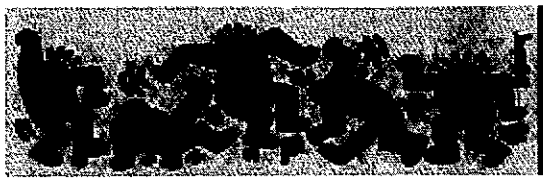
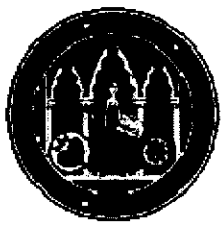


IV Jornadas de Trabajo DYNAMICA



Archena (Murcia), 17, 18 de Noviembre de 2005

Presidente de las Jornadas

José Ambrosio Toval Álvarez

Edición de las Actas

Francisco J. Lucas Martínez
Joaquín Nicolás Ros
José Sáez Martínez

Comité Organizador

José Luís Fernández Alemán
Joaquín Lasheras Velasco
Francisco J. Lucas Martínez
Miguel Ángel Martínez Aguilar
Fernando Molina Molina
Begoña Moros Valle
Joaquín Nicolás Ros
José Sáez Martínez
Cristina Vigueras Ruiz

IV Jornadas de Trabajo DYNAMICA

Archena (Murcia), 17 y 18 de noviembre de 2005

Presentación

El proyecto DYNAMICA (***DYNamic and Aspect-Oriented Modeling for Integrated Component-based Architectures***) es un proyecto coordinado financiado por el Ministerio de Ciencia y Tecnología en el periodo 2004-2006. En DYNAMICA colaboran cinco grupos de investigación de universidades españolas: el grupo de Ingeniería del Software y Sistemas de Información (ISSI) de la Universidad Politécnica de Valencia (UPV), el grupo de División e Ingeniería Electrónica (DSIE) de la Universidad Politécnica de Cartagena (UPCT), el grupo ALARCOS de la Universidad de Castilla-La Mancha (UCLM), el grupo del Departamento de Ingeniería de Sistemas y Automática (ISA) de la Universidad Carlos III (UC3M) y el Grupo de Ingeniería del Software (GIS) de la Universidad de Murcia (UMU).

A partir de aplicaciones reales, como por ejemplo los sistemas teleoperados para limpieza de cascos de buques, los portales web, o las aplicaciones de diagnóstico médico, y de aspectos de calidad y seguridad, el proyecto DYNAMICA tiene un amplio ámbito que incluye el desarrollo de un conjunto de modelos, lenguajes y herramientas CASE para la construcción de modelos arquitectónicos basados en aspectos y componentes para arquitecturas complejas, distribuidas, evolutivas y reutilizables; la definición y validación de métodos de especificación de requisitos, como un "*front-end*" para los modelos arquitectónicos anteriores, teniendo en cuenta aspectos de seguridad y de línea de productos; y la definición y validación, empírica y formal, de un conjunto de métricas que permitan asegurar la calidad de un producto software.

En estas actas se recopilan los trabajos enviados a las IV Jornadas de Trabajo del Proyecto DYNAMICA, celebradas en Archena (Murcia) durante los días 17 y 18 de noviembre de 2005. Estas jornadas de trabajo DYNAMICA continúan con el espíritu de las precedentes, celebradas respectivamente en Cartagena, Málaga y Almagro (Ciudad Real), de aportar un espacio para comentar los avances logrados en la ejecución del proyecto, para discutir los trabajos actuales, y para definir nuevos trabajos a desarrollar en el futuro.

Índice

Formalización de Métricas: OCL vs. Maude

José A. Cruz-Lemus[1], Francisco J. Lucas[2], Marcela Genero[1], Ambrosio Toval[2] y Mario Piattini[1]

[1]Grupo ALARCOS
Departamento de Tecnologías y Sistemas de Información, Universidad de Castilla-La Mancha
Paseo de la Universidad, 4 – 13071 Ciudad Real, España
{JoseAntonio.Cruz, Marcela.Genero, Mario.Piattini}@uclm.es

[2]Grupo de Investigación de Ingeniería del Software
Departamento de Informática y Sistemas, Universidad de Murcia
Campus de Espinardo s/n, - 30071 Murcia, España
{fjlucas, atoval}@um.es

Resumen

En los últimos años se han realizado un gran número de propuestas de métricas para productos de software orientado a objetos (OO), tanto a nivel de modelos conceptuales como a nivel de código. Un gran número de estas propuestas carecen de una formalización, lo que implica que lleven asociados problemas como la ambigüedad, la posibilidad de ser interpretadas de manera diferente o la dificultad en su comprensión, además de la dificultad que se genera a la hora de automatizar métricas no formalizadas. En este trabajo presentamos, a través de unos ejemplos de utilización, dos posibles mecanismos de formalización de métricas: los lenguajes OCL y Maude, mostrando algunas de sus principales ventajas e inconvenientes.

1. Introducción

A pesar del gran número de métricas para productos de software OO, tanto a nivel de modelos como a nivel de código que se han propuesto en los últimos años, muchas de ellas contienen una serie de problemas que se repiten insistentemente. Entre ellos, en [3] se destaca que las métricas carecen de formalización, ya que normalmente se definen o bien de manera informal o bien sólo se formalizan parcialmente usando lenguajes formales, pero definiendo de manera informal los conceptos subyacentes.

A causa de la existencia de éstos y algunos otros problemas asociados a la definición de métricas aparecen consecuencias tales como la presencia de definiciones ambiguas que llevan a interpretaciones diferentes y resultados divergentes aún en los mismos artefactos software, la dificultad en la comprensión y el uso de las métricas, la dificultad a la hora de probar la utilidad de los conjuntos de métricas, con la consecuente imposibilidad de una aceptación ampliamente extendida y la existencia de entidades de diseño que nunca han sido medidas [3].

Con el fin de mitigar esta situación, en varias metodologías [4, 5] se propone una correcta creación, formalización y validación de métricas para modelos UML como un proceso esencial para conseguir, entre otras ventajas comprender mejor lo que las métricas existentes capturan, si realmente son distintas y si son indicadores útiles de

atributos de la calidad como la mantenibilidad, la reusabilidad, etc. Estas métricas correctamente definidas y validadas pueden contribuir a que los diseñadores puedan tomar mejores decisiones en su tarea, lo que debería ser el objetivo final de cualquier propuesta de medición. Además, la correcta formalización de las métricas facilita en gran medida el proceso de automatización del cálculo de las mismas.

En este trabajo nos dedicaremos sólo a la etapa de definición de métricas, mostrando las principales ventajas y desventajas de los lenguajes OCL *(Object Constraint Language)* [15] y Maude [7] a la hora de ser usados en la definición formal de métricas.

El resto del presente trabajo se estructura de la siguiente manera: en el apartado 2 se abordarán las características primordiales del uso de OCL en la formalización de métricas, en el apartado 3 se comentarán estas características pero cuando se utiliza el lenguaje Maude para formalizar. Posteriormente, en el apartado 4 se analizarán a modo de resumen las principales características de ambas alternativas y, por último, en el apartado 5 se presentarán las conclusiones a las que se han llegado tras la realización de este trabajo.

2. Formalización de Métricas usando OCL

Esta sección se compone de tres partes: en la primera se hará una breve descripción del lenguaje OCL, en la segunda se presentará una arquitectura para formalización de métricas basada en OCL propuesta en [2] y, por último, en la tercera parte se presentará un ejemplo del uso de esta arquitectura.

2.1. Fundamentos de OCL

OCL es un lenguaje de expresión y consulta que se utiliza conjuntamente con diagramas de UML. Se basa en la lógica y en la teoría de conjuntos pero su diseño está orientado a la usabilidad y cualquier persona familiarizada con conceptos del modelado conceptual orientado a objetos puede comprenderlo, más aún si se habla de modelado basado en UML.

OCL puede utilizarse para distintos propósitos: como lenguaje de consulta, para especificar invariantes en clases y tipos de un modelo de clases, para especificar invariantes de tipos para estereotipos, para describir pre y post-condiciones en operaciones y métodos, para describir guardas, para especificar mensajes y acciones, para especificar reglas de derivación de un atributo, para especificar cualquier expresión de un modelo UML, para especificar restricciones en operaciones, etc.

OCL permite expresar tres tipos de restricciones: invariantes, pre-condiciones y post-condiciones. Las invariantes representan condiciones que deben cumplir todas las instancias de una clase durante su existencia. Las pre-condiciones deben ser ciertas para que se pueda ejecutar una operación y representan las obligaciones que deben satisfacer

los objetos que requieren un determinado servicio. Por último, las post-condiciones deben ser ciertas cuando una operación termina su ejecución y representa las obligaciones que debe satisfacer el objeto que ofrece un servicio.

En la Figura 1 podemos observar un ejemplo simple del uso de OCL.

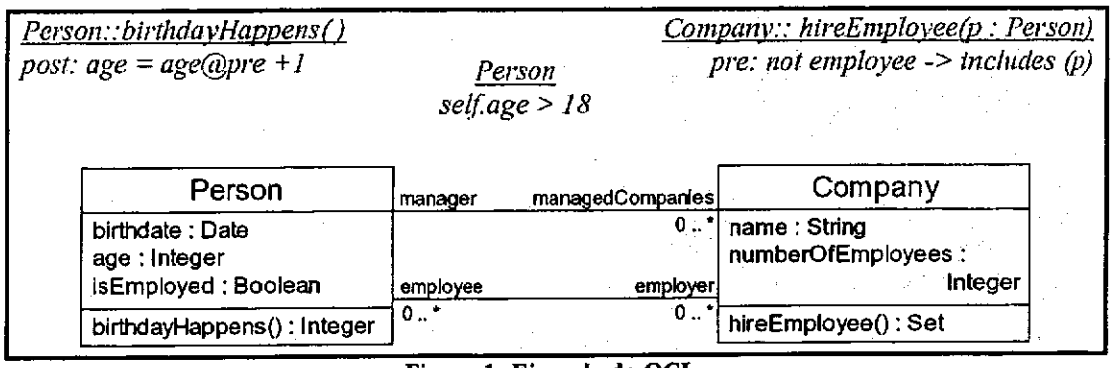


Figura 1. Ejemplo de OCL

En este ejemplo, la invariante asociada a una instancia (*self*) de la clase *Person*, determina que el valor del atributo edad (*age*) ha de ser mayor de 18 para que pueda producirse una asociación de una instancia de la clase *Person* con otra de la clase *Company*.

Estas restricciones permiten mejorar la precisión y conseguir un mejor diseño de la documentación, lo que acabará permitiendo lograr una comunicación carente de ambigüedades entre las partes implicadas, esto es, diseñadores, usuarios, probadores, programadores, gestores, etc.

Las expresiones OCL no poseen efectos laterales y pueden variar desde simples comparaciones en las que se establece el límite superior de un atributo hasta complejas navegaciones dentro de un diagrama de clases a través de sus asociaciones.

2.2. Una arquitectura basada en OCL para formalización de métricas

A la hora de formalizar métricas utilizando OCL, en [1] se indica que las limitaciones de aplicación de las métricas se deben definir con pre-condiciones OCL y que los resultados de las métricas pueden definirse formalmente con post-condiciones OCL.

Según estas indicaciones, en [2] se propone una arquitectura cuya representación a nivel de modelo puede observarse en la Figura 2.

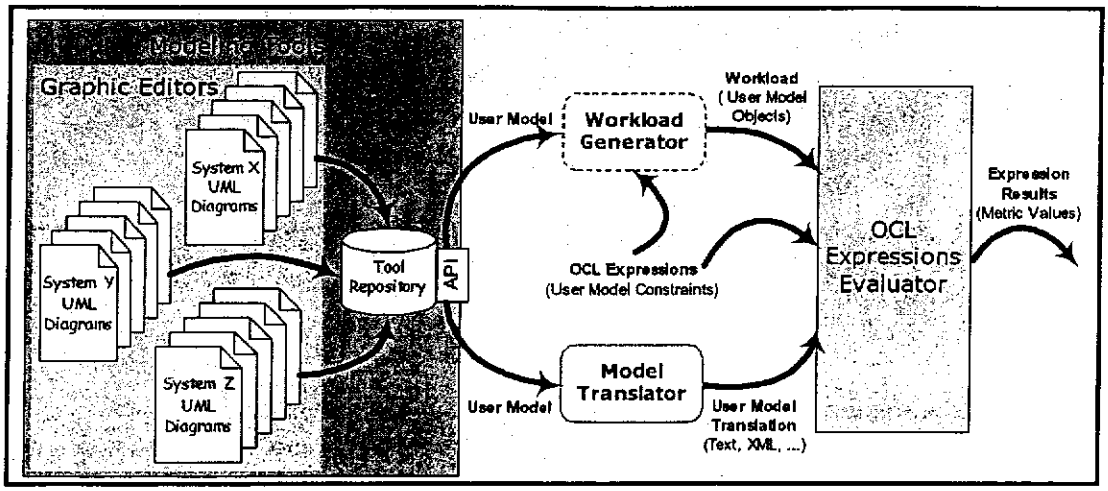


Figura 2. Arquitectura a nivel de Modelo [2]

En esta arquitectura se plantea que hay una serie de herramientas de modelado en el mercado que permiten modelar los sistemas. Estos modelos se almacenan en un repositorio. Desde aquí, los modelos se pueden exportar a través de otras herramientas de traducción a ficheros de texto, basados en XML, que pueden ser comprendidos por herramientas de evaluación de expresiones OCL. Estas herramientas toman la representación de los diagramas, las restricciones OCL añadidas y las instancias del modelo y evalúan cada restricción, mostrando los resultados.

2.3. Ejemplo de formalización de una métrica

Como ejemplo de utilización de esta arquitectura, se presenta la formalización de la métrica DIT (*Depth of the Inheritance Tree*) perteneciente al conjunto de métricas MOOSE [6] y que corresponde a la máxima profundidad de la cadena de herencia de una clase determinada, es decir, el número de relaciones de herencia entre una clase y la clase raíz.

```
Class::DIT():Integer
post: result = (if self.IsRoot() then
                   0
                else
                   1 + self.Parents()->iterate(elem:Class; acc:Integer=0 |
                                  if elem.DIT() > acc then
                                        elem.DIT()
                                  else
                                        acc
                                  endif)
                endif)
```

Figura 3. Formalización de la Métrica DIT

En este caso, *elem* es un iterador y hay una iteración sobre el conjunto de clases que hay por encima de la clase actual (*self*). La variable *acc* es un acumulador para contener el resultado y está inicializada a cero. El proceso de iteración permite calcular la máxima profundidad del árbol de herencia (*DIT( )*) teniendo en cuenta incluso la

herencia múltiple. En cada iteración, el iterador recibe el valor que se indica tras el símbolo '|', que corresponderá al valor de una llamada recursiva a *DIT( )* o a su valor previo cuando no se altere el acumulador.

3. Formalización de Métricas usando Maude

En esta sección abordaremos la formalización de métricas utilizando el lenguaje Maude. Para ello, en el primer apartado se dará una visión global del lenguaje, así como una serie de ventajas que incorpora su utilización a la hora de formalizar métricas. En el segundo apartado se mostrará un ejemplo concreto de formalización, en concreto se trata de la formalización de un conjunto de métricas para diagramas de estados UML definidas en [8] y que se basan en trabajos previos [16].

3.1. Fundamentos de Maude

Maude es un lenguaje de especificación formal basado en lógica ecuacional y lógica de reescritura. Las principales características de este lenguaje consisten en:

- Expresividad. Gracias a los formalismos en los que se basa y la gran abstracción que se consigue, con Maude pueden definirse cualquier tipo de sistemas con muchas menos líneas que en otros lenguajes.
- Sencillez: la forma de programar en este lenguaje es muy sencilla.
- Rendimiento: pese a que este tipo de lenguajes declarativos se han caracterizado siempre por no ser muy eficientes, en Maude se ha hecho un gran esfuerzo en hacer el lenguaje lo más eficiente posible.

Maude es además un lenguaje que se ha utilizado en multitud de dominios y aplicaciones diferentes (definición de lenguajes de programación, análisis de protocolos de comunicaciones,...) [13].

Las principales ventajas de la utilización de Maude como lenguaje de formalización de métricas son, en primer lugar, que se trata de un lenguaje de especificaciones formal, con lo cual se elimina la posible ambigüedad en la definición de las métricas. Otra ventaja consiste en que al poder formalizar los distintos diagramas de UML [9, 10, 12, 17], podemos aprovechar la potencia expresiva del lenguaje para manejar los distintos elementos que forman los diagramas y de esta forma tener métricas más legibles y compactas. Por último, la creación de modelos formales permite la posterior generación automática de prototipos.

3.2. Formalización de diagramas de estados UML usando Maude

Para llevar a cabo esta formalización se tomaron como base las especificaciones formales construidas en [9] y que se utilizaron originalmente para comprobar la ortogonalidad en un diagrama de estados UML. En este caso, el objetivo que se pretende radica en combinar por un lado la intuitiva y bien conocida apariencia de los diagramas UML con la precisión que aporta el uso de un lenguaje formal como Maude.

Sirva el ejemplo que se presenta en la Figura 4 como ejemplo e hilo conductor del proceso que se ha llevado a cabo para formalizar, de nuevo, la métrica DIT.

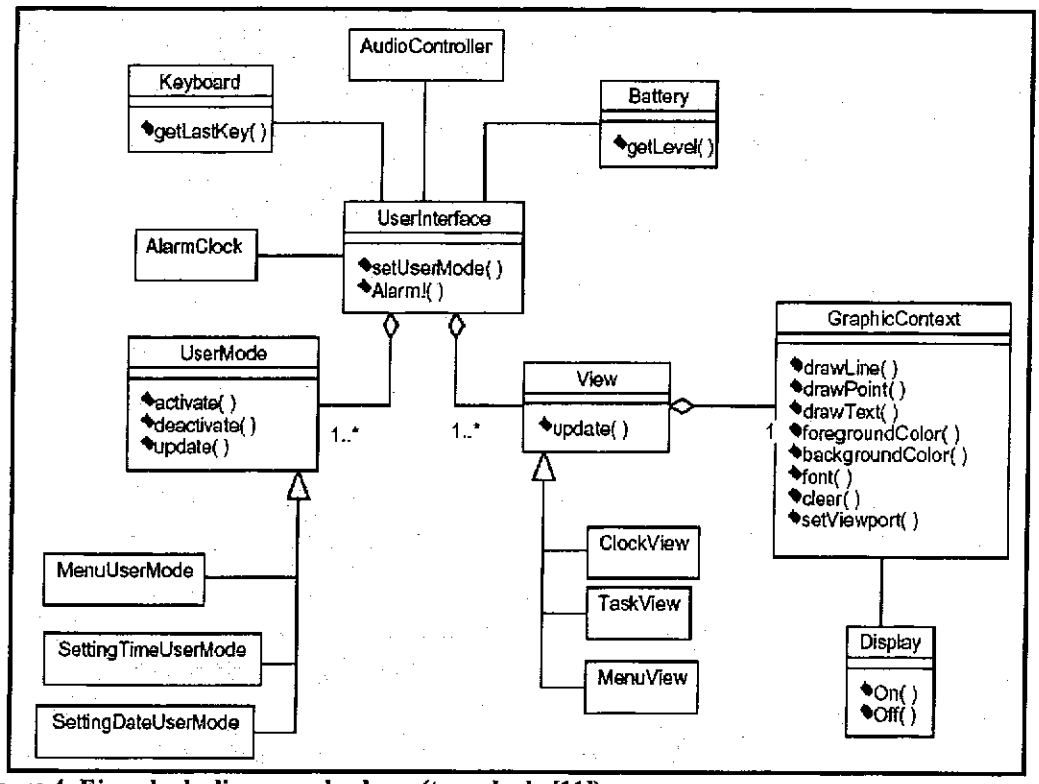


Figura 4. Ejemplo de diagrama de clases (tomado de [11])

El primer paso consiste en realizar la especificación algebraica del diagrama. La Figura 5 muestra parte de esta especificación.

```
...
  op classList : -> ClassList .
  op objList : -> ObjectList .
  op tree : -> ClassTree .   op tree1 : -> ClassTree .
  op tree2 : -> ClassTree .   op tree3 : -> ClassTree .

  op ctList : -> ClassTreeList .   op assocList : -> AssocList .
  op ai : -> AssocInstance .   op ai1 : -> AssocInstance .
  op ai2 : -> AssocInstance .   op assocInstList : -> AssocInstanceList .

  *** DIAGRAMA DE CLASES
  op dcu : -> ClassDiagram .

  *** LISTA DE CLASES
  eq classList =
        class ( 'AudioBlock, attribute (public, 'attribute1, multiplicity (1,1), 'Integer)
                  attribute (public, 'attribute2, multiplicity (1,1), 'Bool) ,
                  operation (public, 'getSample,nullParam,'null )
                  operation (public, 'addSample,nullParam,'null ))
        ...
        class ( 'Display, attribute (public, 'attribute1, multiplicity (1,1), 'Integer)
                  attribute (public, 'attribute2, multiplicity (1,1), 'Bool) ,
                  operation (public, 'On,nullParam,'null )
                  operation (public, 'Off,nullParam,'null )) .


  *** ARBOL DE HERENCIA PARA LA CLASE UserMode
  eq tree1 = 'UserMode [
                'MenuUserMode [empty-forest] inf empty-constr fin :
                'SettingTimeUserMode [empty-forest] inf empty-constr fin :
                'SettingDateUserMode [empty-forest] inf empty-constr fin :
                 empty-forest

       ] inf ( Over  nullLabelList
         Disj ( discr nullDiscrName has
                  'MenuUserMode 'SettingTimeUserMode 'SettingDateUserMod classes ) Comp true )  fin .

  *** ARBOL DE HERENCIA PARA LA CLASE View
  eq tree2 = 'View [
        ...
       ] inf ( Over  nullLabelList
         Disj ( discr nullDiscrName has  'ClockView 'TaskView 'MenuView classes ) Comp true )  fin .


  ...
  *** LISTA DE ARBOLES DE HERENCIA
  eq ctList = addTreeList (tree3, addTreeList (tree2, addTreeList( tree1, nullClassTreeList, classList ), classList),
                    classList) .

  *** LISTA DE ASOCIACIONES
  eq assocList =
         association ('AudioBlock-Message,
       assocEnd('AudioBlock, 'isPartOf, multiplicity (1, 1), nullAttr, none, none, unOrdered, public, 'Class),
       assocEnd('Message, 'isSequenceOf, multiplicity (0, m), nullAttr, none, none, unOrdered, public, 'Class),
       nullAttr)
```

Figura 5. Especificación algebraica del diagrama de ejemplo (tomada de [14])

Una vez se cuenta con la especificación algebraica, el siguiente paso consiste en la propia formalización de las métricas. En el caso que nos ocupa, la formalización de la métrica DIT que se muestra en la Figura 6.

```
...
op DIT : ClassDiagram -> Int .
op profundidadIT : ClassTreeList → Int .
...
eq DIT (classDiagram(CList, CTreeList, AList, OList, LList) =
            profundadIT(CTreeList) .
eq profundidadIT(CTreeList) = depth-forest(CTreeList) .
...
```

Figura 6. Formalización de la métrica DIT

Para el ejemplo que se muestra en la Figura 4, el valor de la métrica DIT sería 2. Como vemos, gracias a la potencia de los operadores con que se definió la herencia en las especificaciones Maude, el número de líneas de esta formalización es mucho menor que la realizada con OCL.

4. Comparación del uso de OCL y Maude para la formalización de métricas

Las principales diferencias entre ambas propuestas vienen motivadas por los fundamentos en los que se basa cada uno de estos lenguajes. Aunque OCL es un lenguaje considerado como formal, éste en realidad no lo es, al menos no al nivel que Maude, ya que éste es un lenguaje con sólidas bases matemáticas que afectan a todo el lenguaje. Desde este punto de vista, OCL ha sido considerado siempre como un lenguaje más cercano a los diseñadores de software, lo cual hace que sea más fácil, a priori, de entender por cualquier persona que esté familiarizada con el modelado conceptual orientado a objetos. Ahora bien, OCL no es un lenguaje completamente formal, por lo que sigue produciendo ambigüedades que no se presentan en las especificaciones creadas con Maude, que es un lenguaje mucho más formal y con un contenido matemático que se refleja en la sintaxis de las especificaciones algebraicas que se realizan con él.

A favor de OCL debemos destacar que es un lenguaje estándar, propuesto por el OMG y que se apoya en el uso de otro lenguaje estándar, tan ampliamente reconocido como UML.

Por su parte, Maude ofrece una potencia de razonamiento sobre las especificaciones que no ofrece OCL, permitiendo así la posibilidad de identificar transformaciones de interés a aplicar sobre un modelo o comparar varias especificaciones para buscar equivalencias semánticas entre ellas.

Así, podríamos concluir que leer expresiones OCL es más sencillo que leer especificaciones algebraicas Maude debido a la que este lenguaje requiere conocimientos que no suelen ser habituales entre los modeladores de software, pero implica la asunción de posibles ambigüedades en las expresiones que se construyan utilizando OCL.

La última diferencia importante entre ambas propuestas viene dada por su capacidad de ejecución. Maude es un lenguaje que permite ejecutar sus especificaciones, lo que posibilita la ejecución de métricas sobre modelos concretos. Mientras que, aunque existen herramientas que permiten ejecutar restricciones OCL sobre modelos, éstas no han alcanzado el nivel de desarrollo que posee el propio lenguaje Maude.

5. Conclusiones

En el campo de la medición de software, el proceso de formalización de las métricas ha sido uno de los que menos en profundidad se ha tratado y aunque ha habido un gran número de propuestas de métricas para modelado de software orientado a objetos muchos de ellos hacen que la falta de formalización se convierta en un serio problema.

En este trabajo hemos presentado dos posibles mecanismos para formalizar las métricas. Por un lado el lenguaje OCL, que se basa en la utilización de pre y post-condiciones para establecer las limitaciones de las métricas y el resultado del cálculo de las mismas respectivamente. Por otro lado, el lenguaje Maude permite, a través de especificaciones algebraicas, representar los distintos modelos que se desean medir y posteriormente calcular de manera automática los valores de las distintas métricas que se hayan definido.

Teniendo en cuenta las ventajas e incovenientes de cada propuesta, en nuestra opinión, leer y trabajar con expresiones OCL puede resultar más sencillo que hacerlo con especificaciones algebraicas Maude, ya que este lenguaje requiere conocimientos que no suelen ser habituales entre los modeladores de software, pero implica la asunción de posibles ambigüedades en las expresiones que se construyan utilizando OCL.

Además, Maude permite identificar transformaciones de interés a aplicar sobre un modelo o comparar varias especificaciones para buscar equivalencias semánticas entre ellas y es más fácilmente automatizable.

Agradecimientos

Este trabajo es parte del Proyecto ENIGMAS (PBI-05-058) financiado por la Consejería de Ciencia y Tecnología de la Junta de Comunidades de Castilla-La Mancha y de los subproyectos DYNAMICA/CALIPO y DYNAMICA/PRESSURE, financiados por la Dirección General de Investigación del Ministerio de Ciencia y Tecnología (TIC2003-07804-C05-03).

Referencias

[1] Baroni, A. L.: Using OCL to Formalize Object-Oriented Design Metrics Definitions. Proc. of ECOOP'02 Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE 2002). Malaga, Spain (2002)

[2] Baroni, A. L., Brito e Abreu, F.: Formalizing Object-Oriented Design Metrics upon the UML Meta-Model. Proc. of Brazilian Symposium on Software Engineering (BSSE 2002). Gramado, Brazil (2002)

[3] Baroni, A. L.: Evaluating Design Properties of UML Behavioral Models. Proc. of Generative and Tranformationl Techniques on Software Engineering Summer School (GTTSE 2005). Braga, Portugal (2005)

[4] Baroni, A. L.: Quantitative Assessment of UML Dynamic Models. Proc. of Doctoral Symposium of the European Software Engineering Conference (ESEC) and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE) (ESEC/FSE 2005). Lisbon, Portugal (2005)

[5] Calero, C., Piattini, M., Genero, M.: Method for Obtaining Correct Metrics. Proc. of 3rd International Conference on Enterprise and Information Systems (ICEIS 2001). Setúbal, Portugal (2001) 779-784

[6] Chidamber, S. R., Kemerer, C. F.: A Metrics Suite for Object-Oriented Design. IEEE Transactions on Software Engineering. 20 (6) (1994) pp. 476-493

[7] Clavel, M., Durán, F., Eker, S., Lincoln, P., Marínez-Oliet, N., Meseguer, J., Talcott, C.: Maude 2.1.1 Manual. University of Illinois at Urbana-Champaign (2003)

[8] Cruz-Lemus, J. A., Genero, M., Piattini, M.: Metrics for UML Statechart Diagrams. In: Genero, M., Piattini, M., Calero, C. (eds.): Metrics for Software Conceptual Models. Imperial College Press, United Kingdom (2005)

[9] Fernández, J. L., Toval, A.: Can Intuition become Rigurous? Foundations for UML Model Verification Tools. Proc. of 4th IEEE International Symposium on Software Reliability Engineering (ISSRE 2000). San José, CA, USA (2000)

[10] Fernández, J. L., Toval, A.: Improving System Reliability via Rigorous Software Modeling: The UML Case. Proc. of 2001 IEEE Aerospace Conference (track 10: Software and Computing). Montana, USA (2001)

[11] Lilius, J., Porres, I.: Digital Sound Recorder: A Case Study on Designing Embedded Systems Using the UML Notation. Deparment of Computer Science, University of Abo, Finland (1999)

[12] Lucas, F. J., Toval, A.: Formal Verification of Properties in the UML Collaboration Diagram. Proc. of 17th International Conference Software & Systems Engineering and their Applications (ICSSEA 2004): 3rd Workshop on System Testing and Validation. Paris, France (2004)

[13] Martí-Oliet, N., Meseguer, J.: Rewriting logic: roadmap and bibliography. Theoretical Computer Science. 285 (2) (2002) pp. 21-154

[14] Molina, F., Sáez, J., Toval, A.: Traducción del Diagrama de Clases de UML a Maude. Departamento de Lenguajes y Sistemas, Universidad de Murcia (2004)

[15] OMG: OCL 2.0 Specification. Object Management Group (ptc/2005-06-06) (2005)

[16] Vigueras, C., Genero, M., Fernández, J. L., Cruz-Lemus, J. A., Toval, A., Piattini, M.: Una Aproximación Formal a las Métricas para Diagramas de Estados UML. Proc. of II Jornadas de Trabajo DYNAMICA. Málaga (2004)

[17] Whittle, J., Araújo, J., Toval, A., Fernández, J. L.: Rigorously Automating Transformations of UML Behavior Models. Proc. of Dynamic Behaviour in UML Models: Semantic Questions in conjunction with UML 2000. York, UK (2000)