Aplicación del patrón de transformación de síntesis para la comparación de los lenguajes ATL vs. QVT

Ana Karen Vega Maqueda, S. Gustavo Peláez Camarena, Ulises Juárez Martínez, Ma. Antonieta Abud Figueroa y Luis Ángel Reyes Hernández

Instituto Tecnológico de Orizaba, Departamento de Posgrado e Investigación, Orizaba, Veracruz, México

karen.vemaqueda@gmail.com, sgpelaez@yahoo.com.mx, ujuarez71@gmail.com, aabud@prodigy.net.mx, l_r_h01@hotmail.com

Resumen. La ingeniería dirigida por modelos (MDE) se caracteriza por asignar a los modelos el papel principal durante todas las etapas de desarrollo de software, aumentar la automatización en el proceso de desarrollo separando los aspectos de tecnología y promoviendo la productividad y mejora de calidad de los sistemas. El enfoque MDE ha surgido como un nuevo paso en el camino hacia una verdadera industrialización de la producción de software. Tras el éxito del paradigma orientado a objetos, el uso sistemático de modelos se presenta ahora como la forma apropiada para conseguir programar con un nivel más alto de abstracción y de aumentar el nivel de automatización. Se presenta un análisis de los lenguajes de transformación de modelos ATL y QVT donde se mencionan las características más relevantes de cada uno de los lenguajes y un caso de estudio.

Palabras clave: MDE, ATL, QVT, EMF, XMI.

1. Introducción

En la actualidad los modelos son parte importante dentro de la ingeniería de software, no obstante en la mayoría de los casos se encuentran plasmados en papel en lugar de incorporarse en el proceso de ingeniería, estos son considerados como la representación exacta o abstracción de las propiedades principales de un objeto, sistema o idea.

ATL es un lenguaje de transformación de modelos, el cual en el área de ingeniería dirigida por modelos, proporciona mecanismos para producir un conjunto de modelos de destino de un conjunto de modelos de origen. El enfoque basado en modelos supone proporcionar a los diseñadores y desarrolladores de modelo un conjunto de operaciones dedicadas a la manipulación de los modelos con propósito de obtener un alto nivel abstracción [1].

El objetivo del presente artículo es identificar las características principales de los lenguajes ATL y QVT y efectuar el proceso de transformación de modelo a modelo en un caso de estudio. La estructura del trabajo se describe a continuación: Sección 2: Información sobresaliente del lenguaje de transformación de modelos. Sección 3: Se presentan las características principales de los lenguajes ATL y QVT. Sección 4: Se

presenta la propuesta. Sección 5: Se presenta el empleo de los lenguajes de transformación de modelo ATL y QVT al caso de estudio. Sección 6: Se presentan los resultados obtenidos sobre el trabajo hasta el momento.

2. Trabajos relacionados

En [2] se comparó la propuesta del lenguaje consulta/vista/transformación (QVT) y el lenguaje de transformación (ATL) con el propósito de reunir conocimientos sobre los enfoques de transformación de modelos existentes, se describió la arquitectura, características del lenguaje y se identifican las categorías a comparar en los lenguajes. La atención se centra en los principales componentes del lenguaje y cómo se relacionan, mediante el análisis de diversas categorías (abstracción, paradigma, direccionalidad, cardinalidad, etc.). Se demostró cómo el lenguaje ATL es ejecutado en los motores de QVT y recíprocamente QVT en la máquina virtual de ATL. Por lo tanto es posible tener una interoperabilidad entre los lenguajes ATL y QVT a nivel conceptual, se espera que esta investigación sea útil en el análisis de interoperabilidad para otros lenguajes de transformación.

En [3] se describió el lenguaje de transformación ATLAS (ATL), las herramientas y el conjunto de documentación con ejemplos disponibles en el subproyecto ATL Eclipse / GMT, se analizó que ATL se apoya en un conjunto de herramientas de desarrollo integradas en la parte superior del entorno de Eclipse: un compilador, una máquina virtual, un editor y un depurador. El estado actual de las herramientas de ATL ya permite resolver problemas no triviales y hasta el momento ATL se utiliza y evalúa a través de diversos sitios de índole académica e industrial.

En [4] se describió el lenguaje de transformación de modelos ATL y su entorno de ejecución basado en la infraestructura de Eclipse, este tipo de entorno desarrollo proporciona apoyo en las tareas más relevantes que intervienen en el uso de lenguaje tales como: edición, compilación, ejecución y depuración, se comprobó que el lenguaje permite la aplicación de reglas de transformación imperativas y declarativas para la solución de problemas no triviales.

En [5] se presentaron transformaciones ATL basadas en las reglas de modelos de transformación, proporcionando una codificación intuitiva y versátil de ATL en OCL (lenguaje de restricción de objeto) utilizada para el análisis de diversas propiedades con respecto a las transformaciones, también se describió cómo generar automáticamente modelos de transformación partiendo de transformaciones ATL declarativas, específicamente este trabajo se enfocó en demostrar si un modelo de salida generado por una transformación ATL será válido para cualquier modelo de entrada.

En [6] se analizaron las características de modularidad en el lenguaje de transformación de modelos ATL, se analizaron dos casos donde las unidades modulares se identifican a través de las relaciones entre metamodelos de origen y destino en base a la funcionalidad de transformación genérica, también se aplicaron 3 técnicas de transformación: reglas explicitas, implícitas y de herencia para evaluar distintas implementaciones de los casos. Se concluyó que al aplicar la regla implícita se obtuvo un bajo acoplamiento, por lo tanto esta regla debe aplicarse cuando es primordial la reutilización y la adaptabilidad además la elección de diferentes descomposiciones en

el metamodelo destino llevo a diferentes conjuntos de reglas de transformación, por lo tanto se recomienda tener en cuenta más de una descomposición en los metamodelos.

En [7] menciona la existencia de 4 patrones de transformación de modelos para lenguajes HOTS (*Higher-Order Transformations*) ATL y QVT por mencionar, el primer patrón de transformación es nombrado de Síntesis en el deben definirse los metamodelos origen, destino y el modelo origen y resultado de la transformación es el modelo destino, en el patrón de transformación de análisis no es obligatoria la generación de un modelo destino y para cada mapeo se generan posibles variabilidades del sistema a transformar, el patrón de transformación de composición se define bajo 3 condiciones: Al menos uno de los modelos de origen debe ser una transformación, como mínimo uno de los modelos de salida debe ser una transformación y finalmente los modelos de origen y/o destino deben contener más de una transformación, por último el cuarto patrón es de transformación de modificación se tiene como elemento de entrada una transformación y se genera una versión actualizada de la misma transformación.

3. Lenguaje de transformación de modelos

3.1. Lenguaje de transformación ATL

ATL (ATL Lenguaje de Transformación) es un lenguaje de transformación de modelos y un conjunto de herramientas para el proceso de transformación de modelos. En el campo de la ingeniería dirigida por modelos (MDE), ATL proporciona formas para producir un modelo o un conjunto de modelos destino a partir del modelo o conjunto de modelos origen.

La sintaxis abstracta de ATL es especificada mediante un metamodelo MOF, y provee de lenguajes en modo textual y gráfico para representar las reglas de transformación del lenguaje [2], se muestra la figura 1.

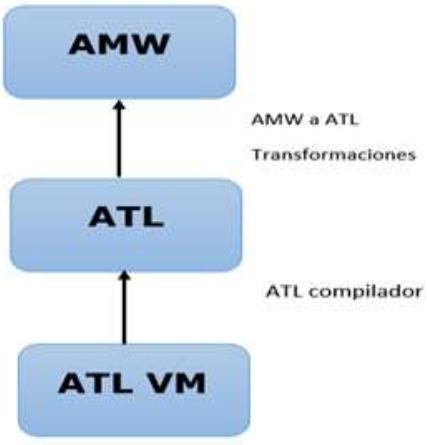

Fig. 1. Arquitectura del lenguaje.

Aquí la primera capa ATL VM significa máquina virtual de ATL, lenguaje ATL, AMW se refiere al modelo vista de ATL. Los programas ATL compilados son ejecutados mediante la ATL VM, que utiliza un conjunto de instrucciones orientadas al modelo. AMW (*ATLAS Model Weaver*) utilizado para establecer y representar las relaciones entre distintos modelos [8].

3.2. Lenguaje de transformación QVT

El lenguaje QVT (*Query, Views and Transformation*) es un estándar propuesto por la OMG (*Object Management Group*) para la definición del proceso de transformación de modelo a modelo, basado en el estándar MOF (Meta Object Facility) para la descripción de la estructura y sintaxis de los metamodelos.

En [9] menciona 2 tipos de transformaciones:

Relaciones: especificaciones de transformaciones multidireccionales. No son ejecutables en el sentido de que son incapaces de crear o modificar un modelo. Permiten comprobar la consistencia entre dos o más modelos relacionados. Se utilizan normalmente en la especificación del desarrollo de un sistema o para comprobar la validez de un mapeo.

Mapeo: implementaciones de transformaciones. A diferencia de las relaciones, los mapeos son unidireccionales y pueden devolver valores. Un mapeo puede refinar una o varias relaciones, en cuyo caso el mapeo debe ser consistente con las relaciones que refina. A continuación se muestra la fig. 2 [2].

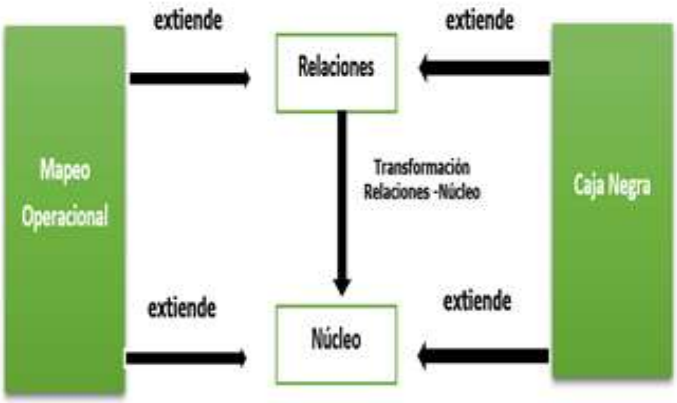

Fig. 2. Arquitectura del lenguaje QVT.

4. Propuesta

El objetivo es mostrar una transformación de modelo a modelo empleando el patrón de transformación de síntesis y utilizando los lenguajes de transformación de modelos ATL y QVT, se propone para ello un caso de estudio definido en el ambiente de desarrollo basado en eclipse (versión luna 4.4), recomendada por la comunidad de

eclipse para el desarrollo software dirigido por modelos, para esto es necesario la instalación de los plugin EMF (*Eclipse Modeling Framework*) para definir los modelos y metamodelos ECORE basados en el lenguaje XMI (*XML Metadata Interchange*), opción ATL en el framework, para la definición de las reglas de transformación en el lenguaje ATL y Model to Model Transformation para declarar las reglas de trasformación en QVT.

5. Justificación

El proceso de transformación de modelos basado en el campo laboral beneficia en la reducción de tiempo y costo durante el desarrollo de una aplicación, obteniendo como resultado una aplicación con un nivel de calidad sólido, por tal razón se pretende el promover la incorporación del enfoque en el proceso de ingeniería viendo más allá del modelado.

6. Aplicación del lenguaje de transformación de modelos ATL vs. QVT al caso de estudio

El caso de estudio basado en el patrón descrito, consiste en transformar un modelo que representa a una Agenda con el objetivo de obtener el modelo Cita, se define el modelo origen (modeloAgenda), el metamodelo origen (MMAgenda) y el meta-modelo destino (MMCita) ambos metamodelos son instancia del meta-metamodelo ECORE utilizado para la definición de metamodelos en Eclipse con el plugin EMF, el bloque llamado Agenda2Cita contiene las declaraciones y reglas para realizar el proceso de transformación ya sea en ATL y QVT, la siguiente figura muestra el esquema de transformación de modelo a modelo para el caso mencionado.

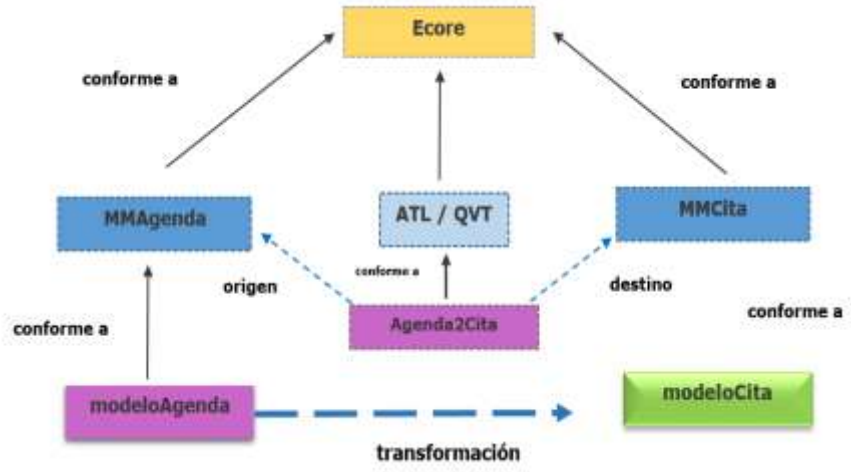

Fig. 3. Esquema de transformación de modelo a modelo.

A continuación se muestra en la figura 4 y 5 los elementos que componen al metamodelo Agenda y metamodelo Cita.

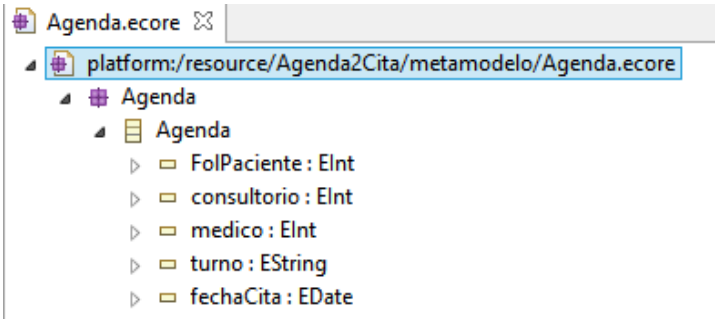

Fig. 4. Metamodelo Agenda.

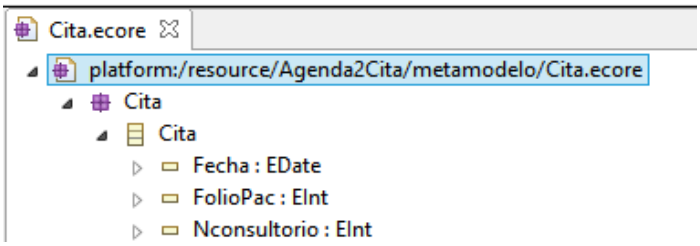

Fig. 5. Metamodelo Cita.

A partir de los metamodelos definidos se genera automáticamente código ECORE en formato XMI especificación utilizada para la definición y manipulación de metamodelos e intercambio de diagramas en UML. En el caso de la definición del modelo origen (modeloAgenda) se realiza de forma manual y de igual manera en formato XMI como se muestra en la siguiente figura.

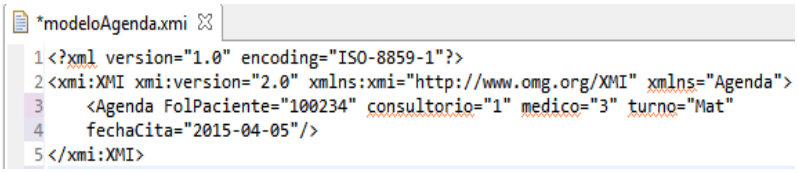

Fig. 6. Modelo Agenda.

El módulo en ATL permite la transformación entre modelos, declarando para ello las reglas de transformación basadas en sintaxis OCL (Object Constraint Language), como se observa en la siguiente figura en la línea 4 se define el nombre del módulo, posteriormente en la línea 7 se define el helper el cual corresponde a un método conforme a el paradigma orientada a objetos, este helper recupera un tipo de dato booleano para identificar si el elemento fechaCita de la clase Agenda se encuentra

definido, posteriormente en línea 14 se declara la regla para escribir los datos tales como fecha, folioPac y Nconsultorio en el modelo destino (modeloCita).

Agenda2Cita.atl

```
1  -- @path Agenda=/Agenda2Cita/Agenda.ecore
2  -- @path Cita=/Agenda2Cita/Cita.ecore
3
4  module Agenda2Cita;
5  create OUT : Cita from IN : Agenda;
6
7  helper context Agenda!Agenda def : getFecha() : Boolean =
8      if not self.fechaCita.oclIsUndefined() then
9          true
10     else
11         false
12     endif;
13
14 rule Agen2Cit {
15 from b : Agenda!Agenda (b.getFecha())
16 to
17 out : Cita!Cita (
18             Fecha <- b.fechaCita,
19             FolioPac <-b.FolPaciente,
20             Nconsultorio <- b.consultorio
21         )
22 }
```

Fig. 7. Módulo ATL.

El resultado de la transformación se muestra en la siguiente figura, dónde se obtuvo el modelo destino (modeloCita) con los datos recuperados como se estableció en la regla en el módulo ATL, generando un archivo en formato XMI.

*modeloCita.xmi

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns="Cita">
3   <Cita Fecha="2015-04-05T00:00:00.000-0600" FolioPac="100234" Nconsultorio="1"/>
4 </xmi:XMI>
```

Fig. 8. Modelo Cita.

Es posible obtener diferentes versiones del modelo destino (modeloCita) de acuerdo a las reglas establecidas en el módulo de cada uno de los lenguajes de transformación, para el siguiente ejemplo se restringe solo aquellos elementos de la clase Agenda asignados al turno matutino, partiendo del modelo origen (modeloAgenda) como se muestra en la siguiente figura.

*modeloAgenda.xmi

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns="Agenda">
3     <Agenda FolPaciente="100234" consultorio="1" medico="3" turno="Mat" fechaCita="2015-04-05"/>
4     <Agenda FolPaciente="100235" consultorio="2" medico="4" turno="Mat" fechaCita="2015-04-06"/>
5     <Agenda FolPaciente="100236" consultorio="3" medico="2" turno="Vesp" fechaCita="2015-04-07"/>
6     <Agenda FolPaciente="100237" consultorio="2" medico="4" turno="Vesp" fechaCita="2015-04-05"/>
7 </xmi:XMI>
```

Fig. 9. Modelo Agenda.

Por tanto se define en el módulo ATL, en línea 8 la declaración de un helper con el fin de validar el elemento turno de la clase Agenda con asignación a 'Mat' refiriéndose a Matutino y posteriormente en línea 18 se declara la regla para la escritura de los atributos del modelo destino (modeloCita) y así obtener la nueva versión del modelo destino.

*Agenda2Cita.atl

```
1  -- @path Agenda=/Agenda2Cita/Agenda.ecore
2  -- @path Cita=/Agenda2Cita/Cita.ecore
3
4  module Agenda2Cita;
5  create OUT: Cita from IN: Agenda;
6  helper context Agenda!Agenda def: OnlyMatutino(): Boolean =
7      if self.turno.equals('Mat') then
8          true
9      else
10         false
11     endif;
12
13 rule Agen2Cita {
14     from
15         b: Agenda!Agenda (
16             b.OnlyMatutino()
17         )
18     to
19         out: Cita!Cita (
20             Fecha <- b.fechaCita,
21             FolioPac <- b.FolPaciente,
22             Nconsultorio <- b.consultorio
23         )
24 }
```

Fig. 10. Módulo ATL.

La nueva versión del modeloCita se muestra en la siguiente figura, teniendo en cuenta que la escritura de un modelo destino, se basa de la existencia un modelo origen (modeloAgenda) de la figura 9.

modCita.xmi

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns="Cita">
3   <Cita Fecha="2015-04-05T00:00:00.000-0600" FolioPac="100234" Nconsultorio="1"/>
4   <Cita Fecha="2015-04-06T00:00:00.000-0500" FolioPac="100235" Nconsultorio="2"/>
5 </xmi:XMI>
```

Fig. 11. Modelo Cita.

A continuación se muestra en la siguiente figura el módulo en QVT en las líneas 1 y 2 se define el nombre y dirección de los metamodelos, después en la línea 4 se define el nombre del módulo y la correspondencia con los metamodelos, en las líneas 6 a la 8 se encuentra el main donde se recupera el conjunto de objetos de tipo Agenda que corresponde a la clase del metamodelo Agenda y finalmente en las líneas 9 a 13 se define el método Agend2Cita en el cual se indica la correspondencia del metamodeloAgendas tiene una clase Agenda y el metamodeloCitas su clase Cita y se recuperan los datos tales como fechacita, folpaciente y consultorio para ser escritos en el modeloCita.

QVTAgenda2Cita.qvto

```
1  modeltype Agendas uses 'http:///Agendas.ecore';
2  modeltype Citas uses 'http:///Citas.ecore';
3
4  transformation QVTAgenda2Cita(in agendaModelo:Agendas, out citaModelo:Citas);
5
6  main() {
7  agendaModelo.objects()[Agenda]->map Agend2Cita();
8  }
9  mapping Agendas::Agenda::Agend2Cita() : Citas::Cita {
10  Fecha:= self.fechaCita;
11  FolioPac := self.FolPaciente;
12  Nconsultorio :=self.consultorio;
13 }
```

Fig. 12. Módulo QVT.

El modelo destino (modeloCita) obtenido de la transformación se visualiza en estilo de árbol de navegación, semejante al de los metamodelos definidos en ECORE, indicando cual es la instancia del modelo destino (modeloCita) en este caso el metamodelo destino (metamodeloCita) y los datos correspondientes a las reglas establecidas en dicho módulo.

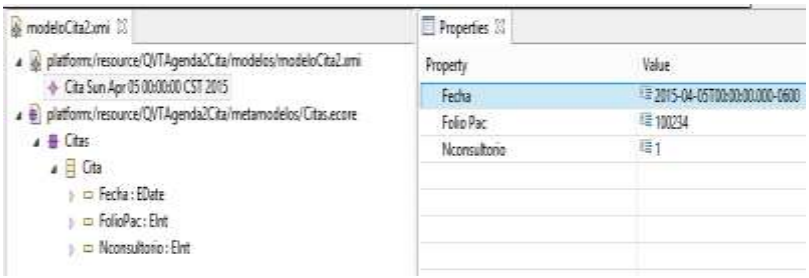

Fig. 13. Modelo Cita.

QVTAgenda2Cita.qvto

```
1  modeltype Agendas uses 'http:///Agendas.ecore';
2  modeltype Citas uses 'http:///Citas.ecore';
3
4  transformation QVTAgenda2Cita(in agendaModelo:Agendas, out citaModelo:Citas);
5
6  main()
7    {
8      agendaModelo.objects()[Agenda]->map Agend2Cita();
9    }
10 mapping Agendas::Agenda::Agend2Cita() : Citas::Cita {
11     if(self.turno.equalsIgnoreCase('Mat'))
12     {
13         Fecha:= self.fechaCita;
14         FolioPac := self.FolPaciente;
15         Nconsultorio :=self.consultorio;
16     }
17 }
```

Fig. 14. Módulo QVT.

El módulo en QVT que se muestra en la siguiente figura, es la continuidad al ejemplo planteado anteriormente, el cual requiere de la clase Agenda los elementos asignados al turno matutino, por tal razón en línea 11 se utiliza el método equalsIgnoreCase() para la comparación con la cadena 'Mat' y el contenido del elemento turno y así validar la escritura del modelo destino (modeloCita).

El nuevo modelo destino (modeloCita) contiene 4 elementos cita 2 de ellos son los que cumplieron con la restricción establecida en el módulo anterior, los 2 restantes tienen valores nulos por no cumplir con la condición definida y se puede visualizar que el modeloCita es una instancia del metamodelo destino (metamodeloCita).

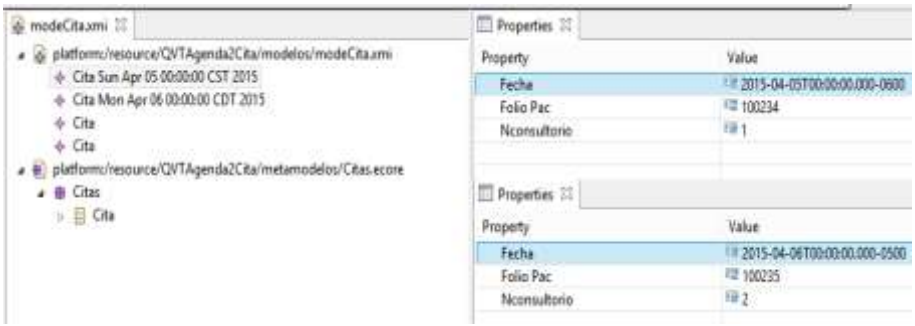

Fig. 15. Modelo Cita.

7. Conclusión

El análisis realizado revela algunas características de los lenguajes de transformación de modelos, el entorno de desarrollo para realizar la transformación de modelos, que el lenguaje ATL tiene su propia máquina virtual y compilador definidos por el lenguaje, que en QVT emplea dos maneras de realizar el proceso de transformación las cuales son; pruebas de caja negra y mapeo, la segunda se utilizó en el presente trabajo. A demás que la aplicación de los lenguajes de transformación a un caso de estudio en el campo laboral, beneficia en la reducción de tiempo y costo durante el desarrollo de una aplicación obteniendo como resultado una aplicación con un nivel de calidad solido por medio de la reutilización, también se menciona que el entorno de desarrollo en eclipse cuenta con un complemento llamado GMF el cual hoy en día resuelve problemas no triviales y una amplia documentación la cual incluye ejemplos básicos de transformación.

La mayoría de las herramientas para la transformación de modelos han sido desarrolladas como complementos del entorno de desarrollo en eclipse. Al desarrollarse sobre esta plataforma con varios años de utilización y desarrollo, se asegura cierta robustez, de manera global las herramientas hasta ahora disponibles otorgan características esenciales como el reconocimiento de la sintaxis y compilador, sin embargo aún existen funcionalidades sin implementar como por ejemplo la detección de errores en la codificación de transformación.

Finalmente en el proceso de transformación de modelo a modelo empleado al caso de estudio utilizando el patrón de transformación de síntesis, permite razonar que se obtuvo el modelo destino esperado en ambos lenguajes de transformación el cual incluyo un archivo en formato XMI, de ambos modelos destino el generado por el lenguaje QVT se obtuvo adicionalmente una vista tipo árbol de navegación, la cual facilita la visualización del resultado y especifica en la zona de espacio de nombres del archivo XMI a que metamodelo hace instancia el modelo generado ventaja no identificada en los resultados del lenguaje ATL.

Referencias

1. Trujillo, J. L., Espinoza, A. D.: Fundamental concepts of engineering headed by models and models of specific domain. Revista de Investigación de Sistemas e Informática, pp. 9–19 (2010)
2. Jouault, F., Kurtev, I.: On the Architectural Alignment of ATL and QVT. In Proceedings of the 2006 ACM symposium on Applied computing, pp. 1188–1195 (2006)
3. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I., Valduriez, P.: ATL: a QVT-like transformation language. In Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications, pp. 719–720 (2006)
4. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I.: ATL: A model transformation tool. Science of computer programming, vol. 72(1), pp. 31–39 (2008)
5. Büttner, F., Egea, M., Cabot, J., Gogolla, M.: Verification of ATL transformations using transformation models and model finders. In Formal Methods and Software Engineering, Springer Berlin Heidelberg, pp. 198–213 (2012)
6. Kurtev, I., van den Berg, K., Jouault, F.: Rule-based modularization in model transformation languages illustrated with ATL. Science of computer programming, vol. 68(3), pp. 138–154 (2007)
7. Tisi, M., Jouault, F., Fraternali, P., Ceri, S., Bézivin, J.: On the use of higher-order model transformations. In Model Driven Architecture-Foundations and Applications, Springer Berlin Heidelberg, pp. 18–33 (2009)
8. Jiménez, A., Vara, J. M., Bollati, V. A., Marcos, E.: Gestión de la trazabilidad en el desarrollo dirigido por modelos de Transformaciones de Modelos: una revisión de la literatura. In: XVI Jornadas de Ingeniería de Software y Bases de Datos–JISBD (2011)
9. Ferreira, M., García, F., Ruiz, F., Bertoa, M. F., Calero, C., Vallecillo, A., Mora, B.: Medición del software ontología y metamodelo. Departamento de Tecnologías y Sistemas de la Información, Castilla La Mancha (2006)