MATLAB EXPO 2018

New Perspective for Large and Complex Production Software Development

대규모 SW 개발에 적합한 모델링 패턴 및 코드 생성 방안

류성연 차장
## Issues for Large-scaled Embedded Software Development

<table>
<thead>
<tr>
<th>Issues</th>
<th>Work Phase</th>
<th>Modeling</th>
<th>Code Generation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Complexity</td>
<td></td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>2. Integration</td>
<td></td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>(Reusability + Scalability)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. Scheduling</td>
<td></td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>4. Multi-instantiation</td>
<td></td>
<td></td>
<td>✔️</td>
</tr>
</tbody>
</table>
What to Consider for Model-Based Design

- Component-based design
- Integration in a composition level
- Component scheduling
- Code generation on SW frameworks
- Generated code customization
For Software Modeling Patterns

Example: Throttle body control system

Throttle Control Model

Acceleration pedal Input

Throttle body controller

Signal processing

PID controller

Actuation command

Code generation model

Plant (throttle body)
Inadequate Software Modeling & Code Generation

❖ Not reflecting SW architecture
1) Modeling in one Simulink file
2) Generated code in one function and one file
3) Hard to analyze interfaces among units
4) Unit execution orders are predefined

Not adequate for larger-scale software !!
Let’s Start from Software Architecture

- If there are many models from other developers or teams…
Integration in a Composition Level

- Modeling based on component and integration as a composition using Model Reference
What the Model Reference…?

- Model Reference enables to design models based on SW component

![Model Reference interface and block diagrams]
Creating Separate Test Harness Model

- Your model for code generation is separate from test harness model

Only for testing (unit test/integration test)

Only for code generation

MATLAB EXPO 2018
## Issues for Large-scaled Embedded Software Development

<table>
<thead>
<tr>
<th>Issues</th>
<th>Work Phase</th>
<th>Modeling</th>
<th>Code Generation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Complexity</td>
<td></td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>2. Integration (Reusability + Scalability)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. Scheduling</td>
<td></td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>4. Multi-instantiation</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

MATLAB EXPO 2018
SW Scheduling for Larger-scale Software

- Requirement to analyze the results according to scheduling

Is your execution sequence always simple?

Or not?

OS/scheduler
Multicore execution
Multi tasking

How guarantee ...

(task1)

t
t+1
t+2
t+3
t+4
t+5

(task2)
a
a+1
a+2
Typical Workflow for Software Integration and Scheduling

- Collecting models for code generation with considering scheduling

Now how do I integrate to base code?
Collect Entry Point Functions for Each Component

File: c1.c
```c
#include "c1.h"

dw rtDW;
ExtU rtU;
ExtY rtY;

void c1_step0(void)
{
    rtY.c1_10ms_out = 2.0 * rtU.c1_10ms_in;
}

void c1_step1(void)
{
    real_T rtb_RateTransition;
    rtb_RateTransition = rtb_RateTransition_Buffer[rtbDW.
    rtb_RateTransition_ActiveBufIdx = 0];
}

void c1_async(void)
{
    rtb_RateTransition_Buffer[rtbDW.
    rtbU.c1_async_in;  
    rtb_RateTransition_ActiveBufIdx = 0; 
}

void c1_initialize(void)
{
}

void c1_terminate(void)
{
}
```

File: c2.c
```c
#include "c2.h"

dw rtDW;
ExtU rtU;
ExtY rtY;

void c2_step0(void)
{
    rtY.c2_10ms_out = rtU.c2_10ms_in +
}

void c2_step1(void)
{
    real_T rtb_In1;
    rtb_In1 = rtU.c2_10ms_in;
    rtb_RateTransition_Buffer[rtbDW.
    rtb_RateTransition_ActiveBufIdx = 0];
}

void c2_async(void)
{
    rtb_RateTransition = rtb_RateTransition_Buffer[rtbDW.
    rtbV.c2_async_out = rtbU.UnitDelay_C;
    rtb_RateTransition_DSTATE = rtbU.UnitTrans;
}

void c2_initialize(void)
{
}

void c2_terminate(void)
{
}
```

And, how do I create scheduling orders?

**c1**
- c1_step0()
- c1_step1()
- c1_async()

**c2**
- c2_step0()
- c2_step1()
- c2_async()
Application Integrated to Base Software

- Integrate entry point functions from components with run-time environment

```
但，我想要了解在集成之前调度效果!
```

- OS/Scheduler
  - Event Handler 1
    - c1_step0()
    - c1_step1()
    - c1_async()
  - Event Handler 2
  - Periodic Task 10ms
    - c2_step0()
    - c2_step1()
    - c2_async()
  - Periodic Task 20ms
Software Testing with Scheduling Effects

Export Function

→ Scheduler makes periodic events (ex. 5ms/10ms)
Redesigned Model with Scheduler and Export Functions

Export Function

Throttle body controller

Scheduler

Signal processing

PID controller

Actuation command
Demo: SW Modeling with Export Functions

Export Function
Testing Scheduling Effects from Different Patterned Models

**Schedulable Component**

- What if there are any other models with different modeling patterns?

**Export-function models**

**Rate-based models**

To integrate, change to export functions

**No, impossible...**

Wow, How do I resolve this struggles?
Creating Schedulable Component from Model Reference

**Schedulable Component**

Rate-based model: Model executing in periodic sampling rate

Event port for scheduling ➔ This port is not for code generation but only for simulation
Demo: SW Modeling with Schedulable Components

**Schedulable Component**
<table>
<thead>
<tr>
<th>Issues</th>
<th>Work Phase</th>
<th>Modeling</th>
<th>Code Generation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Complexity</td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>2. Integration</td>
<td>(Reusability + Scalability)</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>3. Scheduling</td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>4. Multi-instantiation</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Some application SW does not process external signals directly. Or…
Modeling for Access to Hardware Resources

Simulink Function

- Some application SW does not process external signals directly. Or…
- External signals are processed in BSW or HAL and accessed by applications
- Application software use APIs to request or send data

Basic software

Only for simulation
Access to Shared Resources with Simulink Functions

Simulink Function
Demo: SW Modeling with Simulink Functions

Simulink Function
## Issues for Large-scaled Embedded Software Development

<table>
<thead>
<tr>
<th>Issues</th>
<th>Work Phase</th>
<th>Modeling</th>
<th>Code Generation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Complexity</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>2. Integration (Reusability + Scalability)</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>3. Scheduling</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>4. Multi-instantiation</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>
Self-Study Resources for Embedded Code Generation

- Embedded Coder Quick Start Video

- Simulink와 Embedded Coder를 이용한 최적 코드 생성 (MATLAB Expo 2017)

- C code generation from Simulink model (webinar)

- Other Embedded Coder Videos
Multi-instantiation for Large-scaled Software

- **Issues**
  - Limited resources for code size
  - Maintenance problem

- **Solution**
  - Calling the same functions through multi-instantiation when generating code
Configuration for Multi-instantiation

C

Code interface

// Model step function
void sldemo_md1ref_counter_r_step(void)
{
  // Code...
}

C++

// Code...

Before (non-reusable)

/* Model step function */
void sldemo_md1ref_counter_r_step(void)
{
  // Code...
}

After (reusable)

/* Model step function */
void sldemo_md1ref_counter_r_step(RT_MODEL_sldemo_md1ref_counter_r *const
  sldemo_md1ref_counter_r_M, ExtU_sldemo_md1ref_counter_r_T
  *sldemo_md1ref_counter_r_U, ExtY_sldemo_md1ref_counter_r_T
  *sldemo_md1ref_counter_r_Y)
{
  // Code...
}

Creating instances

// model instance variable for 'Root/CounterA'
sldemo_md1ref_counter_rModelClass CounterMDLOBJ1;

// model instance variable for 'Root/CounterB'
sldemo_md1ref_counter_rModelClass CounterMDLOBJ2;

// model instance variable for 'Root/CounterC'
sldemo_md1ref_counter_rModelClass CounterMDLOBJ3;
### Issues for Large-scaled Embedded Software Development

<table>
<thead>
<tr>
<th>Issues</th>
<th>Work Phase</th>
<th>Modeling</th>
<th>Code Generation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Complexity</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>2. Integration (Reusability + Scalability)</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>3. Scheduling</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>4. Multi-instantiation</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

MATLAB EXPO 2018
Emergence of the Software Framework

Conform to a (standard) framework (ex. AUTOSAR)

Scheduler  Memory  COM  I/O
Issue 1: Mapping Generated Code to Software Frameworks

- There are many frameworks (ex. AUTOSAR, ARINC, etc) including bare-metal software
- Solution: Configuration management of code mapping information apart from S/W frameworks
  - Just using code mapping information according to requirements

Maximizing model reusability for S/W maintenance, cost reduction

Requirement of Project A

Requirement of Project B

Code generation for bare-metal Software

Code generation for AUTOSAR ARINC, POSIX, etc
Issue 2: Code Packaging for Efficient Code Management

- There are needs to manage efficient tuning parameters in large-scaled S/W
  - To change only tuning parameters according to requirements
  - Efficient code maintenance

- Solution
  - Configuring storage class for code generation
  - Easy customization using GUI
Effective code generation customization as to SW frameworks

Code Perspective
- Easy configuration for generated code into any C/C++ SW framework

Embedded Coder Dictionary
- GUI for custom code definitions
  - Function template
  - Storage class
  - Memory section
Embedded Coder Dictionary

• **Storage classes**
  - Control the code generated for model data (I/O, signals, data stores, states, parameters)

  - Storage allocation and scope (ex, global, extern, static, register, pointer ...)
  - Bitfield, Constant, Pre-processor, ...
  - Export to or import from external files, ...
  - Etc.: Structure type, Get/Set APIs, ...
• Function customization templates
  - Control naming of model entry-point functions (ex. `model_step`)
  - Apply memory sections to the entry-point functions

• Memory section
  - Control the placement of data and functions in memory (ex. `#pragma`)
Code Perspective

1) Embedded Coder Quick Help
   • Embedded Quick Start
   • Hyperlink to configuration and documents
   • Help video clips

2) Property Inspector
   • Configure model properties

3-1) Model Data Editor
   • Inspect and edit data items
   • Configure storage class of each blocks or signals

3-2) **Code Mapping Editor**
   • Configuring model data elements and entry-point functions for code generation comprehensively
Example on Issue 1: Code mapping implementation

- Code mapping to embedded S/W frameworks
  - Entry-point functions and interfaces can be customized according to SW architecture
Example on Issue 2: Partition and Modularize Generated Code

- Tuning parameter modularization example with customizing storage class

```c
typedef struct {
    int D_Gain;
    int I_Gain;
    int P_Gain;
} rt_SI_Struct_type;
```
Partition and Modularize Generated Code

Example: Tuning parameter modularization
## Issues for Large-scaled Embedded Software Development

<table>
<thead>
<tr>
<th>Issues</th>
<th>Work Phase</th>
<th>Modeling</th>
<th>Code Generation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complexity</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Integration (Reusability + Scalability)</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Scheduling</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Multi-instantiation</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>
Key Takeaway

- SW modeling pattern importance for effective code generation
  - Component-based modeling
  - Integration in a composition level using Model Reference
  - Export functions/ scheduling components modeling patterns
  - Simulink Function models for access to hardware resources

- Code generation customization framework
  - Code Perspective
  - Embedded Coder Dictionary