



Model- Based Software Development for Safety- Critical Systems Methodology and Application

STRESS 2006
 Summer School on Tool- Based Rigorous Software Engineering

Dortmund, May 19, 2006

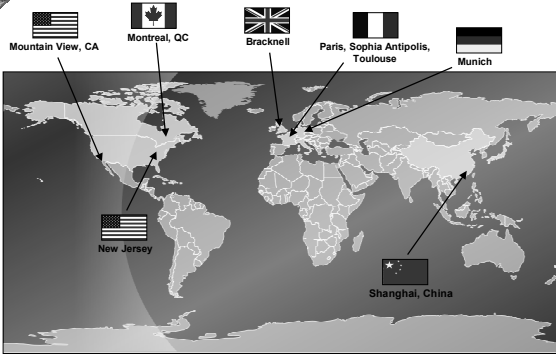
Jakob Gärtner - Esterel Technologies - IEE Safety Critical applications from Aerospace to Automotive 2006

Introducing Esterel Technologies

- ▶ Esterel Technologies is a Software Editor
- ▶ We provide development tools and expertise services
 - ◆ Safety-Critical embedded software (SCADE Suite™, SCADE Drive™)
 - ◆ Critical Electronic Components (Esterel Studio™)
- ▶ Our Customers are OEMs, Tier1s and Tier2s suppliers
 - ◆ Aerospace & Defense
 - ◆ Automotive
 - ◆ Transportation
 - ◆ Semiconductors & Electronics
- ▶ Our Uniqueness
 - ◆ Software design tools and expertise services covering our customers' *design processes* from specification to implementation
 - ◆ Unique formal methods and technologies enabling *automated & certified* implementation while meeting stringent safety requirements

Jakob Gärtner - Esterel Technologies - IEE Safety Critical applications from Aerospace to Automotive 2006

World-Wide Direct Presence



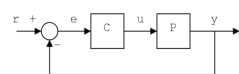
Jakob Gärtner - Esterel Technologies - IEE Safety Critical applications from Aerospace to Automotive 2006

Development of safety critical software in aerospace

- ▶ Process driven
 - ▶ DO-178B is mandatory
- ▶ DO-178B defines the objectives of the process, not its means
- ▶ Process is strictly requirements driven
- ▶ Each project is to be certified by public authorities (EASA, FAA...)

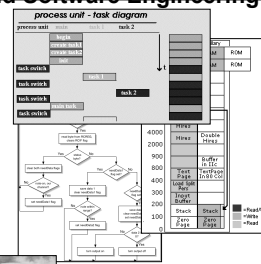
Jakob Gärtner - Esterel Technologies - IEE Safety Critical applications from Aerospace to Automotive 2006

The Chasm between Control and Software Engineering



$$X(z) \triangleq \sum_{n=-\infty}^{\infty} x(n)z^{-n} \quad (\text{bilateral } z \text{ transform})$$

Control engineers describe and analyse systems in terms of block diagrams and z-transfer functions

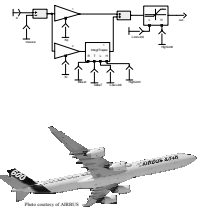


Software engineers describe software in terms of tasks, flowcharts and memory

Jakob Gärtner - Esterel Technologies - IEE Safety Critical applications from Aerospace to Automotive 2006

Convergence of Initiatives

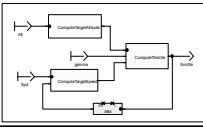
- ▶ Creation of formal synchronous data flow language:
 - ▶ Data flow language 'Lustre', designed at IMAG (Grenoble, 1985)
 - ▶ Control flow language 'Esterel' at École des Mines, Sophia Antipolis
- ▶ Industries developing safety critical software
 - ▶ Aerospatiale created 'SAO' for Airbus
 - ▶ Merlin-Gerin created 'SAGA' for nuclear power plant control
- ▶ SCADE product created in partnership with industry (Aerospatiale, Merlin-Gerin) (1995)



Jakob Gärtner - Esterel Technologies - IEE Safety Critical applications from Aerospace to Automotive 2006

SCADE IS a Bridge Between Control and Software Engineering

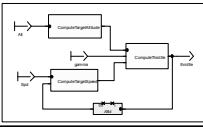
Block diagram



Mathematical Z operator

$$y = az^{-1}x$$


SCADE Formal Data Flow



SCADE pre operator

$$y = a * pre(x)$$

SCADE maps control engineering constructs to rigorous software constructs.



Simple notation familiar to both system and SW teams.

Jakob Gärtner - Esterel Technologies - IEE Safety Critical applications from Aerospace to Automotive 2006

SCADE: Safety-Critical Application Development Environment

- ▶ Unique **model-based** development environment that reconciles **productivity** and **safety**.
- ▶ Familiar graphical notation with block diagrams & state machines, rigorously defined and fully deterministic.
- ▶ Designed **from the beginning** for the development of **DO-178B** projects up to level A. SCADE is also certified to IEC 61508.
- ▶ SCADE Suite Automatic Code Generator is **used in production** for major recent DO-178B certification programs **worldwide**.

The de facto standard for the development of safety-critical embedded software in the Avionics Industry.

Jakob Gärtner - Esterel Technologies - IEE Safety Critical applications from Aerospace to Automotive 2006

Introduction to SCADE semantics

▶ See *technical SCADE introduction*

Jakob Gärtner - Esterel Technologies - IEE Safety Critical applications from Aerospace to Automotive 2006

Development of safety critical software in aerospace

- ▶ Process driven
 - ▶ DO-178B is mandatory
- ▶ DO-178B defines the objectives of the process, not its means
- ▶ Process is strictly requirements driven
- ▶ Each project is to be certified by public authorities (EASA, FAA...)

Jakob Gärtner - Esterel Technologies - IEE Safety Critical applications from Aerospace to Automotive 2006

System & SW Lifecycle

System life-cycle processes (ARP 4754)

System safety assessment process

System requirements allocated to software

Software level(s)

Design constraints

Hardware definition

Fault containment boundaries

Error sources identified/eliminated

Software requirements & architecture

Software life-cycle processes (DO-178B)

part of implementation processes, for ARP 4754

Jakob Gärtner - Esterel Technologies - IEE Safety Critical applications from Aerospace to Automotive 2006

DO-178B life cycle structure

Planning process

Standards environment

Development process

Integral process

Certification Liaison process

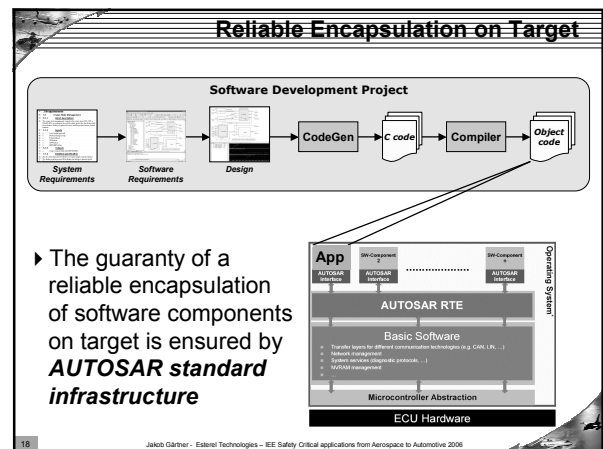
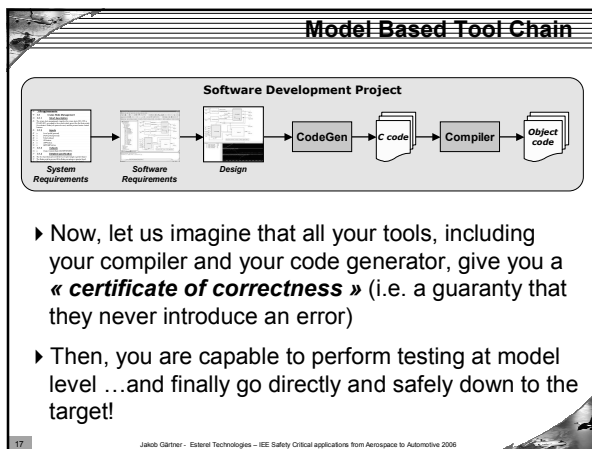
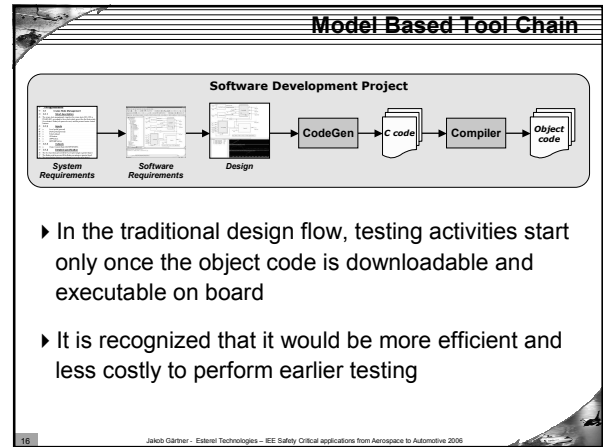
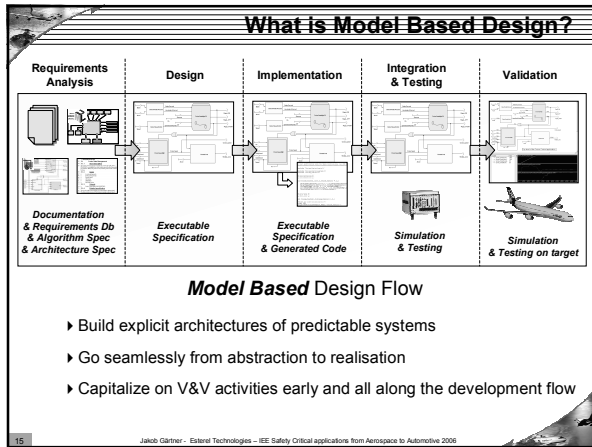
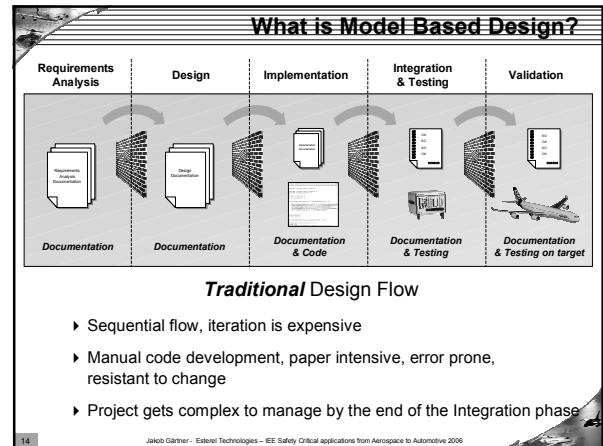
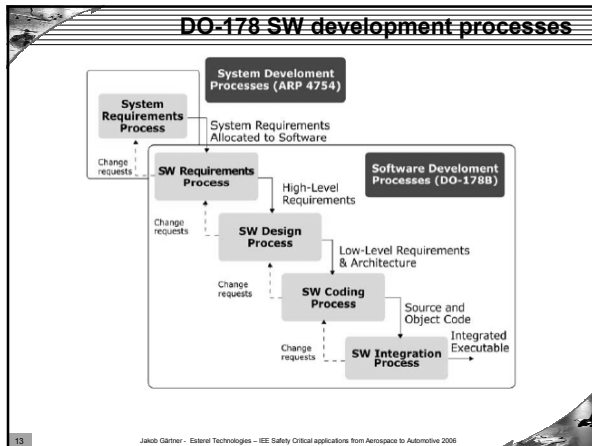
Configuration Management process

Verification process

Testing

Quality Assurance

Jakob Gärtner - Esterel Technologies - IEE Safety Critical applications from Aerospace to Automotive 2006



Verification of Correct Compilation

- ▶ The source code generated by a « **certified** » code generator uses only a **small subset of the C** language, with a low level of complexity
- ▶ A test suite can be built from all the C constructs that can ever be generated from the model, ensuring MC/DC coverage at object code level
- ▶ The subset approach is accepted by safety authorities

19 Jakob Gärtner - Esterel Technologies - EEE Safety Critical applications from Aerospace to Automotive 2006

Verification of Correct Compilation: Process

20 Jakob Gärtner - Esterel Technologies - EEE Safety Critical applications from Aerospace to Automotive 2006

Certified Code Generation

- ▶ A **Certified C Code Generator generates simple C code** that fits the constraints of safety-critical embedded software:
 - ▶ Portable (compiler, target and OS independent)
 - ▶ Structured (by function or by blocks)
 - ▶ Readable, traceable (name/annotation propagation)
 - ▶ Static memory allocation
 - ▶ No pointer arithmetic
 - ▶ No recursion, bounded loops only
 - ▶ Bounded execution time
 - ▶ Size or speed optimisation

21 Jakob Gärtner - Esterel Technologies - EEE Safety Critical applications from Aerospace to Automotive 2006

Certified Code Generation

- ▶ A **Certified C Code Generator** has been designed from the beginning with safety objectives: **DO-178B, IEC 61508**
- ▶ The expected benefits are:
 - ▶ Remove the requirement to perform low level testing
 - ▶ Enable cost effective functional verification at model level
 - ▶ Reduce cycle time for requirements/model changes by 3X.
 - ▶ Reduce time to market by 50%.

22 Jakob Gärtner - Esterel Technologies - EEE Safety Critical applications from Aerospace to Automotive 2006

Requirements-Based Model Verification

- ▶ Functional verification has to demonstrate that the software satisfies its requirements
- ▶ In a model-based approach, **model simulation** allows **requirements-based tests** to be performed at the model level for early detection of specification errors
- ▶ Concurrently, **Model Test Coverage analysis** shall assess how thoroughly a model has been explored by simulation

23 Jakob Gärtner - Esterel Technologies - EEE Safety Critical applications from Aerospace to Automotive 2006

Requirements-Based Model Verification

- ▶ **Model Test Coverage (MTC)** may reveal **unintended** functions, shortcomings in test procedures, and inadequacies in requirements
- ▶ As soon as MTC gives evidence that all elements of the model have been covered with respect to the requirements, the **functional verification activity is done ... down to the target!**

24 Jakob Gärtner - Esterel Technologies - EEE Safety Critical applications from Aerospace to Automotive 2006

Correct-By-Construction Modelling

Software Development Project

System Requirements → Software Requirements → Design → CodeGen → C code → Compiler → Object code

- Correct-by-Construction modelling relies on a graphical language that is very simple and stable, forbidding dangerous constructs (e.g. unbounded loops, wild goto's, dynamic memory allocation,...)
- Interpretation of a model does not depend on the reader or their environment
- Model checking** is continuously allowed to preserve consistence of data types, clocks and sub clocks, data dependencies, cycle detection, etc.

25 Jakob Gärtner - Esterel Technologies - IEE Safety Critical applications from Aerospace to Automotive 2006

In Summary: The SCADE Combined Testing Process

Software Development Project

System Requirements → Software Requirements → Design → CodeGen → C code → Compiler → Object code

- Model Coverage Analysis
- Model Checking
- Certified CodeGen
- Compiler Verification

- ✓ SCADE Editor: Design Consistency Checks
- ✓ SCADE Reqs-Based Model Testing & MTC
- ✓ IEC 61508 Certified SCADE KCG = No Low-Level Verification
- ✓ Compiler Verification Kit (CVK comprehensive test suite)

26 Jakob Gärtner - Esterel Technologies - IEE Safety Critical applications from Aerospace to Automotive 2006

SCADE Addresses the Applicative part

SCADE Application

Most complex and changing software part

Hand Code

I/O and Scheduling

Operating System Drivers

Hardware

27 Jakob Gärtner - Esterel Technologies - IEE Safety Critical applications from Aerospace to Automotive 2006

Certified Software Factory

SCADE

Model Coverage Editor

KCG

Automatic Code Generation

DO-178B Qualified up to Level A

IEC 61508 Certified for up to SIL 4

.C

RTOS Wrapper Generator

Integration on Target

Object Code Verification

Automatic Documentation Generation

Debugging & Verification

Configuration Management

Requirements Management

Formal Verification

Model Coverage Analysis

SCADE™ High-Level Requirements

UML™ Hardware Design

Simulink® Hardware Design

28

Standardization of Platforms: ARINC 653

Secure Partition 1: Ada Program Safety Level: A (High), GMART Ada run-time

Secure Partition 2: C Program, SCADE application Safety Level: A (High)

Secure Partition 3: EC++ Program Safety Level: B (Medium)

Secure Partition 4: C Program Safety Level: D (Low), FAILURE !

INTEGRITY-178B API/ARINC 653 APEX API

NO EFFECT !

INTEGRITY-178B Kernel

Embedded Processor

29 Jakob Gärtner - Esterel Technologies - IEE Safety Critical applications from Aerospace to Automotive 2006

SCADE Solution

The Market Leader in Aerospace & Defense for the Development of Safety- and Mission-Critical Software

30 Jakob Gärtner - Esterel Technologies - IEE Safety Critical applications from Aerospace to Automotive 2006

Typical SCADE Aerospace & Defense Applications

- ▶ Flight control systems
- ▶ Power management
- ▶ Reconfiguration management
- ▶ Autopilots
- ▶ Engine control systems
- ▶ Braking systems
- ▶ Cockpit display and alarm management
- ▶ Fuel management

31 Jakob Gärtner - Esterel Technologies - IEEE Safety Critical applications from Aerospace to Automotive 2006

Typical SCADE Automotive Applications

- ▶ Airbags
- ▶ Braking Systems, ABS & ESP
- ▶ Steering
- ▶ Chassis & Suspension Systems
- ▶ Driver Assistance Systems
- ▶ Restraining systems
- ▶ Engine regulation
- ▶ X-By-Wire applications

32 Jakob Gärtner - Esterel Technologies - IEEE Safety Critical applications from Aerospace to Automotive 2006

Typical SCADE Rail & Heavy Duty Applications

- ▶ Interlocking systems control
- ▶ Signaling
- ▶ Ground stations
- ▶ Automatic Train Operations
- ▶ Train Control Systems
- ▶ Heavy Duty Land systems (tanks, tractors...)

33 Jakob Gärtner - Esterel Technologies - IEEE Safety Critical applications from Aerospace to Automotive 2006

Typical SCADE Nuclear I&C Applications

- ▶ Reactor Protection Systems:
 - ▶ Reactor limitation system
 - ▶ Trip processing
 - ▶ Emergency shutdown
 - ▶ Reactor trip breakers
- ▶ Nuclear Instrumentation Systems:
 - ▶ Power measurement system
 - ▶ Sensor controllers
 - ▶ Pressurizer heating controllers
 - ▶ Rod position instrumentation systems
- ▶ Other Safety Systems
 - ▶ Safety valve control system
 - ▶ Control rod control systems

34 Jakob Gärtner - Esterel Technologies - IEEE Safety Critical applications from Aerospace to Automotive 2006


SCADE Successes: a few Facts & Figures

Company	Product	Specified & Auto coded	Benefits Claimed
AIRBUS	A340/500-600	<ul style="list-style-type: none"> ▶ 70% Fly-by-wire Controls ▶ 70% Automatic Flight Controls ▶ 50% Display Computer ▶ 40% Warning & Maintenance Computer 	<ul style="list-style-type: none"> ▶ 20X Reduction in Errors ▶ Reduced Time to Market
EUROCOPTER	EC 155/135	<ul style="list-style-type: none"> ▶ 90% Automatic Pilot 	<ul style="list-style-type: none"> ▶ 50% Reduction in Development Cycle Time
SCHNEIDER ELECTRIC	Nuclear Power Plant Safety Control	<ul style="list-style-type: none"> ▶ 200,000 SLOC Auto Generated from 1,200 Design Views 	<ul style="list-style-type: none"> ▶ 8X Reduction in Errors while Complexity Increased 4X
PSA	Electrical Management System	<ul style="list-style-type: none"> ▶ 50% SLOC Auto Generated 	<ul style="list-style-type: none"> ▶ 60% Reduction in Development Cycle Time ▶ 5X Reduction in Errors
ANSALDO	Subway Signaling System	<ul style="list-style-type: none"> ▶ 80,000 SLOC Auto Generated 	<ul style="list-style-type: none"> ▶ Improved Productivity from 20 to 300 SLOC/day

35 Jakob Gärtner - Esterel Technologies - IEEE Safety Critical applications from Aerospace to Automotive 2006


A Few of the Current Projects Embedding SCADE

36 Jakob Gärtner - Esterel Technologies - IEEE Safety Critical applications from Aerospace to Automotive 2006




 Specification to safe embedded code
 trained by certification

Document Repository	Knowledge Management
Document Title	SCADE Course
Document Reference	
Document Version	
Document Date	Jan. 2005
Last Modification Date	
Authors	Sabine Sabathier, Bernard Dion, Amar Bouali
Summary	SCADE Course 5.0 Module M1 Course 1
File name	SCADE 5.0 Course M1_C1.ppt
Approbation List	

© 2005 Estarel Technologies 


SCADE Language Properties

- Declarative language
 - Typed & Structured
 - Hierarchical
 - Deterministic semantics
- Provides safe execution
 - No inner loop, no dynamic allocation
 - Maximum time of computation can be calculated
- Based on LUSTRE (textual synchronous data-flow language)
 - Developed at Verimag: www-verimag.imag.fr

© 2005 Estarel Technologies 

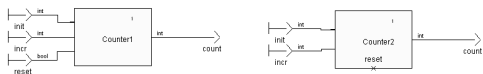
Nodes

- Block diagram nodes
 - Boxes with an I/O interface defining new operators for reuse and hierarchy
 - Inner body can be either graphical or textual
- State machine nodes
 - Flat State Machines
 - Safe State Machines (SSM)
- Imported nodes (C or Ada functions)
 - Nodes defined in C or Ada code extending the language expressivity

© 2005 Estarel Technologies 

Block Diagrams (1/2)

- Graphical SCADE representation
 - A node is a block diagram




The node is seen as a black box and is connected through its formal I/O interface.

- Textual SCADE representation

```

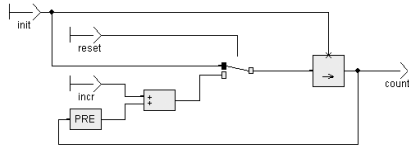
node Counter1(init,incr: int; reset: bool)
returns (count: int) ;

node Counter2(init,incr: int; hidden reset: bool)
returns (count: int) ;
  
```

© 2005 Estarel Technologies 

Block Diagrams (2/2)


- A block diagram example



- Textual SCADE representation

```

count = init -> if reset then init
                else pre (count) + incr;
  
```

© 2005 Estarel Technologies 

Textual Nodes


```

node GetMiddle(
Pt_B : point ;
Pt_A : point)
returns (
middle : point) ;

let equ eq_GetMiddle[ . ]
middle = Vec3Mult(Vec3Sum(Pt_B , Pt_A) , 0.5) ;
tel ;

/* End of blocks of node : GetMiddle */
  
```

- Nodes graphical definitions are automatically translated into textual SCADE
- Also possible to define nodes textually
- Both graphical and textual SCADE are case sensitive

© 2005 Estarel Technologies 

Flat State Machines

- A single initial state
- Transition labels = Textual SCADE Boolean predicates
- Transitions have priorities to ensure determinism
- For each state, an output with same name is created

© 2005 Estrel Technologies

SSM Overview

- Graphical Elements:
 - Simple states as ovals or rectangles, transitions as arrows, macro-states as rectangle boxes
- Textual Elements:
 - Input, outputs, and state names
 - Trigger expressions for transition firing condition
 - Actions expressions related to transitions firing and state activities

© 2005 Estrel Technologies

SSM Features

Set output O as soon as input A and input B have been received. Reset the whole behavior when input R is received.

- Hierarchy
- Macro-states
- Concurrency
 - A and B are independent
- Sequencing
 - O follows AB
- Preemption
 - R resets the entire behavior

© 2005 Estrel Technologies

Imported Nodes

- The aim of an imported node is to describe processing for which SCADE language is not suitable
 - The node name and the interface is defined in SCADE, but not the body
 - The implementation will be provided into the target language

```

/*****
** My ADD function
**
*****/
#include "scade_types.h"
void ImpADD (int A, int B, int *Sum)
{
  *Sum = A + B;
}
  
```

© 2005 Estrel Technologies

Some Choice Operators (1/2)

- The **if-then-else** operator
 - Expresses a decision
 - Because of the data-flow semantics, both "then" and "else" expressions are always evaluated independently of the condition value.
 - Example
 - Count1 and Count2 are 2 counters
 - `S = if c then count1() else count2();`
 - Each counter will be incremented at each execution cycle regardless of the Boolean flow c
 - Note that this semantic suits well with the usual intuition when reading the equivalent graphical flow

© 2005 Estrel Technologies

Some Choice Operators (2/2)

- The **case** operator
 - It's a switch
 - 2N+1 inputs and 1 output
 - N inputs e_i , each corresponding to a possible value for the output
 - N inputs, each associated with a possible effective input e_i and a "default" value
 - 1 input: a switch whose value is compared with each label
 - All the inputs are computed before the choice
 - Example


```

S = case es of
  3 : (e1)
  -2 : (e2)
  C2 : (e2)
  default : (e3);
          
```

 - `if es=3 then S=e1 else if es=-2 or C2 then S=e2 else for all others S=e3`

© 2005 Estrel Technologies

Some Temporal Operators (1/3)

- The \rightarrow initialization operator
 - Allows expressions to be initialised
 - During the first cycle, the previous value is indefinite
 - Example:

E	e1	e2	e3	e4
F	f1	f2	f3	f4
E \rightarrow F	e1	f2	f3	f4

© 2005 Estrel Technologies

Some Temporal Operators (2/3)

- The **pre** delay operator
 - Allows a trace of the value of an expression to be kept from one cycle to another
 - During the first cycle, the previous value is indefinite
 - Example:

E	e1	e2	e3	e4
pre (E)	nil	e1	e2	e3
1 \rightarrow pre (E)	1	e1	e2	e3
10 \rightarrow pre (a+b)	10	a1+b1	a2+b2	a3+b3
Y1 \rightarrow pre (pre (E))	Y1	nil	e1	e2

© 2005 Estrel Technologies

Some Temporal Operators (3/3)

- The **fbby** delay operator
 - Allows a trace of the value of an expression to be kept over several cycles
 - Introduces a delay
 - The number of cycles must be a strictly positive integer value
 - Example

E	e1	e2	e3	e4	e5
fbby (E, 2, Init)	Init	Init	e1	e2	e3

– **fbby (E, n, Init)** is equivalent to:
Init \rightarrow pre (Init \rightarrow pre (... \rightarrow pre (E)))

© 2005 Estrel Technologies

Clock Principles

- Clocks allow sub-systems to run at different rates
- The availability of a flow is defined by its clock. It is the rate at which it is sampled
- An operator is executed when all its inputs are available. This defines the clock of an operator
- Clocks are supported by the **WHEN**, **CURRENT** and **CONTACT**

© 2005 Estrel Technologies

WHEN & CURRENT (1/4)

- WHEN** samples variables on slower rate.
- CURRENT** projects variables on faster rate.
- Let 's assume that :
 - X = (x1, x2, x3, x4, x5, x6, x7...)**
The x flow is defined for each cycle. It is on the basic clock of the operator in which it is used.
 - C = (T, F, T, T, F, F, T, ...)**

© 2005 Estrel Technologies

WHEN & CURRENT (2/4)

Z=current(Y)		x1	x1	x3	x4	x4	x4	x7
Y=X when C		x1		x3	x4			x7
Clock C		T	F	T	T	F	F	T
X		x1	x2	x3	x4	x5	x6	x7
		t1	t2	t3	t4	t5	t6	t7

- The Y flow is defined only when C is true
- The Z is defined on the basic clock of the operator in which it is used

© 2005 Estrel Technologies

WHEN & CURRENT (3/4)

What would happen if C was false at the first tick ?

Z=current(Y)	nil	nil	x3	x4	x4	x4	x7
Y=X when C			x3	x4			x7
Clock C	F	F	T	T	F	F	T
X	x1	x2	x3	x4	x5	x6	x7

t1 t2 t3 t4 t5 t6 t7 t

- Nil means that the value is indefinite at this cycle

WHEN & CURRENT (4/4)

- Filtering the inputs

- Filtering the outputs

Using CONDUCT (1/4)

A "CONDUCT" takes place for both **WHEN** and **CURRENT**

It issues the output initialisation value used at the first clock cycle

Corresponding implementation using WHEN and CURRENT :

Using CONDUCT (2/4)

Y=conduct (Op(X),C, Y1init)	Y1= Op(x1)	Y3= Op(x3)	Y4= Op(x4)	Y4	Y4	Y7= Op(x7)	
Clock C	T	F	T	T	F	F	
X	x1	x2	x3	x4	x5	x6	x7

t1 t2 t3 t4 t5 t6 t7 t

Using CONDUCT (3/4)

What would happen if C was false at the first tick ?

Y=conduct (Op(X),C, Y1init)	Y1init	Y1init	Y3= Op(x3)	Y4= Op(x4)	Y4	Y4	Y7= Op(x7)
Clock C	F	F	T	T	F	F	T
X	x1	x2	x3	x4	x5	x6	x7

t1 t2 t3 t4 t5 t6 t7 t

Using CONDUCT (4/4)

Right-click on the node and select *Operator to Conduct*

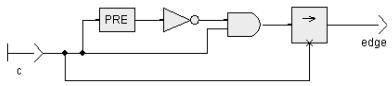

Exercise





© 2005 Esterel Technologies 


M1-C1 Exercise 1

- Textual representation



```
node RisingEdge (c: bool) returns (edge: bool);
let
  edge = c → c and not pre(c);
tel;
```
- Graphical representation
 


 Define the textual and the graphical representations of the **FallingEdge** node which detects a true-false sequence.

© 2005 Esterel Technologies 

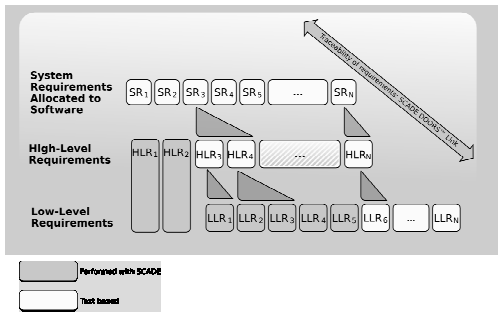

M1-C1: SCADE Designing


- SCADE Requirements and Design Approach
 - High-Level Requirements
 - Preliminary Design
 - Design



© 2005 Esterel Technologies 


High and Low-Level Requirements



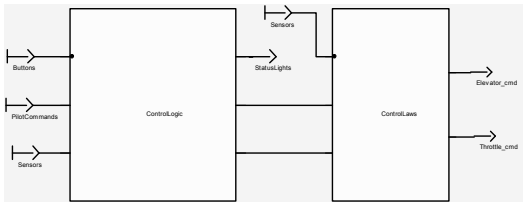
© 2005 Esterel Technologies 

High-Level Requirements


- SCADE can be used to describe part of the High Level Requirements
- The SCADE model is part of the software requirements document, which contains also text
- At this stage, the SCADE model is incomplete and serves to identify:
 - High level functions and data flows
 - Root subsystem interface
 - Main states

© 2005 Esterel Technologies 

High-Level Requirements



Example top-level functions and interfaces

© 2005 Esterel Technologies 

High-Level Requirements

Do a functional decomposition of the application

1. Identify the inputs/outputs of the system
2. Identify main functions and states
3. Describe the relations between the functions
 - Decomposition in sub-systems
 - Decomposition of data
 - Definition of the network view
 - Distribute sub-systems to team members for large projects

31

© 2005 Estarel Technologies ESTAREL

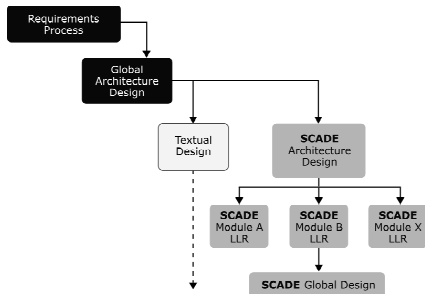
High-Level Requirements

- Using blocks diagrams and state transitions diagram in SCADE:
 - Only hierarchy and data-flows are specified.
 - The low-level components remain empty or are described in natural language (annotations). Example: a low-pass filter is not described at this level.
 - The input/output functions are not described.
 - The types of the data are not completely defined.

32

© 2005 Estarel Technologies ESTAREL

Preliminary Design



Identify what will be developed in SCADE and what will be developed manually

33

© 2005 Estarel Technologies ESTAREL

Software Design

- Refinement process: divide and conquer
 - Decompose according to
 - identified functionalities
 - data
 - temporal aspects
 - And also keep in mind development quality criteria
 - reusability (targeted on the library operators)
 - readability of the description (documentation)
 - testability
 - parallel development
 - etc.

34

© 2005 Estarel Technologies ESTAREL

Divide and Conquer

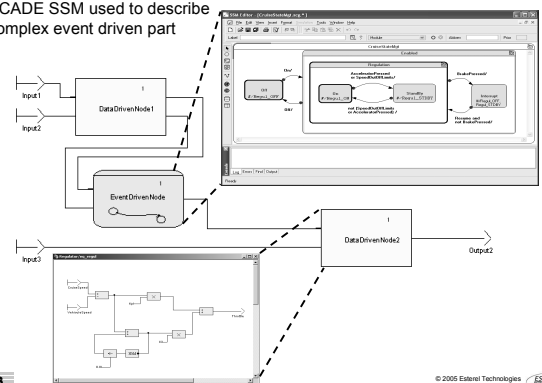
- Applications usually contain
 - an event-driven part
 - a data-driven part
- Usually the event driven part pilots the data driven one. Some functions of the event-driven part are data-driven sub-systems
- Inside SCADE, Flat State Machines or SSMs can be used to describe the event-driven parts

35

© 2005 Estarel Technologies ESTAREL

Divide and Conquer

SCADE SSM used to describe complex event driven part



36

© 2005 Estarel Technologies ESTAREL

Complete the Design

- Develop the algorithms of the basic components:
 - with data-flow description (using SCADE)
 - in a sequential way (using imported operators)
- Select and use libraries
 - Improve consistency and quality
- Reiterate at sub-systems level
 - Stop when reaching basic operators, library nodes or imported nodes.

37

© 2005 Esterel Technologies

Software Coding

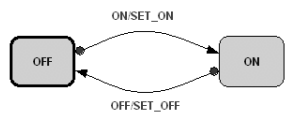
- Develop the code of the input/output functions
- Develop the code of the components which are not described in SCADE
- Generate the code of the whole SCADE model according to the compiling mode chosen during the software design

38

© 2005 Esterel Technologies

State Machines for State-based Logic

- Boxes are states
- Arcs are transitions



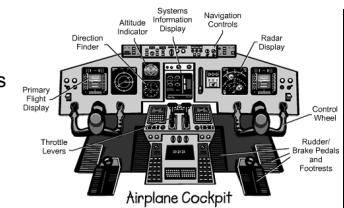
- States and transition have labels to attach names and behavior
- Commonly used for modeling control flow and decision logic with intuitive meaning

39

© 2005 Esterel Technologies

Domains of Application Airplane Cockpit Display management

- States machines for managing the configurations and the interactions with pilots
- Several displays
 - Parallelism for the concurrent behaviors
- Groups of displays
 - Hierarchy for the organization
- There are alarms
 - Preemption to display them in priority



40

© 2005 Esterel Technologies

Domains of Application Car Control and Comfort

- Seat Memory
 - State machines to memorize the seat position preferences and their settings
 - Hierarchy for the preferences
- Cruise Control
 - State machines to decide the functioning modes according to the speed, user requests, events, etc.
 - Preemption for exceptional cases
- Multimedia *Infotainment* Systems
 - States machines to handle the modes, displays, user interface
 - Hierarchy, parallelism, preemption

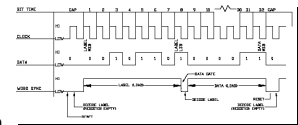


41

© 2005 Esterel Technologies

Domains of Application Communication Protocols

- State machines for controlling the communication aspects of a protocol
 - Data read and data write with buses
 - Encoding / Decoding logic
 - Bus access arbitration
 - Examples: ARINC, CAN, TTP.
 - Parallelism, hierarchy, preemption (reset)



42

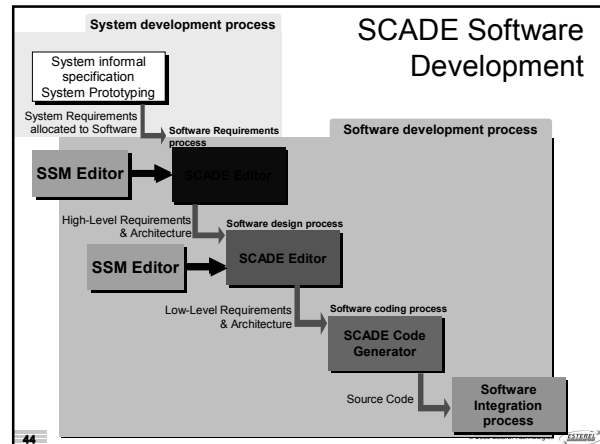
© 2005 Esterel Technologies

Summary of the Need

- State Machines for modeling and designing the control logic aspects of applications with the following fundamental modeling means:
 - Parallelism modeling for capturing the concurrent behaviors
 - Hierarchy for design architecture
 - Preemption for handling exceptions
 - Smooth and consistent mix with data-flow design
- ➔ SCADE Safe State Machine is the solution!

43

© 2005 Estimote Technologies ESTIMOTE



44

ESTIMOTE

SSM in SCADA

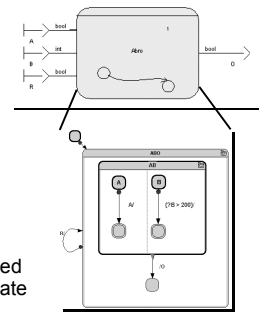
- Capture the state-based logic of an application
- State machine based graphical formalism with parallelism and hierarchy
- Cycle-based computing model
- Tightly coupled with the SCADA data-flow language
- Formal semantics with deterministic computation model

45

© 2005 Estimote Technologies ESTIMOTE

SSM Overview

- Graphical Elements:
 - Simple states as ovals or rectangles, transitions as arrows, macro-states as rectangle boxes
- Textual Elements:
 - Input, outputs, and state names
 - Trigger expressions for transition firing condition
 - Actions expressions related to transitions firing and state activities

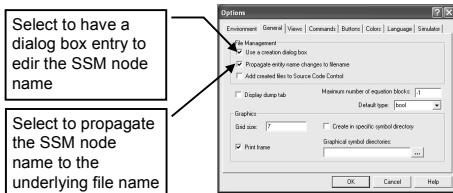


46

© 2005 Estimote Technologies ESTIMOTE

SSM Node Naming

- The node name can be propagated on the file name on demand
- Open *Tools* → *Options* → *General tab*



47

© 2005 Estimote Technologies ESTIMOTE

SSM Node Characteristics

- Share the SCADA predefined types

SCADA	SSM
bool	boolean
int	integer
real	float

- Share all the user constants
- Share the imported type declarations
- Share all the functions related to the imported nodes C functions with one output

48

© 2005 Estimote Technologies ESTIMOTE

SSM Editor Overview

Toolbars and menus

SSM workspace objects

SSM edition area

SSM editor output

49 © 2005 Esterel Technologies

The SSM Inputs and Outputs

50 © 2005 Esterel Technologies

SSM Inputs and Outputs

- SSM I/O signals are declared at the SCADE SSM node level
- 3 kinds of signals:
 - Pure (**bool**)
 - Valued (tuple [**bool**, **type**])
 - Value-only (**type**)

51 © 2005 Esterel Technologies

SSM Inputs and Outputs

Pure Signals

- Carry a Boolean presence status: true when *present* false when *absent*
- At each cycle, absent by default and present if there is some emitter in the cycle
- Predefined signal **tick**, as the SSM global clock, always present
- Inputs are set by the environment

52 © 2005 Esterel Technologies

SSM Inputs and Outputs

Valued Signals

- Equivalent to SCADE flows of type [**bool**, **type**] where **bool** is the presence status; **type** can be: **bool**, **int**, **real**, or any SCADE type
- Library **libssm** provides shortcuts **type_signal** where **type** can be **bool**, **int**, or **real**
- Outputs must be provided with an initial value

53 © 2005 Esterel Technologies

SSM Inputs and Outputs

Value-only Signals

- Equivalent to SCADE flows of same type
- Possible types: **bool**, **int**, **real**, and all user-defined SCADE types
- Assigned a new value from a single emission in a cycle otherwise, keeps the value of previous cycle
- Outputs must be provided with an initial value

54 © 2005 Esterel Technologies

Which Signal Kind?

Pure	To model a logical event whose lifetime and consideration is only meaningful in a cycle of execution: alarm, threshold detection
Value-only	To model a data-value whose value is likely to be coming from the environment or from a continuous internal computation
Valued	Same as value-only and having additionally a status to characterize the cycles where it makes sense to analyze the value

55 © 2005 Estrel Technologies ESTREL



SSM States (1/3)

- Simple State
 - Graphical oval or rectangle that can be named
 - Basic memory element of a SSM
 - At each cycle, is either active or not active

SimpleState

RectangleSimpleState

To draw the simple states as rectangles, edit the text file `estudio_ssm.cfg` in your application data settings and set `RECT_SIMPLE_STATES = 1`.

57 © 2005 Estrel Technologies ESTREL

SSM States (2/3)

- Macro-state
 - Graphical box that can be named
 - SSM container
 - At each cycle, it is either active or not active
 - When active, at least one state of the sub-SSM it contains is active

58 © 2005 Estrel Technologies ESTREL

Creating a State

1. Select the state or macro-state creation button

2. Drag and drop the selection in the edition area

59 © 2005 Estrel Technologies ESTREL

State Basic Attributes

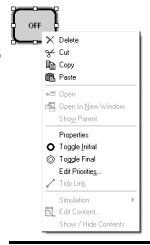
Name: a SCADE identifier, e.g. **AlarmON**, **FlightMode**, **Temperature**

Mark as initial or final

60 © 2005 Estrel Technologies ESTREL

Initial State

- Graphically depicted as a bold circle for simple states and a bold box for macro-states
- The initial states is active whenever the SSM is started or the macro-state it is in is activated
- Right-click on the state and select *Toggle initial* to set it as initial

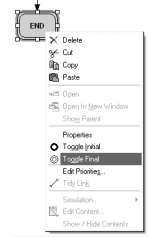


61

© 2005 Esterel Technologies

Final States

- Also called terminal, graphically denoted as a doubled circle
- Only simple states can be final
- When reached, means that the SSM activity terminates (treated later)
- Right-click on the state and select *Toggle final* to set it as final



62

© 2005 Esterel Technologies

Other State Properties

- State look-and-feel is customizable
 - Font of state label
 - Line style
 - Colors
 - Comments

Comment: a state with customized look-and-feel



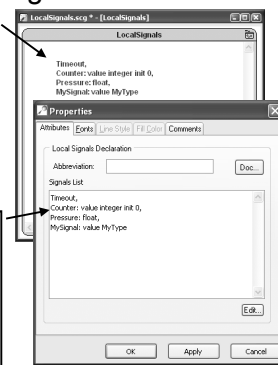
63

© 2005 Esterel Technologies

Local Signals

Defined in the scope of a macro-states as a comma separated list of signal declarations. Can be read or produced at any part of the contents of the macro-state

Right-click in the macro-state area and select Properties. Local signals can be pure, valued, or value-only, possibly initialized



64

© 2005 Esterel Technologies

The SSM Transitions



65

© 2005 Esterel Technologies

SSM Transitions

- Have one source state, and one target state
- If source state is active, *preempts* the activity of source state starting the activity of target state
- A transition can be fired the next cycle after the cycle their source has started activity.



66

© 2005 Esterel Technologies

Creating a Transition

1. Select the transition creation button.

2. Click in the source state and drag to the target state. Release button when highlighting the target state.

© 2005 Estimote Technologies ESTIMOTE

Other Transition Properties

- Transition look-and-feel can be customized:
 - Font for labels
 - Line style
 - Comments

Comment: a customized transition

© 2005 Estimote Technologies ESTIMOTE

Transition Trigger

Fire transition upon the presence of a signal.

- If Signal ON is present (true), and if State OFF is active, then preempts State OFF and activates State ON.
- Signal ON can be an input from the SSM context, an output produced by the SSM, or an emitted local signal.

© 2005 Estimote Technologies ESTIMOTE

Transition Trigger

Right-click on transition and open Properties.

Transition priority.

Enter transition trigger as a Boolean expression of signal presence statuses: (A and not(B)) (C or D).

Click on Apply or OK to validate.

© 2005 Estimote Technologies ESTIMOTE

Transition Priorities

- Priorities are automatically set by the Editor as soon as a state has two or more outgoing transitions
- To ensure determinism, when two transitions have a true triggering condition, only the one with highest priority is fired.
- Implicit negation of all triggers from highest transition

Q: What if transition of priority <2> is fired?

© 2005 Estimote Technologies ESTIMOTE

Transition Effect

Perform an action while firing a transition

- If Signal ON is present (true), and if State OFF is active, then preempts State OFF and activates State ON.
- Perform an action: emits the signal SET_ON.
- Signal SET_ON is an output produced by the SSM.

© 2005 Estimote Technologies ESTIMOTE

Transition Effect

Trigger: has to be true to fire the transition.

Effect: list of signal to emit while firing the transition.

73

© 2005 Estere Technologies ESTEREL

Signal Emission

- Emitted signals are *broadcast* to the whole SSM the signal is defined in.
- In a given cycle, if a signal is emitted then:
 - It is present in that cycle (pure and valued signals).
 - For value-only and valued, the new value is accessible for any reader in that cycle.
 - It is guaranteed that any reader is always scheduled after any updated.

74

© 2005 Estere Technologies ESTEREL

Valued Signals Syntax

- To read the value of a valued or value only signal *s*, write *?S*

```
{ ?Pressure > 10.0f }
```
- To emit a valued signal *s* with value *val*, write *S(val)*, where can be:
 - A constant value: `Alarm(false)`, `Level(LOW)`, `Speed(90)`, `Angle(0.5f)`
 - The value of another signal: `MaxSpeed(?Speed)`
 - The return value of a function call: `Speed(Max(?X, ?Y))`, `Alarm(ValveOpen())`
 - The result of an arithmetic expression over values: `MaxSpeed(?Speed + Max(?X, ?Y) + 1)`

75

© 2005 Estere Technologies ESTEREL

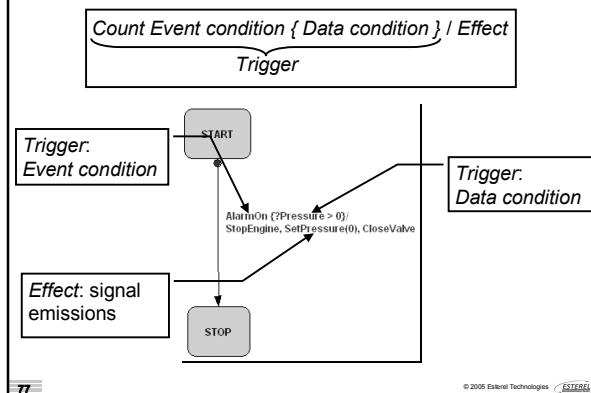
The pre Operator

- Access the previous status or value of a signal
- For any signal *S*, `pre(S)` is the value of the status at the previous cycle
- For a valued or value-only signal *v*, `pre(?v)` is the value of the data carried by *v* at the previous cycle
- The `pre` operator cannot be applied on itself

76

© 2005 Estere Technologies ESTEREL

Transition Labeling (1/3)



77

© 2005 Estere Technologies ESTEREL

Transition Labeling (2/3)

Count Event condition { Data condition } / Effect

Trigger

- Trigger:** Conjunction (AND) of two conditions
 - Event condition:** A formula of signal statuses, e.g.: `A and not(B)`
 - Data condition:** A formula of data conditions surrounded by "{ }", e.g. `{ ?Level >= 10 and (Max(?A, ?B) < ?C) }`
- An optional counter *Count*: number of times the entire trigger has to be true
 - `5 times tick`
 - `3 times CLICK {?KEY = SHIFT}`

78

© 2005 Estere Technologies ESTEREL

Transition Labeling (3/3)

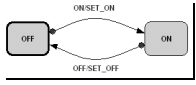
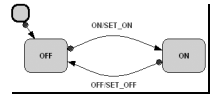
$$\frac{\text{Count Event condition } \{ \text{Data condition} \} / \text{Effect}}{\text{Trigger}}$$

- **Effect:** List of signal emissions:
`Sum(pre(?Sum)+1), AlarmOn, StopEngine`

79 © 2005 Estarel Technologies

Transition and Initial States

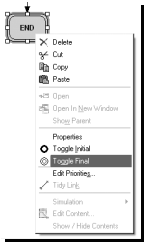
- At the initial cycle, the initial state's transitions are immediately fire-able.
- At the other cycles, the initial state behaves like any other state.
- **Exercise:** explain the difference between the 2 SSM below:

80 © 2005 Estarel Technologies

Transition and Terminal States

- A final state cannot have outgoing transitions: it is a *transient state*, not a state memory element where time can elapse.
- When a triggered transition reaches a final state, the SSM or macro-state it belongs to ends its execution



81 © 2005 Estarel Technologies

Transition: Summary

- At most one transition is fired in a cycle
- A transition can be fired if its trigger condition is true and:
 - If its source was active at the previous cycle, or
 - At the initial cycle if its source state is the initial state.
- Label of the form:
$$\text{Count Event condition } \{ \text{Data condition} \} / \text{Effect}$$
 where:
 - *Count*: occurrence count for the event condition
 - *Event condition*: Boolean expression testing signal presence statuses.
 - *Data condition*: Boolean expression testing values of signals and functions
 - *Effect*: list of signal emissions.

82 © 2005 Estarel Technologies



Specifications to safe embedded code
SCADE TRAINING
 derived by construction

The SSM State Behavior

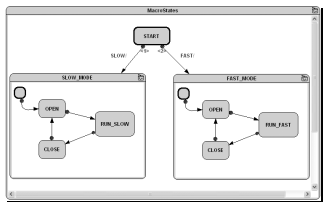
- Macro-states
- Actions in states



83 © 2005 Estarel Technologies

Macro-states

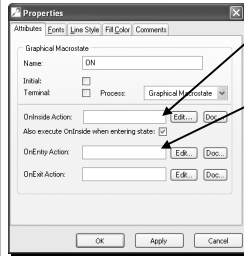
- State containing a sub-SSM
 - When activated, activates immediately the SSM it contains
- Hierarchy construct
 - To increase the readability and maintainability
 - To comply with software functional architecture constraints



84 © 2005 Estarel Technologies

Action Attributes

Actions can be performed upon the activity status of a state and are expressed the same way as transition effects



OnInside: Action list to execute when inside the state

OnEntry: Action list to execute when entering the state.

An initial state active at the first cycle behaves as if it is entered :
 - OnEntry actions are executed,
 - OnInside actions are not executed.

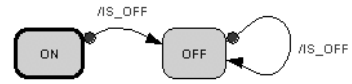
85

© 2005 Estrel Technologies ESTREL

Example: Sustaining a Signal (1/2)

Emit a signal whenever a state is active

- **Solution 1:** create a transition that emits the signal as long as the state is active



86

© 2005 Estrel Technologies ESTREL

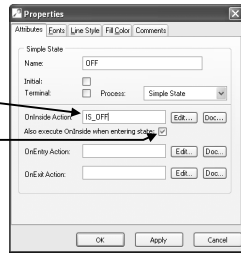
Example: Sustaining a Signal (2/2)

Emit a signal whenever a state is active

- **Solution 2:** create an *OnInside* action to the state



To execute the action *also* when entering the state



87

© 2005 Estrel Technologies ESTREL

Example

88

© 2005 Estrel Technologies ESTREL