MODEL DRIVEN SOFTWARE DEVELOPMENT

LECTURE : 7

MDD

- The first and most known application scenario for MDSD is
  - *software development automation* (typically known as model-driven development (MDD))
  - where model-driven techniques are consistently employed with the goal of automating as much as possible the software lifecycle from the requirements down to the deployed application

SOFTWARE AUTOMATION DEVELOPMENT

- Software development automation consists of
  - starting from a high level (or early) representation of the desired software features and deriving a running application out of it
  - possibly through a set of intermediate steps to enable some degree of user interaction with the generation process
- The running application can be obtained through one or more model transformations producing more and more refined version of the software description, until an executable version of it is reached
  - In each phase, models are (semi)automatically generated using model-to-model transformations
  - taking as input the models obtained in the previous phase (and manually completed/refined when necessary).
  - In the last step, the final code is generated by means of a model-to-text transformation from the design models.

SOFTWARE AUTOMATION DEVELOPMENT

- Software development automation consists of
  - starting from a high level (or early) representation of the desired software features and deriving a running application out of it
  - possibly through a set of intermediate steps to enable some degree of user interaction with the generation process
- The running application can be obtained through one or more model transformations producing more and more refined version of the software description, until an executable version of it is reached
  - In each phase, models are (semi)automatically generated using model-to-model transformations
  - taking as input the models obtained in the previous phase (and manually completed/refined when necessary).
  - In the last step, the final code is generated by means of a model-to-text transformation from the design models.

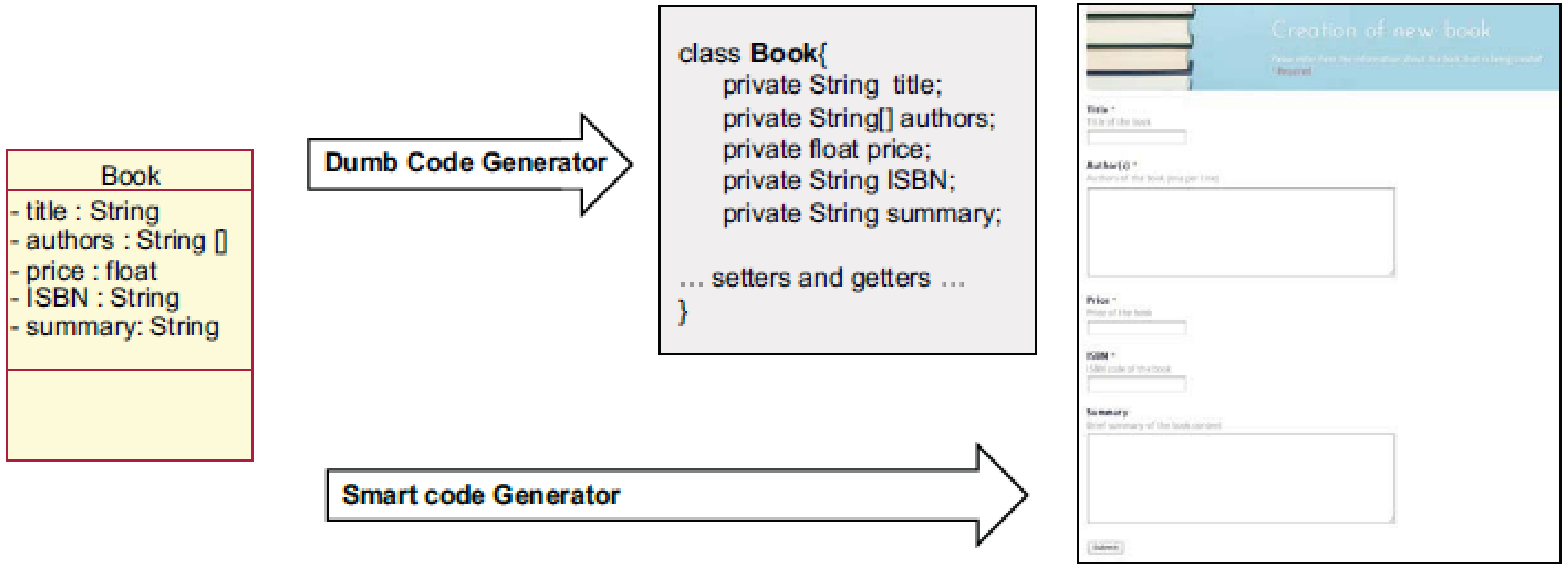
EXECUTABLE MODELS

- In order to be able to generate a running system from the models, they must be executable (not all but some)
- An executable model is a model complete enough to be executable
  - From a theoretical point of view, a model is executable when its operational semantics are fully specified
  - In practice, the executability of a model may depend on the adopted execution engine
  - models which are not entirely specified but that can be executed by some advanced tools that are able to fill the gaps
- Completely formalized models that cannot be executed because an appropriate execution engine is missing

CONTD…

- For instance, following a simple class diagram specifying only the static information of a domain
  - A trivial code-generator will only be able to generate the skeletons of the corresponding (Java) classes
  - while a more complex one would be able to infer most of the system behavior out of it.
- As an example, this advanced generator could assume that for each class in the diagram
  - the system will need to offer all typical CRUD (create/read/update/delete) operations and
  - thus, it could directly decide to create all the forms and pages implementing these operations

EXAMPLE…



```
class Book{
      private String  title;
      private String[] authors;
      private float price;
      private String ISBN;
      private String summary;

… setters and getters …
}
```

Creation of new book

Please enter here the information about the book that is being created
* Required

Title *
Title of the book

Author(s) *
Authors of the book (one per line)

Price *
Price of the book

ISBN *
ISBN code of the book

Summary
Brief summary of the book content

Submit

2 MAIN APPROACHES FOR EXECUTABLE MODES

Two different alternative strategies to “implement” execution tools and thus make executable models actually execute

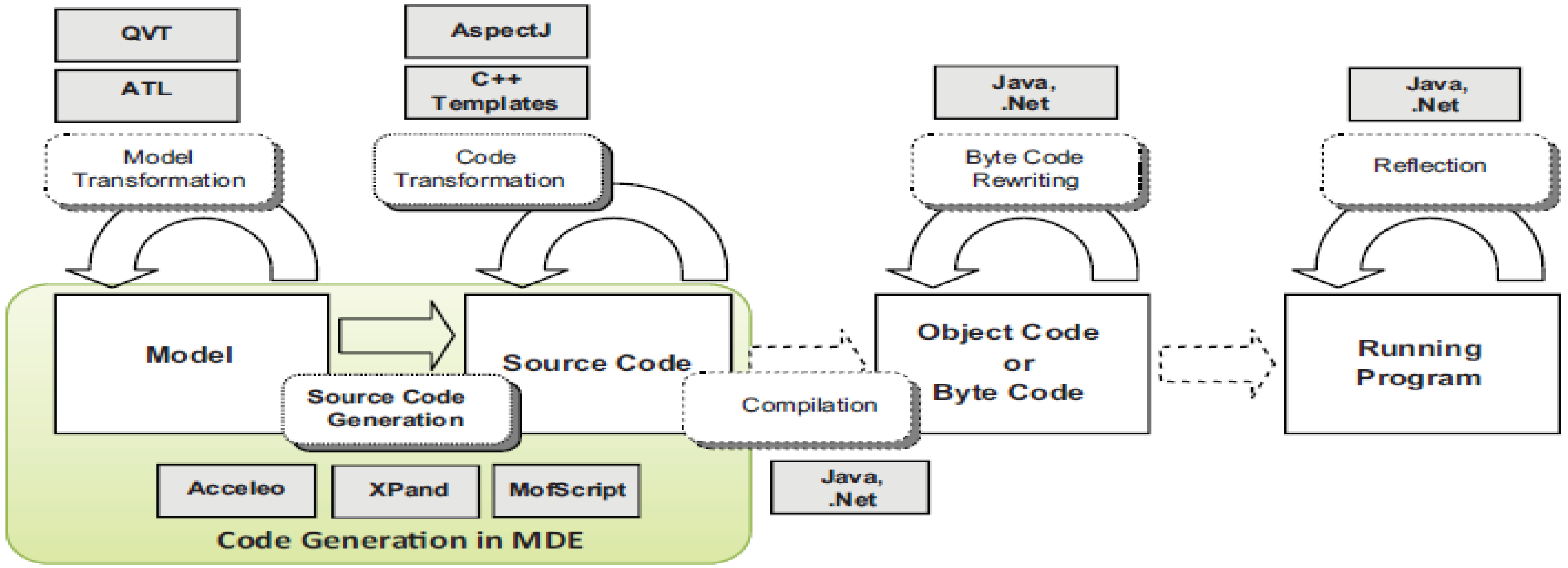
- Code-generation
- model-interpretation

CODE GENERATION

- Generating running code from a higher level model in order to create a working application
  - like compilers are able to produce executable binary files from source code
  - Code-generators are also sometimes referred to as *model compilers*
- Code generation by means of a rule-based template engine
  - The code generator consists in a set of templates with placeholders that once applied (instantiated) on the elements in the model, produce the code
- Once the code is generated, common IDE tools can be used to refine the source code produced during the generation (if necessary), compile it, and finally deploy it.

CONTD…

- The single goal of a code-generation process is to produce a set of source files from a set of models.
- Obviously, by "source code" we do not restrict ourselves to programming languages
- By means of code-generation we can transform many kinds of models to many types of software artifacts (test cases, make files, documentation, configuration files, etc).



ADVANTAGES OF CODE GENERATION

- It protects the intellectual property of the modeler
  - It allows the generation of the running application for a specific client, without sharing the conceptualization and design, which is the actual added value of the work and which can be reused or evolved in the future for other projects.
- The generated code is produced in a standard programming language that any developer can understand.
- Code-generation allows reusing existing programming artifacts.
- A code generator is usually easier to maintain, debug, and track because it typically consists of rule-based transformations, while an interpreter has a generic and complex behavior to cover all the possible execution cases.

DRAWBACKS OF CODE GENERATION

- One of the big hurdles in adopting code-generation techniques is
- The fact that the produced code does not look "familiar" to the developers
- Even if it behaves exactly as expected and it is written in the programming language of choice, it could be very different from the code that actual programmers would write and this makes them reluctant to accept it (for instance, they may not feel confident to be able to tune it or evolve it if needed in the
- future).
- In this sense, it could be useful to define a *Turing test for code-generation tools,* Similar to the classic Turing test for AI

TURING TEST

A human judge examines the code generated by one programmer and one code-generation tool for the same formal specification. If the judge cannot reliably tell the tool from the human, the tool is said to have passed the test.

- Tools passing the test would have proved they can generate code comparable to that of humans and should be acceptable to them.
- The chances of passing the Turing test for code-generation may be increased by building the code generation templates starting from existing code manually written by the developers
- Thus, when running these templates, they should produce code which looks more familiar to the developers.

PARTIAL VS. FULL GENERATION

- When the input models are not complete and the code-generator is not smart enough to derive or guess the missing information
  - we can still benefit from a code-generation approach by creating a partial implementation of the system
- Partial generation means that programmers will need to complete the code manually to obtain a fully functional system
  - This leads to a situation where no single source of information exists:
  - both the models and the code contain important information that may not be replicated in the other artifact
  - but at the same time some pieces of information appear in both places (e.g., the parts of the model that have been directly translated into the code).
- Both in full and partial generation scenarios, one objective to keep in mind is to:
  - Try to generate the smallest possible amount of code to achieve the desired functionality.

MAXIMIZE THE BENEFITS FROM PARTIAL GENERATION

- Defining protected areas in the code, which are the ones to be manually edited by the developer.
  - Thanks to this, a code-generation tool can still regenerate the code with the assurance that the manually added code excerpts will be preserved;
- Using round-trip engineering tool, so that changes on the code are, when possible, immediately reflected back to the model to keep both in sync;
- Focusing on complete generation of parts of the system more than on a partial generation of the full system.