Desarrollo de Software para Sistemas Empotrados

1. Introducción
2. Plataformas para sistemas empotrados
3. *Especificación y análisis de requisitos software en sistemas empotrados*
4. Diseño arquitectónico en sistemas empotrados
5. Implementación software de sistemas empotrados

Desarrollo de Software para Sistemas Empotrados

3. Especificación y análisis de requisitos software en sistemas empotrados

- Introducción
- Especificación de requisitos en sistemas reactivos
- UML/MARTE
- Use Case Maps
- RDAL/AADL

3.1 Introducción

Las técnicas de especificación y análisis de requisitos en sistemas empotrados son similares a las de los sistemas generales, con las siguientes diferencias

- Tenemos más requisitos no funcionales
  - requisitos temporales
  - requisitos derivados de la limitación de recursos: memoria, energía, ...
  - requisitos de fiabilidad y tolerancia a fallos
- Los requisitos tienden a ser expresados en forma reactiva
  - concurrencia, incluyendo sincronización y comunicación
  - gestión de eventos
  - diagramas de estados, diagramas de secuencia, diagramas de actividad, ...
- Al estar el sistema compuesto por muchos subsistemas aparece la necesidad de una jerarquía de requisitos:
  - de comportamiento y estructural

3.2 Especificación de requisitos en sistemas reactivos

UML

- Diagramas de secuencia
- Diagramas de estados
- Diagramas de actividad

MARTE

- The UML Profile for MARTE: Modeling and Analysis of Real-Time and Embedded Systems

Use Case Maps

RDAL/AADL

3.3 UML/MARTE

Ver presentación por Julio L. Medina Pasaje

3.4 Use case maps (UCM)

Son diagramas para modelar escenarios de uso de un sistema

- fácilmente integrables en un proceso MDD

El objetivo es descubrir funciones que se requerirán en la construcción del sistema

- elicitación de requisitos

Ventaja: muestra en el mismo diagrama:

- entidades: actores y sistemas
- acciones realizadas y su secuencia

Ventaja: tienen más información que los casos de uso de UML

- éstos deben completarse con diagramas de actividad o secuencia

Estandarización

Los UCM son parte de la Notación de Requisitos de Usuario (User Requirements Notation, URN

- Estándar de la International Telecommunication Union, ITU-T Z.150 series:
  - Z.150 : URN - Language requirements and framework (2011)
    - https://www.itu.int/rec/T-REC-Z.150/en/
  - Z.151: URN – Language definition (2018)
    - https://www.itu.int/rec/T-REC-Z.151/es

URN

URN soporta la descripción de requisitos no funcionales (goals) (non-functional requirements) y de requisitos funcionales expresados mediante escenarios

URN formaliza e integra dos notaciones:

- Goal-oriented Requirements Language (GRL)
- Use Case Maps (UCMs)

Herramienta

La herramienta *jUCMNav* integrada en "Eclipse modeling tools" permite editar y ejecutar/simular diagramas UCM

https://github.com/JUCMNAV/projetseg-update/wiki

Instalación

- Descargar https://bit.ly/jUCMNav900 (fichero .jar) y colocarlo en el directorio eclipse/dropins
- Arrancar Eclipse

Uso de *jUCMNav*

- Crear un proyecto de tipo General->Project
- Desde el proyecto
  - File->New->Other (Ctrl-N)
  - y seleccionar jUCMNav -> Use Case Map / GRL Graph

Elementos de un UCM

Un diagrama UCM contiene caminos (*paths*) a los que se añaden entidades (actores o agentes/sistemas)

Una entidad es una caja que contiene las acciones realizadas

- se sitúan en un camino que representa la secuencia de acciones (llamada *escenario*)
- el camino puede tener bifurcaciones (`fork`) o confluencias (`join`) con determinadas condiciones
- el `fork` puede ser:
  - `and`, con probabilidades en cada rama
  - `or`, con condiciones asociadas a cada rama
- el camino puede tener *puntos de espera* con condiciones disparadas por eventos que vienen de otros caminos
- existen *temporizadores* que son puntos de espera con timeout

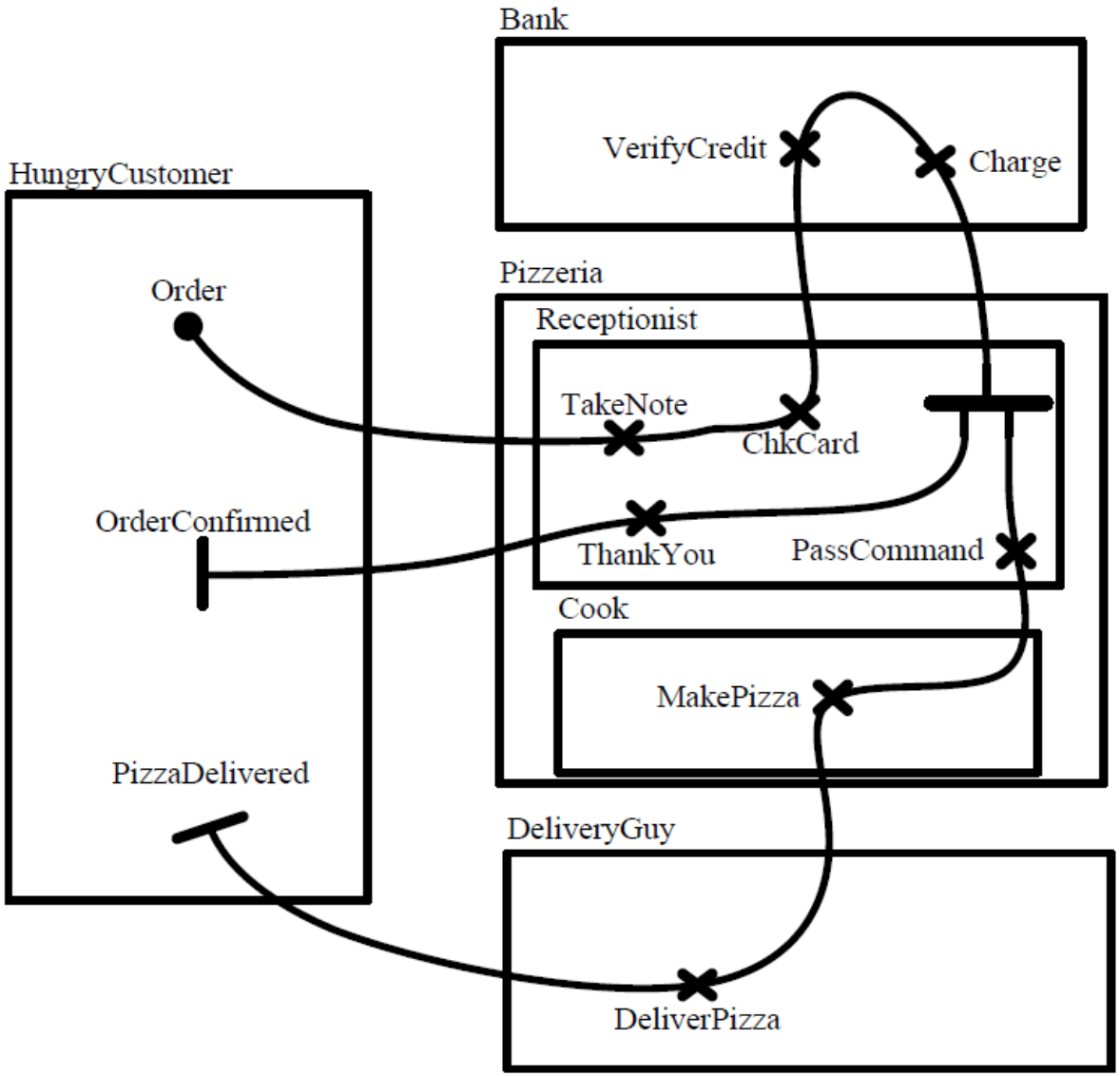
Elementos de un UCM (cont.)

Los caminos pueden tener asociadas responsabilidades o acciones (*steps*), marcados con una **X**

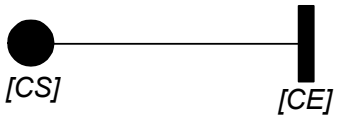
Las llamadas a otros casos o subcasos de uso se representan con rombos

- esto es particularmente útil para modelar casos de excepción
- y para establecer jerarquías de UCMs

Ejemplo de un UCM

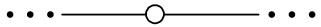


Elementos de un UCM

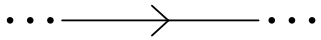


Path with Start Point with Precondition CS and End Point with Postcondition CE

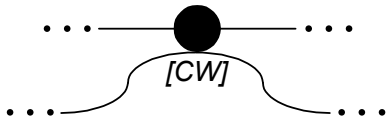
Responsibility



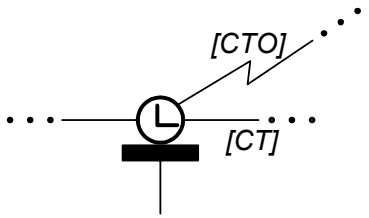
Empty Point



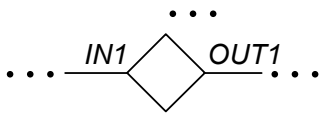
Direction Arrow



Waiting Place with Condition and Asynchronous Trigger

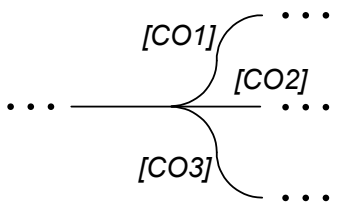


Timer with Timeout Path, Conditions, and Synchronous Release

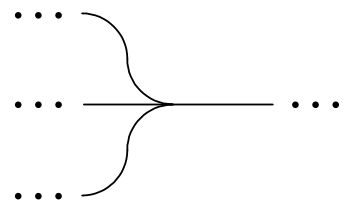


Static Stub with In-Path ID and Out-Path ID

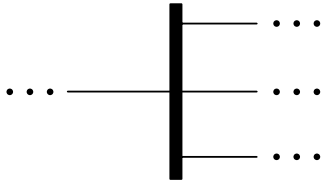
Dynamic Stub with In-Path ID and Out-Path ID



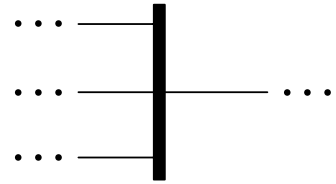
Or-Fork with Conditions



Or-Join



And-Fork



And-Join

Components:

Team

Process

Object

Agent

Actor

Protected

Especificación de requisitos temporales

La versión actual de UCM permite una especificación limitada de requisitos temporales

- periodos: atributo *workload* en el *StartPoint*
  - Se puede poner el tipo de activación (*arrival pattern*) y el valor
    - arrival pattern = periodic, poissonPDF (aperiódico), ...
  - Se puede poner una etiqueta (*label*) de tipo "`T=valor`" en la precondición, para que se refleje en el diagrama
- plazos: expresión en las *postcondiciones* del *EndPoint*
  - Se puede poner una etiqueta (label) de tipo "`D=valor`" en la postcondición, para que se refleje en el diagrama

Transformación de modelos

En etapas posteriores del desarrollo los UCM se pueden transformar (añadiendo la información adicional necesaria) en:

- Diagramas de secuencia UML
- Diagramas de máquinas de estados SDL
- Diagramas de máquinas de estados UML

La herramienta jUCMNav permite exportar el modelo a diversos formatos:

- Test description language
- URN Z151
- DOORS
- Core Scenario Model
- ...

Ejercicio 3.1

- Dentro del proceso de análisis de requisitos, generar un diagrama UCM para los requisitos del sistema SCADA

Ejecución de escenarios en diagramas UCM

Es posible crear escenarios en los diagramas UCM, y ejecutarlos

- esto permite añadir requisitos funcionales y analizar si están bien descritos

Los escenarios se agrupan en grupos de escenarios

En el diagrama se pueden crear variables

- enteras
- booleanas
- enumeradas

Las variables representan los principales elementos de comunicación que se trasladarán al diseño

Papel de las variables

Cada responsabilidad puede tener asignado un código compuesto por instrucciones con las que se pueden consultar las variables y modificarlas

Las bifurcaciones tienen condiciones que se expresan según el valor de las variables

Los puntos de comienzo pueden tener precondiciones expresadas en función de las variables

Los puntos de finalización pueden tener postcondiciones similares

Ejecución de escenarios

Para ejecutar los escenarios se definen:

- valores iniciales de las variables
- precondiciones
- puntos de comienzo

Ver ejemplo de *tratamiento de imágenes*

- Ejecutar e inspeccionar los diferentes escenarios
- Hacer que funcione el escenario `CambiarAModoAutomatico`
- Añadir un escenario para probar el caso de la pulsación de una tecla incorrecta

3.5 RDAL/AADL

AADL: Architecture Analysis & Design Language

- desarrollado por SAE International (Society of Automotive Engineers)
- aprobado como SAE Standard AS-5506 en noviembre de 2004
- la última version es la AS-5506D, de abril de 2022

AADL permite la especificación, análisis integración y generación de código de sistemas distribuidos de tiempo real

- incluyendo aspectos de comportamiento temporal, alta criticidad, seguridad, planificabilidad, tolerancia a fallos, security, etc.

Los objetivos principales son

- poder analizar diseños de sistemas antes de la implementación
- desarrollo basado en modelos (MDD) a lo largo del ciclo de vida

Beneficios de AADL

Reducir coste de desarrollo y mantenimiento:

- proporciona un estándar con sintaxis y semántica precisa
- habilidad para modelar arquitecturas complejas, multi-contratista
- facilita la evaluación temprana y la evaluación de cambios (seguridad, planificabilidad, ...)
- facilita el análisis de la estructura del sistema así como su comportamiento dinámico
- vehículo para producir arquitecturas de referencia y soportar desarrollo basado en componentes

Inconvenientes (hasta ahora)

No permite especificar en el dominio del problema

Necesita extensiones para describir requisitos software

- RDAL (2013)
- ReqSpec (2016)
  - requirement specification language
  - es un lenguaje textual basado en RDAL
- En proceso de estandarización como una parte de AADL
- Soportado en la herramienta RDALTE:
  - https://mem4csd.telecom-paristech.fr/blog/index.php/rdal/

RDAL

Requirements Definition and Analysis Language

- puede ser utilizado con otros lenguajes de diseño

Dispone de puntos de trazabilidad para enlazar elementos de los requisitos con

- modelos UCM, elementos del diseño, modelos funcionales, modelos reactivos, actividades de verificación, ...

Un conjunto bastante completo de lenguajes de desarrollo:

- UCM + RDAL/ReqSpec + AADL

ReqSpec

RDAL está soportado en un lenguaje de especificación textual

- ReqSpec:
  - https://resources.sei.cmu.edu/asset_files/TechnicalReport/2016_005_001_464378.pdf

Bibliografía

[1] "Embedded System Design". Peter Marwedel. Springer, 2010

[2] MARTE: http://www.omg.org/omgmarte/

[3] "UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems". Version 1.1. OMG Document Number: formal/2011-06-02. http://www.omg.org/spec/MARTE/1.1

[4] "Combining Requirements, Use Case Maps and AADL Models for Safety-Critical Systems Design". Dominique Blouin and Holger Giese. 2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA).

[5] AADL: http://www.aadl.info/

[6] "Modeling and Analysis of Real-Time and Embedded Systems with UML and MARTE". Bran Selic and Sébastien Gérard. Elsevier 2014.

[7] "Embedded Systems: Analysis and Modeling with SysML, UML and AADL". Fabrice Kordon, Jérôme Hugues, Agusti Canals, Alain Dohet. John Wiley & Sons, 2013

[8] UCM Home page: https://github.com/JUCMNAV/projetseg-update/wiki

Bibliografía (cont.)

[9] UCM Tutorials: https://github.com/JUCMNAV/projetseg-update/wiki/JUCMNavTutorials
https://github.com/JUCMNAV/projetseg-update/wiki/att/jUCMNav_tutorial.zip

[10] "Modeling Languages for Requirements Engineering and Quantitative Analysis of Embedded Systems". Dominique Blouin. PhD Thesis. https://www.researchgate.net/profile/Dominique-Blouin/publication/260433525_Modeling_Languages_for_Requirements_Engineering_and_Quantitative_Analysis_of_Embedded_Systems/links/00b7d5314504052fe6000000/Modeling-Languages-for-Requirements-Engineering-and-Quantitative-Analysis-of-Embedded-Systems.pdf

[11] RDAL tool environment: https://mem4csd.telecom-paristech.fr/blog/index.php/rdal/