D³SE – Dependency-Driven Design Space Exploration

Activity and Artefact-based Optimization of Distributed Embedded Systems

Alexander Diewald

presented by
Simon Barner
The Systems Engineering Challenge
Ever-increasing complexity of software-defined CPS

- Evolving set of **functionalities**
- Multitude of **contradicting requirements**
- Increase in **system variability**
- **Product-line engineering** and reuse
The Systems Engineering Challenge

Complexity of HW/SW platforms

- Increase of software-defined (critical) functions results into massive performance requirements
- Mixed-criticality integration platforms are only a part of the solution & add additional complexity
- Problem: Now system engineers have to cope with both complexity drivers!
Example: Real world Traffic Jam Assistant

System level engineering

- # of elements imposes complexity
  - ~30 Tasks
  - ~200 Signals
  - ~10 ECUs
  - ~5 networks

- Large number of (non-)functional requirements:
  - Separation constraints
  - Temporal constraints (deadlines)
  - ...

- Source: Waters 2019 Industrial Challenge
Can Development Processes beat Complexity?

Requirements

Abstract Platform

Logical System

HW Platform

Code

“Real” System

© www.heimnetzwerke.net – CC-A

© VIA Technologies Inc
Can Development Processes beat Complexity?

Timing

Safety

Logical System

Requirements

Code

Redundancy

Partitions

Abstract Platform

HW Platform

"Real" System

© www.heimnetzerke.net – CC-A

© VIA Technologies Inc
Can Development Processes beat Complexity?

Huge design space where numerous dependent decisions have to be taken and artifacts have to derived consistently.
Can Design-Space Exploration relieve System Engineers?

Needs and suggested approach

- Compensate complexity of system functionality and platforms
  - Speed-up by means of design automation
  - Frontloading of architectural and design decisions
  - What-if analyses
  - Decompose decisions in different layers of the system (SW, HW)

- User guidance
  - Handle dependencies in development processes
  - Meaningful presentation of design-alternatives
  - Take the user into the exploration loop

- Suggested approach
  - Combination of DSE and Model-based Systems Engineering (MbSE)
  - Model-based Design-Space-Exploration (MB-DSE)
Model-based Design-Space Exploration

Overview
Why are integrated models needed?

Code-centric vs. Model-Based Development

- Textual Requirements (.docx, .xlsx, ...)
- Traceability matrix
- Code (e.g., C, C++)
- Config.
- Platform

Requirement Models

- Consistency Checks
- Application Models
- Deployment
- Platform Models

- Glossary, Use-Cases, MSCs, Interfaces
- Data Types
- Contracts, ...

- Components, State-machine, Code, Tables, ...

- ECUs, BUS, I/O, Cores, Memory
Model-based Design Space Exploration

- DSE aims at compensating design complexity
  - Automated exploration of alternatives
  - Use of optimization and/or formal methods

- MbSE boosts DSE with models that have a strong semantics
  - Validation of user input
  - Evaluation of design alternatives / solution candidates
    - Verification of constraints
    - Optimization of design goals
  - Tracking of dependencies between artefacts
  - Automatic synthesis of implementation artifacts for selected alternative(s)
MB-DSE Engineering Process

Requirements

System Model
- Logical Architecture
- HW Architecture
- Partition Architecture
- Task Architecture
- Deployment
- System Schedule

Deployment Synthesis

Resource Allocation (Scheduling, Reconfiguration, ...)

Configuration Model

Deployable Configuration Files
System Model

Modeling Viewpoints and Assumptions
Models and Artifacts

Terminology

“A model is an *appropriate abstraction for a particular purpose*“

[Broy 2011]

An **artifact** is one of many kinds of tangible (by-)products produced during the development of software.

- e.g., use cases, class diagrams, models, requirements and design documents or artifacts concerned with the process of development itself—such as project plans, business cases, and risk assessments.
System Viewpoints and Views

- A viewpoint reflects the specific interests of dedicated stakeholders and conventions, that enable the generation and analysis of a view.

- These conventions could be languages, notations, model-types, design restrictions or modelling methods, analysis techniques as well as further operations, that can be applied to that view.

- A view is an instance of a certain viewpoint in context to a specific system. A view is generated by a set of models that are representing the relevant characteristics.

In relation to: IEEE 42010
Systems Engineering Viewpoints
SPES Matrix

- Differentiation between viewpoints (according to ISO/IEC 42010)
- Differentiation by granularity levels of a system and its decomposition
- Artefact model with a well-defined semantic of artifact types and their relation to other artifacts
- Overall system properties
Component Architecture

- Hierarchical component network
- Components model behavior (e.g. states)
- Data transfer by typed ports via channels
- Components can be distributed on the target platform
- Agnostic of memory model
  - Shared memory
  - Message passing
Platform Architecture

- Hierarchical Platform
- Technical architecture for HW
  - Distributed systems
  - MPSoCs
- Prerequisite to consider non-functional properties
  - Safety
  - Performance
Task & Partition Architectures

- Flat technical architecture models for system SW

- Task architecture
  - Runtime “containers” for (groups of) components
  - Defines port semantics (sampling, buffering)

- Partition architecture
  - Execution containers for tasks
  - Isolation by safety levels
Allocations & Model Element Annotations

- Allocation Tables model element to element mappings, such as
  - component → task
  - task → execution unit
  - task → memory
  - task → partition → phys. execution unit. (while multi-layer mapping)

- Annotations
  - properties attached to model elements,
  - orthogonal and extensible.
AER Execution Model

- Abstraction that enables accurate prediction of temporal behaviour of tasks
  - at design time
  - for shared memory-based systems

- Separates tasks into acquire, execute, and restitution phases
  - Worst-Case Execution Times (WCETs) of tasks show reduced variance
  - Large variance is caused by interferences at data fetching phases *
  - Enables orchestration of data fetching to avoid interferences at design time

D³SE – Dependency-Driven Design Space Exploration

Framework for Activity and Artefact-based Optimization of Distributed Embedded Systems
Reminder: Model-based Design Space Exploration

- **DSE aims at compensating design complexity**
  - Automated **exploration of alternatives**
  - Use of **optimization and/or formal methods**

- **MbSE boosts DSE with models that have a strong semantics**
  - Validation of **user input**
  - Evaluation of design alternatives / solution candidates
    - Verification of **constraints**
    - Optimization of **design goals**
  - **Tracking of dependencies** between artefacts
  - **Automatic synthesis** of implementation artifacts for selected alternative(s)
Decomposition: DSE and Development Processes

1. **Approach**: Decompose development of embedded system into (a set of) **activities and artefacts**

2. **Artefacts/activities structure the development process:**
   - **Horizontally**, by adding (or synthesizing) additional artefacts from existing ones
   - **Vertically**, by adding details to artefacts

3. An **exploration feature** represents a **development activity**

4. An exploration feature consists of **exploration modules** that **operate on artefacts**
Design Goals for the D³SE Framework

Goals: DSE Expert Productivity, Ease-of-Use, and Performance

- **Supporting system engineers by ease-of-use:**
  - Users can tailor DSE executions to their system by enabling/disabling features
  - Artefact-based approach allows a deep integration in tools

- **Increasing productivity of DSE experts:**
  - Interface-driven „orthogonality“ of features eases problem thinking
  - White box approach simplifies debugging

- **Performance of iterative approach:**
  - Optimizing the elements of a loop pays off
  - Avoid infeasible candidates
Dependencies Dominate the DSE

Example: 1oo2 Safety Function → Task Graph → Task Allocation

Initial Task Graph

Task Graph: Unrolled Safety Function

Mapping the final Task Graph

Separation!
Dependency-Driven Optimization Decomposition

- State of the art methods require compositionality:
  - Convex solution space
  - No couplings of subproblems

$t_x$: Timestep $x$
Dependency-Driven Optimization Decomposition

State of the art methods require compositionality:
- Convex solution space
- No couplings of subproblems

$t_x$: Timestep $x$
Dependency-Driven Optimization Decomposition

- State of the art methods require compositionality:
  - Convex solution space
  - No couplings of subproblems
Dependency-Driven Optimization Decomposition

- State of the art methods require compositionality:
  - Convex solution space
  - No couplings of subproblems

- **Decomposition by dependencies**
  - **Master problem** manages
    - dependencies between subproblems and
    - synchronization between iteration loops.
  - **Subproblems**
    - Are solved in order (in parallel where possible)
    - Includes problem-specifics, e.g.
      - **SP1**: Allocate partitions → execution units
      - **SP2**: Allocate tasks → partitions
      - **SP3**: Schedule tasks s.t. SP2

\[ t_x : \text{Timestep } x \]
Dependency-Driven Optimization Decomposition

- State of the art methods require compositionality:
  - Convex solution space
  - No couplings of subproblems

- Decomposition by dependencies
  - **Master problem** manages
    - dependencies between subproblems and
    - synchronization between iteration loops.
  - **Subproblems**
    - Are solved in order (in parallel where possible)
    - Includes problem-specifics, e.g.
      - **SP1**: Allocate partitions $\rightarrow$ execution units
      - **SP2**: Allocate tasks $\rightarrow$ partitions
      - **SP3**: Schedule tasks s.t. SP2

$\text{Master Problem}$

$\text{SP 1}$

$\text{SP 2}$

$\text{SP 3}$

$t_1$

$t_x$: Timestep $x$
Dependency-Driven Optimization Decomposition

- State of the art methods require compositionality:
  - Convex solution space
  - No couplings of subproblems

- Decomposition by dependencies
  - **Master problem** manages
    - dependencies between subproblems and
    - synchronization between iteration loops.
  - **Subproblems**
    - Are solved in order (in parallel where possible)
    - Includes problem-specifics, e.g.
      - **SP1**: Allocate partitions $\rightarrow$ execution units
      - **SP2**: Allocate tasks $\rightarrow$ partitions
      - **SP3**: Schedule tasks s.t. SP2

\[ t_x: \text{Timestep } x \]
Dependency-Driven Optimization Decomposition

- State of the art methods require compositionality:
  - Convex solution space
  - No couplings of subproblems

- Decomposition by dependencies
  - **Master problem** manages
    - dependencies between subproblems and
    - synchronization between iteration loops.
  - **Subproblems**
    - Are solved in order (in parallel where possible)
    - Includes problem-specifics, e.g.
      - **SP1**: Allocate partitions \( \rightarrow \) execution units
      - **SP2**: Allocate tasks \( \rightarrow \) partitions
      - **SP3**: Schedule tasks s.t. SP2

\[ t_1 \quad \text{SP 1} \quad t_2 \quad \text{SP 2} \quad t_3 \quad \text{SP 3} \]
\[ t_x: \text{Timestep } x \]
Dependency-Driven Optimization Decomposition

- State of the art methods require compositionality:
  - Convex solution space
  - No couplings of subproblems

- **Decomposition by dependencies**
  - **Master problem** manages
    - dependencies between subproblems and
    - synchronization between iteration loops.
  - **Subproblems**
    - Are solved in order (in parallel where possible)
    - Includes problem-specifics, e.g.
      - **SP1**: Allocate partitions \(\rightarrow\) execution units
      - **SP2**: Allocate tasks \(\rightarrow\) partitions
      - **SP3**: Schedule tasks s.t. SP2

- \(t_x\): Timestep \(x\)
Dependencies - Exploration Feature Graph

- Represents **dependencies between development activities**
- Allows switching DSE features on or off according to the system-under-design
- Avoids hard dependencies between artefacts → **Reusability, flexibility**
- High-level dependency consideration, low-level problem thinking
Dependencies - Exploration Modules

- **Exploration features** consist of exploration modules

- **Exploration modules** modify / transform artefacts (N:1), e.g.
  - Task graphs (e.g., include task replica)
  - Platform graphs (Exec. Unit Variance)
  - ...

- **DSE Framework implementation**
  - **Dependency Injection** (Guice)
  - I/O artefacts are **annotated** with their corresponding exploration features

- **DSE Execution**
  - Execution order determined by artefact dependencies
  - Identical artefact types ordered by exploration features

A method signature is sufficient to declare dependencies.
DSE Framework for Architectural Exploration
Population: Set of
- Optimization variables,
- Proposed solutions &
- Evaluation results
DSE Framework for Architectural Exploration

Operators: Modify opt. variables
DSE Framework for Architectural Exploration

Decoders:
Create evaluable solutions, e.g., Deployment → Schedule
DSE Framework for Architectural Exploration

Populators → Population → Decode → Candidate solutions → Evaluation

Selection → Population

Evaluation → Validation → Solutions

Evaluators: Assess solutions w.r.t. objectives & constraints
DSE Framework for Architectural Exploration

Feedback loop:
Pick promising solutions
DSE Framework for Architectural Exploration

Create Partition Set  
Create Task Mapping  
Partition Set  
Task Mapping  
Operators  
Partition Decoder  
Task Decoder  
Decoder 3  
Decoder 4  
Decoder 5  
Partition Layout  
Schedule  
Phenotype 3  
Phenotype 4  
Phenotype 5  
Costs  
Timing  
Eval 3  
Eval 4  
Eval 5
DSE Framework for Architectural Exploration

Exploration Modules

- Combine sub-problems with matching decoders, etc.
- Example:
  Variable: Safety architecture;
  Operators: ± number of safety function channels
DSE Framework for Architectural Exploration

Exploration Modules

- Combine sub-problems with matching decoders, etc.
- Example:
  Variable: Safety architecture;
  Operators: ± number of safety function channels

Process-oriented DSE

- Optimize multiple design steps in a loop
- Automatically resolve dependencies, e.g., safety architecture impacts deployment (fault isolation)
Available DSE features (1/2)

- **Safety function architecture exploration**, e.g., 1oo2D
  - Instantiation of isolated safety channels
  - Instantiation of diagnosis units
  - Operates on a task graph

- **Platform exploration**:
  - Explores an optimal number of execution units
  - Adjust the underlying platform graph

- **Partition exploration**:
  - Optimizes the number of partitions
  - Optimizes task-partition & partition-exec. unit allocations
  - Generates communication channels between partitions
Available DSE features (2/2)

- **Design diversity:**
  Instantiates template tasks (task interfaces) with task implementations from a library.

- **Bare-metal task mapping exploration**
  - Optimizes allocations from tasks to execution units
  - Allocation mechanism shared with the partition exploration

- **Heuristic scheduling**
  Generation of simple time-triggered system-wide task and communication schedules
Tool Support

Demonstration in AutoFOCUS3
AutoFOCUS3

Fully model-based platform to research future CPS engineering principles

Open Source Tool and Research Platform based on the Eclipse Platform

- Foundation for **applied research** with automotive and avionics OEMs and suppliers
- High-quality research platform for **efficient prototyping** of novel engineering methods and collaboration within the team

Research Areas

- **Architecture Analysis and Synthesis**: “What are the cheapest and most efficient HW/SW architectures satisfying all constraints?”
- **Re-Use & Variability**: “How to incrementally develop product-lines and reusable components in an agile manner?”
- **(Co-)simulation**: “Do my components behave as intended? In particular, does my system when I integrate everything?”
- **Safety cases**: “How to build structured modular safety argumentation, and how to maintain it on model changes?”

Latest Publications (see here for all 40+)

AutoFOCUS 3 - Modelling
AutoFOCUS 3 – DSE Perspective

- **Modelling of Constraints & Objectives**

- **Synthesizing artefacts by a DSE algorithm**
  - Platform
  - Deployments
  - Schedules

- **Result visualization & model export**
  - Spider chart
  - Schedule view
AutoFOCUS 3 – DSE Perspective

- **Modelling of Constraints & Objectives**
- **Synthesizing artefacts by a DSE algorithm**
  - Platform
  - Deployments
  - Schedules
- **Result visualization & model export**
  - Spider chart
  - Schedule view
AutoFOCUS 3 – DSE Perspective

► Modelling of Constraints & Objectives

► Synthesizing artefacts by a DSE algorithm
  • Platform
  • Deployments
  • Schedules

► Result visualization & model export
  • Spider chart
  • Schedule view
Take Home and Outlook

- **Model-based System Engineering** supports to
  - define and structure the development of complex embedded systems,
  - ease reuse of development artifacts (e.g., to adapt to new platforms), and
  - uncover the underlying design space.

- **A dependency-driven DSE**
  - Enables reuse and extensibility through **modular exploration features**
  - Can be adapted to different systems under design by means of artifact and I/O thinking,
  - Allows to consider design constraints through dependencies, and to offer design alternatives by multi-objective optimization.

- **Implementation in AutoFOCUS3**
  - [https://www.fortiss.org/veroeffentlichungen/software/autofocus-3](https://www.fortiss.org/veroeffentlichungen/software/autofocus-3)
Thank you for your attention!

Contact:
Simon Barner
barner@fortiss.org
References


Contact

Simon Barner
Model-based Systems Engineering
fortiss GmbH

Guerickestr. 25 • 80805 Munich • GERMANY
www.fortiss.org
barner@fortiss.org
©2021

This presentation was created by fortiss. It is intended for presentation purposes only. The transfer of the presentation includes no transfer of ownership or rights of use. A transfer to third parties is not permitted.
AGENDA

Session 1: Fundamental Issues with Concurrency in Embedded Software Systems from Architectural Point of View
9:30

Session 2: Modelling and DSE Methods for Mixed-Critical Software Systems using Multicore Architectures
10:30
10:45

Session 3: Synchronization in Concurrent Software is an Architectural Decision
11:45
12:00
13:00