Agenda

- Introducción
- Objetivos
- Modelo
- UML
- OCL
- MDA
- Refinamientos
- Micromodelos de Software
- Generación de micromundos representativos
- Conclusiones
- ePlatero
- Comparación con Alloy Analyser

Introducción

- Escribir transformaciones de modelos es una tarea compleja y propensa a errores.
- Son requeridas técnicas de verificación formal.
- Validación por testo.

Objetivos

- Definir e implementar una estrategia eficiente y confiable para validar transformaciones de modelos UML.
- Generar micromundos que permitan determinar la veracidad y correctitud de los refinamientos.
- Automatización de la generación de micromundos.

Modelo

- La construcción de un sistema de software debe ser precedida por la construcción de un modelo.
- El **modelo** de un sistema es una representación conceptual de los elementos que constituyen el problema.
- Actúa como una especificación de los requerimientos que el sistema debe satisfacer.
- Es un medio de comunicación y negociación.

UML

- El modelo de un sistema se expresa utilizando un lenguaje de modelado.

- Es un lenguaje gráfico para visualizar, especificar, construir y documentar los artefactos de sistemas de software.

- Ventaja:
  - ☑ Intercambio de diagramas y formas de representación entre diversas herramientas.
  - ☑ Reutilización de soluciones aplicadas a un proyecto.

OCL

- OCL (Object Constraint Language) es un lenguaje de especificación formal fácil de leer y escribir.
- Es un lenguaje de especificación puro.
- Permite expresar restricciones semánticas del sistema que no se pueden expresar a partir de una notación grafica.
- Es un lenguaje tipado.

Restricciones en OCL

Las restricciones que podemos especificar con OCL son:

- Invariantes
- Definiciones
- Precondiciones
- Postcondiciones

Además, OCL nos permite ligar una expresión de valor inicial a una propiedad de una clase.

MDA (Arquitectura Dirigida por Modelos)

- Los modelos guían todo el proceso de desarrollo.

- Proceso de desarrollo:

5. De los requisitos se obtiene un modelo independiente de la plataforma (PIM) .
6. Luego este modelo es transformado en uno o más modelos específicos de la plataforma (PSM).
7. Y, finalmente, cada PSM es transformado en código.

MDA incorpora la idea de transformaciones entre modelos, por lo que se necesitan herramientas para automatizar esta tarea.

Modelos en MDA

- **PIM (Modelo Independiente de la Plataforma)**
  - Modelo de sistema de alto nivel
  - Representa la estructura, funcionalidad y restricciones del sistema, sin aludir a una plataforma determinada.
  - Debe ser creado íntegramente por el desarrollador.

- **PSM (Modelo Específico de la Plataforma)**
  - Modelo del sistema con detalles específicos de la plataforma en la que será implementado.
  - Representa el mismo sistema que el PIM pero a distinto nivel de abstracción.

Refinamientos

- Desarrollo de un diseño más detallado que una especificación abstracta, a través de una sucesión de pasos matemáticos.
- Para verificar si existe un refinamiento se deben satisfacer tres condiciones de refinamiento: inicialización, aplicabilidad y correctitud.
- En ePlatero, estas condiciones se definen completamente en términos de UML y OCL.
- Para especificar la relación de refinamiento entre elementos del modelo podemos utilizar un artefacto llamado “Abstraction” de UML.

Condiciones de refinamiento

Supongamos que tenemos una clase VueloA que es refinada en VueloC. Las condiciones que se debe satisfacer para decir que “existe” un refinamiento entre estas clases son:

- **Inicialización:**

VueloC.allInstances()-> forAll (vueloC | vueloC.isInit() implies
(VueloA.allInstances()-> exists (vueloA | vueloA.isInit() and
vueloA.mapping(vueloC))))

- **Aplicabilidad:**

VueloA.allInstances()-> forAll (vueloA |
VueloC.allInstances()-> forAll (vueloC |
vueloA.mapping(vueloC) implies (vueloA.preOP$_{ai}$() implies vueloC.preOP$_{ci}$())))

- **Correctitud:**

VueloA.allInstances()-> forAll (vueloA |
VueloC.allInstances()-> forAll (vueloC |
VueloC.allInstances()-> forAll (vueloC_post |
vueloA.mapping(vueloC) and vueloA.preOP$_{ai}$() and vueloC_post.postOP$_{ci}$(vueloC)
implies VueloA.allInstances()-> exists (vueloA_post |
vueloA_post.mapping(vueloC_post) and vueloA_post.postOP$_{ai}$(vueloA)))))

Micromodelos de Software

- Extraer del modelo UML un número relativamente pequeño de instanciaciones (micromundo), y verificar si satisfacen las condiciones mencionadas.

- Esta estrategia es llamada micromodelos de software.

- Se deben tener en cuenta dos consideraciones:
  - Respuesta positiva:
    - posibilidad de que la propiedad se cumple.
    - respuesta no definitiva
  - Respuesta negativa:
    - Existe al menos un mundo que viola la propiedad.
    - Respuesta definitiva.

Generación de micromundos representativos

- Escribir complejas transformaciones de modelos es propenso a errores.

- Hipótesis de Jackson
  - La respuesta positiva es alentadora.
  - La respuesta negativa es definitiva.

- Para calificar los valores de las propiedades: **The category-partition method**

Criterio de testeo para diagramas UML

- **Criterio de multiplicidad de asociaciones (AEM)**: dado un conjunto de test T y un modelo del sistema SM, T debe causar que cada par de multiplicidad representativo en el SM sea creado.

- **Criterio de generalización (GN)**: dado un conjunto de test T y un modelo del sistema SM, T debe causar que cada especialización definida en una relación de generalización sea creada.

- **Criterio de atributos de clases (CA)**: dado un conjunto de test T, un modelo del sistema SM, y una clase C, T debe causar que un conjunto de combinaciones de valores de atributos representativos (en cada instancia de la clase C) sea creado.

Adaptación de los criterios

- **AEM**: para cada asociación, debe ser cubierta cada multiplicidad representativa. Por ejemplo, si una asociación tiene multiplicidad [0..N], puede ser instanciada con la multiplicidad 0, N, y algún valor entre 0 y N.

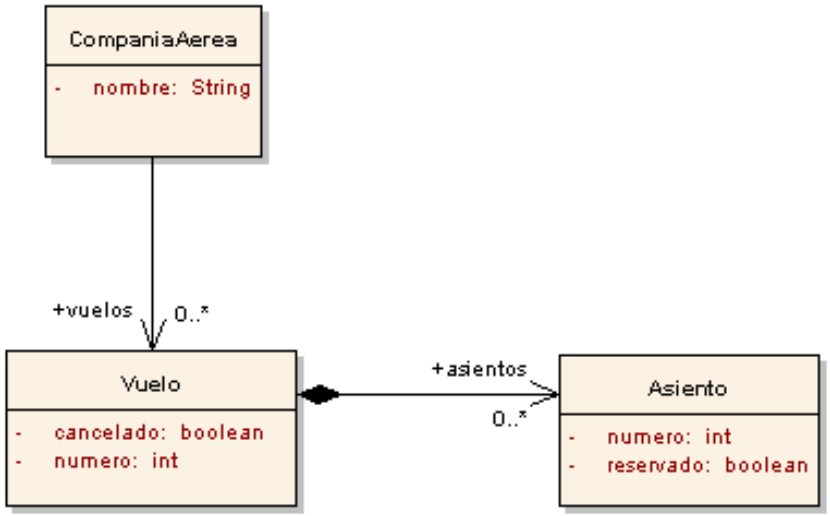
- **CA**: para cada atributo, debe ser cubierto cada valor representativo.
  - Si el tipo del atributo es simple (integer, string, bolean), tiene que ser computado un conjunto de valores representativos.
  - Si el tipo del atributo es complejo, debe ser procesado como una asociación, acorde al criterio AEM.

Generando las particiones

- **Partición:** Una partición de un conjunto de elementos es una colección de n rangos A1,…, An donde A1,.., An no se solapan y la unión de todos los subconjuntos forman el conjunto inicial. Estos subconjuntos son llamados rangos**.**

- Las particiones son una forma práctica para seleccionar los valores representativos.

Ejemplo



Modelo

CompaniaAerea::nombre	{“”}, {“NombreCompania”}
CompaniaAerea::#vuelos	{0}, {1}, {2..MaxInt}
Vuelo::cancelado	{true},{false}
Vuelo::numero	{0}, {1}, {2..MaxInt}
Vuelo::#asientos	{0}, {1}, {2..MaxInt}
Asiento::numero	{0}, {1}, {2..MaxInt}
Asiento::reservado	{true}, {false}

Particiones generadas para el modelo

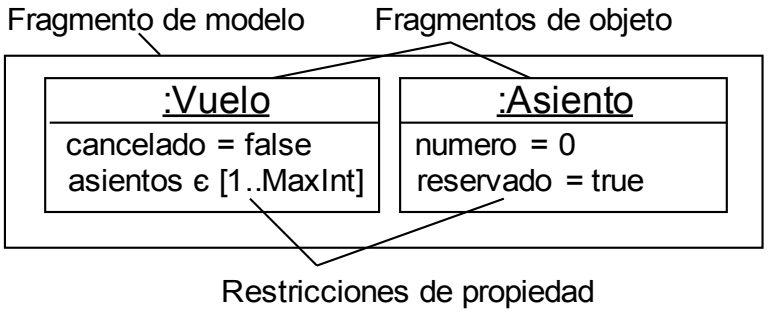
Combinando las particiones

- La cobertura independiente de valores y multiplicidades de cada propiedad del modelo de entrada no es suficiente.
- Se vuelve necesario combinar las particiones.
- Combinar los rangos para todas las propiedades presenta desventajas:
  - Número de combinaciones a cubrir poco realista.
  - No necesariamente es suficiente para asegurar la relevancia de los modelos de test.

Fragmentos de modelo y de objeto

- **Fragmento de modelo**: está compuesto por un conjunto de fragmentos de objeto

- **Fragmento de objeto:** está compuesto por un conjunto de restricciones de propiedades, que especifican los rangos desde los cuales deben ser tomados los valores de las propiedades de un objeto.



Ejemplo de fragmento de modelo

Criterios de test

- **Criterio de Test**: especifica un conjunto de fragmentos de modelo para un modelo de entrada.

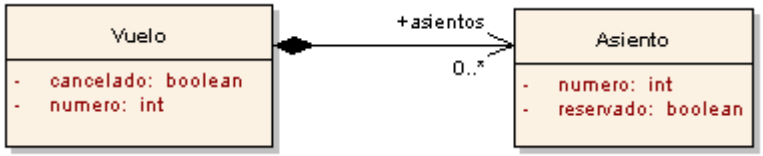
- Los fragmentos de modelo deben cumplir:
  - **Regla 1 - Cobertura de clase**: cada clase concreta debe ser instanciada en al menos un fragmento de modelo.
  - **Regla 2 - Cobertura de rango**: cada rango de cada partición para todas las propiedades del modelo debe ser usado en al menos un fragmento de modelo.

Criterios de test

- Criterios de cobertura clase por clase
  - **OneRangeCombination**: cada rango para cada propiedad de una clase necesita ser usada en al menos un fragmento de objeto.
  - **AllRangesCombination**: todas las combinaciones posibles de rangos para las propiedades de una clase deben aparecer en un fragmento de objeto

Criterio OneRangeCombination

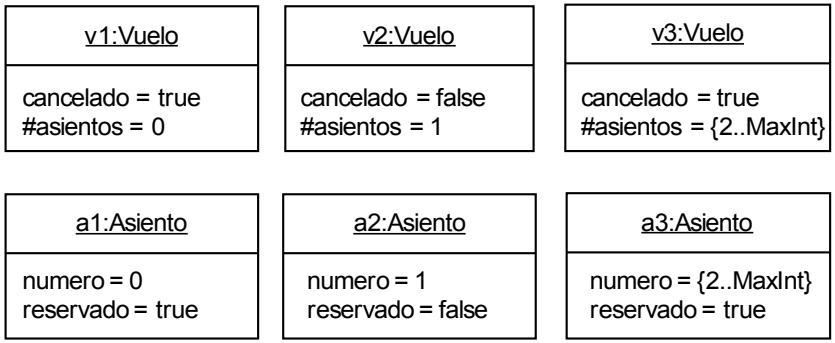
- Cada rango para cada propiedad de una clase necesita ser usada en al menos un fragmento de objeto. Ejemplo:



Modelo

Vuelo::cancelado	{true}, {false}
Vuelo::#asientos	{0}, {1},{2..MaxInt}
Asiento::numero	{0}, {1}, {2..MaxInt}
Asiento::reservado	{true}, {false}

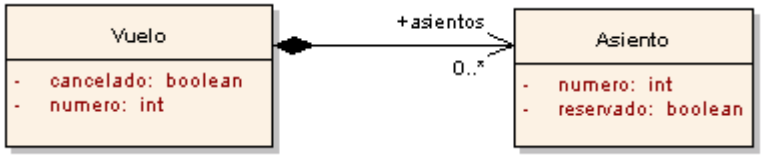
Particiones Generadas



Posibles fragmentos de modelo

Criterio AllRangesCombination

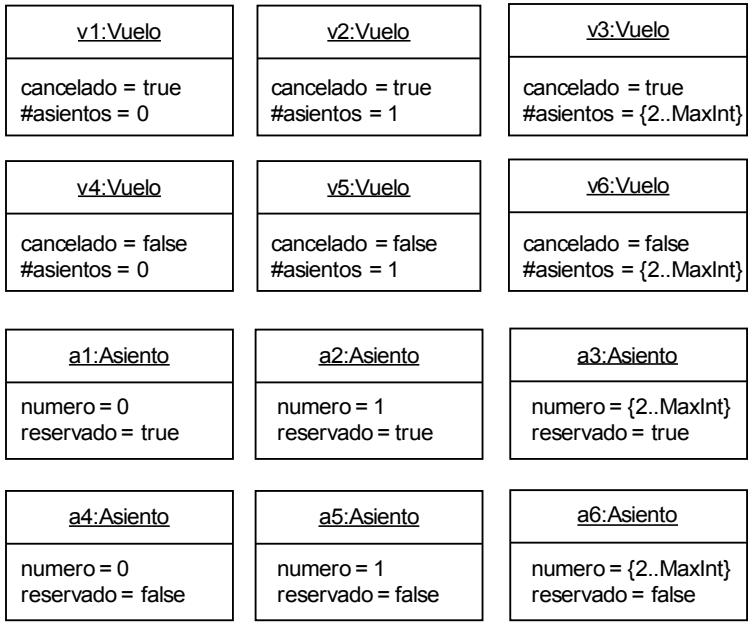
- Todas las combinaciones posibles de rangos para las propiedades de una clase deben aparecer en un fragmento de objeto. Ejemplo:



Modelo

Vuelo::cancelado	{true}, {false}
Vuelo::#asientos	{0}, {1},{2..MaxInt}
Asiento::numero	{0}, {1}, {2..MaxInt}
Asiento::reservado	{true}, {false}

Particiones Generadas



Posibles fragmentos de modelo

Comparación de criterios

- Conclusiones:
  - Todos los criterios propuestos reducen perceptiblemente el número de fragmentos comparado a la estrategia de “todas la combinaciones”.
  - Para todos los criterios que requieran el producto cartesiano de rangos, el número de fragmentos sigue siendo absolutamente alto.

Criterio	Nº de Fragmentos generados
OneRangeCombination	6
AllRangesCombination	12
Todas las combinaciones	36

Comparación de criterios

Conclusiones

- La técnica de micromodelos resulta válida para el testo de transformaciones de modelos (Hipótesis de Jackson)

- Criterios de test basados en cobertura de elementos.

- Para generar los micromundos representativos se debe:
  - Crear particiones representativas para cada una de las propiedades de los objetos del modelo.
  - Combinar los rangos de las particiones creadas utilizando la estrategia OneRangeCombination o la estrategia AllRangesCombination.
  - Representar las combinaciones de rangos realizadas en fragmentos de objetos pertenecientes a un fragmento de modelo (estructura del micromundo).
  - A partir del fragmento de modelo generar el micromundo representativo.

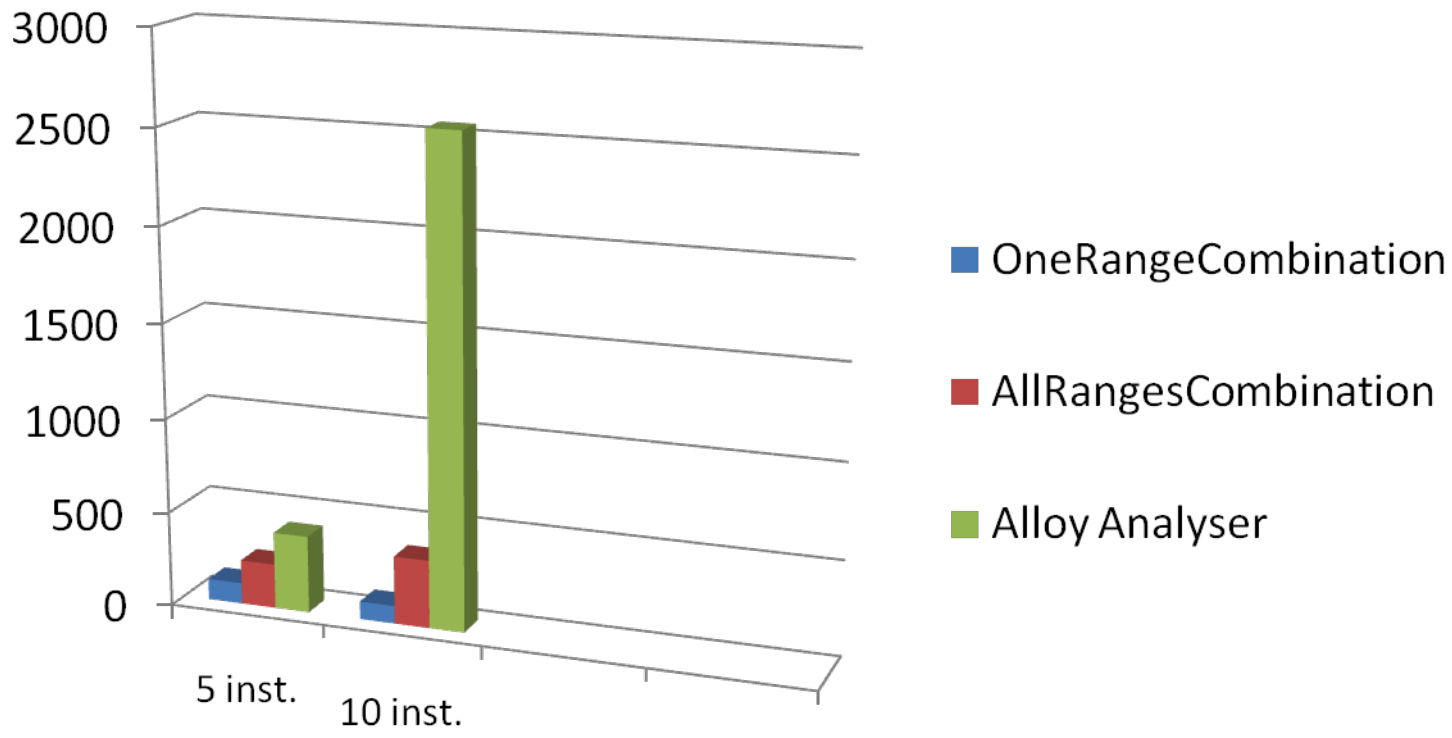
ePlatero

- ePlatero es una herramienta CASE educativa que soporta el proceso de desarrollo de software dirigido por modelos utilizando notación gráfica con fundamento formal.

- ePlatero nos permite:
  - Construir modelos.
  - Escribir refinamientos.
  - Escribir restricciones OCL para desambiguar el modelo y evaluar refinamientos.
  - Generar micromundos representativos para testar las transformaciones de modelos.

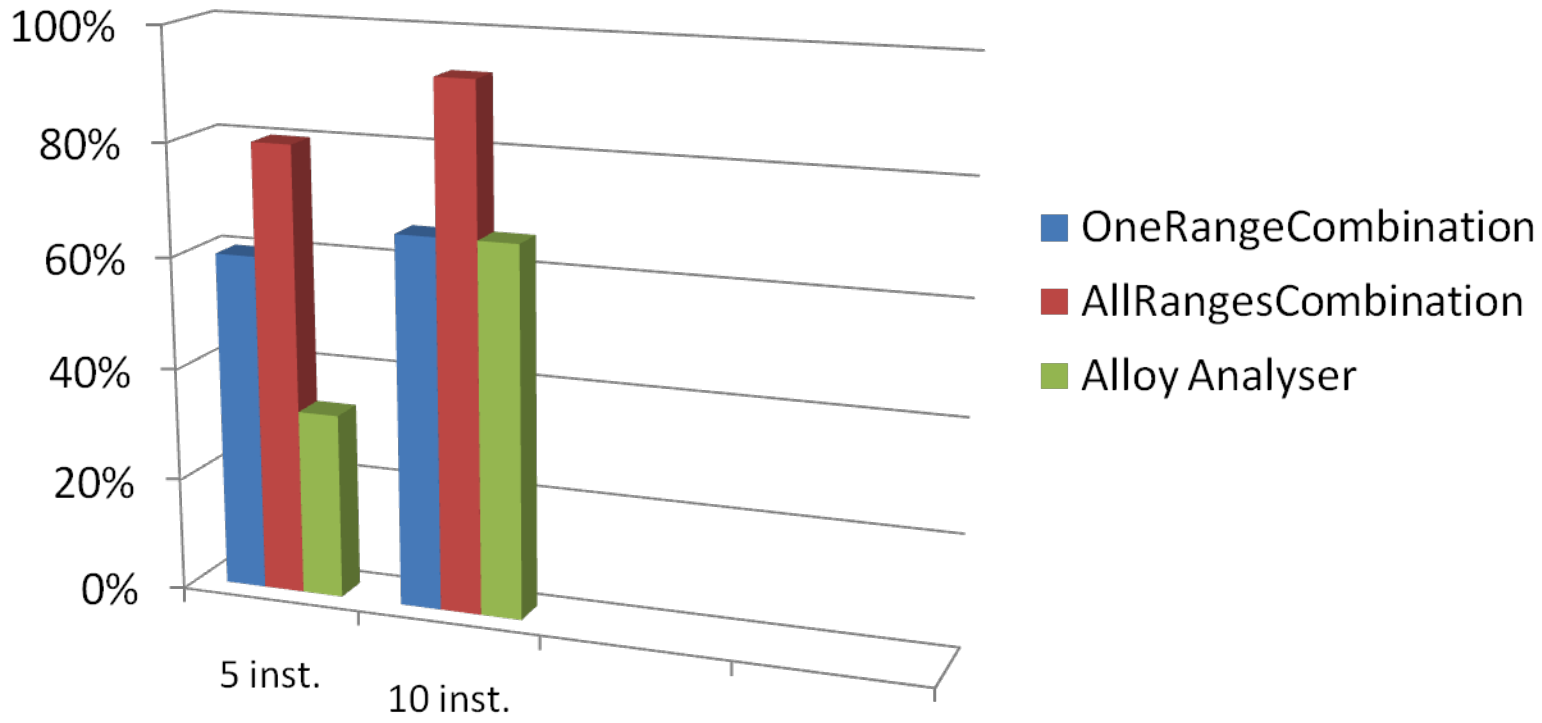
Comparación con Alloy

- Comparación de tiempos promedio entre Alloy y ePlatero



Comparación con Alloy

- Comparación de respuestas correctas en micromundos con errores



Muchas Gracias.-