

'Lightweight MDA' using UML Profiles and XMI

Edward Garson

Senior Consultant

Dunstan Thomas Limited

egarson@dthomas.co.uk



Agenda

- Presenting 'Lightweight' MDA
 - Compared & contrasted to bona fide MDA
 - An *Artefact* Generation technique
- Overview of technical implementation
- Two fast & contrasting case studies
- Where this works well & where it doesn't 😊
- Conclusion

Technical Background

- Developed during several engagements
- Resulted from early adoption attempts
- A proponent of Lightweight MDA
 - An **MDA atheist** for reasons which follow
- Our spin on OMG MDA
 - I do not purport to be an MDA expert
 - Must it be seen to be believed...?

The Problem with Bona Fide MDA

- Many implementations **too complex**
 - Bucks the trend, the Renaissance of Simplicity
 - “Show me ‘Hello World’ in MDA!?” [Fowler]
 - Will fail for human, not technological reasons
 - Developers still wrestle with UML!
 - Psychology of development incongruous w/MDA
- Partially detracts from the UML value proposition
 - Too close to code!

The Problem with Bona Fide MDA (cont'd)

- Unclear how MDA fits into a SDP
 - Process agnostic: A Good Thing?
- Conflicting ideas of what MDA really is
 - Different things to different people
- 3 years on we have few popular tools
 - Many claiming to be MDA are really MDD

Our answer: 'Lightweight' MDA

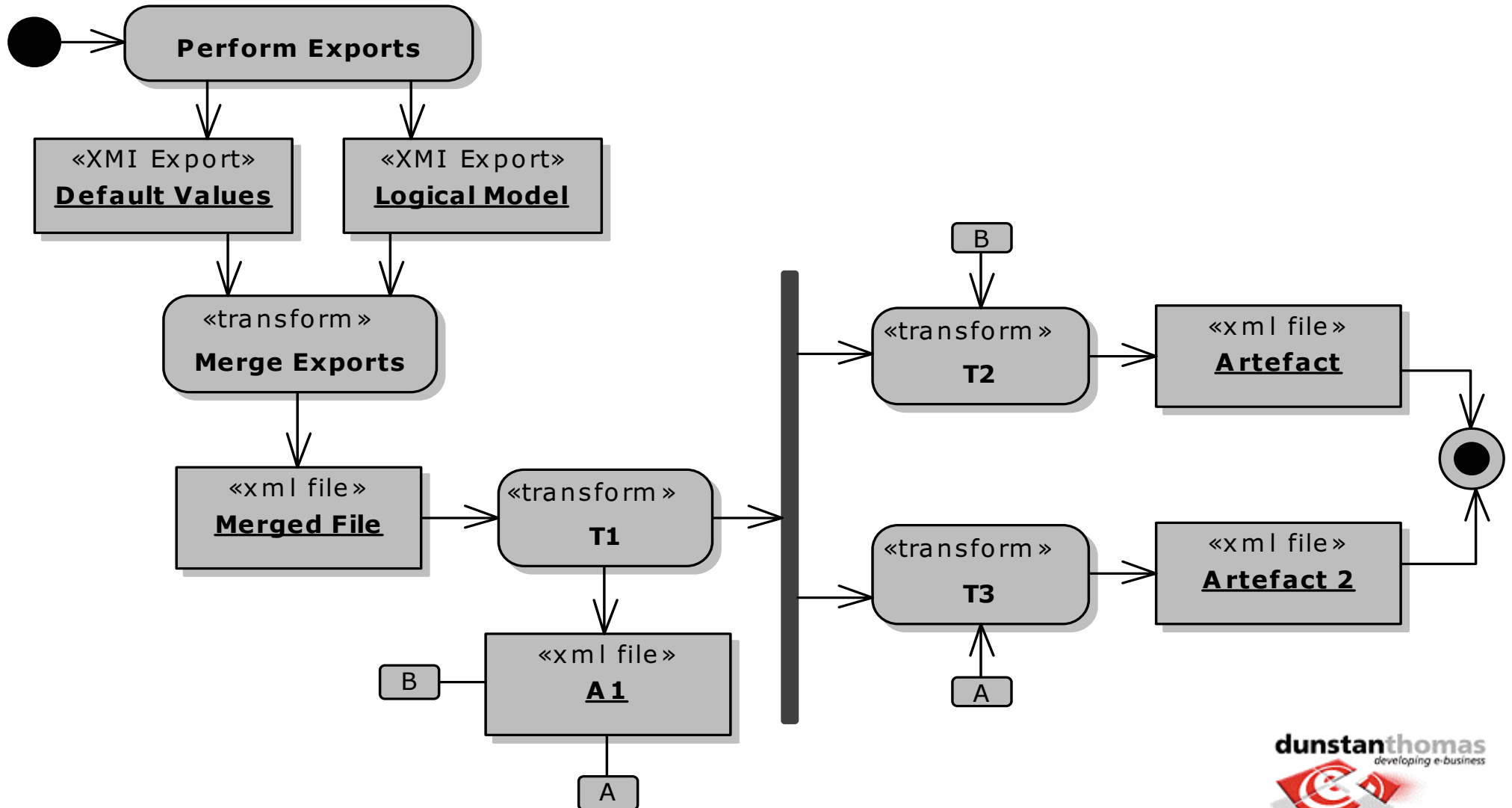
How does this contrast with bona fide MDA?

- No PSMs (no model to model translation)
 - Platform-independent assets manifested as XML
 - Changes only effected upstream
 - Complementary artefacts cascade & branch
- Not looking for 100% code generation
 - Primarily interested in 'quick wins' & ROI
- Really about small **visual metamodels**

'Lightweight' MDA – Key Components

- Platform independent logical model (CD's)
- Custom UML Profiles (domain vocabulary)
 - Stereotypes & Tagged Values
 - Default Values mechanism & callouts
- XML Metadata Interchange (XMI)
- XSL Transformations (XSLT)
- Supporting tools (ANT/NANT, scripts...)

Simplified Process View



Benefits of 'Lightweight' MDA

- Model investment protected over time
 - Ability to change technologies (e.g. COM to .NET)
- RAD realization of architectural mechanisms
 - Sweeping changes possible
- Low defect incidence in generated artefacts
- Yes, platform independence, of a kind...
 - ...and very flexible interpretation of "Platform"

Benefits of 'Lightweight' MDA (cont'd)

- Simple Interface == Wider Audience
 - Technical analysts drive metamodels
 - Collaborative effort between MDA architects and technical analysts
- Auto-generated artefacts easy to produce
 - Virtue of Laziness: less thinking required
 - Never mind all that fancy generic code
 - Moot point nowadays: less polymorphism = better performance

Benefits of 'Lightweight' MDA (cont'd)

- Hand-rolled == ultimate flexibility
- Goes beyond the code
 - Unorthodox implementations possible (indeed, encouraged as we will see!)
- No vendor lock in
 - Only real commitment is to XML and UML
- In retrospect, many benefits of real MDA
 - Furthermore, no fancy tools required

Process Implementation

- Find and ratify architectural mechanisms
 - Proof of concept w/manual implementation
- Discover variances
- Encapsulate variances in UML Profile
 - Tagged values control choices
- Create testable, reduced reference model
- Design & write layered stylesheets
 - Ensure stepwise processing
- Generate & test

Case Study – Business Objectives

- Balance customization and maintenance
 - Achieve rapid time to market
 - Better manage large-scale deployments
 - Huge DB schemas
 - Ability deploy to multiple databases
- Keep framework decoupled
- Reduce DB maintenance headaches

What was Required

- Smart, flexible database schemas
- Ability for persistent entities to keep up
 - Enable codebase to evolve separately
- Core codebase + customization / modularity
- Ability for analysts to adorn + generate
- Development of custom UML Profile
 - Guides analysts for model instantiation

Case Study – Example Aspect

- Object/Relational mapping in light of BO's
- Example implementation unimportant!
 - In fact this example is dry and academic
- Shows a domain mapping to UML semantics
 - Leverages best practices (Ambler, Fowler...)
- Demonstrates how variances are encapsulated
- Communicates the possibilities
 - Not simple code generation!

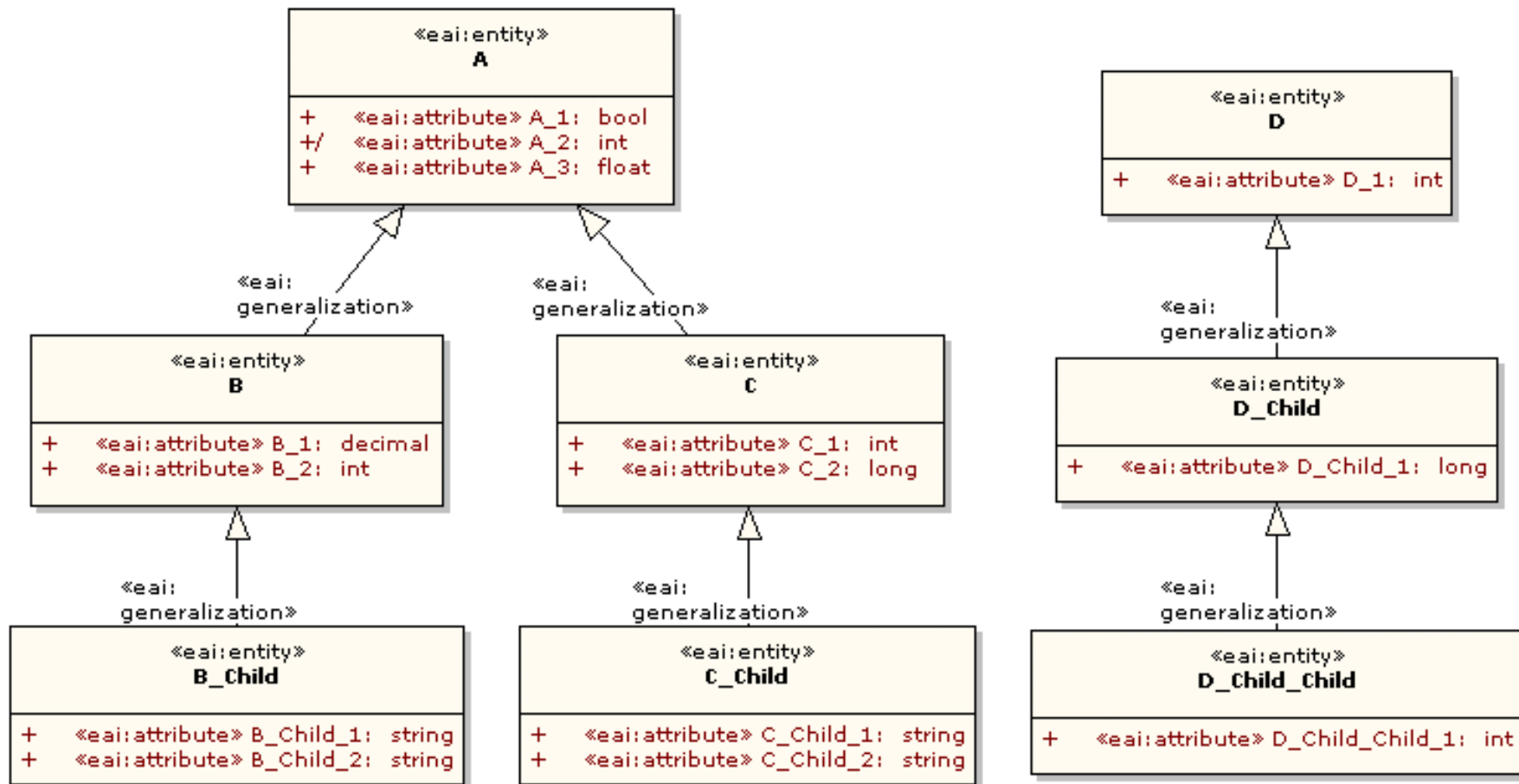
Simple Composite Aggregation



- Create a FK from B to A
- Use of standard UML constructs
- No profile necessary

Cascade Strategy – Less Joins

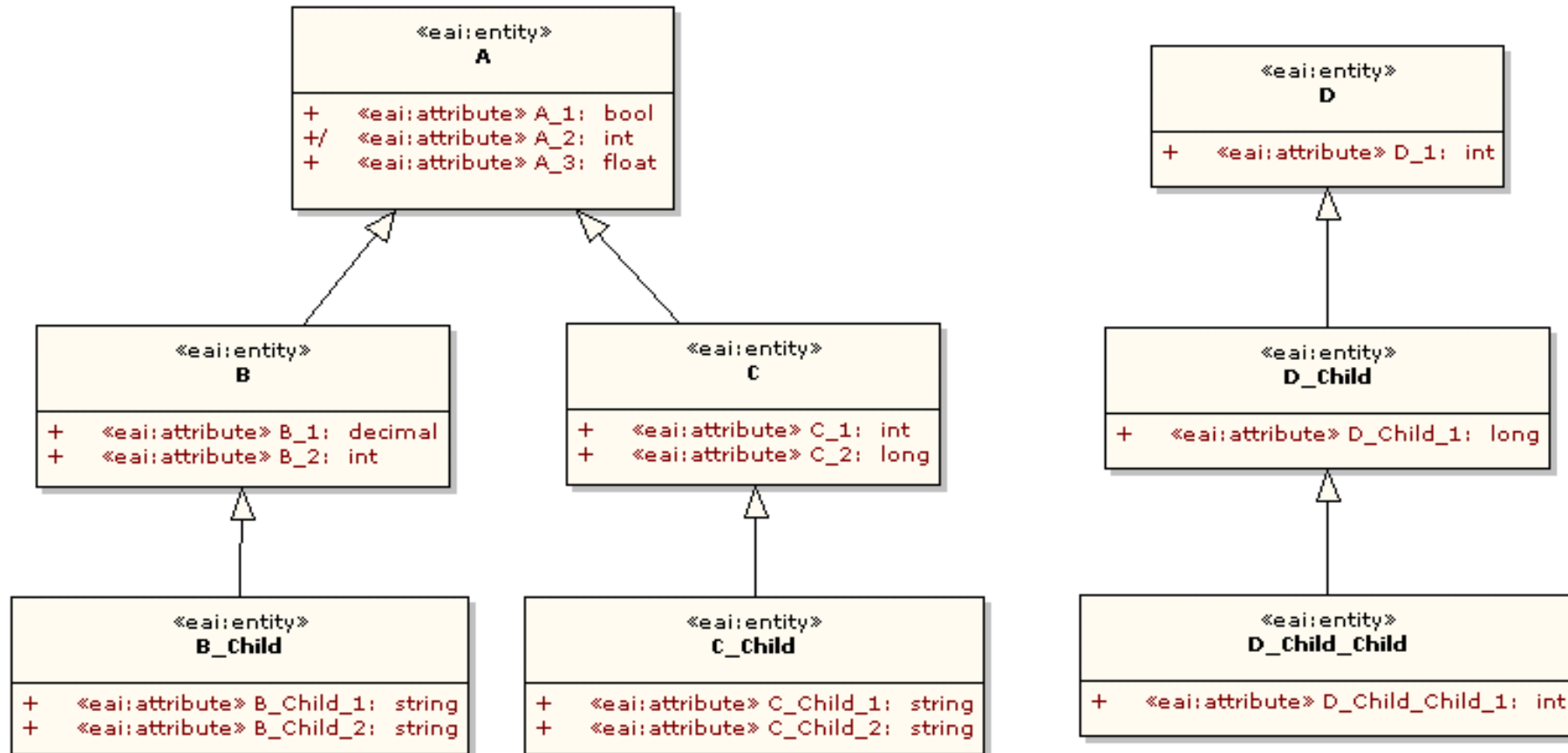
Cascade O/R Strategy



- 3 Tables generated for all domain classes

Roll Up Strategy – Extend Framework

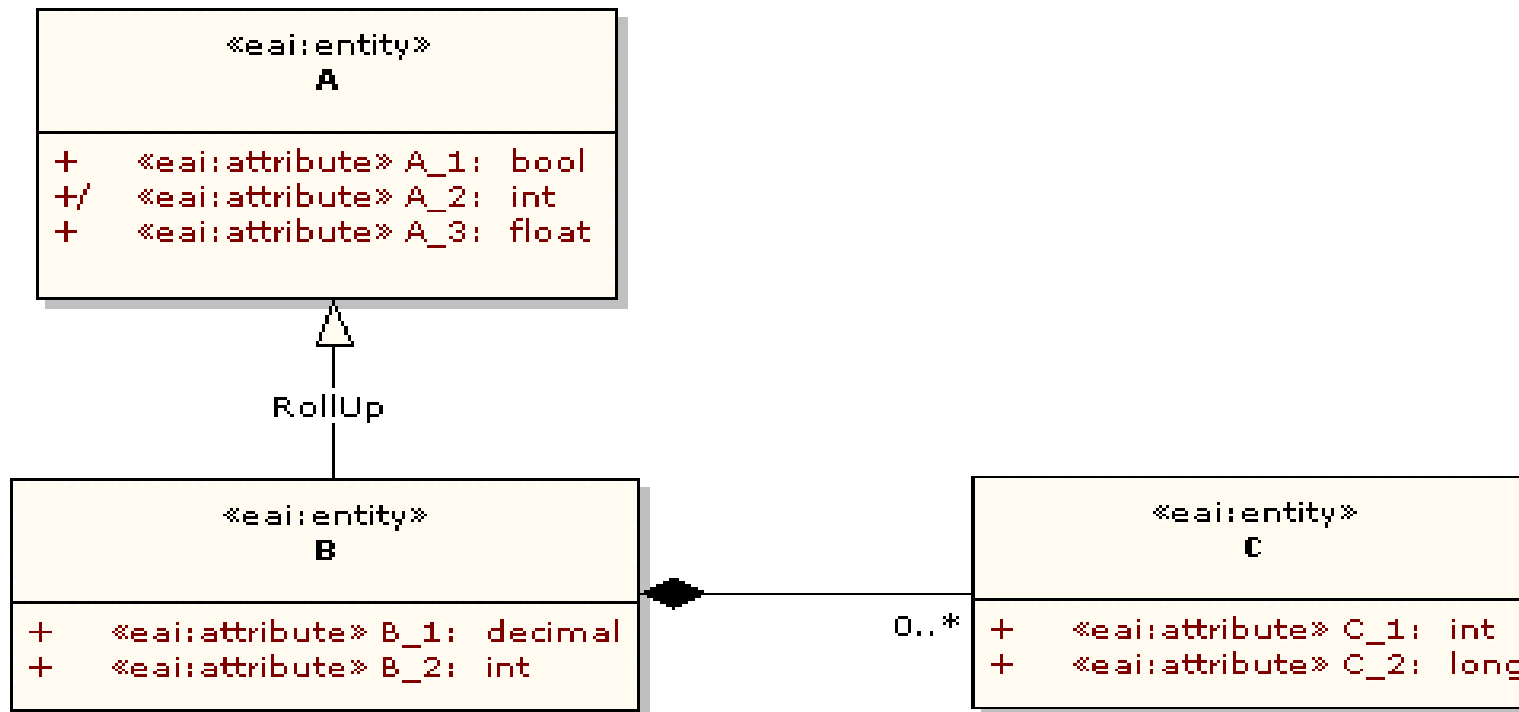
RollUp O/R Strategy



- 2 Tables generated for all domain classes

Strategies Combine

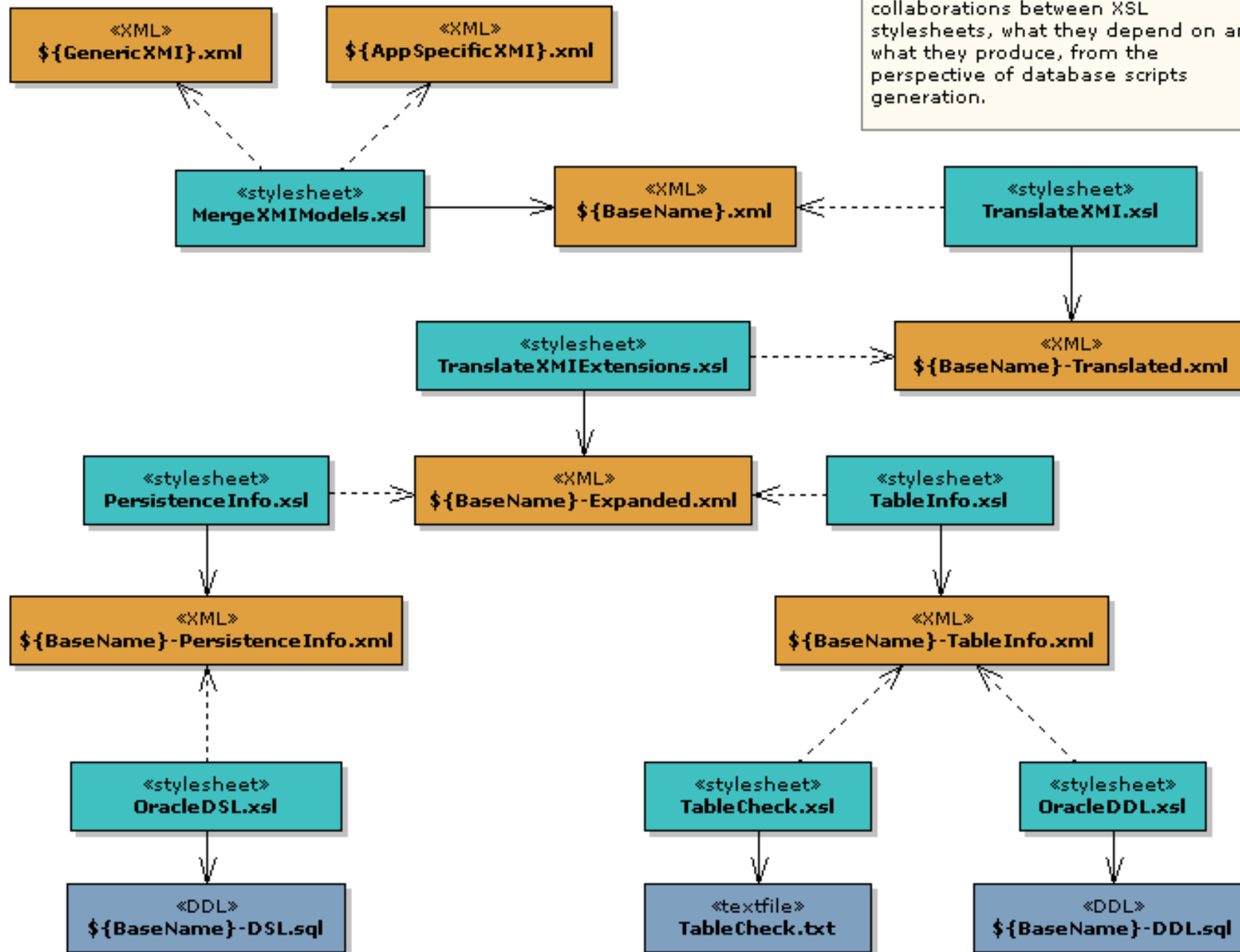
RollUp Link O/R Strategy Test



- Strategies combine in interesting ways
- Metamodel validation required

Database Artifacts XSL Collaborations

This diagram shows high-level collaborations between XSL stylesheets, what they depend on and what they produce, from the perspective of database scripts generation.



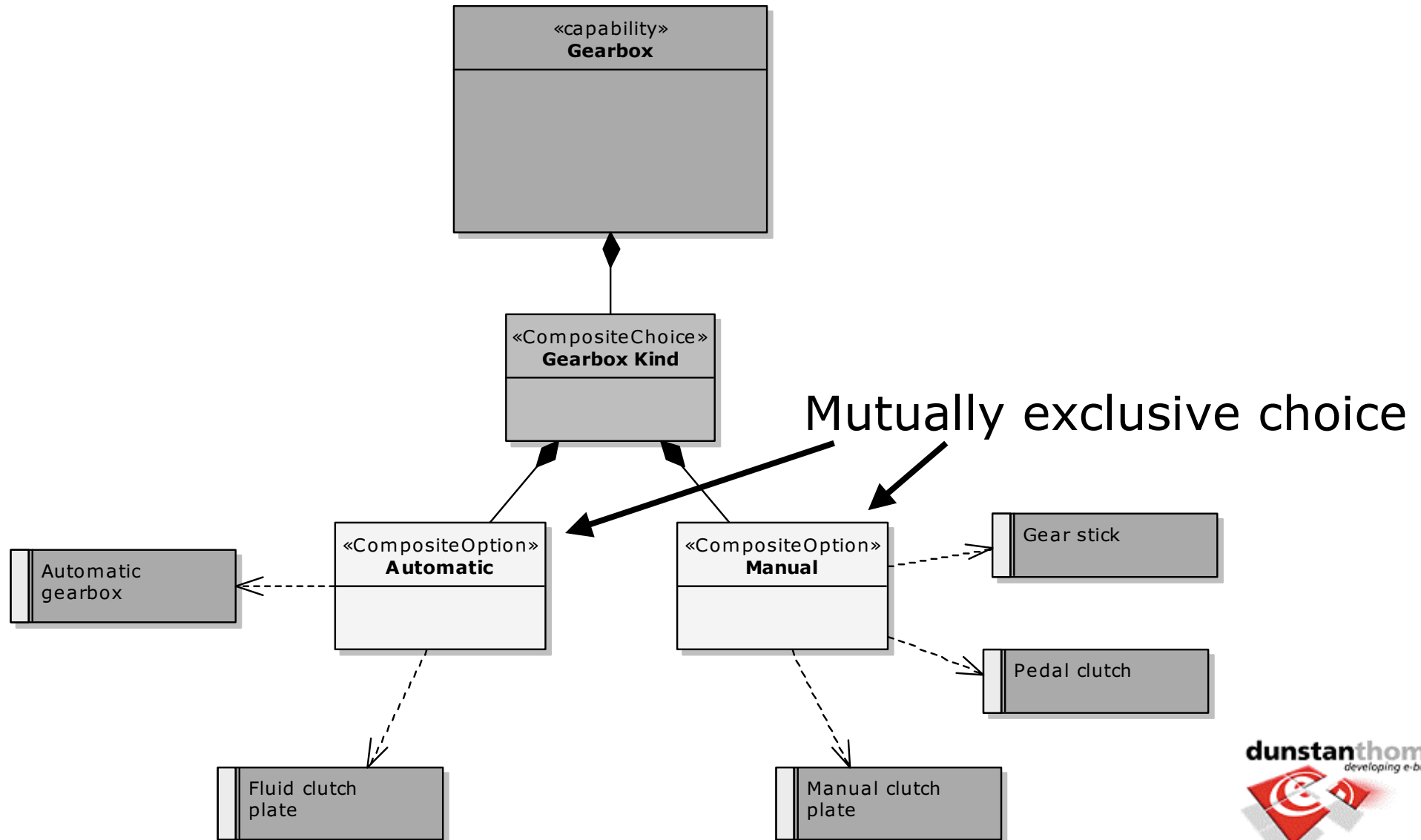
Artefacts Generated

- Entire persistence layer
 - Insert, Delete and Update classes
 - Finders, Factories, Registries...
 - Primary key classes
- Entity classes
 - Request interception for business rules
- Unit tests for the whole shebang
- Vast quantities of tedious codegen

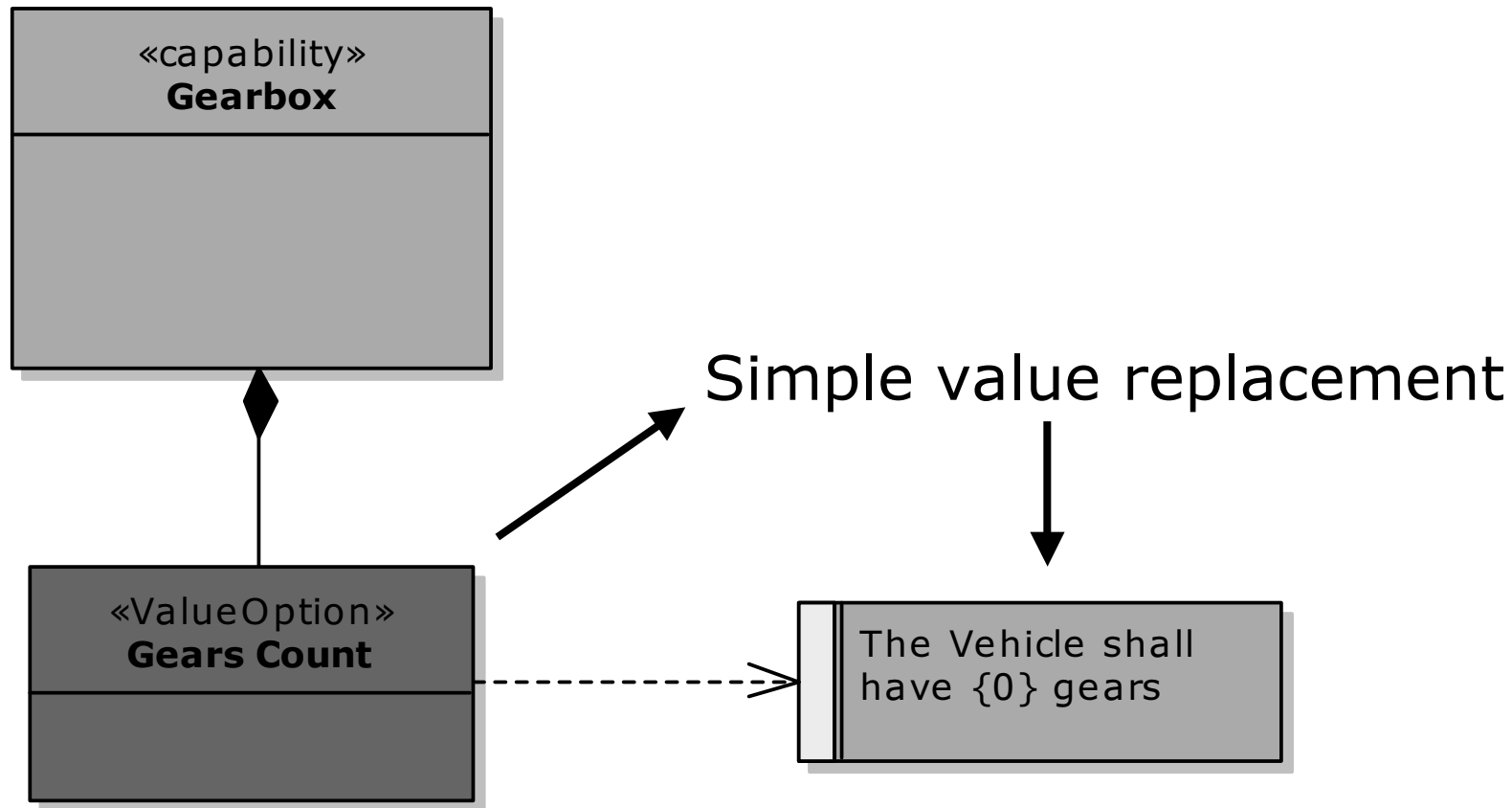
Case Study 2: Unorthodox & More Fun

- Complex requirements management
- Solution: build a DSL-like vocabulary using UML
- Logical model describes generic capabilities
- Requirements associated to individual options
- Options bring requirements into scope in various ways
 - Metamodel encapsulates technical expertise
 - Business users “instantiate” model to produce artefacts

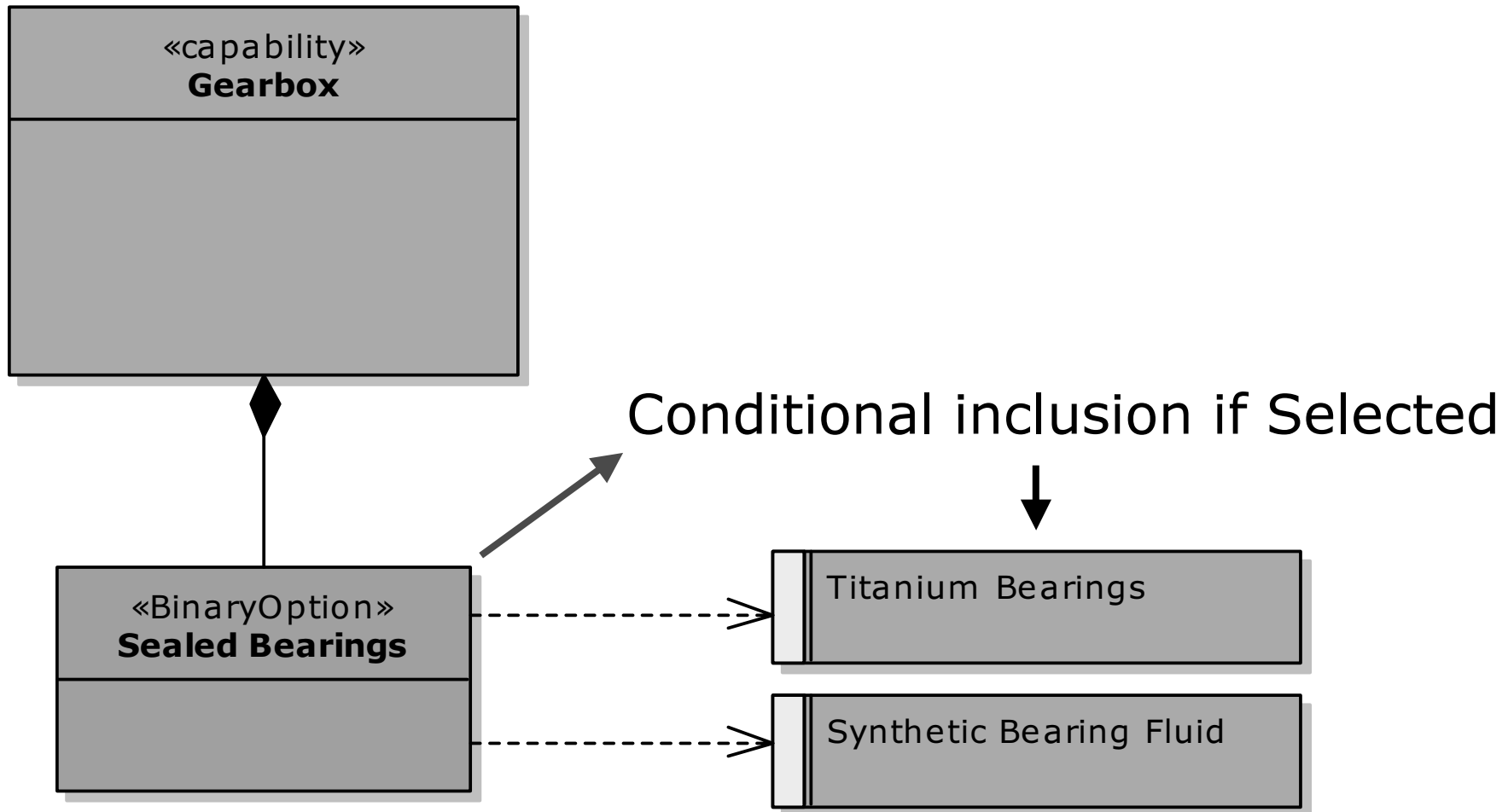
Domain Vocabulary – CompositeChoices



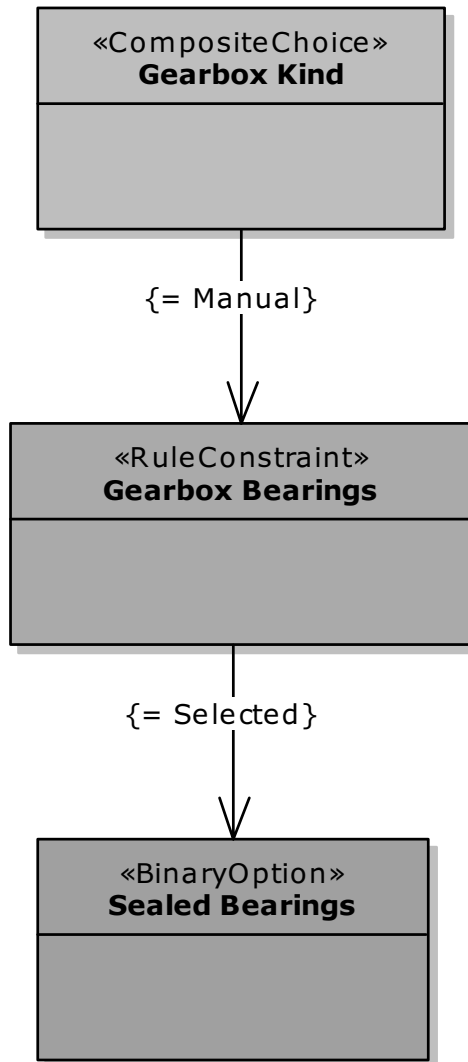
Domain Vocabulary - ValueOptions



Domain Vocabulary - BinaryOptions



Domain Vocabulary – RuleConstraints



- “Instantiation-Time” validation
- Generically programmed
- Programmers require less domain knowledge: A Good Thing?

Business Process Flow

- Market analysts effect high-level choices
 - Business-driven, per-demographic basis
 - Abstracts away technical complexities
- Metamodel transformed to artefacts
 - User Interface
 - Test specifications
 - Technical adherence documents

The Simple Rules

Three categories of generated artefacts

1. Always generated (never edited)
 2. Skeleton generated ('leg up')
 3. Never generated
- Cost / benefit analysis key
 - No middle of the road here

Conclusion

- Technique enjoys some benefits of true MDA
- Low cost of entry & low risk
 - Ironically possible to transform to safety
 - Developers generally prefer and know [insert technology] to OCL / AS
- Methodology-agnostic
- Does not replace high-level tasks
 - OOA/D, technology unaffected

Acknowledgements

- Major contributors to Lightweight MDA
 - Ian Briscoe
 - Edward Garson
 - Jason Young

Thank you for having me by

Edward Garson
egarson@dthomas.co.uk

Dunstan Thomas Consulting

<http://consulting.dthomas.co.uk>

