5. Infraestructura UML y MOF

Referencias

- OMG Unified Modeling Language (UML) Infrastructure, V2.3, 2010 http://www.omg.org/spec/UML/2.3/Infrastructure/PDF/
- OMG Meta Objetc Facility (MOF) Specification, V2.0, 2006 http://www.omg.org/spec/MOF/2.0/PDF/

Referencias

- OMG Unified Modeling Language (UML) Superstructure, V2.3, 2010 http://www.omg.org/spec/UML/2.3/Superstructure/PDF/
- OMG MOF 2.0/XMI Mapping, Version 2.1.1, 2007 http://www.omg.org/spec/XMI/2.1.1/PDF/

Referencias

- F. Budinsky, D. Steinberg, E. Merks, R. Ellersick, T. J. Grose, *Eclipse Modeling Framework: A Developer's Guide*, Addison-Wesley, 2003

Índice

- Introducción
- Metamodelo UML
- Paquete `Core`
- Fusión de paquetes
- Paquete `Core::PrimitiveTypes`
- Paquete `Core::Constructs`

Índice

- Paquete `Core::Profiles`
- MOF
- Superestructura UML
- XMI
- Ecore
- Conclusiones

Introducción

- Desde la versión 2.0 UML está dividido en dos especificaciones:
  - La infraestructura UML
  - La superestructura UML
- La infraestructura UML define un núcleo de metamodelado que sirve para definir meta-metamodelos como MOF

Introducción

- La superestructura UML es el metamodelo de UML descrito en MOF
- Hay un alineamiento arquitectónico
- Así, básicamente el metamodelo para clases UML coincide con el meta-metamodelo de MOF

Metamodelo de UML

- UML se utiliza como notación visual para caracterizar modelos durante el análisis, diseño y despliegue de sistemas
- UML está descrito utilizando un metamodelo
- Dicho metamodelo se ajusta a una serie de principios:
  - Modularidad
  - División según la arquitectura de cuatro capas OMG

Metamodelo de UML

  - División
  - Extensibilidad
    - Modificación del metamodelo
    - Perfiles UML
  - Reusabilidad
- La infraestructura de UML está definida en la `InfrastructureLibrary`

Metamodelo de UML

- Dicha `InfrastructureLibrary` cumple con varias requisitos:
  - Definir un metalenguaje básico que pueda ser reutilizad para definir distintos metamodelos como UML o MOF
  - Alinear arquitectónicamente UML, MOF y XML para soportar el intercambio de modelos
  - Permitir personalizaciones de UML mediante perfiles, y la creación de nuevos lenguajes basados en el mismo núcleo de metalenguaje que UML

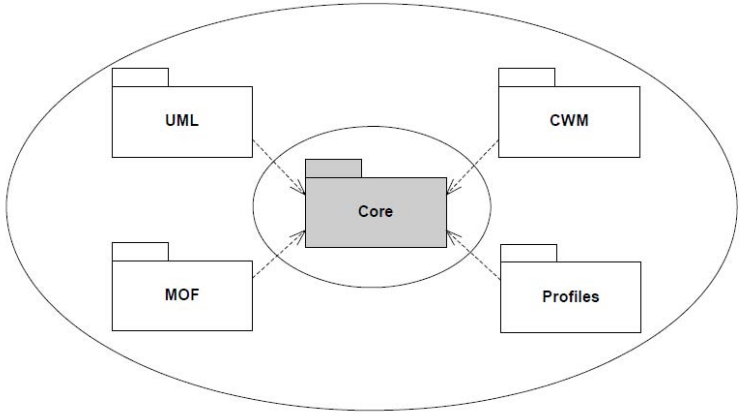
Metamodelo de UML

- La `InfrastructureLibrary` está formada por dos paquetes:
  - Core
  - Profiles

Paquete `Core`

- El paquete `Core` es un metamodelo completo diseñado para una alta reusabilidad
  - Otros metamodelos al mismo metanivel importan o especializan sus metaclases
  - Es el núcleo de MDA

Paquete `Core`

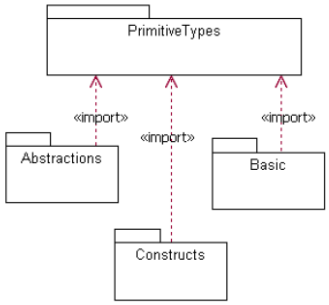


El paquete `Core` como el núcleo de MDA

Paquete `Core`

- El paquete `Core` está formado por otros cuatro paquetes:
  - `PrimitiveTypes`: tipos predefinidos
  - `Abstractions`: metaclases abstractas reutilizables por otros metamodelos
  - `Constructs`: metaclases concretas para modelado orientado a objetos. Reutilizada por MOF y UML
  - `Basic`: fundamentos para el XMI generado para UML y MOF, entre otros

Paquete `Core`

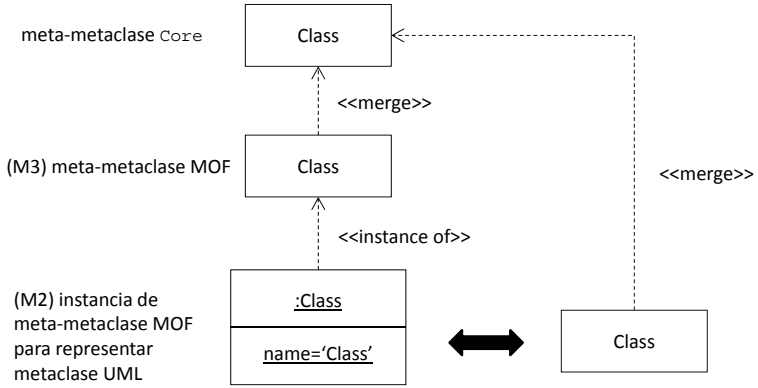


El paquete Core

Paquete `Core`

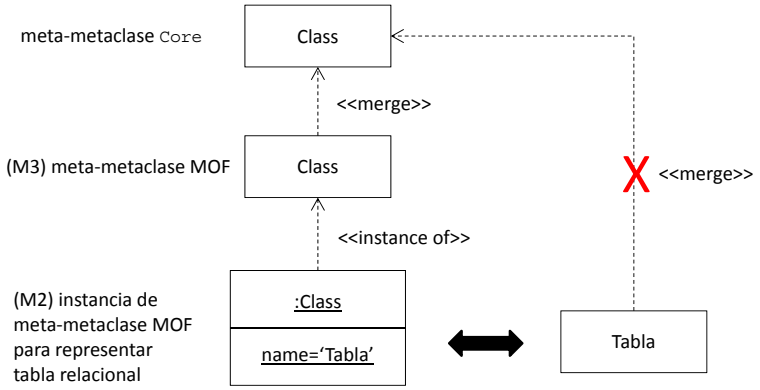
- El paquete `Core` logra el *alineamiento arquitectónico* entre MOF y UML:
  - `Core` es el núcleo
  - MOF está descrito a través de `Core`
  - UML es una instancia de MOF, cuya representación coincide con el propio `Core`

Paquete `Core`



Alineamiento arquitectónico entre `Core`, MOF y UML Superstructure

Paquete `Core`



Falta de alineamiento arquitectónico entre `Core`, MOF y el modelo relacional

Paquete `Core`

- Basicamente el lenguaje de M_n nos da el mecanismo de definición de M_{n-1}
- Por ejemplo, si en M4 tuviéramos XML, en M3 podríamos describir MOF como:

```
<!ELEMENT Class (Attribute+, Function+)>
<!ATTRIBUTE Class name CDATA #IMPLIED>
```

- Y en M2 tendríamos el modelo relacional como:

```
<Class name="Table">…</Class>
<Class name="Column">…</Class>
```

Paquete `Core`

- En OMG MDA:
  - En M4 está `Core`, por eso decimos que no es necesario y que M3 es reflexivo
  - En M3 por tanto, está `Core`
  - En M2 la instancia de `Core` que caracteriza UML, que vuelve a ser `Core` (al menos para los diagramas de clases)
- De ahí el alineamiento arquitectónico

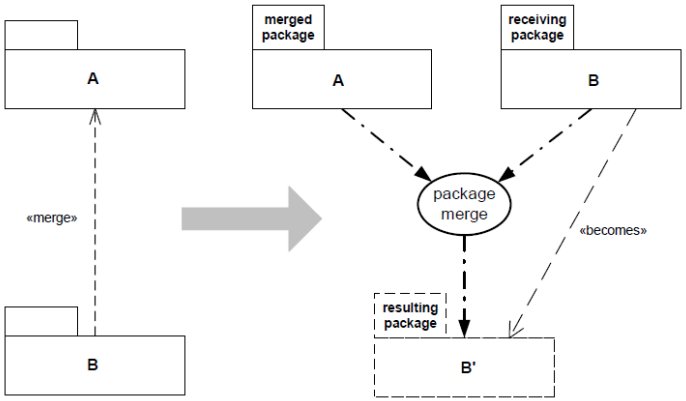
Fusión de paquetes

- Una herramienta fundamental en MDA es el *package merge* o fusión entre paquetes:
  - La fusión es una relación entre dos paquetes que indica que los contenidos de ambos son *combinados*
  - Se utiliza cuando elementos definidos en distintos paquetes tienen el mismo nombre y representan el mismo concepto
  - También se utiliza para proporcionar distintas definiciones de un concepto para distintos propósitos partiendo de una definición base común

Fusión de paquetes

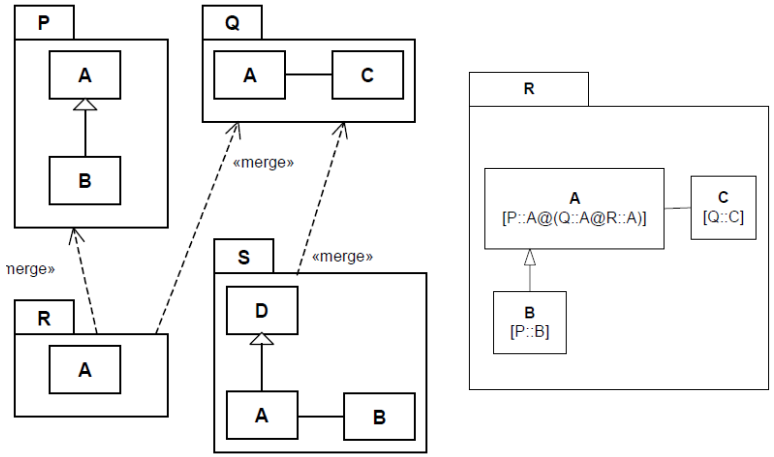
- Una fusión entre dos paquetes implica un conjunto de transformaciones, donde los contenidos del paquete a ser fusionado se combinan con los contenidos del paquete receptor
- Si un elemento está repetido en ambos paquetes, se combina en un único elemento resultante

Fusión de paquetes



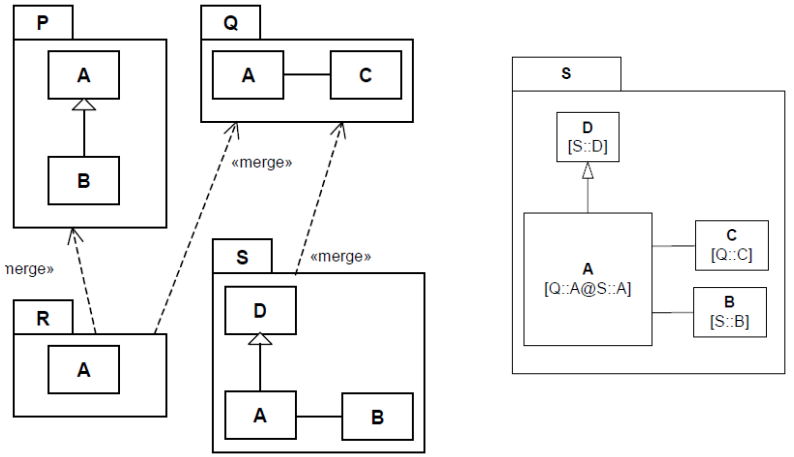
Vista conceptual de la semántica de la fusión de paquetes

Fusión de paquetes



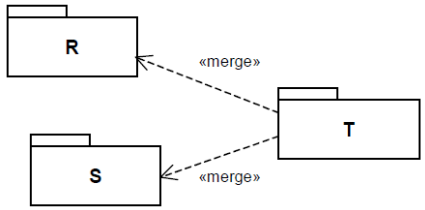
Ejemplo fusión de paquetes

Fusión de paquetes



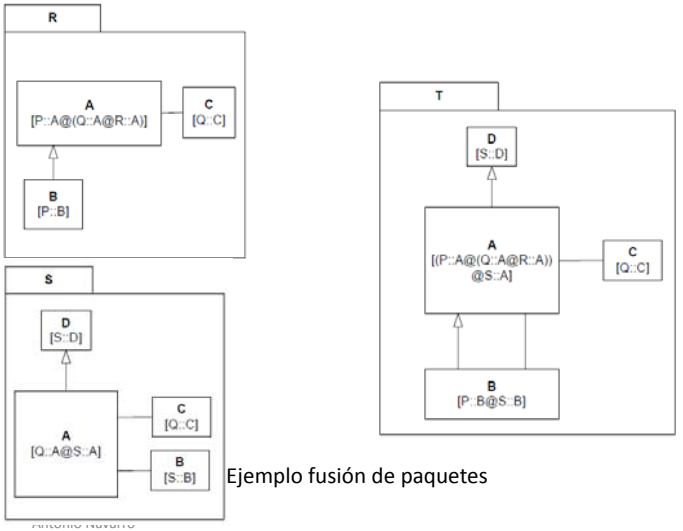
Ejemplo fusión de paquetes

Fusión de paquetes



Ejemplo fusión de paquetes

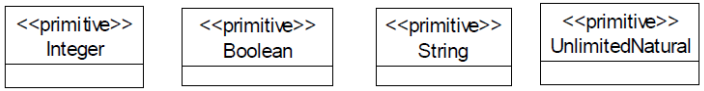
Fusión de paquetes



Ejemplo fusión de paquetes

Paquete Core::PrimitiveTypes

- El subpaquete PrimitiveTypes del paquete Core define los diferentes tipos de valores primitivos que se utilizan para definir el metamodelo Core



Los elementos del paquete PrimitiveTypes

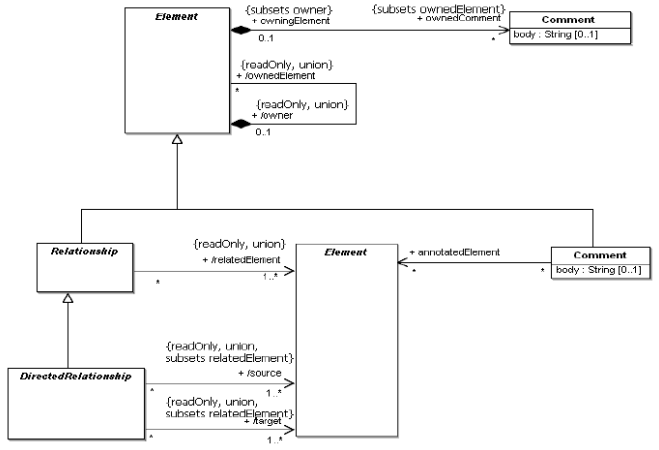
Paquete Core::Constructs

- El subpaquete Constructs del paquete Core importa los elementos del paquete PrimitiveTypes y fusiona múltiples paquetes definidos en el paquete Abstractions
- Está formado por nueve diagramas:
  - Root
  - Namespaces
  - Packages

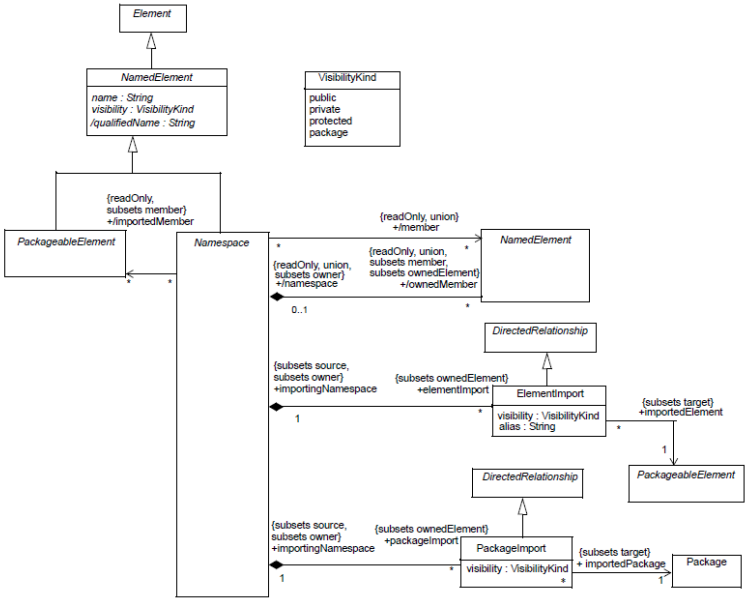
Paquete Core::Constructs

- Classifiers
- Classes
- Operations
- Constraints
- Expressions
- Datatypes

Paquete Core::Constructs

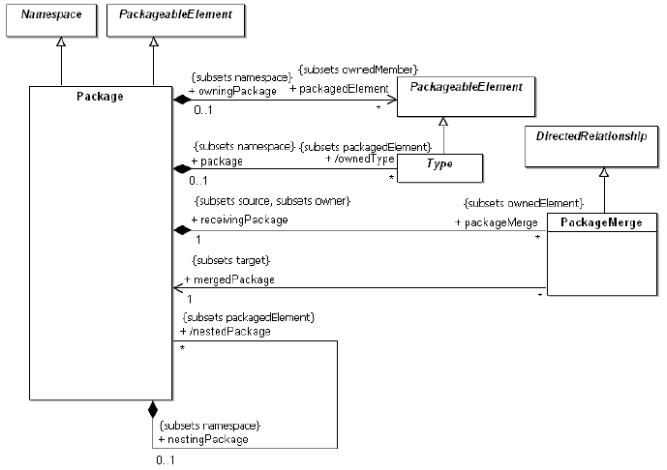


El diagrama Root del paquete Constructs



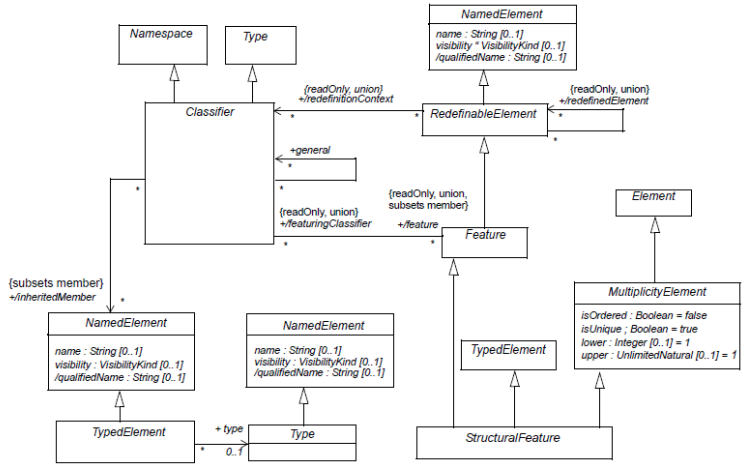
El diagrama `Namespaces` del paquete `Constructs`

Paquete `Core::Constructs`

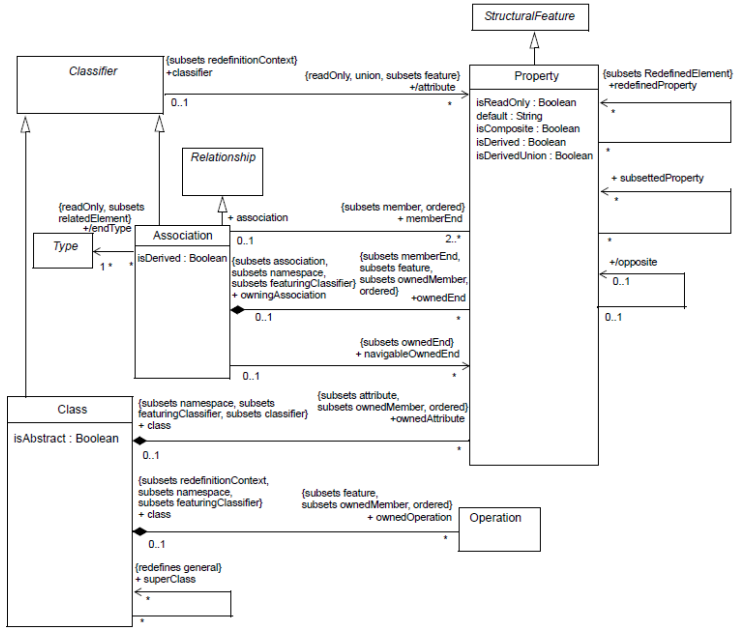


El diagrama `Packages` del paquete `Constructs`

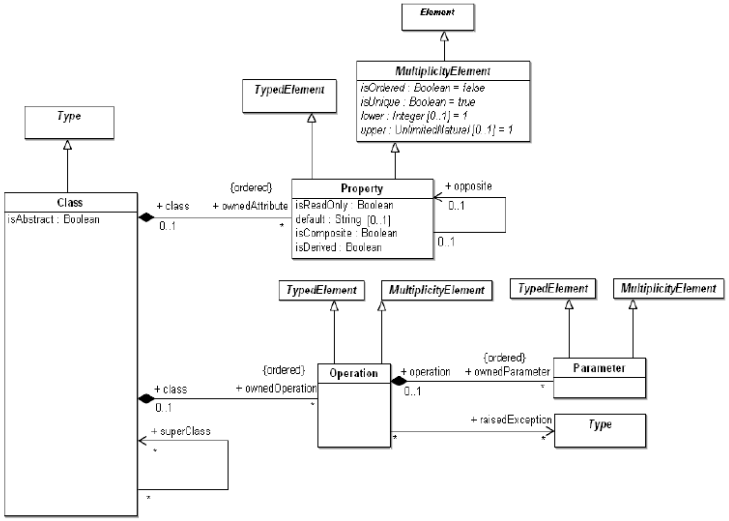
Paquete `Core::Constructs`



El diagrama `Classifiers` del paquete `Constructs`



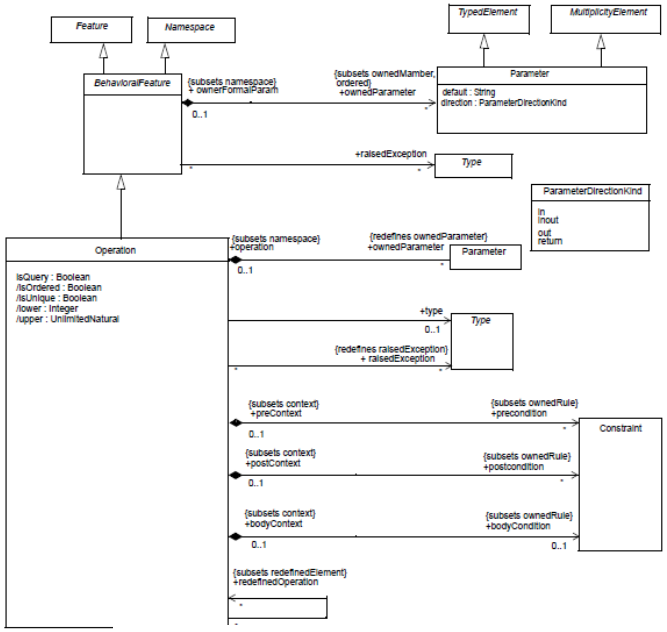
El diagrama `Classes` del paquete `Constructs`



El diagrama `Classes` del paquete `Basic`

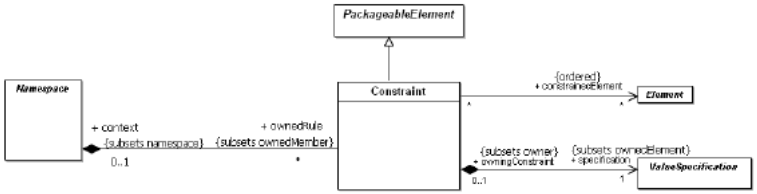
Paquete `Core::Constructs`

El diagrama `Classes` del paquete `Kernel` de UML Superstructure **en M2**



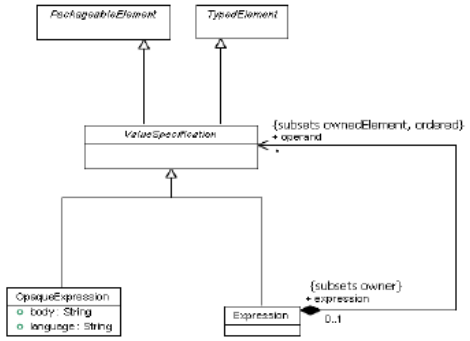
El diagrama `Operations` del paquete `Constructs`

Paquete `Core::Constructs`



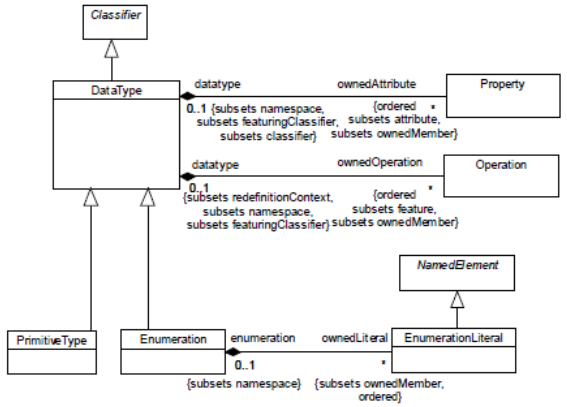
El diagrama `Constraints` del paquete `Constructs`

Paquete Core::Constructs



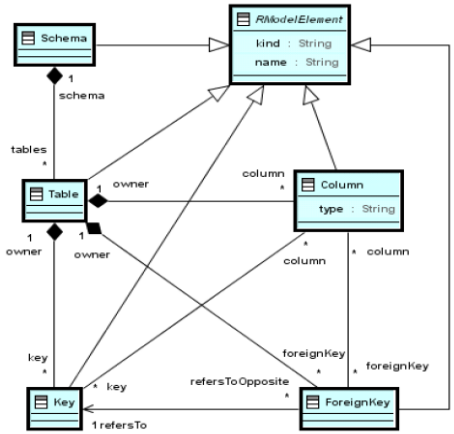
El diagrama `Expressions` del paquete `Constructs`

Paquete Core::Constructs



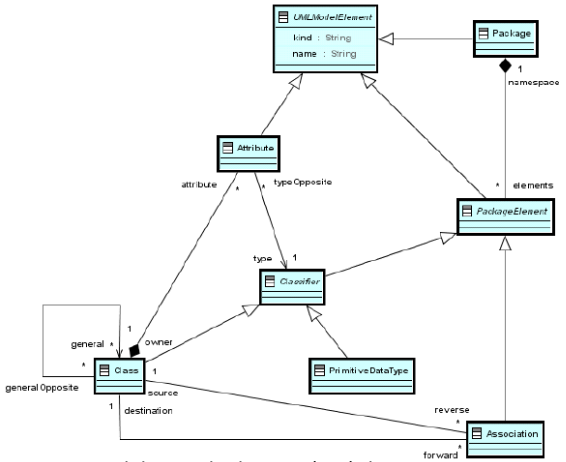
El diagrama `DataTypes` del paquete `Constructs`

Paquete Core::Constructs



Metamodelo simple del modelo relacional (M2) descrito en MOF

Paquete Core::Constructs



Metamodelo simple de UML (M2) descrito en MOF

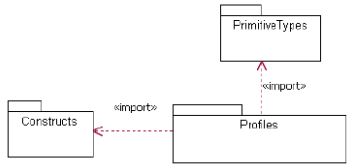
Paquete `Core::Profiles`

- El paquete `Profiles` contiene los mecanismos que permiten extender metaclases de metamodelos existentes para adaptarlas a distintos propósitos
  - Por ejemplo, adaptar el metamodelo UML a plataformas (J2EE) o dominios (p.e. tiempo real)

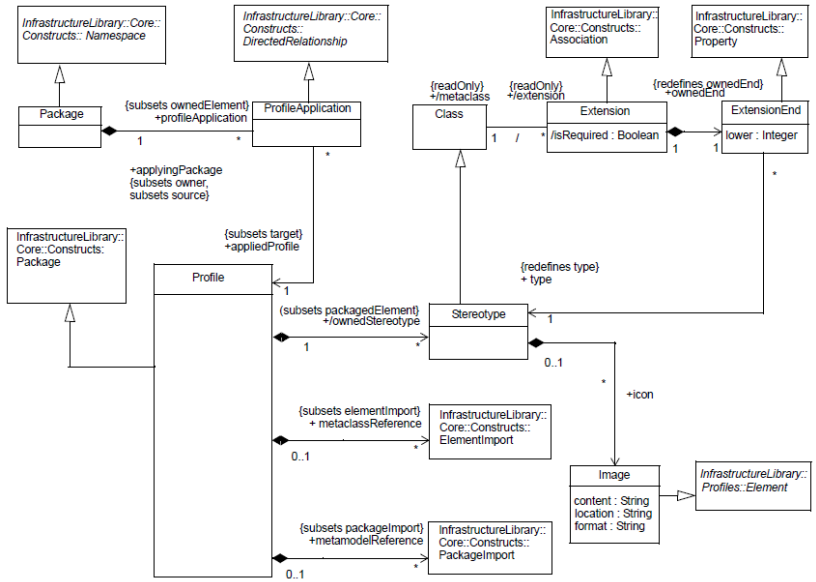
Paquete `Core::Profiles`

- Este paquete está definido al nivel meta-metamodelo (como MOF)
- Así los estereotipos pueden afectar a elementos del metamodelo (p.e. clases, estados, casos de uso UML)
- Los perfiles no modifican un metamodelo, lo adaptan para usos concretos

Paquete `Core::Profiles`

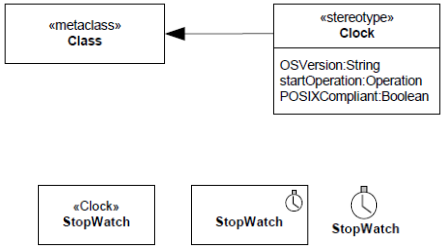


El paquete `Core::Profiles`



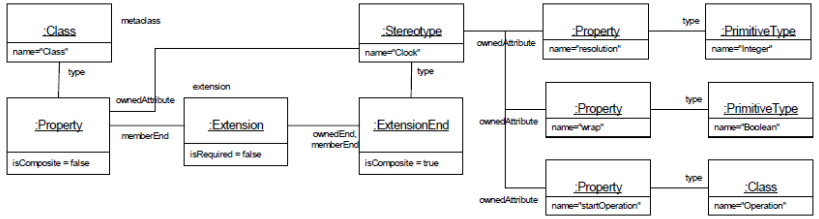
Clases definidas en `Core::Profiles`

Paquete `Core::Profiles`



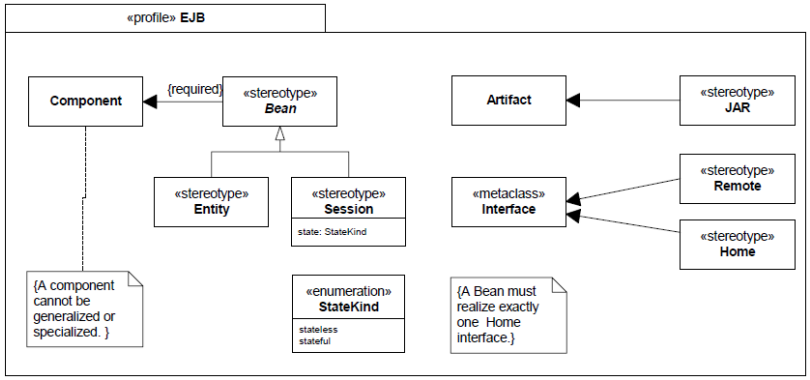
Definición y uso del estereotipo `Clock`

Paquete `Core::Profiles`



Definición del estereotipo `Clock` en términos de la instancia de las clases definidas en `Core::Profiles`

Paquete `Core::Profiles`



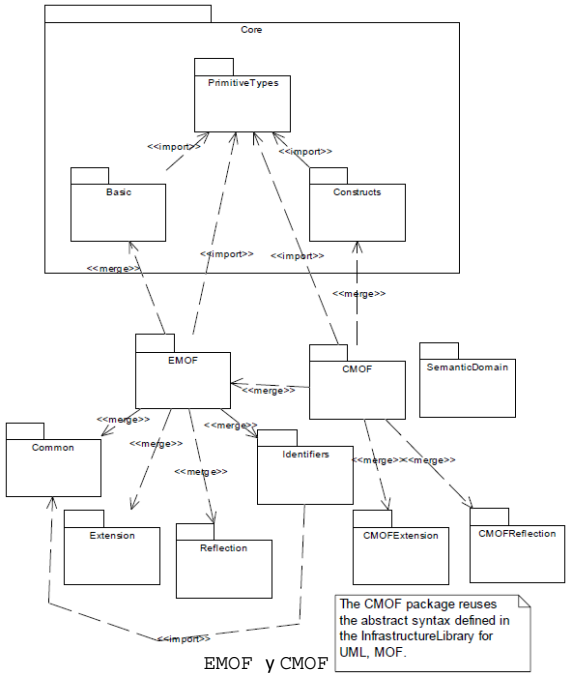
Ejemplo de perfil EJB para UML

MOF

- MOF (*Meta Object Facility*) es el meta-metamodelo OMG
- Al hacer la fusión del paquete Core, al igual que UML, básicamente permite definir modelos utilizando una sintaxis visual similar a la de UML

MOF

- MOF está dividido en dos paquetes, según hagan la fusión de `Core::Basics` o de `Core::Constructs`:
  - EMOF (*Essential MOF*)
  - CMOF (*Complete MOF*)

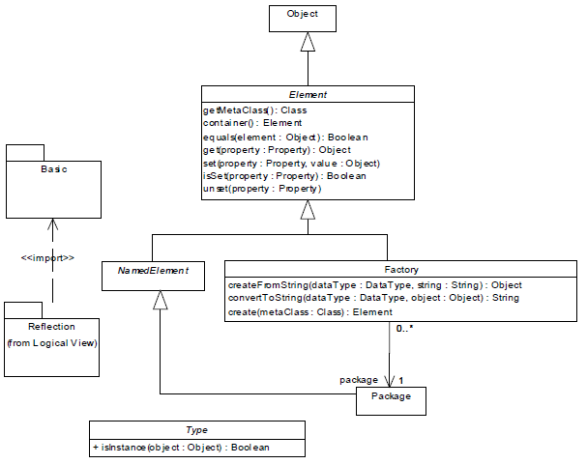


EMOF y CMOF

MOF

- La principal característica que añade MOF es el de la reflexión:
  - Cada elemento tiene una clase que define sus propiedades y operaciones
- Por lo demás, básicamente reutiliza la definición de `Core`

MOF



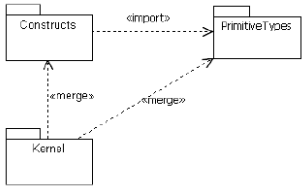
El paquete `Reflection`

Superestructura UML

- La superestructura UML es el metamodelo UML:
  - Instancia de MOF
  - Que hace la fusión de paquetes de `Core::Constructs`
- Es igual al estar alineados arquitectónicamente

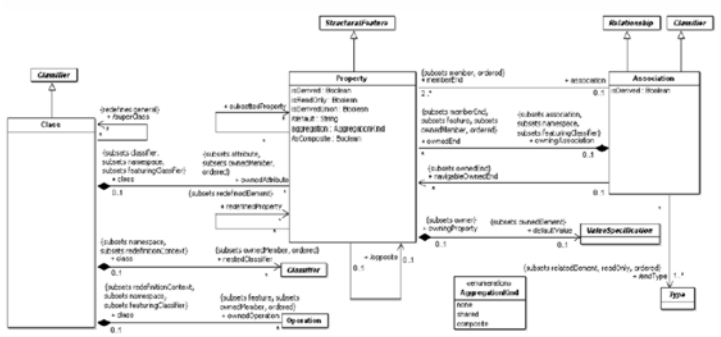
Superestructura UML

- Ejemplo



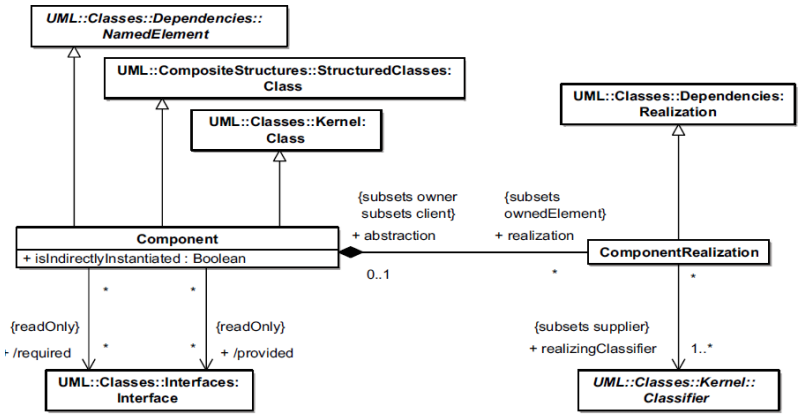
El paquete `Kernel` de la superestructura UML

Superestructura UML



El diagrama `Classes` del paquete `Kernel`

Superestructura UML



El diagrama `Components` de la superestructura UML

UML WAE

- *UML Web Application Extension* es un perfil UML creado por Jim Conallen para modelar aplicaciones web
- Para Conallen, una *aplicación web* es un sistema web (servidor web, red, HTTP, navegador) en el que la interacción del usuario (navegación y entrada de datos) afecta el estado del negocio

UML WAE

- Conallen distingue entre aplicación web y *sitio web*
- Para Conallen, una *página web* es básicamente cualquier cosa que puede ser servida por un servidor web
- Parte de la *separación de intereses*: el comportamiento de una página web en el servidor es distinto que en el cliente

UML WAE

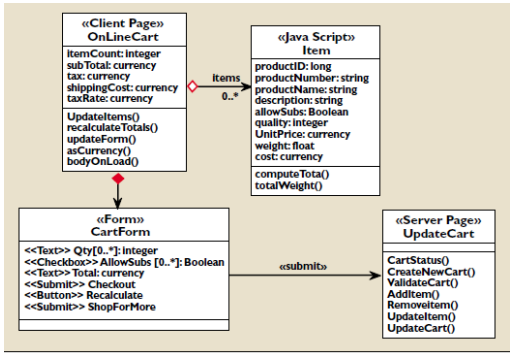
- Conallen opta por asimilar páginas web a clases UML estereotipadas:
  - `client page`: página de cliente (p.e. HTML)
  - `server page`: página de servidor (p.e. Servlet)
  - `form`: formulario de entrada de datos

UML WAE

- Las relaciones entre páginas se establecen con asociaciones navegadas estereotipadas:
  - `link`: navegación entre páginas
  - `build`: construcción de una `client page` a partir de una `server page`
  - `submit`: envío de información de un `form`

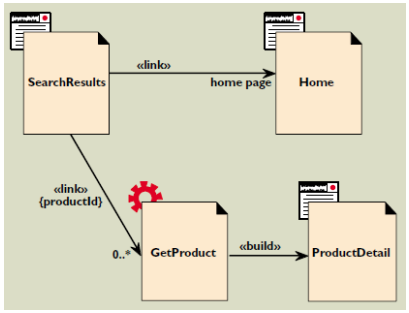
UML WAE

- Ejemplos



Notación UML WAE

UML WAE

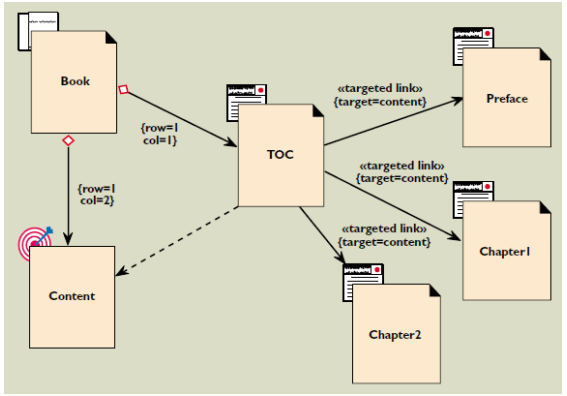


Notación UML WAE

UML WAE

- También utiliza varios estereotipos para caracterizar marcos:
  - `frameset`: una página con marcos
  - `target`: un marco destino de enlaces
  - `targeted link`: el marco destino de un enlace

UML WAE



Marcos en UML WAE

UML WAE

- Nótese que UML WAE es dependiente de la arquitectura
  - Model 1
  - MVC
- En el caso de una arquitectura MVC, UML WAE está restringido a la capa de presentación

UML WAE

- Por tanto, para el modelado de una aplicación multicapa simplemente necesitamos
  - UML WAE para la capa de presentación
  - UML para el resto de capas
- UML WAE básicamente modela el código de los elementos software involucrados
- Normalmente las notaciones de diseño web se centran el modelado de las aplicaciones, en lugar del modelado del código

XMI

- XML Metadata Interchange (XMI) es un mecanismo para generar esquemas XML a partir de un metamodelo descrito en MOF
- Permite por tanto serializar como documentos XML modelos instancia del metamodelo descrito en MOF

XMI

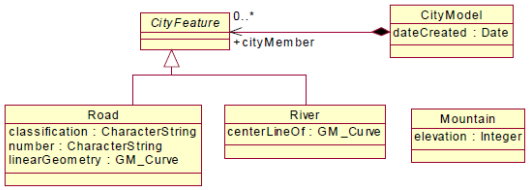
- Por niveles:

Nivel	Modelado	Representación XML
M3	MOF	XML + reglas XMI
M2	UML	Esquema XML de UML
M1	Modelo empresa	Documento XML instancia del esquema

Ejemplo de uso de XMI

XMI

- Ejemplo:



Metamodelo MOF de un sistema de información geográfica

XMI

```
<xsd:annotation>
  <xsd:documentation>CLASS: Road</xsd:documentation>
</xsd:annotation>

<xsd:complexType name="Road">
  <xsd:extension base="CityFeature">
    <xsd:sequence>
      <xsd:element name="classification" type="xsd:string"
nillable="true"/>
       <xsd:element name="number" type="xsd:string"
nillable="true"/>
       <xsd:element name="linearGeometry"
type="xsd:string" nillable="true"/>
    </xsd:sequence>
  </xsd:extension>
</xsd:complexType>

<xsd:element name="Road" type="Road"/>
```

Fragmento del esquema XML generado a partir del metamodelo anterior

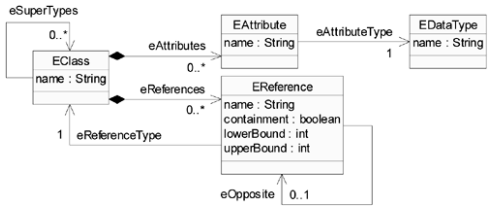
XMI

- A veces hay que elaborar el esquema XML generado automáticamente con las reglas XMI ya que este esquema puede no estar optimizado
  - P. ej., al no haber herencia múltiple en los esquemas XML, los atributos heredados se repiten en cada clase

Ecore

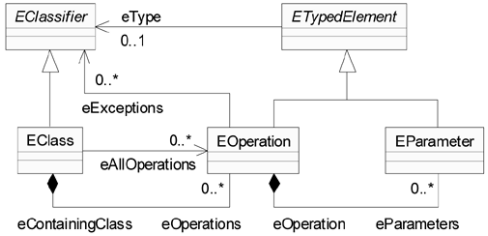
- El soporte del metamodelo definido en MOF (básicamente `Core::Basics` o `Core::Constructs`) puede ser muy costoso para una herramienta
- Eclipse tiene su propio meta-metamodelo: eCore

Ecore



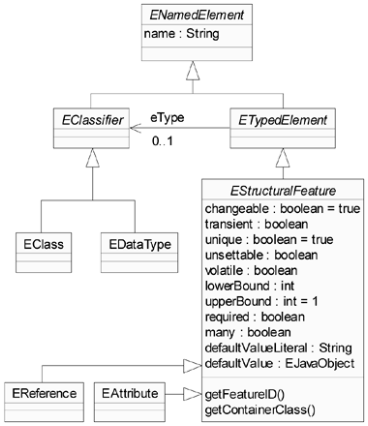
El núcleo del modelo Ecore

Ecore



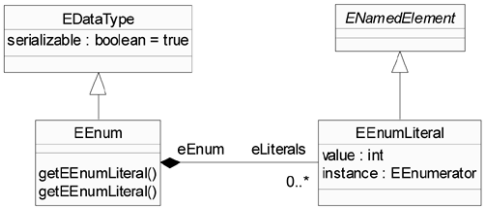
Características de comportamiento Ecore

Ecore



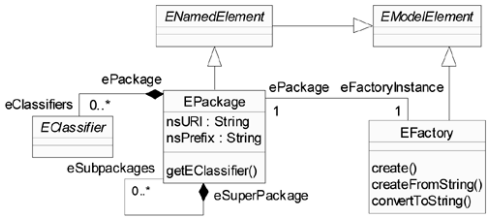
Características estructurales Ecore

Ecore



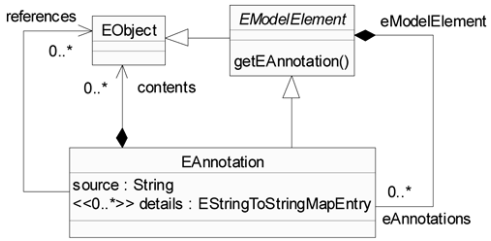
Enumerados y literales Ecore

Ecore



Paquetes Ecore

Ecore



Anotaciones Ecore

Ecore

- Ecore forma parte del *Eclipse Modeling Framework* (EMF), que permite:
  - Definir un metamodelo instancia de Ecore
  - Generar editores para las instancias del metamodelo definido con Ecore
- *Eclipse Ecore Tools* permite definir el metamodelo instancia de Ecore de manera visual

Ecore

- *Eclipse Graphical Modeling Project* (GMP) proporciona un marco para desarrollar editores gráficos basados en EMF y GEF (*Graphical Editing Framework,* el framework gráfico de Eclipse)
  - *Graphic definition*
  - *Tooling definition*
  - *Mapping definition*

Ecore

- ATL es un lenguaje de transformaciones construido sobre EMF

Conclusiones

- Conceptos básicos de modelado OO: `Core::Constructs`
- Definido en UML Infraestructure y reutilizado en MOF y UML Superstructure
- Reutilización basada en el alineamiento arquitéctonico y en la fusión de paquetes
- Mecanismo de extensión de metamodelos: `Core::Profiles`

Conclusiones

- Metamodelo MOF de UML: UML Superstructure
- Ecore: alternativa Eclipse más ligera