Correspondencia entre el Modelo de una Aplicación y su Implementación Utilizando MERODE (Object Oriented Business Modelling) como Metodología de Análisis

Karina Alexandra Chong Escobar
Facultad de Ingeniería en Electricidad y Computación
Escuela Superior Politécnica del Litoral
Guayaquil, Ecuador
kchong@fiec.espol.edu.ec

María Verónica Macías Mendoza
Facultad de Ingeniería en Electricidad y Computación
Escuela Superior Politécnica del Litoral
Guayaquil, Ecuador
mmacias@fiec.espol.edu.ec

Resumen

MERODE es un método de análisis Orientado a Objetos construido sobre el paradigma de modelamiento basado en el dominio. Este método se basa en la elaboración de una estructura de la solución más parecida a la estructura del problema lo que aporta una fácil adaptación a los cambios del problema y finalmente reducciones en el costo de mantenimiento. El presente artículo resume los pasos a seguir para transformar el modelo de un dominio realizado con MERODE a la implementación de una aplicación. Para ello se explican las técnicas utilizadas en MERODE para construir el modelo: el gráfico de la dependencia de existencia, la tabla de eventos de objetos y la máquina de estados finitos; el diseño de la arquitectura compuesta por la capa de la empresa, la capa funcional y la capa de interfaz del usuario y finalmente la correspondencia entre el modelo producto del análisis, el diseño arquitectónico y la implementación. Se añade además un caso de estudio para una mejor explicación de esta correspondencia.

Palabras Claves: Análisis Orientado a Objetos, MERODE, correspondencia, implementación.

Abstract

MERODE is an Object Oriented Analysis method constructed on the domain-driven modeling paradigm. This method is based on the development of a structure of the solution most similar to the structure of the problem which contributes to an easy adaptation to the changes of the problem and reductions in the maintenance costs. The present article summarizes the steps to convert a domain model constructed with MERODE to the implementation of an application. In order to do this, first we explain the three techniques used in MERODE to construct a model: the existence dependency graph, the object event table and the finite states machine; the design of the architecture composed by the enterprise layer, the functionality layer and the user interface layer and finally the correspondence between the domain model, the architectonic design and the implementation. We also added a case of study for a better explanation of this correspondence.

1. Introducción

El modelamiento basado en el dominio o Domain-Driven Modeling, es un nuevo paradigma de Software que destaca la construcción de herramientas de software sobre la base del Modelo del negocio o dominio [1]. MERODE es un método de análisis Orientado a Objetos que se basa en este paradigma [1]. En la ESPOL se dictan cursos basados en esta metodología, sin embargo por razones de tiempo se le da un enfoque teórico y exclusivo al análisis y se deja de lado las siguientes fases necesarias para la realización de aplicaciones prácticas.

Es por esto que se propone el desarrollo del presente artículo, donde se presenta de una manera breve la metodología usada, su arquitectura y una síntesis de los pasos para realizar la correspondencia entre el modelo de una aplicación utilizando MERODE y su implementación. Se presenta luego un caso de estudio que permita afianzar la información proporcionada en el artículo y finalmente se dan las conclusiones y recomendaciones.

2. Contenido

2.1. Descripción de la metodología

MERODE es un método de análisis orientado a objetos que fue desarrollado en la Universidad Católica de Leuven, Bélgica, y cuyas siglas significan Model-based, Existence – dependency Relationship Object – oriented Development (desarrollo orientado a Objetos basado en el modelo de las relaciones de Dependencia de Existencia) [2].

MERODE posee 3 vistas que son: el gráfico de la dependencia de existencia, la tabla de eventos de objetos y finalmente las restricciones de secuencia y la máquina de estado finito. Las semánticas formales establecidas sobre cada una de estas vistas o esquemas aseguran su consistencia [3].

2.1.1. El gráfico de la dependencia de existencia (EDG. Existence Dependency Graph) EDG. Es un gráfico que representa al conjunto de clases en el modelo del dominio. El EDG modela la parte estática del dominio y está basado principalmente en la relación que existe entre un objeto maestro y su dependiente [2]. Esta relación es denominada dependencia de existencia y se basa en la vida de un objeto, que es el periodo que existe entre el momento de su creación hasta el momento de su eliminación, la existencia del objeto dependiente depende de la vida del objeto maestro [2]. En la figura 1 se puede apreciar la representación gráfica del EDG.

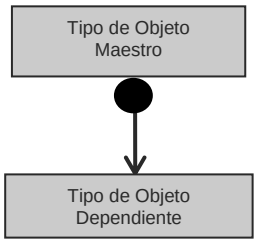

Figura 1. Representación gráfica del EDG.

2.1.2. La tabla de eventos de objetos (OET, Object-Event Table). Es una tabla que contiene una fila por cada tipo de evento y una columna por cada tipo de objeto. Cada celda contiene una 'C' (si representa un objeto creador), una 'E' (si representa un objeto finalizador), una 'M' (si representa un objeto modificador) o un espacio en blanco (cuando el objeto de la columna correspondiente no es afectado por el evento de la fila correspondiente). El OET sirve para representar aspectos dinámicos, mostrando que objetos del negocio son afectados por cada evento del negocio y en que forma [2], en la figura 2 se muestra gráficamente el OET.

	Obj1	Obj2
crear_obj1	C	
fin_obj1	E	
crear_obj2		C
fin_obj2		E
mod_obj1	M	
mod1_obj2		M
mod2_obj2		M
fin2_obj2		E

Figura 2. OET

2.1.3. Restricciones de secuencia y Maquina de Estado Finito. Las restricciones de secuencia son la combinación de las restricciones específicas de secuencia de eventos para cada tipo de objeto y sirven para representar aspectos dinámicos del dominio [2]. Para modelar las restricciones de secuencia MERODE utiliza la Máquina de Estado Finito (FSM), cuyos nodos representan los estados de un objeto específico y los arcos representan los eventos que crean, modifican o eliminan dicho objeto. La figura 3 muestra el FSM del objeto obj1.

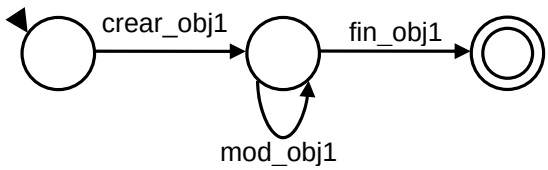

Figura 3. Ciclo de vida por defecto representado como FSM del objeto obj1

2.2. Diseño Arquitectónico

La arquitectura que utiliza MERODE está compuesta por tres capas: la capa de la empresa, la capa funcional y la capa de interfaz del usuario [2].

2.2.1. La capa de la empresa. Contiene los objetos de la empresa así como la lógica del negocio.

2.2.2. La capa funcional. Contiene la lógica de la aplicación así como los datos de la aplicación. Es llamada también modelo de servicios ya que es construida por un conjunto de servicios de entrada y salida que ofrecen información funcional a los usuarios del sistema de información [5]. Los servicios de entrada permiten a los usuarios registrar o modificar información que es relevante para el negocio. Los servicios de salida permiten al usuario extraer información del modelo del dominio del negocio y presentarlo en un formato correcto.

2.2.3. La capa de interfaz del usuario. Representa la interacción con el cliente. Sirve como interfaz para las entradas de información que realiza el cliente y para las salidas del sistema.

2.3. Correspondencia entre el modelo de una aplicación y su implementación

En esta sección se procederá a hacer una correspondencia entre el modelo producto del análisis de MERODE, las capas del diseño arquitectónico y la implementación.

La capa de la empresa perteneciente al diseño arquitectónico a su vez se divide en objetos del negocio y eventos del negocio, de la misma forma la capa funcional también perteneciente al diseño arquitectónico se divide en servicios de entrada y salida. A continuación se describe la relación que existe entre estas divisiones y las vistas de MERODE.

2.3.1. Objetos del negocio. A cada objeto del EDG le corresponde una columna en el OET y por cada una de estas columnas se define una clase. Cada clase que representa a un objeto contendrá un método por cada C, M, y E que contenga en las columnas del OET. Es decir que existirá una clase por cada objeto existente en el EDG a la que se le añadirá un método por cada evento del OET que afecta al objeto que representa esa clase.

2.3.2. Eventos del negocio. Se define una clase por cada tipo de evento. Cada una de estas clases de evento contiene dos métodos, validar y ejecutar. El método *validar* chequea si el evento puede ser aceptado verificando que se cumplan todas las precondiciones especificadas en términos de estados de objetos, valores de parámetros, restricciones de integridad referencial, restricciones de cardinalidad, etc. Si una de las restricciones no se cumple el método validar retorna un código de error y un mensaje que explica al usuario cual fue el motivo del error. Si todas las condiciones se satisfacen el método *ejecutar* transmitirá el evento a todos los objetos involucrados [4].

2.3.3. Servicios de entrada y salida. Cada clase que representa un evento del negocio es llamada a través de su servicio de entrada [6]. Este servicio de entrada es a su vez otra clase que recibe como parámetros la información que ingresa el usuario a través de la interfaz y que llama al evento o eventos del negocio involucrados y le envía los parámetros que este o estos necesitan. Los servicios de salida son clases que se comunican con los objetos del negocio y extraen información necesaria para presentar al usuario [6].

2.3.4. Interfaz del usuario. Son las páginas que se presentan al usuario, éstas servirán para mostrar la información que nos proporcionan las funciones o servicios de salida y para ingresar la información que necesitan las funciones de entrada [6].

2.4. Caso de estudio

A continuación se presentará un ejemplo que permitirá ilustrar los pasos a seguir para realizar la correspondencia descrita en las secciones anteriores.

Se tienen 3 objetos el Objeto 1, Objeto 2 y Objeto 3. Los Objetos 1 y 2 son objetos maestros y el Objeto 3 es un objeto dependiente del Objeto 1 y también del Objeto 2. En la figura 4 podemos observar el EDG que nos muestra gráficamente las relaciones de los objetos, además se presenta el OET en la Tabla 1 y los FSM de los objetos 1, 2 y 3 en las figuras 5, 6, y 7 respectivamente.

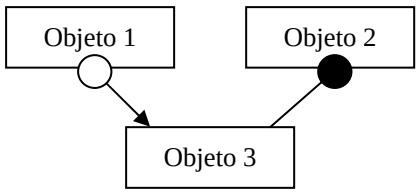

Figura 4. EDG del ejemplo

Tabla 1. OET del ejemplo

	Objeto 1	**Objeto 2**	**Objeto 3**
crear_Objeto 1	O/C		
fin_Objeto 1	O/E		
crear _Objeto 2		O/C	
fin_Objeto 2		O/E	
crear_Objeto 3	**A/M**	**A/M**	**O/C**
fin_Objeto 3	A/M	A/M	O/E
modificar_Objeto 3	A/M	A/M	O/M

Como se puede apreciar en la tabla 1 existe una columna por cada objeto y una fila por cada evento que afecta a estos objetos. La notación utilizada en cada celda denota la proveniencia (O: Owned o propio, y A: Acquired o adquirido) y la manera (C, E, M) en que afectan los eventos a los objetos. Por ejemplo el evento cr_Objeto 3 es un evento adquirido que modifica a los Objetos 1 y 2 (A/M) y es un evento propio que crea al Objeto 3 (O/C).

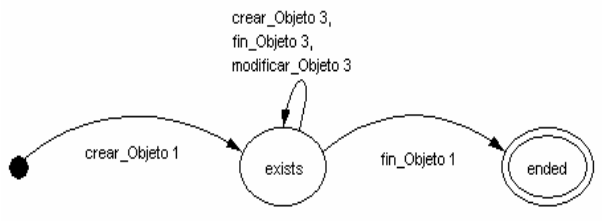

Figura 5. FSM del Objeto 1

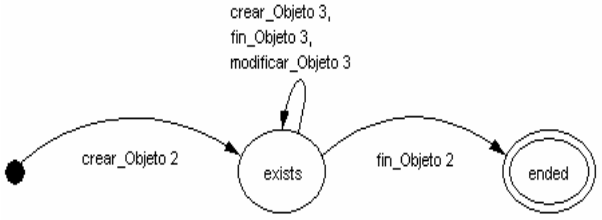

Figura 6. FSM del Objeto 2

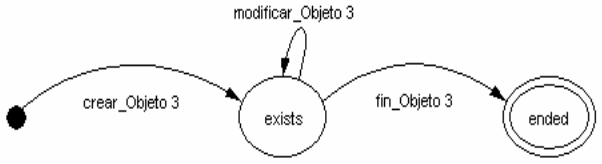

Figura 7. FSM del Objeto 3

Los FSM presentados en las figuras 5, 6 y 7 son FSM que tienen un ciclo de vida por defecto, es decir se crean, se modifican y luego se eliminan.

A continuación se realizará la respectiva correspondencia entre el modelo, el diseño arquitectónico y la implementación.

2.4.1. Objetos del negocio. Se crearán 3 clases, una por cada objeto, llamaremos a las clases Clase_Objeto 1, Clase_Objeto 2 y Clase_Objeto 3. Cada clase contendrá un método por cada C, M y E que contenga en las columnas del OET el objeto que representa a esa clase. Por lo tanto:

La clase Clase_Objeto 1 tendrá los métodos crear_Objeto 1(), fin_Objeto 1(), crear_Objeto 3 (), fin_Objeto 3 () y modificar_Objeto 3 ().

La clase Clase_Objeto 2 tendrá los métodos crear_Objeto 2(), fin_Objeto 2(), crear_Objeto 3 (), fin_Objeto 3 () y modificar_Objeto 3 ().

La clase Clase_Objeto 3 tendrá los métodos crear_Objeto 3 (), fin_Objeto 3 () y modificar_Objeto 3 ().

En las figuras 8, 9, y 10 se muestran las representaciones en UML de las Clase_Objeto 1, Clase_Objeto 2 y Clase_Objeto 3 respectivamente.

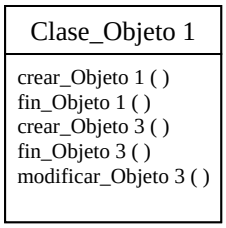

Figura 8. Representación UML de la Clase correspondiente al Objeto 1

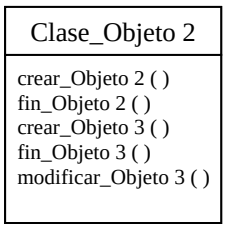

Figura 9. Representación UML de la Clase correspondiente al Objeto 2

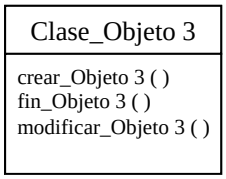

Figura 10. Representación UML de la Clase correspondiente al Objeto 3

Se puede observar que los métodos que participan en cada una de las clases de las figuras 8, 9 y 10 son el conjunto de todos los métodos que se presentan en el FSM de cada clase, ver figuras 5, 6 y 7 respectivamente.

2.4.2. Eventos del negocio. Existirá una clase por cada evento en el OET lo que significa que en nuestro ejemplo constarán 7 clases: crear_Objeto 1, fin_Objeto 1, crear_Objeto 2, fin_Objeto 2, crear_Objeto 3, fin_Objeto 3, modificar_Objeto 3. Cada una con dos métodos: validar y ejecutar. Desde la figura 11 hasta la figura 17 se puede apreciar cada clase en notación UML.

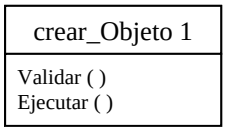

Figura 11. Representación UML de la Clase correspondiente al evento crear_Objeto 1

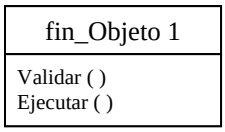

Figura 12. Representación UML de la Clase correspondiente al evento fin_Objeto 1

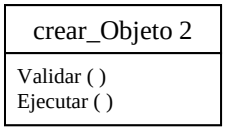

Figura 13. Representación UML de la Clase correspondiente al evento crear_Objeto 2

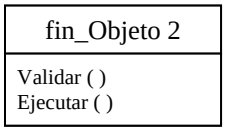

Figura 14. Representación UML de la Clase correspondiente al evento fin_Objeto 2

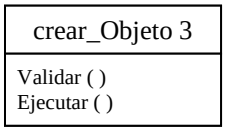

Figura 15. Representación UML de la Clase correspondiente al evento crear_Objeto 3

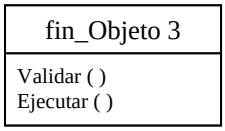

Figura 16. Representación UML de la Clase correspondiente al evento fin_Objeto 3

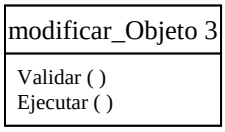

Figura 17. Representación UML de la Clase correspondiente al evento modificar_Objeto 3

2.4.3. Servicios de entrada y salida. Cada evento del negocio es llamado a través de una clase que representa su servicio de entrada. Para nuestro ejemplo existirán 7 de estas clases: F_crear_Objeto 1, F_fin_Objeto 1, F_crear_Objeto 2, F_fin_Objeto 2, F_crear_Objeto 3, F_fin_Objeto 3, F_modificar_Objeto 3. Y los servicios de salida nos permitirán mostrar información de los objetos. Para nuestro ejemplo existirán 3 de estos: F_mostrar_Objeto 1, F_mostrar_Objeto 2, F_mostrar_Objeto 3.

A continuación se listan las clases a construirse para los servicios de entrada y los servicios de salida:

Servicios de entrada:
- F_crear_Objeto 1
- F_fin_Objeto 1
- F_crear_Objeto 2
- F_fin_Objeto 2
- F_crear_Objeto 3
- F_fin_Objeto 3
- F_modificar_Objeto 3

Servicios de salida:
- F_mostrar_Objeto 1
- F_mostrar_Objeto 2
- F_mostrar_Objeto 3

2.4.5. Interfaz del usuario. Dependerá del lenguaje de programación o la herramienta que se está utilizando para interactuar con el usuario.

Para este caso particular se obtuvo como resultado un total de 20 clases, generadas a partir de 3 objetos del modelo y 7 eventos que se realizan sobre estos objetos.

3. Conclusiones y recomendaciones

Podemos concluir que el proceso de correspondencia incluye los siguientes pasos:
- Por cada objeto del EDG creamos una clase. A cada uno de estos objetos se añade un método por cada evento del OET que afecta al objeto que representa esa clase.
- Por cada evento del negocio habrá una clase con 2 eventos, validar y ejecutar.
- Por cada servicio de entrada y salida habrá una clase.
- El número de clases correspondiente a la interfaz de usuario es indefinido.

Implementar las capas que utiliza MERODE sigue un proceso mecánico y extenso, debido a sus múltiples capas.

Debido a que la implementación de las capas que utiliza MERODE es un proceso mecánico, se recomienda utilizar una herramienta que permita agilizar este proceso. Actualmente existe un generador de código desarrollado por el grupo de Administración de Sistemas de Información de la Universidad Católica de Leuven (KULeuven), pero sólo puede ser utilizado en la intranet de dicha Universidad.

El curso introductorio de MERODE que se dicta en la FIEC, permite desarrollar ciertas destrezas a nivel de técnicas de MERODE, pero falta profundidad en cuanto a la correspondencia del análisis y diseño con la implementación. Por lo que se recomienda que en los próximos cursos que se dictan semestralmente tanto a estudiantes como a profesionales se dé énfasis en este tema para que los conocimientos adquiridos puedan ser aplicados a problemas reales de una manera práctica.

4. Agradecimientos

Este artículo fue realizado gracias al apoyo del Subcomponente de Ingeniería de Software del Componente 8 del Proyecto VLIR-ESPOL.

5. Referencias

[1] Macías, M., "Modelamiento basado en el Dominio: Estado del Arte", Proceedings Jornadas de Ingeniería de Software 2004, Noviembre 2004, pp. 33-41.

[2] Snoeck, M., Dedene, G., Verhelst, M., Depuydt, A-M., Object-Oriented Enterprise Modelling with MERODE, Leuven University Press, 1999.

[3] Snoeck M., Michiels C., Dedene G., "Consistency by construction: the case of MERODE", in Jeusfeld, M. A., Pastor, O., (Eds.) Conceptual Modeling for Novel Application Domains, ER 2003 Workshops ECOMO, IWCMQ, AOIS, and XSDM, Chicago, IL, USA, October 13, 2003, *Proceedings*, 2003 XVI, 410 p., Lecture Notes in Computer Science, Volume 2814, pp.105-117.

[4] Lemahieu W., Snoeck M., Michiels C., Goethals F., "An Event Based Approach to Web Service Design and Interaction", accepted for APWEB'03, Web Technologies and Applications: 5th Asia-Pacific Web Conference, APWeb 2003, Xian, China, April 23-25, 2003. *Proceedings*, LNCS, Volume 2642, 2003, Springer.

[5] Snoeck M., Lemahieu W., Michiels C., Dedene G., "Event-based Software Architectures", in Konstantas, D., Leonard, M., Pigneur, Y., Patel, S., (Eds.) , Object-Oriented Information Systems, 9th International Conference, OOIS 2003, Geneva, Switzerland, September 2-5, 2003, Proceedings, 2003 XII, 426 p., Lecture Notes in Computer Science, Volume 2817, pp. 107-117.

[6] Wilfried Lemahieu, Monique Snoeck, Cindy Michiels, Frank Goethals, Guido Dedene, Jacques Vandenbulcke, "Event Based Web Service Description and Coordination", accepted for WES'03, Web Services, e-Business, and the Semantic Web, in Workshop Proceedings of CAiSE 2003, Klagenfurt, Austria, June 16-18, 2003, *Proceedings*, 2003 XV, 740 p. Lecture notes in Computer Science, Forthcoming Volume, 2003 Springer