Heterogeneous Function Composition to Eliminate a Class of Direct Relationships in Software Components of Dynamic Systems

Pieter J. Mosterman
Senior Research Scientist
Design Automation Department
The MathWorks
Adjunct Professor
School of Computer Science
McGill

The problem!

- Demo … 😊
The problem!

- Demo ... 😊

- What happened?!
Basic Simulink syntax

- An example model …

\[
\begin{align*}
\text{sin} & \quad f() \quad \sum \quad z^{-1} \quad \text{out} \\
\text{fcn} & \quad -1
\end{align*}
\]

Basic Simulink syntax

- Network of blocks with directed lines connected between ports
The execution hierarchy

- The lines reflect input/output relations
- The (nonvirtual) blocks are dynamic systems

### Dynamic System
- State: \( x(t) \)
- Parameters: \( P \)

\[
\frac{dx(t)}{dt} = g(u(t), x(t), P, t)
\]

Output: \( y(t) = f(u(t), x(t), P, t) \)

### Example block implementation

- A unit delay

\[
u(t) = x(t) \quad \rightarrow \quad Z^{-1} \quad y(t) = x(t)
\]

State, \( x(t) \)

\[
x(0^-) = P_{dc}
\]

```cpp
class UnitDelayBlock : public Block {
public:
    ErrorStatus BlockDrawIcon() {
        // Draw '1/z' on the icon
        …………………
    }

    BlockParameterData BlockGetParameterData() {
        // Return initial_condition as block data
        …………………
    }

    ErrorStatus BlockOutput() {
        // Implement \( y(t) = x(t) \)
        …………………
    }

    ErrorStatus BlockUpdate() {
        // Implement \( x(t) = u(t) \)
        …………………
    }

private:
    double initial_condition;
};```

10/25/2010
The basic structure of a discrete time block

- Data dependencies between
  - Input signals, output signals, current state, new state, update function and output function

Data dependencies to create a sorted list for efficient execution

- Static dependency analysis

Sorted list:
0:0 sin
0:1 fcn
1:0 f()
0:2 z^{-1}
0:3 Σ
0:4 out

Generate block sorted list
Algebraic dependencies

- What if we replace the delay block by a gain?

\[
\text{fcn} \rightarrow -1 \leftarrow \text{fcn} \rightarrow f() \rightarrow \Sigma \rightarrow \text{out} \leftarrow K
\]

- Strongly connected components

\[
y(t) = u(t) + Ky(t) \rightarrow y(t) = \frac{u(t)}{1-K}
\]

Sorted list:
- 0:3 Out
- 0:2 AlgLoop
- 2:0 \Sigma
- 2:1 K
- 0:1 fcn
- 1:0 f()
Algebraic dependencies

- Strongly connected components

\[ y(t) = u(t) + Ky(t) \]
\[ \rightarrow y(t) = \frac{u(t)}{1-K} \]

No embedded code generation!

Sorted list:
0:0 sin
0:1 fcn
1:0 f()
0:2 AlgLoop
2:0 \sum
2:1 K
0:3 Out

Agenda

- Simulink preliminaries
- Executing a Simulink model
- Dealing with hierarchy
- Heterogeneous composition
- An implementation in Simulink
- Conclusions
Let’s assume single-tasking execution

- Allows multi-rate systems
- All blocks run in a single task
  - Single execution time line
  - Base rate at greatest common denominator
  - Blocks execute when they have a sample hit
  - No data integrity issues

A simulation algorithm for networks of dynamic systems

\[
\begin{align*}
\text{Input, } & u(t) \\
\text{Dynamic System:} & \quad \text{State: } x(t), \text{ Parameters: } P \\
\text{Output, } & y(t) \\
\text{Output:} & \quad y(t) = f_y(u(t), x(t), P, t) \\
\text{Update:} & \quad x_f(t) = f_x(u(t), x(t), P, t) \\
\text{Derivative:} & \quad \frac{dx_f(t)}{dt} = f_x(u(t), x(t), P, t)
\end{align*}
\]
First generate the sorted list

- Determine data dependencies
  - Based on a direct feedthrough (df) flag

![Diagram of data dependencies]

Generate execution lists for all of the block methods

```plaintext
Sorted list: 0:0 F 0:1 E 0:2 D 0:3 A 0:4 B 0:5 C
Output execution list: 0:0 F (0.1) 0:1 E (0.1) 0:2 D (1.0) 0:3 A (0.1) 0:4 B (1.0) 0:5 C (1.0)
Update execution list: 0:0 F (0.1) 0:1 E (0.1) 0:2 D (1.0)
```
Repeatedly evaluate the execution lists

- Initialize
- Time < Stop time
- Execute active output execution list methods
- Execute active update execution list methods
- Time = Time + Step size
- Simulation complete

Output execution list:
- 0:0 F (0.1)
- 0:1 E (0.1)
- 0:2 D (1.0)
- 0:3 A (0.1)
- 0:4 B (1.0)
- 0:5 C (1.0)

Update execution list:
- 0:0 F (0.1)
- 0:1 E (0.1)
- 0:2 D (1.0)

Logical time: 0.0 0.1 0.2 1.0 1.1
Output: FEDABC FEA FEA...
Update: FED FE FE...
Physical time: 0.0 0.1 0.2 1.0 1.1

Agenda

- Simulink preliminaries
- Executing a Simulink model
- Dealing with hierarchy
- Heterogeneous composition
- An implementation in Simulink
- Conclusions
### Hierarchy

- **Virtual** blocks to organize graphical hierarchy
  - Referential transparency; no semantic bearing
- **Nonvirtual** blocks to organize
  - Execution hierarchy
  - Data scope hierarchy

### A graphical hierarchy

- Group blocks in a **virtual** subsystem
  - Subsystem C does not appear in the sorted list
Let’s create a component …

- Make the virtual subsystem **nonvirtual**
  - Becomes a dynamic system in its own right

- Does that affect sorting?

Let’s start with a virtual subsystem …

- Dependencies derived from df flag
Make subsystem an atomic component

- Direct feedthrough moves to component level!

Now we have a dependency cycle!
Make subsystem an atomic component

- Now we have a dependency cycle!

How did the direct feedthrough flag end up there?! And how can we fix it?!

Agenda

- Simulink preliminaries
- Executing a Simulink model
- Dealing with hierarchy
- Heterogeneous composition
- An implementation in Simulink
- Conclusions
Gain followed by delay

General

Gain

Delay

Gain/Delay

Gain followed by delay

Gain/Delay

Gain followed by delay
Gain followed by delay

Gain/Delay

\[ u \rightarrow f_{op} \rightarrow x' \rightarrow f_{op} \rightarrow x \rightarrow y \]

Gain followed by delay

Gain/Delay

\[ u \rightarrow f_{op} \rightarrow x' \rightarrow f_{op} \rightarrow x \rightarrow y \]
How do we create a component?

Homogeneous composition
Put the component in context

We have a dependency cycle!
Is there another way …?

Heterogeneous composition!
Putting it in context again

![Diagram](image)

No cycles!

Agenda

- Simulink preliminaries
- Executing a Simulink model
- Dealing with hierarchy
- Heterogeneous composition
- An implementation in Simulink
- Conclusions
How do we fit this into the model compilation machinery?

- Create sorted list after clearing df flag

![Diagram](image1)

How do we fit this into the model compilation machinery?

- Create sorted list after clearing df flag

![Diagram](image2)
How do we fit this into the model compilation machinery?

- Create sorted list after clearing df flag
- Mark where output for update should be used

```c
56 /* Model output function */
57 void gaindelay_output (void)
58 {
59    real rtb_E;
60
61    /* Outputs for atomic Subsystem: '<Root>/C' */
62    gaindelay_E();
63
64    /* and df Outputs for Subsystem: '<Root>/C' */
65
66    /* Gain: '<Root>/E' */
67    rtb_E = gaindelay_E.E_Gain * gaindelay_E.Delay;
68
69    /* Sum: '<Root>/B' incorporates:
70       * Constant: '<Root>/A'
71       */
72    gaindelay.E_B = gaindelay_E.A_Value + rtb_E;
73 }
```

What does the generated code look like?

Output execution list:

- 0:0 C
- 0:0 Delay
- 0:1 A
- 0:2 E
- 0:3 B

Update execution list:

- 0:0 C
- 0:0 Gain
- 0:1 Delay
What does the generated code look like?

Output execution list:

0:0 C(op)
0:0 Delay(op)
0:1 A(op)
0:2 E(op)
0:3 B(op)

Update execution list:

0:0 C(up)
0:0 Gain(op)
0:1 Delay(up)

What does the generated code look like?

Output execution list:

0:0 C(op)
0:0 Delay(op)
0:1 A(op)
0:2 E(op)
0:3 B(op)

Update execution list:

0:0 C(up)
0:0 Gain(op)
0:1 Delay(up)
The general analysis

- For an inport block
  - Depth-first search for a df path to output
  - If a port without df is reached, mark visited nodes for potential move of output method

- If output found
  - Clear all blocks visited from the initial input port
The general analysis

- For an inport block
  - Depth-first search for a df path to output
  - If a port without df is reached, mark visited nodes for potential move of output method
- If output found
  - Clear all blocks visited from the initial input port
The synthesis part of the algorithm

- Move marked df blocks into an atomic subsystem

Sorted list:

```
0:0 A
0:1 C
1:0 TmpSyn...
0:0 Gain
0:1 Gain1
0:2 Gain2
1:1 Delay
1:2 Delay1
```

Output execution list:

```
0:0 A(op)
0:1 C(op)
1:0 Delay(op)
1:1 Delay1(op)
```

Update execution list:

```
0:0 C(up)
0:0 TmpSyn...(op)
0:0 Gain(op)
0:1 Gain1(op)
0:2 Gain2(op)
0:1 Delay(up)
0:2 Delay1(up)
```

Nested cyclic dependencies

Sorted list:

```
0:0 A
0:1 C
1:0 C1
0:0 C2
0:0 Gain3
0:1 Delay3
0:1 B2
0:2 D2
1:1 Delay1
0:2 D
0:3 B
```

Output execution list:

```
0:0 A(op)
0:1 C(op)
1:0 Delay1(op)
0:2 D(op)
0:3 B(op)
```

Update execution list:

```
0:0 C(up)
0:0 C1(op)
0:0 C2(op)
0:0 Delay3(op)
0:1 B2(op)
0:2 D2(op)
0:1 D(op)
0:0 C2(up)
0:0 Delay3(up)
0:2 Delay1(up)
```
Nested cyclic dependencies

Sorted list:
0:0 A
0:1 C
1:0 C1
0:0 C2
0:0 Gain3
0:1 Delay3
0:1 B2
0:2 D2
1:1 Delay1
0:2 D
0:3 B

Output execution list:
0:0 A(op)
0:1 C(op)
1:0 Delay1(op)
0:2 D(op)
0:3 B(op)

Update execution list:
0:0 C(up)
0:0 C1(up)
0:0 C2(up)
0:0 Gain3(up)
0:1 Delay3(up)
0:1 B2(up)
0:2 D2(up)
0:1 C1(up)
0:1 C2(up)
0:0 Gain3(up)
0:1 Delay3(up)
0:2 Delay1(up)

Scope of applicability

- Applies to derivative and zerocrossing as well
- But, the df path has to be breakable

- In general, four basic data dependency classes
Agenda

- Simulink preliminaries
- Executing a Simulink model
- Dealing with hierarchy
- Heterogeneous composition
- An implementation in Simulink
- Conclusions

Conclusions

- Simulink as a network of dynamic systems
  - Method set to generate behavior (output, update, …)
- Lists for the execution phases
  - `df` to sort according to data dependencies
  - Block methods and sample times to create lists
- Atomic subsystems for componentization
  - Heterogeneous composition to avoid dependencies
  - Heterogeneous execution lists (flat and hierarchical)
  - Scoped the class of systems where this applies
References


Acknowledgments

John E. Ciolfi  
MathWorks

Ben Denckla  
Independent Thinker

Many thanks for their continuing collaboration!