ReCIBE, Año 4 No. 4, Diciembre 2015

Reducciones temporales para convertir la sintaxis abstracta del diagrama de flujo de tareas no estructurado al álgebra de tareas

Carlos Alberto Fernández-y-Fernández
Universidad Tecnológica de la Mixteca
caff@mixteco.utm.mx

José Angel Quintanar Morales
Universidad Tecnológica de la Mixteca
joseangel@mixteco.utm.mx

Resumen: Este artículo describe nuestro trabajo en el modelado de software usando reducciones temporales para representar diagramas de flujo no estructurado, como una representación intermedia para construir una expresión textual en una álgebra de procesos particular. Este trabajo fue realizado para poder construir una herramienta CASE de apoyo para la fase del modelado de tareas en el Método Discovery para el desarrollo de software. Inicialmente explicaremos las similitudes entre dos tipos de diagramas, el diagrama de actividades de UML y el diagrama de flujo de tareas con su representación formal (el álgebra de tareas). Posteriormente, ofreceremos una explicación explicando la generación automática, usando las reducciones temporales, de expresiones en el álgebra de tareas usando información abstracta que es obtenida de los diagramas de flujo de tareas.

Palabras Clave: **Reducciones temporales, modelado visual, diagramas de actividad.**

Temporary reductions for converting the abstract syntax from an unstructured task flow diagram to the task algebra

Abstract: **The present paper describe our work modeling software using temporary reductions to represent unstructured flow diagrams as an intermediate representation to build textual expression in a particular process algebra. This work was realized in order to build a CASE tool supporting the task modeling phase from the Discovery Method for software development. We begin explaining the similarities between two types of flow diagrams, the UML activity diagram and the task flow diagram with its formal representation (task algebra). Next, we offer an explanation of the work to automatically generate, using the temporary reductions, expressions in the task algebra using abstract information from the task flow diagrams.**

Keywords: **Temporary reductions, visual modelling, activity diagrams.**

1. Introducción

La programación para el modelado de sistemas se refiere al desarrollo de software donde las notaciones gráficas y los componentes visuales manipulables interactivamente son usados principalmente para definir el comportamiento de un sistema (Budinsky et al., 2003).

Para que el modelo de un sistema logre un grado de compresión mayor, se busca semejar uno de los objetivos de la programación visual, el cual es facilitar la comprensión de los programas y simplificar la programación en sí. Más allá, la programación visual debe permitir a los usuarios finales el construir sus propios programas de una manera más sencilla y comprensible (Hernández Valdemar, & León, 2012.). Al cumplir con esta expectativa, las

personas sin conocimientos técnicos que estén involucradas en el proyecto, podrán comprender el comportamiento del sistema modelado (Chonoles, 2003).

Si bien, los lenguajes visuales en su mayoría son creados para que los usuarios poco experimentados en algún tipo de lenguaje de programación puedan llevar a cabo la construcción de sus propios programas, los desarrolladores profesionales también utilizan este tipo de lenguajes (Fernández-y-Fernández et al., 2011). Este enfoque se aplica a la verificación del software. Para iniciar este proceso, primero es indispensable contar con un lenguaje de modelado. Como principal ejemplo tenemos a UML.

UML es utilizado como una notación estandar de modelado visual, ya que posee 14 diferentes diagramas que pueden ser utilizados para representar un sistema de software detallando diversos aspectos, así como diversas perspectivas.

Desde el año de 1997 y bajo la supervisión del OMG, ha evolucionado, sin embargo su notación se ha vuelto más compleja, con la finalidad de abarcar todo el proceso completo para el modelado de sistemas, así como de procesos (Graham, & Simons,1999). Esto ha originado algunas críticas respecto a su composición semántica y las diferentes formas de interpretar los modelos.

El problema principal, es la ambigüedad con la que se pueden crear los diversos diagramas existentes en el lenguaje, esto provoca que se carezca de un sentido específico y la validez del mismo no puede establecerse con la claridad necesaria (Graham, & Simons,1999), (Schmuller, 2009).

Existen otras alternativas a UML, con la cual se pueden modelar sistemas de una forma similar a la que presenta el diagrama de actividades de UML, por ejemplo la notación del diagrama de flujo de tareas del Método Discovery. Este diagrama de flujo de tareas y su respectiva notación, puede tomarse como un subconjunto del diagrama de actividades definido en UML (Simons, 1998). Fue

probado en diversos proyectos por estudiantes de posgrado de la Universidad de Sheffield. La notación simple y sin ambigüedades lo hace una opción viable y consistente en comparación con UML.

1.1 El diagrama de actividades

El diagrama de actividades muestra un flujo de acciones, generalmente secuenciales, el propósito de un diagrama de actividades es: Modelar el flujo de las tareas. Modelar las operaciones. Todos los diagramas de actividades tienen una serie de elementos básicos, su comportamiento se describe mediante la integración de los elementos siguientes:

- Particiones.
- Nodos de acción.
- Nodos de control.
- Nodos de Objeto.
- Control de flujo.

Es a partir de estos objetos que se pueden modelar sistemas completos, sin embargo, como se mencionó en la sección anterior, presenta ciertas ambigüedades. Una posible mejora estaría dada por una notación orientada a objetos más sencilla y pequeña, fácil de aprender, que sea más exacta y replicable, dicha notación debe estar soportada por un lenguaje formal para representar la semántica de una manera precisa, de manera que los modelos se puedan verificar unos con otros (Simons, 2006).

Una posible solución para tal notación es restringir un perfil UML, denotado por el Método Discovery y definido en el diagrama de flujo de tareas, el cual pone especial atención al minimalismo y la coherencia.

1.2 El diagrama de flujo de tareas

Durante el proceso de modelación del sistema, se necesitará realizar la identificación de tareas y la estructuración de relaciones para las diferentes perspectivas de los actores en el modelo, se deberá representar el flujo de las relaciones de trabajo. El Método Discovery representa este flujo de trabajo por medio del Diagrama de flujo de tareas.

Básicamente un diagrama de flujo de tareas representa el orden en que las tareas se llevan a cabo en la organización, también denota la relación lógica entre todas las tareas que componen el diagrama.

La notación utilizada en el Método Discovery se basa en el diagrama de actividades de UML, sin embargo, se mantiene constante la figura de una elipse para la representación de una tarea. Con la inclusión del componente final de flujo en UML 2.0 (círculo cruzado por dos líneas inclinadas), se cubre la funcionalidad de Failured (círculo cruzado por una sola línea inclinada) logrando una compatibilidad mayor, con un ligero cambio en la representación visual. En la tabla 1 se muestra la notación para un diagrama de flujo de tareas, así como el nombre y un identificador numérico asociado al tipo de componente (Fernández-y-Fernández, 2010).

Componente visual	Nombre	Identificador numérico
	*Start*	0
	*Task*	1
	*Fork*	2
	*Join*	3
	*Exception*	4
	*Failure*	5
	*Choice*	6
	*End*	7

Tabla 1.

Componentes visuales para el diagrama de flujo de tareas.

1.3 Del diagrama de flujo de tareas al álgebra de tareas

Una vez que los elementos gráficos han sido presentados, se detallará como es que estos componentes gráficos poseen una representación semántica la cual es representada por medio del álgebra de tareas, con ello se tiene la representación semántica de la que carece UML.

La representación formal del diagrama de flujo de tareas está dada por el álgebra de tareas, ésta álgebra es una traducción o representación abstracta de dicho diagrama, la sintaxis abstracta del álgebra de tareas se muestra en la Figura 1.

1.4 Naturaleza no estructurada del diagrama de flujo de tareas

La meta del diseño de los diagramas de flujos de tareas, es aprovechar el comportamiento de los diagramas de actividad de UML, con la intención de brindar una facilidad a los usuarios y tener una mejor aceptación y difusión.

Esta decisión tiene puntos a favor y en contra, se puede tomar como un ejemplo práctico el lenguaje HTML. HTML es muy flexible, y los exploradores HTML aceptan cualquier cosa que se parezca a HTML, esta característica ayudó a la temprana adopción de HTML, pero ahora es un problema la solución para esta problemática en particular fue la aparición de lenguajes más estrictos (Benot, 2001).

En resumen, implementar la automatización de la traducción de un diagrama no estructurado, significa no poder utilizar métodos o técnicas que si podrían utilizarse a diagramas estructurados, por tal motivo se debe emplear otro tipo de técnicas.

```
Activity ::= ε                    -- empty activity
         | σ                      -- succeed
         | ϕ                      -- fail
         | Task                   -- a single task
         | Activity ; Activity    -- a sequence of activity
         | Activity + Activity    -- a selection of activity
         | Activity || Activity   -- parallel activity
         | μx.(Activity ; ε + x)  -- until-loop activity
         | μx.(ε + Activity ; x)  -- while-loop activity

Task::= Simple                    -- a simple task
         | { Activity }           -- encapsulated activity
```

Figura 1.

Sintaxis abstracta del álgebra de tareas [10].

2. Definición del problema

Como se mencionó en la sección 1.4, se debe mantener la simplicidad en cuanto al diseño de los diagramas de flujos de tareas, sin embargo, se presenta una problemática dada esta misma simplicidad, la causa es que a partir del elemento Choice, mostrado en la tabla 1, se pueden construir tres estructuras tal como lo demuestra la sintaxis abstracta en la Figura 2, dichas estructuras son:

- Selección binaria figura 2.a.
- Ciclo Until figura 2.b.
- Ciclo While figura 2.c.

De tal manera que la generación de la respectiva álgebra no pudo realizarse en una pasada, ya que para llevar a cabo el proceso completo destinado a obtener el álgebra de tareas es necesario realizar varias pasadas. Esta forma de generar el álgebra de tareas se debe a que se necesita analizar una ramificación completa de un diagrama de flujos de tareas y así detectar que estructura de la sintaxis del álgebra de tareas representa. Cabe mencionar que los diagramas son diseñados con la herramienta presentada en (Fernández-Santos, H. et al., 2011).

3. Reducciones temporales

Por los motivos anteriores, se optó por la vía de reducción a expresiones temporales, una expresión temporal la podemos definir como, una estructura compuesta por componentes básicos de la sintaxis del álgebra de tareas, que a su vez tiene un representación dentro de la misma sintaxis abstracta.

La reducción a expresiones temporales retoma el concepto de la optimización local de la fase de generación y optimización de código del desarrollo de compiladores (Aho, A. V. et al., 1990), dicho proceso consiste en examinar

grupos de instrucciones para sustituirlas por instrucciones más baratas. Las optimizaciones pueden dar lugar a otras optimizaciones de manera que es necesario repetir varias veces la optimización local para obtener la mejora del código (Teufel, B. et al., 1995). Por su parte la reducción de expresiones temporales trata de obtener un diagrama elemental formado por un inicio y un componente terminal.

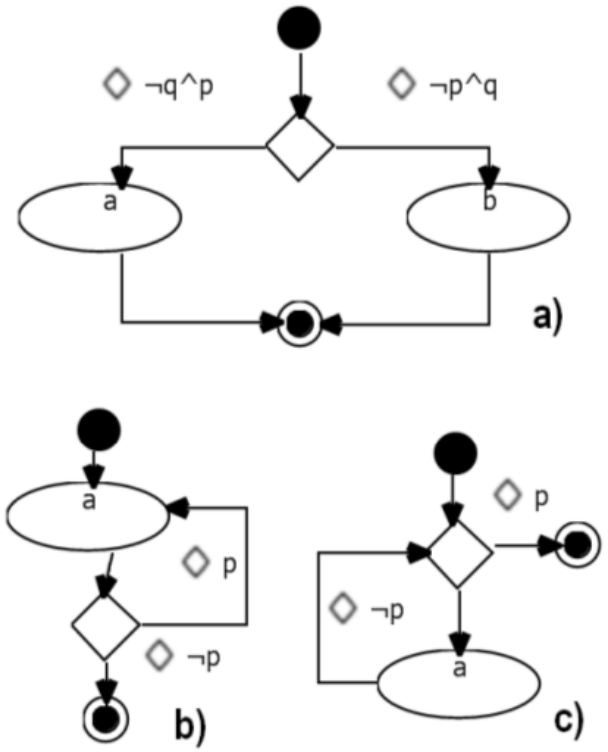


Figura 2.

Estructuras construidas con el componente Choice.

3.1 Reducción de expresiones temporales

Para llevar a cabo la reducción del diagrama, se tiene la necesidad de abstraer el modelo visual y almacenarlo en las estructuras de la figura 3 y figura 4, y realizar una cantidad indeterminada de pasadas para formar estructuras completas con los componentes hasta tener un diagrama elemental.

A continuación se describen los datos que almacenan cada uno de los campos de la estructura 3.

- NOMBRE: el nombre del componente gráfico con un valor de texto.
- TIPO: el tipo de identificador conforme la tabla 1.
- ID: identificador único asignado automáticamente.
- OP1: campo destinado a información introducida por el usuario, en este caso para los componentes Task, Choice y Exception.
- OP2: campo destinado a información introducida por el usuario, en este caso para los componentes Choice, Exception.
- ENTRA T: número total de entradas del componente.
- SAL T: número total de salidas del componente.
- MARCA: campo auxiliar libre y sin representación de información.

Ahora se detallará la información que contiene la estructura dinámica 4 que representa los flujos del sistema.

- TIPO: el tipo de identificador del componente origen conforme la tabla 1.
- ORIGEN: id único del componente origen.
- TIPO: el tipo de identificador del componente destino conforme la tabla 1.
- DESTINO: id único del componente destino.
- RAMA: campo para validación de ramas.
- MARCA: campo auxiliar libre y sin representación de información.
- ENT_T: número total de entradas del componente.
- SALT_: campo auxiliar libre y sin representación de información.

3.1.1 Reducción de tareas lineales

Esta reducción se enfoca a la detección y posterior unión de tareas lineales. Una tarea lineal se define como la sucesión de componentes Task, la fusión de un par de tareas lineales da como origen a una tarea compuesta. Para llevar a cabo este proceso, primero se deben determinar el número de entradas y

salidas de los componentes Task, si para ambos es uno, entonces se procede a la fusión.

El procedimiento para fusionar las tareas se puede explicar suponiendo la existencia de dos tareas, A con destino B, y consiste en:

1. Obtener el id destino de la tarea B.
2. Obtener la información de la localidad marca de B.
3. Sustituir el destino de la tarea A por el de B.
4. Fusionar el valor de la localidad marca de A con el de B.
5. Eliminar a B de la estructura flujos.

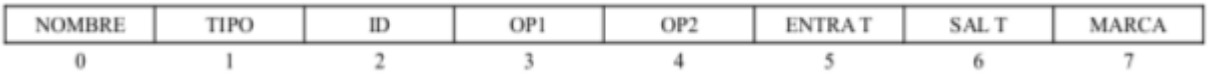


Figura 3.

Estructuras para abstraer información de los componentes visuales.

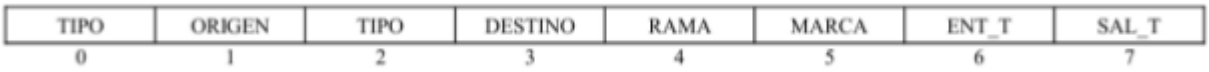


Figura 4.

Sintaxis abstracta del álgebra de tareas [10].

3.1.2 Reducción de tareas compuestas

El proceso para reducir tareas compuestas lleva a cabo un proceso similar al método para tareas lineales, excepto que éste opera sobre tareas compuestas. Una tarea compuesta se define como la sucesión de componentes reducidos, la fusión de un par de tareas compuestas da como origen a otra tarea compuesta. Para llevar a cabo este proceso, primero se deben determinar el número de entradas y salidas de las tareas compuestas, si para ambos es uno, entonces se procede a la fusión.

El procedimiento para fusionar las tareas compuestas se puede explicar suponiendo la existencia de dos elementos, A con destino B, y consiste en:

1. Obtener el id destino del elemento B. O
2. Obtener la información de la localidad marca de B.
3. Modificar el tipo de componente en las estructuras dinámicas flujos y componentes.
4. Sustituir el destino de la tarea A por el de B.
5. Fusionar el valor de la localidad marca de A con el de B.
6. Eliminar a B de la estructura flujos.

La eliminación del componente B es relativo, ya que ahora quedará contenido en A.

3.1.3 Reducción de tarea lineal con terminal

La reducción consiste en generar un elemento terminal etiquetado como final. Definimos al elemento terminal como la estructura compuesta por una secuencia de tareas lineales o tareas compuestas seguido de un componente End, o un componente Failure o un elemento terminal. Para obtenerlo, es necesario identificar una secuencia de tarea lineal o compuesta que apunta a un elemento terminal, una vez hecho esto, se lleva a cabo el siguiente procedimiento.

1. Identificar el tipo de componente C.
2. Si no se tratará de un componente End o Failure, se obtiene el valor de la localidad marca de la estructura flujos.
3. Se obtiene el valor de la localidad marca de B.
4. Se sustituye destino de A por el valor de C.
5. Se fusiona la localidad marca de A con la B y con C, sólo si éste no fuera un componente End o Failure.
6. Se marca a B como elemento terminal.
7. Se elimina a C.

3.1.4 Reducción del ciclo For

Basa su búsqueda únicamente en los componentes Choice, en cada recorrido donde se encuentre este elemento, será analizada la posible formación de una estructura For. Cuando se detecta una estructura de este tipo, y suponiendo que se tiene la estructura de la figura 5 donde el elemento Choice apunta hacia A para formar la estructura For y hacia B cuando la condición no se cumple, tenemos que:

1. Se identifica el flujo que da salida al For, es decir el flujo que seguiría cuando la condición de iteración ya no se cumpla, en este caso B.
2. Se obtiene el valor de la localidad marca del componente al que apunta el Choice cuando la condición se cumple, componente A.
3. A la localidad marca del Choice se le asigna la información obtenida en el paso 2 bajo el formato Mu(Epsilon+datos;x).
4. Se elimina el componente A.

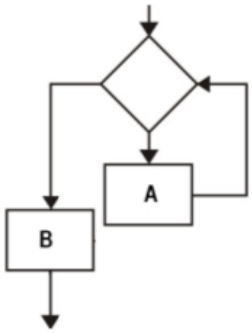


Figura 5.

Estructura For identificada para ser reducida.

3.1.5 Reducción del ciclo Until

De la misma forma en que el proceso para reducir una estructura For, se analizan los componentes Choice. se busca la posible formación de estructuras Until, con la diferencia de que el componente interno debe ser a su vez origen y destino, además el componente interno debe poseer dos flujos de entrada y uno de salida.

El procedimiento para reducir una estructura Until es el siguiente, suponiendo se tiene la imagen de la Figura 6, donde el elemento A apunta un Choice, éste último apunta al elemento A y finalmente cuando la condición no se cumple, el elemento Choice apunta al elemento B, tenemos que:

1. Se identifica el destino del componente Choice cuando la condición no se cumple, es decir cuando el flujo apunta a B.
2. Se sustituye el destino de A, por el destino obtenido anteriormente, es decir, el flujo de A tendrá como destino a B.
3. El campo marca del componente A se modifica para tener el formato Mu(datos;Epsilon+x).
4. El componente Choice es eliminado.

3.1.6 Reducción del paralelismo

La identificación de la estructura Fork-Join resulta hasta cierto punto sencilla. Para su identificación se necesita que por cada componente Fork, exista una tarea lineal o compuesta seguido de un componente Join, si hubiese una tarea terminal, no es necesario que esta apunte a un componente Join. Para su reducción se lleva a cabo el siguiente procedimiento.

Suponiendo que se tiene la estructura de la Figura 7, donde el componente Fork apunta a las tareas compuestas A1, A2 ,..., An donde todas las tareas desde A1 hasta An apuntan a un único componente Join y éste último apunta a un componente B, tenemos qué:

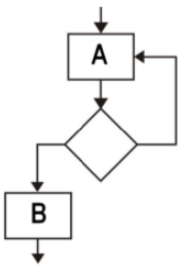


Figura 6.

Estructura Until identificada para ser reducida.

1. El contenido de la localidad marca del componente Fork se fusiona con el elemento A1 mediante el formato A1|| A2|| ... An.
2. Se elimina el componente A1.
3. Esto se repite hasta An.
4. Se sustituye el destino del componente Fork por el destino del componente Join, en este caso B.
5. Se elimina el componente Join.

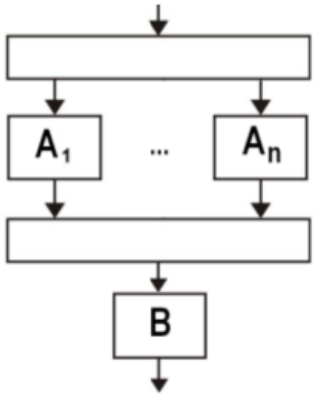


Figura 7.

Estructura Fork/Join identificada para ser reducida.

3.2 Reducción de la estructura Or

La reducción a la estructura Or es la de mayor complejidad, ya que existen 6 diferentes formas de reducir una estructura Or.

1. Or simple.
2. Or terminal doble.
3. Or terminal simple.
4. Or terminal simple vacio.
5. Or vacio simple.
6. Or vacio doble

A continuación se detalla el proceso de reducción para cada una de ellas.

3.2.1 Reducción de un Or simple

Esta reducción es considerada como el mejor de los casos, ya que este tipo de estructura posee contenido en ambos destinos del componente Choice, por lo tanto no es necesario llevar a cabo ninguna evaluación. Para llevar a cabo este proceso suponemos que se tiene la estructura de la Figura 8:

Los destinos del componente Choice se dirigen a un elemento A y B respectivamente, y cada uno de estos componentes tiene como destino común al elemento C, luego.

1. Se identifica el destino de A ó B, en este caso el elemento C.
2. Se obtiene el valor de la localidad marca de A y de B.
3. Se crea el contenido de la localidad marca del componente Choice bajo el formato (A+B).
4. Se sustituye en el componente Choice el destino B por C.
5. Se elimina el destino A y B.
6. Se modifica el tipo de componente de Choice a Or.

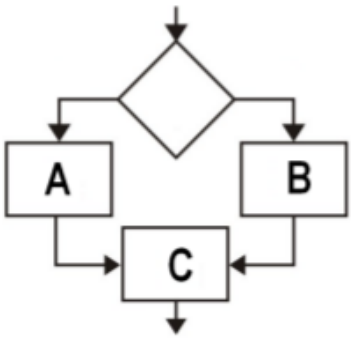


Figura 8.

Estructura Or simple identificada para ser reducida.

3.2.2 Reducción de un Or terminal doble

Este tipo de reducción es un caso especial de Or, ya que se convertirá en un elemento terminal, esto, debido a que ambos destinos del componente Choice tienen como destino alguna combinación de los elementos A y B del tipo End, Failure o un componente terminal. El proceso de reducción de la estructura de la Figura 9 se detalla a continuación.

1. Se identifica el tipo de componente destino del elemento Choice, si es un componente terminal se almacena el valor de la localidad marca, en caso contrario se almacena el tipo.
2. Se identifica el tipo de componente del siguiente destino del elemento Choice, si es un componente terminal se almacena el valor de la localidad marca, en caso contrario se almacena el tipo.
3. Si uno o ambos tipos de destino corresponden a un componente terminal esta información se almacena en la localidad marca del elemento Choice, bajo el formato(A+B).
4. Si ambos componentes son elementos del tipo End o Failure o alguna combinación de ellos, se almacena el tipo del componente en la localidad marca del elemento Choice bajo el formato (A+B).
5. Se modifica el tipo de componente de Choice a Terminal.
6. Se eliminan los componentes A y B.

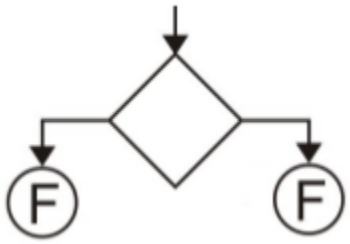


Figura 9.

Sintaxis abstracta del álgebra de tareas (Fernández-y-Fernández, 2010).

3.2.3 Reducción de un Or terminal simple

Este tipo de reducción es un caso especial del Or, ya que como se observa en la Figura 10 uno de los destinos del componente Choice es un componente terminal y el otro destino, por el contrario es un elemento no terminal el cual pertenece a la estructura Or, es decir el único flujo que apunta a dicho componente no terminal, es el flujo destino del elemento Choice. Para el proceso de reducción, definimos un componente Choice que tiene como uno de sus destinos a un elemento terminal F, el otro destino se dirige a un elemento no terminal A, el cual a su vez tiene como destino un componente B. Entonces:

1. Se identifica el componente terminal F, si es un elemento End o Failure, se guarda su tipo de componente, en caso contrario se obtiene el valor de la localidad marca de la estructura dinámica Flujos.
2. Se obtiene el valor de la localidad marca del componente no terminal A.
3. Se obtiene el destino del componente A, en este caso el elemento B.
4. Se fusiona el valor obtenido de los componentes F y A bajo el formato (F+A).
5. Se sustituye el destino A del componente Choice por B.
6. Se elimina el componente F.
7. Se elimina el componente A.
8. Se modifica el tipo de componente Choice por Or.

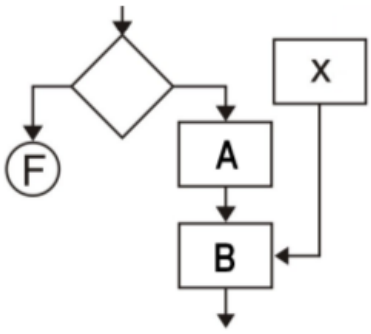


Figura 10.

Estructura Or terminal simple identificada para ser reducida.

3.2.4 Reducción de un Or terminal simple vacio

Este tipo de reducción es similar a la llevada a cabo en Or terminal simple, la variación que presenta, es que ningún componente no terminal pertenece a la estructura Or, tal como se muestra en la Figura 11. Para el proceso de reducción, definimos un componente Choice que tiene como uno de sus destinos a un elemento terminal F, el otro destino se dirige a un elemento no terminal A el cual no pertenece a la estructura Or, es decir el elemento A también es destino de otro componente. Por lo tanto tenemos que:

1. Se identifica el componente terminal F, si es un elemento End o Failure, se guarda su tipo de componente, en caso contrario se obtiene el valor de la localidad marca de la estructura dinámica Flujos.
2. Se fusiona el valor obtenido del componentes F y el simbolo Epsilon bajo el formato (F+Epsilon).
3. Se elimina el componente F.
4. Se modifica el tipo de componente Choice por Or.

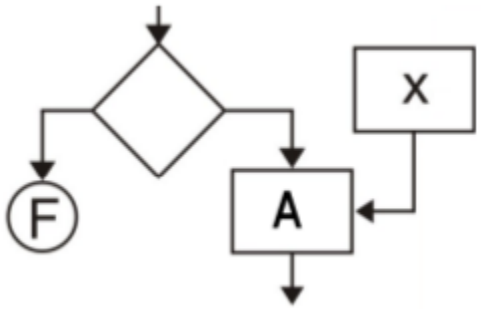


Figura 11.

Estructura Or terminal simple vacía identificada para ser reducida.

3.2.5 Reducción de un Or terminal vacio simple

La reducción de este componente es un tanto simple, similar al proceso llevado a cabo en orTerminalSimple, con la excepción de que el componente terminal no existe, ese destino se dirige a un componente que está fuera de la estructura Or, tal como se muestra en la Figura 12. Este proceso de reducción se inicia suponiendo que el componente Choice tiene como uno de sus destinos al elemento A, y como segundo destino al elemento B, el cual también es destino del elemento A. Dadas estas suposiciones tenemos que:

1. Se obtiene el valor de la localidad marca del componente A en la estructura dinámica flujos.
2. Se sustituye el valor de la localidad marca del componente Choice por el contenido de A bajo el formato (A+Epsilon).
3. Se elimina el componente A.
4. El tipo de componente del Choice es modificado a Or.

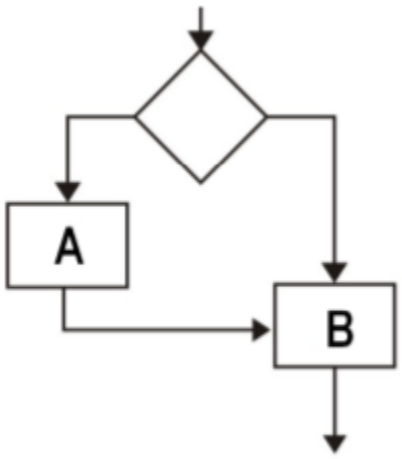


Figura 12.

Estructura Or vacía simple identificada para ser reducida.

3.2.6 Reducción de un Or vacio doble

Esta reducción es la más simple para la estructura Or y corresponde a la mostrada en la Figura 13. Para llevarla a cabo se identifica la estructura de la siguiente forma. El componente Choice tiene ambos destinos dirigidos a un único componente A, por lo tanto se realiza el siguiente proceso.

1. Se sustituye el valor de la localidad marca del componente Choice por la cadena (Epsilon+Epsilon).
2. Se elimina un flujo que apunta hacia A. El tipo de componente Choice se modifica por un Or.

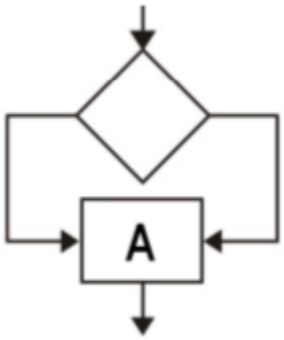


Figura 13.

Estructura Or vacía doble identificada para ser reducida.

4. Resultados

Como resultado final se obtiene el álgebra de tareas asociado a un diagrama de flujo de tareas, sin embargo, para comprobar la salida, es necesario utilizar el compilador del álgebra de tareas, desarrollado en (Fernández-y-Fernández, 2010). Dicho compilador transforma la expresión del álgebra tareas y, si la expresión es correcta, genera un conjunto de trazas, una sola traza es una cadena simbólica que denota una ruta de posible ejecución a través del sistema y el conjunto de trazas denota todas las rutas de ejecución posibles (Fernández-y-Fernández, 2010).

4.1 Estructuras de reducción

Como se ha explicado, el plugin genera el álgebra de tareas al llevar a cabo una reducción de componentes a estructuras temporales; cada una de estas estructuras se pusieron a prueba; para ello se creó el diagrama correspondiente y se obtuvo el álgebra asociada al diagrama. El plug-in genera el álgebra de tareas asociado a un diagrama de flujo de tareas, sin embargo, para comprobar la salida, es necesario utilizar el compilador del álgebra de tareas. Dicho compilador transforma la expresión del álgebra tareas y, si la expresión es correcta, genera las trazas correspondientes para la expresión. El resultado es un sistema de tareas dado como un conjunto de trazas, en el que una sola traza es una cadena simbólica que denota una ruta de posible de ejecución a través del sistema y el conjunto de trazas denota todas las rutas de ejecución posibles (Fernández-y-Fernández, 2010). A continuación se presentan las estructuras temporales implementadas en el plug-in.

4.1.1 Estructura lineal

La estructura lineal de tareas (mostrada en la Figura 14) es la composición básica para reducir diagramas, su correspondiente álgebra de tareas se muestra en la Tabla 2.

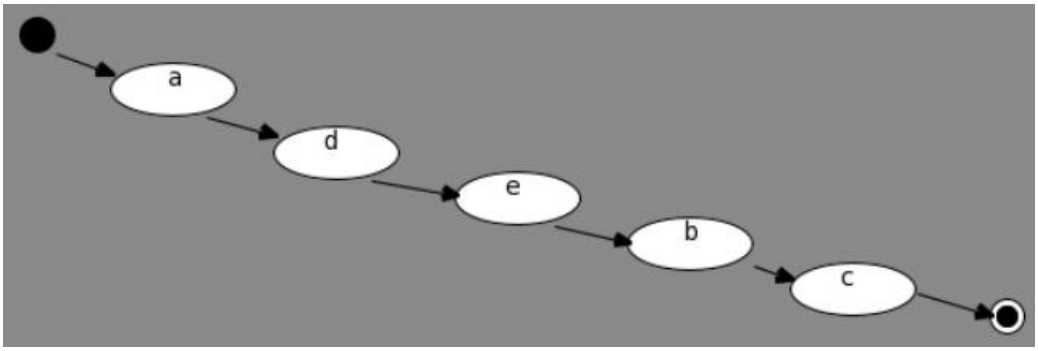


Figura 14.

Diagrama de flujos de tareas mostrando una estructura lineal.

{a;d;e;b;c;Sigma}
fromList [[a,d,e,b,c]]

Tabla 2.

Álgebra de tareas para una estructura lineal y las trazas derivadas del álgebra.

4.1.2 Estructura Fork

Una forma compleja de la estructura Fork se muestra en la Figura 15, ya que contiene un elemento terminal el cual no posee un flujo que vaya hacia el componente Join. El álgebra de tareas de este diagrama y las trazas originadas derivadas del álgebra de tareas se presentan en Tabla 3.

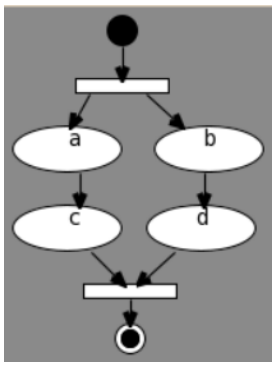


Figura 15.

Diagrama de flujos de tareas mostrando una estructura Fork/Join.

{((a;c)||(b;d));Sigma}
fromList [[a,b,c,d],[a,b,d,c],[a,c,b,d],[b,a,c,d],[b,a,d,c],[b,d,a,c]]

Tabla 3.

Álgebra de tareas para una estructura Fork/Join y las trazas derivadas del álgebra.

4.1.3 Estructura Or

La estructura or presenta una serie de variantes, debido a que puede contener componentes terminales o vacíos, la Tabla 4 muestra las posibles formas de construir la estructura or, así como su correspondiente álgebra de tareas.

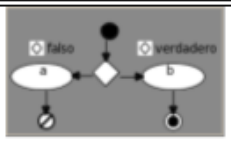
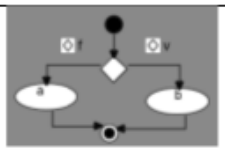
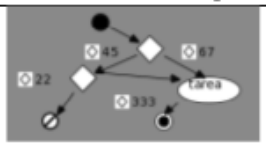
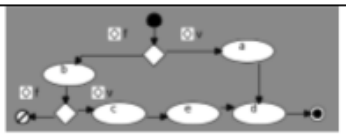
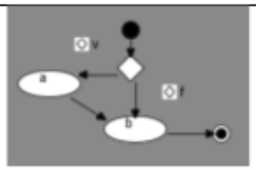
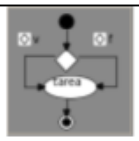
Diagrama de flujo de tareas	Álgebra de tareas	Trazas
 Estructura Or terminal doble.	{(a;Phi)+(b;Sigma)}	fromList [[!,a,Phi],[!,b]]
 Estructura Or simple.	{((b)+(a));Sigma}	fromList [[!,a],[!,b]]
 Estructura Or terminal simple vacio.	{(((Phi+Epsilon))+Epsilon);tarea;Sigma}	fromList [[!,tarea],[!,Phi]]
 Estructura Or terminal simple.	{((a)+(b;(Phi+(c;e))));d;Sigma}	fromList [[!,a,d],[!,b,!,c,e,d],[!,b,!,Phi]]
 Estructura Or vacio simple.	{((a)+Epsilon);b;Sigma}	[[!,a,b],[!,b]]
 Estructura Or vacio doble.	{(Epsilon+Epsilon);tarea;Sigma}	fromList [[tarea]]

Tabla 4.

Posibles formas de construir la estructura Or.

4.1.4 Estructura Until

La forma básica de la estructura Until se presenta en la Figura 16, de igual forma en Tabla 5 se muestra el álgebra de tareas asociado a dicha estructura y sus correspondientes trazas originadas por el álgebra de tareas.

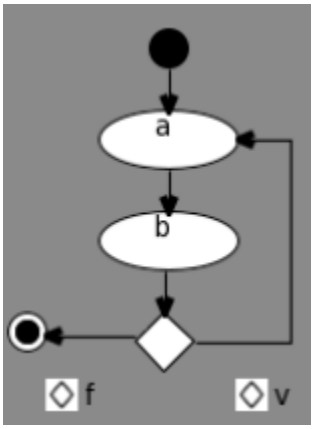


Figura 16.

Diagrama de flujos de tareas mostrando una estructura Until.

{Mu.x((a;b);Epsilon+x);Sigma} fromList [[a,b,!],[a,b,!,a,b]]

Tabla 5.

Álgebra de tareas para una estructura Until y las trazas derivadas del álgebra.

4.1.5 Estructura For

La Figura 17, muestra un diagrama de flujo de tareas con la estructura For. El álgebra de tareas de este diagrama y las trazas originadas derivadas del álgebra de tareas se presentan en la Tabla 6.

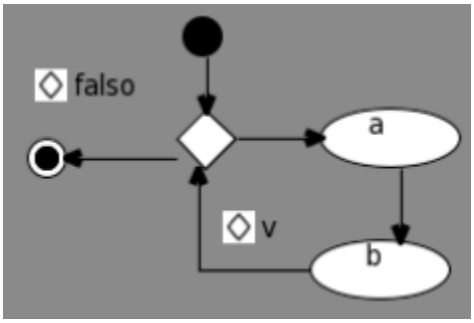


Figura 17.

Diagrama de flujos de tareas mostrando una estructura For.

{Mu.x(Epsilon+(a;b);x);Sigma} fromList [[!],[!,a,b,!],[!,a,b,!,a,b]]

Tabla 6.

Álgebra de tareas para una estructura For y las trazas derivadas del álgebra.

4.2 Generación con un diagrama de flujos completo

Para ejemplificar la correcta generación del álgebra de tareas mediante la reducción de expresiones temporales, se utiliza parte del trabajo propuesto en (Fernández-y-Fernández, C. A. et al., 2012), se analiza un diagrama de flujo de tareas obtenido a partir del análisis de un caso de uso.

El diagrama de la Figura 14 se compone de cinco tareas, una de ellas conforma un ciclo de repetición Until, dos más se realizan de forma paralela, las tareas son las siguientes.

1. Proyectar la palma de la mano sobre el componente deseado.
2. El componente es sombreado y anclado al cursor.
3. Buscar posición del componente.
4. Emitir alerta visual.
5. Asignar el ID del componente.

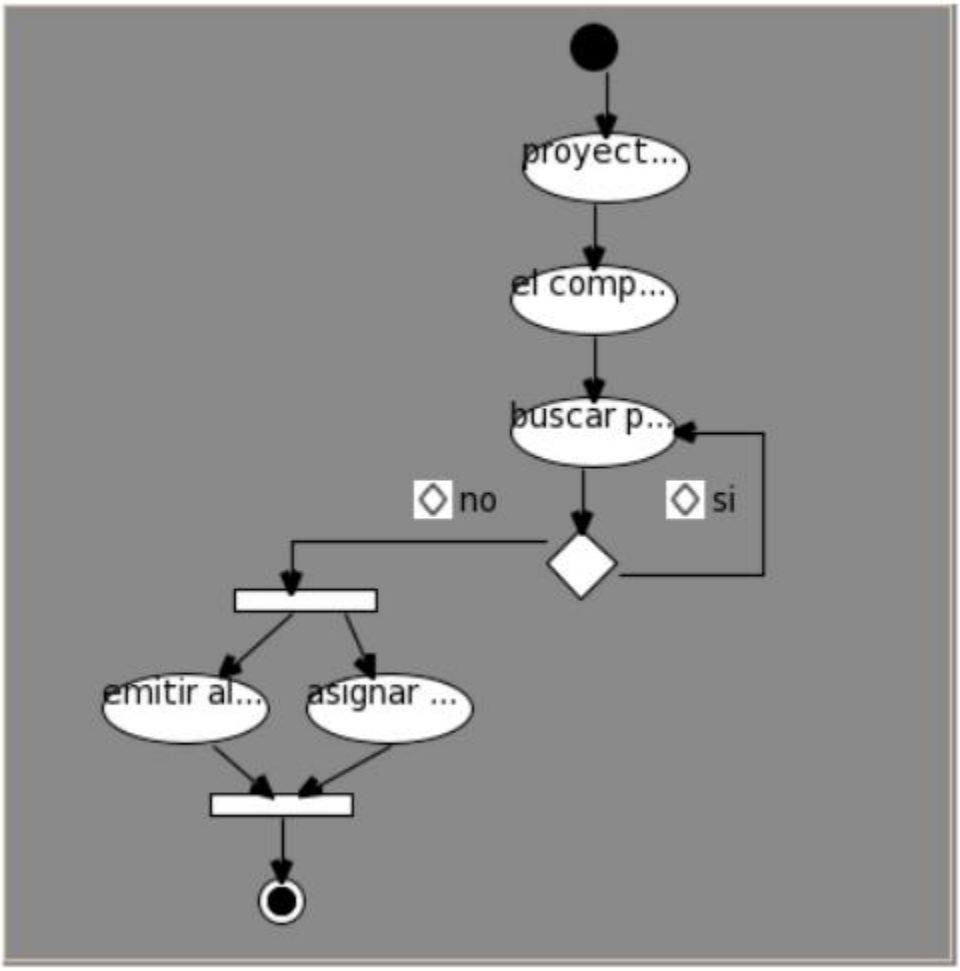


Figura 18.

Diagrama de flujo de tareas diseñado sobre el plugin.

Para una mejor comprensión del álgebra de tareas generado, se muestra la tabla 7, donde el contenido ha sido sangrado de acuerdo al estilo Allman. Cuando se obtiene el álgebra de tareas correspondiente al diagrama de flujos de tareas, esta álgebra es procesada por el compilador del álgebra de tareas. Como resultado se obtienen las trazas de la tabla 8. Al obtener las trazas correctas correspondientes al diagrama de flujo de tareas, queda de manifiesto que los resultados obtenidos son válidos.

```
{
   proyectarLaPalmaDeLaManoSobreElComponenteDeseado;
   elComponenteEsSombreadoYAncladoAlCursor;
   Mu.x
   (
      (
         buscarPosiciónDelComponente
      );
      Epsilon
      +
      x
   );
   (
      emitirAlertaVisual
      ||
      asignarElIDDelComponente
   );
   Sigma
}
```

Tabla 7.

Álgebra de tareas del diagrama de la Figura 14 sangrado conel estilo Allman.

```
fromList
[[proyectarLaPalmaDeLaManoSobreElComponenteDeseado,
elComponenteEsSombreadoYAncladoAlCursor,
buscarPosiciónDelComponente,!,asignarElIDDelComponente,
emitirAlertaVisual],
[proyectarLaPalmaDeLaManoSobreElComponenteDeseado,
```

elComponenteEsSombreadoYAncladoAlCursor,
buscarPosiciónDelComponente,!,
buscarPosiciónDelComponente,asignarElIDDelComponente,
emitirAlertaVisual],[
proyectarLaPalmaDeLaManoSobreElComponenteDeseado,
elComponenteEsSombreadoYAncladoAlCursor,
buscarPosiciónDelComponente,!,
buscarPosiciónDelComponente,emitirAlertaVisual,
asignarElIDDelComponente],[
proyectarLaPalmaDeLaManoSobreElComponenteDeseado,
elComponenteEsSombreadoYAncladoAlCursor,
buscarPosiciónDelComponente,!,emitirAlertaVisual,
asignarElIDDelComponente]]

Tabla 8.

Trazas correspondientes al diagrama de flujo de tareas trazar componente.

Una imagen del pulgin ejecutándose en Eclipse puede verse en la Figura 19, donde se muestra el diagrama realizado dentro del mismo IDE.

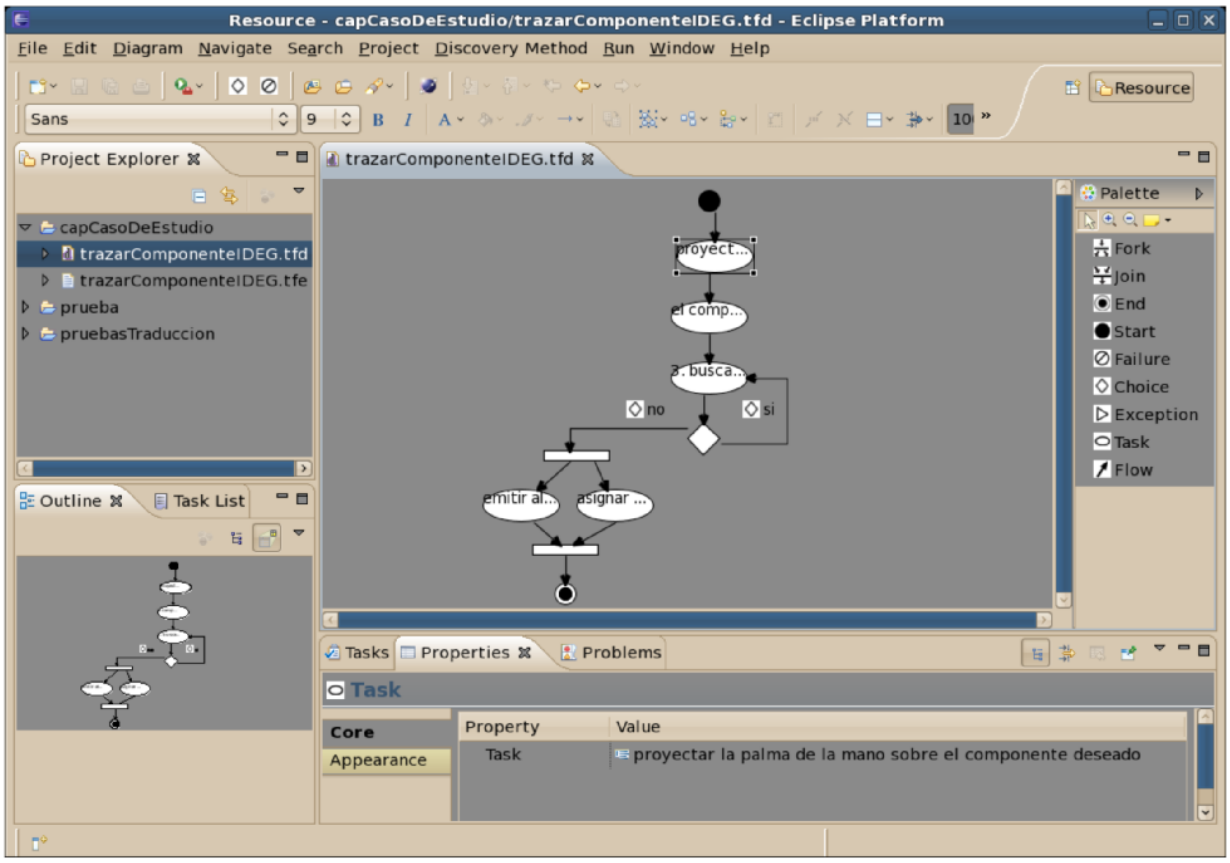


Figura 19.

Diagrama de flujos de tareas diseñado sobre el plugin.

5. Conclusiones

Generar el álgebra de tareas por medio de la utilización de expresiones temporales, es una forma adecuada, sobre todo al saber que los diagramas de flujo de tareas tienen la característica de ser no estructurados. Al usar esta técnica, se evita la necesidad de modificar los componentes visuales (Tabla 1) pudiendo mantener el objetivo planteado de hacer del diagrama de flujo de tareas algo sencillo de construir, además, se mantiene la similitud con el diagrama de actividades definido en UML.

Una de las ventajas de utilizar esta técnica de reducción de expresiones temporales, es que se podría diseñar e implementar una herramienta que

refleje de forma visual dichas reducciones, es decir, por cada reducción dada por una expresión temporal, el diagrama debe ser modificado para reflejar esa expresión temporal específica. De esta manera al realizar paso a paso la reducción completa de un diagrama de flujo de tareas, se puede llegar a una compresión más rápida del proceso para generar el álgebra de tareas. También se reconoce la necesidad de establecer mejoras en el álgebra de tareas, tal como la que se presentó en (Fernández-y-Fernández, 2012).

Agradecimientos

Esta investigación fue financiada parcialmente por la Universidad Tecnológica de la Mixteca.

Referencias

Aho, A.V., Sethi, R., Ullman, J.D. & Flores-Suarez, P. 1990. Compiladores: principios, técnicas y herramientas. México: Addison-Wesley Iberoamericana.

Benot, M. 2001. XML con ejemplos. México: Pearson education.

Budinsky, F., Steinberg, D., Merks, E., Ellersick, R. & Grose, T.J. 2003. Eclipse Modeling Framework: A Developer's Guide. Boston, Massachusetts: Addison Wesley, firts ed.

Chonoles, M. J. 2003. UML 2 for dummies. 909 Third Avenue New York, NY 10022: Wiley Publishing, Inc.

Fernández-Santos, H., Fernández-y-Fernández, C.A. & Quintanar-Morales, J.A. 2011. An ide to build and check task flow models, Advances in Computer Science and Applications, Research in Computer Science, vol. 53, pp. 23–33.

Fernández-y-Fernández, C.A. & Quintanar-Morales, J.A. 2012. Integrated development environment gesture for modeling workflow diagrams, Congreso Internacional de Investigacion e Innovacion en Ingenieria de Software (Conisoft 2012), vol. abs/1205.0751.

Fernández-y-Fernández, C.A. 2010. The Abstract Semantics of Tasks and Activity in the Discovery Method. PhD thesis, The University of Sheffield, Sheffield, UK.

Fernández-y-Fernández, C.A. 2012. Towards a new metamodel for the task flow model of the discovery method, Congreso Internacional de Investigación e Innovación en Ingeniería de Software.

Fernández-y-Fernández, C.A., Acosta-Mesa, H.G. & Cruz-Ramírez, N. 2011. Apuntes para un aprendiz de programador app inventor, programación de dispositivos móviles al alcance de todos, Temas de ciencia y Tecnología, vol. 15.

Graham, I. & Simons, A.J.H. 1999. 30 things that go wrong in object modelling with uml 1.3, Kluwer Academic Publishers.

Hernández Valdemar, E. J. & León, U. 2012. El paradigma de la programación visual, Fundación Arturo Rosenblueth.

Schmuller, J. 2009. UML en 24 hrs. Madrid España: Prencice Hall.

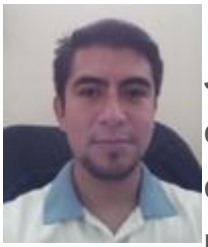
Simons, A.J.H. 1998. Object Discovery: A process for developing applications. Oxford : BCS.

Simons, A.J.H. 2006. Discovery history, url: http://staffwww.dcs.shef.ac.uk/people/A.Simons/discovery/, Último acceso Junio 23 2012.

Teufel, B., Schmid, S. & Teufel, T. 1995. Compiladores conceptos fundamentales. México: AaddisonWesley Iberoamericana.

Notas biográficas:

Carlos Alberto Fernández y Fernández Egresado de la Facultad de Informática de la Universidad Veracruzana, con una Maestría en Ciencias de la Computación en la Fundación Arturo Rosenblueth. Recibió el grado de Doctor en Ciencias de la Computación en la Universidad de Sheffield, Inglaterra. Se encuentra adscrito al Instituto de Computación de la Universidad Tecnológica de la Mixteca. Ha sido coordinador de la Universidad Virtual y de la Maestría en Computación con especialidad en Sistemas Distribuidos. Trabaja dentro del área de Ingeniería de Software, particularmente en las líneas de modelado visual, métodos de desarrollo y especificación formal de software. Ha sido responsable del Cuerpo Académico de Ingeniería de Software en la UTM y miembro del Verification and Testing Research Group en la Universidad de Sheffield.



José Ángel Quintanar Morales Ingeniero en Computación egresado de la Universidad Tecnológica de la Mixteca. Realizó estudios de posgrado en la Maestría en Medios Interactivos en la misma universidad, de la cual esta realizando su tesis de grado. Actualmente se desempeña como líder de proyectos en KadaSoftware.