Using UML as a front-end for an efficient Simulink-based multithread code generation targeting MPSoCs

Lisane Brisolara, Marcio Oliveira, Francisco A. Nascimento, Luigi Carro, Flávio R. Wagner

Institute of Informatics
UFRGS - Universidade Federal do Rio Grande do Sul
Porto Alegre - Brazil
Introduction

• In the SoC era, product personalization is mainly performed with **software**

• Several applications in the same product
  • Cell phone - camera, GPS, MP3 player, video, …

• In embedded software, one has to deal with multiple Models of Computation

• Short time-to-market

• Hard constraints: high performance, low power, small memory size

**Embedded software development is getting harder**
Motivation

• **UML** and **Simulink** are considered attractive for ESL design

• However, which is the best language for embedded software specification?
UML vs. Simulink

- **UML**
  - Advantages from OO (reuse, modularity, etc)
  - Better for system specification (functionality, QoS requirements)
  - Requires much user-provided source code

- **Simulink (FB)**
  - Better for expressing time-continuous and time-discrete problems (support different MoCs)
  - Better for specifying dataflow behavior
  - Complete code can be automatically generated from a Simulink model (e.g. using RTW)
  - Lack of OO concepts decrease maintainability
UML vs. Simulink

• **UML**
  • Advantages from OO (reuse, modularity, etc)
  • Better for system specification (functionality, QoS requirements)
  • Requires much user-provided source code

• **Simulink (FB)**
  • Better for expressing time-continuous and time-discrete problems (support different MoCs)
  • Better for specifying dataflow behavior
  • Complete code can be automatically generated from a Simulink model (e.g. using RTW)
  • Lack of O.O. concepts decrease maintainability

How can we take the benefits from both languages?
UML vs. Simulink

• UML
  • Advantages from OO (reuse, modularity, etc)
  • Better for system specification (functionality, QoS requirements)

• Simulink (FB)
  • Better for expressing time-continuous and time-discrete problems (support different MoCs)
  • Better for specifying dataflow behavior
  • Complete code can be automatically generated from a Simulink model (e.g. using RTW)

How to take the benefits from both languages?

Integration of UML and Simulink in a unique flow

• Lack of O.O. concepts decrease maintainability
Our proposal

• Use UML to model Application and Platform

• Generate a Simulink model from a UML model
  • Use UML as front-end for an MPSoC multithread code generation flow
    • Generate complete application SW code targeted to MPSoC architectures
    • Generate complete HW-SW platform
  • Better support for dataflow applications

• Reuse UML models for different code generation approaches
MDA - Model-Driven Architecture

• Specific MDE approach, proposed by OMG, which uses the standards
  - UML (Unified Modeling Language)
  - MOF (Meta Object Facility)
  - XMI (XML Metadata Interchange)
  - QVT (Query/View/Transformations)

• Main concepts
  - PIM (Platform-Independent Model)
  - PSM (Platform-Specific Model)
  - Model transformations (mapping)

• “Model once, build it on any platform”
Proposal: an MDA-based approach

UML diagrams: Sequence, State

PIM

Application model

Platform model

Mapping

PSM

Mapped model

Simulink-based flow

Simulink translation

CAAM

Code generation

UML-based flow

UML tool code generation

Implementation
Proposal: an MDA-based approach

UML diagrams: Sequence, State

Application model
Platform model
Mapping
Mapped model

Problem: UML tools do not generate code from sequence diagrams

Simulink-based flow

translation
Simulink CAAM
Code generation

Implementation

UML-based flow

UML tool code generation
Proposal: an MDA-based approach

UML diagrams: Sequence, State

Dataflow

Simulink-based flow

- Translation
- Simulink CAAM
- Code generation

UML-based flow

- Translation
- FSM model
- UML tool code generation

Implementation
Proposal: an MDA-based approach

UML diagrams: Sequence, State

PIM -> Application model -> Platform model -> Mapping -> Mapped model

PSM

Dataflow:
- Simulink-based flow
  - translation
    - Simulink
    - CAAM
    - Code generation

Control-flow:
- UML-based flow
  - translation
    - FSM model
    - UML tool code generation

Implementation
Outline

• Simulink-based MPSoC design flow
• MDA-based generation of Simulink
• Prototype
• Case study
• Conclusions and future work
Outline

• Simulink-based MPSoC design flow
• MDA-based generation of Simulink
• Prototype
• Case study
• Conclusions and future work
Simulink-based MPSoC design flow

1. Application Architecture Combining
   - Simulink functional model
   - Architecture
   - Simulink CAAM
   - CAAM: Combined Architecture and Application Model

2. HW architecture generator
3. Multithread code generator

Collaboration with TIMA Laboratory, Grenoble, France
Simulink-based MPSoC design flow

1. Combine application and architecture into a single Simulink model

CAAM: Combined Architecture and Application Model

Collaboration with TIMA Laboratory, Grenoble, France
Simulink-based MPSoC design flow

1. Application Architecture Combining

2. Generate SystemC models for HW architecture

3. Multithread code generator

Simulink functional model

Architecture

CAAM: Combined Architecture and Application Model

Collaboration with TIMA Laboratory, Grenoble, France
Simulink-based MPSoC design flow

1. Application Architecture Combining

2. HW architecture generator

3. Multithread code generator

- 3. Generate multithreaded C code for target architecture

Collaboration with TIMA Laboratory, Grenoble, France
Outline

• Simulink-based MPSoC design flow
• MDA-based generation of Simulink
• Prototype
• Case study
• Conclusions and future work
MDA-based generation of Simulink

1: Create UML model containing application and architecture

2: According to mapping rules, translate UML model to a Simulink model, as a model-to-model transformation

3: Optimize the mapped Simulink model:
   - group threads
   - insert temporal barriers

4: Transform the Simulink model into the .mdl file format using model-to-text transformation
Mapping from UML to Simulink

Modeling and mapping are based on **deployment** and **sequence** diagrams

- Partition application into threads
  - The stereotype `<active>` indicates that object is a thread
- Threads are mapped to processors
- Describe a sequence diagram for each thread

Deployment diagram
From UML to Simulink: example

Some UML to Simulink mappings

- Active objects (<<SAschedRes>>) → Threads
- Objects → Subsystem of Threads
- Methods invoked from the same object (m2 & m3 from Obj) → Block
- Parameter directions (in, out) and return → Block’s input/output ports
- Message arguments → datalinks between Ports
- Setter and getter methods invoked between threads (explicit communication) → Communication
  - Method parameters and message arguments are used to define ports of Threads and Communications
From UML to Simulink: example

Some UML to Simulink mappings

- Active objects (<<SAschedRes>>) → Threads
- Objects → Subsystem of Threads
- Methods invoked from the same object (m2 & m3 from Obj) → Block
- Parameter directions (in, out) and return → Block’s input/output ports
- Message arguments → datalinks between Ports
- Setter and getter methods invoked between threads (explicit communication) → Communication
  - Method parameters and message arguments are used to define ports of Threads and Communications

CAAM
Simulink
Thread-SS
Subsystem of Threads
CPU1
CPU
From UML to Simulink: example

Some UML to Simulink mappings

- Active objects (<<SAschedRes>>) → Threads
- Objects → Subsystem of Threads
- Methods invoked from the same object (m2 & m3 from Obj) → Block
- Parameter directions (in, out) and return → Block’s input/output ports
- Message Arguments → datalinks between Ports
- Setter and getter methods invoked between threads (explicit communication) → Communication
  - Method parameters and message arguments are used to define ports of Threads and Communications
From UML to Simulink: example

Some UML to Simulink mappings

• Active objects (<<SAschedRes>>) → Threads
• Objects → Subsystem of Threads
• Methods invoked from the same object (m2 & m3 from Obj) → Block
• Parameter directions (in, out) and return → Block’s input/output ports
• Message Arguments → datalinks between Ports
• Setter and getter methods invoked between threads (explicit communication) → Communication
  • Method parameters and message arguments are used to define ports of Threads and Communications
From UML to Simulink: example

Some UML to Simulink mappings

- Active objects (<<SAschedRes>>) → Threads
- Objects → Subsystem of Threads
- Methods invoked from the same object (m2 & m3 from Obj) → Block
- Parameter directions (in / out / inout) and return → Block’s input/output Ports
- Message Arguments → datalinks between Ports
- Setter and getter methods invoked between threads (explicit communication) → Communication
  - Method parameters and message arguments are used to define ports of Threads and Communications

Simulink CAAM
From UML to Simulink: example

Some UML to Simulink mappings

- **Active objects** (<<SAschedRes>>) → Threads
- **Objects** → Subsystem of Threads
- **Methods invoked from the same object** (m2 & m3 from Obj) → Block
- **Parameter directions** (in, out) and return → Block’s input/output ports
- **Message Arguments** → datalinks between Ports
- **Setter and getter methods invoked** between threads (explicit communication) → Communication
  - Method parameters and message arguments are used to define ports of Threads and Communications
From UML to Simulink: example

Some UML to Simulink mappings

- Active objects (<<SAschedRes>>) → Threads
- Objects → Subsystem of Threads
- Methods invoked from the same object (m2 & m3 from Obj) → Block
- Parameter directions (in, out) and return → Block’s input/output ports
- Message Arguments → datalinks between Ports
- Setter and getter methods invoked between threads (explicit communication) → Communication
  - Method parameters and message arguments are used to define ports of Threads and Communications
Outline

- Simulink-based MPSoC design flow
- MDA-based generation of Simulink
- Prototype
- Case study
- Conclusions and future work
Prototype

1: Create UML model and export XMI to EMF

1. UML editor tool (MagicDraw) → EMF/UML (E-Core) → Transf. engine (ATL) → Mapping rules

2: According to mapping rules, translate UML model to a Simulink model, as a model-to-model transformation

2. Simulink meta-model → Simulink (E-core) → Transf. engine (ATL) → Simulink (E-core) → MDL generator JavaCC parser

3: Optimize the mapped Simulink model:
   - group threads
   - insert temporal barriers

3. optimize

4: Read the E-core Simulink model and generate the .mdl file according to the Simulink grammar (using Javacc)

4. Simulink. mdl
Prototype

1: Create UML model and export XMI to EMF

2: According to mapping rules, translate UML model to a Simulink model, as a model-to-model transformation

3: Optimize the mapped Simulink model:
   - group threads
   - insert temporal barriers

4: Read the E-core Simulink model and generate the .mdl file according to the Simulink grammar (using Javacc)
Transformation engine: using ATL

```java
rule SampleModel {
    from
        uml_model : uml!Model
    to
        simulink_model : simulink!Model (name<-uml_model.name, system<-system),
        top_system : simulink!System (name<-uml_model.name)
}
```
Transformation engine: using ATL

```plaintext
rule SampleModel {
    from
        uml_model : uml!Model
    to
        simulink_model : simulink!Model (name<-uml_model.name, system<-system),
        top_system : simulink!System (name<-uml_model.name)
}

rule SampleProcessor {
    from
        node : uml!Node
    to
        processor : simulink!Processor(name<-node.name)
}

rule SampleThread {
    from
        uml_thread : uml!Lifeline (uml_thread.represents.type.isActive)
    to
        thread : simulink!Thread (name<-uml_thread.name)
}
```
Transformation engine: using ATL

``` ATL
rule SampleModel {
  from
    uml_model : uml!Model
  to
    simulink_model : simulink!Model ( name<-uml_model.name, system<-system ),
    top_system : simulink!System ( name<-uml_model.name, processors<- Sequence { uml!Node.allInstances() } )
}
rule SampleProcessor {
  from
    node : uml!Node
  to
    processor : simulink!Processor( name<-node.name )
    threads<- Sequence { uml!Lifeline.allInstances() ->select ( lifeline | lifeline.represents.type.isActive ) }
}
rule SampleThread {
  from
    uml_thread : uml!Lifeline ( uml_thread.represents.type.isActive )
  to
    thread : simulink!Thread ( name<-uml_thread.name )
}
```
Prototype

1: Create UML model and export XMI to EMF

- **UML editor tool (MagicDraw)**
- **EMF/UML (E-Core)**
- **Mapping rules**

2: According to mapping rules, translate UML model to a Simulink model, as a model-to-model transformation

- **Transf. engine (ATL)**
- **Simulink (E-core)**

3: Optimize the mapped Simulink model:
   - group threads
   - insert temporal barriers

- **Simulink meta-model**

4: Read the E-core Simulink model and generate the .mdl file according to the Simulink grammar (using JavaCC)

- **MDL generator JavaCC parser**
- **optimize**
- **Simulink mdl**
Optimizations

• Automatic insertion of temporal barriers

• Group communicating threads to reduce the communication cost
Optimizations

• Automatic insertion of temporal barriers

• Group communicating threads to reduce the communication cost
From UML to Simulink: temporal barriers

- Identify cyclic paths in the UML model
- Insert temporal barriers in the corresponding Simulink model

Case 1: The output of the function block is connected to its input

Case 2: Cyclic path exists between different sub-systems or hierarchical levels

Case 1

Case 2
Case 1: cyclic path in the same hierarchical level

- The output of function (m1) is connected to its input
From UML to Simulink: temporal barriers (case 2)

- Case 2: Cyclic path exists between different sub-systems or hierarchical levels

(a) Sequence diagram of T1
(b) Sequence diagram of T2

Simulink model
Optimizations

- Automatic insertion of temporal barriers
- Group communicating threads to reduce the communication cost
Grouping threads

- Start from an initial deployment diagram
- Check threads with dependencies
- Group threads with dependencies
  - allocate them to the same processor
  - reduce communication cost
- The algorithm is based on Linear Clustering
  - linear cluster: sequence of dependent nodes
1. Choose the heaviest edge;
2. If nodes n3 or n6 are not taken
   1. Add nodes n3 and/or n6 to cluster C1;
3. Find incoming edges of node n3;
4. Choose the heaviest edge of step 3;
5. If node n1 is not taken
   1. Add node n1 to C1;
6. Find outgoing edges of node n6;
7. Choose the heaviest edge of step 6;
8. If node n8 is not taken
   1. Add node n1 to C1;
9. Repeat steps 1-8 while possible;
10. Store cluster C1 and create a new one;
11. Goto step 1;
12. Stop when every node has a cluster;

Grouping threads
Grouping threads

1. Choose the heaviest edge;
2. If nodes n3 or n6 are not taken
   1. Add nodes n3 and/or n6 to cluster C1;
3. Find incoming edges of node n3;
4. Choose the heaviest edge of step 3;
5. If node n1 is not taken
   1. Add node n1 to C1;
6. Find outgoing edges of node n6;
7. Choose the heaviest edge of step 6;
8. If node n8 is not taken
   1. Add node n1 to C1;
9. Repeat steps 1-8 while possible;
10. Store cluster C1 and create a new one;
11. Goto step 1;
12. Stop when every node has a cluster;
Grouping threads

1. Choose the heaviest edge;
2. If node n3 or n6 aren’t taken
   1. Add nodes n3 and/or n6 to cluster C1;
3. Find incoming edges of node n3;
4. Choose the heaviest edge of step 3;
5. If node n1 is not taken
   1. Add node n1 to C1;
6. Find outgoing edges of node n6;
7. Choose the heaviest edge of step 6;
8. If node n8 is not taken
   1. Add node n1 to C1;
9. Repeat steps 1-8 while possible;
10. Store cluster C1 and create a new one;
11. Goto step 1;
12. Stop when every node has a cluster;
1. Choose the heaviest edge;  
2. If node n3 or n6 aren’t taken  
   1. Add nodes n3 and/or n6 to cluster C1;  
3. Find incoming edges of node n3;  
4. Choose the heaviest edge of step 3;  
5. If node n1 is not taken  
   1. Add node n1 to C1;  
6. Find outgoing edges of node n6;  
7. Choose the heaviest edge of step 6;  
8. If node n8 is not taken  
   1. Add node n1 to C1;  
9. Repeat steps 1-8 while possible;  
10. Store cluster C1 and create a new one;  
11. Goto step 1;  
12. Stop when every node has a cluster;
1. Choose the heaviest edge;
2. If node n3 or n6 aren’t taken
   1. Add nodes n3 and/or n6 to cluster C1;
3. Find incoming edges of node n3;
4. Choose the heaviest edge of step 3;
5. If node n1 is not taken
   1. Add node n1 to C1;
6. Find outgoing edges of node n6;
7. Choose the heaviest edge of step 6;
8. If node n8 is not taken
   1. Add node n1 to C1;
9. Repeat steps 1-8 while possible
10. Store cluster C1 and create a new one;
11. Goto step 1;
12. Stop when every node has a cluster;
Example

- Extract threads and communication between them from the UML **sequence diagrams**
- Run linear clustering algorithm
Example

- Extract threads and communication between them from the UML **sequence diagrams**
- Run linear clustering algorithm
  - Determine the **number of processors**

Each node is a thread
Example

- Extract threads and communication between them from the UML **sequence diagrams**
- Run linear clustering algorithm

- Generate top-level for the Simulink CAAM (CPU and INTER-COMM blocks)

Each node is a thread
Outline

- Simulink-based MPSoC design flow
- MDA-based generation of Simulink
- Prototype
- Case study
- Conclusions and future work
Case study: application model

Wheelchair control system

- Movement control, collision avoidance, navigation,
- Real-time and resource constraints

Movement controller
- Sensing of movement
- Actuating on the motors
Wheelchair – Movement controller

(a) Deployment diagram
Wheelchair – Movement controller

(a) Deployment diagram

(b) Sequence diagram of MovCtrl

(c) Sequence diagram of Nav
Wheelchair – Movement controller

(a) Deployment diagram

(b) Sequence diagram of MovCtrl

(c) Sequence diagram of Nav

(d) Simulink CAAM for CPU1
Wheelchair – Movement controller

Simulink CAAM for CPU1

MovCtrl
Calc Angle
Calc Speed
Nav

Simulink CAAM for CPU2

JoyDrv
Calc Axis
Act

MovCtrl
Calc Angle
Calc Speed

If/action subsystem

Thread-SS
Intra-CPU COMM
CPU-SS
Outline

- Simulink-based MPSoC design flow
- MDA-based generation of Simulink
- Prototype
- Case study
- Conclusions and future work
Conclusions

• Proposal of a mixed UML-Simulink approach
  • Simultaneously exploit the benefits from UML and Simulink
  • Allow reuse of the same UML model for different code generation flows

• UML used as front-end for a multithread code generation flow for an MPSoC
  • Generation of complete application code targeted to MPSoC architectures
  • Generation of the HW-SW platform through SystemC
Future work

• Generation of FSM from sequence diagrams
• Automatic generation of the deployment diagram (partitioning and mapping)
  • Current implementation: Determine the number of processors and map group of threads to processors
    • Based on the communication between threads
  • Future work: Determine also the grouping of tasks into threads
    • To reduce scheduling overhead and intra-CPU communication
• Integrate the UML to Simulink transformation flow into a model-based design space exploration framework
Thanks for your attention

Questions?

See our demo!

Contacts
{lisan, mfsoliveira, flavio}@inf.ufrgs.br
Sequences are used to capture dataflow

- Methods invoked from the same object (like \(m1\)) are mapped to a **subsystem**

- **Parameter directions** (in, out) are used to define the input and output ports for the corresponding Simulink block

- **Parameter names** are used to capture datalinks between two blocks (\(\text{var3}\) is passed from \(m2\) to \(m3\))

- **I/O** is indicated by a method invocation from an **Actor** (ex: \(\text{getInput()}\))

- **Communication** is explicitly indicated in this diagram through method invocation between threads (using \text{get} and \text{set} as prefix)

  - \(\text{setValue}: \) create an output in T1 and a communication block between T1 and T2
Software synthesis

PIM

UML Application Model

Mapping

PSM

UML Platform Model

Transformation

Java code

Java meta-model
MDA-based Software Synthesis

PIM: Platform Independent Model
PM: Platform Model
PSM: Platform Specific Model
Proposal front-end

1. Step i
   - Model

2. HW architecture generator

3. Multithread code generator

- UML model
- Mapping
- Simulink CAAM

- CPU-SS0
- Mem-SS0
- CPU-SSn

- Physical Interconnect
- MPSoC Architecture
- Multithreaded program

- Main code
  - Hds API
  - Comm. Library
  - Thread Library
  - HAL API
  - Hardware Abstraction Layer (HAL)
Simulink-based MPSoC design flow

Combine application and architecture into a single Simulink model

1. **Simulink functional Model**
   - **Appl. Arch. Combining**
     - Simulink CAAM

2. **HW architecture generator**

3. **Multithread code generator**

**Physical Interconnect**
- CPU-SS0
- Mem-S
- ... CPU-SSn
- ... Mem-S
- ... CPU-SSn

**MPSoC Architecture**
- Main code
  - hd8 API
  - Comm. Library
  - Thread Library
  - HAL API

**Multithreaded program**

**CAAM: Combined Architecture and Application Model**
MDA-based software synthesis from UML

UML diagrams: Sequence, State

PIM
- Application model
- Platform model

Mapping

PSM
- Mapped model

Simulink-based flow
- Simulink CAAM
  - Code generation

Translation

UML-based flow
- FSM model
  - UML tool code generation

Implementation

FSM - Finite State Machine
EMF - Eclipse framework

- A modeling framework and code generation facility for building tools and other applications based on a structured data model
- From a model specification described in XMI, EMF provides tools and runtime support to produce a set of Java classes for the model, along with
  - a set of adapter classes that enable viewing and command-based editing of the model, and
  - a basic editor
- the MOF-like core meta model in EMF is called Ecore
ATL - model transformations

- ATL: a set of tools supporting model transformations
  - editor for transformation rules
  - parser for transformation rules
  - transformation engine
- Transformation engine works upon the APIs for manipulating models created by Ecore
  - UML and Simulink APIs according to their Ecore meta-models