

# Templates for Event-B Code Generation

Andy Edmunds  
University of Southampton  
[ae2@ecs.soton.ac.uk](mailto:ae2@ecs.soton.ac.uk)

# Code Generation with Tasking Event-B

- Tasking Event – B is
  - an extension to Event-B.
  - flow control language and annotations.
- Tasking Machines (1) map to task implementations.
- Shared Machines map to protected objects,
  - provide monitor-style protection.
- Environ Machines (2) map to tasks for simulation.

# Code Generation with Tasking Event-B

- Tasking/Environ machines have 'Task Bodies'
  - to describe program flow.
  - which map to program statements.
- Program flow such as,
  - IF event1 ELSE event2 END
  - event1 ; event2
- Events 'populate' sequences, branches, loops, update statements, procedures, procedure calls.

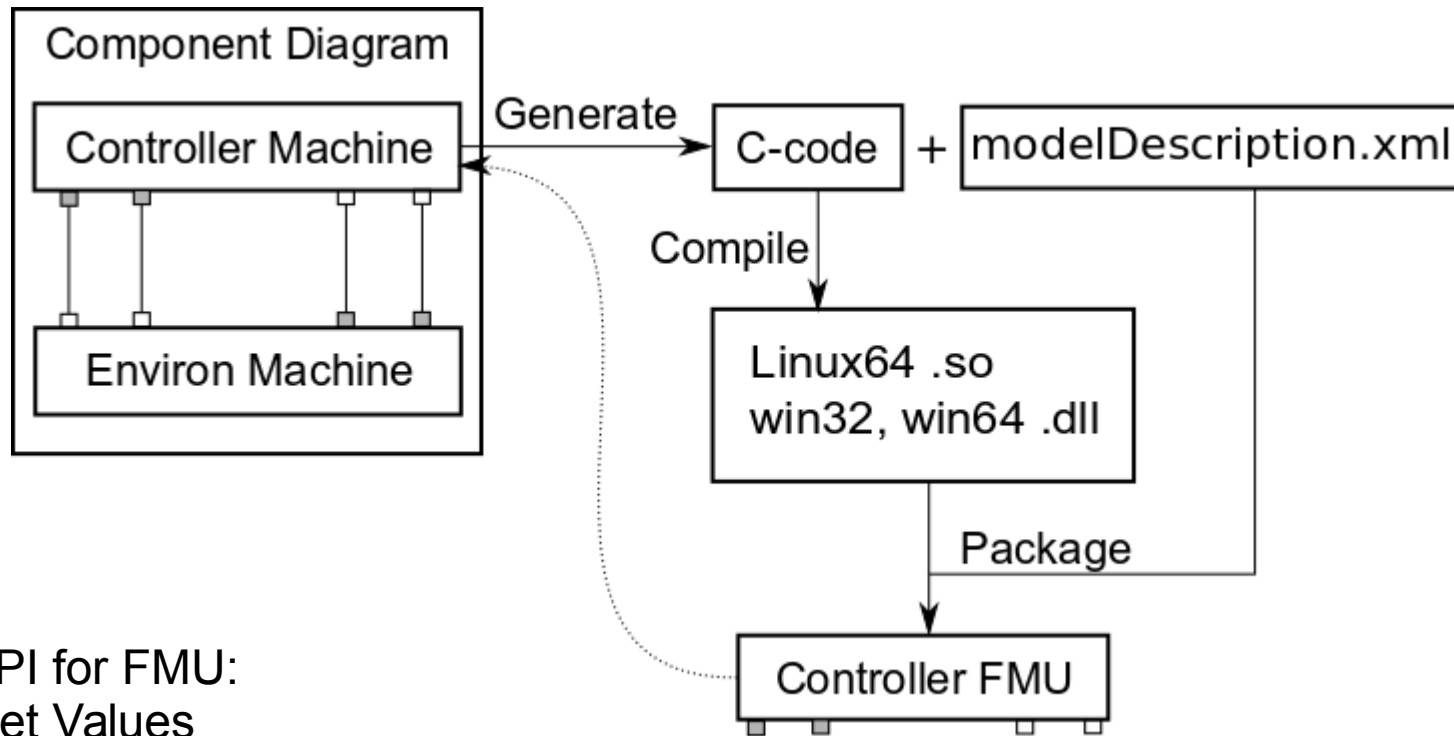
# Why Templates?

- Configuration of code generation **targets**.
  - 'Templates' can be used for configuration,
  - and avoids hard-coding in the translators.
  - Templates are re-usable.
  - But, Acceleo / JET etc. – more than we need?
- Reuse of the generated code,
  - The same Event-B model can be used to generate simulation code and deployable code.
- Used in the EU FP7 **Advance** project for FMI-C code generation.

# Co-simulation with FMI

- Master and Slaves communicate through API.
  - Enables Discrete/Continuous Simulation.
  - Slaves are FMUs.
  - The master is cyclic; slaves are initialized, then does simulate-update cycle.
- We can generate an FMU from a machine.
  - But, **tasking machines map to protected objects**,
  - ... because of the 'hidden' master.
- Simulation uses a 'component diagram'.
  - We can replace the Event-B Machine in a diagram, with an FMU and simulate/test with executable code.

# FMUs from Machines



API for FMU:

Get Values

Set Values

Query Status

Instantiation

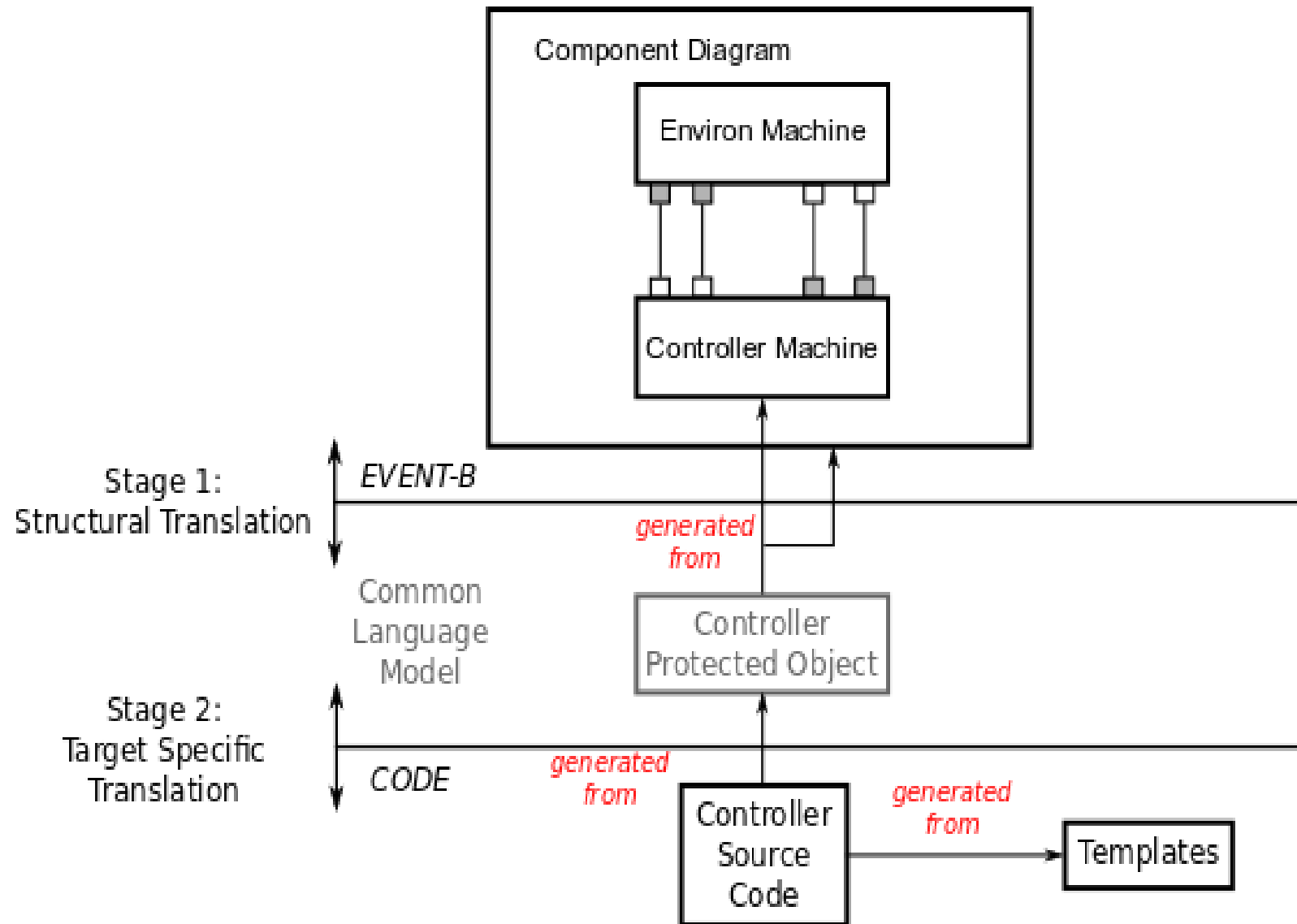
Initialisation

Simulation Step

# Using Templates for FMI-C

- The initial idea: Some code needs configuration, depending on the target;
  - e.g. FMI life-cycle functions.
  - but doesn't need to be modelled formally.
  - and is re-usable.
- The code generated from Event-B models should be the 'critical' code.
  - Can be sent to different targets.
- Some merging of the two is needed.

# Translation Stages





# Template Tags

- Tags facilitate:
  - code injection points, and use of generators.
  - further template expansion.
  - production of meta-data, using generator.
- The notation:
  - `///name`  
where *name* identifies a template or generator name.
- Generators are stored internally in a map of name to class.

# An Example Template

```
///  
## <addToHeader>
```

```
fmiStatus fmiInitializeSlave(fmiComponent c,  
    fmiReal tStart, fmiBoolean StopTimeDefined,  
    fmiReal tStop) {
```

```
    ModellInstance* comp = (ModellInstance*) c;
```

```
    ///  
    ## <initialisationsList>
```

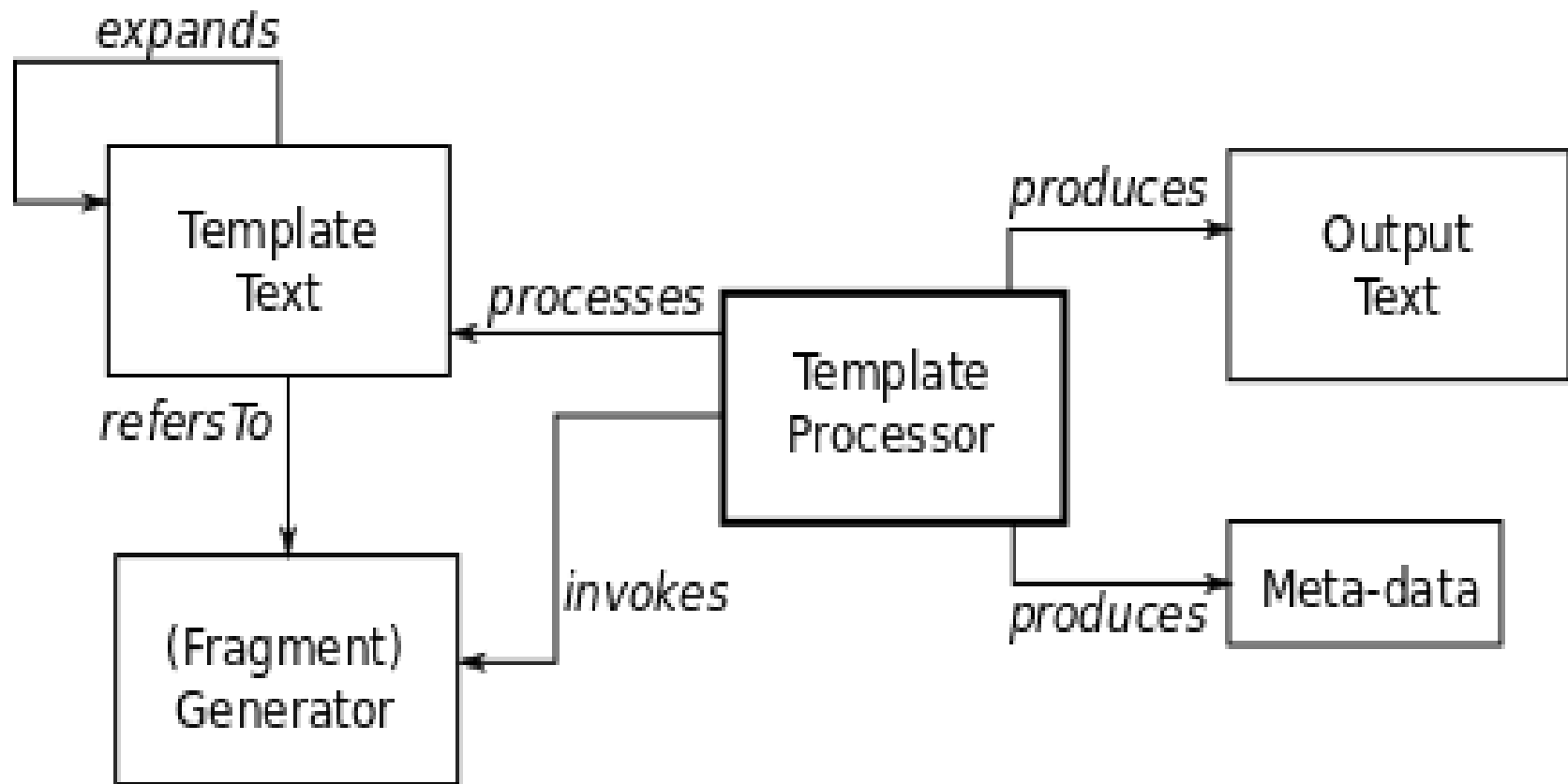
```
    ///  
    ## <stateMachineProgramCounterIni>
```

```
    return fmiOK;
```

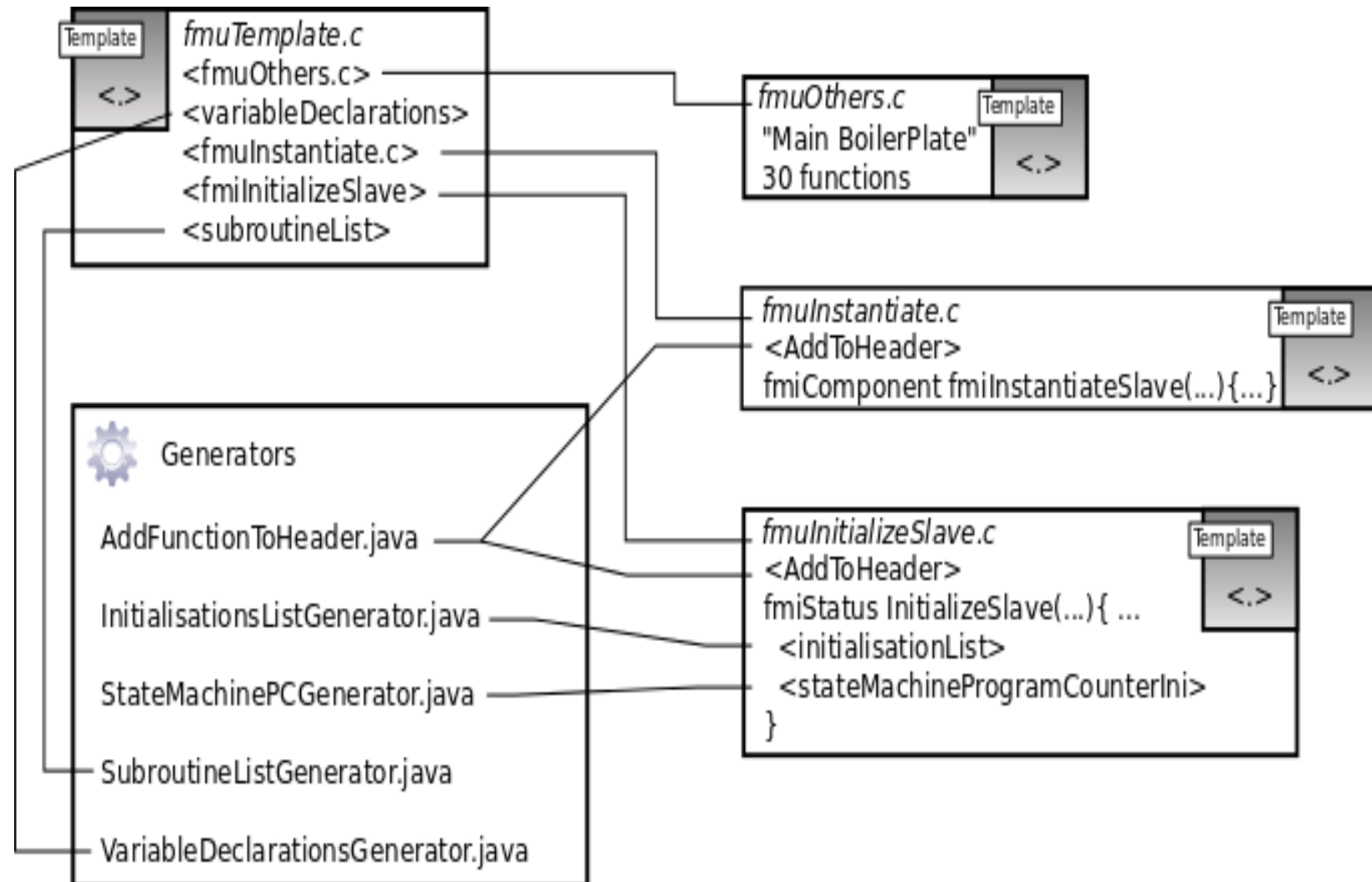
```
}
```

Tags are 'processed'.  
Other lines are output verbatim.

# Template Processor Architecture



# Templates and Generators



# Extension Points

- Extend IGenerator to provide a new generator,
  - use the extension point
    - *org.eventb.codegen.templates.generator*.
  - then implement the method,
    - `public List<String> generate(IGeneratorData data).`
    - and supply any necessary data using *IGeneratorData*.
- A *Tag*'s characters can be specified; so that it matches the comment characters of a target language.

# Translating...

- TemplateHelper.generate(“fmilInitialiseSlave”)
  - finds the generator, and calls it.
  - For fmilInitialiseSlave it does the following,

```
public List<String> generate(IGeneratorData data){
```

```
    //(1) Un-pack the generator data.
```

```
    //(2) for each protected object...
```

```
        translate each variable declaration/initialisation.
```

```
    //(3) Return the new code listing.
```

```
}
```

# Resulting Code

```
fmiStatus fmiInitializeSlave(fmiComponent c,  
    FmiReal tStart, fmiBoolean StopTimeDefined,  
    fmiReal tStop) {  
    ModelInstance* mc = (ModelInstance*) c;  
    // Generated By InitialisationsListGenerator  
    mc->i[c_level_ControllerImpl_] = 100;  
    mc->b[c_pumpOnReq_ControllerImpl_] = fmiFalse;  
    ...  
    return fmiOK;  
}
```

# Summary

- Templates could be used to replace much of the hard-coding that exists in the current translators.
- A GUI could assist with target configuration, and produce a template.
- The principle may be of use to others working on similar activities.
- Maybe of use with product-lines?