Implementación ATL de la HOT núcleo de la meta-herramienta

Por definición, la estructura de cualquier HOT codificada en ATL está orientada a la creación de las instancias adecuadas de las clases del meta-modelo de ATL tales que, mediante su asociación y composición así como posterior serialización, se obtenga una cierta transformación M2M implementada en ATL. En particular, la HOT de la que es objeto este trabajo tiene como misión generar una transformación de chequeo que exhiba una estructura como la mostrada en la Tabla 3-X. Por tanto, se antoja como óptima una implementación consistente en una única regla *matched* definida sobre la clase *CC_Model* y responsable de la creación, entre otras cosas, de una instancia de módulo ATL.

Tabla 3.1

Además de esta única *matched rule*, la HOT contendrá un conjunto de *called rules* que pueden ser agrupadas en cuatro bloques. La Tabla 3-X muestra el esqueleto de la HOT, con diferente sombreado para cada uno de los cuatro bloques de *called rules*[1].

Tabla 3.1

```
module CC_2_ATL;
create OUTmodel : ATL from INmodel : CC;
-- ********************** MATCHED RULES
rule CC_Model_2_Module {
   from
      input : CC!CC_Model
   to
       output1 : ATL!Module (...),
       output2 : ATL!OclModel (...), -- input model
       output3 : ATL!OclModel (...), -- output model
       output4 : ATL!OclModel (...), -- DomainMM
       output5 : ATL!OclModel (...), -- TargetMM (CVD)
       output51 : ATL!OclModel (...) -- Ecore
       output6 : ATL!CalledRule (...), -- entrypoint rule Create_CVDmodel
       output10 : ATL!OutPattern(...),
       output11 : ATL!SimpleOutPatternElement (...),
       output12 : ATL!OclModelElement (...),
       output15 : ATL!ActionBlock (...)
   do {
       -- ---------------------------------
       -- CALLED RULES section of the module
       -- ---------------------------------
       thisModule.CreateRule_CVD(...);
```

[1] Aunque no presentes en la Tabla 3-X, existe un cierto número de *helpers* auxiliares que también participan en el código de la HOT. La razón para soslayarlos es que, aunque necesarios para extraer la clase contenedor principal del meta-modelo de dominio, como *helpers* que son, no crean instancias de las clases del meta-modelo de ATL.

```
        thisModule.CreateRule_BasedOnPropertiesOfContextClass(...);
        ...
        -- -------------------------------
        -- ATTRIBUTES section of the module
        -- -------------------------------
        for (class in domainMM_Classes) {
           thisModule.CreateAttrHelper_ForEClass(...);
        }
        for (class in domainMM_Classes) {
           for (eAttr in class.eAttributes) {
               thisModule.CreateAttrHelper_ForEAttribute(...);
           }
        }
        for (class in domainMM_Classes) {
           for (eRef in class.eReferences) {
              thisModule.CreateAttrHelper_ForEReference(...);
           }
        }
        -- -----------------------------
        -- HELPERS section of the module
        -- -----------------------------
        for (cc in input.constraintCharacterizations) {
           thisModule.CreateCheckerHelper(...);
        }
        -- ------------------------------------------------
        -- using{...} block of the entrypoint rule "output6"
        -- ------------------------------------------------
        for (contextClass in domainMM_ContextClasses) {
           thisModule.CreateLocalSeq(...);
        }
        -- ---------------------------------------------------------------------
        -- statements for the do{...} block "output15" of the entrypoint rule
        -- ---------------------------------------------------------------------
        for (contextClass in domainMM_ContextClasses) {
           thisModule.CreateCheckerLoop(...);
        }
    }
}

-- ********************** CALLED RULES

-- ------------------------
-- Creation of called rules
-- ------------------------
rule CreateRule_CVD(modul :  ATL!Module,
                    diagnosticMM : ATL!OclModel,
                    ecoreMMM : ATL!OclModel) {
   ...
}
rule CreateRule_BasedOnPropertiesOfContextClass(...) {...}

...

-- -----------------------------
-- Creation of helper ATTRIBUTES
-- -----------------------------
rule CreateAttrHelper_ForEClass (modul :  ATL!Module,
                                 eClass : Ecore!EClass,
                                 domainMM : ATL!OclModel,
                                 ecoreMMM : ATL!OclModel){
```

```
   ...
}
rule CreateAttrHelper_ForEAttribute (modul :  ATL!Module,
                                     eAttr : Ecore!EAttribute,
                                     domainMM : ATL!OclModel,
                                     ecoreMMM : ATL!OclModel){
   ...
}
rule CreateAttrHelper_ForEReference (modul :  ATL!Module,
                                     eRef : Ecore!EReference,
                                     domainMM : ATL!OclModel,
                                     ecoreMMM : ATL!OclModel){
   ...
}
-- ------------------------------
-- Creation of functional HELPERS (for checking)
-- ------------------------------
rule CreateCheckerHelper (modul :  ATL!Module,
                          cc : Constraints!CC,
                          domainMM : ATL!OclModel){
   ...
}
```

```
-- --------------------------------
-- Creation of rule local variables
-- --------------------------------
rule CreateLocalSeq(scopeRule : ATL!MatchedRule,
                    domainMM : ATL!OclModel,
                    seqType : String) {
   ...
}
-- ----------------------------------------
-- Creation of loops for constraints checking
-- ----------------------------------------
rule CreateCheckerLoop(doBlock : ATL!ActionBlock,
                       localSeq : ATL!RuleVariableDeclaration,
                       ccs : Sequence(Constraints!CC),
                       outPatternElem : ATL!OutPatternElement) {
   ...
}
rule CreateConstraintChecker(loop : ATL!ForStat,
                             cc : Constraints!ConstraintCharacterization,
                             outPatternElem : ATL!OutPatternElement,
                             iter : ATL!Iterator) {
   ...
}
```

```
-- --------------------------------
-- Rules for completing invocations
-- --------------------------------
rule CompleteRuleInvocation_MultiplicityRestriction(invocation : ATL!OperationCallExp
                                          cc : Constraints!MultiplicityRestriction) {
   ...
}
rule CompleteRuleInvocation_AttributeValueValidity(invocation : ATL!OperationCallExp,
                                           cc : Constraints!AttributeValueValidity) {
   ...
}
...
```

En primer lugar, el código de la HOT incluye una única regla *matched* (*CC_Model_2_Module*). Su misión es crear el módulo ATL de la transformación de chequeo a generar. Esta regla *matched* toma como entrada una instancia *CC_Model* y genera en primer lugar una instancia *Module*, junto con sus elementos típicos (modelos de entrada y de salida así como meta-modelos origen y destino) que deben aparecer en cualquier transformación de chequeo (en realidad en cualquier tipo de transformación M2M implementada mediante ATL). También se crea para el módulo una instancia de *CalledRule* que represente a la regla *entrypoint* propia de una transformación de chequeo, junto a su patrón de salida obligatorio. Puesto que en esa regla *entrypoint* es también fija la presencia de una sección imperativa, también se instancia la clase del meta-modelo de ATL apropiada (*ActionBlock*).

El bloque imperativo de esta única regla *matched* de la HOT comienza dedicado a la creación de las restantes partes del módulo ATL, esto es, las secciones de reglas *called*, de atributos y de *helpers* funcionales – o bloques IV, I, y II tal y como se ha expuesto en la sección 3.3.2 –. Esto se consigue invocando las reglas *called* apropiadas, que serán detalladas posteriormente:

- El conjunto de reglas *CreateRule_XXX*
- *CreateAttrHelper_ForEClass,*
  CreateAttrHelper_ForEAttribute,
  CreateAttrHelper_ForEReference
- *CreateCheckerHelper*

La sección imperativa continua con la creación de aquellas partes de la regla *entrypoint* de la transformación de chequeo que han de estar presentes pero a priori se desconoce en qué cantidad, como por ejemplo las variables locales de tipo secuencia dentro de la sección *using* así como los bucles para chequeo de restricciones dentro de la sección *do*. Esta creación se lleva a cabo invocando a las reglas *called* codificadas con ese propósito: (*CreateLocalSeq* y *CreateCheckerLoop*, que a su vez llama a la regla *CreateConstraintChecker*), que conforman el bloque III de reglas *called* y se describen en mayor detalle posteriormente.

A continuación se expone una descripción detallada de los bloques *called*.

- Bloque I. Se trata de la sección del código de la HOT que engloba las reglas *called* encargadas de crear el bloque IV de una transformación de chequeo (las reglas *called* encargadas de crear instancias de clases *CVD*).
- Bloque II. Sección de código que engloba las reglas *called* encargadas de crear los bloques I y II de una transformación de chequeo (los atributos y los *helpers* de testeo).[2]
  - *CreateAttrHelper_ForEClass* sirve para crear las declaraciones de atributos propias de una transformación de chequeo correspondientes a las clases del meta-modelo de dominio. Se generan fragmentos de código según el patrón:

[2] Notar que, por simplicidad, se generan declaraciones de atributos correspondientes a todas las clases, a todos los atributos y a todas las referencias pertenecientes al meta-modelo de dominio.

```
helper def : classJ : Ecore!EClass = MyMM!ClassJ;
```

- *CreateAttrHelper_ForEAttribute* y *CreateAttrHelper_ForEReference* sirven para crear las declaraciones de atributos propias de una transformación de chequeo correspondientes a las propiedades de las clases del meta-modelo de dominio. Se generan fragmentos de código según los patrones:

```
helper def : ClassJ_Attr1 : Ecore!EAttribute =
   thisModule.ClassJ.eAttributes -> select(a | a.name = 'Attr1') -> first();

helper def : ClassJ_Ref1 : Ecore!EReference =
   thisModule.ClassJ.eReferences -> select(r | r.name = 'Ref1') -> first();
```

- *CreateCheckerHelper* sirve para crear las definiciones de *helpers* propias de una transformación de chequeo, correspondientes a las restricciones impuestas sobre el meta-modelo de dominio. Se generan fragmentos de código según el patrón:

```
helper context DomainMM!ClassJ def : constraint_JK() : Boolean =
                                                    constraint_JK_OCLexpr;
```

Estas reglas *called* de los bloques I y II son invocadas desde la sección *do* de la regla *matched* de la HOT y las salidas producidas se incorporan a la instancia de módulo ATL creada en primer lugar.

- Bloque III. Sección de código que engloba las reglas *called* encargadas de completar la regla *entrypoint* de la transformación de chequeo respecto a aquellas construcciones que han de aparecer en ella en un número variable.

  - *CreateLocalSeq* sirve para crear las variables locales de tipo secuencia dentro de la sección *using* de la regla *entrypoint*. Se generan fragmentos de código según el patrón:

```
ClassJ_Seq : Sequence(DomainMM!ClassJ) = DomainMM!ClassJ.allInstances();
```

  - *CreateCheckerLoop* sirve para crear los bucles requeridos para el testeo de restricciones dentro de la sección *do* de la regla *entrypoint*. Se generan, en colaboración con la regla *CreateConstraintChecker*, fragmentos de código según el patrón:

```
for (c in ClassJ_Seq) {
   if (not c.constraint_JK())
      thisModule.CalledRuleName(...);
}
```

    En realidad, *CreateCheckerLoop* por sí misma sólo genera

```
for (c in ClassJ_Seq) {
   if (...
```

    Invocando a *CreateConstraintChecker* para el resto.

  - *CreateConstraintChecker* sirve para crear el código del cuerpo de la estructura condicional *if* en la que el *helper* correspondiente es testeado, así como la subsiguiente invocación de la regla *called* para la instanciación de una clase *CVD* si el *helper* evalúa a falso, es decir, si la

restricción asociada se incumple. Puesto que el número y tipo de parámetros de las reglas *called* invocadas son variables, en este punto sólo puede ser producido el código correspondiente a la invocación parcial, basada en los primeros cinco parámetros, que son comunes a todas las reglas. Por tanto, se generan fragmentos de código según el patrón:

```
if (not c.constraint_JK())
   thisModule.CalledRuleName(output,
                             'constraint_JK',
                             '',
                             #Severity,
                              e,
                              ...);
```

- Bloque IV. Sección de código que engloba las reglas *called* cuya misión es completar la construcción del código de invocación de la regla *called* correspondiente comenzado en la regla *CreateConstraintCheker*. Esto implica básicamente añadir a la invocación los restantes parámetros según el tipo de instancia *CVD* que sea generada por la regla invocada.

```
-- @path Family=/Diagnostics2/MMs/Family.ecore
-- @path Diagnostic=/Diagnostics2/MMs/Diagnostic.ecore
-- @nsURI Ecore=http://www.eclipse.org/emf/2002/Ecore

-- Header section
module Family_2_Diagnostic;
create OUTmodel : Diagnostic from INmodel : Family;


-- ********************** ATTRIBUTES
****************************************

helper def : family : Ecore!EClass = Family!Family;
helper def : member : Ecore!EClass = Family!Member;
helper def : parent : Ecore!EClass = Family!Parent;
helper def : child : Ecore!EClass = Family!Child;
helper def : pet : Ecore!EClass = Family!Pet;

helper def : familyName : Ecore!EAttribute = thisModule.family.eAttributes ->
select(a | a.name =

'Family_Name') -> first();
helper def : name : Ecore!EAttribute = thisModule.member.eAttributes -> select(a
| a.name =

'Name') -> first();
helper def : age : Ecore!EAttribute = thisModule.member.eAttributes -> select(a
| a.name = 'Age')

-> first();
helper def : numYearsMarried : Ecore!EAttribute = thisModule.parent.eAttributes
-> select(a |
                                                    a.name =
'Num_Years_Married') -> first();
helper def : numYearsAtSchool : Ecore!EAttribute = thisModule.child.eAttributes
-> select(a |
                                                  a.name =
'Num_Years_At_School') -> first();

helper def : members : Ecore!EReference = thisModule.family.eReferences ->
select(r | r.name =

'Members') -> first();
helper def : pets : Ecore!EReference = thisModule.member.eReferences -> select(r
| r.name =

'Pets') -> first();
helper def : children : Ecore!EReference = thisModule.parent.eReferences ->
select(r | r.name =

'Children') -> first();
helper def : parents : Ecore!EReference = thisModule.child.eReferences ->
select(r | r.name =

'Parents') -> first();
```

```
helper def : owner : Ecore!EReference = thisModule.pet.eReferences -> select(r |
r.name = 'Owner')

-> first();


-- ************************ HELPERS ****************************************

-- Family

---
helper context Family!Family def : maxTwoParents() : Boolean =
      Family!Parent.allInstances() -> size() <
3;


-- Parent

---
helper context Family!Parent def : numYearsMarriedLowerThanAge() : Boolean =
      self.Num_Years_Married < self.Age;


-- Child

---
helper context Family!Child def : notSharesName() : Boolean =
      not Family!Child.allInstances() -> exists(ch | ch <> self and ch.Name =
self.Name);

---
helper context Family!Child def : numYearsAtSchoolLowerThan11() : Boolean =
      self.Num_Years_At_School <= 10;

---
helper context Family!Child def : isUsed() : Boolean =
      Family!Parent.allInstances() -> notEmpty() and
Family!Parent.allInstances() -> exists (p | p.Children -> includes(self));


-- Pet

---
helper context Family!Pet def : ownerNotPet() : Boolean =
      self.Owner.oclIsTypeOf(Family!Parent) or
self.Owner.oclIsTypeOf(Family!Child);


-- ************************ MATCHED RULES ***********************************

---
rule Family {
      from
            input : Family!Family
      using {
```

```
rule CreateCheckerHelper (
      modul :  ATL!Module,
      constraint : Constraints!Constraint,
      sourceMM : ATL!OclModel)
```

```
rule CreateLocalSeq(
      scopeRule : ATL!MatchedRule,
      sourceMM : ATL!OclModel,
      seqType : String)
```

One per constraints context

```
Family_Seq : Sequence(Family!Family) =
Family!Family.allInstances();
Parent_Seq : Sequence(Family!Parent) =
Family!Parent.allInstances();
Child_Seq : Sequence(Family!Child) = Family!Child.allInstances();
Pet_Seq : Sequence(Family!Pet) = Family!Pet.allInstances();
}
to
output : Diagnostic!Incoherences_Model (
--     attrs.
Model <- input
--     refs.
--                 Problems <-
)
do {
-- Constraints checking for Family
```

One per constraints context / local seq

```
rule CreateCheckerLoop(
      doBlock : ATL!ActionBlock,
      localSeq : ATL!RuleVariableDeclaration,
      contextConstraintsSeq : Sequence(Constraints!Constraint),
      outPatternElem : ATL!OutPatternElement)
```

One per constraint

```
rule CreateConstraintChecker(
      loop : ATL!ForStat,
      constraint : Constraints!Constraint,
      outPatternElem : ATL!OutPatternElement,
      iter : ATL!Iterator)
```

```
for (f in Family_Seq) {
      if (not f.maxTwoParents())
            thisModule.NumOfChildrenOutOfRange(output,
                  f,
                  #Error,
                  parentSeq ->
                  size().toStri
                  ng(),
                  thisModule.pa
```

```
rule CompleteRuleInvocation_NumOfChildrenOutOfRange(
      invocation : ATL!OperationCallExp,
      potentialProblem : Constraints!Num_Of_Children_Out_Of_Range,
      iter : ATL!Iterator)
```

```
rent,
'0',
'2');
}

-- Constraints checking for Parent
for (p in parentSeq) {
      if (not p.numYearsMarriedLowerThanAge())
            thisModule.WrongRelationalOrder(output,
                  p,
                  #Error,

                  thisModule
                  .numYearsM
                  arried,
```

```
rule CompleteRuleInvocation_WrongRelationalOrder(
      invocation : ATL!OperationCallExp,
      potentialProblem : Constraints!Wrong_Relational_Order,
      iter : ATL!Iterator)
```

```
thisModule.age,
#Lower);
}
```

```
		-- Constraints checking for Child
		for (c in childSeq) {
			if (not c.notSharesName())
				thisModule.ReplicatedID(output,
c,
#Error,

'familiy
children');

			if (not c.numYearsAtSchoolLowerThan11())
				thisModule.ValueOutOfRange(output,
c,
#Warning,
thisModule.numYearsAt
School,
'0',
'10');

			if (not c.isUsed())
				thisModule.UnusedElem(output,
c,
#Warning);
		}

		-- Constraints checking for Pet
		for (p in petSeq) {
			if (not p.ownerNotPet())
				thisModule.InvalidAssoc(output,
p,
#Error,

thisModule.owner,
p.Owner);

		}
	}
}
```

```
rule CompleteRuleInvocation_ReplicatedID(
	invocation : ATL!OperationCallExp,
	potentialProblem : Constraints!Replicated_Identifier)
```

```
rule CompleteRuleInvocation_ValueOutOfRange(
	invocation : ATL!OperationCallExp,
	potentialProblem : Constraints!Value_Out_Of_Range,
	iter : ATL!Iterator)
```

```
rule CompleteRuleInvocation_InvalidAssoc(
	invocation : ATL!OperationCallExp,
	potentialProblem : Constraints!Invalid_Association,
	iter : ATL!Iterator)
```

```
rule CreateLocalSeq(
      scopeRule : ATL!MatchedRule,
      sourceMM : ATL!OclModel,
      seqType : String)
```

```
rule CreateLocalSeq(
      scopeRule : ATL!MatchedRule,
      sourceMM : ATL!OclModel,
      seqType : String)
```

```
rule CreateCheckerLoop(
      doBlock : ATL!ActionBlock,
      localSeq : ATL!RuleVariableDeclaration,
      contextConstraintsSeq : Sequence(Constraints!Constraint),
      outPatternElem : ATL!SimpleOutPatternElement)
```

```
rule CreateCheckerLoop(
      doBlock : ATL!ActionBlock,
      localSeq : ATL!RuleVariableDeclaration,
      contextConstraintsSeq : Sequence(Constraints!Constraint),
      outPatternElem : ATL!OutPatternElement)
```

```
rule CreateConstraintChecker(
      loop : ATL!ForStat,
      constraint : Constraints!Constraint,
      iter : ATL!Iterator,
      outPatternElem : ATL!OutPatternElement)
```

```
rule CreateConstraintChecker(
      loop : ATL!ForStat,
      constraint : Constraints!Constraint,
      iter : ATL!Iterator,
      outPatternElem : ATL!OutPatternElement)
```

```
rule CompleteRuleInvocation_ReplicatedID(
      invocation : ATL!OperationCallExp,
      potentialProblem : Constraints!Replicated_Identifier)
```

```
rule CompleteRuleInvocation_ReplicatedID(
      invocation : ATL!OperationCallExp,
      potentialProblem : Constraints!Replicated_Identifier)
```

```
rule CompleteRuleInvocation_ValueOutOfRange(
      invocation : ATL!OperationCallExp,
      potentialProblem : Constraints!Value_Out_Of_Range,
      iter : ATL!Iterator)
```

```
rule CompleteRuleInvocation_ValueOutOfRange(
      invocation : ATL!OperationCallExp,
      potentialProblem : Constraints!Value_Out_Of_Range,
      iter : ATL!Iterator)
```

```
rule CompleteRuleInvocation_NumOfChildrenOutOfRange(
      invocation : ATL!OperationCallExp,
      potentialProblem : Constraints!Number_Of_Children_Out_Of_Range,
      iter : ATL!Iterator)
```

```
rule CompleteRuleInvocation_NumOfChildrenOutOfRange(
      invocation : ATL!OperationCallExp,
      potentialProblem : Constraints!Number_Of_Children_Out_Of_Range,
      iter : ATL!Iterator)
```

```
rule CompleteRuleInvocation_WrongRelationalOrder(
      invocation : ATL!OperationCallExp,
      potentialProblem : Constraints!Wrong_Relational_Order,
      iter : ATL!Iterator)
```

```
rule CompleteRuleInvocation_WrongRelationalOrder(
      invocation : ATL!OperationCallExp,
      potentialProblem : Constraints!Wrong_Relational_Order,
      iter : ATL!Iterator)
```

```
rule CompleteRuleInvocation_InvalidAssoc(
      invocation : ATL!OperationCallExp,
      problemData : Constraints!Invalid_Association_Data,
      iter : ATL!Iterator)
```

```
rule CompleteRuleInvocation_InvalidAssoc(
      invocation : ATL!OperationCallExp,
      potentialProblem : Constraints!Invalid_Association,
      iter : ATL!Iterator)
```

```
rule CreateCheckerHelper (
      modul :  ATL!Module,
      constraint : Constraints!Constraint,
      sourceMM : ATL!OclModel)
```

```
rule CreateCheckerHelper (
      modul :  ATL!Module,
      constraint : Constraints!Constraint,
      sourceMM : ATL!OclModel)
```