Mapping Code Generation Templates to a Reference Implementation

Towards Automatic Code Migration

Daniel Lucrédio
Renata Pontin de Mattos Fortes

This work was partially supported by the National Institute of Science and Technology for Software Engineering (INES - www.ines.org.br), funded by CNPq and FACEPE, grants 573964/2008-4 and APQ-1037-1.03/08.
Agenda

- The motivation
- The mapping
- The implementation
- Current status
- Future work
- Some conclusions
A well-known MDA description

- CIM
- PIM
- PSM
- Code
- Requirements
- Analysis
- Design
- Implementation
- Maintenance

Code
A less-known side of MDA

- CIM
- PIM
- PSM

Requirements

Analysis

Design

Implementation

Maintenance

Code
The motivation

- Ways to facilitate the work on the back-end side of MDE

Template-based code generation

Reference implementation
The motivation

- Ways to facilitate the work on the back-end side of MDE

Template-based code generation

Text file (code to be generated)
+ Tags (how to generate it)

Reference implementation
The motivation

• Ways to facilitate the work on the back-end side of MDE

1. Build example application
   (“Normal” development ... and testing, debugging ...)

2. Migrate the code to templates

Reference implementation
Reference implementation

```html
<html><body>
E-mail: <input type="text" size="30"> <br/>
Name: <input type="text" size="50"> <br/>
Password: <input type="password" size="30"> <br/>
</body></html>
```

Iterative structure

```html
<html><body>
<wrapper>
  <% Field[] fields = Input.getAllFields();
      foreach(Field f : fields) { %>
    Label: <input type="text" size="30"> <br/>
  <% } %>
</wrapper>
</body></html>
```

Repetition: 1 per field

```html
<html><body>
Label: <input type="text" size="30"> <br/>
Label: <input type="text" size="30"> <br/>
Label: <input type="text" size="30"> <br/>
</body></html>
```
Reference implementation

```html
<html><body>
E-mail: <input type="text" size="30"> <br/>
Name: <input type="text" size="50"> <br/>
Password: <input type="password" size="30"> <br/>
</body></html>
```

Repetition: 1 per field

```html
<html><body>
Label: <input type="text" size="30"> <br/>
Label: <input type="text" size="30"> <br/>
Label: <input type="text" size="30"> <br/>
</body></html>
```
Reference implementation

Code migration

Iterative structure

Field[] fields = Input.getAllFields();
foreach (Field f : fields) {
    Label: <input type="text" size="30"> <br/>
}

<h4>Repetition: 1 per field</h4>

<html><body>
Label: <input type="text" size="30"> <br/>
Label: <input type="text" size="30"> <br/>
Label: <input type="text" size="30"> <br/>
</body></html>
Reference implementation

```html
<html><body>
E-mail: <input type="text" size="30"> <br/>

Name: <input type="text" size="50"> <br/>

Password: <input type="password" size="30"> <br/>

<% Field[] fields = Input.getAllFields();
foreach (Field f : fields) {
    Label: <input type="text" size="30"> <br/>
}%>

Repetition: 1 per

</body></html>
```
Reference implementation

```html
<html><body>
<% Field[] fields = Input.getAllFields();
   foreach(Field f : fields) { %>
   <f:label> <input type="text" size="30"> <br/>
<% } %>
</body></html>
```

Value replacement

```html
<html><body>
E-mail: <input type="text" size="30"> <br/>
Name:  <input type="text" size="30"> <br/>
Password: <input type="text" size="30"> <br/>
</body></html>
```
Reference implementation

Code migration

Field[] fields = Input.foreach (Field f : fields)

<f:label> <input type="text" size="30"> <br/>

Value replacement

E-mail: <input type="text" size="30">
Name: <input type="text" size="30">
Password: <input type="text" size="30">

</body></html>
The motivation

• Templates + reference implementation:
  + **Reuse** of large amounts of code
  + **Non-trivial** generators
  + **Pre-tested** code in the templates
  + **Any** programming environment

  – **Consumes** 20–25% of (reference implementation) development time
  – Causes code **duplication**
A solution

• Automatic code migration?

• No existing tool supports it
  – At least not the commercial ones!

• Literature does not seem to worry about it
  – Except round-trip engineering, but not exactly
A solution

- We started an investigation
  - Non-trivial problem

Here we present the first (baby) steps

Mapping

Templates – ref. implementation

Using code annotation
The mapping

Seven mapping types

According to common template constructions

1. Simple copy
2. Direct substitution
3. Indirect substitution
4. Repetition
5. Conditional
6. Inclusion
7. New file
Type 1 – Simple copy

Mapping

Template

Generated code
**Type 1 – Simple copy**

**Mapping**

Template ➔ Generated code

**Example**

<table>
<thead>
<tr>
<th>Input model</th>
<th>Template</th>
<th>Generated code</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>s = &quot;abc&quot;</code></td>
<td><code>s = &quot;abc&quot;</code></td>
</tr>
<tr>
<td></td>
<td><code>x = y++</code></td>
<td><code>x = y++</code></td>
</tr>
<tr>
<td></td>
<td><code>if(x&gt;32) {</code></td>
<td><code>if(x&gt;32) {</code></td>
</tr>
<tr>
<td></td>
<td><code>   y--;</code></td>
<td><code>   y--;</code></td>
</tr>
<tr>
<td></td>
<td><code>}</code></td>
<td><code>}</code></td>
</tr>
<tr>
<td></td>
<td><code>else y=0;</code></td>
<td><code>else y=0;</code></td>
</tr>
</tbody>
</table>
Type 2 – Simple substitution

Model

Template

Generated code

Mapping

attribute=value

attribute

value
Type 2 – Simple substitution

Example

Input model

Template

Generated code

s = “abc”
x = y++
if(x>?VAL_1?){
y--;
} else y=0;
s = “abc”
x = y++
if(x>32){
y--;
}
else y=0;
Type 4 – Repetition

Mapping

Template

Generated code
Type 4 – Repetition

Template
Generated code
Mapping

Example

Input model
A=3

Template
!repeat A times!
s = “abc”
x = y++
if(x>32) y--;
!end repeat!

Generated code
s = “abc”
ex = y++
if(x>32) y--;
s = “abc”
ex = y++
if(x>32) y--;
s = “abc”
ex = y++
if(x>32) y--;
The implementation

- Code annotations

15. int someCode = 0;
16. // Java comments that delineate
17. if (someCode == someValue) {
18.    methodThatDoesNothing();
19. }
20. this.is = justASample();
21. // the boundaries of each mapping type
22. int restOfCode = 1;
The implementation

- We used JET’s custom tags
- We created our own version of most JET tags

- They do the same things, but with a side effect...
  ... the generated code is annotated according to each mapping type

1 Java Emitter Templates
An example

Type 1 mapping

```html
<template language="m:copy">
  s = "abc"
  x = y++
  if(x>32) {
    y--;
  }
  else y=0;
</template>
```

Generated code

```html
// T1:temp11.jet#1
s = "abc"
x = y++
if(x>32) {
  y--;
}
else y=0;
// T1:temp11.jet#1
```
And another

Type 4 mapping

```
<table>
<thead>
<tr>
<th>templ2.jet</th>
<th>Generated code</th>
</tr>
</thead>
<tbody>
<tr>
<td>s = “abc”</td>
<td>s = “abc”</td>
</tr>
<tr>
<td>&lt;m:iterate&gt; x = y++ print(s+”:+x);</td>
<td>// T4:templ2.jet#1 x = y++ print(s+”:+x);</td>
</tr>
<tr>
<td>&lt;/m:iterate&gt; x = 0</td>
<td>// T4:i=1 x = y++ print(s+”:+x);</td>
</tr>
<tr>
<td></td>
<td>// T4:i=2 x = y++ print(s+”:+x);</td>
</tr>
<tr>
<td></td>
<td>// T4:i=3 x = y++ print(s+”:+x);</td>
</tr>
<tr>
<td></td>
<td>// T4:templ2.jet#1 x = 0</td>
</tr>
</tbody>
</table>
```
The strategy

New/Tested \rightarrow \text{Synchronized} \quad \text{Initial code migration}

\text{Synchronized} \rightarrow \text{Unsynchronized}

\text{Unsynchronized} \rightarrow \text{Changes made on the IR/templates} \rightarrow \text{Code migration} \rightarrow \text{Synchronized}
The strategy

Here we replace the original reference implementation with an automatically annotated one!
Current status

We are currently **finishing** these 7 mapping types

We are also implementing automated text analyzers to perform the **evaluation**
Future work

Case study
Correctness
Completeness

Do the new tags correctly generate code?

Could there be other mapping types?
Future work

How to detect these changes?

New/Tested

- Initial code migration

Synchronized

- Code migration

- Changes made on the IR/templates

Unsynchronized

How to migrate these changes?
Some conclusions

• We will probably not be able to achieve full automation

• But any help is welcome!
  – We envision a **partially** automated process (and partially guided)

• If we can reduce the 20%–25% extra effort spent in code migration
  – A nice *contribution* will be made
Thanks!

Mapping Code Generation Templates to a Reference Implementation
Towards Automatic Code Migration
Daniel Lucrédio and Renata Pontin de Mattos Fortes

This work was partially supported by the National Institute of Science and Technology for Software Engineering (INES - www.ines.org.br), funded by CNPq and FACEPE, grants 573964/2008-4 and APQ-1037-1.03/08.