A Code Generation Framework for Distributed Real-Time Embedded Systems

M. Bambagini and M. Di Natale

17th International Conference on Emerging Technologies & Factory Automation
Krakow, Poland, September 17-21, 2012
Outline

1. Introduction
2. Proposed framework
3. Code generation issues
4. Evaluation example
5. Conclusions
Introduction

Model creation and analysis tools have been helping engineers to face the design complexity of the current systems.
Introduction

Model creation and analysis tools have been helping engineers to face the design complexity of the current systems.

The two most widespread approaches in embedded computing are:

- Model Based Design (MBD), Matlab/Simulink/Scade
  - formally models the execution semantics
  - lets designers simulate and test the model since the earliest design stages
  - provides facility for the code generation (only for single-core)
Model creation and analysis tools have been helping engineers to face the design complexity of the current systems.

The two most widespread approaches in embedded computing are:

- **Model Based Design (MBD), Matlab/Simulink/Scade**
  - formally models the execution semantics
  - lets designers simulate and test the model since the earliest design stages
  - provides facility for the code generation (only for single-core)

- **Model Driven Architecture (MDA), SysML/MARTE**
  - models computational and communication platforms and software architecture
  - issues in the formal definition of semantics
  - limited capability of simulation, verification and SW generation
Proposed framework

We propose a modeling framework/methodology that bridges the gap between MBD and MDA approaches while:

▶ providing separation of concerns between functional modeling and platform modeling
  ▶ improves portability and extensibility
  ▶ enables the definition of functional components with re-targetable interfaces
▶ exploiting the best-in-class software: Matlab/Simulink and Eclipse/EMF

M. Bambagini
Proposed framework

We propose a modeling framework/methodology that bridges the gap between MBD and MDA approaches while:

▶ providing separation of concerns between functional modeling and platform modeling
  ▶ improves portability and extensibility
  ▶ enables the definition of functional components with re-targetable interfaces

▶ exploiting the best-in-class software: Matlab/Simulink and Eclipse/EMF

▶ generating a code implementation for distributed systems with only changes in the platform and mapping models

▶ enabling back-annotation of time performance and evaluation of response time in the average and worst-case (vs. deadlines)
Proposed framework
1. By model-based tools, algorithms are developed and validated without considering any implementation or hardware detail
2. The Platform Independent Model is parsed and translated into another model according to the internal functional meta-model
3. Designers define the deployed HW and SW resources: ECUs, buses, devices, RTOSs, tasks, drivers and communication protocols
4. Designers map functional components onto the computation and communication resources. Timing back-annotations are added...
5. The Platform Specific Model is completed
6. The system performance (time, energy...) can be statically analyzed to evaluate if the mapping satisfies the constraints.
7. If the performance is not as good as required: the design may be changed at the platform and functional levels
8. The binary files are obtained by compiling together the functional code, the RTOSs, drivers and the generated code.
Proposed framework - Implementation
Proposed framework - Implementation

Functional area: Matlab/Simulink/Embedded Coder
Proposed framework - Implementation

Functional model

Functional code

Functional/Mapping/Architectural areas: set of Eclipse Plug-ins

Architectural and Mapping models

MDE file (PSM)

RTOS and drivers API

PSM

Code Generation

Analyzer

Tasks, configuration and middleware

Compiler

Executable binary files

Performance

©2012 Scuola Superiore Sant’Anna
Proposed framework - Implementation

Code generation: Acceleo scripts (Eclipse Plug-in)
Code generation issues

Code generation issues:

▶ execution of the functional blocks
▶ sustaining the communication among the blocks
Code generation issues:

- execution of the functional blocks
- sustaining the communication among the blocks

Each subsystem behavior is encoded in: `<subsystem name>_step()`
Code generation issues:

- execution of the functional blocks
- sustaining the communication among the blocks

Each subsystem behavior is encoded in: `<subsystem name>_step()`

```
TASK(task_1) {
    ...
    subSystem1_step();
    ...
}
```

```
TASK(task_2) {
    ...
    subSystem2_step();
    ...
}
```
Code generation issues

Communication between the blocks

A middleware level provides the actual implementation of the communications
Code generation issues

Communication between the blocks

A middleware level provides the actual implementation of the communications

\[
\begin{align*}
\text{TASK(} \text{task}_1 \text{)} \{ \\
\quad \ldots \\
\quad \text{subSystem1_step();} \\
\quad \text{write_sub1_Out2();} \\
\quad \ldots \\
\}\ 
\begin{align*}
\text{TASK(} \text{task}_2 \text{)} \{ \\
\quad \ldots \\
\quad \text{read_sub2_In1();} \\
\quad \text{subSystem2_step();} \\
\quad \ldots \\
\\}
\end{align*}
\]
Code generation issues

Communication between blocks: Intra-Task communication

```cpp
TASK(singleTask) {
    subSystem1_step();
    write_sub1_Out2();
    ... read_sub2_In1();
    subSystem2_step();
}
```

```
inline void write_sub1_Out2 () {
    X = subSystem1_Y . Out2 ;
}
inline void read_sub2_In1 () {
    subSystem2_U . In1 = X;
}
```
Communication between blocks: Intra-Task communication

```
TASK(singleTask) {
    subSystem1_step();
    write_sub1_Out2();
    ...
    read_sub2_In1();
    subSystem2_step();
}
```

```c
TypeX X;

inline void write_sub1_Out2 ()
{
    X = subSystem1_Y.Out2;
}
```

```
inline void read_sub2_In1 ()
{
    subSystem2_U.In1 = X;
}
```
Code generation issues

Communication between blocks: Inter-Task communication

\[
\begin{align*}
\text{TASK}(\text{task}_1) \{ \\
\quad \text{subSystem1\_step()}; \\
\quad \text{write\_sub1\_Out2()}; \\
\}
\end{align*}
\]

\[
\begin{align*}
\text{TASK}(\text{task}_2) \{ \\
\quad \text{read\_sub2\_In1()}; \\
\quad \text{subSystem2\_step()}; \\
\}
\end{align*}
\]
Code generation issues

Communication between blocks: Inter-Task communication

```c
TypeX X;
Mutex mutex_X;

inline void write_sub1_Out2 () {
    GetResource(mutex_X);
    X = subSystem1_Y.Out2;
    ReleaseResource(mutex_X);
}

inline void read_sub2_In1 () {
    GetResource(mutex_X);
    subSystem2_U.In1 = X;
    ReleaseResource(mutex_X);
}
```

TASK(task_1) {
    subSystem1_step();
    write_sub1_Out2();
}

TASK(task_2) {
    read_sub2_In1();
    subSystem2_step();
}
Code generation issues

Communication between blocks: Inter-ECU communication

```
TASK(task1) {
    subSystem1_step();
    write_sub1_Out2();
}

inline void write_sub1_Out2 () {
    getResource(mutex_X);
    X = subSystem1_Y.Out2;
    ReleaseResource(mutex_X);
}

TASK(task2) {
    read_sub2_In1();
    subSystem2_step();
}

inline void read_sub2_In1 () {
    getResource(mutex_X);
    subSystem2_U.In1 = X;
    ReleaseResource(mutex_X);
}
```

X

X
Code generation issues

Communication between blocks: Inter-ECU communication

```
TASK(task1) {
    subSystem1_step();
    write_sub1_Out2();
}

inline void write_sub1_Out2 () {
    GetResource(mutex_X);
    X = subSystem1_Y.Out2;
    ReleaseResource(mutex_X);
}

TASK(task2) {
    read_sub2_In1 ();
    subSystem2_step();
}

inline void read_sub2_In1 () {
    GetResource(mutex_X);
    subSystem2_U.In1 = X;
    ReleaseResource(mutex_X);

    Physical channel

    TASK(networkSender) {
        GetResource(mutex_X);
        typeX localX;
        localX = X;
        ReleaseResource(mutex_X);
        send(localX, FRAME_ID_X);
    }

    TASK(networkReceiver) {
        typeX localX;
        receive(localX, FRAME_ID_X);
        X = localX;
        ReleaseResource(mutex_X);
    }
```
Evaluation example

Evaluation case: Ball&Plate system

- **Target**: keep the ball as close as possible to the plate center
- **Sensing**: a touch screen sensor on the top of the plate to track the ball
- **Actuation**: two servomotors to control the pitch and roll angles of the plate
Evaluation example

Functional model (in Simulink): 5 subsystems and 3 custom blocks

subTouch

subController

subServoX

subServoY

Plant

Simulink simulation

Error (m)

Time (s)
Evaluation example

Functional model (in Simulink): 5 subsystems and 3 custom blocks

The C functional functions `subTouch_step()`, `subController_step()`, `subServoX_step()` and `subServoY_step()` are generated.
Design 1: Intra-Task communication
Design 1: Intra-Task communication
Design 1: Intra-Task communication
Design 2: Inter-Task communication

channel 0

subTouch

channel 1

subController

channel 2

subServoY

sensing  ctrl  action

RTOS

M. Bambagini
Design 2: Inter-Task communication
Design 2: Inter-Task communication
Design 3: Inter-ECU communication
Design 3: Inter-ECU communication
Design 3: Inter-ECU communication
Performance measurement

![Graph showing error (m) over time (s) for Simulink simulation]
Performance measurement

![Graph showing Simulink simulation, Intra-task communication, and Inter-task communication over time.](image-url)
Performance measurement

![Graph showing performance measurement over time](image-url)
Future work

Next step: functional model back-annotation with the estimated/measured communication delays
Future work

Next step: functional model back-annotation with the estimated/measured communication delays
Conclusions

A framework/methodology for code generation for real-time distributed systems has been introduced.

It aims at combining MBD and MDA approaches for covering all the steps from the idea devising to the development.

In the experiments, three different designs were considered by changing only the architectural and mapping models, without writing any line of C code.
thank you

m.bambagini@sssup.it