Formalización de transformaciones de refinamiento de componentes

Francisca Rosique Contreras, Pedro Sánchez Palma, Diego Alonso Cáceres
División de Sistemas e Ingeniería Electrónica (DSIE)
Universidad Politécnica de Cartagena,
Campus Muralla del Mar s/n, Cartagena E-30202, Spain
paqui.rosique@upct.es, pedro.sanchez@upct.es, diego.alonso@upct.es

Resumen. *En el proceso de desarrollo de software dirigido por modelos es posible encontrar situaciones en las que las transformaciones entre modelos de distintos niveles de abstracción necesitan un tratamiento previo que facilite la transformación, sobre todo cuando se dispone de varios modelos específicos de la plataforma. En este documento se propone como solución la realización de una transformación de refinamiento, previa a la obtención del modelo específico de la plataforma.*

1 Introducción

En el proceso de desarrollo de software dirigido por modelos es posible encontrar situaciones en las que las transformaciones entre modelos de distintos niveles (la mayoría de las veces entre PIM – PSM) necesitan un tratamiento previo que facilite la transformación, sobre todo cuando se dispone de varios PSMs.

Este problema se ha encontrado en el framework de desarrollo de sistemas domóticos HABITATION [1]. En el momento de realizar la transformación de los modelos de componentes del nivel PIM (Modelo Independiente de la Plataforma) a modelos de la plataforma específica del nivel PSM (Modelo Específico de la Plataforma), aparece un problema de adaptación debido a la propia dependencia de la plataforma.

Este mismo problema puede encontrarse en otros dominios y en otras situaciones donde se necesite seleccionar una determinada distribución de los componentes con anterioridad a la transformación PIM-PSM.

En este documento se propone como solución realizar una transformación de refinamiento previa a la obtención del modelo específico de la plataforma

2 Proceso de Refinamiento

Por norma general los dispositivos domóticos están compuestos de una o varias funcionalidades, pero dependiendo de la plataforma final de implementación los dispositivos pueden variar su composición, de manera que en una tecnología un dispositivo implemente 4 funcionalidades integradas todas en el mismo dispositivo y en otra tecnología estas mismas funcionalidades se tengan que integrar en dos dispositivos diferentes en lugar de un uno solo.

Por esta razón se decidió trabajar con unidades funcionales en lugar de trabajar directamente con dispositivos, de modo que se pueda asegurar la independencia de plataforma en las primeras fases de desarrollo. El problema de trabajar con unidades funcionales surge en el salto de un modelo totalmente independiente de plataforma a un modelo de plataforma donde se modelan dispositivos reales. El gran problema es, en resumen, intentar dotar a un modelo totalmente independiente de una cierta dependencia de la plataforma.

Una posible solución a este problema es disponer de un catálogo de dispositivos domóticos predefinidos para cada una de las tecnologías existentes (por lo menos de las más importantes). En este catálogo se indicará para cada dispositivo que funcionalidades lo compone.

Por este motivo se hace indispensable tener catalogados los distintos dispositivos que se pueden encontrar en las diferentes tecnologías teniendo en cuenta las funcionalidades que ofrecen. Y dado que las unidades funcionales son representadas en el modelo de componentes como un componente simple, la forma más sencilla de realizar esta operación de catalogación es mediante un refinamiento del modelo de componentes.

El refinamiento se define en términos de MDA [2] como la adición de los detalles necesarios para pasar de un modelo abstracto (por ejemplo un PIM) a un modelo más detallado (que puede ser un PSM) que en general estará más cerca de una plataforma de implementación. En el caso del dominio domótico los detalles que se han añadido han sido las agrupaciones de las distintas funcionalidades que conforman el sistema a desarrollar según la disponibilidad de dispositivos de la plataforma final seleccionada.

Para realizar el proceso de refinamiento se necesita:

1. Disponer de un metamodelo de componentes.
2. Disponer de un catálogo de dispositivos representados como componentes compuestos. Estos componentes compuestos indicarán de que funcionalidades se componen (representadas como componentes simples) y cómo interactúan entre ellas.

3. Reglas de transformación donde los modelos origen y destino son conformes al mismo metamodelo de componentes.

   a. Se utilizarán dos modelos de entrada, 1)el modelo de componentes simples directamente obtenido de la transformación del modelo de aplicación DSL del nivel CIM donde se modelaban funcionalidades de forma independiente a cualquier plataforma y 2) el modelo de componentes compuestos que representa el catálogo de dispositivos para una determinada plataforma.

   b. Buscar mediante pattern matching coincidencias entre los componentes simples que componen cada uno de los dispositivos catalogados como componentes compuestos y los componentes simples del modelo de aplicación. (Ver Figura 1) dando lugar al modelo de componentes compuestos de la aplicación.

   c. El modelo destino se corresponde con un modelo de componentes que modela una aplicación domótica estando ahora representados los dispositivos concretos a utilizar en la implantación final. El modelo de componentes obtenido ya contiene una cierta dependencia de plataforma y está preparado para ser transformado a un modelo de plataforma final.

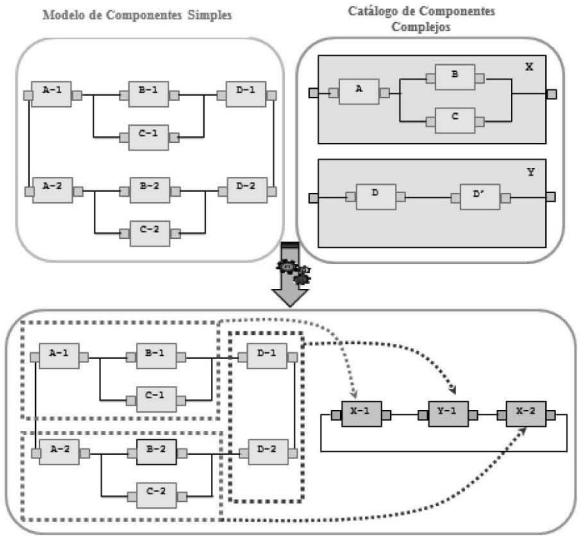


Figura 1: Esquema pattern matching aplicado.

3 Metamodelo de Componentes

Una arquitectura de componentes se caracteriza por tener como elementos arquitectónicos componentes y conectores. Para la definición de estos componentes y conectores se han definido una serie de metaclases y se han tenido en cuenta las posibles relaciones de inclusión o transformación que serán necesarias. A continuación se puede observar dicho metamodelo.

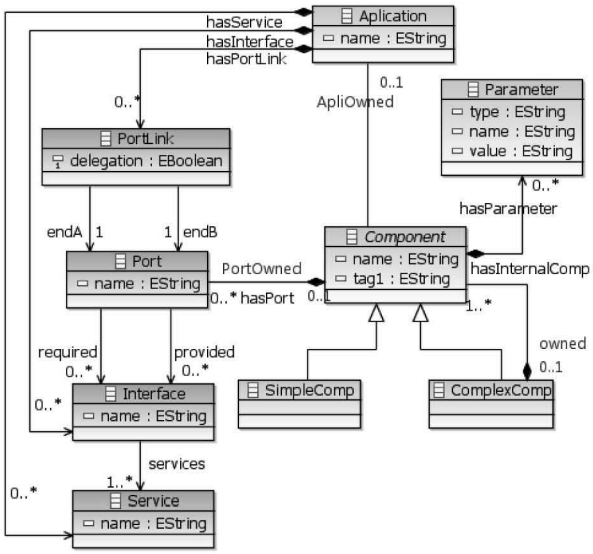


Figura 2: Metamodelo de componentes.

4 Catálogo de Componentes

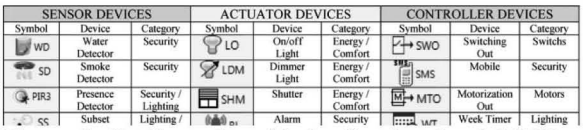
Para poder realizar el refinamiento de un modelo de componentes es necesario tener un modelo donde se recojan en forma de catálogo aquellos componentes complejos o incluso componentes con una funcionalidad de componentes

ComplexComponent: Un componente complejo representa un dispositivo domótico para una determinada plataforma. Este componente está formado por uno o varios componentes simples, pudiendo darse el caso que los componentes simples que lo compone sean entre ellos del mismo tipo. Los atributos más importantes son:

- Name (EString): Indica el nombre del componente y funciona como atributo identificador. Este nombre debe coincidir con el nombre del dispositivo real.
- Tag1 (EString): es un campo auxiliar que puede utilizarse para añadir una descripción más detallada del componente.

SimpleComponent: Un componente simple representa una funcionalidad interna de un dispositivo. Dentro del catálogo sólo se pueden encontrar integrados en un componentes complejo. Los atributos más importantes de un SimpleComponent son:

- Name (EString): Indica el nombre del componente, este nombre coincide con el nombre que reciben las definiciones de unidades funcionales del DSL. En cierto modo identifica el tipo de funcionalidad que ofrece este componente.
- Owned: Muestra el ComplexComponent donde está encapsulado este SimpleComponent
- Tag1 (EString): Indica el índice del SimpleComponent. Este campo toma la función junto con el nombre de identificador, dado que un mismo ComplexComponent puede contener varios componentes del mismo tipo.



SENSOR DEVICES			ACTUATOR DEVICES			CONTROLLER DEVICES		
Symbol	Device	Category	Symbol	Device	Category	Symbol	Device	Category
WD	Water Detector	Security	LO	On/off Light	Energy / Comfort	SWO	Switching Out	Switchs
SD	Smoke Detector	Security	LDM	Dimmer Light	Energy / Comfort	SMS	Mobile	Security
PIR3	Presence Detector	Security / Lighting	SHM	Shutter	Energy / Comfort	MTO	Motorization Out	Motors
SS	Subset	Lighting /	BL	Alarm	Security	WT	Week Timer	Lighting

Figura 3: Catálogo de unidades funcionales del DSL

Port: Todo componente, ya sea simple o complejo, va a tener una serie de puertos. Los puertos representan los puntos de iteración del componente con el resto del sistema. Los atributos más importantes de un Port son:

- Name (EString): Indica el nombre del puerto, este nombre sigue una nomenclatura del tipo "port_NombreComponente_Indice".
- Provided: Indica la interfaz provista.
- Required: Indica la interfaz requerida.

PortLink: Este elemento permite crear un vínculo entre los componentes o mejor dicho entre los puertos de los componentes. Estos enlaces pueden ser del tipo "delegation", este tipo de enlace conecta un puerto definido en un componente interno con un puerto compatible definido en su componente complejo contenedor. Los enlaces no delegation conectan dos puertos de dos componentes definidos al mismo nivel.

- delegation (EBoolean): Indica si es un portlink del tipo delegation o no.
- endA (Port): Indica el puerto origen.
- endB (Port): Indica el puerto destino.

5 Reglas de Transformación

Una regla de transformación de modelos debe definir, evitando cualquier ambigüedad, la relación implícita que existe entre sus partes. MDA no especifica ni prescribe ningún lenguaje para la transformación de modelos.

Se han propuesto varios lenguajes de transformación para especificar las transformaciones. La mayoría de ellos lenguajes asumen que los modelos involucrados en la transformación cuentan con una definición formal de su sintaxis, expresada en términos de metamodelos MOF [3]. En este trabajo se ha utilizado ATL (Atlas Trasformation Language) [4] para describir la transformación de refinamiento. Un ejemplo de regla de transformación implementada es la siguiente:

Regla 2: Regla del tipo matched rule donde por cada parámetro en el modelo de entrada de componentes simples se crea un parámetro en el modelo de salida, dándole el mismo nombre que el modelo de componentes simples.

```
rule Parametros{
      from
            s: A!Parameter
      to
            t: C!Parameter(
                  name<-s.name,
                  type<-s.type,
                  value<-s.value
            )
}
```

Regla 6: Regla del tipo called rule, utilizada para crear los puertos de los componentes complejos. Esta regla recibe dos argumentos de entrada que son un puerto del modelo de entrada del catálogo y un string. Esta regla crea un puerto con un nombre en formato name-X. donde name será el mismo nombre que el puerto del catálogo y X es un índice que coincide con el índice del componente Complejo que contiene a este puerto. El resto de valores son idénticos a los del catálogo. Una vez creado el puerto este es añadido al componente complejo correspondiente.

```
rule CrearPuertosComplex(p:B!Port, s:String){
     to t:C!Port(      name<-s  )
     do{
     thisModule.puerto<-t;
     for(i in thisModule.apli. hasInterface){
      if(p.required->notEmpty()){
        if(i.name=(p.required->first()).name)
           t.required<-i;}
      if(p.provided->notEmpty()){
        if(i.name=(p.provided->first()).name)
           t.provided<-i;}
       }
       t.PortOwned<-thisModule.complex;
     }
  }
```

Agradecimientos

This work was partially supported by the Spanish CICYT project EXPLORE (TIN2009-08572).

Referencias

[1] Jiménez, M., Rosique, F.,Sánchez, P.,Álvarez, B. and Iborra, A., 2009.Habitation: A Domain-Specific Language for Home Automation. IEEE Software. 26(4): 33-38.

[2] Mellor, S., Scoot, K., Uhl, A.and Weise, D., 2004. MDA Distilled: Principles of Model-Driven Arquitecture.Addison Wesley.

[3] MOF. Mata Object Facility 1.3. OMG (1999).

[4] Jouault, F, Kurtev, I. Transforming Models with ATL. 2005. Proceedings of the Model Transformations in Practice Workshop at MoDELS 05, Montego Bay, Jamaica.