Fast FSM with Synchronous Languages
An approach based on LUSTRE and on the COSMIC ISO 19761 standard

H. Soubra

IWSM MENSURA, October 23-25 2012, Ankara
Agenda

• Introduction: Challenges facing FSM

• Objective

• Synchronous Languages, LUSTRE & SCADE

• COSMIC based FFSM for LUSTRE

• Conclusions & future work
Introduction: Challenges facing FSM

• FSM has become an important task in software development projects of RTES.

• Still may be considered a complicated, tedious and time-consuming task when performed manually; and also because staff resources may not be available.

• Automating an FSM is a solution to help in applying it and using it.
Introduction: Challenges facing FSM

• It is **impossible** to automate an FSM method directly, because a method defines generic concepts and descriptions.

• An FSM procedure **specific** to an input type should be designed first.

• The **complexity** of automation is highly related to the complexity of the FSM procedure to be automated, which in turn is dependent on the complexity and the **integrity** of its input models.
Objective

To help in applying and using FSM:

Design simple and easy-to-apply FSM procedures for RTES!
The Synchronous Languages: industrial context

• The new family of French nuclear reactors: the most critical functions were realized by a computer system.

• The very first fully “fly-by-wire” aircraft: The Airbus A320 was designed.

• French TGV (very high speed train) were more and more computerized.
The Synchronous Languages

• Based on solid mathematical foundations for “correctly” designing safety-critical reactive real-time systems, where the notion of time is a crucial aspect.

• Allow defining systems time and concurrency properties directly in the functional requirements specifications.

• Give access to formal verification techniques.

• Full equivalence between various levels of representation is also guaranteed.
The Safety-critical Application Development Environment (SCADE)

• System design and modeling tool suite developed in 1995 and based on the synchronous language LUSTRE.

• SCADE allows the design of explicit safely-constructed formal models, where the interpretation of a model designed using SCADE is unique and reader-independent.

• SCADE fully supports industrial systems engineering processes -such as ISO 26262 - and it has a C code generator developed in compliance with DO-178B level A.
LUSTRE

• A functional data flow language

• Operates on flows of values
  
  Flow = an infinite sequence of values

• Any variable X represents a flow: X= \((x_0, x_1, ..., x_n, ...)\)
  where \(X_n\) is the value of X in the nth cycle

• Has two temporal operators:
  • “pre” (“previous”)
  • “->” (“followed by”)

©2013 H. Soubra
### LUSTRE: Flows & Cycles!

<table>
<thead>
<tr>
<th></th>
<th>Cycle 1</th>
<th>Cycle 2</th>
<th>Cycle 3</th>
<th>...</th>
<th>Cycle n</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A</strong></td>
<td>a1</td>
<td>a2</td>
<td>a3</td>
<td>...</td>
<td>an</td>
</tr>
<tr>
<td><strong>B</strong></td>
<td>b1</td>
<td>b2</td>
<td>b3</td>
<td>...</td>
<td>bn</td>
</tr>
<tr>
<td><strong>C=A+B</strong></td>
<td>c1 = a1 + b1</td>
<td>c2 = a2 + b2</td>
<td>c3 = a3 + b3</td>
<td>...</td>
<td>cn = an + bn</td>
</tr>
<tr>
<td><strong>pre(B)</strong></td>
<td>nil</td>
<td>b1</td>
<td>b2</td>
<td>...</td>
<td>bn-1</td>
</tr>
<tr>
<td><strong>A-&gt;B</strong></td>
<td>a1</td>
<td>b2</td>
<td>b3</td>
<td>...</td>
<td>bn</td>
</tr>
<tr>
<td><strong>A-&gt; pre(B)</strong></td>
<td>a1</td>
<td>b1</td>
<td>b2</td>
<td>...</td>
<td>bn-1</td>
</tr>
</tbody>
</table>

Where:
- \( C = A + B \)
- \( pre(B) \)
- \( A \rightarrow B \)
node *Name*(input:type)
returns (output:type)
var *loc*:type ;
let
  instruction1;
  instruction2;
tel
node DEVICE (SET: bool; CLK: bool; DELAY: int) returns (LEVEL: bool);

var count: int;

let
LEVEL = (count>0);

count = if SET then DELAY
    else if CLK then
        if false->pre(LEVEL) then pre(count)-1}else 0
    else (0->pre(count));
tel
... Using SCADE?
COSMIC & data flows!

• Software **functionality** is considered within the functional flows of data groups.

• Four distinct types of data movements:
  • ENTRY and EXIT: across a boundary.
  • READ and WRITE: exchange with the persistent storage.

• The measurement result corresponds to the functional size of the FUR of the software measured, and is expressed in COSMIC Function Points (CFP).
LUSTRE and COSMIC

• One can easily notice the notion of “Data Flows” in both.

• Functionality is associated to these data flows.

Hence a “natural” mapping must exist between the two!!
LUSTRE and COSMIC: Mapping!

1. LUSTRE Nodes receive, manipulate, and produce data flows = COSMIC Functional Processes

2. LUSTRE input flows = sequences of data values received by nodes = COSMIC Entry data movements

3. LUSTRE output flows = sequences of data values produced by nodes = COSMIC Exit data movements

4. LUSTRE manages the availability of past values of flows automatically = no data storage mechanisms are needed = no LUSTRE elements mapped to COSMIC Read nor Write data movements
LUSTRE and COSMIC: Mapping Rules!

<table>
<thead>
<tr>
<th>Rule number</th>
<th>LUSTRE Element</th>
<th>COSMIC Element</th>
<th>Rule description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Node</td>
<td>Functional Process (FP)</td>
<td>Identify 1 functional process for each Node identified in the system</td>
</tr>
<tr>
<td>2</td>
<td>Input Flow</td>
<td>Entry (E)</td>
<td>Identify a 1E data movement for each Input flow identified in a Node</td>
</tr>
<tr>
<td>3</td>
<td>Output Flow</td>
<td>Exit (X)</td>
<td>Identify a 1X data movement for each Output flow identified in a Node</td>
</tr>
</tbody>
</table>
### RULES FOR OBTAINING THE SIZE OF THE FP AND THE WHOLE SYSTEM

<table>
<thead>
<tr>
<th>Rule number</th>
<th>Rule description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Aggregate the CFP related to the data movements identified in a specific Node (FP) to obtain the functional size of that Node.</td>
</tr>
<tr>
<td>5</td>
<td>Aggregate the CFP related to the data movements of (identified in) the Nodes of (identified in) the whole system to obtain the functional size of that system.</td>
</tr>
</tbody>
</table>
The functional size of the simple Control Device
The functional size of the simple Control Device

<table>
<thead>
<tr>
<th>Name of Nodes/FP identified</th>
<th>Flows/ data group movements identified</th>
<th>Type of data group movements identified</th>
<th>Size (in CFP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEVICE</td>
<td>SET</td>
<td>E</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>CLK</td>
<td>E</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>DELAY</td>
<td>E</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>LEVEL</td>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td><strong>Total</strong></td>
<td></td>
<td><strong>4 CFP</strong></td>
</tr>
</tbody>
</table>
Automation Algorithm

1. Search for all the Nodes in the input model. Each Node is a Functional Process.
2. For each Functional Process identified:
   2.1 For each Input Flow, identify one Entry data movement.
   2.2 For each Output Flow, identify one Exit data movement.
3. Assign 1 CFP value for each Data Movement identified.
4. Aggregate the size of all the Data Movements inside each FP identified.
5. Aggregate the size of all the Functional Processes identified.
Why “Fast” FSM? 1/2

The proposed procedure for LUSTRE introduces 5 measurement rules to be applied!!
The automation algorithm proposed consists of 7 steps and sub steps!!
Conclusions

1. A **fast FSM** procedure for safety-critical real-time systems expressed using **LUSTRE** (COSMIC based)

2. A set of **5 simple rules** that allow speeding up the FSM process, reducing the workload of measurement specialists, as well as eliminating measurement delays.

3. This work also provides a basis for the development of **simple and fast automated** measurement tools.
Future work

- Implementing the fast measurement algorithm proposed to study and validate its performance

- Studying translation mechanisms from different modeling tools to LUSTRE/SCADE
Q & A

hassan.soubra@estaca.fr