Generación de código Ada con UML
MARTE

Generación de código Ada para aplicaciones embebidas y de tiempo real desde modelos dinámicos UML

Alejandro Pérez Ruiz, Julio Medina

XIII Jornada Técnica Ada-Spain

Abril, 2013

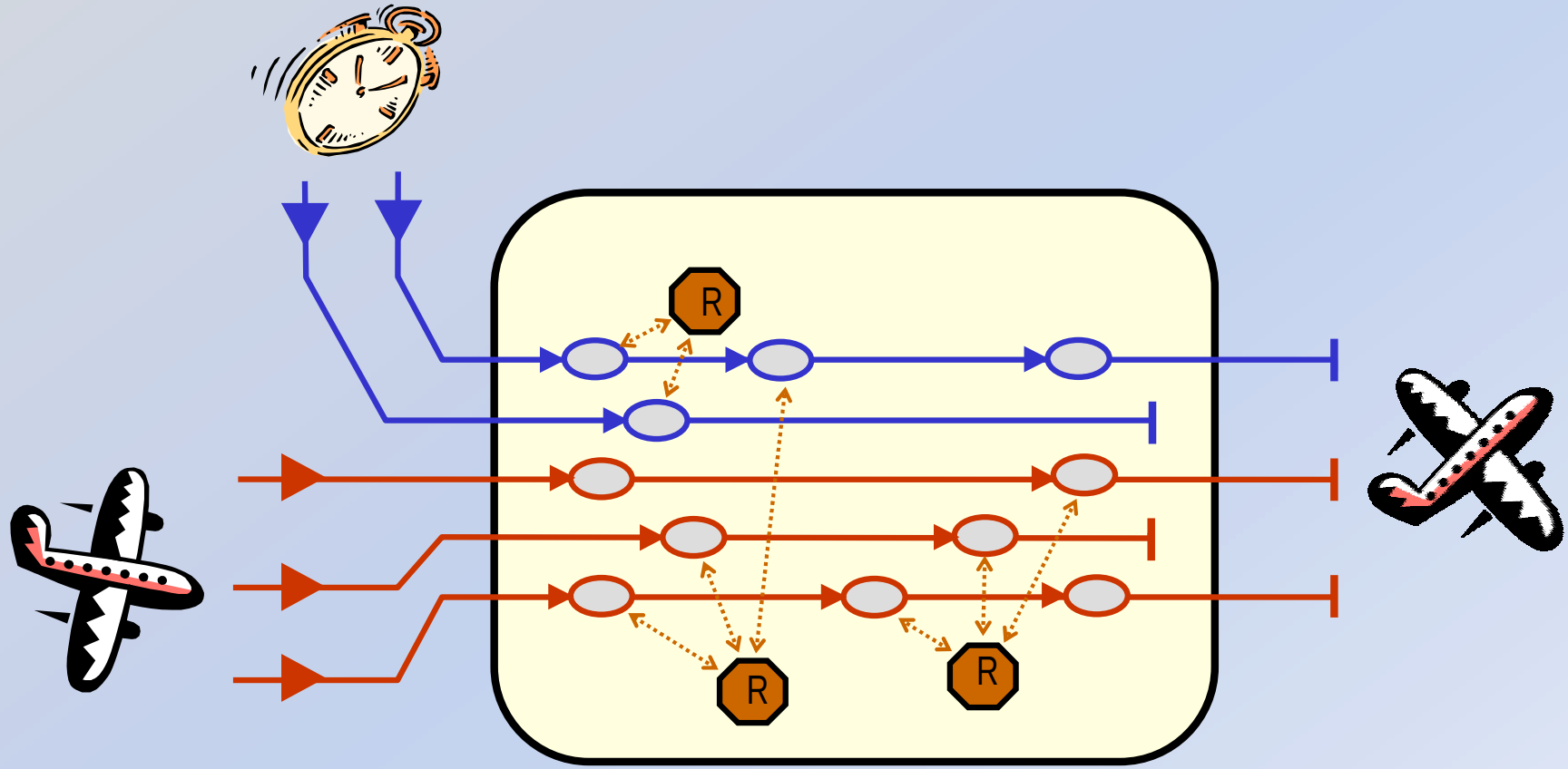
Índice

- Introducción
- Marco del proyecto
- Generación de código
- Desarrollo e implementación del generador de código
- Casos de estudio
- Despliegue
- Conclusiones y trabajos futuros

Índice

Sistemas de Tiempo Real

- Son sistemas informáticos que mantiene una relación interactiva y temporizada con su entorno.



Desarrollo de aplicaciones de Tiempo Real

Desarrollo basado en transacciones	Desarrollo basado en orientación a objetos
• La aplicación se concibe como un conjunto de transacciones (*end_to_end_flow*) que se ejecutan concurrentemente en la plataforma en que se encuentra instalada. • El diseño se basa en organizar la ejecución de las actividades que componen las transacciones para que finalicen en los plazos temporales que tienen asignados.	• La aplicación se simplifica a través de la abstracción, eliminando las peculiaridades del ente o fenómeno bajo estudio. • Se diseña la aplicación utilizando el paradigma de orientación a objetos, identificando tipos de objetos en su dominio de aplicación a través de las clases.

Desarrollo de aplicaciones de Tiempo Real

Desarrollo basado en transacciones	Desarrollo basado en orientación a objetos
• La aplicación se concibe como un conjunto de transacciones (*end_to_end_flow*) que se ejecutan concurrentemente en la plataforma en que se encuentra instalada. • El diseño se basa en organizar la ejecución de las actividades que componen las transacciones para que finalicen en los plazos temporales que tienen asignados.	• La aplicación se simplifica a través de la abstracción, eliminando las peculiaridades del ente o fenómeno bajo estudio. • Se diseña la aplicación utilizando el paradigma de orientación a objetos, identificando tipos de objetos en su dominio de aplicación a través de las clases.

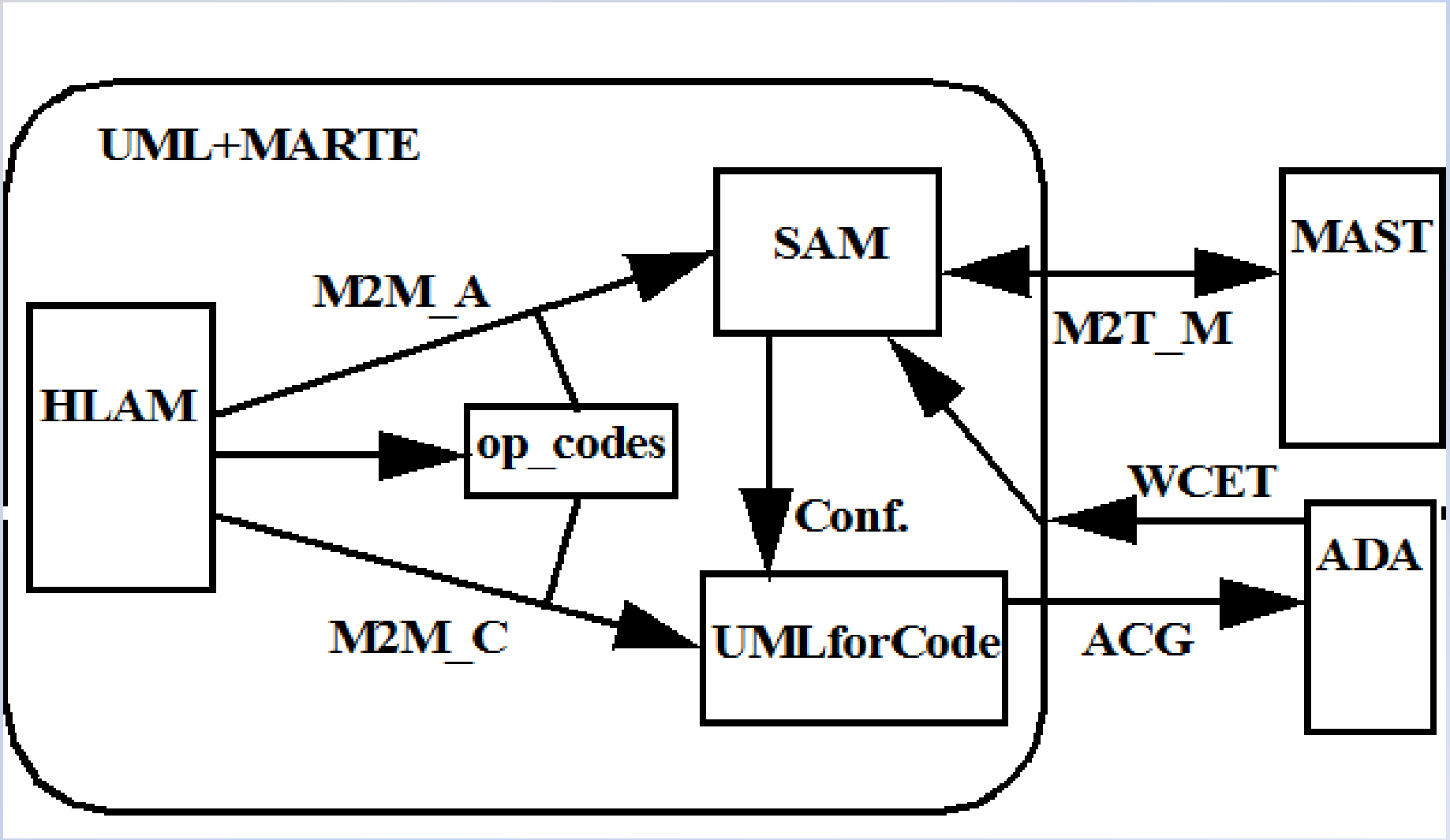
Real Time Unit (RtUnit)

El perfil MARTE de UML

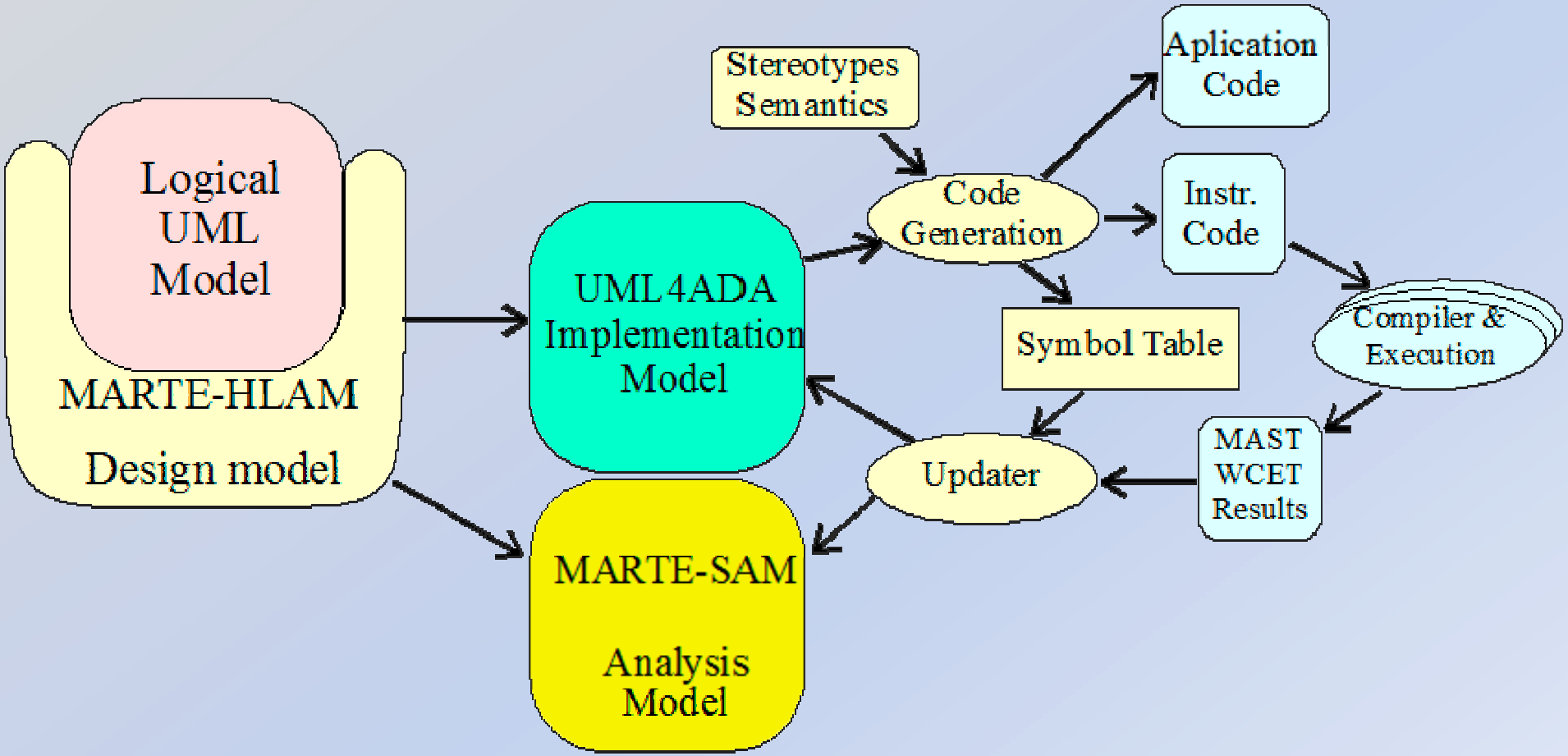
- Los perfiles: mecanismos para extender UML a dominios específicos.
- El perfil MARTE (*Modeling and Analysis of Real-Time Embedded Systems*): análisis y modelado de sistemas embebidos y de tiempo real.
- HLAM (*High-Level Application Modeling*): diseño de alto nivel de aplicaciones de tiempo real.
  - Objeto activo de tiempo real (*RtUnit*)
  - Recurso pasivo protegido (*PpUnit*)
  - Características de tiempo real (*RtFeatures*)
  - Servicios de tiempo real (*RtService*)

Índice

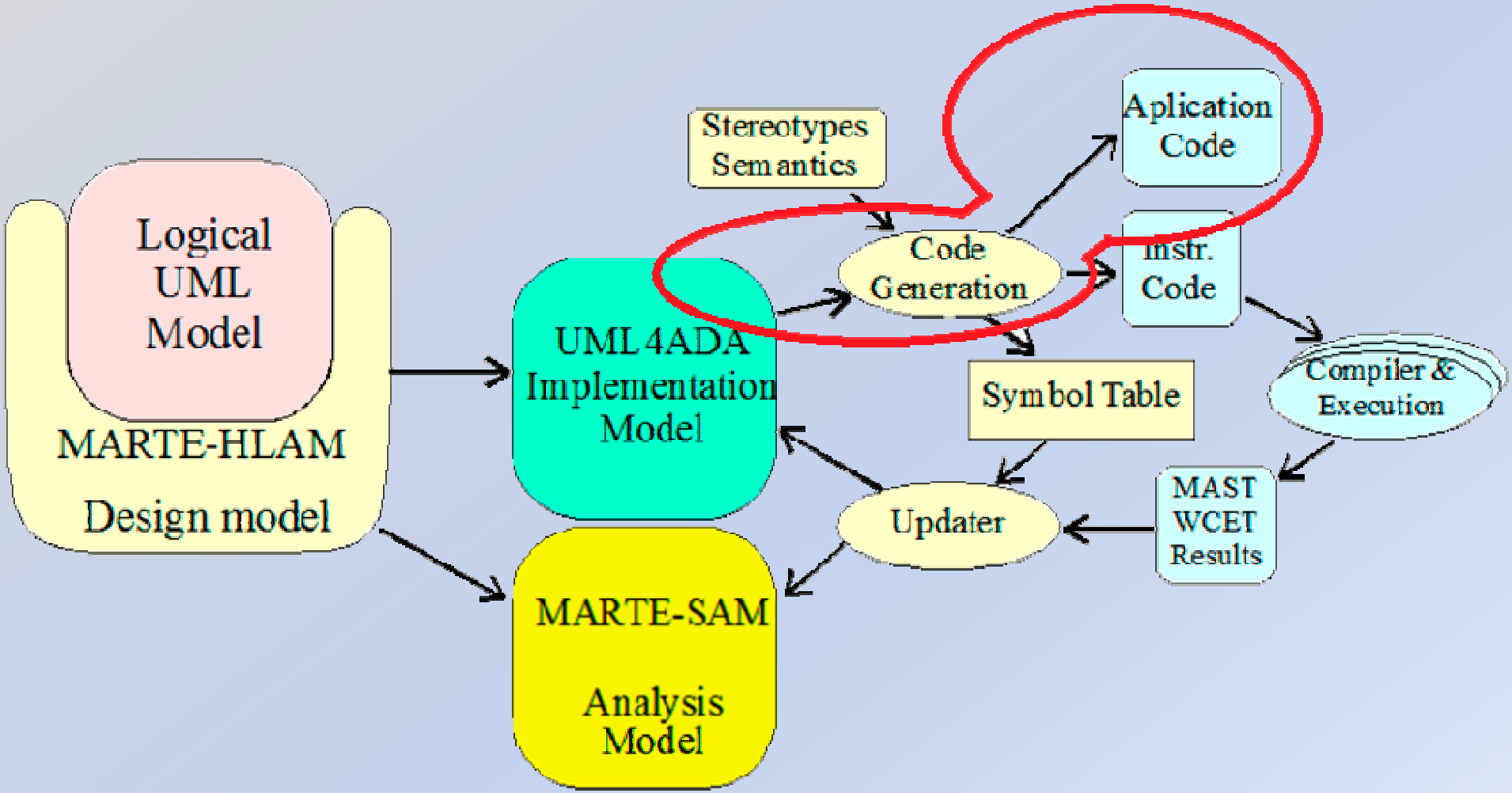
Marco del proyecto (I)



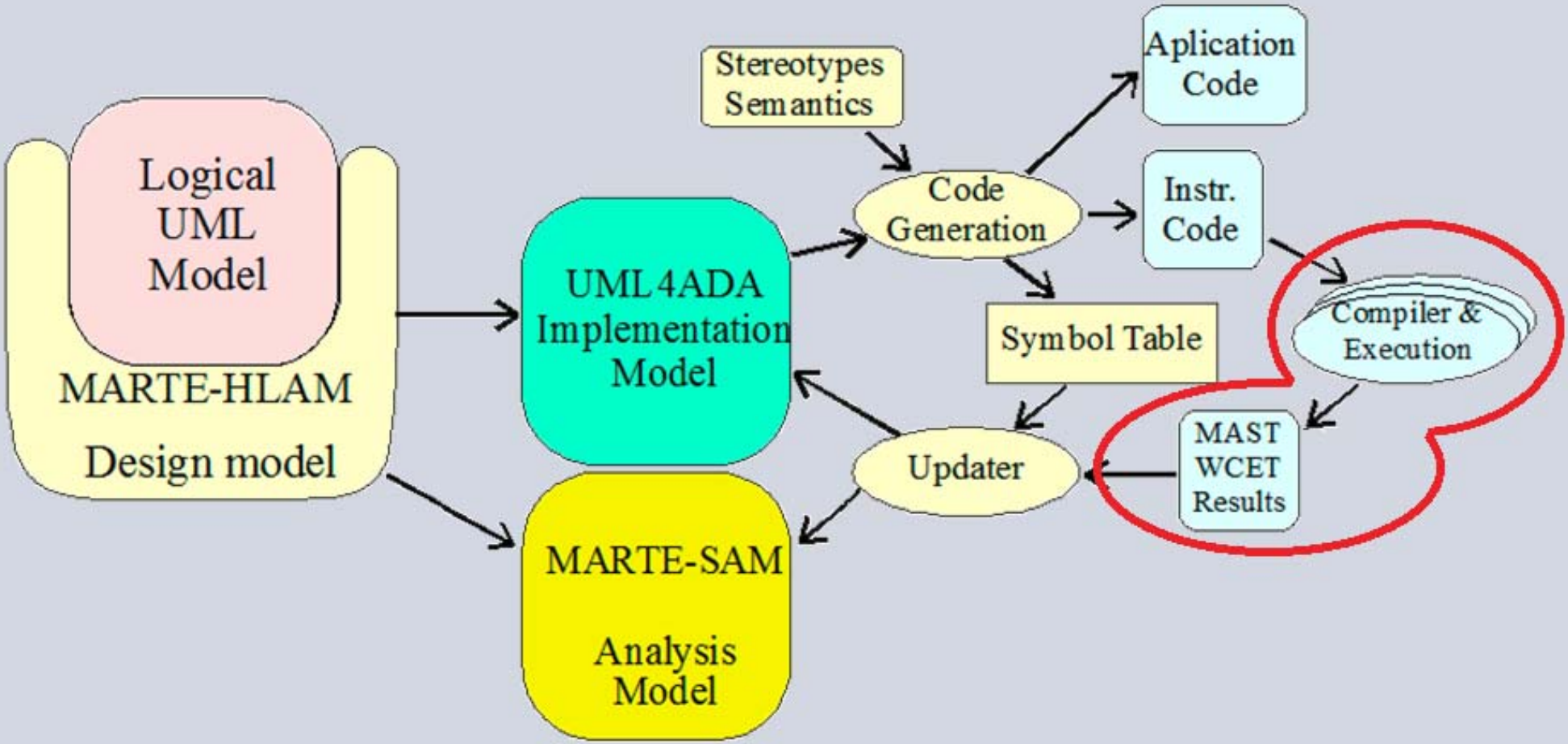
Marco del proyecto (II)



Marco del proyecto (II)



Marco del proyecto (III)



Necesidad de un nuevo generador

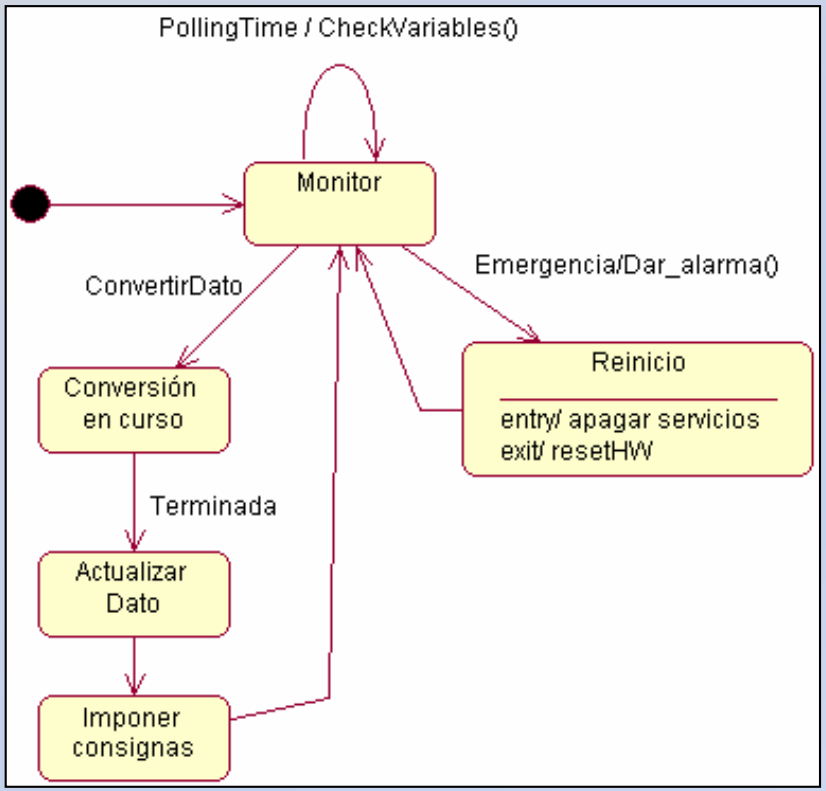
- Los principales generadores de código presentan inconvenientes para el desarrollo de sistemas de tiempo real:

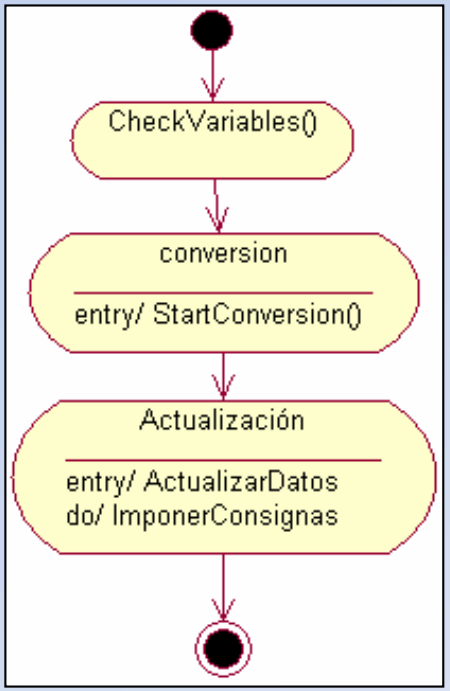
1. La mayoría generan solo el código estructural de la aplicación.

2. Los pocos que son capaces de generar el código de comportamiento de las operaciones lo hacen a través de los diagramas de estados.

Diagramas de estado y de actividad

- Diagrama de estados: describe el comportamiento del objeto mediante sus estados y las transiciones entre ellos en respuesta a eventos.
- Diagrama de actividad: muestra un flujo de acción que describe la operación del objeto en una situación determinada.





Índice

- Introducción
- Marco del proyecto
- Generación de código
- Desarrollo e implementación del generador de código
- Casos de estudio
- Despliegue
- Conclusiones y trabajos futuros

Cómo crear un generador de código

- Las diversas tecnologías existentes comparten el mismo proceso:
  1. Se utiliza algún tipo de representación de un metamodelo (en nuestro caso, UML).
  2. Hay alguna clase de lenguaje imperativo que nos permite escribir generadores de código.

- De entre las herramientas y tecnologías analizadas se decidió utilizar **Acceleo**.

¿Por qué Acceleo?

- Utiliza un lenguaje basado en plantillas denominado MOFM2T (MOF Model to Text Transformation Language) aprobado por la OMG como estándar.

- La herramienta Marte2Mast desarrollada en el contexto de este trabajo emplea esta tecnología.

Funcionamiento de MOFM2T

```
[template public generateSpecificationTask(class : Class)]
[file(getNameFile(class).concat('s').concat('.ads'),
false, 'UTF-8')]
package [class.name.concat('s')/] is

[if((class.attribute->size()=0)._and(class.ownedOperation
                                      ->size()=0))]
       task type Task_[class.name/];
[else]
…
[/file]
[/template]
```

Índice

- Introducción
- Marco del proyecto
- Generación de código
- Desarrollo e implementación del generador de código
- Casos de estudio
- Despliegue
- Conclusiones y trabajos futuros

Elementos UML soportados por el generador

Estructurales (Diagramas de clases):

- Clases
  1. Clases abstractas.
  2. Clases activas.
  3. Clases con el estereotipo *PpUnit* del perfil MARTE.
- Paquetes.
- Relaciones entre clases: dependencia, asociación y generalización.
- Interfaces.

Comportamiento (Diagramas de actividad):

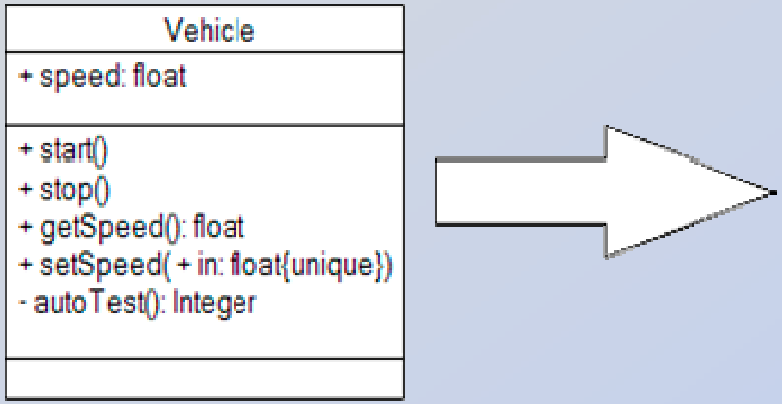
- Nodo inicial
- Acciones opacas
- Nodos de decisión
- Control flows
- Nodo final de actividad

Transformaciones de UML a código Ada (I)

- Surge la necesidad de escribir una serie de reglas de equivalencia.
  - Teniendo en cuenta los estándares de UML y Ada se debe alcanzar una correspondencia semántica entre ambos.

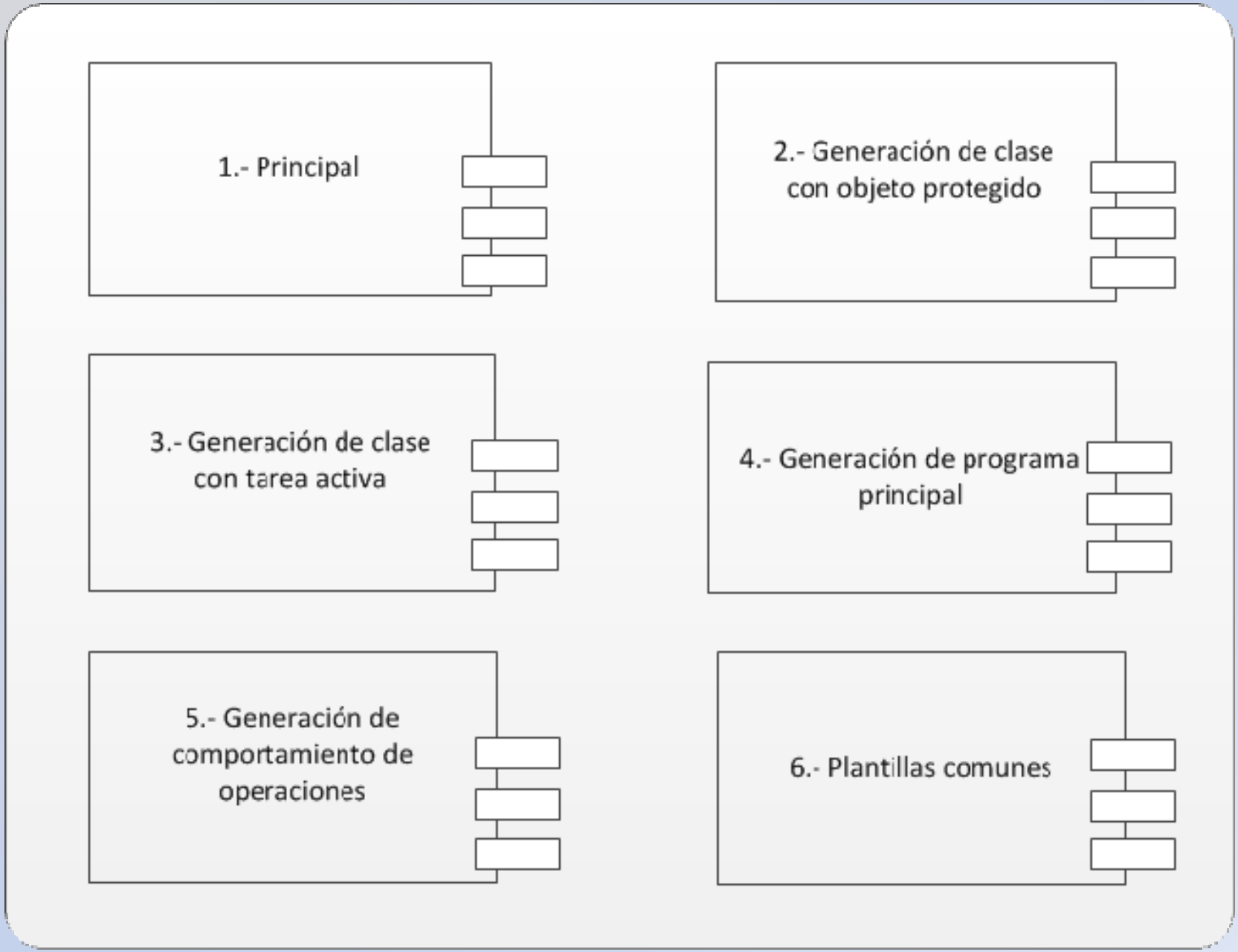
- En algunos casos la equivalencia no es directa, por ejemplo: encapsulación de clases, algunos tipos de visibilidad o definición de tareas entre otros.

Transformaciones de UML a código Ada (II)




```
package Vehicles is
       type Public_Part is abstract tagged record
                     speed : float;
       end record;

       -- Complete_type

       type Vehicle is  new Public_Part with private;

       type P_Vehicle is access Vehicle;

       -- Public methods

       procedure start (Self : in out Vehicle'Class );
       procedure stop (Self : in out Vehicle'Class );
       function getSpeed (Self : Vehicle'Class ) return float;
       procedure setSpeed (Self : in out Vehicle'Class; speed : in float);

private
       type Vehicle is new Public_Part with record
              null;
       end record;
end Vehicles;
```

Implementación y estructuración del generador



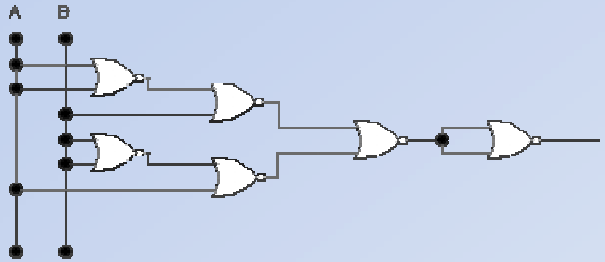
Índice

- Introducción
- Marco del proyecto
- Generación de código
- Desarrollo e implementación del generador de código
- Casos de estudio
- Despliegue
- Conclusiones y trabajos futuros

Casos de estudio

- Problema de la cena de los filósofos
  - Elementos concurrentes: Tareas y objetos protegidos.

- Simulador lógico combinacional
  - Elementos básicos de la orientación a objetos.



- Productor – Consumidor con requisitos de tiempo real
  - Caso de estudio para la medición de tiempos de ejecución

Índice

- Introducción
- Marco del proyecto
- Generación de código
- Desarrollo e implementación del generador de código
- Casos de estudio
- Despliegue
- Conclusiones y trabajos futuros

Despliegue

- Empaquetado y creación de un plugin para el entorno de desarrollo Eclipse.
- Manuales.
- Página web: mast.unican.es/umlmast/uml2ada/

Índice

- Introducción
- Marco del proyecto
- Generación de código
- Desarrollo e implementación del generador de código
- Casos de estudio
- Despliegue
- Conclusiones y trabajos futuros

Conclusiones

- Generador de código funcional empaquetado en forma de plugin para Eclipse.
  - Capaz de generar el código de la parte estructural y comportamiento del sistema.
- Definición de una serie de reglas de equivalencia entre UML y Ada.
- Publicación internacional (WATERS 2012, Pisa (Italia)).

Trabajos futuros

- Ampliar el número de características UML soportadas.
- Utilizar el lenguaje OCL para especificar restricciones sobre los modelos de entrada.
- Añadir la capacidad de instrumentar el código para obtener cotas estadísticas del tiempo de ejecución de las operaciones del sistema.

Gracias por su atención

¿Preguntas?