MODEL DRIVEN SOFTWARE DEVELOPMENT

LECTURE : 10

META-PROGRAMING

- A traditional program is mainly code that manipulates a data structure and produces output.
- An example of this is an sqrt() function that takes an integer (i.e. a data structure) as input, manipulates it, and outputs its square root.
- Of course, data structures can be a lot more complex than a simple integer or a double, but that's the general idea of a program.
- A Compiler, on the other hand, is a program that takes source code (again, a data structure) as input, transforms it into a bunch of data structures, it can understand better, and produces output
- That might be Binary code, Bytecode, or Intermediate language among other formats/structures.
- An Interpreter is *roughly* just a compiler that does this process every time you run the code.

CONTD…

- Now, if we can write code that manipulates data structures,
- and write compilers that treat human written code as data structures,

why can't we write code that writes, or manipulates, other code?

Meta programming is a programming technique in which computer programs have the ability to treat programs as their data.

META-PROGRAMS

- Metaprograms – programs that generate other programs.
- Code generators are meta-programs that process specifications (or models) as input parameters, and which generate source code as output.
- Meta-programs can be run at different times in relation to the generated program:
  - Completely independently of the base program – that is, before it.
  - During compilation of the base program.
  - While the base program runs.

META-PROGRAMS

- The metaprogram and the part of the base program to be created manually are usually specified separately.
- The generated code is also separated from the manually-created code, and both must be integrated by the developer.
- **base program** and **metaprogram** are mixed, and similarly the result of the generation process already contains manually-created as well as generated code, so is also mixed.
- However, the created program no longer knows anything about the metaprogram.
- We refer to that as static metaprogramming

M2T TRANSFORMATION LANGUAGES

- Template-based approach at a glance
- Components of a template-based approach

Templates

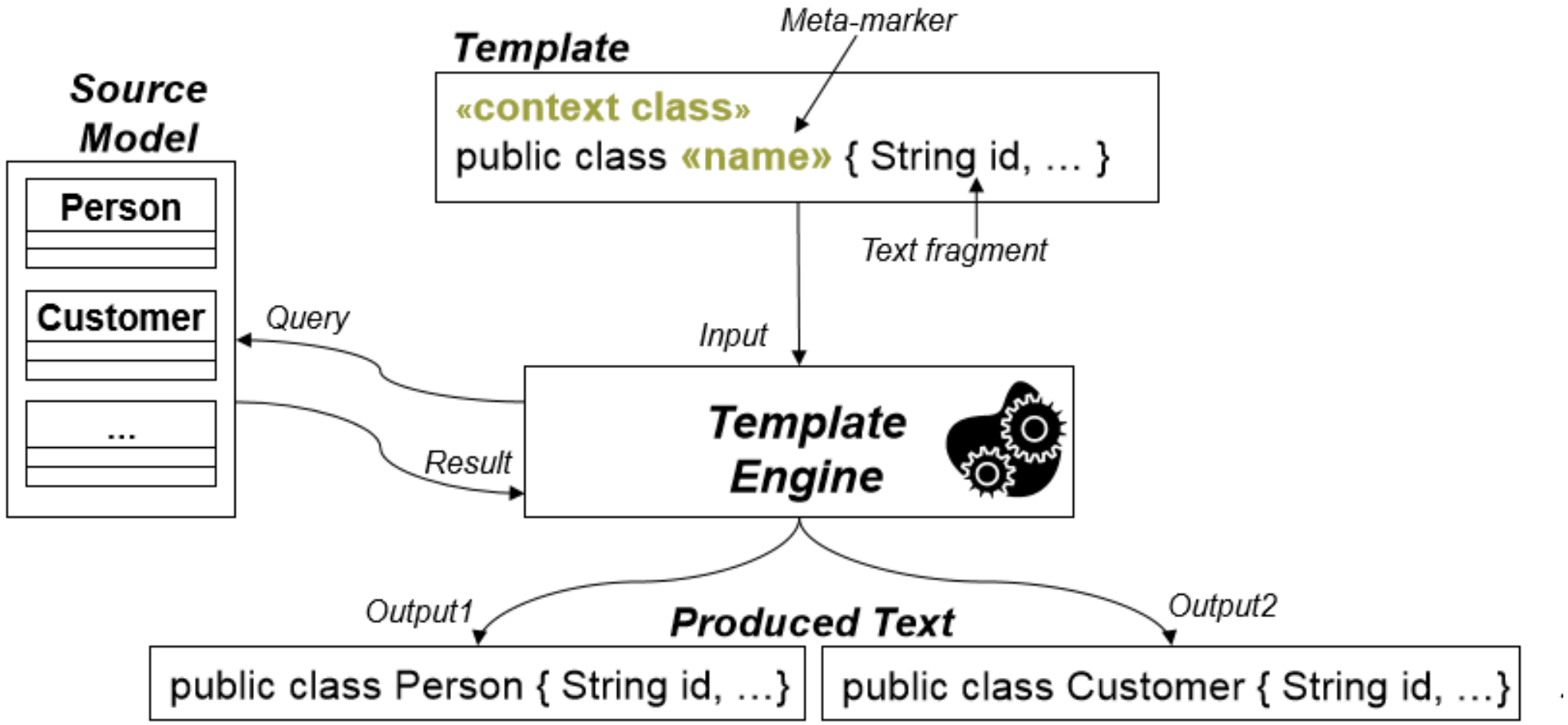
- Text fragments and embedded meta-markers

Meta-markers query an additional data source

- Have to be interpreted and evaluated in contrast to text fragments

Template engine

- Replaces meta-markers with data at runtime and produces output files


Meta-marker
Template
«context class»
public class «name» { String id, … }
Source Model
Person
Customer
...
Text fragment
Query
Input
Template Engine
Result
Output1
Produced Text
Output2
public class Person { String id, …}
public class Customer { String id, …}

M2T TRANSFORMATION LANGUAGES

A bunch of template languages for M2T transformation available

- XSLT
- JET, JET2
- Xpand, Xtend
- OFScript
- Acceleo

M2T TRANSFORMATION LANGUAGES

- **Separated static/dynamic code**
- **Templates separate static code, i.e., normal text, from dynamic code that is described by meta-markers**
  - A template can be seen as a kind of blueprint which defines static text elements
  - Shared by all artifacts as well as dynamic parts which have to be filled with information specific to each particular case
  - A template contains simple **text fragments** for the **static part** and ***meta-markers*** for the **dynamic part**.
  - Meta-markers are placeholders and have to be interpreted by a ***template engine*** which processes the templates and queries additional data sources to produce the dynamic parts.

M2T TRANSFORMATION LANGUAGES

- **Explicit output structure**
  - Using templates allows to explicitly represent the structure of output text within the template by embedding the producing code in the produced text
- **Declarative query language**
  - Within the meta-markers, code is used to access the information stored in the models
- **Reusable base functionality**
  - CurrentM2Ttransformation languages come with tool support
  - Support for reading in models, serialize text to files, …
    - to directly read in models and to serialize text into files by just defining configuration files.
    - no tedious redefinition of model loading and text serializing has to be developed manually