Specification of Distributed and Embedded Systems

MDE, Distributed Components & Specification Environments

Eric Madelaine
eric.madelaine@sophia.inria.fr

INRIA Sophia-Antipolis
Oasis team

Spécification des Systèmes Distribués et Embarqués -- UNSA -- Master 2009
Schéma du cours

1. Introduction: concurrence/parallelisme, synchrone/asynchrone, embarqué/distribué  RS
2. MDE: machines d'états, diagrammes d'activité, composants EM
3. Calculs de processus et SOS LH
4. Composants asynchrones et fondements de ProActive LH
5. Sémantique synchrone (Esterel) RS
6. Logique temporelle EM
7. Model Checking RS
8. EXPOSES
Flash back & keywords…

- Formal methods in the design flow of distributed/embedded systems
- Provide mathematical semantics to models so that their relation to implemented product can be asserted and proved:
  - model checking, equivalence checking
  - test generation
- Communication and control (control-flow): interactions, protocols
- Modeling languages:
  - UML and variants (StateCharts, SysML,...)
  - Dedicated IDLs and ADLs for system decomposition (…)

Spécification des Systèmes Distribués et Embarqués -- UNSA -- Master 2009
Systems: structure and behavior

In general, a system is:

- constituted of components, interacting in a collaborative or hierarchical fashion (structure)
- evolving, as a result of the composed functional of its components (behavior)

A system changes state through time; time is counted in number of actions/operations

- In highly dynamic systems the division is blurred, as structure is transformed by behaviors; e.g. in large scale software services (= business grids, SOA, ...)
- rarely the case in embedded systems

See UML and elsewhere, models divided between structural and behavioral ones
Cycle de développement/ design cycle

Requirements capture
Cahier des charges

(Initial) specification

Architectural division

Component design
/ programmation

Component testing

Integration

Global testing

Sign-off / Recette

Implementation

IP component reuse

libraries
Cycle de développement/ design cycle

- Requirements capture
- Cahier des charges
- (Initial) specification
- Architectural division
- Component testing
- Integration
- IP component reuse
- Libraries
- Black box specification
- Synthesis
- Correct-by-Construction
- Implementation
- Test generation
- Component testing
- Global testing
- Sign-off / Recette
- Proof of requirements
- Early specification of Architecture and Interaction
- Correct composition: interface compatibility, deadlock freeness, spec implementation
- Correct-by-Construction

Spécification des Systèmes Distribués et Embarqués -- UNSA -- Master 2009
Agenda

• Graphical Modeling Languages :
  » A zoo of UML diagrams

• Components models :
  » Fractal, GCM

• Tools
  » Build development platforms ?

• Hands-on exercises
UML -- MDE -- Visual models

Single (unified)

Too many different languages, platforms, formalisms….
- Unified visual Language
  - Everybody must speak the same language
- Language for specification / code generation
  - Supposedly precise and non-ambiguous

One single view is not enough:
- Class diagrams
- Sequence diagrams
- Activity diagrams
- State machines
- Composite structure diagrams
- Deployment diagrams
- Marte profile
A single model is not enough!

- Create several independent models but with common points and relations.
Class diagrams
Sequence diagram

1: faire EDT( )
2: get cours offerts( )
3: get cours offerts(pour Le Semestre)
4: get cours offerts( )
5: afficher cours offerts ( )
6: afficher EDT vide( )
Spécification des Systèmes Distribués et Embarqués -- UNSA -- Master 2009

Activity diagram

- Sélectionne cours
  - [ ajout cours ]
  - [ efface cours ]

Guard

- Vérifie EDT
- Vérifie Pré-requis

Choice

- Concurrent executions

Action

- Efface cours

Synchronisation (Fork)

- Synchronisation (Join)

Transition

- Transition

- Efface cours

- OK
- KO

- Affecte Cours
- résout conflits

- metAJour EDT

Spécification des Systèmes Distribués et Embarqués -- UNSA -- Master 2009
State machine diagram
Component and Composite structure diagrams

Provided / required interfaces

Hierarchical components

Ports & Bindings
MARTE: UML Profile for
Modeling and Analysis of Real-Time and Embedded Systems

Slide courtesy of Sebastien Gerard, CEA-LETI

Spécification des Systèmes Distribués et Embarqués -- UNSA -- Master 2009
Components

- Hardware / software

- Synchronous / Asynchronous

- Flat / Hierarchical
Agenda

- Graphical Modeling Languages:
  - A zoo of UML diagrams
- Components models:
  - Fractal, GCM
- Tools
  - Build development platforms?
- Hands-on exercises
The Fractal component model

• Systems and middleware engineering
  • Generic enough to be applied to any other domain
  • Fine grain (wrt EJB or CCM), close to a class model
  • Lightweight (low overhead on top of objects)
  • Independent from programming languages
  • Homogeneous vision of all layers (OS, middleware, services, applications)
Fractal

- Open and adaptable/extensible
- Usable as a component framework to build applications
  - with “standard” Fractal components
- Usable as a component framework framework
  - building different kinds of components
  - with minimum introspection and simple aggregation (à la COM)
  - with binding and lifecycle controllers (à la OSGi)
  - with a two-level hierarchy and bindings (à la SCA)
  - with persistence and transaction controllers (à la EJB)
  - with attribute controllers (à la MBean)
Fractal : controllers

- **Control**
  - Non functional (tech’al) properties
  - Implemented in the membrane
  - Made of a set of controllers
  - E.g. security, transaction, persistence, start/stop, naming
  - Controllers accessible through a control interface
  - Controllers and membranes are open
Fractal tools

• Fraclet
  – programming model based on annotations (within Java programs)
• Fractal ADL
  – XML-based architecture description language (ADL)
• Fractal API
  – set of Java interfaces for
    • introspection
    • reconfiguration
    • dynamic creation/modification
  – of Fractal components and component assemblies
Fractal : development tools

F4E: Eclipse development environment for Fractal applications
Case Study

- Source: France Telecom / Charles Un. Prague
- Specification of an Airport Wifi Network
- Hierarchical, real-size
  - Fractal specification
  - Sofa “behavior protocols”
  - Model-checking
GCM

Grid Component Model

A Fractal Extension

Scopes and Objectives:
- Grid Codes that Compose and Deploy
- No programming, No Scripting, …

Innovations:
- Abstract Deployment
- Multicast and GatherCast
- Controller (NF) Components

Standardization
- By the ETSI TC-GRID
GCM: N×M communication

- 1 to N = multicast / broadcast / scatter
- N to 1 bindings = gathercast
- Attach a behaviour (policy) to these interfaces
GCM: components for controllers

“Componentize” the membrane:

• Build controllers in a structured way
• Reuse of controller components
• Applications: control components for self-optimization, self-healing, self-configuring, interceptors for encryption, authentication, …
GCM architecture specifications: VCE tool
Agenda

• Graphical Modeling Languages :
  » A zoo of UML diagrams

• Components models :
  » Fractal, GCM

• Tools
  » Build development platforms ?

• Hands-on exercises
VCE
VerCors Component Editor

A “Domain Specific Language” for Fractal/GCM
– Component architecture diagrams
– Behaviour diagrams
– Model generation for verification tools
– Code generation

Agenda:
– Tool architecture
– Validation rules
– “hands-on” exercices
VCE
Architecture
VCE Architecture
(middle term)

Vercors

- Graphical Editor (Eclipse Plugin)
- JDC Specification
- JDC Formula

Code Generator

- ADL/IDL (final)
- Java Skeletons
- Business code

Model Generator

- Finite model

Formula Compiler

- pNets/Fiacre

Prover

- GCM/ProAct

Runtime

Spécification des Systèmes Distribués et Embarqués -- UNSA -- Master 2009
VCE
Eclipse and MDE Tools

Eclipse Modelling Tools:

- EMF (Eclipse Modeling Framework): XMI model definition and Java code generation
- GEF (Graphical Editing Framework)
- GMF (Graphical Modeling Framework) for developing graphical editors
- Model Development Tools
- Atlas Transformation Language (ATL)
- ....
A binding should not cross the Components Bound. (CrossingComponentBc Example-multicast.components:vcetest)
Several notions of correctness in the diagram editors:

- Structural correctness, by construction: the graphical tools maintain a number of constraints, like bindings attached to interfaces, interfaces on the box borders, etc.
- But some rules are related to the model structure, not to the graphical objects. E.g. bindings should not cross component levels, or sibling objects should have distinct names…
- There is a “Validation” function (and button), that must be checked only on “finished” diagrams, before model/code generation. It is defined using OCL rules.
VCE : Validation, OCL

OCL example:

- In Content, Bindings must go from Client to Server.

\[ B = \langle It_{f1}, It_{f2} \rangle \in CtC \Rightarrow It_{f1}, \rho = \text{client} \land It_{f2}, \rho = \text{server} \]

context Binding inv FromClientToServer_InContent_ROLES:
  ( Content.allInstances() ->exists(c : Content | c.bindings->includes(self))
    and
    Content.allInstances() ->any(bindings->includes(self)).subcomponents
    ->exists(sc : Component | sc.oclAsType(ComponentDefinition).externalInterfaces
    ->includes(self.sourceInterface))
    and
    Content.allInstances() ->any(bindings->includes(self)).subcomponents
    ->exists(sc : Component | sc.oclAsType(ComponentDefinition).externalInterfaces
    ->includes(self.targetInterface))
  )
  implies self.sourceInterface.role = InterfaceRole::client
  and self.targetInterface.role = InterfaceRole::server
VCE

Examples for the SSDE course

1. Component: external view
2. Component: internal architecture
3. Multicast: example, workflow style
4. Multicast: build a matrix application
5. Master/slave, RPC style
6. Matrix: parameterized style
1. External view
2. Internal architecture

Build a composite component, with:

- **Outside:**
  - 1 serveur interface SI
  - 2 client interface CI1, CI2
  - A number of control (NF) interfaces

- **Inside:**
  - 2 subcomponents
  - One connected to SI
  - Each connected to one client interface
  - One binding between them

Check its validity and produce the ADL
3. Multicast and gathercast, workflow style
4. Composite, multicast, matrix

Build a composite component, with:

• One server interface, with an internal multicast interface

• 2 x 3 subcomponents representing matrix blocks, each linked to its left neighbour
5. Master/slave, RPC style
6. Matrix, parameterized style
7. Exercice

• Analyze this diagram (semantics, errors, …)
Exercice 2

Exercice 4

Exercice 7:
- 1 true error: Bindings crossing component bounds
- 1 false error (bug in a validation rule): more than one component in membrane

Interesting features:
- 1 provided service is not connected (thus not implemented…); is this a problem?
- 2 client interfaces are not used; is this a problem?
- The logger component has no visible interface; is this a problem?
- The life-cycle controller does not control anything; this may be a problem…