GENERACIÓN AUTOMÁTICA DE UN ESQUEMA CONCEPTUAL OO A PARTIR DE UN MODELO DE FLUJO DE TRABAJO

Hugo Estrada E.[1,2], Alicia Martínez R.[1,3], Oscar Pastor L.[1], Javier Ortiz H.[2], Octavio A. Ríos T.[4]

[1] Universidad Politécnica de Valencia, Departamento de Sistemas Informáticos y Computación, Valencia, España,
Email: {hestrada,opastor, alimartin}@dsic.upv.es

[2] Centro Nacional de Investigación y Desarrollo Tecnológico, Departamento de Ciencias Computacionales Morelos, México,
Email: {hestrada, ortiz}@cenidet.edu.mx

[3] Instituto Tecnológico de Zacatepec, Departamento de Ciencias Computacionales Morelos, México,

[4] Instituto Tecnológico de Tuxtla Gutiérrez 1080 Tuxtla Gutiérrez, Chiapas.
Email:oarios@hotmail.com

Resumen. Actualmente se reconoce la importancia de incorporar información sobre procesos de negocios en la especificación de requisitos, donde el software es parte activa de este modelo de negocios. Trabajos recientes han intentado utilizar las técnicas de modelado conceptual de sistemas de software, como es el caso de UML, para modelar procesos de negocio, lo cual constituye un enfoque desafortunado, debido a que estas técnicas no permiten especificar en forma clara relaciones organizacionales entre actores también organizacionales. En este artículo se presenta una aproximación para generar un modelo de flujo de trabajo (como punto de inicio para el desarrollo de los modelos estratégicos del framework i*), a partir de una plantilla de requisitos organizacionales, posteriormente este modelo de flujo de trabajo es traducido a un esquema conceptual del cual es posible obtener implementaciones del sistema software en forma automática utilizando la herramienta CASE OO-Method.

Palabras Clave: modelado de procesos de negocio, ingeniería de requisitos, especificación formal, modelos de flujo de trabajo.

1. Introducción

Uno de los grandes retos de la Ingeniería de Software a partir de su aparición a finales de los 60's, ha sido la producción de software que realmente satisfaga las necesidades del usuario. A pesar de los notorios avances que se han tenido en los últimos 30 años en el desarrollo de técnicas y metodologías para lograr mejores resultados en las diversas etapas del ciclo de vida del software, aún es alto el índice de proyectos que no

resultan a la entera satisfacción del cliente. La inhabilidad para producir requisitos de software que sean completos, correctos y sin ambigüedad, aún es considerada como la principal causa de que los productos de software sean incongruentes con las necesidades de los usuarios[1][2]. En años recientes se ha hecho patente la necesidad de contar con métodos, en el sentido estricto de la ingeniería, para capturar, elicitar y modelar estas necesidades, surgiendo así la Ingeniería de Requisitos como un proceso previo a la Ingeniería de Software, cuya función es determinar y especificar la forma de respuesta de una entidad ante los estímulos que se le presentan en un determinado ambiente operacional, haciendo notar que al hacer referencia a una entidad que es un objeto de negocios se habla de Ingeniería de Requisitos Organizacionales (etapa temprana de la ingeniería de requisitos), y al referirse a la entidad como un objeto software, entonces hablamos de Ingeniería de Requisitos del Sistema de Software (etapa tardía). Una clasificación más detallada de los requisitos muestra que éstos pueden ser establecidos en tres niveles de abstracción [3]: nivel de requisitos organizacionales, de requisitos del sistema y de especificación del sistema, cada una de estas etapas tiene una contraparte tecnológica que permite trabajar con los conceptos antes definidos en la clasificación:

- En el nivel de especificación del sistema se encuentran muchos métodos semiformales como Objectory, OMT, Fusion, Coad-Yourdon, y UML que proveen de ambientes que permiten realizar una descripción de un esquema conceptual funcional del sistema en términos de clases y objetos. También han sido propuestas notaciones formales en este nivel, tales como RML [4], OBLOG [5], TROLL [6], LCM [7] y OASIS [8].
- En el nivel de requisitos del sistema se pueden encontrar muchos métodos semiformales para lo que se ha denominado Análisis de Requisitos orientado a Objetos, donde se ha hecho extensivo el uso de casos de uso y escenarios con el objetivo de describir el rol del sistema desde la perspectiva de los usuarios externos. En esta etapa han sido utilizados también los diagramas de contexto, de secuencia y los diagramas entidad-relación. Ejemplos de lenguaje formales en esta etapa son: GIST [9] y ALBERT [10], este último permitiendo realizar el modelado funcional desde una perspectiva orientada a agentes.
- En el nivel de requisitos organizacionales han sido desarrolladas notaciones por los proyectos Esprit II y Esprit III [11], además de los trabajos del grupo de J. Mylopoulos en la universidad de Toronto, Canadá que han desarrollado el framework i* [12].

Hoy en día se hace patente la necesidad de modelar los sistemas de software a partir de modelos organizacionales que reflejen las razones estratégicas de la existencia de ese producto software en un determinado proceso de negocios, en este sentido, el modelado de procesos de negocios es una práctica que permite representar diferentes aspectos de una organización. Existen diferentes métodos y técnicas para modelar negocios, cada uno con un objetivo específico [13][14][15]. En la investigación presentada en este artículo se hace uso de una de estas técnicas de modelado de procesos de negocio: framework i* [12], el cual utiliza conceptos de agentes con relaciones y razonamientos estratégicos.

La ingeniería de requisitos se ha visto fortalecida por el uso de métodos formales, que contribuyen a obtener especificaciones más precisas y menos ambiguas [16]. Recientemente los métodos formales están siendo aplicados a la ejecución automática de especificaciones, y también a la generación automática de prototipos de sistemas de información a partir de esquemas conceptuales, un ejemplo de ello es el uso del lenguaje formal Oasis [8]. Estos trabajos llevaron a explorar el uso de este lenguaje formal en la etapa de modelado de procesos de negocios, bajo la hipótesis de que los requisitos para un sistema de información tienen su origen en los requisitos del proceso de negocio.

En este artículo se presenta un método que parte de requisitos organizacionales; especificados en una plantilla de requisitos[17]; para crear un modelo de la organización (Modelo de flujo de trabajo) que pueda posteriormente ser traducido a un esquema conceptual descrito formalmente en lenguaje OASIS, el cual permite ya sea la animación de la especificación [18], o la generación de código en un lenguaje seleccionado (Visual Basis, Java, Delphi) [19]. Este trabajo forma parte de la unión de dos ambientes integrados de desarrollo: el proyecto IRIS (Ingeniería de requisitos), desarrollado en CENIDET en Cuernavaca, Morelos, México que tiene como objetivos la fase temprana de requisitos contemplando desde la elicitación de requisitos hasta la generación automática del modelo conceptual, y del proyecto OO-Method, desarrollado en la Universidad Politécnica de Valencia, España, cuya función es generar o completar un modelo gráfico de un esquema conceptual generado (en el que se incluyen aspectos dinámicos y estáticos en la forma de restricciones de integridad, pre y poscondiciones, relaciones de herencia, de herencia, de especialización, etc) y producir, en forma automática, aplicaciones ejecutables en ambientes imperativos.

2. El modelo de flujo de trabajo y su relación con los modelos del framework i*

Como se mencionó anteriormente, muchas técnicas que originalmente fueron creadas para describir sistemas de información, se empezaron a utilizar en modelado de procesos de negocio, aún y cuando estas técnicas están mal equipadas para capturar requisitos que tiene que ver con objetos de negocio que persiguen metas organizacionales. Las técnicas más conocidas bajo este enfoque son el Análisis Estructurado y Diseño Estructurado (SA/SD), la definición de manufactura asistida por computadora integrada (IDEF), y la Técnica de análisis y diseño estructurado (SADT). Estas técnicas son muy parecidas, en el sentido de que utilizan notaciones semejantes al Diagrama de Flujo de Datos [13]. Más recientemente han utilizado los modelos orientados a objetos como OMT [20], Martin/Odell [21] y UML. Estas formas de modelado tienen en común un enfoque orientado a capturar el "qué" operaciones o actividades se realizan. Sin embargo actualmente están desarrollándose técnicas que permiten modelar una organización con notaciones más cercanas a este dominio del problema, y entender en forma clara el “por qué” de las operaciones o actividades, tal es el caso del framework i*, del cual se dan detalles a continuación.

2.1 Modelos estratégicos de i*

En la Universidad de Toronto se ha desarrollado un framework denominado i*[12], que permite obtener modelos organizacionales de empresas enfocándose en las metas que persigue cada proceso de negocios. En este marco de trabajo las organizaciones están constituidas por actores sociales los cuales tienen libertad de acción, pero dependen de otro actor para lograr sus metas, para desarrollar sus tareas y para que se le suministren recursos.
Se ha elegido este framework porque permite modelar los procesos intencionales y capturar las motivaciones para llevar a cabo las actividades, además brinda un mejor entendimiento del "por qué", mediante relaciones del modelado organizacional, que son la base para los requisitos del sistema.
Este marco de trabajo incluye dos modelos: modelo de dependencias estratégicas y el modelo de razonamiento estratégico.

Modelo de Dependencias Estratégicas
Modela las dependencias que existen entre los actores para alcanzar sus metas, ejecutar sus tareas y proporcionar recursos a otros actores. Este modelo provee una descripción intencional de los procesos, en lugar de los usuales modelos de entidades y actividades (no intencionales y no estratégicos), creando una caracterización de alto nivel de los procesos.
Este modelo se representa a través de un grafo en donde cada nodo equivale a un actor y las ligas indican las dependencias que existen entre los actores para alcanzar sus metas, ejecutar tareas y suministrar recursos(ver figura 1).

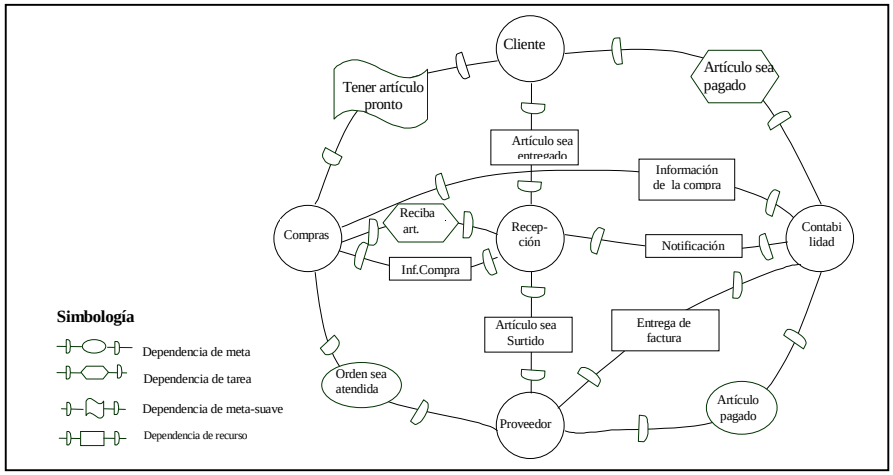

Figura 1. *Ejemplo de un Modelo de Dependencia Estratégica [12]*

El conjunto de nodos y ligas en el modelo de Dependencias Estratégicas forman una red de dependencias, que ayudan a representar gráficamente las relaciones externas entre actores.

La figura 1 muestra un ejemplo del modelo de dependencias estratégicas para un proceso de adquisición de bienes. Algunas dependencias que se pueden observar son: El cliente depende de compras para lograr su meta de tener un artículo. Compras, a su vez, depende del proveedor para lograr la meta de entregar el artículo al cliente, y también del departamento de Recepción, para que se realice la tarea previamente definida mediante un procedimiento, de recibir el artículo [12, 22].

Modelo de Razonamiento Estratégico

Este modelo permite una descripción más detallada de los razonamientos detrás de las dependencias, es decir describe las relaciones intencionales que son internas a los actores.

Este modelo contiene los conceptos de "liga de medios-fines", que indica la relación que existe entre un fin y el medio que requiere para lograrlo, y "liga tarea-descomposición" describe los componentes de una tarea en términos de submetas, subtareas y recursos, toda esta información es necesaria para poder aplicar conceptos de reingeniería de procesos de negocio.

2.2 El Modelo de Flujo de trabajo (Workflow)

El modelo de dependencias permite descubrir las oportunidades, dependencias y vulnerabilidades que tiene cada actor respecto a los demás. Para llegar a un modelo con estas características es necesario partir de una descripción del proceso de negocios más simple, que permita tener una clara idea de los actores que intervienen y de la forma en que se relacionan, para este trabajo de investigación se ha elegido un Modelo de Flujo de Trabajo.

Como se buscaba un tipo de Workflow[23] que tuviera ciertas cualidades que lo hicieran compatible con los modelos del framework i*, se eligió uno que estuviera basado no sólo en el flujo de trabajo, sino en los artefactos o productos que fluyen en el trabajo, los cuales puedan después traducirse (en el modelo de dependencias estratégicas), en el objeto que hace depender a un actor de otro, a este tipo particular de Workflow le llamamos *Modelo de Flujo de Productos de Trabajo (MFPT)*[23]. La idea básica de utilizar el MFPT fue la factibilidad de desarrollar una herramienta que aprovechara las buenas propiedades y potencialidades de los modelos Workflow [24], por ejemplo, contar con una sintaxis visual bien definida, la flexibilidad de reconfiguración, la capacidad de reificar entidades de alto nivel en entidades más específicas, etc, y que además abriera el camino para trabajar con los razonamientos de los modelos de dependencias estratégicas y razonamiento del framework i*. Se opto por generar un modelo de flujo de trabajo en lugar de generar un modelo de dependencias debido a que estimamos que la creación de éste último requiere de un mayor grado de conocimientos por parte de los ingenieros de requisitos, ya que en los modelos estratégicos de i* existen dependencias que no siempre son productos de trabajo, sino relaciones intencionales entre actores, por otra parte, las capacidades y potencialidades de los sistemas Workflow permiten bajar hasta un nivel de abstracción en el cual cada objeto es una persona o un sistema software, siendo este nivel el que nos intere-

sa ya que tomando las definiciones de actores y productos es posible generar un esquema conceptual OO-Method.
Es necesario comentar que la herramienta que se presenta en este artículo no tiene incorporado un motor de WorkFlow, ya que no se necesitaba ejecutar el WorkFlow sino transformar cada entidad en una clase de modelo orientado a objetos con una signatura compuesta por estado y comportamiento.

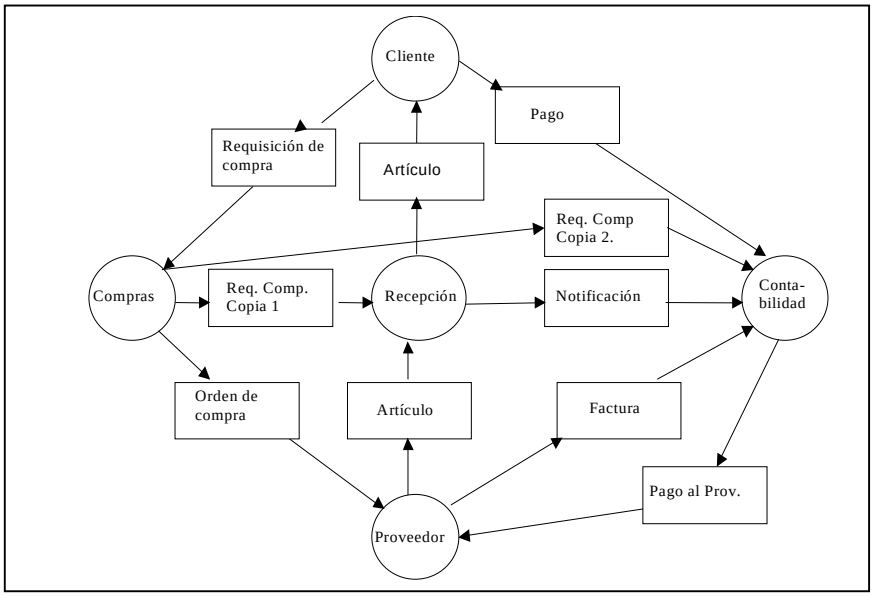

Figura 2. *Modelo de Flujo de Productos de Trabajo*

Un MFPT se puede describir de la siguiente manera: en un proceso de negocio, hay *actores* que realizan actividades que generan productos, denominados *artefactos*, éstos serán utilizados por otros *actores*. Las *Relaciones* entre *actores* se definen en términos de la entrega de *artefactos*. Un Actor se relaciona con otro, en el sentido de que entrega o recibe de él un Artefacto. En la figura 2, se presenta un ejemplo de un MFPT para un caso de estudio específico como es la *Adquisición de Bienes.*
Basados en estos elementos, es posible definir una relación entre dos actores, representada por Rel(Actor1, Actor2, Artefacto), como un lazo de unión entre Actor1 y Actor2, en el sentido de que Actor1 realizará una actividad con la cual obtendrá un Artefacto, el cual es enviado a Actor2 para que éste logre sus propias metas y las metas generales del negocio. Un análisis más profundo determina [25] el tipo de dependencia de la que se trata, y si existen otras dependencias en esa misma relación. De esta manera, el MFPT da una pauta inicial para producir un Modelo de Dependencia. Así, un proceso especificado en un MFPT es considerado como un plan para alcanzar una meta general del proceso de negocio [12].

3. OO-Method

OO-Method [19] es un método de producción automática de software basado en técnicas de especificación formal. La etapa del proceso de producción de software inicia con la fase de modelado conceptual donde se recogen con modelos gráficos las propiedades relevantes del sistema a desarrollar. Una vez que se tiene la descripción del sistema se obtiene de forma automática una especificación formal y OO del sistema. Esta especificación es la fuente de un modelo de ejecución que determina de forma automática las características del sistema dependientes de la implementación, ésta proporciona un marco bien definido que permite construir herramientas de modelado y generación de código, basadas en el paradigma de programación automática.

4. Capturando requisitos con un MFPT

Como ya se ha mencionado, el objetivo de este trabajo fue generar un MFPT (como punto de inicio para el desarrollo de los modelos estratégicos de la metodología i*), a partir de una plantilla de requisitos organizacionales, este modelo se traduce a un esquema conceptual del cual es posible obtener implementaciones completas del sistema software en forma automática utilizando la herramienta CASE OO-Method.
Para lograr semi-automatizar este proceso fue necesario desarrollar una etapa previa que consistió en determinar un método y un formato para especificar requisitos organizacionales, el resultado fue una herramienta [17] que produce una plantilla de requisitos, la cual fue desarrollada utilizando las teorías de análisis de discurso de Gross y Zinder [26] para la etapa de adquisición de conocimiento, y la plantilla de requisitos del método Volere [27] para la estructuración de la información. El sistema permite seleccionar el Léxico Extendido del Lenguaje (LEL) [28] de un dominio de negocios específico con el cual poder estructurar el discurso. Una vez determinado el LEL, el sistema comienza a formular preguntas ya sea al ingeniero de requisitos o al usuario final guiando el discurso para obtener respuestas que permitan obtener la información para crear un MFPT, de tal forma que el ingeniero de requisitos o el futuro usuario del sistema expresan a través de la interacción con el sistema, los nombres de procesos, de actores, sus relaciones, etc. El discurso obtenido por la interacción usuario-sistema es analizado generando múltiples segmentos y aplicando en cada uno de estos las reglas de análisis del discurso de Gross, obteniéndose una especificación formal de estos requisitos, como paso final, el sistema traduce la especificación formal de requisitos en una plantilla que es la entrada de la herramienta descrita en este artículo.

4.1 La plantilla de requisitos

La plantilla [17] está organizada en 4 secciones (figura 4): a) Datos generales, b) Detalles de los datos de artefactos, c) Detalles de actores y d) Detalles de identificación. En la primera de las secciones (*Datos generales*) en el punto 1.0 se encuentra el nombre del proceso de negocio. En el punto 2.0 se ubica la información de cada uno de los artefactos o productos de trabajo, los cuales se detallan haciendo uso de la sección b) *Detalles de artefactos*, haciendo notar que para cada artefacto se deben

repetir los puntos 2.1 al 2.7. En el punto 3.0 se describen los datos para los actores, utilizando para ello la sección c) *Detalles de actores*, donde para cada actor se deben repetir los puntos 3.1 al 3.5. La sección d) *Detalles de los datos para identificación*, se utiliza para describir los puntos 2.3 y 3.3, que corresponden a los datos de identificación del artefacto y del actor respectivamente. Normalmente se utiliza un solo dato de identificación, pero queda abierta la posibilidad de utilizar más de uno. Por ejemplo, un cheque bancario podría ser identificado por su número de folio solamente, o por el número de folio y el nombre del banco que lo emite.

Cada sección está organizada en cuatro columnas. La primera columna contiene la numeración progresiva de los datos. La segunda contiene el nombre que se ha asignado a cada dato, esto con el fin de poder hacer referencia a ellos en las reglas de transformación. La tercera columna contiene una descripción de cada dato, y la cuarta columna contiene los espacios a ser llenados con los valores de cada uno de estos datos.

La tabla2 muestra el contenido de la plantilla de requisitos para un fragmento del proceso de *Adquisición de Bienes* mostrado en la figura 2. Se muestra solamente una pequeña parte de este proceso de negocios con la finalidad de hacer más sencilla la explicación del método.

Línea	**Ref. Plantilla**	**Descripción**	**Valor**
1	1.0	Nombre del Proceso	Adquisicion_de_bienes
2	2.1	Nombre del Artefacto	Req_Compra
3	2.2	Mecanismo de Identificación	Clave
4	2.3.1	Nombre del Dato de Identificación	numRequisicion
5	2.3.2	Tipo del Dato	int
6	2.4	Acción de creación	Elaborar
7	2.5	Acción de Eliminación	Cancelar
8	2.6	Nombre del actor fuente	Cliente
9	2.7	Actor Receptor	Compras
10	2.1	Nombre del Artefacto	Orden_Compra
11	2.2	Mecanismo de Identificación	clave
12	2.3.1	Nombre del Dato de Identificación	numero
13	2.3.2	Tipo del Dato	int
14	2.3.1	Nombre del Dato de Identificación	Fecha_emisión
15	2.3.2	Tipo del Dato	date
16	2.4	Acción de creación	Elaborar
17	2.5	Acción de Eliminación	Cancelar
18	2.6	Nombre del actor fuente	Compras
19	2.7	Actor Receptor	Proveedor
20	2.1	Nombre del Artefacto	Articulo
21	2.2	Mecanismo de Identificación	Clave
22	2.3.1	Nombre del Dato de Identificación	Numero
23	2.3.2	Tipo del Dato	Int
24	2.4	Acción de creación	dar_Alta

25	2.5	Acción de Eliminación	dar_Baja
26	2.6	Nombre del actor fuente	Proveedor
27	2.7	Actor Receptor	Recepción, Cliente
28	3.1	Nombre del actor	Cliente
29	3.2	Mecanismo de Identificación	Clave
30	3.3.1	Nombre del Dato de Identificación	NumCliente
31	3.3.2	Tipo del Dato	Int
32	3.4	Acción de creación	dar_Alta
33	3.5	Acción de Eliminación	dar_Baja
34	3.1	Nombre del actor	Compras
35	3.2	Mecanismo de Identificación	Nombre
36	3.3.1	Nombre del Dato de Identificación	Nombre
37	3.3.2	Tipo del Dato	String
38	3.4	Acción de creación	dar_Alta
39	3.5	Acción de Eliminación	dar_Baja
40	3.1	Nombre del actor	Proveedor
41	3.2	Mecanismo de Identificación	Clave
42	3.3.1	Nombre del Dato de Identificación	NumProveedor
43	3.3.2	Tipo del Dato	Int
44	3.4	Acción de creación	dar_Alta
45	3.5	Acción de Eliminación	dar_Baja
46	3.1	Nombre del actor	Recepcion
47	3.2	Mecanismo de Identificación	Nombre
48	3.3.1	Nombre del Dato de Identificación	Nombre
49	3.3.2	Tipo del Dato	String
50	3.4	Acción de creación	dar_Alta
51	3.5	Acción de Eliminación	dar_Baja

Tabla 1 Plantilla de datos para MFPT de un proceso de negocio

Como puede observarse en la plantilla, los actores que intervienen en este proceso son *Cliente, Compras, Proveedor y Recepción*. Los artefactos que intervienen son *Requisición de Compra, Orden de compra y Artículo.* El artefacto *Requisición de compra* es enviado por el *Cliente a Compras*, quien a su vez, envía una *Orden de compra al Proveedor*, el cual envía el *Artículo a Recepción*; el mismo *Artículo* es enviado de *Recepción a Cliente*. En este caso aparece un artefacto que tiene más de un actor receptor; se trata de Artículo, el cual nace en *Proveedor* (actor fuente) y es recibido por los actores *Recepción y Cliente* (actores receptores).
La información de la plantilla es cargada en el sistema a través de la interfaz mostrada en la figura 3.

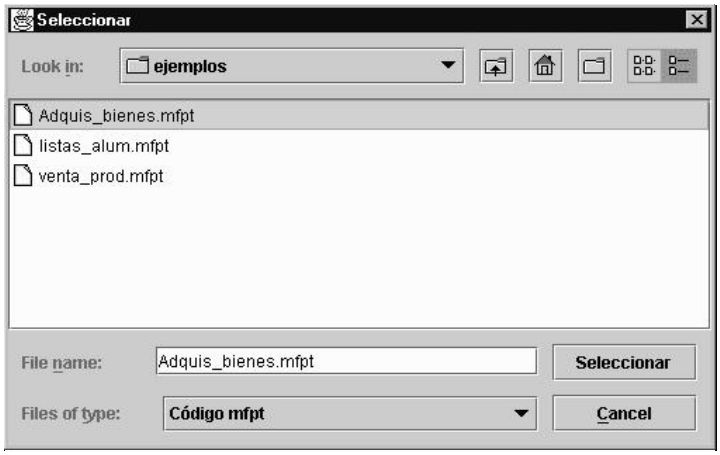

Figura 3 Ventana de selección de archivos

A partir de la información obtenida de la plantilla, el sistema estructura la información de tal forma que sea posible traducir los requisitos en un esquema gráfico de un MFPT, donde el usuario puede modificar, agregar o eliminar actores, productos de trabajo o relaciones, realizando las validaciones pertinentes para tener siempre un MFPT correcto. Esta parte de la herramienta CASE se encuentra completamente terminada (figura 4) y actualmente está desarrollándose la interfaz y el mecanismo de refinamiento necesario para trabajar con el MFPT a distintos niveles de abstracción, de tal modo que cada actor o producto pueda refinarse en entidades de más bajo nivel. La figura 4 muestra el resultado del modelo gráfico del MFPT para el ejemplo del proceso *de Adquisición de Bienes*.

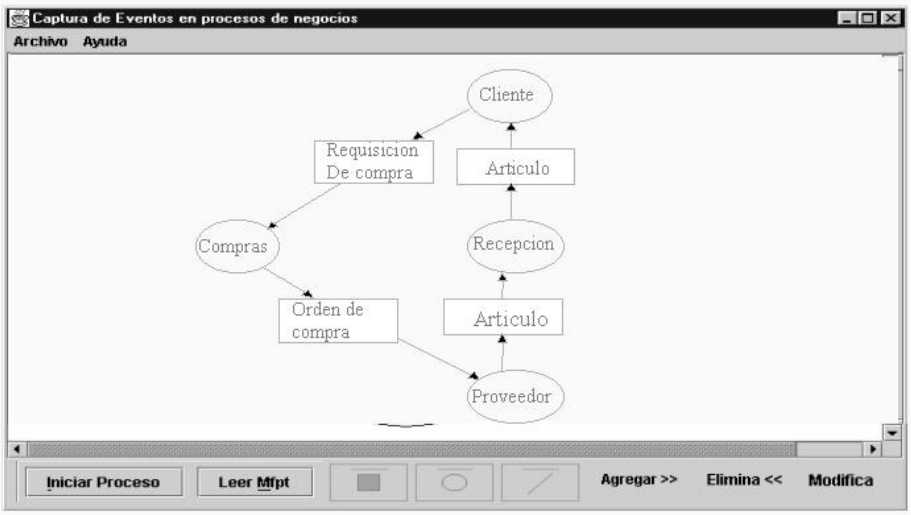

***Figura 4**. Ambiente de visualización y modificación del MFPT*

4.2 Validación de los datos de la Plantilla

Inicialmente se determinó que la plantilla de requisitos no necesariamente tendría que ser llenada en forma automática por la herramienta descrita anteriormente, sino que podría ser llenada manualmente por un usuario, por lo tanto, es posible que se lleguen a presentar algunas inconsistencias, o que la especificación no esté completa, por lo que la herramienta realiza las validaciones al momento de leer los datos de una plantilla.

4.3 Elementos del lenguaje OASIS usados en la transformación.

Una tarea clave de este trabajo, fue definir un conjunto de reglas de transformación que permita transformar una especificación de un proceso de negocios representada en MFPT, a una especificación del esquema conceptual en lenguaje Oasis.
Un MFPT describe sólo aspectos estáticos de un proceso de negocio, ya que no se muestra un modelo dinámico que exprese los cambios de estado que le acontecen a los actores por la ocurrencia de eventos. El lenguaje Oasis[8], en cambio, permite especificar esquemas conceptuales de sistemas software desde el punto de vista estático y dinámico, es decir, se pueden especificar las estructuras y propiedades de un sistema, y las acciones que se llevan a cabo junto con sus restricciones. Como resultado de la transformación, aparecen en el código Oasis algunas acciones de manera explícita, como las de envío de artefactos de un actor a otro.
Oasis no ve a un sistema como un proceso de negocio propiamente dicho, sino como una colección de objetos autónomos y concurrentes que se comunican a través de las acciones. Esta característica será muy útil para poder representar un proceso de negocio, ya que la forma natural en que los actores trabajan es de manera concurrente. El trabajo de transformación consiste en representar los elementos de un proceso de negocio en esa colección de objetos en Oasis, pero especificándolos de tal forma que no se pierda el sentido de la información original.
Es de hacer notar que la especificación del MFPT no utiliza todos los elementos de Oasis.

Los actores y artefactos definidos como Clases
Dentro del contexto de un MFPT, los actores son entidades que realizan actividades para generar artefactos, los cuales son enviados a otros actores. Esta situación puede ser modelada en Oasis definiendo a los actores y a los artefactos como clases, cuyas entidades se comunican entre sí. Debido a que cada actor tiene una estructura propia, y realiza acciones distintas a las que realizan otros actores, se debe definir una clase para cada actor. Este mismo razonamiento se aplica para los artefactos. El identificador para cada clase, es tomado del nombre del actor o del artefacto, según lo que se esté especificando.

El uso de un Mecanismo de Identificación
En Oasis, cada clase debe tener un mecanismo de identificación para poder hacer referencia a los objetos. Este mecanismo está compuesto por un identificador para el

mecanismo de identificación, y una lista de atributos, los cuales sirven para identificar al objeto. Por ejemplo, para la clase *Factura*, cuyas entidades son identificadas por un número de factura, la especificación Oasis es la siguiente:

```
identification
        clave:(numFactura);
```

donde *clave* es el nombre del mecanismo de identificación, y *numFactura* es el atributo con el cual son identificados los objetos. El parámetro *numFactura* se define como un atributo constante, ya que el valor que se le asigne no cambia en toda la vida del objeto. La definición del atributo en Oasis se realiza de la siguiente forma:

```
constant attributes
        numFactura : int;
```

En este caso los atributos utilizados están relacionados con el dominio del problema. En la plantilla de datos para MFPT se describe la información necesaria para este mecanismo de identificación.

Los servicios para creación y destrucción de instancias

En cada clase definida en Oasis, se pueden especificar acciones de creación y de destrucción de instancias, añadiendo al nombre de la acción, la palabra new o destroy.

Los servicios de envío de artefactos

Debido a que en un MFPT solamente se considera una vista estática del proceso de negocios, las acciones de envío de artefactos son las únicas que necesitan ser especificadas (además de las de creación y destrucción), en el lenguaje Oasis. Estas acciones aparecen indicadas en la definición de las clases de los artefactos. Por ejemplo, *Proveedor* envía una *Factura* a *Contabilidad*. Aquí existe una relación *Rel(Proveedor, Contabilidad, Factura)*. En la clase *Factura* se define una acción llamada *Enviara-Contabilidad*, la cual significa que uno de los destinatarios de la factura será *Contabilidad*. La clase *Factura* se convierte en servidora del servicio *EnviaraContabilidad*. El servicio *EnviaraContabilidad* aparecerá en la sección *Events* en la definición de la clase *Factura.*

```
Events
        EnviaraContabilidad;
```

Esta expresión indica que *EnviaraContabilidad* es uno de los servicios que ofrece la clase *Factura*.

La utilidad de las interfaces

En la sección interfaces del lenguaje Oasis se definen clases como servidoras de servicios. En la transformación de una especificación MFPT a Oasis, se crea una interfaz para cada relación entre un actor y un artefacto, y esa relación se da en términos de la creación y/o envío del artefacto a otro actor. Por ejemplo, para especificar la relación entre *Proveedor* y *Factura*, se realiza a través de la siguiente interfaz:

```
interface Proveedor  with Factura
        attributes(all);
        services(EnviaraContabilidad, CrearNueva, Cancelar)
end interface
```

Esta expresión indica que *Proveedor* actúa como cliente de los atributos y servicios que se especifican en la clase Factura. Los servicios que el *Proveedor* tiene a su disposición son: *CrearNueva*, *Cancelar* y *EnviaraContabilidad*. El *Proveedor* tiene la facultad de crear o eliminar una instancia de *Factura*, solicitando los servicios de *crear nueva* o *Cancelar*. Otro servicio que proveedor tiene a su disposición es *EnviaraContabilidad*; al ser activado este servicio, se realiza el envío del artefacto al destino indicado.

Las variables de ubicación del artefacto

En todo proceso de negocio es necesario representar quiénes son los actores fuente de un artefacto, además, de los actores que, de acuerdo a la especificación MFPT, son receptores de ese artefacto.

Para especificar en Oasis quién es el actor fuente de un artefacto, se utiliza una variable "de ubicación" de tipo booleano. En el caso de la creación de una factura por el actor *Proveedor*, la variable de ubicación correspondiente es *enProveedor*. El nombre de la variable está definido de la forma:

<idVariable> := 'en'<nomActor>

Cuando un artefacto es creado, se encuentra ubicado en el actor fuente. Por lo tanto, la variable *enProveedor* tiene inicialmente el valor *true*. Cuando el artefacto es enviado a otro actor, esta variable cambia su valor a *false*. Existe una variable de ubicación por cada actor que es receptor del artefacto <*nomAcRec*>. En el caso de los actores que no son fuente de este artefacto, su variable de ubicación se encuentra inicializada con un valor de *false.*

En Oasis, el resultado del envío de un artefacto está reflejado por un cambio de estado del objeto que representa al artefacto. Este estado está representado por los valores de las variables de ubicación, que están especificadas en la sección *variable attributes* y son utilizadas para indicar qué actor tiene posesión de un artefacto en algún momento del proceso. Por ejemplo, en la definición de la clase *Factura*, en la sección *variable attributes* se escriben las variables *enProveedor* y *enContabilidad*, indicando que la factura puede estar en algún momento determinado en manos de cualquiera de los actores Proveedor y Contabilidad:

```
variable attributes
        enProveedor: bool(true);
        enContabilidad: bool(false);
```

El valor *true* que tiene la variable *enProveedor*, significa que en el momento de la creación del artefacto, éste se encuentra en posesión de Proveedor. En algún momento del proceso, Proveedor solicitará al artefacto *Factura* la ejecución del servicio *enviaraContabilidad*. Esta acción deberá cambiar el estado del objeto *Factura*, modificando el valor de *enProveedor* a *false* y el valor de *enContabilidad* a *true*. Esto se indica en la sección *valuations* dentro de la definición de la clase *Factura*:

```
valuations
        [EnviaraContabilidad] enProveedor := false, enContabilidad := true;
```

Esta expresión indica que al ser ejecutada la acción *EnviaraContabilidad*, las variables mencionadas asumen el valor que se indica.

El uso de Precondiciones

Las precondiciones son condiciones que se deben cumplir para que una acción sea ejecutada. Esta característica de Oasis permite dar más claridad a la especificación. Para que un artefacto sea enviado de un actor a otro, debe estar en poder del actor que lo está enviando. Esta situación se puede modelar a través de la siguiente precondición:

preconditions
EnviaraContabilidad if {enProveedor = true}

De esta manera, la acción EnviaraContabilidad es ejecutada solamente si el artefacto está en posesión de Proveedor.

5. Las Reglas de Transformación

En la sección anterior se realizó un análisis de los elementos de Oasis requeridos para poder representar el modelo MFPT. Esta información proporciona la pauta para la construcción de las reglas de transformación[23], tarea que consiste en establecer una correspondencia de cada elemento del modelo MFPT, con un elemento o estructura de un esquema conceptual OASIS.

5.1 Definición de las Reglas de Transformación

La tabla 2 presenta el conjunto de reglas de transformación para convertir la especificación de un MFPT a una especificación de un esquema conceptual Oasis. El conjunto consta de 13 reglas, y toma como información de entrada los datos capturados en la plantilla de requisitos.

En la primera columna se muestra la referencia de cada regla con las secciones de la plantilla de requisitos cuyo nombre aparece en la segunda columna, en la tercera columna se muestra el número de regla y finalmente La cuarta columna indica el código equivalente en Oasis.

Ref. Plantilla	**Nombre de sección**	**Regla**	**Transformación a Oasis**
1.0	<nomProceso>	1	conceptual schema <nomProceso>
2.1	<nomArtefacto>	2	Class <nomArtefacto> identification constant attributes variable attributes events valuations preconditions end class
2.2 2.3.1 2.3.2	<nomMecId> <nomDato> <tipoDato>	3a	identification <nomMecId> ':' (<nomDato> {,<nomDato>}) ';'

2.3.1 2.3.2	{<nomDato> <tipoDato>}	3b	constant attributes <nomDato> ':' <tipoDato> ';' {<nomDato> ':' <tipoDato> ';'}
2.4	<accionNace>	4	events <accionNace> new ';'
2.5	<accionMuere>	5	events <accionMuere> destroy ';'
2.6	<nomAcFuente>	6	variable attributes en<nomAcFuente> ':' bool '(' true ')' ';'
2.7	<nomAcRec>	7a 7b 7c 7d 7e	variable attributes en<nomAcRec> ':' bool '(' false ')' ';' events Enviara<nomAcRec> ';' valuations '['Enviara<nomAcRec>']' en<nomAcFuente>'='false','en<NomAcRec>'=' true';' preconditions Enviara<nomAcRec> if '{' en<nomAcFuente> '=' true '}' ';' interface <nomAcFuente> with <nomArtefacto> ';' attributes '(' all ')' ';' services '(' Enviara<NomAcRec> ','<accionNace> ','<accionMuere> ')' ';' end interface
2.7 A partir del 2o. elemento de la lista de actores receptores, considerar al elemento actual como <nomAcRec> y al elemento anterior como <nomAcRecAnt>	{<nomAcRec>}	8a 8b 8c 8d 8e	variable attributes en<nomAcRec> ':' bool '(' false ');' events Enviara<nomAcRec> ';' valuations '[' Enviara<nomAcRec>']' en<nomAcRecAnt>'='false ','en<NomAcRec>'=' true';' preconditions Enviara<nomAcRec> if '{' en<nomAcRecAnt> '=' true '}' ';' interface <nomAcRecAnt> with <nomArtefacto> ';' attributes '(' en<NomAcRec> ',' en<nomAcRecAnt> ')' ';' services '(' Enviara<NomAcRec> ')' ';' end interface
3.1	<nomActor>	9	class <nomActor> identification constant attributes events end class
3.2 3.3.1 3.3.2 3.3.1	<nomMecId> <nomDato> <tipoDato> {<nomDato>	10a 10b	Identification <nomMecId> ':' (<nomDato> {,<nomDato>}) ';' constant attributes

3.3.2	<tipoDato>}		<nomDato> ':' <tipoDato> ';' {<nomDato> ':' <tipoDato> ';'}
3.4	<accionNace>	11	events <accionNace> new ';'
3.5	<accionMuere>	12	Events <accionMuere> destroy ';'
	<finFatos>	13	end conceptual schema

Tabla 2 *Regla para transformar un MFPT a Oasis*

A continuación se presenta una explicación de algunas de las reglas de transformación mostradas en la tabla 2, mostrándose con ejemplos del caso de estudio *Adquisición de Bienes*. La explicación detallada de cada regla se encuentra en [23].
El nombre del proceso en la Plantilla <*nomProceso*>, corresponde al nombre del esquema conceptual en Oasis. Ejemplo en tabla 3:

Línea	**Información de la Plantilla de Datos**	**Especificación en Oasis**
1	<nomProceso> = Servicio_al_cliente	Conceptual schema Servicio_al_cliente

Tabla 3 *Ejemplo de aplicación de regla*

Por cada nuevo artefacto en la plantilla, se crea una nueva clase en Oasis, con el nombre del artefacto <*nomArtefacto*> (ver sección 4.3.1). Es necesario aplicar las reglas 2 a la 8 para cada artefacto. Ejemplo en tabla 4:

Línea	**Información de la Plantilla de Datos**	**Especificación en Oasis**
20	<nomArtefacto> = Articulo	Class Articulo Identification constant attributes variable attributes events valuations preconditions end class

Tabla 4 *Ejemplo de aplicación de regla*

Definición del mecanismo de identificación de las instancias de la clase (ver sección 4.3.2), Ejemplo tabla 5:

Línea	**Información de la Plantilla de Datos**	**Especificación en Oasis**
21	<nomMecId>= Clave	Identification
22	<nomDato> = numero	Clave:(numero);

Tabla 5 *Ejemplo de aplicación de regla*

Definir los datos de identificación como atributos constantes. Ejemplo en tabla 6:

Línea	Información de la Plantilla de Datos	Especificación en Oasis
22	<nomDato> = numero	constant attributes
23	<tipoDato> = int	numero : int;

Tabla 6 *Ejemplo de aplicación de regla*

Para la lista de actores receptores se debe definir un atributo variable con el nombre del actor receptor (utilizando el nombre del primer actor receptor <nomAcRec>) y el prefijo "en" (ver sección 4.3.6). Ejemplo en tabla 7:

Línea	Información de la Plantilla de Datos	Especificación en Oasis
27	<nomAcRec> = Recepcion	Variable attributes En Recepcion : bool(false);

Tabla 7 *Ejemplo de aplicación de regla*

Utilizar el nombre del actor receptor para definir el nombre de un evento con el prefijo "*Enviara*" (ver sección 4.3.4). Ejemplo en tabla 8:

Línea	Información de la Plantilla de Datos	Especificación en Oasis
27	<nomAcRec> = Recepcion	Events Enviara Recepcion;

Tabla 8 *Ejemplo de aplicación de regla*

Definir una evaluación que asigne valores a las variables de ubicación que están relacionadas con el envío del artefacto (ver sección 4.3.6). Ejemplo en tabla 9:

Línea	Información de la Plantilla de Datos	Especificación en Oasis
27 26	<nomAcRec> = Recepcion <nomAcFuente> = Proveedor	Valuations [Enviara Recepcion] en Proveedor = false, enRecepcion = true;

Tabla 9 *Ejemplo de aplicación de regla*

Definir una precondición para el envío del artefacto (ver sección 4.3.7). Ejemplo en tabla 10:

Línea	Información de la Plantilla de Datos	Especificación en Oasis
27 26	<nomAcRec> = Recepcion <nomAcFuente> = Proveedor	Preconditions Enviara Recepcion if {en Proveedor = true};

Tabla 10 *Ejemplo de aplicación de regla*

Las reglas de transformación se aplican de esta forma para cada actor y producto de trabajo que exista en el modelo de procesos de negocio. El resultado final es un esquema conceptual en OASIS funcionalmente equivalente con la especificación de requisitos.

5.2 Un ejemplo de aplicación de las Reglas de Transformación

A continuación se presenta el modelo conceptual en lenguaje Oasis resultado de haber aplicado las reglas de transformación a la especificación mostrada en la tabla 1: la primera columna muestra el número de línea en la segunda y quinta columna aparece el número de la regla que se aplica en cada línea de código. La tercera y sexta columna muestran el código resultante del proceso de Transformación (tabla 11).

1	1	Conceptual schema Adquisicion_de_bienes
2	2	class Req_Compra
3	2	identification
4	3a	Clave:(numRequisicion);
5	2	constant attributes
6	3b	numRequisicion : int;
7	2	variable attributes
8	6	enCliente : bool (true);
9	7a	enCompras : bool (false);
10	2	Events
11	4	Elaborar new;
12	5	Cancelar destroy;
13	7b	EnviaraCompras;
14	2	Valuations
15	7c	[EnviaraCompras] enCliente:=false , enCompras:=true;
16	2	Preconditions
17	7d	EnviaraCompras if {enCliente = true };
18	2	end class
19	2	class Orden_Compra
20	2	identification
21	3a	clave:(numero, Fecha_emision);
22	2	constant attributes
23	3b	numero : int;
24	3b	Fecha_emision : date;
25	2	variable attributes
26	6	EnCompras : bool (true);
27	7a	enProveedor : bool (false);
28	2	Events
29	4	Elaborar new;
30	5	Cancelar destroy;
31	7b	EnviaraProveedor;
32	2	Valuations
33	7c	[EnviaraProveedor] enCompras:=false , enProveedor:=true;
34	2	preconditions

35	7d	EnviaraProveedor if {enCompras = true };
36	2	end class
37	2	class Artículo
38	2	identification
39	3a	clave:(numero);
40	2	constant attributes
41	3b	numero : int;
42	2	variable attributes
43	6	enProveedor : bool (true);
44	7a	enRecepción : bool (false);
45	8ª	enCliente : bool (false);
46	2	Events
47	4	dar_Alta new;
48	5	dar_Baja destroy;
49	7b	EnviaraRecepción;
50	8b	EnviaraCliente;
51	2	Valuations
52	7c	[EnviaraRecepción] enProveedor:=false , enRecepción:=true;
53	8c	[EnviaraCliente] enRecepción:=false , enCliente:=true;
54	2	preconditions
55	7d	EnviaraRecepción if {enProveedor = true };
56	8d	EnviaraCliente if {enRecepción = true };
57	2	end class
58	9	class Cliente
59	9	identification
60	10a	Clave:(numcliente);
61	9	constant attributes
62	10b	numcliente : int;
63	9	Events
64	11	dar_Alta new;
65	12	dar_Baja destroy;
66	9	end class
67	9	class Compras
68	9	identification
69	10a	nombre:(nombre);
70	9	constant attributes
71	10b	nombre : string;
72	9	Events
73	11	dar_Alta new;
74	12	dar_Baja destroy;
75	9	end class

76	9	class Proveedor
77	9	identification
78	10a	clave:(numProveedor);
79	9	constant attributes
80	10b	numProveedor : int;
81	9	Events
82	11	dar_Alta new;
83	12	dar_Baja destroy;
84	9	end class
85	9	class Recepción
86	9	identification
87	10a	nombre:(nombre);
88	9	constant attributes
89	10b	nombre : string;
90	9	end class
91	9	class Recepción
92	9	identification
93	10a	nombre:(nombre);
94	9	constant attributes
95	10b	nombre : string;
96	9	Events
97	11	dar_Alta new;
98	12	dar_Baja destroy;
99	9	end class
100	7e	Interface Cliente with Req_Compra
101	7e	attributes(all);
102	7e	services(EnviaraCompras, Elaborar, Cancelar);
103	7e	end interface
104	7e	Interface Compras with Orden_Compra
105	7e	attributes(all);
106	7e	services(EnviaraProveedor, Elaborar, Cancelar);
107	7e	end interface
108	7e	Interface Proveedor with Artículo
109	7e	attributes(all);
110	7e	services(EnviaraRecepción, dar_Alta, dar_Baja);
111	7e	end interface
112	8e	Interface Recepción with Artículo
113	8e	attributes(enCliente, enRecepción);
114	8e	services(EnviaraCliente);
115	8e	end interface
116	13	end conceptual schema

Tabla 11 *Esquema Conceptual en lenguaje OASIS*

El código resultante, que representa el esquema conceptual del MFPT, es la entrada para la herramienta CASE OO-Method, que permite complementar el esquema conceptual añadiendo propiedades del modelo dinámico, funcional y de ejecución, el resultado de este proceso es la generación automática de código en algún lenguaje imperativo como java, delphi o visual basic.

6. Conclusiones

En este artículo se presenta un ambiente integrado de desarrollo que permite la generación de aplicaciones ejecutables a partir de una plantilla de requisitos organizacionales, donde los requisitos son modelados en un sistema Workflow que hace énfasis en los flujos de trabajo que pasan de un actor a otro. Esta aproximación permite que el mayor esfuerzo y tiempo de desarrollo sean dedicados al modelado organizacional, y a los posibles cambios que en su reingeniería se requieran, dejando a la herramienta CASE la tarea de generación automática de aplicaciones.

El MFPT presentado es el punto de inicio para el modelado de requisitos tempranos con el framework i*, al hacer implícitas las relaciones entre actores organizacionales y detallar los productos de trabajo que dan origen a estas relaciones, lo cual es un conocimiento básico para lograr el objetivo del framework i*: describir los razonamientos que llevan a un actor a tomar una decisión. El MFPT es creado a partir de una plantilla de requisitos que permite describir y modificar un proceso de negocios sin detenernos en detalles de implementación. Posteriormente esta MFPT es traducido a una especificación de un esquema conceptual en lenguaje OASIS.

Para ligar los requisitos tempranos con los tardíos se propone el uso de la herramienta CASE OO-Method, que toma como entrada el esquema conceptual y mediante un método de refinamiento semi-automático permite completar dicho esquema con servicios, restricciones de integridad, etc. Con este esquema conceptual completo se realiza una traducción para producir una aplicación completa en alguno de los lenguajes objetivo de OO-Method: java, delphi o visual basic.

Los trabajos presentados en este artículo: plantilla de requisitos y MFPT forman parte del proyecto de investigación IRIS[29], en donde actualmente se están desarrollando otros trabajos de investigación para utilizar todo el potencial de los modelos del framework i* para modelar el negocio y el potencial de OO-Method para la generación de aplicaciones.

Referencias

Artículos en anales de congresos

[4] Sol Greenspan, John Mylopoulos, Alex Borgida, On Formal Requirements Modeling Languages: RML Revisited, Proceedings of the 16th International Conference on Software Engineering, 1994.

[7] R.J. Wieringa, A toolkit for requirements and design engineering: Tools definitions and directions for use.'Technical report, Faculty of Mathematics and Computer Science, Vrije Universiteit, Amsterdam, 1996. http://soling.cs.vu.nl/ ~tcm/

[8] O. Pastor, I. Ramos, "OASIS version 2: A class definition languaje to model information systems", Servicio de publicaciones de la Universidad Politécnica de Valencia, España, 1995.

[10] P. Du Bois, E. Dubois and J.M. Zeippen. On the Use of a Formal RE Language: The Generalized Railroad Crossing Problem. in Third Intl. Symposium on Requirements Engineering (RE'97), Annapolis (USA), January 1997.

[11] Janis A. Bubenko Esprit II (From Fuzzy to Formal) and Esprit III (Advanced Knowledge Based Environments for Large Database Systems) Projects

[12] Eric S. Yu and John Mylopoulos. An Actor Dependency Model of Organizational Work –With Application to Business Process Reengineering. (Proceedings of the Conference on Organizational Computing Systems COOCS'93.

[17] Pineda M. A., Ortiz H. J., Estrada E. H., Una Propuesta para la Obtención de Requisitos de Sistemas, Basada en Preceptos del Modelado Estratégico de Dependencias y la Teoría del Discurso. Congreso Internacional de Investigación en Ciencias Computacionales (CIICC'00), Cd. Madero. 2000.

[19] Pastor O., Insfrán E.,Pelechano V., OO-Method: an software production environment combining conventional and formal methods. In 9th Conference on Advanced Information System Engineering CAISE 99, Barcelona, España. 1999.

[28] Leite Do Prado J.C.S., Leonardi Ma. C. Business Rules as Organizational Policies, Proceedings of the 9th International Workshop on Software Specification & Design, Ise-Shima (Isobe), Japan

[29] Ortiz H. Javier, Estrada E. Hugo., M. Rebollar Alicia. Ingeniería de requisitos, Proyecto de investigación financiado por el CONACYT y desarrollado en el Centro Nacional de Investigación y Desarrollo Tecnológico CENIDET, Cuernavaca, Morelos, México. 1996-2001.

Artículos en revistas

[5] Sernadas, C. Sernadas, and J. F. Costa. Object specification logic. Journal of Logic and Computation, 1995.

[9] Feather Martin Language support for the specification and development of composite systems, ACM transactions on Programming Languages and System, Volume 9, 1987.

[16] Jeannette M. Wing. A Specifier's Introduction to Formal Methods. (IEEE Computer, 23(9), September 1990) pp. 8-24

Libros

[1] Richard H. Thayer and Merlin Dorfman. Software Software Requirements Engineering, Second edition, by Merlin Dorfman and Richerd H. Tahyer, 1997).

[2] Hossein S. Formal Methods in Information Systems Engineering (Software Requirements Engineering, Second edition, by Merlin Dorfman and Richerd H. Tahyer, 1997).

[3] Proyect 2RARE, 2 Real Applications for Requirements Engineering. Final Report on User' s Results.http://www.info.fundp.ac.be/~phe/2rare.htm

[6] G. Saake, T. Hartmann, R. Jungclaus, And H.-D. Ehrich. Object-Oriented Design of Information Systems: TROLL Language Features. *Advances in Database Systems, Implementations and Applications*. Paredaens and L. Tenenbaum, editors. pages 219--245. Springer Verlag, Wien, 1994

[13] Ivar Jacobson, Maria Ericsson and Agneta Jacobson. The Object Advantage: Business Process Reengineering with Object Technology. Addison- Wesley Publishing Company, 1995.

[14] E. Benjamín Franklin. Organización de Empresas análisis, diseño y estructura. Mc Graw Hill, México, 1998.

[15] A.Dardene, A. Van Lamsweerde, And S. Fickas. Goal directed requirements acquisition. Science of Computer Programming, April 1993.

[18] Letelier T. P., Animación Automática de Especificaciones OASIS utilizando Programación Lógica Concurrente (Memoria para optar al Grado de Doctor en Informática, Depto. de Sistemas Informáticos y Computación. Universidad Politécnica de Valencia)

[20] Rumbaugh J., Blaha M., Premerlani W., F. Eddy and W. Lorensen, Modelado y Diseño Orientado a Objetos, Metodología OMT (Prentice Hall International, España, 1996)

[21] Martin J. And Odell J., Object-Oriented Methods: considerations practice (Prentice Hall Hispanoamericana, 1997)

[22] Yu Erik, A Framework for Process Modelling and Reengineering, Ph. D. Thesis, Dept. of Computer Science, Univ. Of Toronto.

[23] Rios T. Octavio, Estrada E. H., Ortiz H., Especificación formal de Procesos de Negocios a partir del Modelo de Flujo de Productos de Trabajo, Tesis de Maestría. Centro nacional de Investigación y Desarrollo Tecnológico, Cuernavaca, 2001.

[24] Hollingswoth D., "Workflow Management Coalition: The Workflow Reference Model", Hampshire, Uk. 1995

[25] Nieto E, Ortiz J, Estrada H. Caracterización Formal Basada en el Lenguaje OASIS del Modelado de Dependencias Estratégico, Tesis de maestría Centro nacional de Investigación y Desarrollo Tecnológico, Cuernavaca, 2001.

[26] Grosz B.J. and Candence L. Sidner, Attention, Intention and the Structure of Discourse, en: Computational Linguistics. 1986.

[27] Robertson S. And Robertson J., Mastering the Requirements Process (Addison Wesley, 1999) .