

Model Engineering in Practice

ProSoftwarica GmbH
TS1008

Technoparkstrasse 1
8053 Zürich

www.proSoftwarica.com

Milan Ignjatovic
abstractionist, architect,
coach, trainer, consultant
mig@proSoftwarica.com

Agenda

- Part 1:

- A word about proSW

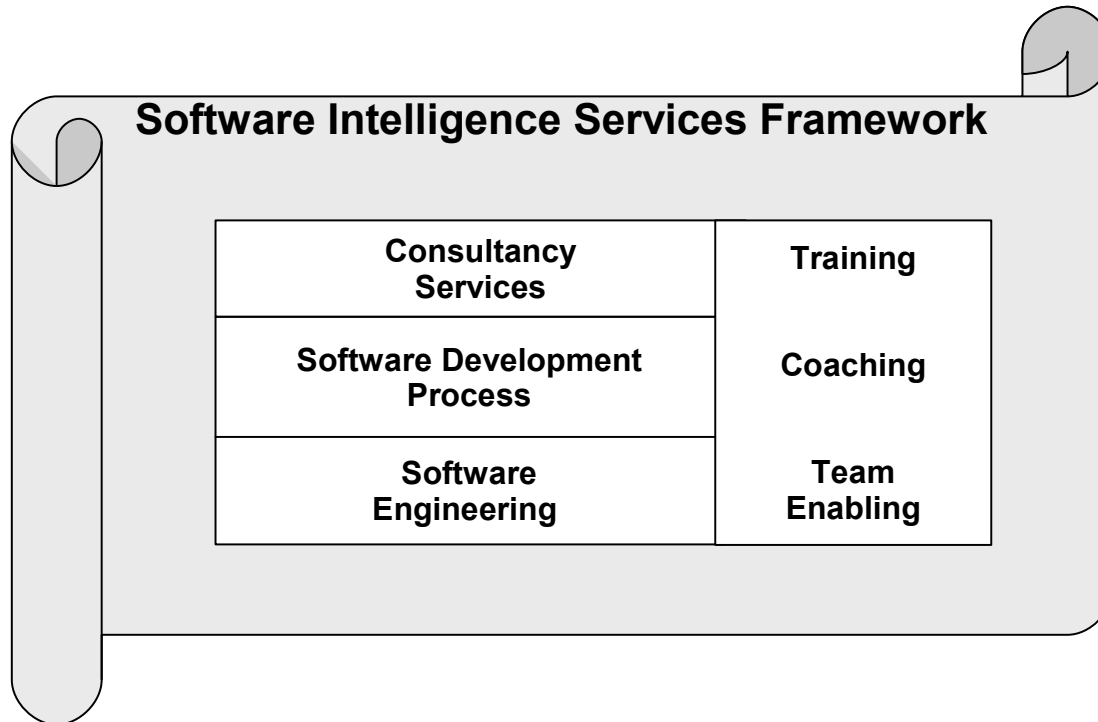
- Part 2:

- A necessary overview of QVT
- A sufficient overview of ATL

- Part 3:

- Product Line in Practice
- Demonstration

About proSW



Course offering

- SAIP
- MDSD
- UML2
- OCL
- QVT
- ATL

- Provider of innovative and practical software solutions
- Create value for our customers by applying cutting edge technologies and elitist principles

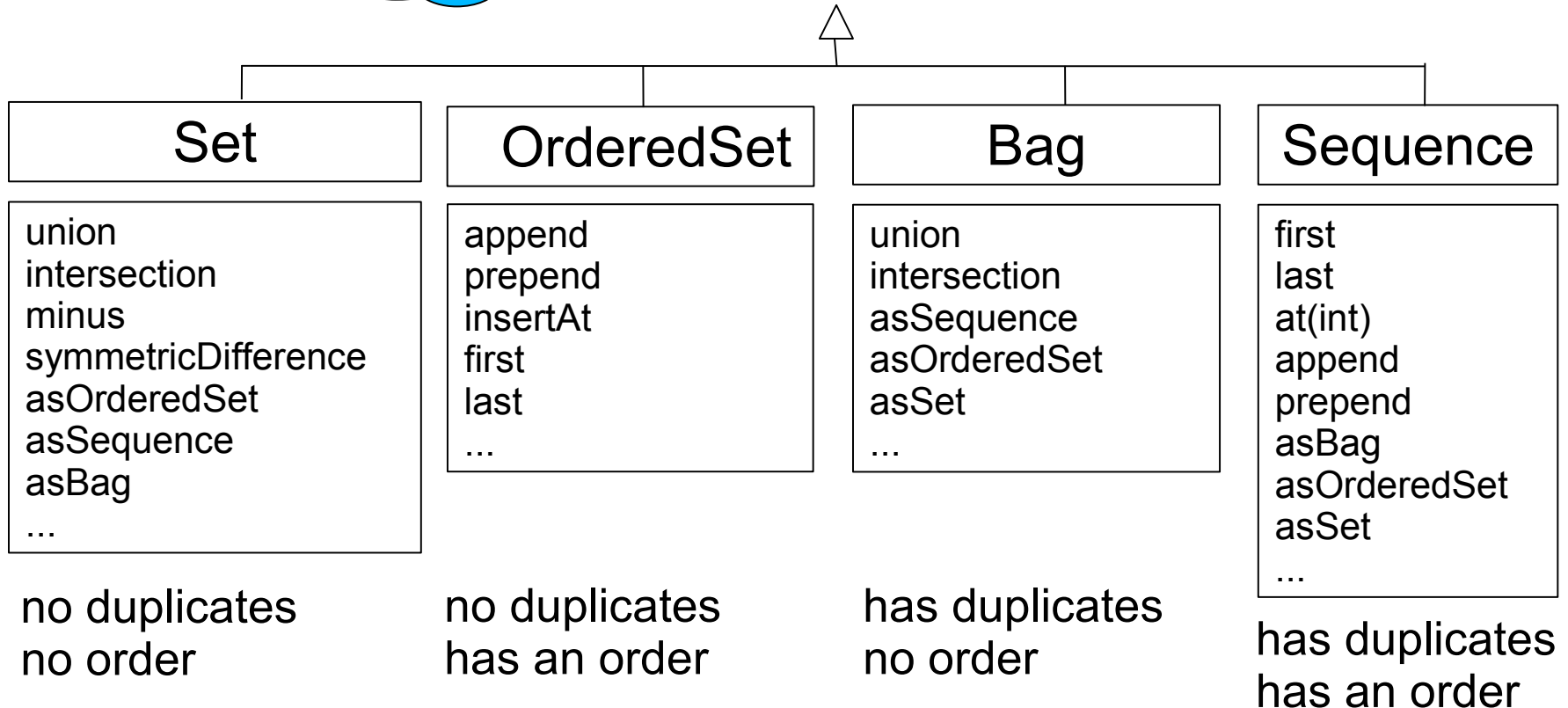
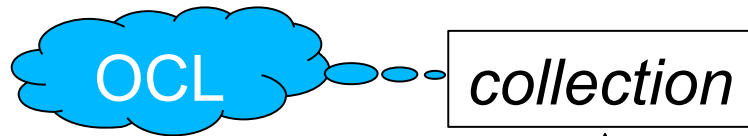
Expression: Pro + Software + ica

- „ica“

- Greek > Latin: a suffix
- forms collective nouns meaning "a collection of something [as information] concerning or relating to" the subject indicated by the combining form

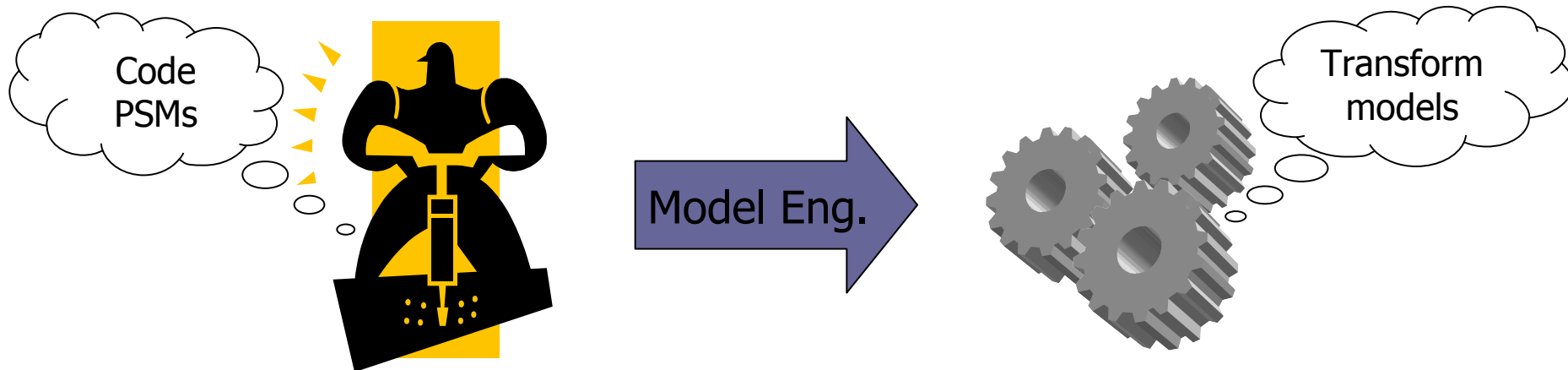
- ProSoftwarica = Professional Software Collection

e.g. professional software collection



Separation of concerns @proSW

- The principle of separation: human / machine
 - Never let a machine do a human job
 - Never let a human do a machine job
- We believe in intelligently crafted systems realised by automating repeatable human tasks

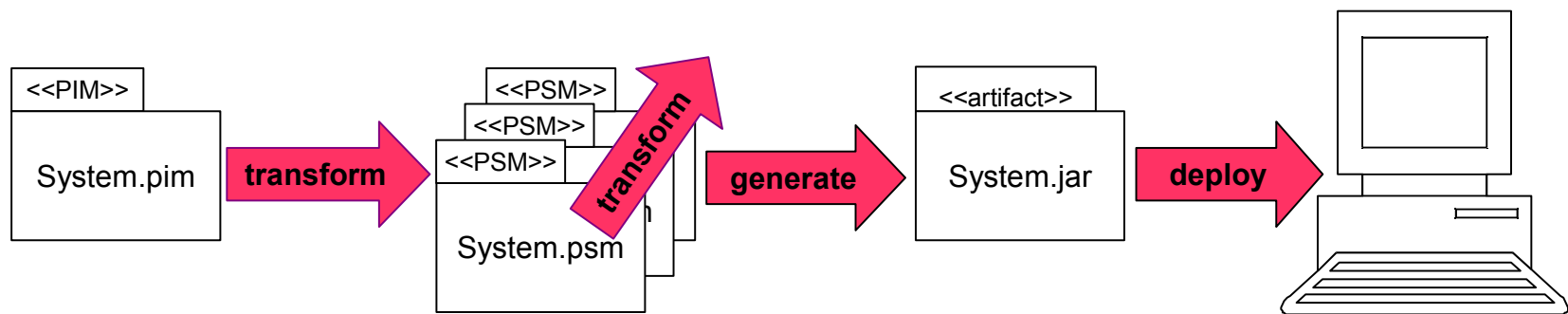




A necessary overview of QVT

MDA revisited

- A source, platform independent model, PIM, is transformed into a target, platform specific model, PSM, by a number of successive transformations of type:
 - PIM to PIM, PIM to PSM, PSM to PSM ...code

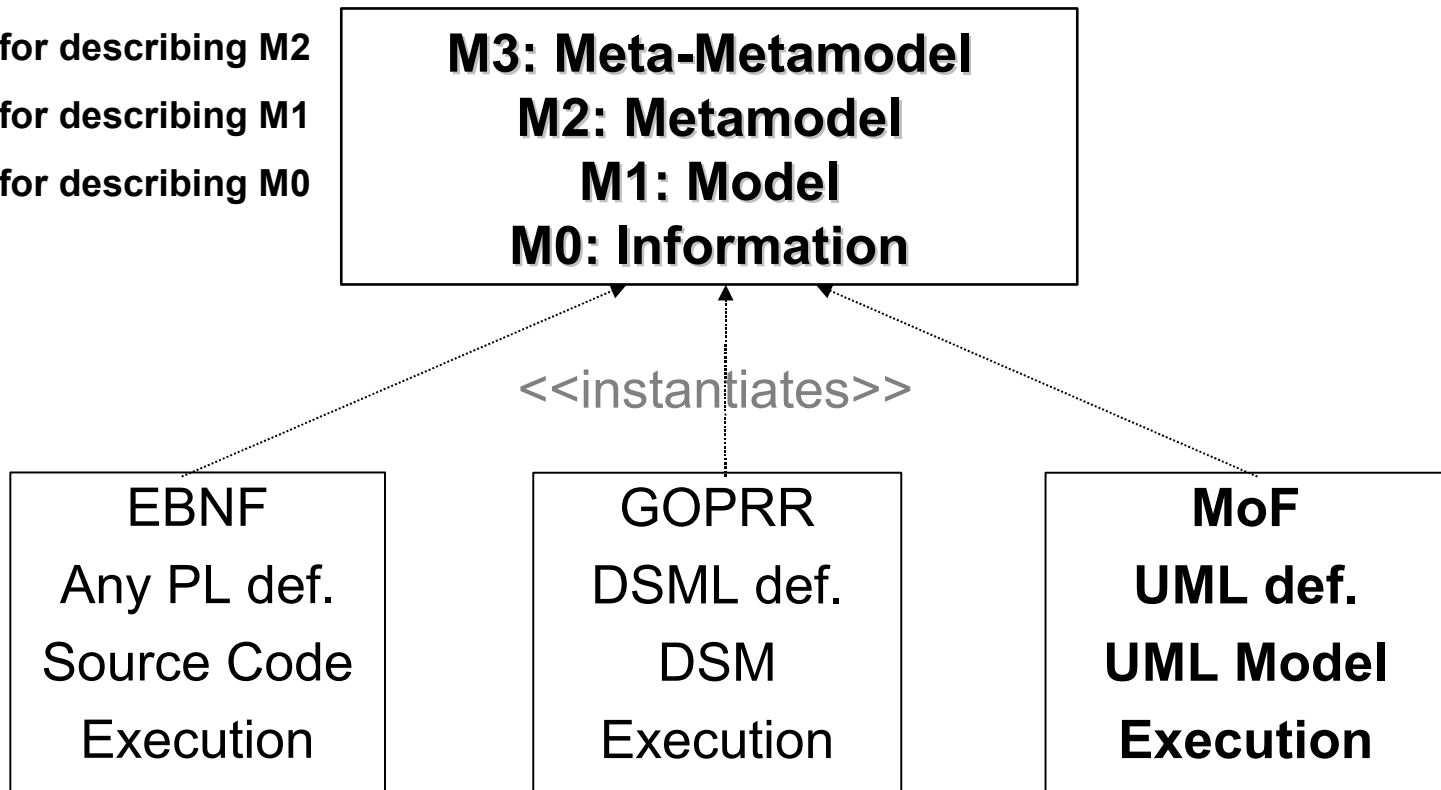


What is QVT?

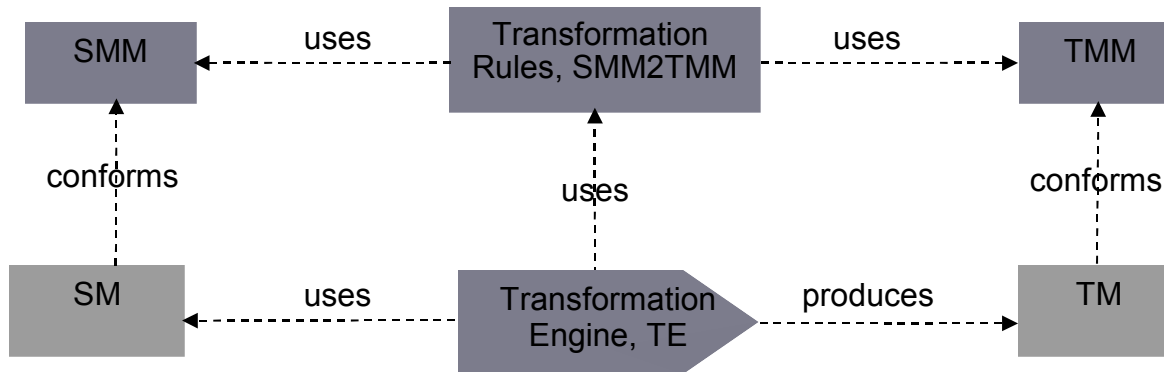
- QVT is an important enabler for the model-driven approach to software engineering
- Defines the semantics of a transformation definition language
- Provides a rich end user-oriented transformation language with a concrete syntax
- We use QVT to define a chain of transformations, thus realizing the PIM to PSM transformation paradigm
- The focus of transformation activity are models and respective meta models

M3 – Meta Stack

Language for describing M2
Language for describing M1
Language for describing M0



Generic Transformation Architecture

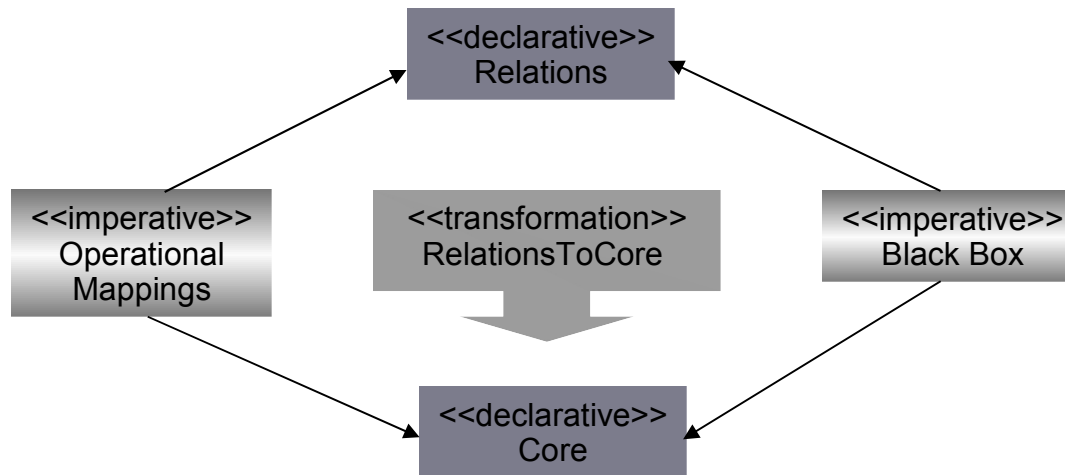


- SM: source model, conforming to the source meta model SMM
- TM: target model, conforming to the target meta model TMM
- SMM2TMM: set of transformation rules for transforming models of type SMM into models of type TMM
- TE: transformation engine uses a set of transformation rules and a source model SM to transform it and produce TM as output

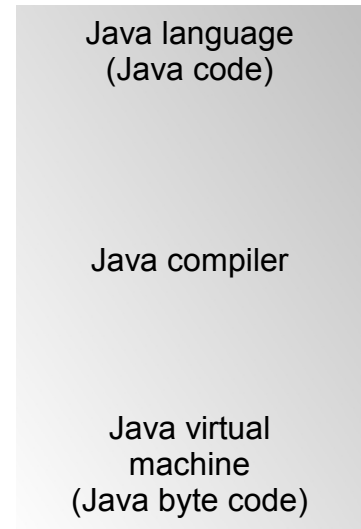
Declarative language

- One mandatory QVT requirement states that the transformation definition language shall be declarative, thus, promoting ease of use.
- QVT is a hybrid i.e. declarative and imperative (in order to satisfy a variety of requirements)
- The result is a clear separation of concerns in its architecture i.e. declarative part is separated from the imperative part

Language Architecture



Analogy to Java



Relations Language

- A transformation consists of a number of special rules called relations
- Relations relate modelling elements defined in each of the meta-models of the source and target models
- After a successful execution of a transformation all defined relations must hold between source and target models

Relation semantics

- A **relation** is a declarative specification of constraints that *must hold* between elements of models declared in a **transformation**
- A relation is defined by two or more **domains** and the **when** and **where** clauses
- Each domain defines a **pattern**
- A **pattern** defines a set of **variables** and binds them to **model elements** defined in the meta-model. It also **defines a set of constraints** that these model elements must satisfy

Execution scenarios

- **Check-only:** transformations are used to verify that models are related in a specified way
- **Enforce:** single direction transformations
- **Enforce:** bi-directional transformations
- **Incremental** updates (in any direction) when one related model is changed after an initial execution
- The TE must have the ability to create as well as delete objects in models, while also being able to specify which objects must not be modified

Additional features -1-

- Scalability of QVT is based on reuse by providing mechanisms for subclassing, composition, use of transformation libraries
- Incremental model updates
- Symmetric declarative transformations
- Imperative transformations and queries
- Dynamic definition i.e. programmatic transformation definition and processing

Additional features -2-

- Parallel launching of transformations
- Object creation and population
- Helper operations perform operations on a set of objects
- Intermediate classes and properties
- OCL and MOF extensions
- Imperative OCL Package

Who uses QVT?

IBM Rational Systems Developer:

- “Rational Systems Developer supports MDA by allowing the user to define multiple levels of models coupled with user-defined transformations between those models and code, resulting in a clearer separation of concerns across the lifecycle”
- ftp://ftp.software.ibm.com/software/rational/web/datasheets/rsd_ds.pdf

Who uses QVT...

Borland Together Architect 2006:

- New Model-Driven Architecture™ (MDA) features include OMG's Query View Transformation (QVT) used in model-to-model transformations and support for OCL 2.0 with syntax highlighting, validation, and code sense
- <http://www.borland.com/us/products/together/index.html>



A sufficient overview of ATL

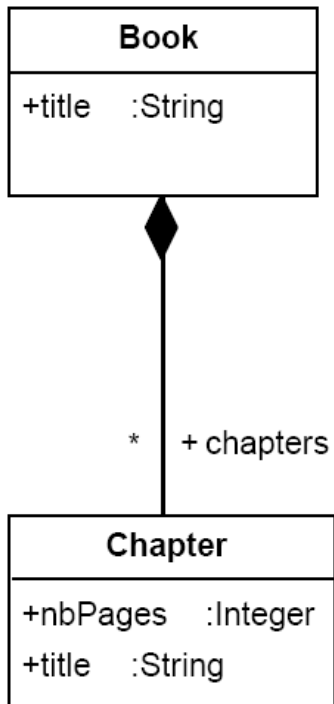
What is ATL?

- ATL is one of a dozen answers to the OMG QVT RFP in Oct 2002: it is a QVT dialect
 - Project leader: Jean Bezivin
 - Team: ATLAS group, INRIA, Uni Nantes
- ATL is a popular Eclipse Plugin with ca. 50 contributed transformations and growing...
- ATL transformations rely heavily on OCL
- Industrial use: adopted by Airbus

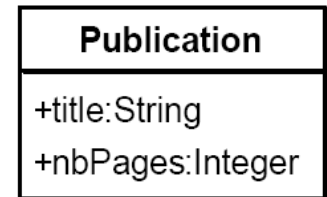
Elements of ATL

- ATL Module
 - Header
 - Import section
 - Helpers and Attributes
 - Queries
 - Rules
- Module execution modes
 - default, refines

Working example: Book2Publication



- Input Metamodel: Book
- Output Metamodel: Publication
- Input model: inputBooks
- Objective:
 - Write a transformation, that visits all chapters of a Book object and calculates the sum of all pages per Book
 - For each Book object create one Publication object where $\text{pub.nbPages} = \text{sum chapter.nbPages}$



Module header

- An ATL module defines a transformation
- The header section defines the transformation module:
 - **module** Book2Publication
 - **create** OUT : Publication **from** IN : Book
- **create** and **from** keywords declare output and input models where OUT and IN are variables i.e. model names that will be assigned values just before execution

Import section

- Import section enables import of existing ATL libraries i.e. drives reuse:
 - uses strings;
 - uses Java;
- Import section is optional

Helpers

- Helpers are analog to Java methods
- Helper syntax:
helper [*context context_type*]? **def** :
helper_name(parameters) : return_type = exp;
- **exp** - the body of a helper is specified as an OCL expression
- **context** is optional
if not specified then the enclosing ATL module is the default context

Helpers: Examples

I like to read OCL backwards:
Calculate the sum of all pages
over all chapters of the context
instance (the Book object)
and return it to the caller.

- Define a helper called `getSumPages`, returning an `Integer` to collect the sum of `nbPages` over all existing `Book` objects:

```
-- getSumPages using the OCL sum operation
helper context Book!Book def :
getSumPages () : Integer =
self.chapters->collect (e|e.nbPages) .sum ();
```

- `Book!Book` - `!` is the qualification operator
 - Context is the `Book` class from the `Book` Metamodel

Rules

- ATL language defines transformation rules which are used to generate target model elements
- ATL defines two different kinds of transformation rules:
 - Matched rules match specified model elements of a source model and generate from them a number of distinct target model elements
 - Matched rules are the declarative programming language part of ATL – preferred
 - Called rules have to be invoked from an ATL imperative block in order to be executed

Matched Rule details

- Matched rule enable us to specify:
 - which source model element must be matched
 - the number and type of generated target model elements
 - how these target model elements should be initialized using the matched source elements

Matched rule: Book2Publication

```
rule Book2Publication {  
  from  
    b : Book!Book  
  
  to  
    out : Publication!Publication  
    (  
      title <- b.title,  
      nbPages <- b.getSumPages()  
    )  
}
```

Matched rule: Book2Publication

```
rule Book2Publication {  
  from  
    b : Book!Book  
    ( b.getSumPages().debug('Book sum pages:') > 2 )  
  to  
    out : Publication!Publication  
    (  
      title <- b.title,  
      nbPages <- b.getSumPages()  
    )  
}
```

Attributes

- Attribute is a constant that is specified within a specific context
- Attribute accepts no parameter i.e. it is a “parameterless helper”
- The code of a helper is executed each time it is invoked (because it is passed a parameter)
- For a given execution context (an input model element or the ATL module), an attribute will always return the same value
- The ATL engine therefore computes the return value of an attribute only once

Queries

- An ATL query is an operation that computes a single primitive value (type b.i.r.s.) from a set of source models
 - **query** query_name = exp;
- A query unit contains a single query element and a number of helpers and attributes

Module execution modes

- The ATL execution engine defines two execution modes:
 - Default execution mode forces the ATL developer to explicitly specify how all target model elements are to be generated from source model elements
 - The refining execution mode allows ATL developers to focus on the ATL code dedicated to the generation of modified target elements
 - Model elements that remain unchanged (between source and target) are implicitly copied from the source to the target model by the ATL engine



Product Line Demo

- At this point, I would like to give word over to Dennis...



For more info

- http://www.modelbased.net/mda_tools.html