Working Group) para comenzar el proceso de creación de XML**.** Sus ventajas, y componentes básicos se relacionan a continuación (Sturm).

Ventajas de XML:

- Es internacional.
- Puede estar estructurado.
- Sus documentos se pueden construir por composición
- Puede ser un contenedor de datos.
- Proporciona flexibilidad.
- Es sencillo de utilizar.
- Posee formatos estándar.

Los componentes básicos de un documento XML son: elementos (elements), y atributos (attributes).

Elementos:

Los elementos se utilizan para marcar las secciones de un documento XML. Un elemento de XML tiene la forma siguiente.

```
<NombreDeElemento> Elemento </NombreDeElemento>
```

El elemento aparece entre las etiquetas de XML.

Atributos:

El atributo es un mecanismo para agregar información descriptiva a los elementos. Por ejemplo al no conocer si el peso del paciente se mide en libras o en Kilogramos, se agrega un atributo unit al que se le asigna la unidad de medida:

```
<PesoPaciente unit="Libras"> 55 </PesoPaciente>
```

I.9.2. Esquemas para documentos XML.

Un esquema XML especifica la estructura válida para un tipo de documento XML el cual se dice que es válido, si está bien formado y se verifica la gramática que describe el contenido del documento.

Las definiciones de tipo de documento (DTD Document Type Definition) como forma de definir reglas para un documento XML es un método excelente, aunque las DTD presentan algunos problemas. Los esquemas (schemas) se inventaron para resolverlos.

Al contrario de las DTD, que disponen de su peculiar sintaxis propia, 1os esquemas de XML están escritos en XML. Además de proporcionar la misma información que las DTD, 1os esquemas permiten especificar tipos de datos, utilizar espacios de nombres y definir intervalos de valores para 1os atributos y 1os elementos.

I.10. Conclusiones Parciales.

En este capítulo se ha brindado un cuadro teórico sobre las reglas de negocio, haciendo hincapié en las de tipo restricción, de las cuales se ofreció un patrón que las define para su posible implementación y almacenamiento adecuado. Esto proporciona un conocimiento preliminar sobre los objetivos generales de este trabajo, y facilita la comprensión de las decisiones a tomar posteriormente. Se estudiaron los recursos de Base de Datos que pudieran ser empleados para implementar reglas y se introdujeron los documentos XML con sus esquemas para la concepción del repositorio de reglas.

Capítulo II:

PROPUESTA DE IMPLEMENTACIÓN PARA REGLAS DE NEGOCIO, TIPO RESTRICCIÓN.

II. PROPUESTA DE IMPLEMENTACIÓN PARA REGLAS DE NEGOCIO, TIPO RESTRICCIÓN.

En este capítulo se recogen los puntos más relevantes sobre la implementación de las reglas de negocio, tipo restricción. Se analizan las mejores variantes para la representación y generación de estas reglas, así como consideraciones para su edición.

Un patrón es creado. De este se detallan sus componentes y su vínculo con la generación automática de la regla, definiendo minuciosamente los recursos de las base de datos que permiten su ejecución.

Igualmente se diseña un repositorio que sea capaz de almacenar este patrón y su implementación en el negocio. Además, mediante figuras y ejemplos, se explica la estructura de este.

II.1. Modelo y Definiciones.

De las diversas formas de clasificaciones de las reglas vistas en el Capítulo I se tomaron, como modelo, las reglas de tipo restricción (Morgan, 2002). Este autor ofrece una definición detallada, que se ajusta más a la idea de implementación y brinda las características óptimas para un lenguaje de restricción de objetos.

A este modelo se le realizaron modificaciones en función de mejorarlo y adaptarlo a la necesidad de obtener efectos palpables. Resultando el patrón de restricción que sigue (manteniendo sus convenios):

<determinante> <sujeto> (no puede tener <características>) |
(puede tener <características> sólo si <hechos>).

Elemento	Significado
<determinante>	Es el determinante para cada sujeto, por ejemplo: Una, Uno, El, La, Cada, Todos. Según el mejor sentido en la redacción.
<sujeto>	Es un elemento de la Base de Datos del negocio, tal como entidades y objetos. La entidad puede ser cualificada por otros atributos descriptores, tales como la existencia en un estado particular o relacionada con una aplicación específica de la regla.
<características>	Describe las características del sujeto en el negocio, tanto internas como relacionadas con otras entidades.
<hechos>	Hechos relativos al estado o comportamiento de la Base de Datos del negocio, incluyendo o no al sujeto.

Tabla II.1 Elementos del Patrón.

II.1.1. Restricción básica y lista de restricción.

En la definición de Reglas de Negocios, tipo restricción, según la filosofía de Morgan (Morgan, 2002) se establecen dos patrones principales (Restricción Básica y Lista de Restricción). Ambos muy parecidos en su estructura, la única diferencia notable es que la lista de restricción permite incluir más de un hecho sin relaciones entre sí, a los cuales se les imponen restricciones numéricas. Así se flexibiliza en alguna medida este patrón y de manera general las prohibiciones, para lograr reglas poco más complejas, necesarias en los negocios.

Sin embargo, fueron imprescindibles las modificaciones a estos patrones de estudio. Variaciones inducidas por la rigidez de sus definiciones al enfrentarse a un negocio real, principalmente por seguir la máxima de atomicidad al permitir una sola característica, surgiendo la dificultad de aceptar reglas en las cuales el <sujeto> necesite más especificaciones, lógicas en todo negocio, por ejemplo:

Lenguaje Natural:
Un Paciente no puede tener Donantes Potenciales con más de 4 Exámenes Físicos o tener en su Evolución una temperatura máxima mayor de 42 grados.

Como Tony Morgan (Morgan, 2002) afirma se debe evitar la utilización de las conjunciones “o”, “y” para lograr reglas sencillas y fáciles de mantener, tratando de separarlas en reglas más pequeñas, según sea posible:

Lenguaje Natural:

- Un Paciente no puede tener Donantes Potenciales con más de 4 Exámenes Físicos.
- Un Paciente no puede tener una temperatura máxima de 42 grados.

Intencionalmente, en este ejemplo es sencilla la división por el uso del “o” exclusivo, pero en el caso de la utilización imprescindible del “y” no es tan sencillo encontrar una forma de dividirla en dos reglas, donde cada una tenga en cuenta a la otra. Es por esto que el modelo de Morgan (Morgan, 2002) no sugiere usar conjunciones; sin embargo, este trabajo extiende el concepto y lo permite.

II.2. Utilización de la Notación Punto.

Este estilo es utilizado en el lenguaje técnico para acceder a los atributos de las entidades y navegar en las relaciones entre estas.

Acceso simple a un atributo:
Entidad_1.Atributo

Navegar entre relaciones de entidades:
Entidad_1.Entidad_2. (…) .Entidad_n.Atributo

Por ejemplo, si se desea acceder al nombre del paciente se puede alcanzar anotando Paciente.Nombre, pero también es posible navegar en las relaciones. En el siguiente esquema:

Ilustración II.1 Porción del Esquema E/R para Transplante Renal.

Al indicar Cirujano.DonantesVivos.Evolución se quiere expresar en lenguaje natural "Las Evoluciones de los Donantes Vivos atendidos por un Cirujano…".

Nótese que es importante el orden, pues al formular Evolución.DonantesVivos.Cirujano se estaría expresando en lenguaje informal "El Cirujano cuyo Donante Vivo tiene una Evolución… ".

Al utilizar la navegación de la notación punto para establecer relaciones entre entidades y acceder a los atributos de estas, se obtendrán dos tipos de resultados principales debido a la cardinalidad entre los objetos del negocio; pues como mismo se puede especificar un solo miembro del negocio, también es viable hacer referencia a un conjunto de estos y resulta necesario discernir ambos casos.

II.2.1. Elementos Individuales.

Surgen cuando al utilizar esta notación se obtiene como resultado un solo elemento, son útiles para el acceso a atributos específicos de una entidad. En el siguiente diagrama:

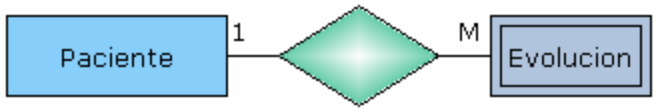


Ilustración II.2 Porción del Esquema E/R para Transplante Renal.

Al expresar: "El paciente de la evolución…" (Evolucion.Paciente) se está lógicamente refiriendo a un único paciente, pues al ser la relación uno a muchos, se tiene que una evolución sólo posee un paciente.

II.2.2. Elementos Múltiples.

Surgen cuando a partir de la notación punto se obtienen como resultado varios elementos, por ejemplo, utilizando la Ilustración II.2, al indicar "Las Evoluciones del Paciente…" (Paciente.Evolucion) se conciben varias evoluciones relacionadas con un único paciente como muestra la cardinalidad del diagrama. Nótese nuevamente que el orden de la notación es muy importante.

Con los elementos múltiples se pueden realizar operaciones de conjunto como:

Operador	Utilización
sizeof <collection>	Define cuántos elementos contiene la colección de elementos.
empty <collection>	Retorna verdadero si la colección no contiene elementos.
exists(<element>, <collection>)	Retorna verdadero si el elemento especificado existe al menos una vez en la colección.
max	Retorna el elemento máximo de la colección.

Tabla II.2 Algunos Operadores de Elementos Múltiples.

II.3. Lenguaje Técnico.

El lenguaje técnico creado para captar las reglas tratadas en este estudio se obtiene de la estructura que ofrece el patrón de restricción, en la cual mediante las <características> y los <hechos> se especifican las prohibiciones al <sujeto>. Para lograr entonces, reglas flexibles y poderosas, se debe contar con un lenguaje dentro de las <características> y los <hechos> que permita restringir correctamente al <sujeto>. Esto es posible navegando por la notación punto que brinda la posibilidad de establecer relaciones y acceder a los atributos deseados, comparados a través de operadores como:

- mayor que (>).
- menor que (<).
- igual a (=).
- Otros.

Es necesario también tener en cuenta la utilización de funciones como las anteriormente vistas para las operaciones con elementos múltiples, además del uso de las conjunciones “and” y “or” para tener varias restricciones en una misma regla. Esto, en conjunto con la lógica del patrón de restricción, ofrece un lenguaje con la potencia necesaria para establecer restricciones a los objetos del negocio, obteniendo reglas equivalentes al lenguaje OCL.

Una expresión de regla en lenguaje técnico sería:
Un Paciente no puede tener sizeof(Evolucion.idEvolucion) > 30
<Sujeto>: Paciente
<Características>: sizeof(Evolucion.idEvolucion) > 30
A esta regla le corresponde en lenguaje natural la siguiente:
Un paciente no puede tener más de 30 evoluciones.

II.4. Ventajas y desventajas en recursos de Base de Datos para implementar reglas de negocio, tipo restricción.

De los diferentes recursos presentados en el primer capítulo, es necesario conocer su aplicación al trabajo. Para ello se analizan sus principales ventajas y desventajas para implementar las reglas de negocio, tipo restricción. También se realizan comparaciones entre los recursos con más condiciones para determinar su empleo.

Vistas:

Demuth (Demuth et al., 2001) plantea que una vista permite evaluar una condición de búsqueda compleja que es parte de una restricción de integridad, puede sustituir la evaluación de las Assertions no soportadas en Bases de Datos, está basada en código SQL declarativo, es sujeto de la optimización en la Base de Datos y además se puede integrar en diferentes estrategias de evaluación de restricciones. Las vistas en general se presentan como un método híbrido de verificar la base de datos desde una aplicación, donde un objeto-relacional intermedio con sus propias transacciones evalúa la vista justo cuando la transacción se ejecute, y si alguna de ellas retorna una tupla que no satisfizo la condición, este es capaz de anular la transacción o darle algún tratamiento a los datos.

Assertions:

Las assertions de la base de datos son un chequeo de restricción no específico a una sola tabla, y pueden usarse al crear restricciones multi-tablas. Pero incluso ni siquiera este acercamiento pudiera ser viable, la mayoría de los sistemas de bases de datos comerciales no llevan a cabo las assertions como define la norma de SQL92/99. Así, cuando se llevan a cabo las reglas como restricciones en los negocios, deben usarse triggers hechos "a mano" (Zimbrão).

Las Restricciones (Constraints) CHECK:

Son un lugar ideal para introducir restricciones sencillas que sólo involucren los valores de un atributo en la entidad del negocio y así se explotan al máximo las utilidades a brindar por el sistema gestor. Se debe tener cuidado porque si esta regla sencilla se desea con un carácter permanente sería mejor tenerla en el modelo estático, debido a que su estado siempre activado ahorrará trabajo.

Triggers:

Parece ser el lugar ideal para introducir las prohibiciones pues en el instante de la inserción o actualización se puede verificar que la nueva fila cumpla con las restricciones del negocio. Sólo tiene algunos puntos desfavorables como la utilización de las conjunciones lógicas, se podría pensar que al utilizar el "o" se pueden crear tantos triggers como conjunciones existan, pero un caso bien diferente sería al enfrentar el "y" porque ambos triggers tienen que ser capaces de tenerse en cuenta, esto sin pensar en una mezcla de conjunciones. Otro punto desfavorable sería que un trigger sólo puede dispararse en una entidad del negocio y si se emplea más de una navegación de punto, posiblemente, tendría que ser necesario desencadenar triggers en varias tablas.

Comparación entre Triggers y Constraints:

Los constraints y triggers gozan de beneficios en situaciones especiales. La ventaja primaria de los triggers es que pueden contener procesamiento lógico complejo usando código Transact-SQL. Aunque los triggers pueden apoyar toda la funcionalidad de constraints, no son siempre el mejor método para una característica dada.

Los constraints pueden comunicar sobre errores sólo a través de un sistema mensaje-error estandarizado. Si la aplicación requiere (o consigue beneficiarse) personalizando mensajes y manejando un error más complejo, se debe usar un triggers (SQLServer2000, 2004).

II.5. Consideraciones sobre el <Sujeto>.

En general se deben tener algunas consideraciones a la hora de redactar una regla de restricción sobre todo con respecto al <sujeto> de la regla. Este, como se observa en su significado, no sólo puede especificar a entidades reconocibles sino también ser objetos especificados con antelación mediante otras reglas, como las de Clasificación. Por esto es recomendable no "tipear" al <sujeto> mediante sus <características>, sobre todo cuando esta característica se hace común en las restricciones; por ejemplo, remitiéndose al negocio de transplante renal, si se quiere que los pacientes del hospital poseedores de la mayoría de edad no tengan más de 5 exámenes físicos, tendría que redactarse una regla.

Lenguaje Natural:
Un paciente no puede ser mayor de 18 años y tener más de 5 exámenes físicos.

Lenguaje Técnico:
RN#1 Un paciente no puede tener
edad >18 and sizeof(ExamenFisico.idExamenFisico) > 5
<Sujeto>: Paciente
<Características>: Edad > 18 and sizeof(ExamenFisico.idExamenFisico) > 5

En este propio ejemplo si reglas posteriores siguieran utilizando la clasificación de mayoría de edad, aunque no esté incorrecta la previamente descrita, sería mejor dividirla en dos reglas siguiendo el principio de atomicidad y comodidad para su posterior uso:

Regla de Clasificación:
Un Paciente_Mayor es definido como los pacientes con más de 18 años.

Regla de Restricción:
Un Paciente_Mayor no puede tener más de 5 exámenes físicos.

<Sujeto>: Paciente_Mayor
<Características>: sizeof(ExamenFisico.idExamenFisico) > 5

Por tanto, se debe tratar de excluir de las <características> los valores de atributos que clasifiquen al <sujeto> de la regla, siempre y cuando esta clasificación sea reiteradamente aplicada.

II.6. Declaración de las <Características> y su implementación.

Cualquier regla, según la definición lógica del patrón de restricción, puede ser violada solamente en las <características>, por eso se le debe prestar gran atención a este elemento del patrón, además porque aquí se descubren las entidades del negocio donde se debe chequear la integridad de las restricciones.

II.6.1. Implementar una regla. Los Triggers.

Es significativo llegar a un consenso sobre cuándo es recomendable utilizar los diferentes recursos del SQL99, según las peculiaridades de la regla en cuestión y cómo van a ser implementadas estas restricciones del negocio. Esto proporcionará una herramienta para seleccionar el modo de generar la regla.

- Cuando las <características> sean simples (una sola especificación al sujeto), traten sobre un atributo del <sujeto> que es una entidad, y no existen <hechos>, se logran implementar mediante restricciones CHECK. Una regla que cumple esta condición sería la siguiente.

  Lenguaje Natural:

  Un Donante Potencial no puede tener más de 75 años de edad.

Lenguaje Técnico:

RN#2 Un DonantesPotenciales no puede tener Edad > 75.

<Sujeto>: DonantesPotenciales

<Características>: Edad > 75

- En cualquier otro caso se realizarán triggers que invoquen una función.

Como se ha podido apreciar, la idea de utilizar solamente un trigger no es conveniente. Por esto se desarrolla una alternativa que consiste en emplear los triggers para la extracción del sujeto en la navegación de la notación punto, y con este averiguar, a través de una función (en la cual se implementará toda la regla) si está violando o no alguna prohibición impuesta por el negocio.

En el siguiente ejemplo se consigue apreciar una <característica> compleja que relaciona al <sujeto> con otras entidades, del negocio de transplante renal se tiene el siguiente diagrama:

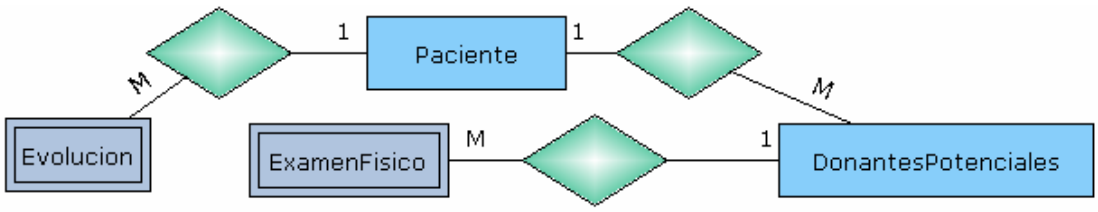


Ilustración II.3 Porción del Esquema E/R para Transplante Renal.

Lenguaje Natural:

Un Paciente no puede tener Donantes Potenciales con más de 3 Exámenes Físicos o tener en su Evolución una temperatura máxima mayor de 48 grados.

Lenguaje Técnico:

RN#3 Un Paciente no puede tener

sizeof(DonantesPotenciales.ExamenFisico.idExamenFisico) > 3 or

max(Evolucion.Temperatura) > 48

<Sujeto>: Paciente

<Características>: sizeof(DonantesPotenciales.ExamenFisico.idExamenFisico) > 3 or
max(Evolucion.Temperatura) > 48

Analizando la primera especificación de las <características> sizeof(DonantesPotenciales.ExamenFisico.idExamenFisico) > 3 se puede notar sin duda que esta condición puede ser violada únicamente al insertar un nuevo Examen Físico o modificarlo, por lo que se crea un trigger, el cual permita extraer el Paciente (@Subject) con el que se establece la condición, en este caso quedaría como sigue.

```
CREATE Trigger TRN#30 ON ExamenFisico
AFTER INSERT,UPDATE
AS
declare @Subject int
set @Subject = (SELECT a.idPaciente
         FROM Paciente a,DonantesPotenciales b,Inserted c
     Where (a.idPaciente=b.idPaciente) and
       (b.idDonantesPotenciales=c.idDonantesPotenciales))
if dbo.FRN#3(@subject) = 1
begin
 raiserror('RN#3',16,1);
 rollback transaction;
end
```

Este trigger resulta ser similar al generado por la segunda parte de las <características> sólo que el lugar de prevención es en Evolución.

```
CREATE Trigger TRN#31 ON Evolucion
AFTER INSERT,UPDATE
AS
declare @Subject int
set @subject = (SELECT a.idPaciente
               FROM Paciente a, Inserted b
                Where (a.idPaciente=b.idPaciente))
if dbo.FRN#3(@subject) = 1
begin
 raiserror('RN#3',16,1);
 rollback transaction;
end
```

Estos triggers, como se ha podido observar, declaran un @subject que es el <sujeto> condicionado de la regla, el que se extrae a través de una consulta definida por la notación punto. Después en una condicional se cuestiona si la regla es quebrantada o no para el sujeto específico al cual se le realizó una modificación relacional, en caso afirmativo se toman las medidas correspondientes.

II.6.2. Mensajes de Error en los triggers.

Los constraints actúan cuando se viola lo que ellos indican sin que el usuario pueda especificar acción alguna para esa infracción, el Gestor de Base de Datos simplemente envía un mensaje de error e impide que se realice la operación que produjo la violación. Esto no ocurre así al trabajar con los triggers ya que es permitido enviar en el momento de la violación un mensaje de error, pues si cualquier regla es vulnerada se debe levantar una excepción (Zimbrão). Esta excepción debe tener el ID de la regla comercial quebrantada, y permite que el sistema recupere el mensaje específico del error con la descripción de la regla en el idioma natural o cualquier información más extensa que se quiera comunicar.

Es por esto que al usar raiserror se devuelve un mensaje de error definido y se establece un indicador del sistema para registrar que se ha producido un error. Con el primer parámetro se está informando qué regla se infringió, el segundo trata el nivel de gravedad con el que se asocia este mensaje y el tercero representa la información acerca del estado de llamada del error (SQLServer2000, 2004).

II.6.3. Elementos del Trigger.

Tiempo:

Le dice al SGBD cuándo ejecutar el cuerpo del trigger, lo cual debe ser después (AFTER) que ocurra el evento del trigger. Así se garantiza que al ejecutarse la función esta pueda ver los efectos de la instrucción sobre la Base de Datos y detectar cualquier violación.

Evento:

Indica cuál es el comando de actualización sobre la entidad que va a activar el trigger, este evento puede ser uno o más de los siguientes: INSERT, DELETE y UPDATE. Pero en este caso específico sólo es usado el INSERT y el UPDATE pues son los eventos potenciales a romper una restricción.

Por supuesto, si se ha violado una regla entonces debe impedirse que los cambios se almacenen y esto se puede lograr mediante un ROLLBACK TRANSACTION, que según (SQLServer2000, 2004) deshace todas las modificaciones de datos realizadas hasta el punto de transacción actual, incluidas las que realizó el desencadenador.

II.6.4. La Función, cuerpo de la regla.

Como se ha podido observar los triggers hacen referencia a una función. Esta es la encargada de implementar el cuerpo de la regla, es decir, en ella se verifica si el <sujeto> viola las condiciones impuestas por la restricción. Su objetivo es responder si se infringe alguna de las prohibiciones mediante un valor booleano.

Continuando con el ejemplo RN#3, como se pudo apreciar estos pacientes que han sido extraídos de los triggers cuyas entidades han sufrido transformaciones, son los que pudieran violar alguna de las restricciones del negocio, lo cual se debe verificar mediante una función en la cual se implementará la regla en su totalidad. La función FRN#3 que se ha visto llamar desde los triggers quedaría como sigue:

```
CREATE FUNCTION FRN#3(@idSujeto Integer)
Returns BIT AS
begin
declare @result BIT
set @result = 0
if ((SELECT COUNT(c.idExamenFisico)
   FROM Paciente a, DonantesPotenciales b, ExamenFisico c
      Where(@idSujeto=a.idPaciente)and
           (a.idPaciente=b.idPaciente)and
  (b.idDonantesPotenciales=c.idDonantesPotenciales))>3)
  or((SELECT MAX(b.Temperatura)
     FROM Paciente a, Evolucion b
      Where(@idSujeto=a.idPaciente)and
                (a.idPaciente=b.idPaciente)) > 48)
set @result = 1
return @result
end
```

Como se puede percibir el <sujeto> es recibido por la función mediante @idSujeto, que se utilizará en las consultas posteriores; se crea un @result de tipo bit (booleano) en el cual quedará reflejado si la regla se infringe o no, por lo que se inicializa en 0 (False). Después una condición, donde se verifican las restricciones mediante consultas y se modifica el valor de @result en caso afirmativo a la violación; al final se retorna el valor del resultado.

II.6.5. Navegación por la Notación Punto y Cardinalidad.

La cardinalidad es un tema complejo, su sentido de dificultad viene dado por la incertidumbre de relaciones en la notación punto, dado que puede existir cualquier tipo de correspondencia al expresar Entidad_1.Entidad_2. En este trabajo sólo se trataron dos modos simples aunque muy útiles para navegar entre las relaciones, exponiéndose además los restantes.

En el ejemplo anterior se ejemplificó una relación en la cual se navegó la relación Paciente.DonantesPotenciales.ExamenesFisicos, este comprende un tipo de navegación posible, relaciones de uno a muchos sucesivas ([1]---[M 1]---[M 1]---…---M). Pero este un solo tipo de los tantos existentes. Aquí se muestran otros.

Relaciones de muchos a uno sucesivas ([M]---[1 M]---[1 M]---…---[1]):
Partiendo de una entidad se va navegando por otras entidades relacionadas entre sí, de tal forma que siempre la entidad hija tenga un elemento único, finalizando así con un solo resultado como el mostrado a continuación.

Lenguaje Natural:
Un Paciente no puede tener como Hospital de Origen al Hospital de Matanzas.

Lenguaje Técnico:
RN#4 Un Paciente no puede tener HospitalOrigen.Nombre = ‘Hospital de Matanzas’.

<Sujeto>: Paciente
<Características>: HospitalOrigen.Nombre = 1

De este modo el trigger debe ser colocado en el propio sujeto, donde puede ocurrir la violación de la regla, quedaría de la siguiente manera:

```
CREATE Trigger TRN#40 ON Paciente
AFTER INSERT,UPDATE AS
declare @Subject int
set @subject = (SELECT a.idPaciente
                    FROM Inserted a)
if dbo.FBR#4(@Subject) = 1
begin
 raiserror('RN#4',16,1);
 rollback;
end
```

Otras formas no tratadas pueden ser relaciones de muchos a muchos sucesivas ([M]---[M M]---[M M]---…---[M]) y las combinaciones entre todas los tipos de relaciones sin ningún orden . El caso de relaciones uno a N (1---N) se puede tomar como un caso particular de uno a muchos (1---M) y en el caso de uno a uno (1---1) tendría que asegurarse dónde se ha implantado la llave foránea.

II.7. Elección del estilo de la consulta.

No existe una única manera de implementar reglas de negocio. A continuación se exponen dos formas de implementar una misma regla, detallando sus diferencias para posteriormente llegar a una conclusión relativa a la sobresaliente. Para concluir es necesario basarse en resultados computacionales, teóricos y prácticos sobre ambas variantes.

Lenguaje Natural:
Un Examen Físico no puede realizarse a un Paciente cuyo Tiempo de Terapia supera los 20 días.

Lenguaje Técnico:
RN#5 Un ExamenFisico no puede tener un Paciente.TiempoTerapia > 20.

<Sujeto>: ExamenFisico
<Características>: Paciente.TiempoTerapia > 20

Parte de su implementación será la variante uno, la que se nombrará función#1:

```
CREATE FUNCTION FRN#5(@idSujeto Integer)
Returns BIT AS
begin
     declare @result BIT
     set @result = 0
     if ((SELECT b.TiempoTerapia
     FROM ExamenFisico a, Paciente b
      Where(@idSujeto=a.IdExamenFisico)and
           (a.idPaciente=b.idPaciente)) > 20)
set @result = 1
return @result
end
```

Como se aprecia, la consulta SELECT devuelve el resultado obtenido y después este se compara con el valor no deseado. Pero, ¿por qué no poder realizar esta comparación dentro de la consulta? Bueno en este caso se pensaría en la variante dos, llamada función #2:

```
CREATE FUNCTION FRN#5(@idSujeto Integer)
Returns BIT AS
begin
 declare @result BIT
 set @result = 0
 if exists(SELECT *
           FROM ExamenFisico a, Paciente b
          Where (@idSujeto = a.IdExamenFisico) and
                (a.idPaciente = b.idPaciente) and
                (b.TiempoTerapia > 20))
 set @result = 1
 return @result
end
```

Para obtener elementos con los cuales comparar las dos variantes se buscó el plan de ejecución de ambas funciones, específicamente en sus diferencias; esto se logró con las siguientes líneas en el Query Analyser del SQL-Server (SQLServer2000, 2004):

```
GO
SET SHOWPLAN_ALL ON
GO
     if ((SELECT b.TiempoTerapia
      FROM ExamenFisico a, Paciente b
          Where (2 = a.IdExamenFisico) and
                (a.idPaciente = b.idPaciente)) > 20)
print ('Función #1')
GO
SET SHOWPLAN_ALL OFF
GO

GO
SET SHOWPLAN_ALL ON
GO
if exists(SELECT *
          FROM ExamenFisico a, Paciente b
           Where (2 = a.IdExamenFisico) and
                      (a.idPaciente = b.idPaciente) and
                           (b.TiempoTerapia > 20))
print ('Función #2')
GO
SET SHOWPLAN_ALL OFF
GO
```

Los resultados más significativos se muestran en la siguiente tabla.

Atributos del Plan	Función #1	Función #2
Costo estimado (acumulado) de esta operación y todas las que surgen de ella.	2.5328303E-2	2.5328504E-2
Suma de costos estimados en la CPU para los operadores utilizados.	85,19322	89,99321

Tabla II.3 Resultados del Plan de Ejecución.

Desde el punto de vista de eficiencia se tienen diferencias poco significativas en pequeñas consultas como la mostrada de ejemplo, pues se realizan operaciones muy similares, sólo que en el cálculo de tuplas es más eficiente la primera función debido precisamente a la operación (b.TiempoTerapia > 20) que logra la diferencia en el costo estimado de CPU para la segunda variante. Mirándolo desde otro ángulo, la modularidad de esta última pierde mucho terreno pues se vincula la navegación de la notación punto al operador lógico y no permite, como en el ejemplo de la RN#2, solucionar operaciones aparentemente complejas de forma sencilla.

Incluso al tener en cuenta la modularidad de las consultas se pueden crear restricciones interesantes y complejas como la siguiente:

max(Entidad_1.Entidad_2.Atributo) > Entidad_3.atributo

Por tanto, se arriba a la conclusión que la variante uno y por demás utilizada en este trabajo, logra mejores resultados computacionales, teóricos y prácticos.

II.8. Declaración de los <Hechos> y su implementación.

Los <hechos> toman lugar en la regla para, de alguna forma, establecer una excepción ante una característica dada; ellos por sí solos no influyen en que determinado sujeto viole una condición, y su uso fundamental es establecer un conocimiento general del negocio sin estar necesariamente dependiendo del sujeto.

Por esto necesitan tener su propia sintaxis y ser tratados diferente a las <características>, aún cuando sus semejanzas son grandes.

En los <hechos>, al igual que en las <características>, se utiliza la navegación del punto para acceder a los atributos y establecer restricciones. De esta forma se logra, como ya se ha experimentado, obtener dos tipos de resultados: elementos individuales y elementos múltiples. Aunque en los <hechos> surgen ligeras diferencias.

II.8.1. Elementos Individuales.

A diferencia de lo visto en los elementos individuales, los <hechos> no tienen un vínculo directo con el <sujeto> el cual especifica la singularidad. Entonces, al establecer una relación o tratar de acceder a un solo elemento habrá que pasar una descripción única a través de la cual se conozca a quién de la entidad principal (la que comienza la navegación) se quiere chequear.

Obsérvese la cardinalidad de la navegación que puede dar un elemento individual ([M]---[1 M]---[1 M]---…---[1]).

Obviamente al querer obtener un solo objeto, se debe especificar una representación única de la entidad con la que comienza la relación.

Es interesante destacar, aunque el <hecho> no se relacione con el <sujeto>, la práctica lo necesita con frecuencia. Es importante, por ende, tener alguna forma de lograrlo como en el siguiente ejemplo.

Lenguaje Natural:
Un Donante Vivo puede tener más de 5 Evoluciones sólo si este es atendido por el Cirujano jefe Antolín.

Lenguaje Técnico:
RN#6 Un DonanteVivo puede tener sizeof(Evolucion.idEvolucion) > 5 sólo si DonanteVivo(@idSujeto).Cirujano.Nombre = 1.

<Sujeto>: DonanteVivo

<Características>: sizeof(Evolucion.idEvolucion) > 5

<Hechos>: DonanteVivo(@idSujeto).Cirujano.Nombre = 1

```
CREATE FUNCTION FBR#6(@idSujeto Integer)
Returns BIT AS
begin
declare @result BIT
set @result = 0
if ((SELECT COUNT(b.idEvolucion)
      FROM DonanteVivo a, Evolucion b
          Where (@idSujeto=a.CiDonante) and
                     (a.CiDonante=b.idDonanteVivo))>5)
 if not ((SELECT b.Nombre
      FROM DonanteVivo a, Cirujano b
       Where (a.CiDonante=@idSujeto) and
             (a.idCirujano = b.idCirujano))= 1)
set @result = 1
return @result
end
```

De forma general se tiene que los <hechos> agregan a la función la sentencia “if not”, después de la condicional “if”, obteniéndose:

```
If <características>
   If not <hechos>
   set @result = 1
```

Como logra apreciarse, la novedad según lo visto radica en el “if not” respondiendo al contexto del patrón en el cual se encuentran los <hechos>, porque si las <características> no se violan, entonces se debe chequear si no se cumplen los <hechos> para tomar las medidas necesarias, en este caso sería asignarle el valor 1 (True) a @result, expresando así que se ha violado la condición.

II.8.2. Elementos Múltiples.

Otra cosa es cuando existe una colección de elementos ([1]---[M 1]---[M 1]---…---M) porque quizás no sea necesario detallar a la entidad principal, especialmente cuando sólo se quiere una estadística; por ejemplo, saber cuántos cirujanos hay en total, por supuesto, aquí no hace falta especificar qué cirujano se quiere investigar. Otra situación completamente diferente es cuando sí se necesite, por ejemplo:

Lenguaje Natural:
Un Paciente puede tener como Hospital de Origen al Hospital Viejo sólo si la cantidad de pacientes enviados por este Hospital no supera los 100 Pacientes.

Lenguaje Técnico:
RN#7 Un Paciente puede tener como HospitalOrigen.Nombre = 2 sólo si sizeof(HospitalOrigen(2).Paciente.idPaciente) <= 100.

<Sujeto>: Paciente
<Características>: HospitalOrigen.Nombre = 2
<Hechos>: sizeof(HospitalOrigen(2).Paciente.idPaciente) <= 100

```
CREATE FUNCTION FBR#7(@idSujeto Integer)
Returns BIT AS
begin
declare @result BIT
set @result = 0
if ((SELECT b.Nombre
      FROM Paciente a, HospitalOrigen b
       Where (@idSujeto = a.idPaciente) and
        (a.idHospitalOrigen=b.idHospitalOrigen)) = 2)
if not ((SELECT COUNT(b.idPaciente)
          FROM HospitalOrigen a, Paciente b
      Where (a.idHospitalOrigen = 2) and
     (a.idHospitalOrigen = b.idHospitalOrigen)) <= 100)
set @result = 1
return @result
end
```

II.9. Implementación en SQL Estándar.

Hasta estos momentos sólo se ha hecho alusión a ejemplos implementados para SQL-SERVER, ¿será necesario generar código para otros gestores? Esta respuesta es bien sencilla, basta recordar que el negocio hacia el cual va dirigido esta investigación puede tener su almacén de datos bajo cualquier sistema conocido, entonces ¡sí es necesario abarcar todos los lenguajes!

En la actualidad son muchos los sistemas gestores de bases de datos, sin embargo existe un estándar para todos ellos: el SQL Estándar. Es entonces recomendable manejar este lenguaje estándar para abarcar cualquier sistema posible en el cual se mueva un negocio.

A modo de explicación se muestra el código generado en SQL-99, según Kellenbenz (Kellenbenz et al., September 1999), para la RN#6, que se muestra a continuación en su forma de expresión informal.

Lenguaje Natural:
Un Donante Vivo puede tener más de 5 Evoluciones sólo si este es atendido por el Cirujano jefe Antolín.

```
CREATE FUNCTION FRN#0(idSujeto Integer)
Returns BIT
BEGIN
     declare result BIT default 0;
     if ((SELECT COUNT(b.idEvolucion)
      FROM DonanteVivo a, Evolucion b
            Where(idSujeto=a.CiDonante)and
                 (a.CiDonante=b.idDonanteVivo)) > 5)Then
      elseif not ((SELECT b.Nombre
                FROM DonanteVivo a, Cirujano b
                 Where(a.CiDonante=idSujeto)and
                      (a.idCirujano=b.idCirujano))=1)
          set result = 1;
     end if
     return result;
END
```

```
CREATE Trigger TRN#00
AFTER INSERT,UPDATE ON Evolucion
REFERENCING NEW ROW AS Inserted
FOR EACH ROW
BEGIN ATOMIC
     declare subject integer;
     set subject = (SELECT a.CiDonante
                    FROM DonanteVivo a, Inserted b
                    Where (a.CiDonante=b.idDonanteVivo))
     if FRN#0(subject) = 1 Then
          DECLARE messg CONDITION FOR SQLSTATE RN#0;
          Signal messg;
          rollback;
     end if;
END
```

II.10. Diseño del Repositorio.

Según el repositorio tratado por Tony Morgan (Morgan, 2002), las reglas independientemente de su tipo, han de tener su propia tabla que se define como:

	Nombre	**Tipo**	**Descripción**
PK	ID	AutoNumérico	Único ID para la regla.
	Versión	Texto	Número de la Versión.
	Ultimo Cambio	Fecha/Tiempo	Cuándo la regla fue creada o el último cambio.
	Ultimo Cambio por	Texto	Nombre de la persona responsable.
	Tipo	Numérico	Llave foránea al tipo de la regla.
	Nombre Descriptivo	Texto	Descripción textual de la regla.
	Estado	Texto	Indicador que muestra el actual estado de la regla.
	Lenguaje Formal	Memo	Representación formal de la regla.

Tabla II.4 Tabla para el almacenamiento de reglas.

Específicamente en las reglas de negocio, tipo restricción que se han desarrollado existe, además de los posibles atributos anteriormente vistos, una función donde se encuentra el cuerpo de la regla y uno o varios triggers que usan dicho recurso. Si se fuera a diseñar un repositorio que aceptara estas condiciones en una Base de Datos quedaría como sigue:

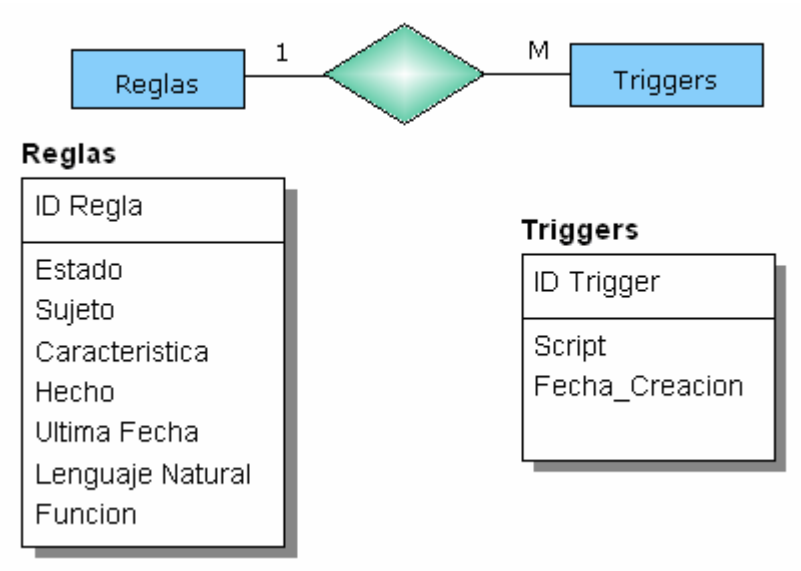


Ilustración II.4 Esquema E/R del Repositorio para Reglas de Negocio, tipo Restricción.

Donde en la tabla Reglas, el Estado brinda información si está activada o desactivada, el Sujeto, la Característica y el Hecho son los mismos del patrón para reglas de tipo restricción, la Última Fecha en que sufrió una modificación, el Lenguaje Natural que fue expresado por el usuario y la Función única donde radica el cuerpo de la regla implementada. Referida con la tabla Reglas se tendrían los Triggers en una relación uno a muchos, los cuales cuentan con un script que los implementa y una Fecha de Creación.

Es recomendable el diseño en XML por las ventajas que poseen estos documentos, sobre todo su estandarización, lo que permitiría luego generalizar independientemente de cualquier sistema gestor de base de datos, ya sea importando el documento desde un gestor específico o exportándolo hacia uno. Un esquema XSD brinda todas las facilidades para validar un documento XML, y muestra la estructura del repositorio.

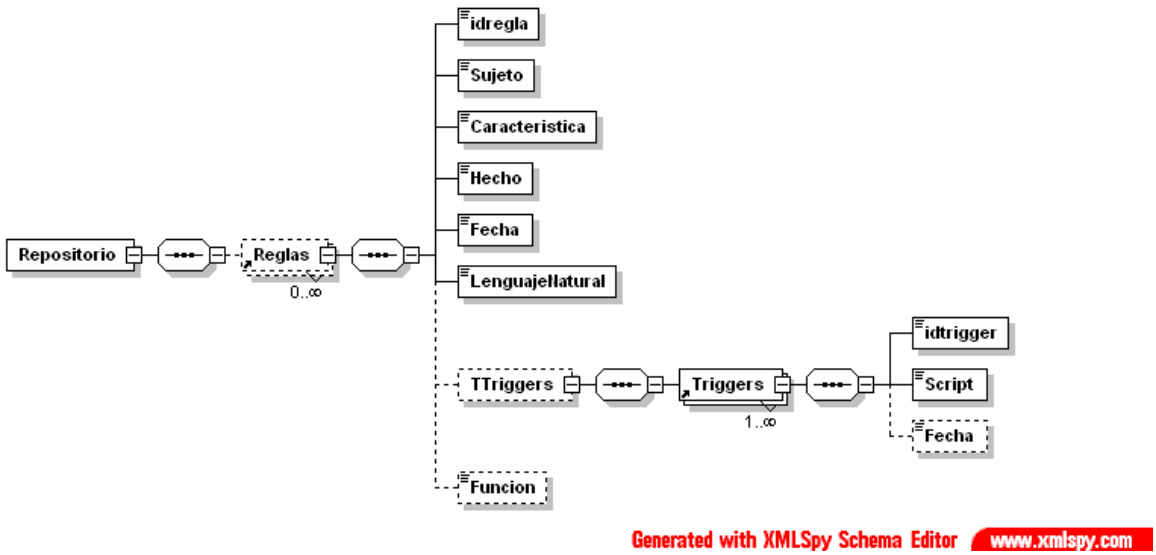


Ilustración II.5 Esquema XSD para validar el Repositorio.

Como se puede apreciar un repositorio está conformado por 0 o más reglas, las que a su vez contienen los atributos vistos incluyendo opcionalmente a los triggers y la función, pues esta regla puede no estar implementada aún. Aquí se utilizó un TTrigers para almacenar a uno o varios triggers que verifican se cumplan las restricciones del negocio, su uso es principalmente para la comodidad y estructuración del documento, además de tenerlos bien localizados y almacenados; nótese que los TTriggers después de implementados deben contener al menos un trigger pues toda regla genera alguno en su automatización.

Un ejemplo sencillo de un repositorio en XML puede ser el siguiente tomando la RN#1.

Lenguaje Natural:

Un paciente no puede ser mayor de 18 años y tener más de 5 exámenes físicos.

Lenguaje Técnico:

RN#1 Un paciente no puede tener

edad >18 and sizeof(ExamenFisico.idExamenFisico) > 5

El documento XML que almacena esta regla sin implementar sería el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<Repositorio xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="C:\ Repositorio.xsd">
  <Reglas>
    <idregla>RN#1</idregla>
    <Estado>INACTIVA</Estado>
    <Sujeto>Paciente</Sujeto>
    <Caracteristica> Edad &gt; 18 and
                    sizeof(ExamenFisico.idExamenFisico)&gt;5
    </Caracteristica>
    <Hecho/>
    <Ultima_Fecha>2008-08-13</Ultima_Fecha>
    <LenguajeNatural>Un paciente no puede ser mayor de 18 años
                     y tener más de 5 exámenes físicos.
    </LenguajeNatural>
  </Reglas>
</Repositorio>
```

Después de la extracción de la regla y su implementación, esta debe almacenarse nuevamente en el repositorio siguiendo el esquema definido previamente. Para el ejemplo tratado resulta como se muestra.

```
<?xml version="1.0" encoding="UTF-8"?>
<Repositorio xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="C:\ Repositorio.xsd">
  <Reglas>
    <idregla>RN#1</idregla>
    <Estado>INACTIVA</Estado>
    <Sujeto>Paciente</Sujeto>
    <Caracteristica> Edad &gt; 18 and
                    sizeof(ExamenFisico.idExamenFisico) &gt; 5
    </Caracteristica>
    <Hecho/>
    <Ultima_Fecha>2008-08-13</Ultima_Fecha>
    <LenguajeNatural>Un paciente no puede ser mayor de 18 años
        y tener más de 5 exámenes físicos.
    </LenguajeNatural>
```

```
<TTriggers>
       <Triggers>
         <idtrigger>TRN#10</idtrigger>
         <Script>
           CREATE Trigger TRN#10 ON Paciente
             AFTER INSERT,UPDATE AS
             declare @subject int
             set @subject = (SELECT a.idPaciente
                             FROM Inserted a)
             if dbo.FRN#1(@Subject) = 1
               begin
                 raiserror('RN#1',16,1);
                 rollback transaction;
               end
         </Script>
       </Triggers>
       <Triggers>
         <idtrigger>TRN#11</idtrigger>
         <Script>
           CREATE Trigger TRN#11 ON ExamenFisico
           AFTER INSERT,UPDATE AS
           declare @Subject int
           set @subject = (SELECT a.idPaciente
                  FROM Paciente a, Inserted b
                    Where (a.idPaciente=b.idPaciente))
               if dbo.FRN#1(@Subject) = 1
               begin
                 raiserror('RN#1',16,1);
                 rollback transaction;
                 end
         </Script>
       </Triggers>
     </TTriggers>
```

```
    <Funcion>
         CREATE FUNCTION FRN#1(@idSujeto Integer)
         Returns BIT AS
         begin
           declare @result BIT
           set @result = 0
           if ((SELECT a.Edad
             FROM Paciente a) &gt; 18)
               and ((SELECT COUNT(b.idExamenFisico)
                 FROM Paciente a, ExamenFisico b
                   Where (@idSujeto=a.idPaciente) and
                        (a.idPaciente=b.idPaciente)) &gt; 5)
           set @result = 1
           return @result
         end
    </Funcion>
  </Reglas>
</Repositorio>
```

II.11. Conclusiones Parciales.

Con la finalización de este capítulo ya se conoce la estructura del lenguaje técnico para las reglas de tipo restricción mediante el patrón creado, además de algunas recomendaciones para su edición. Se domina igualmente las características y la forma en que se generan los scripts para la base de datos del negocio, así como la estructura del repositorio que los almacena mediante un documento XML. Con esta información sólo resta ponerla en práctica mediante una aplicación, una aplicación que utilice todos los resultados de este capítulo y logre finalmente implementar y almacenar las reglas de negocio.

Capítulo III:

IMPLEMENTACIÓN DEL EDITOR.

III. IMPLEMENTACIÓN DEL EDITOR.

Este capítulo trata el desarrollo del editor para reglas de negocio, tipo restricción. Consta de tres secciones, una primera en la cual se aborda el proceso de compilación del lenguaje técnico y se especifica por tanto la sintaxis de este y sus características. La segunda sección es dedicada a la presentación del esquema con el cual se valida el repositorio, mostrándose completamente para aclarar sus elementos principales. La tercera y última sección discurre sobre la aplicación en sí, sus casos de uso, su estructura y funcionamiento; se describe igualmente el ambiente de trabajo y las comodidades de este.

III.1. El Compilador de Reglas.

Las reglas en lenguaje técnico deben ser automatizadas, para convertir estas reglas en triggers y función, expuestos en el Capítulo II, se crea un compilador, el que va a ser utilizado por una aplicación que ha de permitir editar la regla por un analista.

En la figura siguiente se muestra el caso de uso **Compilar Regla**. En la cual como se puede apreciar es el editor de reglas el que emplea al compilador.

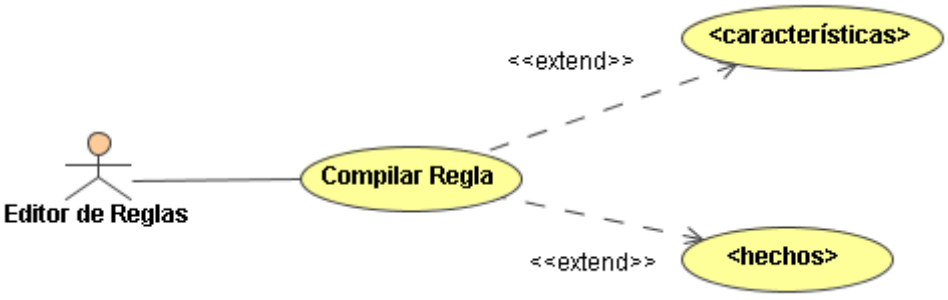


Ilustración III.1 Caso de uso del compilador.

Para compilar la regla se parte del lenguaje técnico y tras el proceso de compilación se genera código SQL el cual implementa la regla.

Este código es ejecutado por el gestor de base de datos del negocio al insertarse o modificarse los datos y retorna los resultados de su ejecución (ver Ilustración III.2).

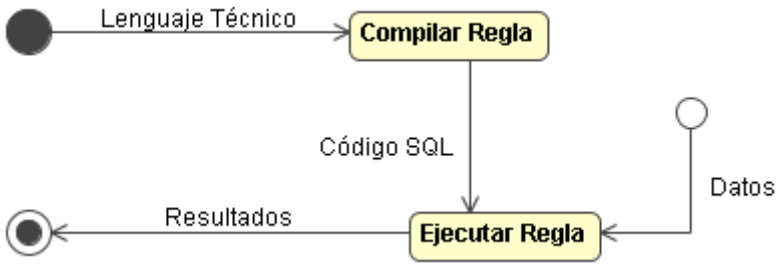


Ilustración III.2 Diagrama de Estado para Compilar Regla.

III.1.1. Implementación, Operadores y Funciones.

El proceso de compilación de las Reglas de Negocios, tipo restricción, utilizado en la versión actual se hace en una sola pasada. Durante este proceso se va generando directamente parte del código resultante, el resto de este se formaliza en la aplicación.

Este compilador está compuesto por otros dos compiladores, uno se encarga de compilar las <características> y otro los <hechos>. Este diseño responde a la diferencia en la gramática de estos elementos y también a que los <hechos> no tienen, necesariamente, que formar parte de una regla de restricción; además de proporcionar una comodidad adicional al separar el análisis sintáctico y darle al usuario el lugar específico donde se cometió el error. El proceso de compilación se muestra mediante este diagrama de actividad:

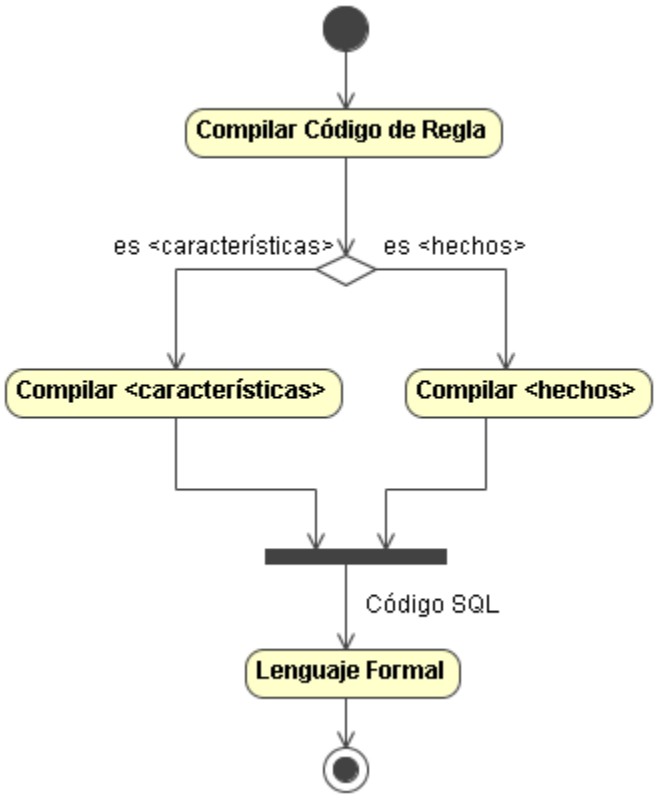


Ilustración III.3 Diagrama de Actividad en la Compilación.

El compilador fue realizado en el software Parser Generator v1.12 con Lex y YACC para Delphi. Lex es un lenguaje generalizado para crear patrones de coincidencia en códigos. Los patrones son especificados usando expresiones regulares, cada una de estas tiene una acción correspondiente asociada a ellos. Cuando se encuentra una cadena que coincide con la expresión regular se realiza la acción respectiva.

YACC es una herramienta que, dado un estilo de especificación BNF (Forma Normal de Backus) de la gramática, puede generar el parser correspondiente. YACC no acepta cada gramática presentada a él; sin embargo, la clase de gramática que acepta es generalmente poderosa, más que suficiente para las necesidades de la programación (Bumble-Bee, 2000).

En el Lex se definen principalmente los tipos de datos, los operadores y las funciones que se aceptan del lenguaje técnico, estos son útiles para conformar la gramática de la regla, tipo restricción.

Operadores	Nombre
>=	Mayor igual
<=	Menor Igual
<>	Diferente
(	Paréntesis abierto
)	Paréntesis cerrado
>	Mayor que
<	Menor que
=	Igual a
.	Punto
,	Coma
and	“y” lógico
not	“not” lógico
or	“or” lógico

Tabla III.1 Operadores del Lenguaje Técnico creado.

A continuación se ofrecen las funciones.

Funciones	Significado
`sizeof`	Retorna cuántos elementos contiene la colección de elementos.
`exists`	Retorna verdadero si el elemento especificado existe al menos una vez en la colección.
`empty`	Retorna verdadero si la colección no contiene elementos.
`avg`	Retorna el promedio de los elementos de la colección.
`sum`	Retorna la suma de los elementos de la colección.
`min`	Retorna el mínimo de los elementos de la colección.
`max`	Retorna el elemento máximo de la colección.
`avgdif`	Retorna el promedio de los elementos diferentes de la colección.
`sizeofdif`	Retorna cuántos elementos diferentes contiene la colección de elementos.
`sumdif`	Retorna la suma de los elementos diferentes de la colección.

Tabla III.2 Funciones del Lenguaje Técnico creado.

En esta versión del compilador no se implementa el tipo “string” de datos, por lo cual en los ejemplos anteriormente vistos, los textos constantes se tratan como números. Es igualmente apreciable que a las tablas del negocio se les otorgan alias alfabéticamente en minúsculas, por lo tanto una navegación con la notación punto debe contener menos de 26 entidades.

Igualmente sólo se generan triggers y funciones, restando implementar reglas simples mediante restricciones CHECK. Estas son creadas con funciones y triggers; aunque, como se trató en el Capítulo II, presenta desventajas que deberán ser solucionadas en versiones posteriores.

III.1.2. Sintaxis de la Gramática.

En el Yacc se describe la gramática creada para las <características> y los <hechos>, separada en dos archivos debido a su compilación disjunta.

En el caso de las <características>, su gramática se define como sigue:

```
Características : Exp;
Exp         :
            | Exp TOR Exp
            | TNOT Exp
            | Exp TAND Exp
            | Elemento GT Elemento
            | Elemento LT Elemento
            | Elemento GEQ Elemento
            | Elemento LEQ Elemento
            | Elemento NEQ Elemento
            | Elemento EQ Elemento
            | Token_Logico
            | LPAREN Exp RPAREN
            ;
Token_Logico
            :  TEXISTS  LPAREN  Elemento  TCOM  Caract_Multiple
RPAREN
            | TEMPTY LPAREN Caract_Multiple RPAREN
            ;
Elemento
            : NUMBER
            | Token_Elemental
            | Caract_Individual
            ;
Token_Elemental
            : TSIZEOF LPAREN Caract_Multiple RPAREN
            | TAVG LPAREN Caract_Multiple RPAREN
            | TSUM LPAREN Caract_Multiple RPAREN
            | TMIN LPAREN Caract_Multiple RPAREN
            | TMAX LPAREN Caract_Multiple RPAREN
            | TAD LPAREN Caract_Multiple RPAREN
            | TSOD LPAREN Caract_Multiple RPAREN
            | TSD LPAREN Caract_Multiple RPAREN
            ;
```

```
Caract_Individual
           : Entidad TPTO Caract_Individual
           | Atrib_Individual
           ;
Caract_Multiple
           : Entidad TPTO Caract_Multiple
           | Entidad TPTO Atrib_Multiple
           ;
Atrib_Individual
           : IDENT;
Atrib_Multiple
           : IDENT;
Entidad
           : IDENT;
```

La gramática de los <hechos> es bastante similar a las <características>, sólo tienen pequeñas diferencias, dadas fundamentalmente por la forma en que se necesita captar y generar:

```
Hechos     : Exp;
Exp   :
           | Exp TOR Exp
           | TNOT Exp
           | Exp TAND Exp
           | Elemento GT Elemento
           | Elemento LT Elemento
           | Elemento GEQ Elemento
           | Elemento LEQ Elemento
           | Elemento NEQ Elemento
           | Elemento EQ Elemento
           | Token_Logico
           | LPAREN Exp RPAREN
           ;
Token_Logico
           :  TEXISTS  LPAREN  Elemento  TCOM  Hecho_Multiple
RPAREN
           | TEMPTY LPAREN Hecho_Multiple RPAREN
           ;
```

```
Elemento
            : NUMBER
            | Token_Elemental
            | Hecho_Individual
            ;
Token_Elemental
            : TSIZEOF LPAREN Hecho_Multiple RPAREN
            | TAVG LPAREN Hecho_Multiple RPAREN
            | TSUM LPAREN Hecho_Multiple RPAREN
            | TMIN LPAREN Hecho_Multiple RPAREN
            | TMAX LPAREN Hecho_Multiple RPAREN
            | TAD LPAREN Hecho_Multiple RPAREN
            | TSOD LPAREN Hecho_Multiple RPAREN
            | TSD LPAREN Hecho_Multiple RPAREN
            ;
Hecho_Individual
            : Entidad LPAREN Macheo RPAREN TPTO Hecho_Ind
            ;
Macheo
            : IDENT
            | NUMBER
            ;
Hecho_Ind
            : Entidad TPTO Hecho_Ind
            | Atrib_Individual
            ;
Hecho_Multiple
            : Entidad LPAREN Macheo RPAREN TPTO Hecho_Mult
            | Hecho_Mult
            ;
Hecho_Mult
            : Entidad TPTO Hecho_Mult
            | Atrib_Multiple
            ;
Atrib_Individual
            : IDENT
            ;
Atrib_Multiple
            : IDENT
            ;
Entidad
            : IDENT;
```

Cabe destacar que el compilador toma la llave primaria de las tablas en la unit UMLClassDiagram, con las cuales se realizarán las relaciones entre entidades, toma además la llave foránea de una tabla con respecto a otra. Se asume en esta versión que sólo existe una llave primaria para cada entidad. Esta unit es el único vínculo con el negocio de transplante renal, por lo que sólo se necesita modificarla para trasladar la aplicación a otro negocio.

En esta versión del compilador sólo se tienen errores sintácticos. No se hacen chequeos de tipos debido a que el lenguaje técnico, generado previamente sin cometer errores, es tomado del repositorio.

III.2. Creación del esquema del Repositorio.

Para la creación de un esquema personalizado es necesaria alguna herramienta como el XMLSPY. Este ofrece la opción de construir un esquema para verificar que el documento XML este bien formado y sea válido (Altova, 2004).

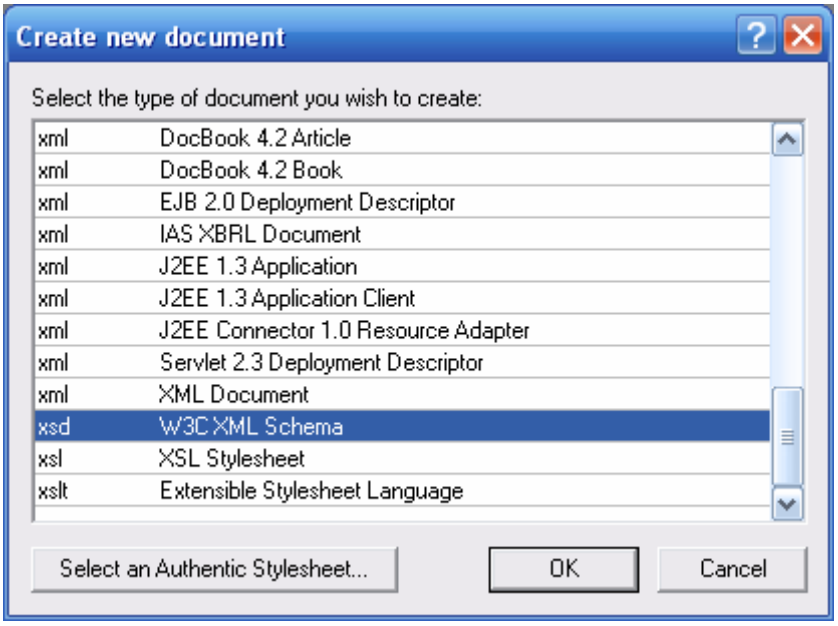


Ilustración III.4 Crear Nuevo Esquema XSD con XMLSPY.

El esquema elaborado finalmente, visto en forma de celdas es el siguiente:

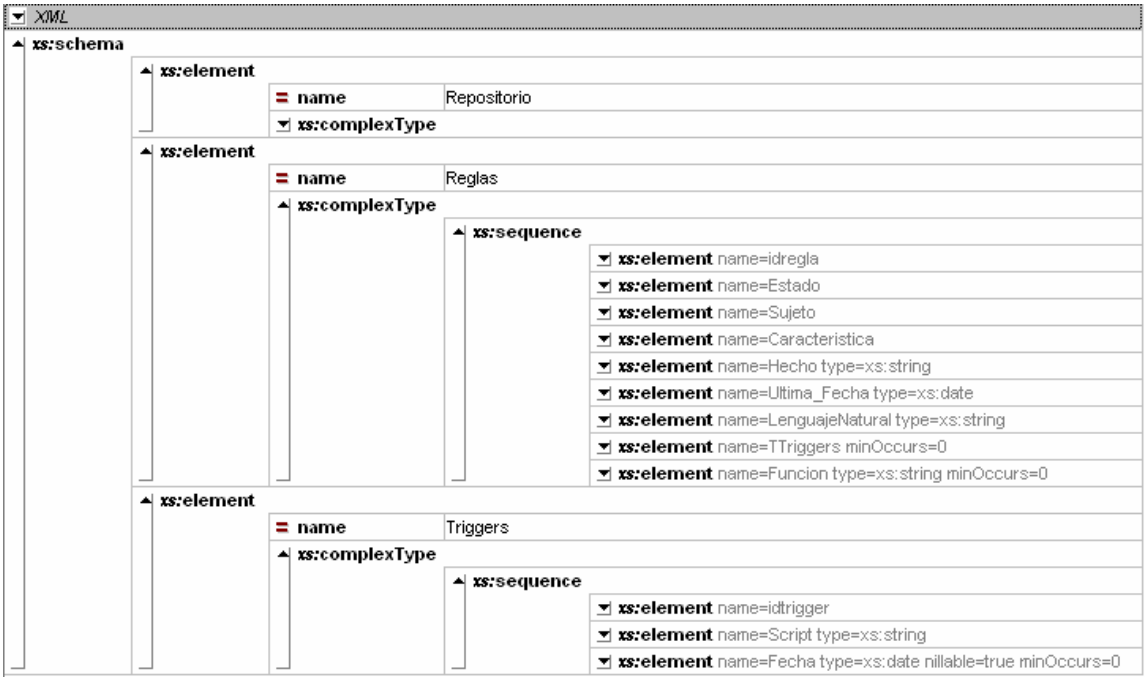


Ilustración III.5 Esquema ilustrado en celdas.

Es necesario apuntar que se restringieron los valores a tomar por los nombres de las reglas y los triggers, ya que deben ser únicos al insertarse como llaves primarias en una base de datos o quedar almacenados aquí, por ello se les asoció el tipo `memberTypes="xs:ID"`.

A partir de este esquema es posible en XMLSPY crear la estructura de una base de datos a la cual posteriormente se le exportan los datos contenidos en el documento XML. Esto es útil para la creación de un sistema gerenciador de reglas de negocio en el cual se desee tener un repositorio montado en un servidor. En el menú **Convert** la opción **Create DB Structure based on Schema** después de escoger o conectar a la base de datos crea las tablas correspondientes (Altova, 2004).

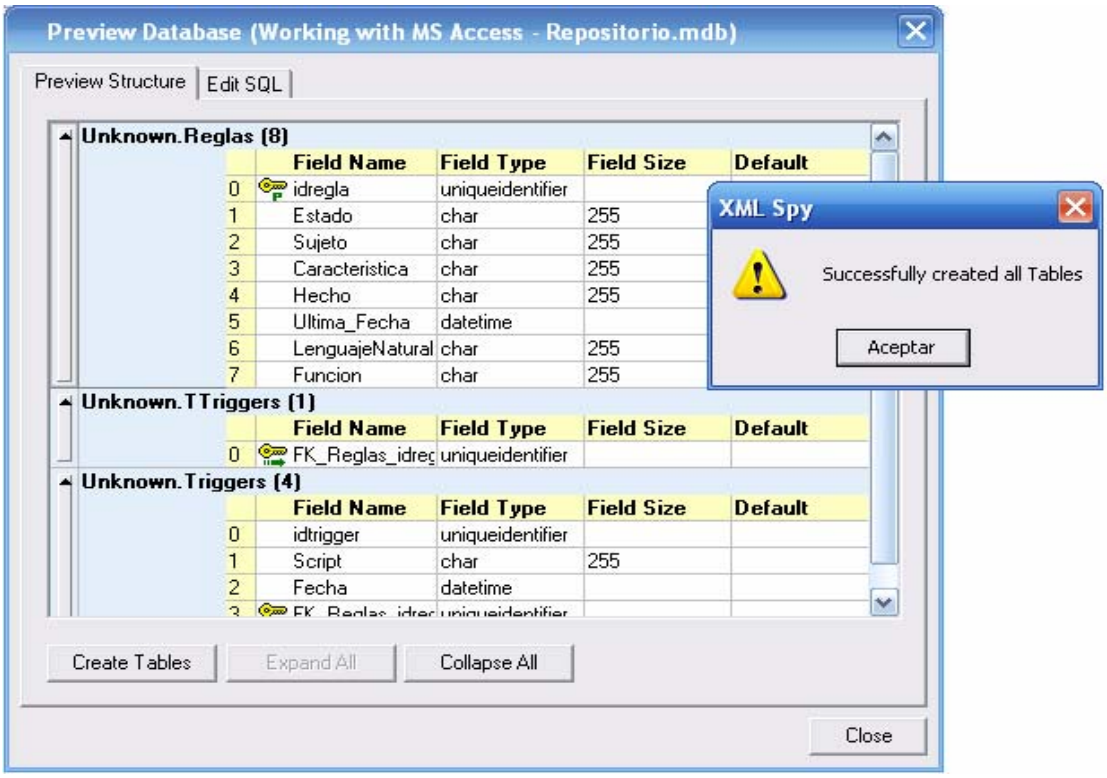


Ilustración III.6 Estructura de la Base de Datos basado en el esquema.

Para exportar las reglas hacia una base de datos se puede en el menú **Convert** escoger la opción **Export to Text files / Database...** (Altova, 2004).

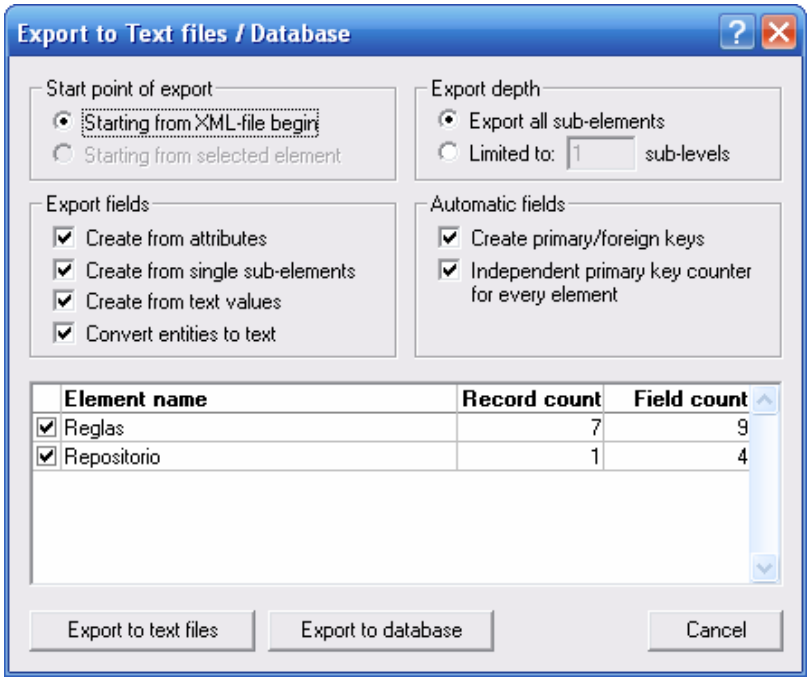


Ilustración III.7 Exportar del XML a la Base de Datos.

III.3. El Editor para Reglas de Negocio.

En esencia este software se realizó para el negocio de Transplante Renal en el Hospital Universitario “Arnaldo Milián Castro” de la ciudad de Santa Clara, pero es un editor independiente al negocio, y sólo necesita pequeños ajustes para trasladarlo a otro negocio.

III.3.1. Análisis y Diseño.

Esta aplicación fue concebida para los especialistas del negocio, que conociendo el lenguaje técnico, pueden trabajar en las reglas tal como muestra el siguiente diagrama de casos de uso:

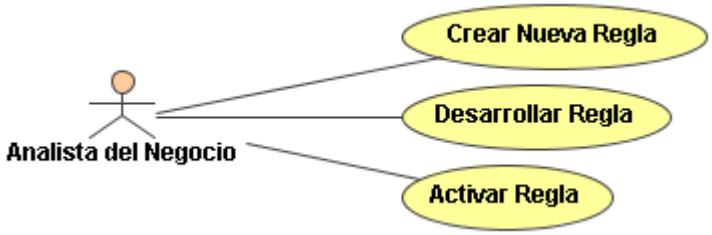


Ilustración III.8 Casos de uso del Editor.

Para el caso de uso **Crear Nueva Regla** el diagrama de estado correspondiente sería:

Ilustración III.9 Diagrama de Estado para Crear Nueva Regla.

Seguidamente el diagrama de estado para el caso de uso **Desarrollar Regla**.

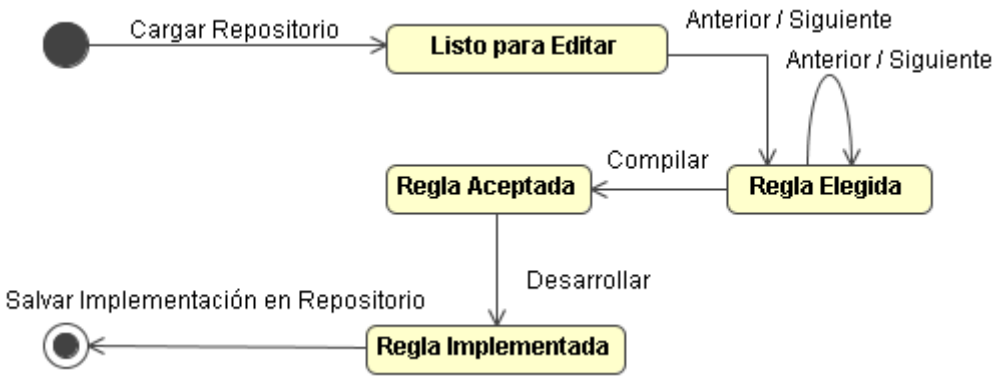


Ilustración III.10 Diagrama de Estado para Desarrollar Regla.

Y finalmente el diagrama de estado para el caso de uso **Activar Regla**.

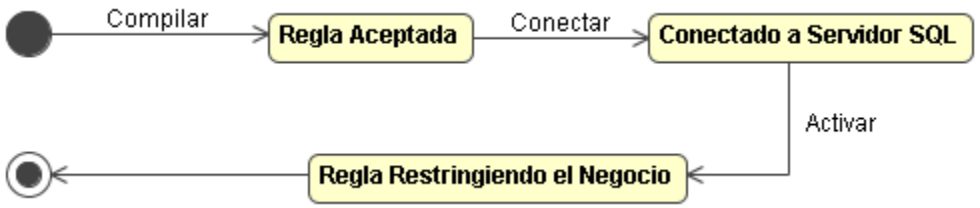


Ilustración III.11 Diagrama de Estado para Activar Regla.

En los anexos dos y tres se muestran el diagrama de clases y el diagrama de componentes respectivamente, los cuales brindan la información final para interactuar con la aplicación.

El editor utiliza un repositorio con las reglas de restricción expuestas en esta investigación, confeccionadas con parte del esquema E/R para Transplante Renal. Ver Anexo 4 y Anexo 5.

La plataforma de desarrollo para esta aplicación fue el Delphi7, con el cual se realizó un proyecto que consta de nueve units y cuatro formas para los casos de uso del editor.

III.3.2. Manual de usuario.

El ambiente es sencillo y de fácil manejo, se compone de un cuadro de chequeo y siete cajas de edición, siendo cuatro de selección. Este ambiente simula el patrón de restricción para las reglas de este tipo y primeramente comienza con el nombre de la regla editada, que es su identificador del repositorio o automático si es creada en el editor. La primera caja de edición es para el <determinante> según el mejor sentido de la regla, no cumple ningún otro rol en el posterior desenlace; la segunda pertenece al <sujeto> de la regla; la tercera a las <características> y la cuarta a los <hechos>. Las tres restantes cajas de edición son de ayuda para el analista, ya que les brindan las entidades del negocio sus atributos y los operadores y funciones factibles en el lenguaje técnico.

El cuadro de chequeo ofrece la opción de jugar con el sentido de la regla suprimiendo palabras con su fuente de color en rojo para lograr introducir o no los <hechos>, por defecto está deshabilitada la elección de incluirlos a la regla, resultando lo siguiente en el patrón:

<determinante> <sujeto> no puede tener <características>

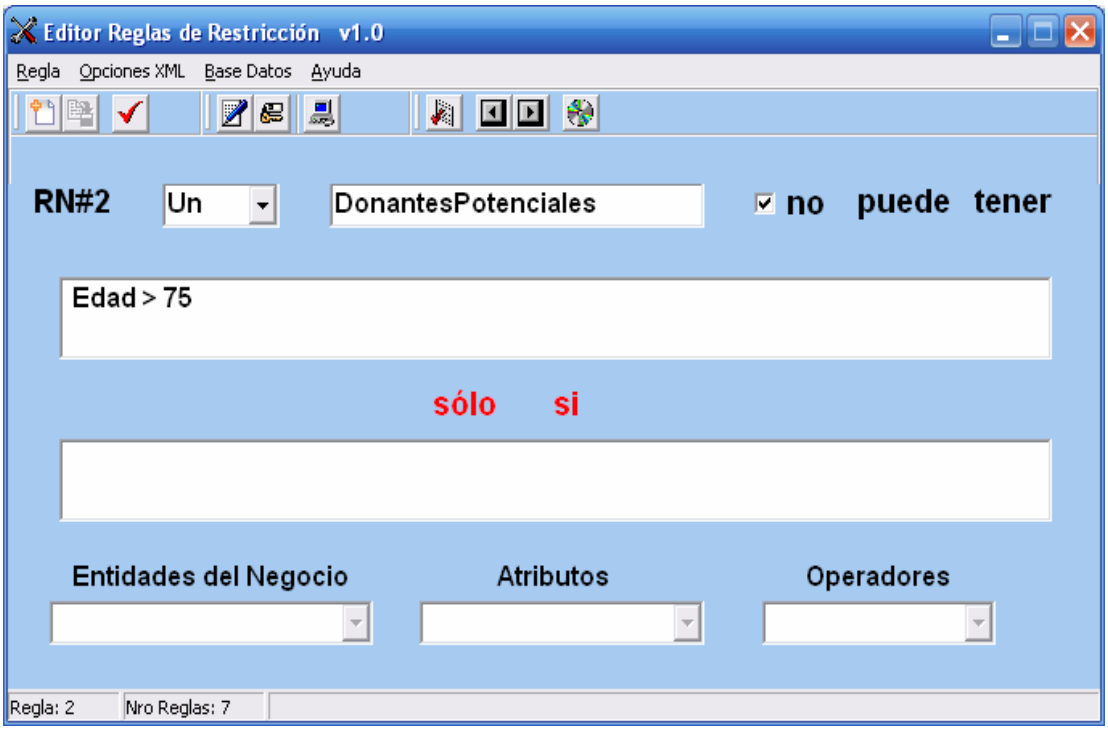


Ilustración III.12 Ambiente de Edición.

Primeramente, después de exhibir una breve presentación, la aplicación muestra la ventana principal del editor, donde se aprecian los siguientes menús relacionados a continuación:

- Regla.
- Opciones XML.
- Base Datos.
- Ayuda.

Ilustración III.13 Menú del Editor.

El menú **Regla** es el encargado del trabajo con las reglas, para esto cuenta con cuatro opciones, con las cuales el analista puede crear nuevas reglas no implementadas, compilarlas y guardarlas en el repositorio tal como se muestra.

Ilustración III.14 Menú Regla.

Ilustración III.15 Barra de Herramientas para Reglas.

El menú de **Opciones XML** brinda cuatro posibilidades de trabajo con el repositorio. En un inicio se carga este de un archivo XML, posteriormente facilita el desplazamiento por las reglas de tipo restricción no implementadas para dar la opción de guardar la implementación en el repositorio, después de compilada la regla satisfactoriamente.

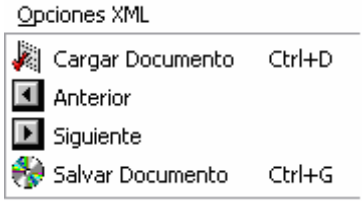


Ilustración III.16 Menú Opciones XML.

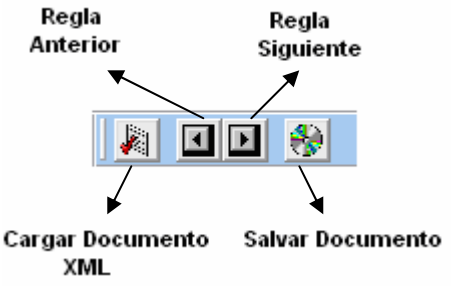


Ilustración III.17 Barra de Herramientas para Opciones XML.

En la barra de estado del editor se encuentra la posición actual del usuario en su movimiento por las reglas no implementadas del repositorio, indicando específicamente la actual dentro del total de reglas.

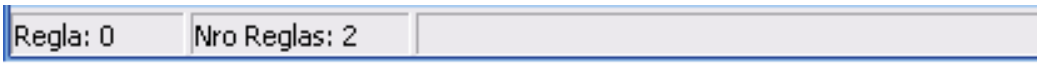


Ilustración III.18 Barra de Estado del Editor.

El menú de **Base Datos** brinda cinco opciones. Inmediatamente después de haber cargado las reglas del repositorio, ofrece la opción de escoger cuál es el lenguaje deseado para generar el script de la base de datos, por defecto aparece el dialecto Transac-SQl pero también se puede generar SQL-Estándar.

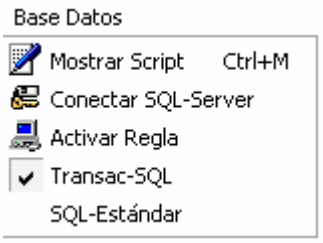


Ilustración III.19 Menú Base de Datos.

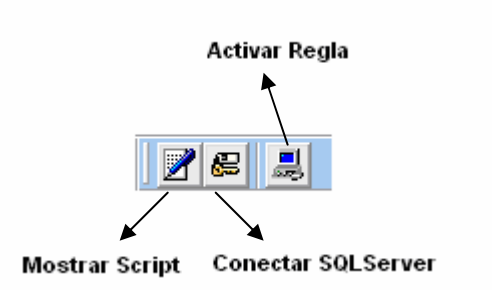


Ilustración III.20 Barra de Herramientas para Base de Datos.

Inmediatamente después de compilar satisfactoriamente la regla se puede mostrar el script generado mediante una ventana no editable, aquí se muestran la función que implementa la regla y los triggers que la utilizan como se puede notar:

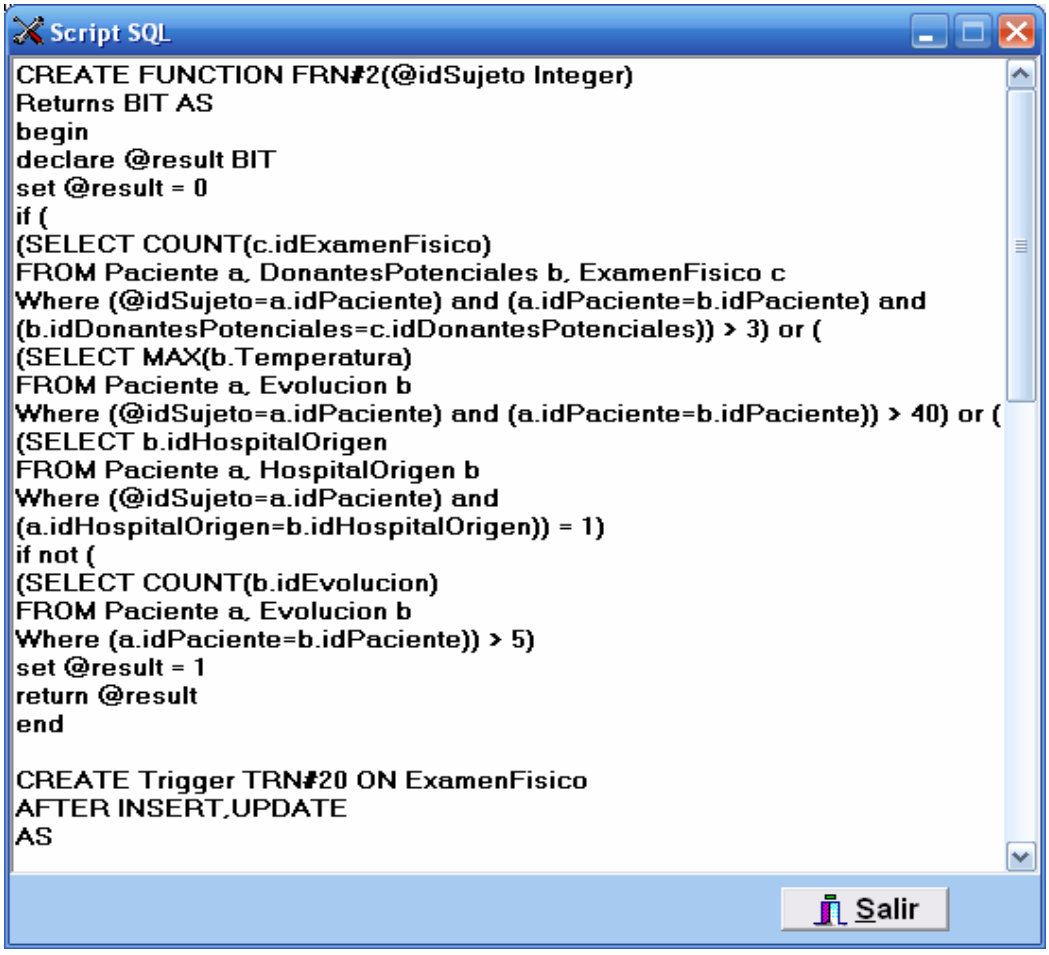


Ilustración III.21 Muestra del Script para la Base de Datos.

Si el lenguaje escogido a generar es Transac-SQL entonces habilita la opción de conexión a la base de datos del negocio y permite comprobar que funcione correctamente:

Ilustración III.22 Conexión a la Base de Datos.

Si la conexión es satisfactoria entonces se puede activar dicha regla, por activar se entiende almacenar en la base de datos del negocio los triggers y la función generada, quedando así la regla totalmente activa en la base de datos del negocio.

La aplicación también posee tres barras de herramientas con la indicación sobre el uso de los botones para una mayor comodidad en el uso del editor.

La aspiración a extender este editor en futuras versiones requerirá desarrollar una ayuda que carece de significado en estos momentos.

III.4. Conclusiones Parciales.

De esta forma se culmina este capítulo obteniendo una aplicación sencilla y útil que almacena e implementa reglas de tipo restricción. Aplicación que utiliza un compilador para generar el script de la base de datos del negocio mediante una regla en lenguaje técnico. Además de tomar las reglas de un repositorio en formato XML y validarlo por un esquema XSD.

Conclusiones.

Para automatizar las reglas de negocio, tipo restricción:

- Se definió un patrón de restricción a partir de uno ya existente.
- Se utilizaron los recursos en base de datos que posibilitan la implementación de reglas de negocio, tipo restricción.
- Se determinó la estructura de almacenamiento para reglas de negocio, tipo restricción.
- Se creó una aplicación para editar reglas en lenguaje técnico y generarlas automáticamente.

Recomendaciones.

Para investigaciones y trabajos futuros se recomienda:

- Investigar como implementar todo tipo de navegación mediante la notación punto.
- Expandir la generación de reglas a otros tipos, como Clasificación.
- Extender la aplicación de forma que un usuario sea capaz de editar las reglas.
- Investigar sobre el posible empleo en la implementación de restricciones al crear la Base de Datos.

Bibliografía.

ALTOVA (2004) XMLSPY 2004 rel. 2 ed.

BESEMBEL, I. M. & CHACÓN, E. Objetos y reglas de negocios en la integración y automatización de procesos de producción continua.

BUMBLE-BEE (2000) Parser Generator 1.12 ed.

DEMUTH, B., HUSSMANN, H. & LOECHER, S. (2001) OCL as a Specification Language for Business Rules in Database Applications. UML'01 4th Intl. Conf Unified Modeling Language,. Toronto, Ontario, Canada, October 2001.

KELLENBENZ, M. J., HEIL, M. S., FONG, M. W., MIYAMOTO, M. H., URMAN, M. J., RUTT, M. T., MONTGOMERY, M. J., NELSON, M. T. J., DEESE, M. M., URQUHART, M. K., OKSALA, M. S. & O'REILLY, M. K. (September 1999) Persistent Stored Modules (SQL/PSM).

LOWENTHAL, B. (2005) Rule Enabling Applications with Oracle Business Rules.

MORGAN, T. (2002) Business Rules and Information Systems: Aligning IT with Business Goals. Addison Wesley.

MOTA, S. A. (2005) Bases de Datos Activas.

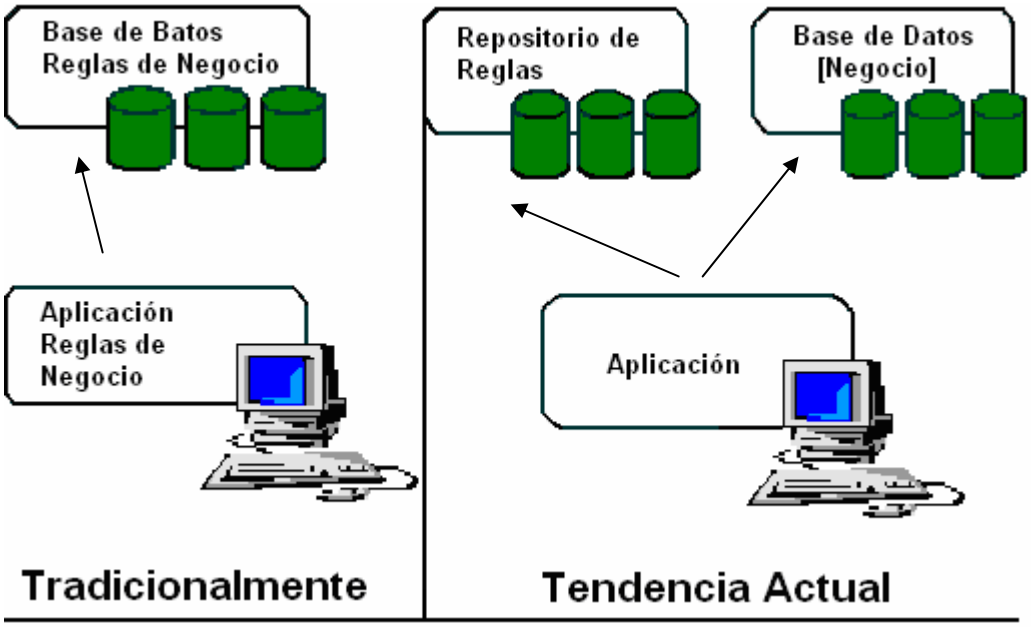
SQLSERVER2000, M. (2004) Libros en pantalla de Microsoft SQL Server. 8.0 ed.

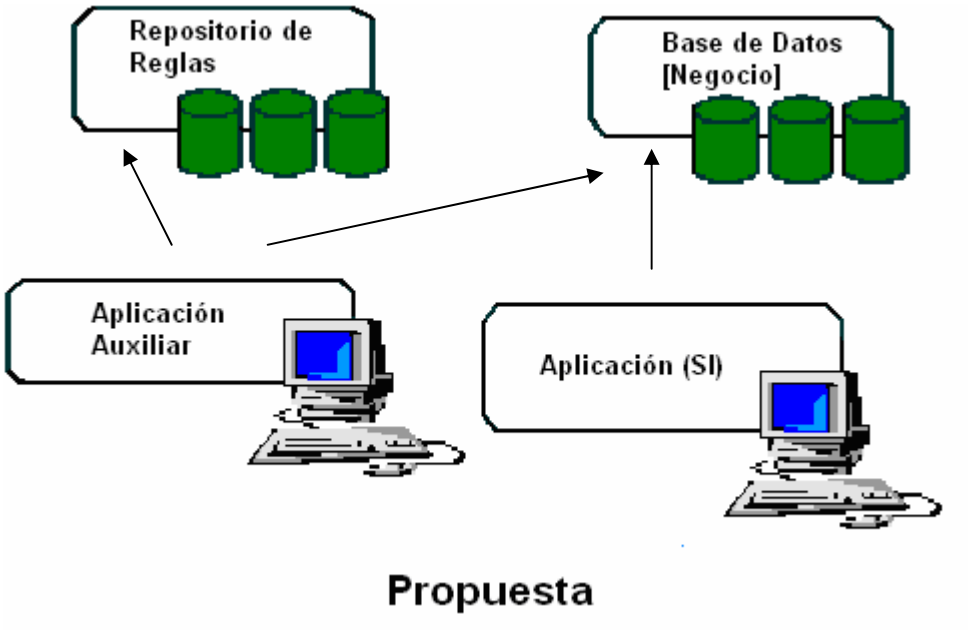
STURM, J. Desarrollo de Soluciones XML.

ZIMBRÃO, G. Enforcement of Business Rules in Relational Databases Using Constraints.

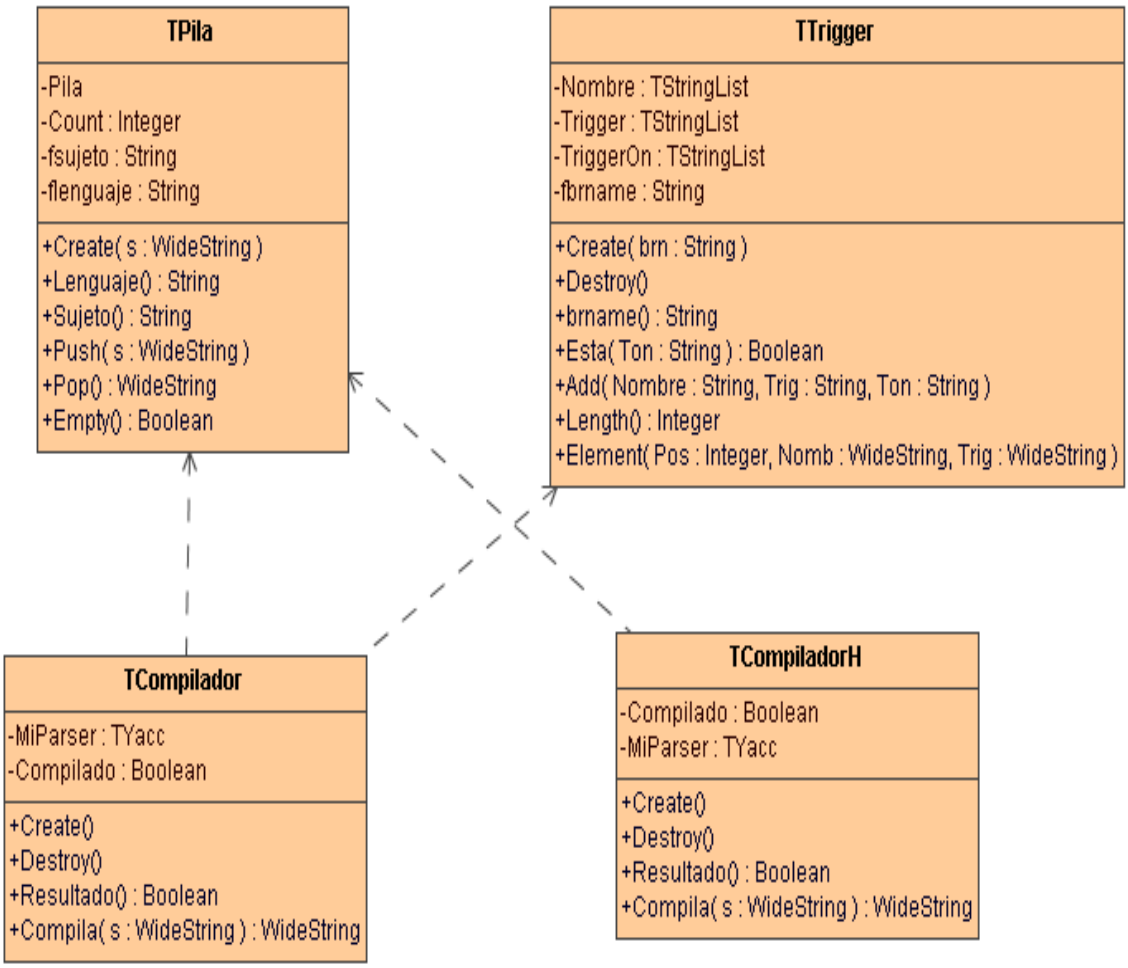
Anexos.

Anexo 1: Tradición, Tendencia Actual y Propuesta para el trabajo con Reglas de Negocio.

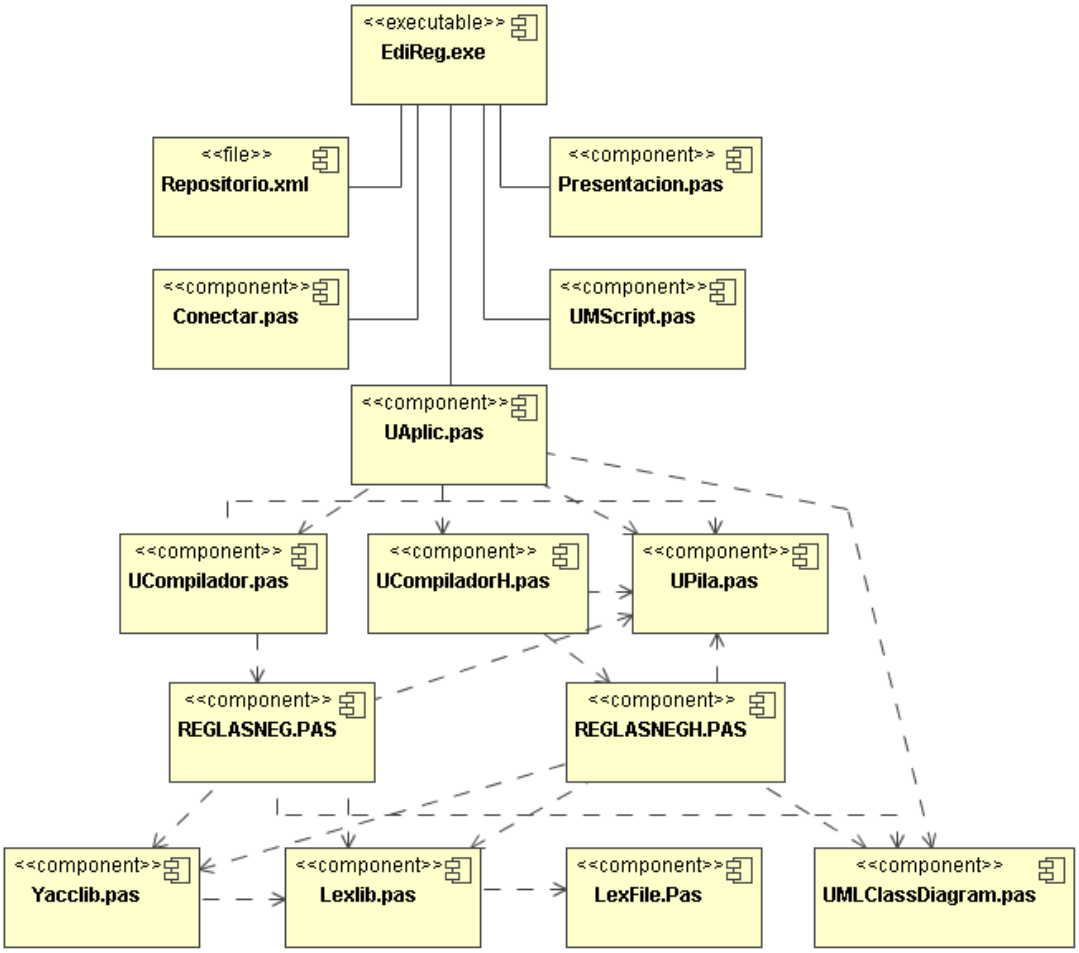




Anexo 2: Diagrama de Clases para el Editor de Reglas.



Anexo 3: Diagrama de Componentes.



Anexo 4: Parte del Repositorio utilizado para los ejemplos.

```
<?xml version="1.0" encoding="UTF-8"?>
<Repositorio xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\Documents and
Settings\Alain\Escritorio\Proyecto-Tesis\XML\Repositorio.xsd">
   <Reglas>
      <idregla>RN#1</idregla>
      <Estado>INACTIVA</Estado>
      <Sujeto>Paciente</Sujeto>
      <Caracteristica>Edad &gt; 18 and
                      sizeof(ExamenFisico.idExamenFisico) &gt;5
      </Caracteristica>
      <Hecho/>
      <Ultima_Fecha>2008-08-13</Ultima_Fecha>
      <LenguajeNatural>Un paciente no puede ser mayor de 18 años y
                  tener más de 5 exámenes físicos.</LenguajeNatural>
   </Reglas>
   <Reglas>
      <idregla>RN#2</idregla>
      <Estado>INACTIVA</Estado>
      <Sujeto>DonantesPotenciales</Sujeto>
      <Caracteristica>Edad > 75</Caracteristica>
      <Hecho/>
      <Ultima_Fecha>2008-06-13</Ultima_Fecha>
      <LenguajeNatural>Un Donante Potencial no puede tener más de 75
                       años de edad.</LenguajeNatural>
   </Reglas>
   <Reglas>
      <idregla>RN#3</idregla>
      <Estado>INACTIVA</Estado>
      <Sujeto>Paciente</Sujeto>
      <Caracteristica>
          sizeof(DonantesPotenciales.ExamenFisico.idExamenFisico) > 3
          or max(Evolucion.Temperatura) > 48
      </Caracteristica>
      <Hecho/>
      <Ultima_Fecha>2003-01-01</Ultima_Fecha>
      <LenguajeNatural>Un Paciente no puede tener Donantes
        Potenciales con más de 3 Exámenes Físicos o tener en su
        Evolución una temperatura máxima mayor de 48 grados.
      </LenguajeNatural>
   </Reglas>
   <Reglas>
      <idregla>RN#4</idregla>
      <Estado>INACTIVA</Estado>
      <Sujeto>Paciente</Sujeto>
      <Caracteristica>HospitalOrigen.Nombre = 1</Caracteristica>
      <Hecho/>
      <Ultima_Fecha>2008-08-22</Ultima_Fecha>
      <LenguajeNatural>Un Paciente no puede tener como Hospital de
                    Origen al Hospital de Matanzas.</LenguajeNatural>
   </Reglas>
</Repositorio>
```

Anexo 5: Parte del esquema E/R para Transplante Renal.