Modeling and SW Synthesis for Heterogeneous Embedded Systems in UML/MARTE

Tutorial SD1: High-Level Specifications to Cope With Design Complexity

Hector Posadas, Pablo Peñil, Alejandro Nicolás, Eugenio Villar

University of Cantabria
Spain
Motivation

- Design productivity gap
  - Raising the abstraction level

- Multi-Processing & Heterogeneous platforms

- Increasing SW content
  - SW-centric design methodologies
Usual SW development flow

- Architectural Design
- HW/SW platform
- Architectural mapping
- Ad/Hoc SW development
  - System calls
  - Communication functions
  - I/O functions & drivers
- Verification & Debug
- Costly fixing of wrong design decisions
Usual SW development flow

- Lack of reusability
  - Ad-hoc code
  - Large re-engineering effort
Outline

- Introduction
- State of the Art
- The PHARAON approach
  - Design Flow
  - Modeling Methodology
  - SW Synthesis
- Conclusions
Introduction

- Model-Driven Architecture (MDA)
  - High-abstraction level
  - Mature SW engineering methodology

- UML language
  - Application to embedded systems design
Introduction

Why UML?

- Natural way to capture system architecture
- Standard way
Introduction

- Why UML?
  - Natural way to capture system architecture
  - Standard way

- UML language
  - Semantics lacks
    - What is each component?
    - What kind or interaction each link actually means?
  - Domain-specific profiles
    - UML/MARTE
Introduction

- **MARTE**
  - Standard UML profile for real-time embedded systems
    - Platform-Independent Model (PIM)
    - Platform Description Model (PDM)
    - Platform-Specific Model (PSM)
  - Rich semantics content
  - Single-source approach
State-of-the-Art

Discussion

- System modeling in MARTE
  - Methods based on specific MoCs and/or profiles
    - Requiring additional semantics
    - Non-standard

- SW Synthesis
  - Commercial code generation available
  - Limited support for heterogeneity
  - Limited flexibility for different architectural mappings
  - Limited support for the MARTE semantics
PHARAON Single-Source Design Flow

C/ C++

Scenarios

PIM (App)

Architectural Mappings

HW/SW Platform

M2T Tools

SW Synthesis (SWSyn)

Compilers Linkers

SW stacks

CPUs/DSPs/ASIPs

GP-GPUs

HW Accelerators

Tutorial SD1: High-Level Specifications to Cope With Design Complexity
ASP-DAC 2014, Singapore
PHARAON Modeling Methodology

Main features

- MDD concepts
  - Separation of Concerns
- CBE: Component-Based Engineering approach
- SW centric
- Standard
  - MARTE profile
Data Types for Communication Interfaces

- Primitive Types
  - Bit
  - Word16

- Bit Arrays
  - BitStream
  - Word16

- Data Structures
  - Info4VadT
    - r_h: array11Coeff
    - r_l: array11Coeff
    - scal_acf: Word16
    - rc: array4Lags
    - pitch: Word16

- Arrays
  - Coef
    - Word16
  - Lag
    - Word16
PHARAON Modeling Methodology

- Component model
  - Hierarchical functional encapsulation
  - Ports
    - provided or required

![Diagram showing hierarchical functional encapsulation and ports](image-url)
Component Interfaces

- clientServerSpecification
  - Interface: VADIF
    - detect( in vadInfo: InfoVadT, out vad_flag: Bit) + reset( in reset: Word16)

- clientServerSpecification
  - Interface: CoderIF
    - code( + in: rawSFrameT, + out: txSFrameT) + init_coder()

- clientServerSpecification
  - Interface: AudioControllerIF
    - TXControl() + RXControl()

- clientServerSpecification
  - Interface: DecoderIF
    - decode( + in: rxSFrameT, + out rawSFrameT) + init()
PHARAON Modeling Methodology

- Component model
  - Interfaces
    - sequential, guarded or concurrent, Max. threads available
    - argument sizes (data splitting), Num. of incoming channels

Diagram showing components C2.1, C2.2, C2.3 connected in a network.
Component model

- Channels manage communications
  - BlockingFunctionCall, BlockingFunctionReturn, both or none
  - Timeout
  - Priority

- Buffer Size
  - ResMult

Diagram:

```
  C2
   ├── C2.1
   │    └── ResMult
   └── C2.2
     └── C2.3
```
PHARAON Modeling Methodology

The Platform Description Model

- HW/SW Components using MARTE stereotypes
  - Software Components
    - OS, Hds, Drivers, …
  - Hardware Components
    - Processors, Memories, Buses, Custom HW, I/O
PHARAON Modeling Methodology

- Platform-Specific Model
  - Memory spaces mapped to platform resources
  - Mapping of functional components
    - to memory spaces and/or platform resources
PHARAON Modeling Methodology

- Architectural Design
- Code reuse and/or development
  - platform independent
- HW/SW platform
- Architectural mapping
- SW Synthesis
  - Fast design optimization
SW Synthesis

- System heterogeneity
- Full support for
  - any architectural mapping decided for each component
  - any specific processing resource selected
  - any processing resource type
  - any memory space
  - any OS used by the processing resource
  - any communication infrastructure
SW Synthesis

- Functional synthesis
  - One executable per memory-space
  - Platform-Independent (C/C++) code
    - Highest reusability
    - Non-recommended explicit calls to platform services
      - communication, concurrency, etc.
    - Platform services should derived from the UML/MARTE model
    - POSIX and/or OpenMP as alternatives
    - Static execution flows
      - \(<\text{SwScheduledResource}>\)
SW Synthesis

- Functional synthesis
  - Platform-Specific code
    - Optimized C code for DSPs
    - OpenCL/GL for GPUs
    - OpenMP for SMPs…

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Original</th>
<th>Optimized Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM</td>
<td>908.28 sec</td>
<td>572.92 sec</td>
</tr>
<tr>
<td>ARM-NEON</td>
<td>325.81 sec</td>
<td>255.28 sec</td>
</tr>
<tr>
<td>ARM+DSP blocking call</td>
<td>206.01 sec</td>
<td>193.45 sec</td>
</tr>
<tr>
<td>ARM+DSP non-blocking call</td>
<td>895.98 sec</td>
<td>431.68 sec</td>
</tr>
<tr>
<td>ARM-NEON+DSP non-blocking call</td>
<td>247.93 sec</td>
<td>215.96 sec</td>
</tr>
</tbody>
</table>
SW Synthesis

- Communication synthesis
  - Essential activity in heterogeneous SW synthesis
  - Client-Server paradigm
  - Dependent on the architectural mapping
SW Synthesis

- Communication synthesis
  - Architectural mapping
    - Same memory space
    - Same OS
    - Different processing nodes
  - Benefits / Drawbacks
    - Communication Speed
    - Memory protection
    - Memory/cache use
    - Scheduling
    - Parallelism…
SW Synthesis

- Communication synthesis
  - 3 Layers automatically inserted in service calls
    - Layer 1: communication semantics
      - Allocation independent
    - Layer 2: management for allocation-dependent communications
      - Thread generation, data splitting, synchronization
    - Layer 3: Low-level communications
      - Inter-thread, Inter-process, distributed communication
Communication synthesis

- Layer 1: Independent of architectural mapping
  - Channel properties
    - RPC

- BlockingFunctionCall (T)
- BlockingFunctionReturn (T)
SW Synthesis

Communication synthesis

- Layer 1: Independent of architectural mapping
  - Channel properties
    - Pipeline

BlockingFunctionCall (F)
BlockingFunctionReturn (T)
SW Synthesis

- Communication synthesis
  - Layer 1: Independent of architectural mapping
    - Channel properties
      - Pipeline & Parallel

BlockingFunctionCall (F)
BlockingFunctionReturn (F)
SW Synthesis

- Communication synthesis
  - Layer 1: Independent of architectural mapping
    - Interface properties
      - Data splitting
SW Synthesis

- Communication synthesis
  - Layer1: Independent of architectural mapping
    - Interface properties
    - Data splitting
SW Synthesis

- Implementation alternatives
  - Channel semantics can be implemented in multiple ways
    - Different OS services
      - shared memory
      - message queue
      - socket
      - file...
  - Performance is highly dependent on platform and OS
  - Synthesis enables fast exploration
    - optimal channel implementation for specific platform and code
SW Synthesis

- MultiCore Association APIs
  - Standard APIs for communication and synchronization
  - Closely distributed embedded systems
    - MCAPI - communication
    - MRAPI - synchronization
    - MTAPI - task generation
  - Independence from OS
  - OS-agnostic channel implementation
    - Components treated as MCAPI nodes
    - Ports treated as MCAPI endpoints
SW Synthesis

- Platform Inputs & Outputs
  - Drivers associated to environmental components

Diagram:
- Camera Test-Bench code
- Camera 1 driver
- Camera 2 driver
- PLC Test-Bench code
- PLC driver
- Environment Model
- System Model
- Memory Space
- OS1
- SMP Node
- Communication Infrastructure

January 20, 2014
Conclusions

- **UML/MARTE**
  - Powerful modeling methodology
  - Single-Source approach
  - Platform-Independent Modeling
  - Maximizing reusability
Conclusions

- SW Synthesis
  - Functional modeling
  - Functional synthesis
  - Communication synthesis
  - Platform Inputs & Outputs

- Actually enables platform-independent code
- Reduces in-depth knowledge of platforms
- Support shorter design optimization cycles
  - wider design exploration
  - shorter code generation on heterogeneous platforms
Additional Information

- H. Posadas, P. Peñil, A. Nicolás, E. Villar
  "Automatic synthesis of embedded SW for evaluating physical implementation alternatives from UML/MARTE models supporting memory space separation“
  "EU FP7-288307 PHARAON project: Parallel and heterogeneous architecture for real-time applications“
- H. Posadas, P. Peñil, A. Nicolás, E. Villar
- P. Peñil, H. Posadas, A. Nicolás, E. Villar
  "Automatic synthesis from UML/MARTE models using channel semantics“