Formal Modeling for UML/MARTE
Concurrency Resources

Authors:
P. Peñil H. Posadas E. Villar
Index

• Motivation
• Design Flow
• UML/MARTE Methodology
• Conclusions
• Future Work
Motivation
Motivation (1)

MARTE provides semantics to UML

Select a subset of MARTE

Relate UML/MARTE to SystemC

SystemC enables a link to Co-Design

ForSyDe

Formal Support

Marte

Generic Resource

SystemC

Co-Design
Motivation (2)

- **Massive Concurrency**
  - Data Dependencies
  - Relations

- **Characteristic of the interactions**
  - Formal Semantics
  - Univocal Description

- **Models of Computation & Communication (MoCCs)**
  - Behaviors Semantics Heterogeneity
Contribution

UML/MARTE Models

Abstraction

ForSyDe Formal Metamodel

Structural Modeling

Behavioral Modeling
Formal System Design

- ForSyDe formal metamodel
  - Process
  - Signals
    - Separation Communication-Computation
  - MoCC generic characteristics
    - Untimed MoCCs
    - Synchronous MoCCs
    - Timed MoCCs
Process ForSyDe

\[ p(s_1 \ldots s_n) = s'_1 \ldots s'_m \]

\( \pi \) partition function \( S_n \)

\( \pi' \) partition function \( S'_m \)

\( g() \) next-state function

\( f_1() \ldots f_j() \) output functions
Formal Notation

The partitions functions:

\[ \pi(\nu_n, s_n) = \langle a_n(z) \rangle \quad \pi'(\nu'_m, s'_m) = \langle a'_m(z) \rangle \]

where \( a_n(z) \) is a subsignal of \( S_n \).

The function \( \nu_n \) gives the size of \( a_n(z) \):

\[ \nu_n(z) = \gamma(\omega_q) \quad \nu'_m(z) = \text{length}(a'_m(z)) \]

\[ \nu_n(0) = \text{length}(a_n(0)); \quad \nu_n(1) = \text{length}(a_n(1))... \]

The outputs are calculated:

\[ f_\alpha((a_1...a_n), \omega_q) = (a'_1...a'_m) \]

And the next internal state:

\[ g((a_1...a_n), \omega_q) = \omega_{q+1} \]
UML/MARTE Methodology
Structural Modelling

System

<<ConcurrencyResource>>: models the concurrent computation

<<CommunicationMedia>>: models the communication
Communication Abstraction

ForSyDe Signals

\[ \text{abstraction} \]

\[ \text{<<CommunicationMedia>>} \]

No insert any change in the data sequence:

- No data losses or injections
- No data values changes
Behavioral Modeling

• Concurrent element described by a Finite State Machine
  – UML Finite State Machines
    • Explicit States
    • Modeling behavior State denoted by the label *do* by an UML Activity Diagram
  – UML Activity Diagram
    • Implicit States as a sequence of actions
Activity Diagram

• In each state
  – A set of inputs are received
  – These inputs are computed
  – The computation results are generated
  – The concurrency resource calculates its new internal state
Activity Diagram(2)

ConcurrencyResource

Receive_1

multiplicity[p..q]

m

Receive_N

multiplicity[r..s]

function_i

multiplicity[a..b]

multiplicity[c..d]

Send_1

Send_M

AcceptEventActions

ω_j = \begin{pmatrix} P_j \\ D_j \end{pmatrix}

SendObjectAction
Activity Diagram(3)

• **AcceptEventAction**
  – Call to a specific method with a specific data-receiving behavior
  – A new datum is available

• **SendObjectAction**
  – Call to a specific method with a specific data-sending behavior
  – A new datum is generated
Computation Abstraction

- The data received: \( a_1 \ldots a_N \) ForSyDe subsignals
- The data send: \( a'_1 \ldots a'_N \) ForSyDe subsignals
- The function \( \text{function}_i \) action: \( f_\alpha \) ForSyDe function
- The Concurrency Resource State:
  - \( P_j \); segments of the behavioural modelling between two waiting states
  - \( D_j \); internal values that characterizes the state
- \( \{ P_j, D_j \} \); \( \omega_j \) ForSyDe state
- The function \( g() \) calculates the new \( \{ P_{j+1}, D_{j+1} \} \)
Computation Abstraction(2)

- The multiplicity values are abstracted as:

\[
\begin{align*}
\nu_1(z) &= \gamma(\omega_j) = \begin{cases} p \\ q \end{cases} \\
\nu_n(z) &= \gamma(\omega_j) = \begin{cases} r \\ s \end{cases}
\end{align*}
\]

\[
\begin{align*}
\nu'_1(z) &= \text{length}(a'_1) = \begin{cases} a \\ b \end{cases} \\
\nu'_M(z) &= \text{length}(a'_M) = \begin{cases} c \\ d \end{cases}
\end{align*}
\]

\[
\text{length}(f_j(a_1 \ldots a_N), \omega_j) = \begin{cases} ... \end{cases}
\]
Computation Abstraction(3)

- The advance of time is a totally ordered set of *evaluation cycles* (ec).
- In each ec “a process consumes inputs, computes its new internal state, and emits outputs”.
- The interpretation of a ec depends on the time domain capture in the models.
Example
Example(1)
Example(2)
data inputs \( a_{cm1}(z), a_{cm2}(z) \)

data outputs \( a'_{cm3}(z), a'_{cm4}(z) \)

\[
g(\omega_0, a_{cm1}) = \begin{cases} 
\omega_1 \\
\omega_2 
\end{cases}
\]

\[
g(\omega_1, (a_{cm1}, a_{cm2})) = \omega_0
\]
Example(4)

In the state $\omega_0$:

\[
\nu_{a_{cm1}}(0) = \gamma(\omega_0) = 2
\]

\[
\nu_{a_{cm4}}(0) = \text{length}(\text{function1}(a_{cm1}(0), \omega_0)) = \text{length}(a'_{cm4}(0)) = 1
\]

In the state $\omega_1$:

\[
\nu_{a_{cm1}}(1) = \gamma(\omega_1) = 1
\]

\[
\nu_{a_{cm2}}(0) = \gamma(\omega_1) = 3
\]

\[
\nu_{a_{cm3}}(0) = \text{length}(\text{function2}((a_{cm1}(1), a_{cm2}(0)), \omega_1)) = \text{length}(a'_{cm3}(0)) = 1
\]

\[
\nu_{a_{cm4}}(1) = \text{length}(\text{function2}((a_{cm1}(1), a_{cm2}(0)), \omega_1)) = \text{length}(a'_{cm4}(1)) = 2
\]
Conclusions

• The need of a formalism to UML/MARTE models that supports the generation of executable specifications (SystemC)
• ForSyDe can provides this formal support