Templates for Event-B Code Generation

Andy Edmunds
University of Southampton
ae2@ecs.soton.ac.uk

Why?

- Configuration of code generation targets.
  - Templates may be configurable,
  - and re-usable
  - Acceleo/ JET – more than we need?
- Reuse of generated code,
  - The same Event-B model can be used to generate simulation code and deployable code.
- A chance to try it out on the Advance project.

Code Generation 1

- Tasking Machines (1) map to task implementations.
- Environ Machines (2) map to tasks for simulation.
- Shared Machines map to protected objects.

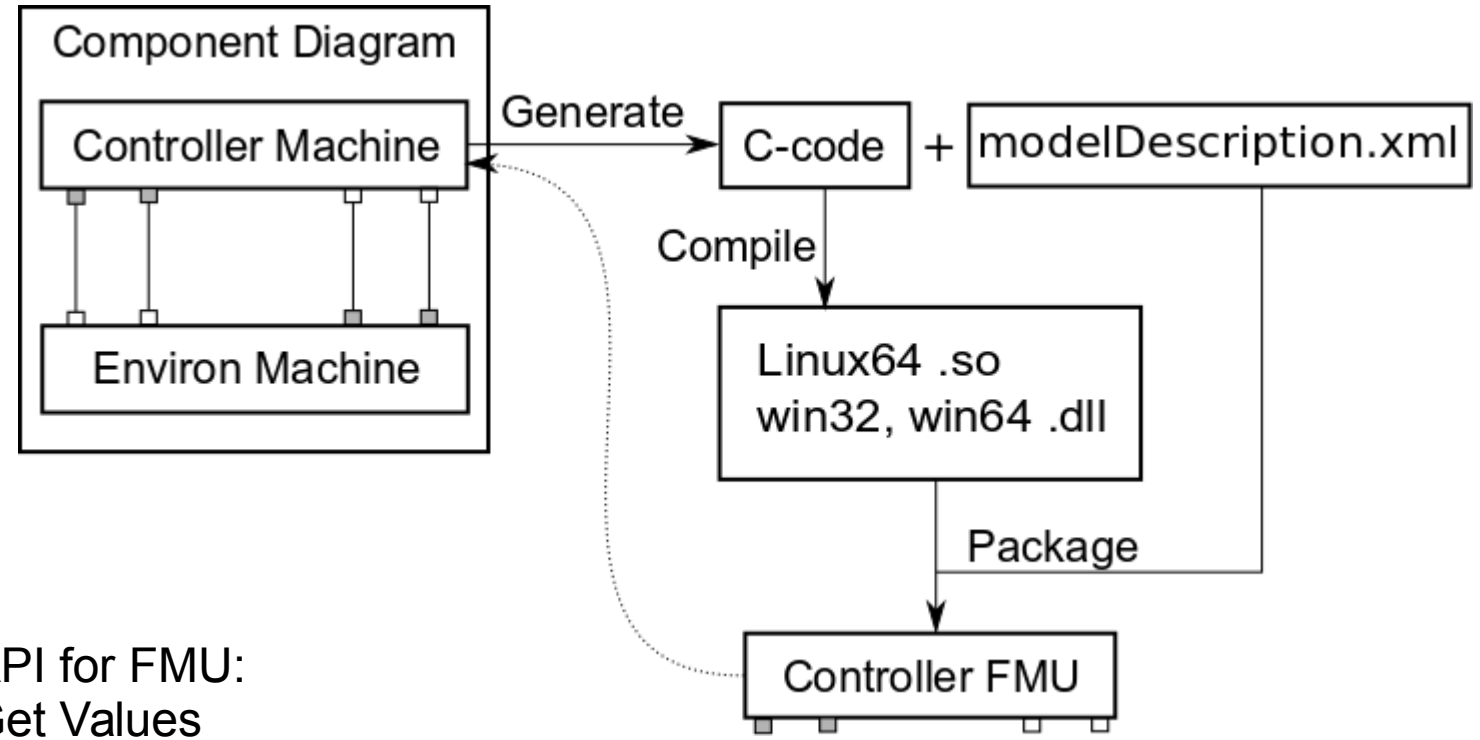
- Task Bodies in 1 and 2 map to program statements.
  - IF event1 ELSE event2 END
  - event1 ; event2

- Events 'populate' sequences, branches, loops, actions, procedures, procedure calls.

Co-simulation with FMI

- Master and Slaves communicate through API.
  - Slaves are FMUs.
  - The master is cyclic; slaves are initialized,
    - … then master 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

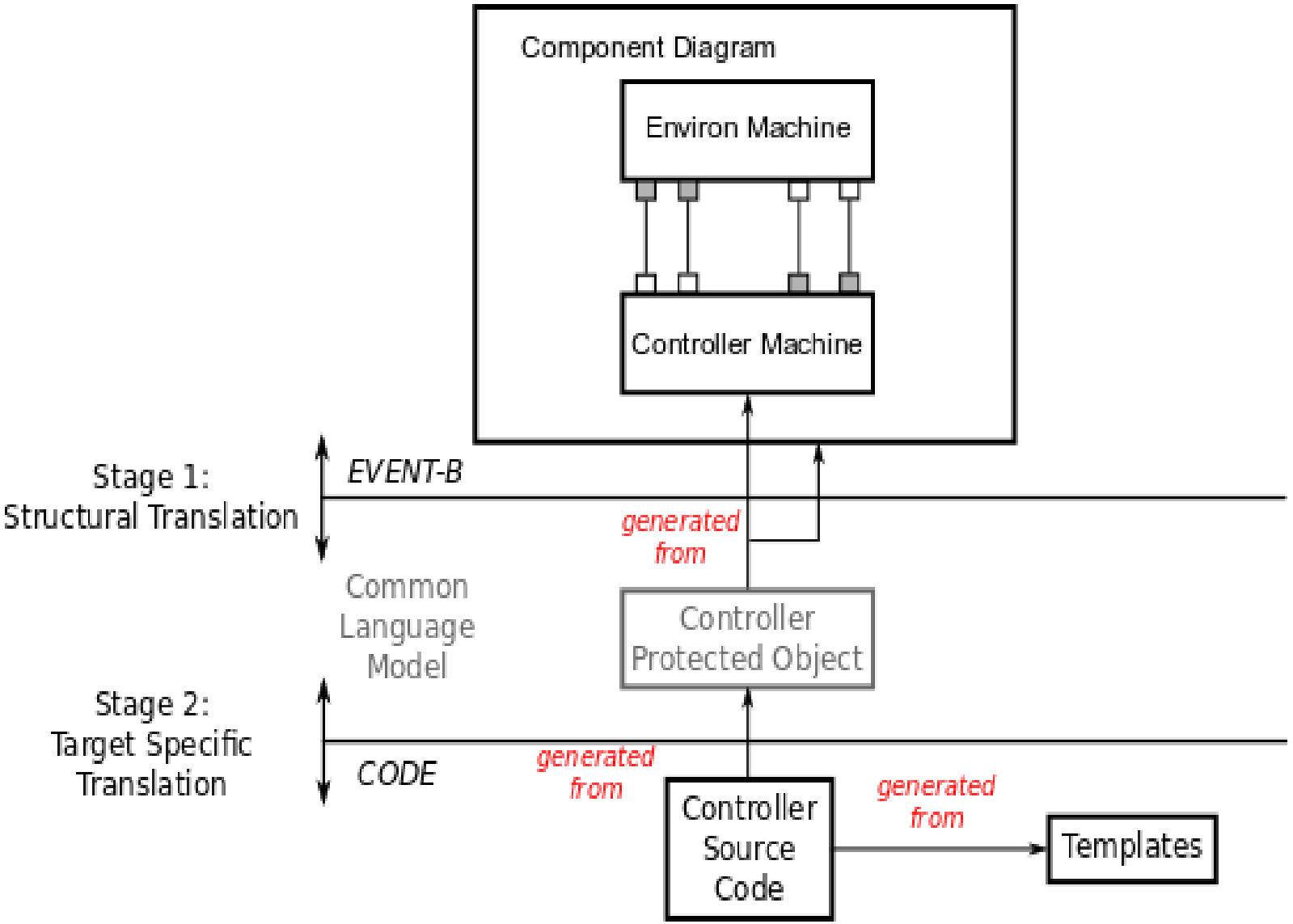


API for FMU:
Get Values
Set Values
Query Status
Instantiation
Initialisation
Simulation Step

How do we use Templates?

- 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.

FMI With Templates



Template Tags

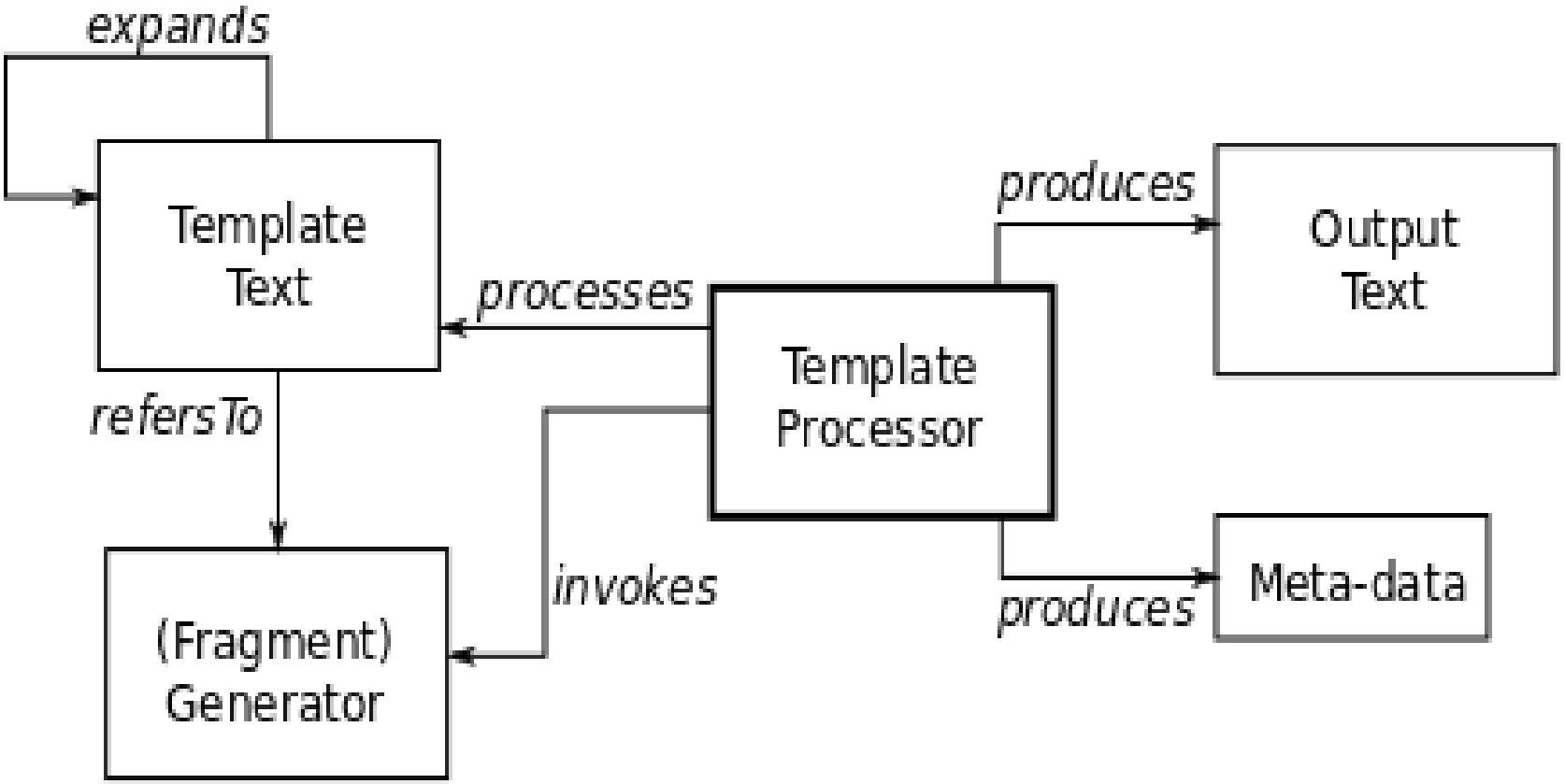
- Tags facilitate:
  - code injection points, and use generators.
  - further template expansion.
  - production of meta-data, using generator.

- The notation:
  - *//## <name>*

    where *name* identifies a template or generator name

- Generators are stored internally in a map of name to class.

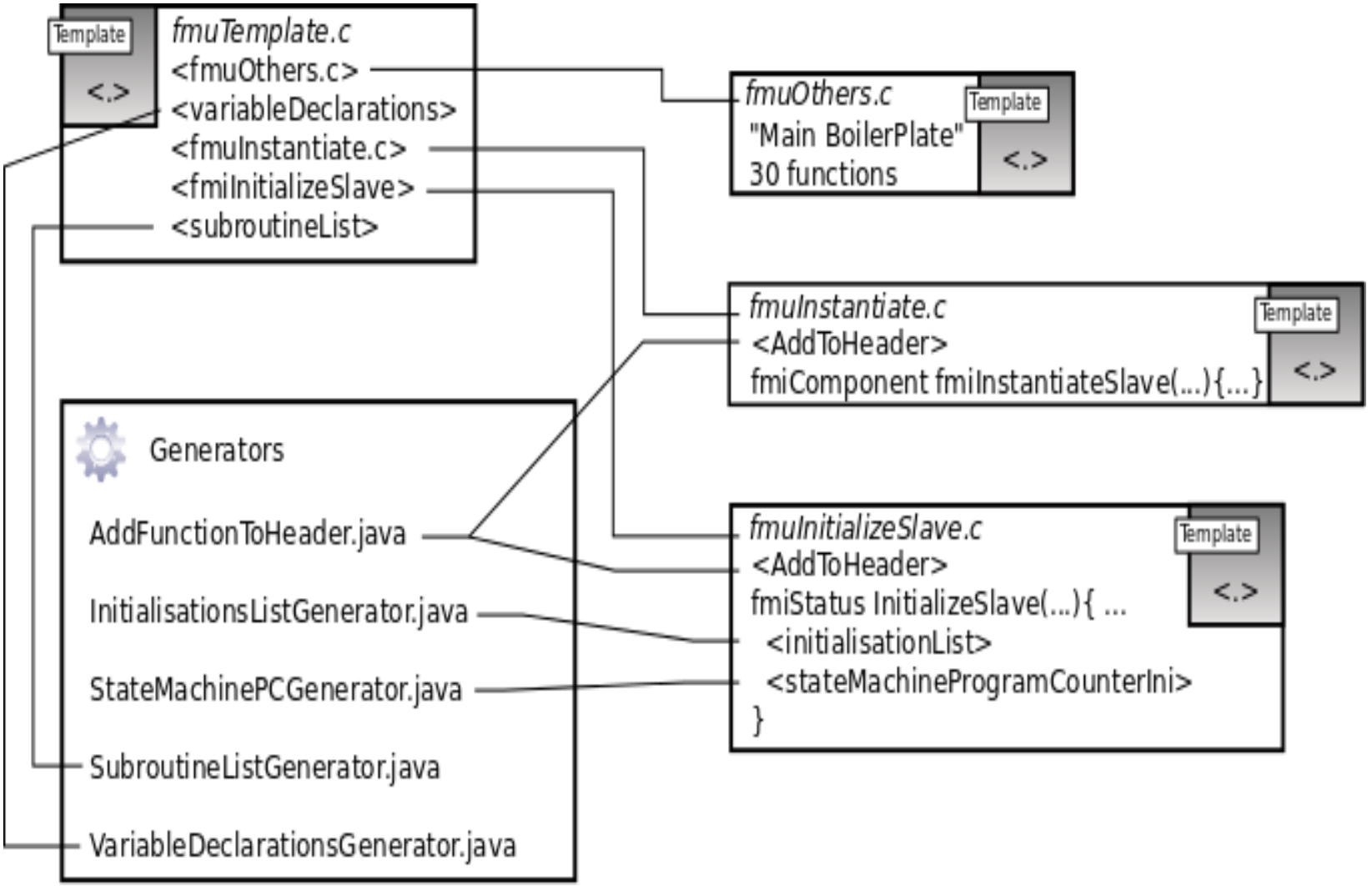
Template Processor Architecture



An Example Templates

```
//## <addToHeader>
fmiStatus fmiInitializeSlave(fmiComponent c,
        fmiReal tStart, fmiBoolean StopTimeDefined,
        fmiReal tStop) {
    ModelInstance* comp = (ModelInstance*) c;
    //## <initialisationsList>
    //## <stateMachineProgramCounterIni>
    return fmiOK;
}
```

The Other Templates



The IGenerator Interface

- To provide a new generator,
  - use the extension point
    - *org.eventb.codegen.templates.generator,*
  - Implement
    - public List<String> generate(IGeneratorData data)
    - Supply any necessary data using *IGeneratorData*
- A Tag string can be specified, so that it matches comments in a target language.

Translating

- TemplateHelper.generate(“fmiInitialiseSlave”)
  - finds the generator and calls it.

```
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;
}
```