APP4MC Tools: EMF based Model Transformation Framework

APP4MC Tools : EMF-based Model Transformation Framework

How is it different ??

Yet another model transformation framework (for M2M and M2T)

How is this different from other frameworks like ATL, QVT, Xpand, Xtend etc.,

This framework acts like a wrapper around model transformation technologies (*like Xtend2*)and provides the complete infrastructure for easily specifying meta-models, hooking loaders for the models, caching mechanism, defining transformation code, building update sites or command line products and testing of the transformation code.

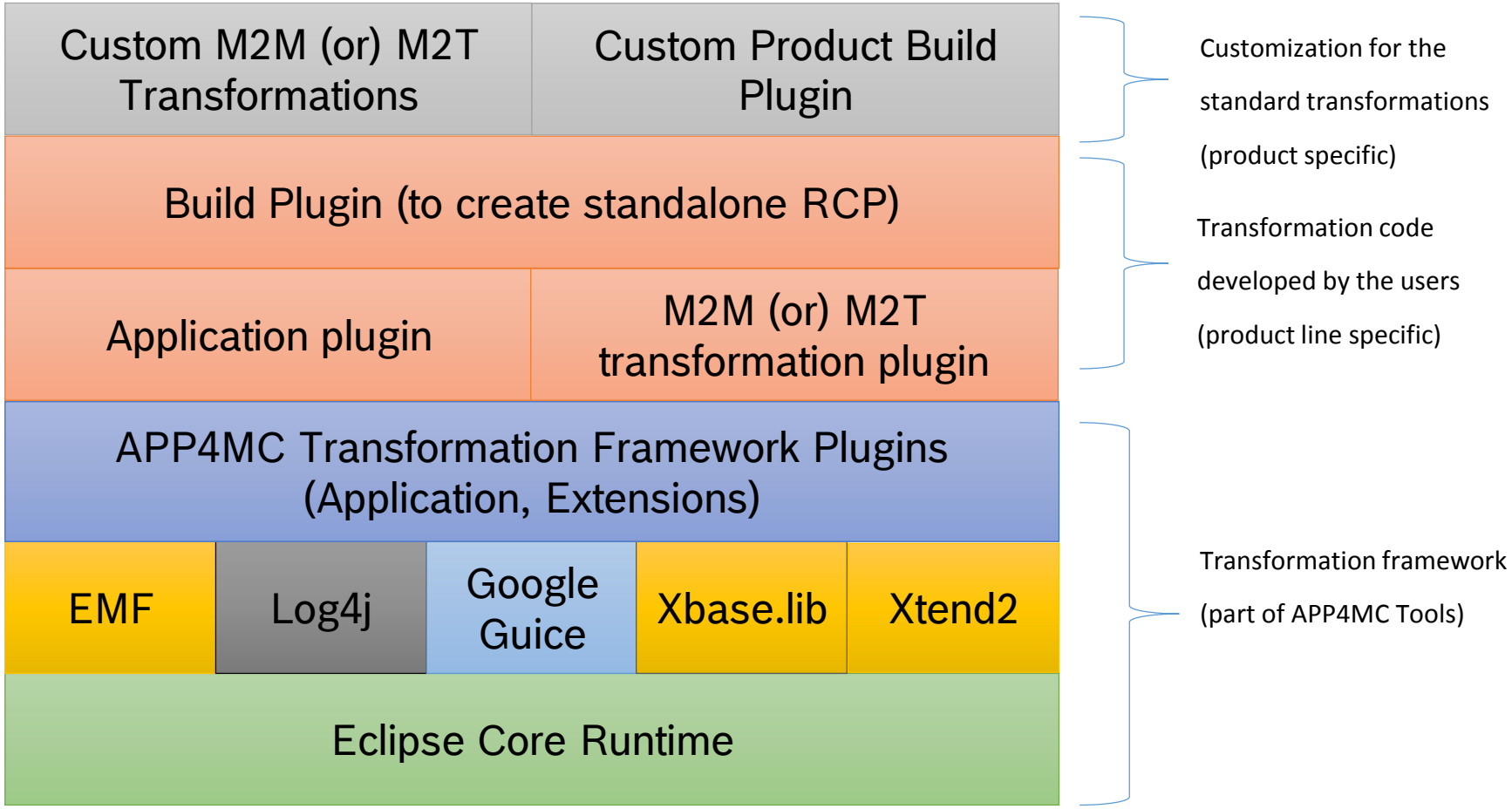
EMF-based M2M Transformation Framework

Features

- Generic framework (based on Eclipse extension mechanism): Easy to plug and play with meta models to be transformed
- Usage of Google Guice to easily inject bindings of Transformation classes
- Usage of Xtend2 as the core technology for transformation (both M2M and M2T) [java like syntax]
- Framework provides cache objects across each Transformation class, to easily exchange the data
- Easy to create standalone RCP product of the transformation and use it as a command line application (or) create a eclipse update site which can be added to an eclipse IDE
- Maven build infrastructure to create the update site, standalone products for the base transformation and customized transformations
- Easy to extend the base transformation by providing new transformation classes and binding them using Google Guice.
- Creating custom products with the available Maven build infrastructure

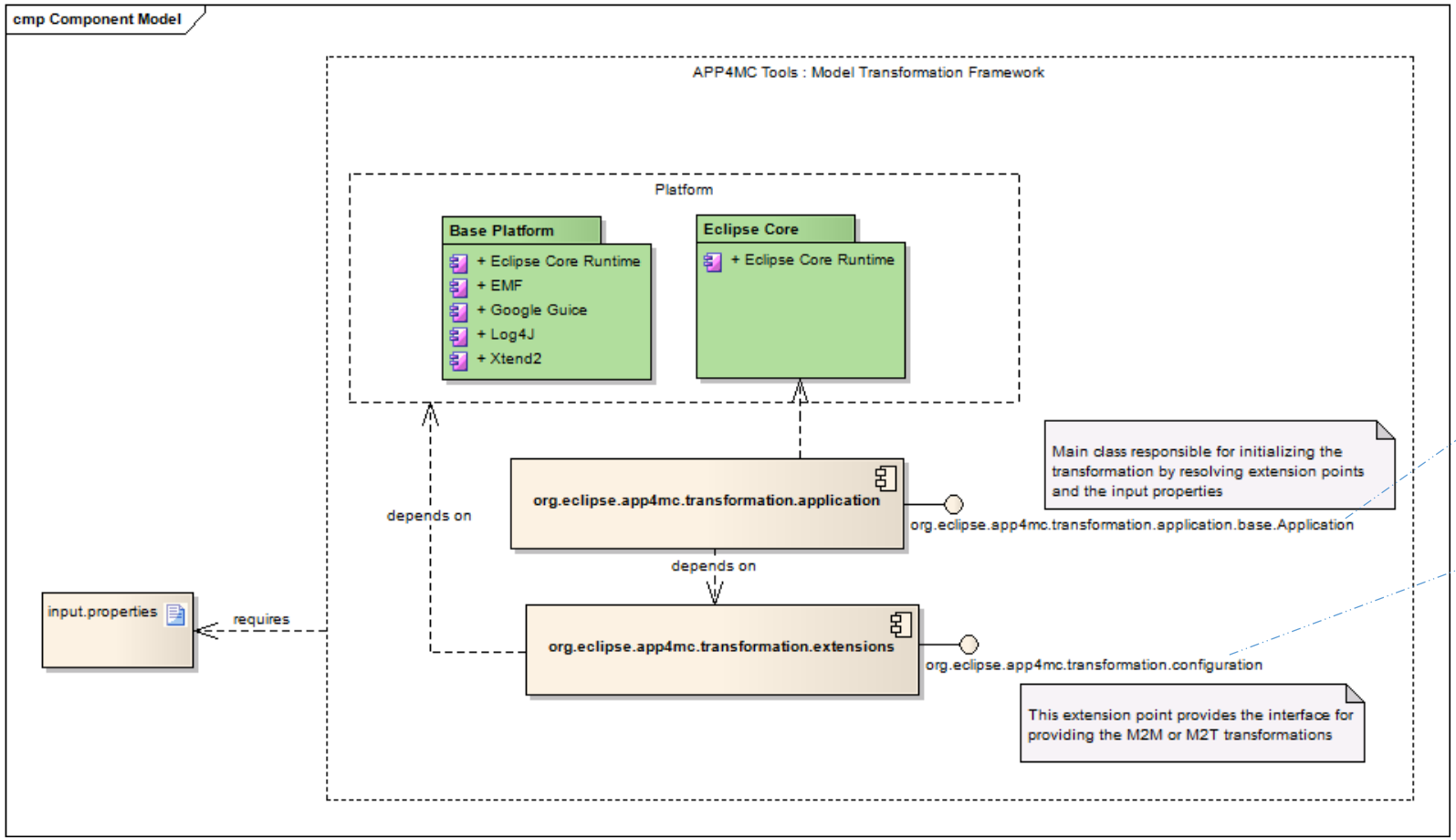
EMF-based M2M Transformation Framework

Architecture



EMF-based M2M Transformation Framework

High Level Design

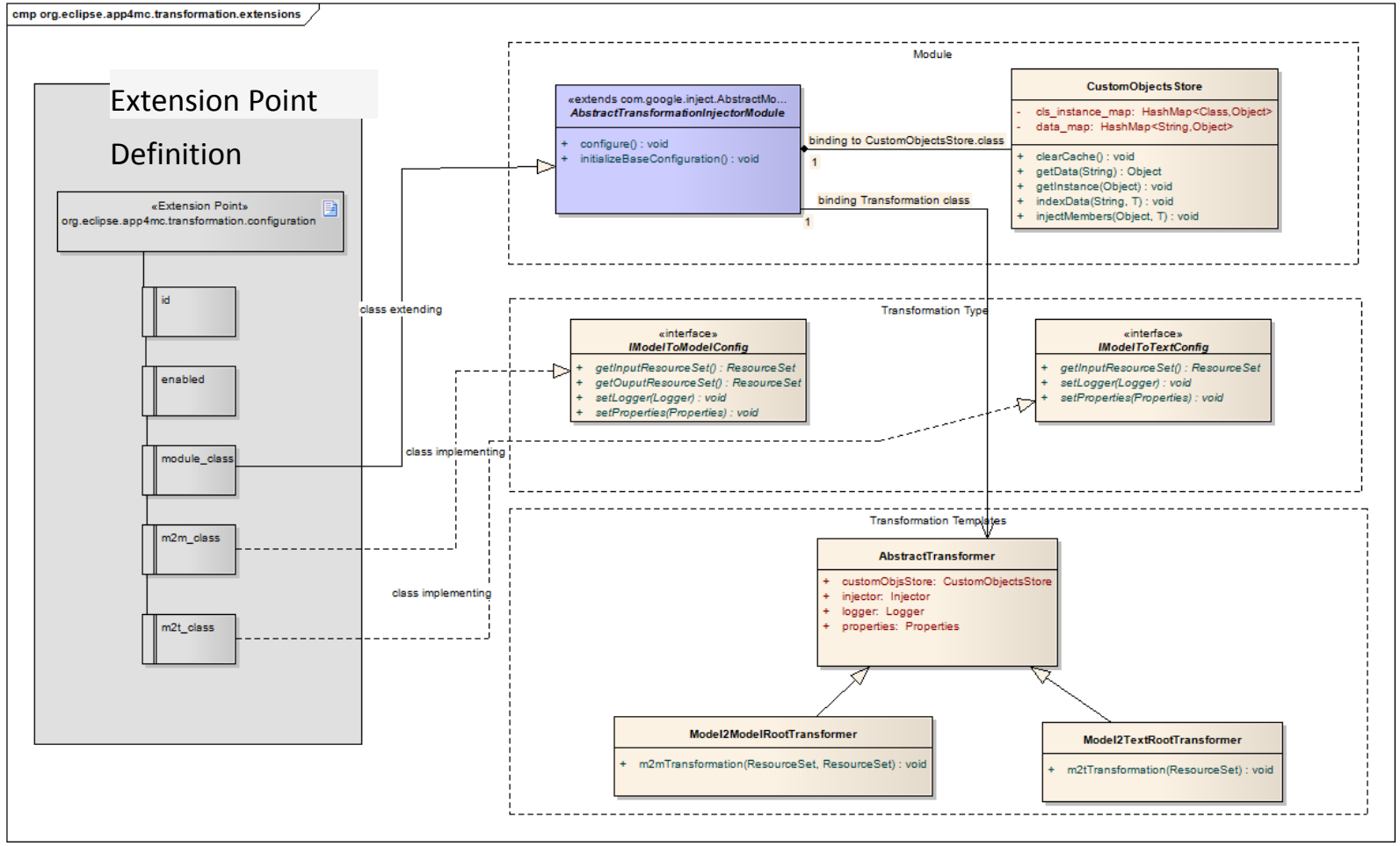


Java Class to be extended for developing new application

Eclipse Extension Point

EMF-based M2M Transformation Framework

Low Level Design : org.eclipse.app4mc.transformation.extensions



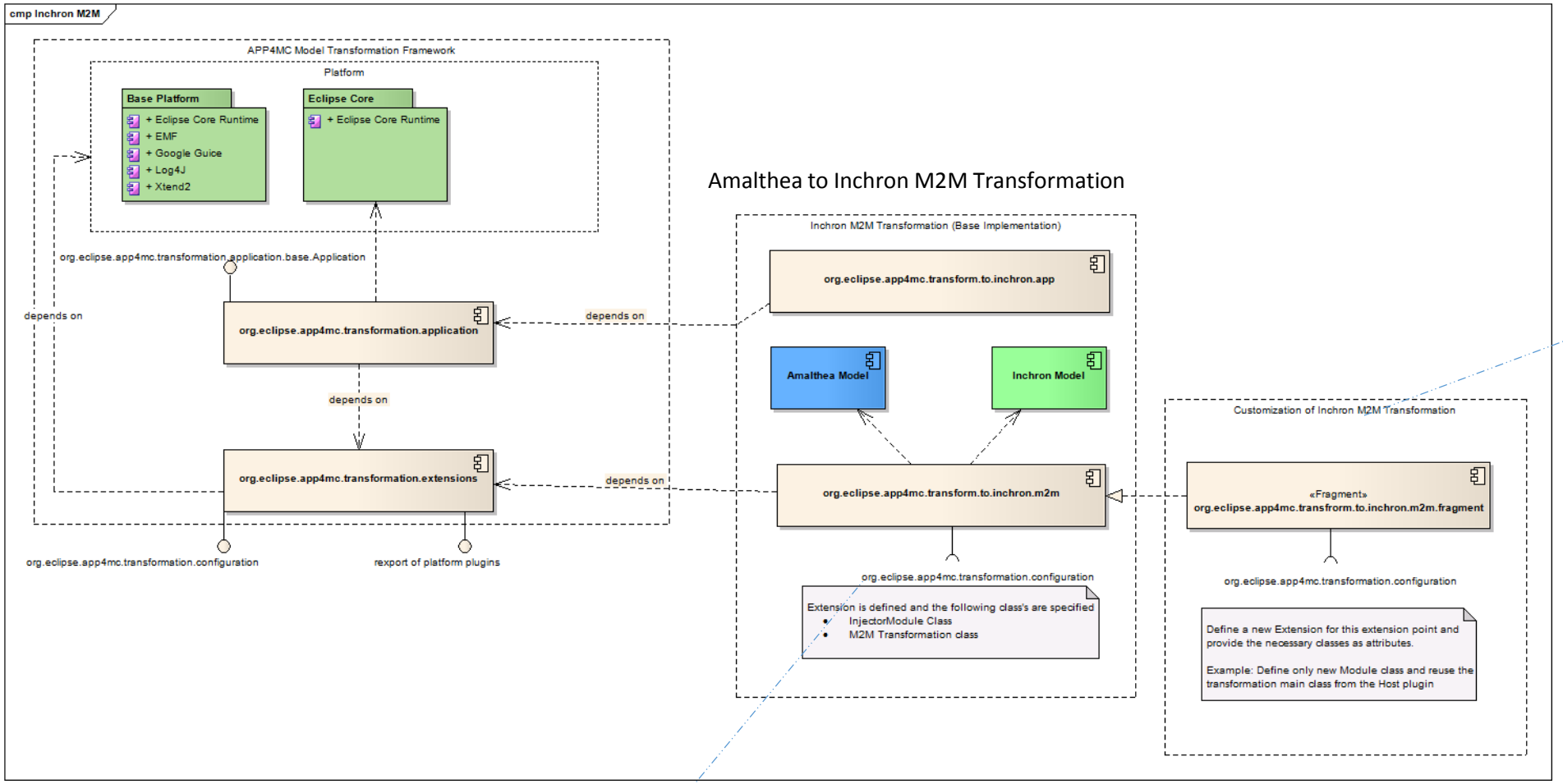
Classes required for specifying the Google Guice Module and binding Cache object and Transformer class

Configuration classes: containing code for defining both input resourceset and output resourceset

“Xtend classes” required for transformation

EMF-based M2M Transformation Framework

Inchron Model Transformation



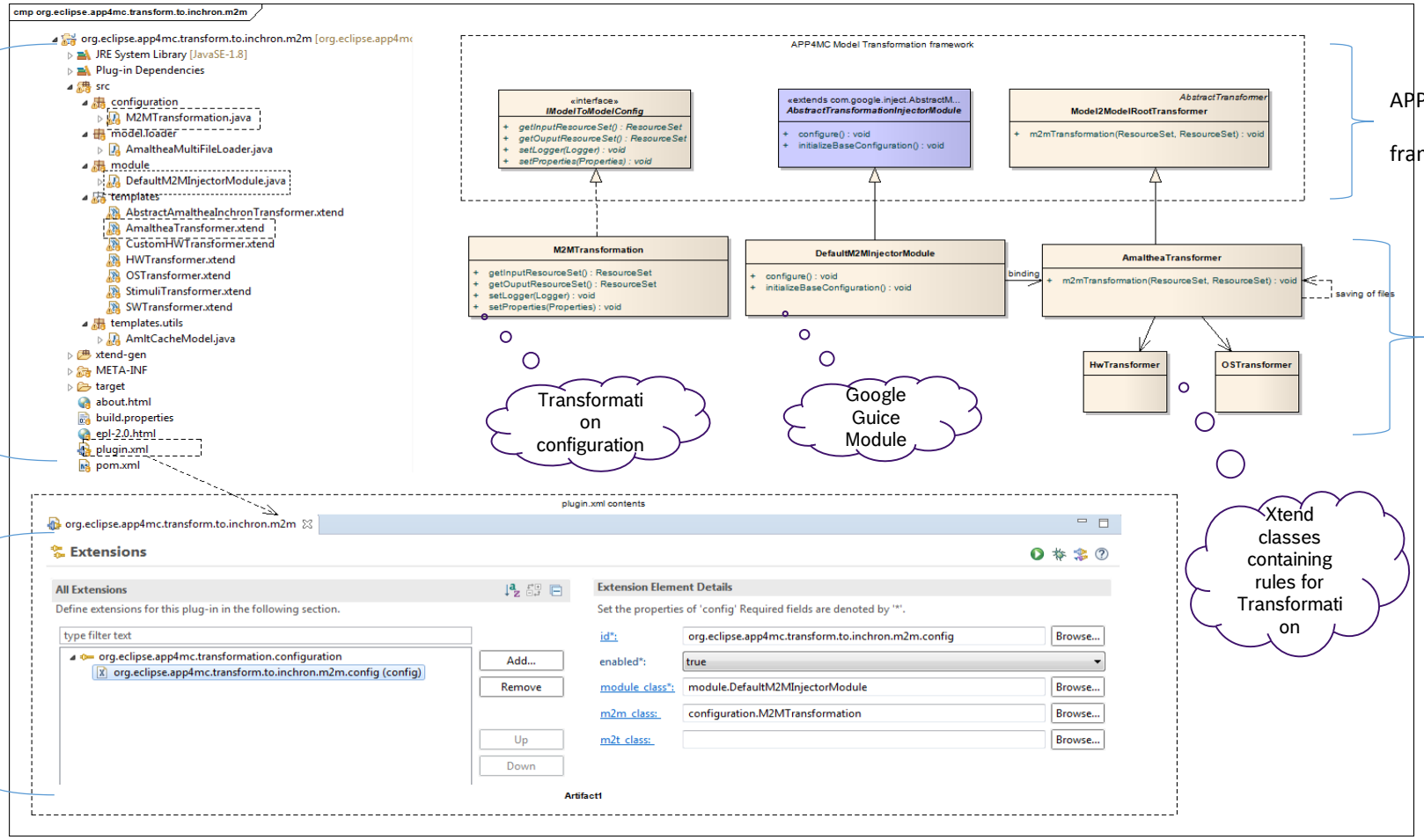
Customization of Amalthea to Inchron M2M Transformation based on specific requirements

Extension point for Model Transformation

EMF-based M2M Transformation Framework

Inchron Model Transformation

Contents of M2M plugin (Amalthea to Inchron)



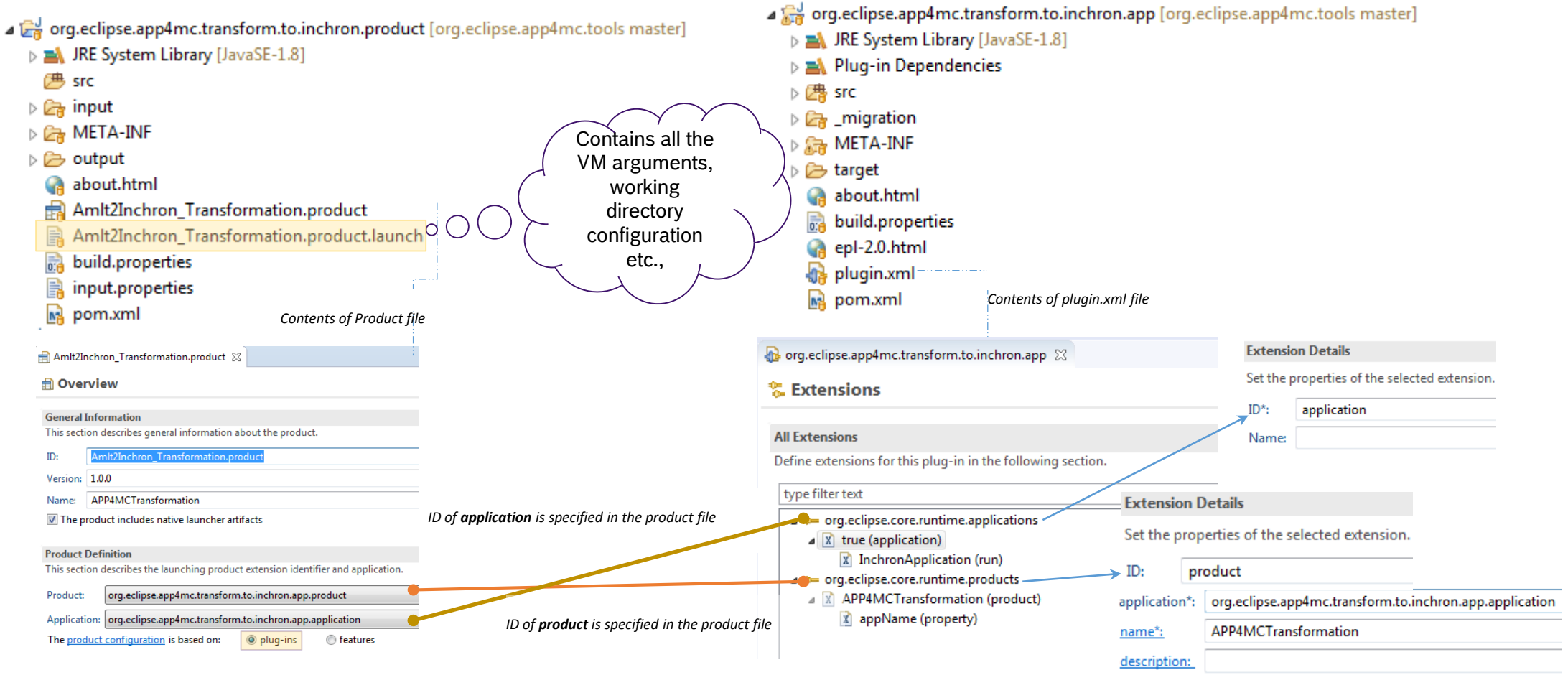
APP4MC Model transformation framework

Classes responsible for M2M transformation from Amalthea to Inchron model

Plugin.xml of M2M plugin

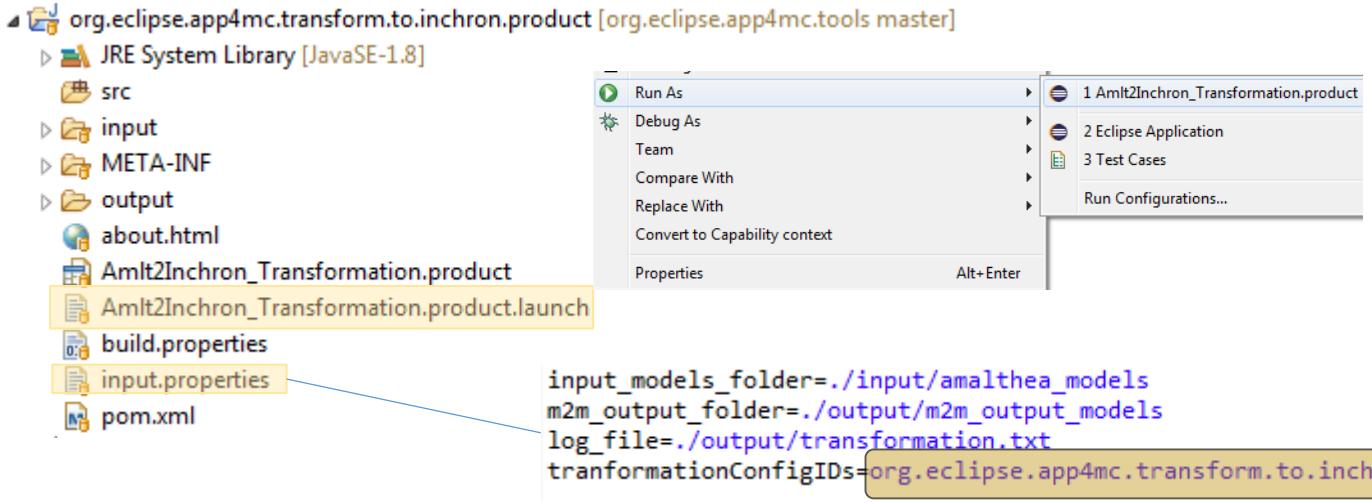
EMF-based M2M Transformation Framework

Inchron Model Transformation : Product definition

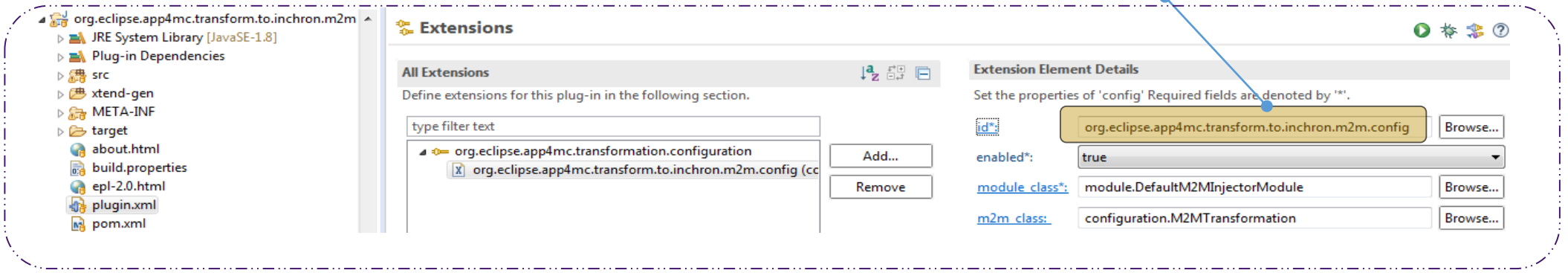


EMF-based M2M Transformation Framework

Inchron Model Transformation : Execution



Note: In case of **change in location of input.properties file**, working directory location should be updated in launch configuration of the Amlt2Inchron_Transformation



Backup slides

EMF-based M2M Transformation
Xtend

- Easy to read, developed on Java source code
- Compiles Java code, can be called from Java
  - Code generation can be influenced with active expressions
  - Can run in command line
- Model transformation and code generation within the same template
- Supports overloading and overriding functions
- Include features: debug, call-hierarchy, name refactoring, profiling etc.
- JUnit test cases can be used (maybe interesting for certification)

EMF-based M2M Transformation
Google Guice

- Loose coupling to extend functionality based on injection mechanism
- It can dynamically bind a default transformation with the user specific transformation
- Can be easily integrated into Xtend
- Generic transformation should provide top level transformation where the user can
  - Provide all customized classes → injection (overriding) is done dynamically by Google Guice

Extensions mechanisms

EMF-based M2M Transformation
Example

- Label Access:
  - Generic: Calculate static latency → add to Runtime
  - ExtentionA: Labelaccess with certain tags behave differently → override labelTransformation Function
  - ExtentionB: Labelaccess calls TLM function in SystemC hardware model for Co-Simulation → override

Generic

```
….
Class transformClassRunnableItems(Task t) {
        ForAll Instructions do transformInstructions(instr);
        ForAll LabelAccess do transformLabelAccess(labelAcc);
        ….
        }
….
Class transformLabelAccess(labelAcc) {
                delay = calcLatency();
                addToDelay(delay);
}
```

ExtentionA:

```
Class extendA extending transformLabelAccess(labelAcc) {
                if (tag)
                    delay = …..;
                addToDelay(delay);

}
```

ExtentionB:

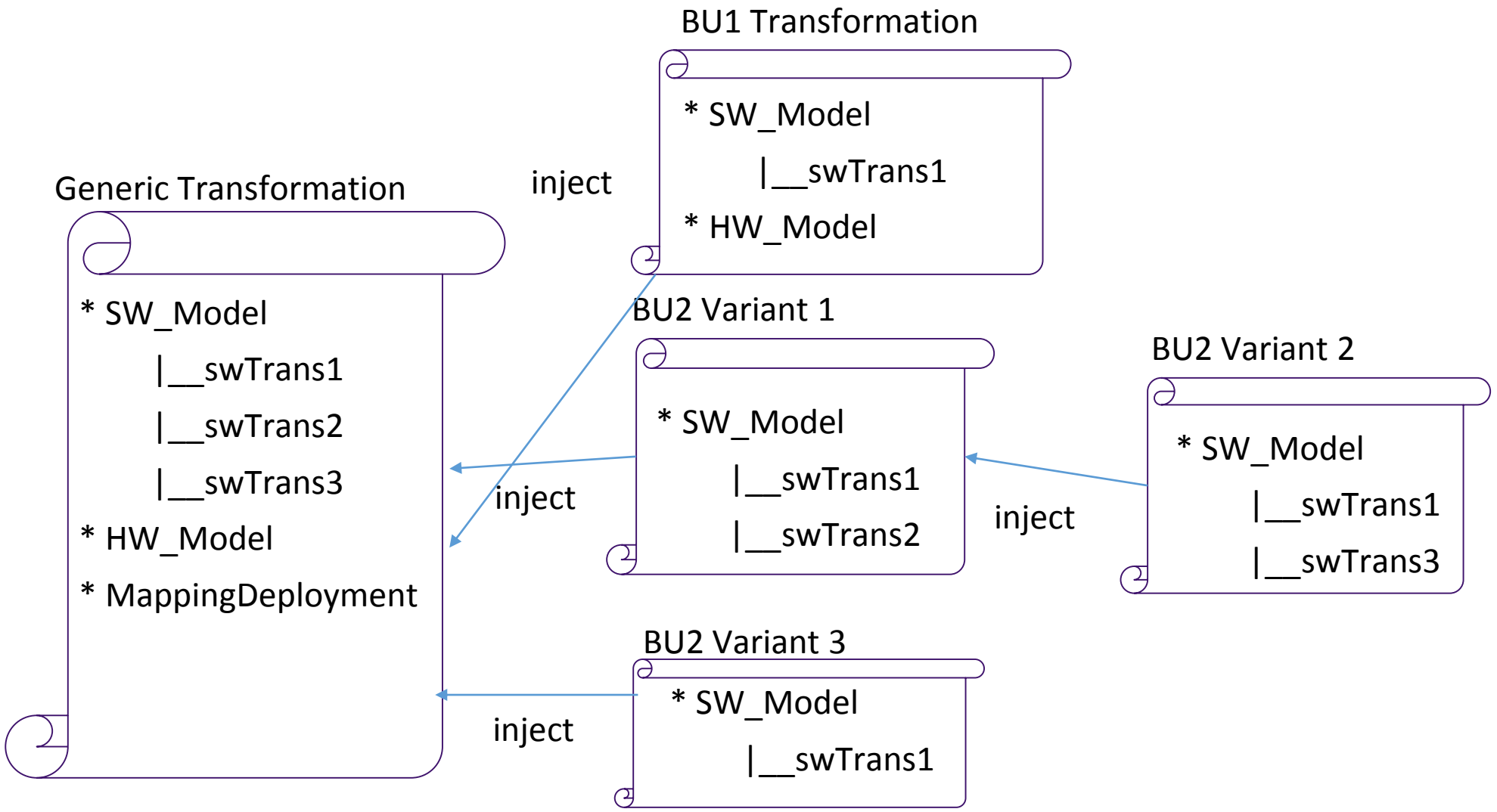
```
Class extendB extending transformLabelAccess(labelAcc) {
                generateITlmHardwareCall(core, mem);
}
```

ExtentionA:

```
startGenericTransformation(extendA, ….);
```

ExtentionB:

```
startGenericTransformation (extendB, …..);
```


BU1 Transformation
* SW_Model
|__swTrans1
* HW_Model
inject
Generic Transformation
* SW_Model
|__swTrans1
|__swTrans2
|__swTrans3
* HW_Model
* MappingDeployment
BU2 Variant 1
* SW_Model
|__swTrans1
|__swTrans2
inject
BU2 Variant 2
* SW_Model
|__swTrans1
|__swTrans3
inject
BU2 Variant 3
* SW_Model
|__swTrans1
inject

Thank you