A Modular Reference Structure for Component-based Architecture Description Languages

ModComp, Ottawa, 28.09.2015
Misha Strittmatter, Kiana Rostami, Robert Heinrich, Ralf Reussner
Palladio Component Model (PCM)

- DSL for component-based software architectures
- Initial focus design time performance prediction

Introduction

Dependencies

Modularization

Application

Related Work

Conclusion
Palladio Component Model (PCM)

- DSL for component-based software architectures
- Initial focus design time performance prediction
- Design did not consider extensibility
- Research spectrum broadened
  - Initial extensions: intrusively
  - Later: external extensions

Dependencies

Performance

Introduction Dependencies Modularization Application Related Work Conclusion

Misha Strittmatter, Kiana Rostami, Robert Heinrich, Ralf Reussner – A Modular Reference Structure for Component-based ADLs
Motivation

- Challenge: extension & evolution
Motivation

- Challenge: extension & evolution
- Inconsistent structuring

Introduction

Dependencies

Modularization

Application

Related Work

Conclusion
Motivation

- Challenge: extension & evolution
- Inconsistent structuring
- Improper separation of concerns

Interface Relations

Resource Interfaces: Relations & Roles

Events Infrastructure

Variable Characterization
Parameter Class is in Repo

(Only examples on class level)
Motivation

- Challenge: extension & evolution
- Inconsistent structuring
- Improper separation of concerns
- Dependency cycles ⇒ all or nothing reuse

Introduction ➤ Dependencies ➤ Modularization ➤ Application ➤ Related Work ➤ Conclusion

**Motivation**

- Challenge: extension & evolution
- Inconsistent structuring
- Improper separation of concerns
- Dependency cycles ⇒ all or nothing reuse

Introduction ➤ Dependencies ➤ Modularization ➤ Application ➤ Related Work ➤ Conclusion
Motivation

- Challenge: extension & evolution
- Inconsistent structuring
- Improper separation of concerns
- Dependency cycles ⇒ all or nothing reuse
- Degradation of understandability and maintainability
**Problem**
- Metamodels not designed for extensibility or modularity, are hard to evolve and maintain
- Intrusive modification and extension degrades structure over time
- Variants/branches lead to duplication

**Idea**
- An approach, which categorizes and divides information into modules and layers
- Guidance and mechanisms for metamodel composition and extension
- Constrainment of dependencies

**Benefit**
- Reduced complexity  
  ⇒ Improved understandability & maintainability
- Flexibility (pick which modules you really need)  
  ⇒ Extensibility, reuse
Dependencies

At least one element in Ext does somehow depend on an element in Base

(EMOF based or similar)
Dependencies

Reference

- Dependencies
- Modularization
- Application
- Related Work
- Conclusion

Introduction

27.09.2015  Misha Strittmatter, Kiana Rostami, Robert Heinrich, Ralf Reussner – A Modular Reference Structure for Component-based ADLs

Software Design and Quality Group
Institute for Program Structures and Data Organization
Dependencies

Reference

- Should be a “thing” for itself
- If not: see later

Introduction ➤ Dependencies ➤ Modularization ➤ Application ➤ Related Work ➤ Conclusion
Dependencies

- Reference ➔
- Contains ➔
Dependencies

- Reference
- Contains
- Inheritance

Extend metamodel by new subclass

Diagram:

- Base Metamodel
  - Repo
  - Basic Comp
  - Repo Comp
  - BlackBox Comp
  - PCM
  - Extension Metamodel

Related Work

Introduction

Modularization

Application

Conclusion
Dependencies

- Reference →
- Contains ↔
- Inheritance →
- ???

Extend existing class by new information:
- Attribute
- Reference
- Containment
Dependencies

- Reference
- Contains
- Inheritance
- ???

- Not a plain relation as the others
- Only needed when working with multiple metamodels
Dependencies

- Reference
- Contains
- Inheritance
- Extends

Diagram:

- PCM
- Base Metamodel
- Repo
- Comp

- Sec
- Extension Metamodell
- Comp
- Security
- +sec

Introduction | Dependencies | Modularization | Application | Related Work | Conclusion

17 27.09.2015 Misha Strittmatter, Kiana Rostami, Robert Heinrich, Ralf Reussner – A Modular Reference Structure for Component-based ADLs
Dependencies

- Reference
- Contains
- Inheritance
- Extends

= Metamodel extension types

Beware: there are ways to use inheritance to model the extends relation. One of them is really bad.
Dependencies

- Reference
- Contains
- Inheritance
- Extends

All these relations are allowed between metamodels if used correctly.

= Metamodel extension types

Beware: there are ways to use inheritance to model the extends relation. One of them is really bad.
Extends: Dependency Inversion

Introduction

Dependencies

Modularization

Application

Related Work

Conclusion
Extends: Dependency Inversion

Introduction

Dependencies

Modularization

Application

Related Work

Conclusion
Modularization

- (Metamodel) modules
  - Set of classes
  - Explicit dependencies
  - No dependency cycles

Diagram:

```
  Module
    /   \
   /     \
  Module  Module
```

Topics:
- Introduction
- Dependencies
- Modularization
- Application
- Related Work
- Conclusion
Modularization

- (Metamodel) modules
  - Set of classes
  - Explicit dependencies
  - No dependency cycles

Modularization

- Module contains set of concerns
- Must be meaningful to use module with or without extension
Modularization

- (Metamodel) modules
  - Set of classes
  - Explicit dependencies
  - No dependency cycles

- Modularization
  - Module contains set of concerns
  - Must be meaningful to use module with or without extension
  - If not: module has to be base for multiple Extensions
Layering

Modules form cycle free, directed graph
Layering

- Modules form cycle free, directed graph
- Modules are organized in layers
- Specific types of dependencies between layers
Layers

- Π: Paradigm
- Δ: Domain
- Ω: Quality
- Σ: Analyses
Layers

- \( \Pi \): Paradigm
  - Abstract structure
  - E.g. composition, object orientation, behavioral formalism

- \( \Delta \): Domain

- \( \Omega \): Quality

- \( \Sigma \): Analyses
Layers

- **Π: Paradigm**
  - Abstract structure
  - E.g. composition, object orientation, behavioral formalism

- **Δ: Domain**
  - Assign domain semantics to paradigm
  - E.g. software, embedded systems, cyber-physical infrastructures

- **Ω: Quality**

- **Σ: Analyses**
Layers

- **Π**: Paradigm
  - Abstract structure
  - E.g. composition, object orientation, behavioral formalism

- **Δ**: Domain
  - Assign domain semantics to paradigm
  - E.g. software, embedded systems, cyber-physical infrastructures

- **Ω**: Quality
  - Inherent quality properties assigned to domain entities

- **Σ**: Analyses
Layers

Π: Paradigm
- Abstract structure
- E.g. composition, object orientation, behavioral formalism

Δ: Domain
- Assign domain semantics to paradigm
- E.g. software, embedded systems, cyber-physical infrastructures

Ω: Quality
- Inherent quality properties assigned to domain entities
- Static during execution
- Mainly second class entities
- E.g. performance, security, resilience

Σ: Analyses
Layers

- **Π**: Paradigm
  - Abstract structure
  - E.g. composition, object orientation, behavioral formalism
- **Δ**: Domain
  - Assign domain semantics to paradigm
  - E.g. software, embedded systems, cyber-physical infrastructures
- **Ω**: Quality
  - Inherent quality properties assigned to domain entities
  - Static during execution
  - Mainly second class entities
  - E.g. performance, security, resilience
- **Σ**: Analyses
  - In-/output, state
  - Analysis configuration
Application: Starting Point

(Foreward elaboration of the PCM’s concerns in [Strittmatter14])
Application: Modularization

π

Core Entities

Composition

Introduction
Dependencies
Modularization
Application
Related Work
Conclusion
Application: Modularization

π

Core Entities

Composition

Software Components

Static Dependencies

Staff Specification

Development Artifacts

Modifications

Behavior

Introduction Dependencies Modularization Application Related Work Conclusion

Misha Strittmatter, Kiana Rostami, Robert Heinrich, Ralf Reussner – A Modular Reference Structure for Component-based ADLs

Software Design and Quality Group
Institute for Program Structures and Data Organization
Application: Modularization

Introduction  Dependencies  Modularization  Application  Related Work  Conclusion

Misha Strittmatter, Kiana Rostami, Robert Heinrich, Ralf Reussner – A Modular Reference Structure for Component-based ADLs
Application: Modularization

Core Entities

Composition

Software Components

Static Dependencies

Development Artifacts

Modifications

Behavior

Failure Types

Performance Metrics

Performance

Reliability

KAMP Input

KAMP Result

Performance Results

Performance Configuration

Introduction Dependencies Modularization Application Related Work Conclusion

Misha Strittmatter, Kiana Rostami, Robert Heinrich, Ralf Reussner – A Modular Reference Structure for Component-based ADLs
Application: Modularization

Institute for Program Structures and Data Organization

38 27.09.2015 Misha Strittmatter, Kiana Rostami, Robert Heinrich, Ralf Reussner – A Modular Reference Structure for Component-based ADLs

Software Design and Quality Group
Institute for Program Structures and Data Organization
Application: Benefit

\[ \pi \]

Core Entities

Composition

Software Components

Behavior

Failure Types

Static Dependencies

Development Artifacts

Performance

Reliability

Staff Specification

Modifications

KAMP Input

KAMP Result

Performance Results

Performance Configuration

Introduction Dependencies Modularization Application Related Work Conclusion

Misha Strittmatter, Kiana Rostami, Robert Heinrich, Ralf Reussner – A Modular Reference Structure for Component-based ADLs
Application: Benefit

\[\pi\]

\[\Delta\]

\[\Omega\]

Core Entities

Composition

Software Components

Development Artifacts

Modifications

Behavior

Failure Types

Static Dependencies

Staff Specification

Performance Metrics

KAMP Input

KAMP Result

Performance Results

Performance Configuration

Introduction

Dependencies

Modularization

Application

Related Work

Conclusion
Application: Benefit

\[ \Pi \]

- Core Entities
- Composition

\[ \Delta \]

- Software Components
- Static Dependencies
- Staff Specification
- Development Artifacts
- Modifications

\[ \Omega \]

- Performance Metrics
- Performance Results
- Performance Configuration
- Reliability
- KAMP Input
- KAMP Result

Introduction \quad Dependencies \quad Modularization \quad Application \quad Related Work \quad Conclusion

Misha Strittmatter, Kiana Rostami, Robert Heinrich, Ralf Reussner – A Modular Reference Structure for Component-based ADLs
Application: Benefit

π

Core Entities

Composition

Software Components

Static Dependencies

Development Artifacts

Staff Specification

Modifications

Behavior

Failure Types

Ω

Σ

KAMP Input

KAMP Result

Performance Results

Performance Configuration

Introduction Dependencies Modularization Application Related Work Conclusion
Application: Benefit

Core Entities

Composition

Software Components

Static Dependencies

Staff Specification

Development Artifacts

Modifications

Behavior

Performance Metrics

Performance

Reliability

Confidentiality

KAMP Input

KAMP Result

Performance Results

Performance Configuration

Introduction Dependencies Modularization Application Related Work Conclusion

Misha Strittmatter, Kiana Rostami, Robert Heinrich, Ralf Reussner – A Modular Reference Structure for Component-based ADLs
Application: Benefit

- Core Entities
- Composition
- Development Artifacts
- Failure Types
- Behavior
- Performance
- Reliability
- KAMP Input
- KAMP Result
- Performance Results
- Performance Configuration

Introduction ➞ Dependencies ➞ Modularization ➞ Application ➞ Related Work ➞ Conclusion
Related Work

Structuring:
- Orthographic Software Modeling [Atkinson10]
- Deep modeling [Atkinson12]
- UML archetypes [Coad99]
- Software “blood types” [Siedersleben04]
Related Work

Structured:
- Orthographic Software Modeling [Atkinson10]
- Deep modeling [Atkinson12]
- UML archetypes [Coad99]
- Software “blood types” [Siedersleben04]

Extensibility:
- JetBrains MPS [Voelter12]
- Arch Studio [Dashofy05]
Related Work

- Structuring:
  - Orthographic Software Modeling [Atkinson10]
  - Deep modeling [Atkinson12]
  - UML archetypes [Coad99]
  - Software “blood types” [Siedersleben04]

- Extensibility:
  - JetBrains MPS [Voelter12]
  - Arch Studio [Dashofy05]

- Modularity:
  - Generators [Jung15]
  - Transformations [Rentschler14]
Future Work

- Guidance/rules to assign information to layers
- Specifics of layers concerning dependency types and module types
- Survey of extension mechanisms
Future Work

- Guidance/rules to assign information to layers
- Specifics of layers concerning dependency types and module types
- Survey of extension mechanisms
- Further application (whole PCM, …)
- Tool support
  - modularization
  - structure visualization
  - modular graphical editors
Conclusion

Idea
- Design for extensibility and modularity
- Guidance for structuring in modules
- Classification of information into layers
Conclusion

Idea

- Design for extensibility and modularity
- Guidance for structuring in modules
- Classification of information into layers
- Explicit dependencies between modules
- Avoid dependency cycles between modules

53  27.09.2015  Misha Strittmatter, Kiana Rostami, Robert Heinrich, Ralf Reussner – A Modular Reference Structure for Component-based ADLs
Conclusion

Idea

- Design for extensibility and modularity
- Guidance for structuring in modules
- Classification of information into layers
- Explicit dependencies between modules
- Avoid dependency cycles between modules

Benefit

- Reduced complexity
  - Understandability & maintainability
- Flexibility (pick which modules you really need)
  - Extensibility, reuse
Summary

\[ f(\Pi, \Delta, \Omega, \Sigma) = \text{Metamodel} \]

Paradigm → Domain → Quality → Analysis → Module Selection

Introduction Dependencies Modularization Application Related Work Conclusion

Misha Strittmatter, Kiana Rostami, Robert Heinrich, Ralf Reussner – A Modular Reference Structure for Component-based ADLs

Software Design and Quality Group Institute for Program Structures and Data Organization
Application: Paradigm

CoreEntities

Interface

Component

Composition

Composed Structure

Connector

Assembly Context

\[ \pi \]

\[ \downarrow 2 \]
Application: Domain

\[\pi\]

- Component Repository
- Operation Interface
- Software Component
- Business Component
- Infrastructure Component
- Modification
- Mod Interface
- ModComp
- Propagation
- IntraComp
- Internal Action
- External Call
- Start Action
Application: Quality

Failure types:
- Internal Action
- Failure Type
- Software Failure
- Hardware Failure

Performance:
- Resource Demand

Reliability:
- Failure Occurrence
- Failure Probability

Failure Types diagnoses:
- Δ Internal Action
- Ω Resource Demand

Reliability factors:
- Failure Occurrence
- Failure Probability
Application: Analysis

- Staff Specification
- Development Artifact
- KAMP Input
- Evolution Scenario
- Modification
- Propagation
- Assembly Context
- Operation Interface
- KAMP Result
- Impact Propagation
- Performance Results
- Operation Response Time
- Performance Results

Impact Propagation Development Artifact

KAMP Input KAMP Result Performance Results

Evolution Scenario Impact Propagation

Unit
## Dependencies within the Layering

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Pi$</td>
<td>$\Delta$</td>
<td>$X$</td>
<td>$X$</td>
<td>$Y$</td>
</tr>
<tr>
<td>$\uparrow$</td>
<td>$\uparrow$</td>
<td>$\uparrow$</td>
<td>$\uparrow$</td>
<td></td>
</tr>
</tbody>
</table>

Dependence structure within the layering.
### Dependencies within the Layering

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Π</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Δ</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Ω</td>
<td></td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

- **X**: Indicates a dependency relationship between layers.
- **Y**: Indicates a dependency relationship between layers.
- **Δ**: Represents a change or transition between layers.
- **Π**: Represents a broad category or layer.
## Dependencies within the Layering

<table>
<thead>
<tr>
<th></th>
<th>Π</th>
<th>Δ</th>
<th>Ω</th>
<th>Σ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Π</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Δ</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ω</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Σ</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- X
- Y

- Arrows indicate dependencies between layers.
Scope

- Design-oriented
- (Quality-describing)
- (Analysis-oriented)
- (Component-based)
- Architecture vs. Software
- Description Language
Dependencies

- Reference $\rightarrow$ “… knows …”
- Contains $\leftrightarrow$ “… contains …”
- Inheritance $\rightarrow$ “… is a …”
- Extends $\rightarrow$ “… is a property of …”
  “… is a part of …”
  “… is a trait of …”
Extension Mechanisms

- Implementation of extends relation
  - Simple reference
  - Simple inheritance
  - Abstract reference
  - Decorator
  - Stereotyping [Kramer12]
  - Aspect(-oriented) extension [Jung14]
  - Roles
  - Completions [Kapova13]
  - MIRs?
  - Model weaving
A Modular Reference Structure for Component-based ADLs

Legend:
- **Concern**
- **Dependency**

- Entity
- Variable Characterization
- Composition
- Allocation
- Environment
- Resource Types
- Performance
- Resource Environment
- RDSEFF
- SEFF
- Repository
- DataTypes
- Component Type Hierarchy
- Usage Model
Misha Strittmatter, Kiana Rostami, Robert Heinrich, Ralf Reussner – A Modular Reference Structure for Component-based ADLs