Unifying Software Reuse

Jörg Kienzle
Software Composition and Reuse Laboratory (SCORE)
School of Computer Science
McGill University
Montreal, Canada
Email: Joerg.Kienzle@mcgill.ca

How Did We Get Here?

- 2009: 1st Workshop on Aspect-Oriented Modelling
  - Applying AOM Approaches to the Crisis Management Case Study (JK, GM, JJ)
  - CMS case study, TAOSD Special Issue with papers from participants
- 2010: 2nd Workshop on Aspect-Oriented Modelling
  - Worked on Integrating AOM Approaches around the CMS (JK, GM, SM)
  - Comparing AOM Approaches Paper (ECMFA 2011)
- 2011: 3rd Workshop on Aspect-Oriented Modelling
  - Worked on elaborating Comparison Criteria for AOM approaches (JK, GM, JJ)
  - bCMS case study, Comparing Modelling Approaches workshops, Comparison Report of AOM Approaches
- 2012: 4th Workshop on Aspect-Oriented Modelling
  - Worked on refining of Comparison Criteria (JK, GM, BC, MS)
- 2015: Workshop on Concern-Oriented Reuse
  - Worked on comparing Units of Reuse, Interfaces
  - VCU paper at ICSR
- 2017: Workshop on Language Reuse
  - Worked on applying Concern-Oriented Reuse at the language level (COLD)
  - Started expanding CORE meta model to include languages, Paper submitted to <Programming>, but rejected

Software Reuse Approaches

- Code-level Reuse
  - Applications are built by writing code that makes use of the API provided by (Class) Libraries / Frameworks
  - Unit of Reuse: Classes / Frameworks
- Component-based Software Engineering (CBSE) / Service-Oriented Architecture (SOA)
  - Applications are built by putting together software components or composing services
  - Unit of Reuse: Components / Services
- Software Product Lines (SPL)
  - Multiple applications (from a common domain) are built by sharing common software artefacts
  - Unit of Reuse: Feature + related base models
- Model-Driven Engineering (MDE)
  - Applications are built by building models describing the software to be built from different points of view / levels of abstraction.
  - Unit of Reuse: Model Transformation / Compiler
- Domain-Specific (Modelling) Languages (DSL and DSML)
  - Applications are built by building models describing the software to be built using the most appropriate language(s) to express the problem and solution domains
  - Unit of Reuse: Language (and dedicated tools) + Model Transformation / Compiler

Reuse Roles

Designer of Unit

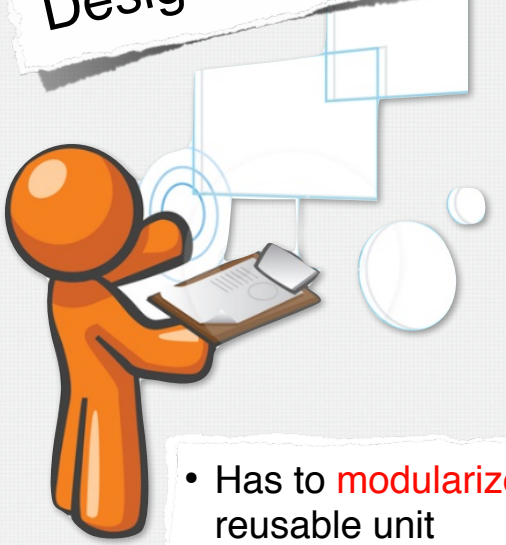
- Domain expert of what is to be made reusable
- Knows specific details of the encapsulated functionality / properties / solutions / qualities
- Does not know in what contexts and how exactly the reusable unit will be used

User of Unit

- Application Expert
- Knows the requirements of what is under construction
- Usually knows very little about the complexity of the reusable unit and it's inner workings

Reuse Needs

Designer of Unit



- Has to modularize / package his reusable unit
- Wants to maximize versatility of the reusable unit (offer different solutions, possibility for customization), without compromising integrity

Interface

User of Unit

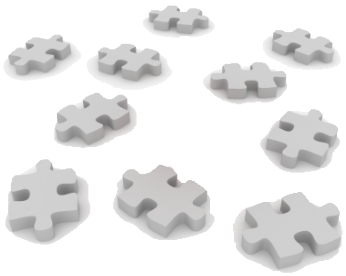
- Has to be able to determine whether the reusable unit is applicable
- Has to be able to determine which variant is best
- Has to be able to customize the reusable unit to his specific application context
- Has to be able to use the reusable unit correctly
- Wants to be unaware of the reusable units inner workings / complexities

Considerable Development Effort

Low Reuse Effort

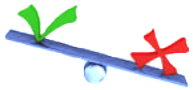
Modularization Approaches

- Within a single (modelling) language
  - Hierarchical Decomposition
    - Composition specification / operator: containment, inheritance
  - Functional Decomposition
    - Composition specification / operator: calling, invoking / binding
  - Component-Oriented Decomposition
    - Composition specification / operator: connectors / middleware
  - Aspect-Oriented (Modelling) / Feature-Oriented Decomposition / SoC
    - Composition specification / operator: pointcuts, patterns / matching, weaving
  - Multilevel Modelling
    - Composition specification / operator: potency-based instantiation / model transformation
- Across (modelling) languages
  - Hierarchical Decomposition
  - Functional Decomposition
    - Composition specification / operator: service orchestration specification / middleware
  - Language-Oriented / View-Oriented Decomposition / Multiparadigm or Heterogenous Modelling
    - Composition operator: structural & behavioural mappings, consistency rules / model transformations, coordinated execution



VCU - Interfaces for Reuse

- 2015 Workshop on Reuse at Bellairs
- Survey of existing units of reuse
  - Classes, Components, Frameworks, Software Product Lines, Services
- Identification of a Canonical Set of Interfaces for Reuse



- Variation Interface

- Customization Interface

- Usage Interface

[1] Jörg Kienzle, Gunter Mussbacher, Omar Alam, Matthias Schöttle, Nicolas Belloir, Philippe Collet, Benoit Combemale, Julien DeAntoni, Jacques Klein, and Bernhard Rumpe: "VCU: The Three Dimensions of Reuse", International Conference on Software Reuse, ICSR 2016, Limassol, Cyprus, June 5-7, 2016, no. 9679 in LNCS, pp. 122–137, Springer, June 2016.

Current MTheory Metamodel

Workshop Objectives

- To be discussed :)
- Understanding relationships between reuse approaches and how to exploit modularization techniques for reuse
- Document the understanding within a “M-Theory” Metamodel

Concrete Outcome

- To be discussed, to evolve :)
- Setup a new vision
- Setup a practical demonstration of the M-Theory

Potential Workgroups

- Terminology of each domain (MDE, DSL, CBSE, AO, Multi Paradigm, Multi Level, …)
  - Ontology
  - Metamodel
- Model Transformations: How to integrate them within the M-Theory?
- Language for creating perspective actions from language actions
- Customization Interface: What should be customizable? Is it possible to specify the CI in a language-independent way?
  - Mapping cardinalities / model types / posteriori typing
- Multilevel modelling: promotion from model to language
- Negative Variability: How to handle it within the M-Theory