ForSyDe: A Denotational Framework for Heterogeneous Models of Computation

Axel Jantsch, Ingo Sander, Jun Zhu
Royal Institute of Technology, Stockholm

Models of Computation and Communication
ARTIST2 Workshop
November 2006
ForSyDe Features

Processes
- Communicate through signals only;
- Functional
- State-full
- Blocking read
- Partition input and output signals
- Evaluate when required input is available

Signals:
- Sequences of events
- Preserve event order
- Have one writer and multiple readers
- Untimed MoC: Events are partially ordered
- Discrete Time MoCs: Signals carry timing information
The ForSyDe Design Flow

Ideal System Model
No resource limitation on processors, communication bandwidth and delay memory

Implementation Model
with Finite resources
processors, HW blocks, reconfigurable resources, buffers, communication architecture, schedulers, arbiters

C Program
VHDL Design
SystemC Model
Process Constructors

process = constructor + function + initial_state + invocation_condition

\[ f \]

Invokation condition

Process functionality

[Diagram showing process components: input signal \( \gamma \), state, output signal]
Layered View of Process Constructors

Models of Computation
- Untimed MoC (Datflow, SDF, Rendezvous)
- Synchronous MoC (Perfectly, Clocked)
- Discrete Time MoC
- Soon: Continuous Time MoC

Process Combinators
- Sequential Composition
- Parallel Composition
- Feed-back Composition

Process Constructor Types
- State-less Processes
- FSM Machines
- Zip / Unzip Processes
- Sources and Sinks
The \textit{mapU} Process Constructor

\[ \text{mapU}(c, f) = p \]

where

\[ p(\dot{s}) = \dot{s}' \]
\[ f(\dot{a}_i) = \dot{a}'_i \]
\[ \pi(\nu, \dot{s}) = \langle \dot{a}_i \rangle, \nu(i) = c \]
\[ \pi(\nu', \dot{s}') = \langle \dot{a}'_i \rangle, \nu'(i) = \# f(\dot{a}_i) \]
The \textit{mapU} Process Constructor

\[
\text{mapU}(c, f) = p \\
\text{where } p(\dot{s}) = \dot{s}' \\
f(\dot{a}_i) = \dot{a}'_i \\
\pi(\nu, \dot{s}) = \langle \dot{a}_i \rangle, \nu(i) = c \\
\pi(\nu', \dot{s}') = \langle \dot{a}'_i \rangle, \nu'(i) = \# f(\dot{a}_i)
\]

Example:

\[
A = \text{mapU}(c, f) \\
\text{where } \\
c = 1 \\
f(x) = 2x
\]
Definition of a Model of Computation

The **Untimed Model of Computation (Untimed MoC)** is defined as Untimed MoC = \((C, O)\), where

\[
C = \{ \text{mapU}, \text{scanU}, \text{scandU}, \text{mealyU}, \text{mooreU}, \text{zipU}, \text{zipUs}, \text{zipWithU}, \text{unzipU}, \text{sourceU}, \text{sinkU}, \text{initU} \}
\]

\[
O = \{ \|, \circ, \text{FB}_P \}
\]

- Synchronous Model of Computation
- Clocked Synchronous Model of Computation
- Discrete Time Model of Computation
The Integrated Model of Computation (Integrated MoC) is defined as Integrated HMoC = (M, C, O), where

\[ M = \{ \text{U-MoC, S-MoC, CS-MoC, T-MoC} \} \]

\[ C = \{ \text{intSup, intSdown, intTup, intTdown, stripT2S, stripT2U, stripS2U, insertS2T, insertU2T, insertU2S} \} \]

\[ O = \{ \|, \circ, FBP \} \]
Process Migration

Untimed Domain

$p_1 \rightarrow p_2 \rightarrow p_{\text{insertU2S}} \rightarrow p_3$

Synchronous Domain

$p_1 \rightarrow p_{\text{insertU2S}} \rightarrow p'_2 \rightarrow p_3$
Process Refinement - FM Software Radio Example

SDF Model

Synchronous Model
Process Refinement - FM Software Radio Example

SDF domain

Communication layer:

\[
\begin{align*}
\text{mapSDF } f_{\text{sum}} \quad &\text{sIn} = \text{sOut} \\
\end{align*}
\]

Synchronous domain

\[
\begin{align*}
\text{mapSDF } f_{\text{sum}} \quad &\text{sIn} = \text{sOut} \\
\end{align*}
\]

Computation layer:

\[
\begin{align*}
\text{mapSDF} \\
\end{align*}
\]

Library of Algorithms:

```c
// ***/
// ......
```
Transformation Rules - Scheduling

- A combinational process with $m$ input signals is modeled with $\text{zipWithSY}_m(f)$
- In each event cycle the function $f$ is applied to the current values of the input signals
- A large amount of computational resources may be required for these processes
Transformation Rules - Scheduling

Combinational process

\[ f(x_1, ..., x_m) = x_1 \otimes x_2 \otimes ... \otimes x_m \]

If the following schedule using only one computational unit can be derived:
Transformation Rules - Scheduling

$\text{zipWithSY}_m (f)$

SerialClockDomain

$\text{Parallel/Serial}$ $\text{Downsample}$
Transformation Rules - Scheduling
Transformation Rules - Scheduling

Parallel/Serial

Downsample

\[ \text{Parallel/Serial} \quad = \quad \text{Downsample} \]

\[ \text{zipWithSY}_m (f) \quad \rightarrow \quad \text{delaySY}_1 (m_0) \quad \rightarrow \quad o' \]
NoC Simulator Case Study
ForSyDe Status

Ideal System Model
No resource limitation on
- processors
- communication bandwidth and delay
- memory

Implementation Model
with Finite resources
- processors, HW blocks,
- reconfigurable resources
- buffers
- communication architecture
- schedulers, arbiters

Stable modeling technique
ForSyDe library based on Haskel

Set of transformations defined
Verification of local transformations proposed

Mapping for synchronous HW defined
Mapping for sequential SW defined
ForSyDe Status

Ideal System Model
No resource limitation on processors, communication bandwidth and delay, memory

Implementation Model
with Finite resources
processors, HW blocks, reconfigurable resources, buffers, communication architecture, schedulers, arbiters

Set of transformations defined
Verification of local transformations proposed

Mapping for synchronous HW defined
Mapping for sequential SW defined

Stable modeling technique
ForSyDe library based on Haskel

Ongoing work
Definition of a CT-MoC
Modeling of non-functional properties
Modeling of adaptive resources
GME based tool support for transformations
Communication refinement method

C Program

17