mancoosi

managing software complexity

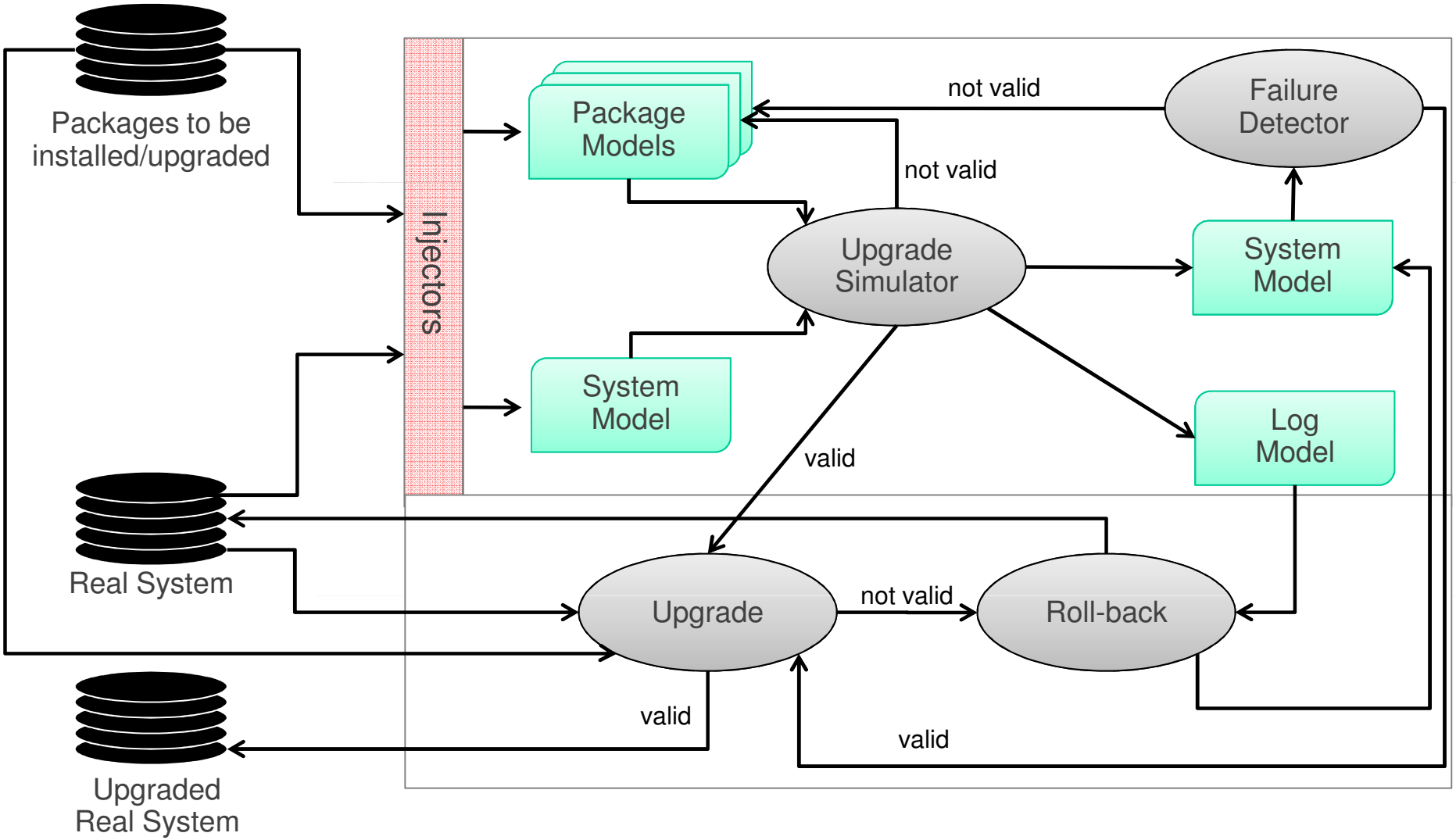
Instantiation of the Mancoosi Metamodel on a GNU/Linux distribution

Davide Di Ruscio – University of L'Aquila

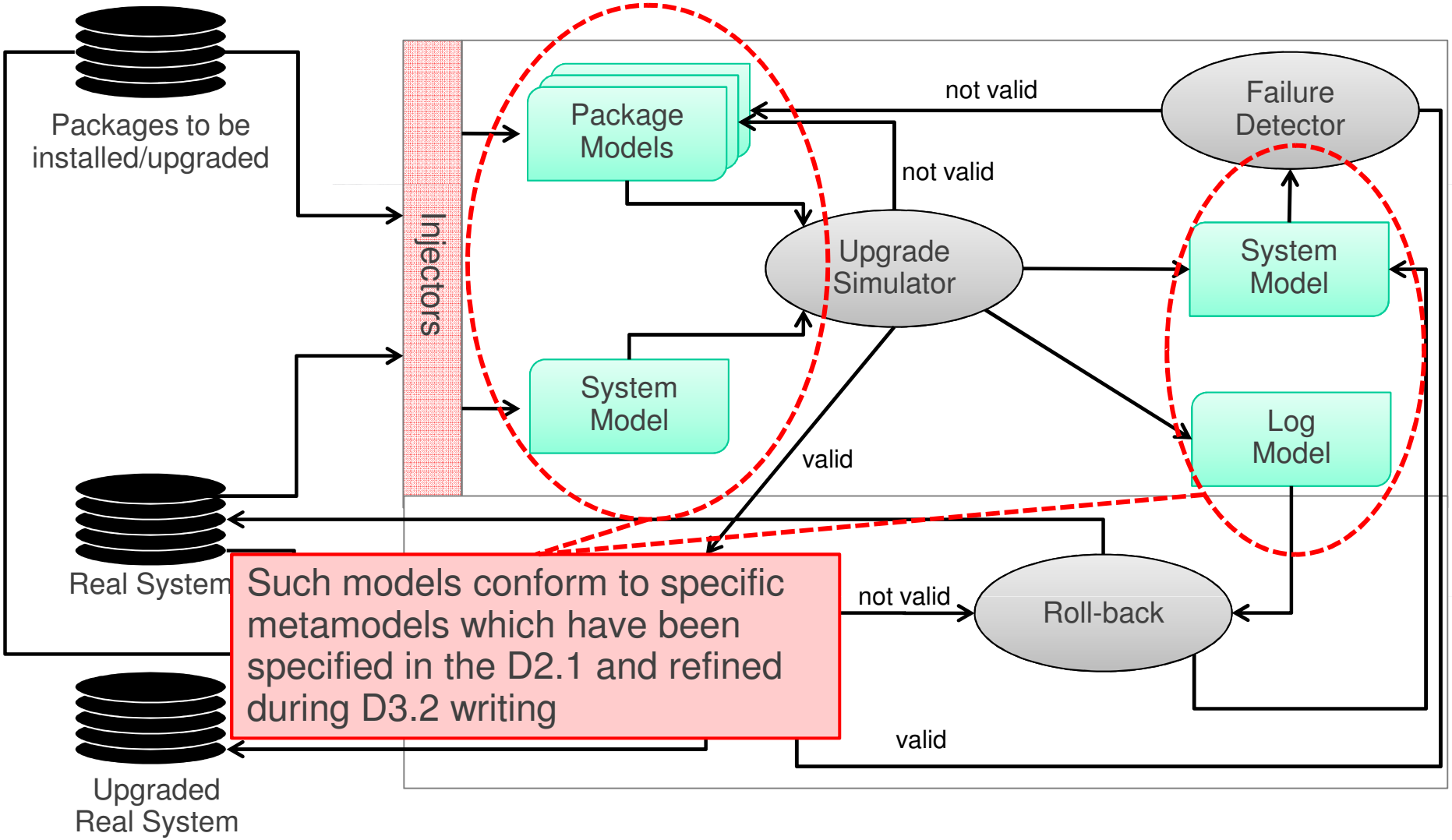
Outline

- Introduction
- Towards D2.2
- Model Injection
  - System Configuration
  - Maintainer scripts
- Demo
- Next steps

Introduction

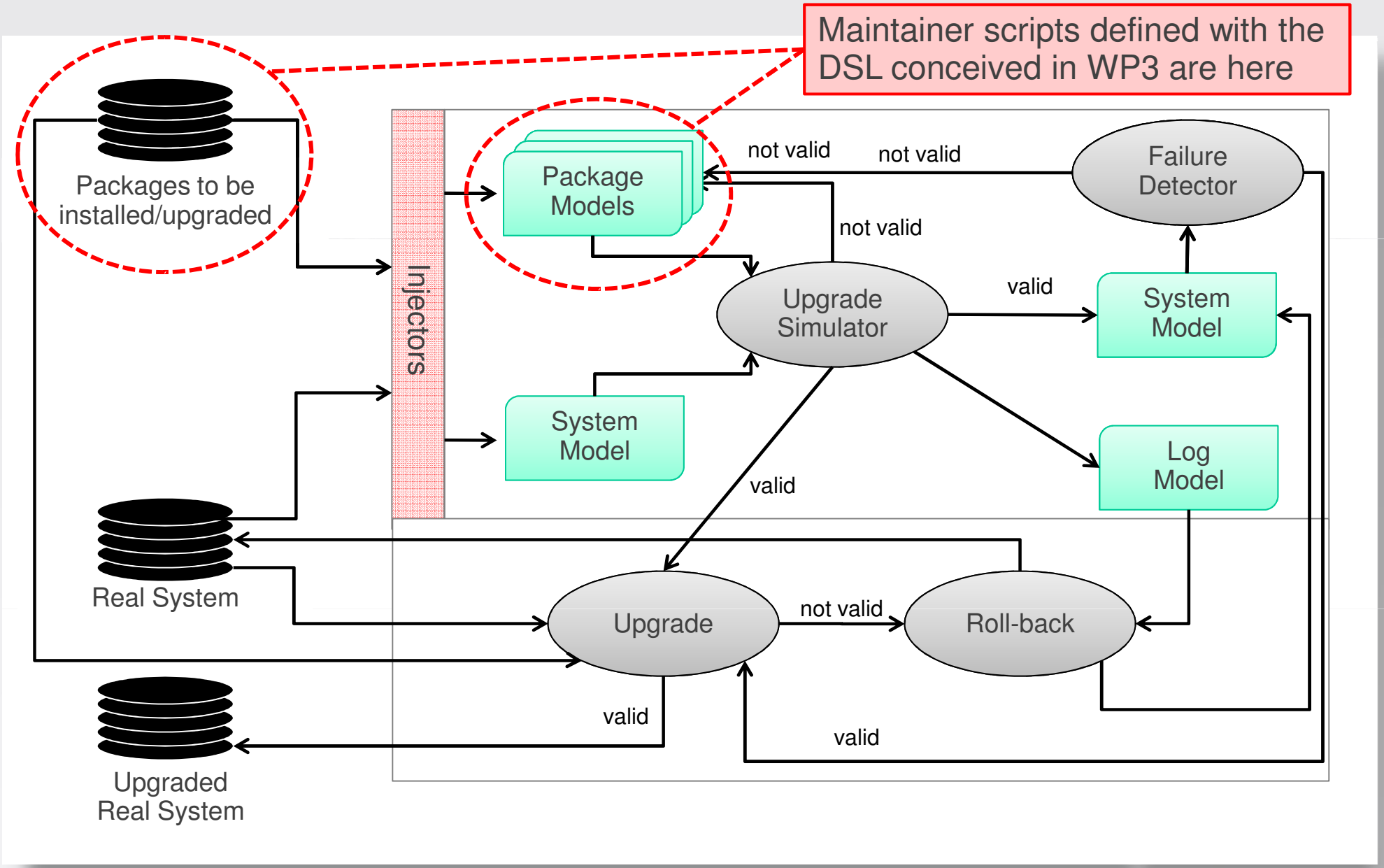


Introduction

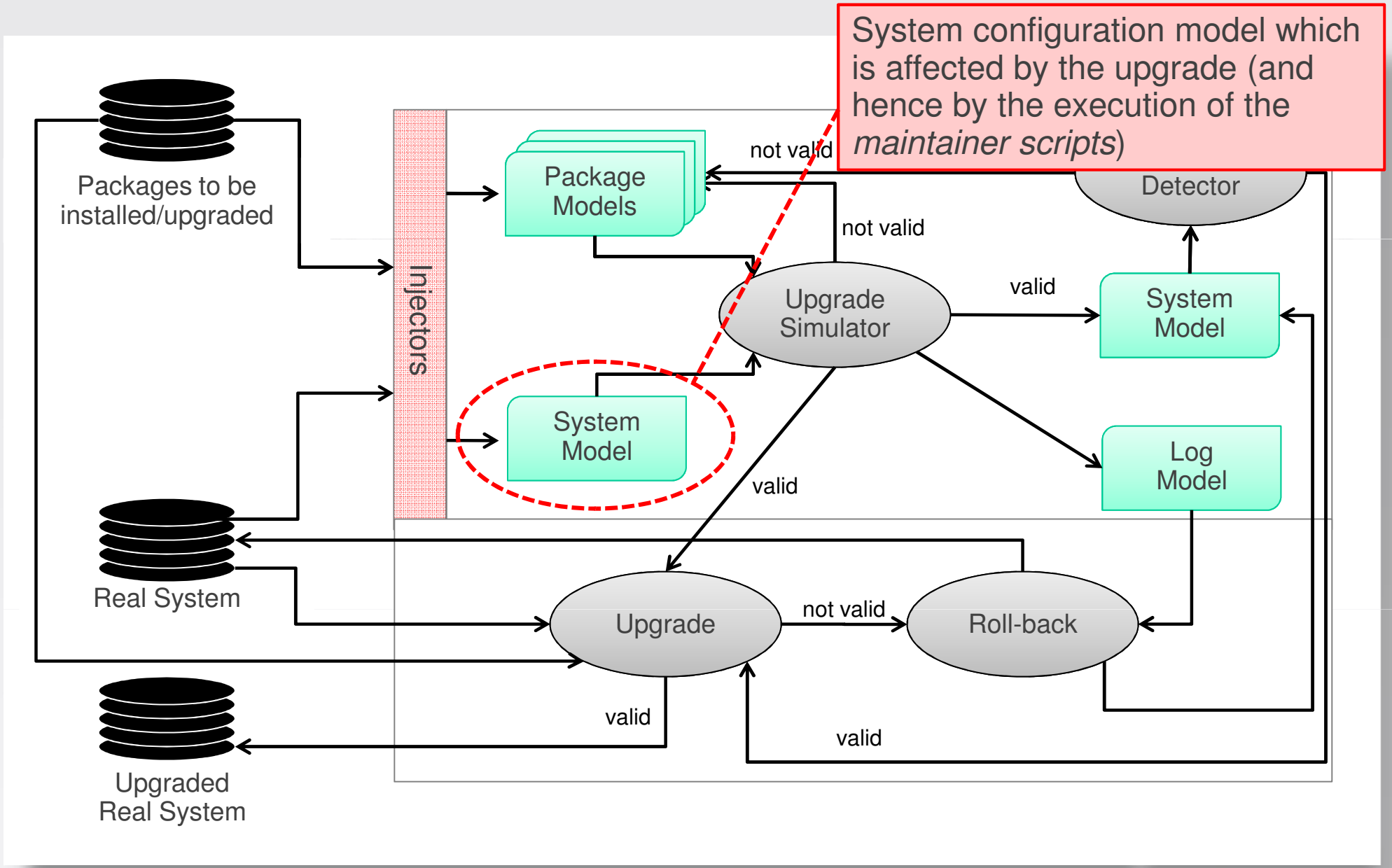


Such models conform to specific metamodels which have been specified in the D2.1 and refined during D3.2 writing

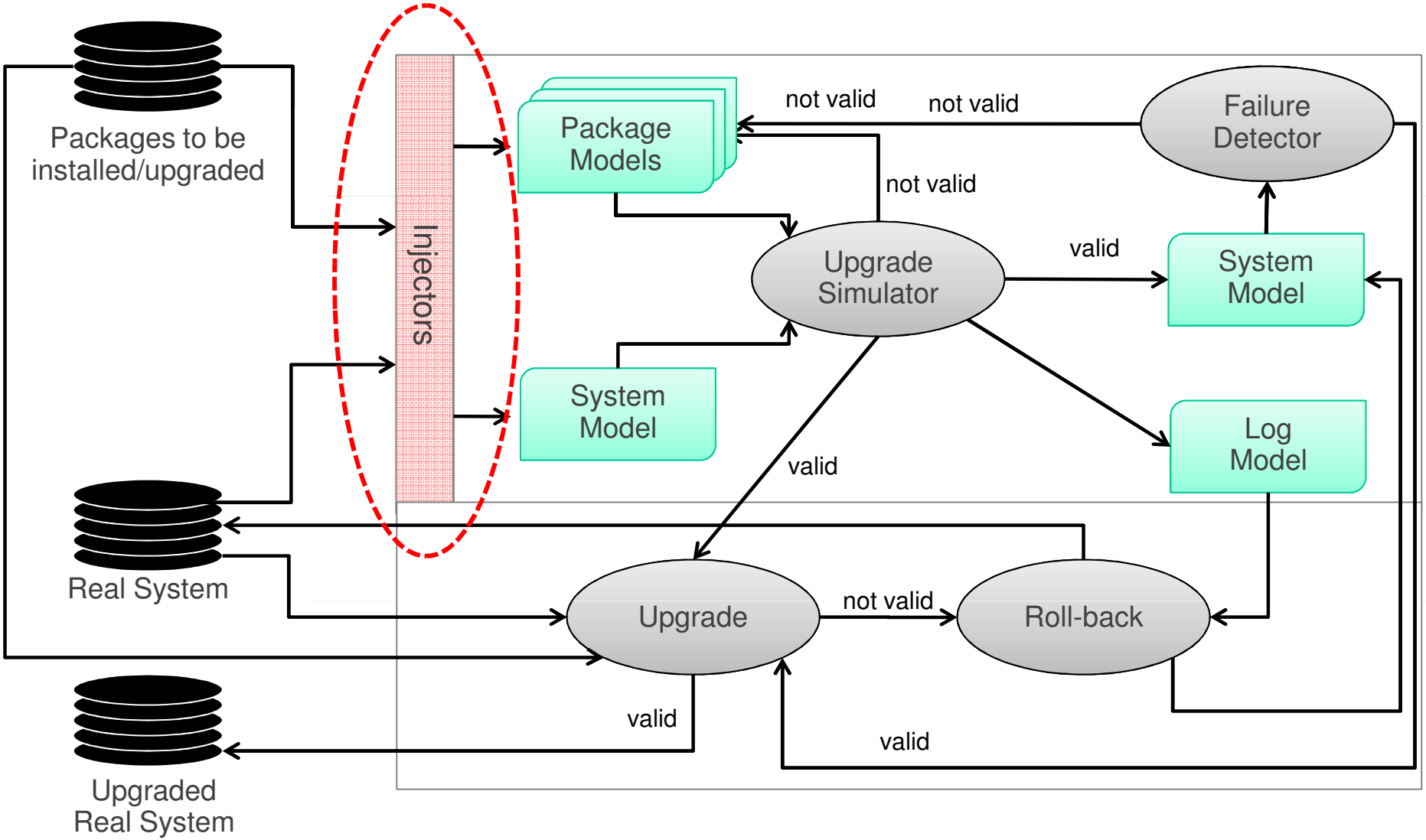
Introduction



Introduction



Introduction



Scenario

» Existing and running systems have to be specified in terms of models which describe system configurations

» Packages have to be represented in terms of models and their maintainer scripts have to be specified by using the DSL

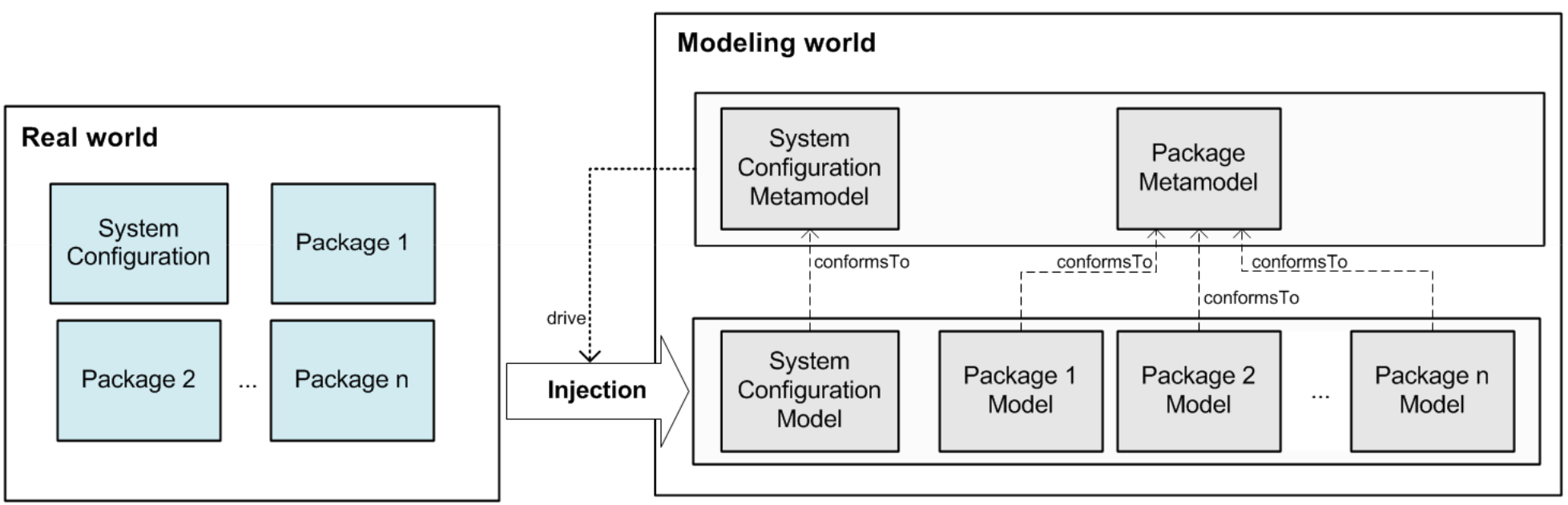
Towards D2.2

Title of D2.2: Instantiation of the metamodel on a widely-used GNU/Linux distribution

Due at: T0+24

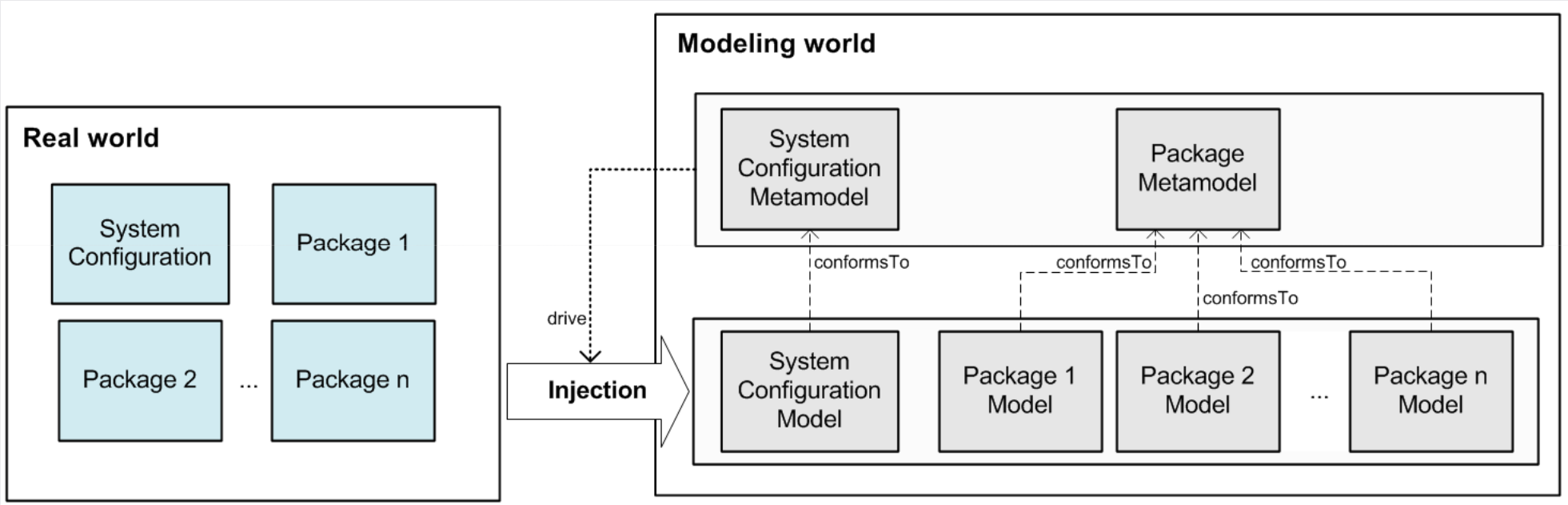
- The deliverable has to describe techniques and tools which have been adopted and developed to deal with the model injection
- Starting from an existing Linux installation, corresponding models conforming to the MANCOOSI metamodel (presented in D2.1 and refined during the D3.2 writing) have to be produced

Model Injection



- By means of the model injection, given a real software system a corresponding representation in the modeling world is obtained
- It relies on tools (injectors) that transform software artifacts into corresponding models
- The process is driven by the metamodels

Model Injection



» The elements which have to be *injected* are

- The configuration of the real system
- Packages to be installed (maintainer scripts included)

Model Injection: System Configuration

» Existing systems have to be “inspected” in order to generate corresponding models which are defined in terms of the following metaclasses (among others):

- *FileSystem*, to represent the file system by including all the files which build up the configuration (e.g. user files which do not compromise the systems are not taken into account)
- *Init*, used to model the typical `/etc/init.d` location and to maintain the services which have to be started when the system is booted
- *Service*, to model service which are running
- *Alternative*, to model all the existing alternatives. For instance, for the `java` alternative, all the installed versions of the java virtual machine are maintained

Model Injection: System Configuration

» Existing systems have to be “inspected” in order to generate corresponding models which are defined in terms of the following metaclasses (among others):

- *PackageSetting*, for each package a corresponding package setting element is available to refer to its configuration files
- *MimeTypes,* to model all the mime types and the corresponding handlers available in the considered system
- *SharedLibrary*, to model the shared libraries
- *Module*, to represent all the kernel modules
- *User*, to model all the users of the considered system
- *Group*, to model all the groups of the considered system
- …

Model Injection: System Configuration

- The generation of system configuration models from existing systems is performed programmatically by using Java and the Eclipse Modeling Framework
- Specific shell commands (like *dpkg-query, ps,* etc.) are invoked by ad-hoc Java programs which parse their results and opportunely create modeling elements

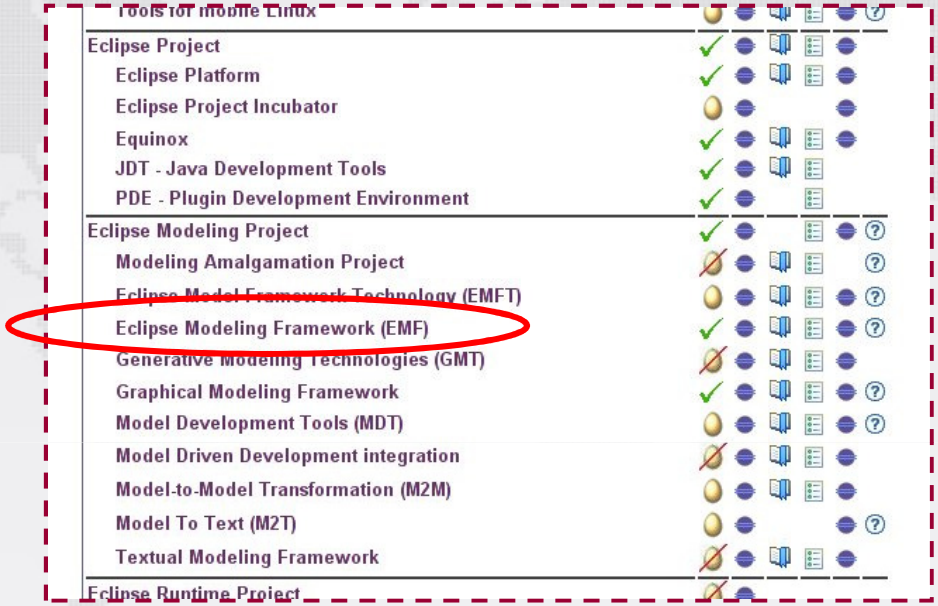
Model Injection: System Configuration

» **Main adopted technologies**

- Eclipse platform – http://www.eclipse.org
- Eclipse Modeling Framework (EMF) – http://www.eclipse.org/emf

Eclipse Modeling Framework

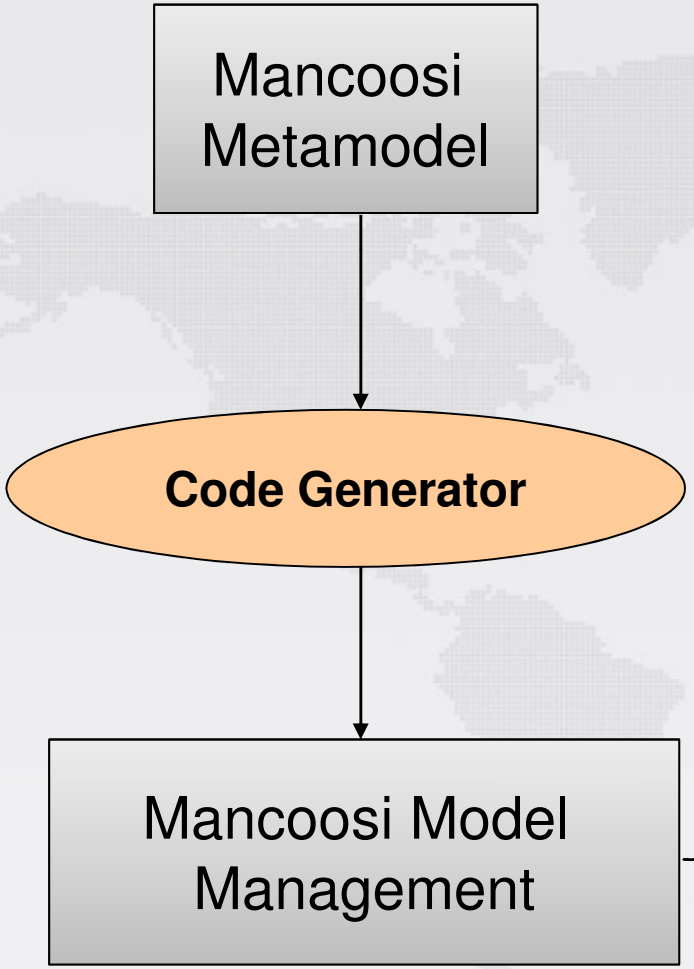
- EMF is a Java framework and code generation facility for building tools and other applications based on a structured data model
- EMF is part of the modeling project for Eclipse



Model Driven Development with EMF

- Almost every program we write manipulates some data model
  - Defined using UML, XML Schema, some other definition language, or implicitly in Java™
- EMF aims to extract this intrinsic “model” and generate some of the implementation code to provide benefits like
  - persistence support
  - model validation
  - model change notification
  - foundation for interoperability with other EMF-based tools and applications

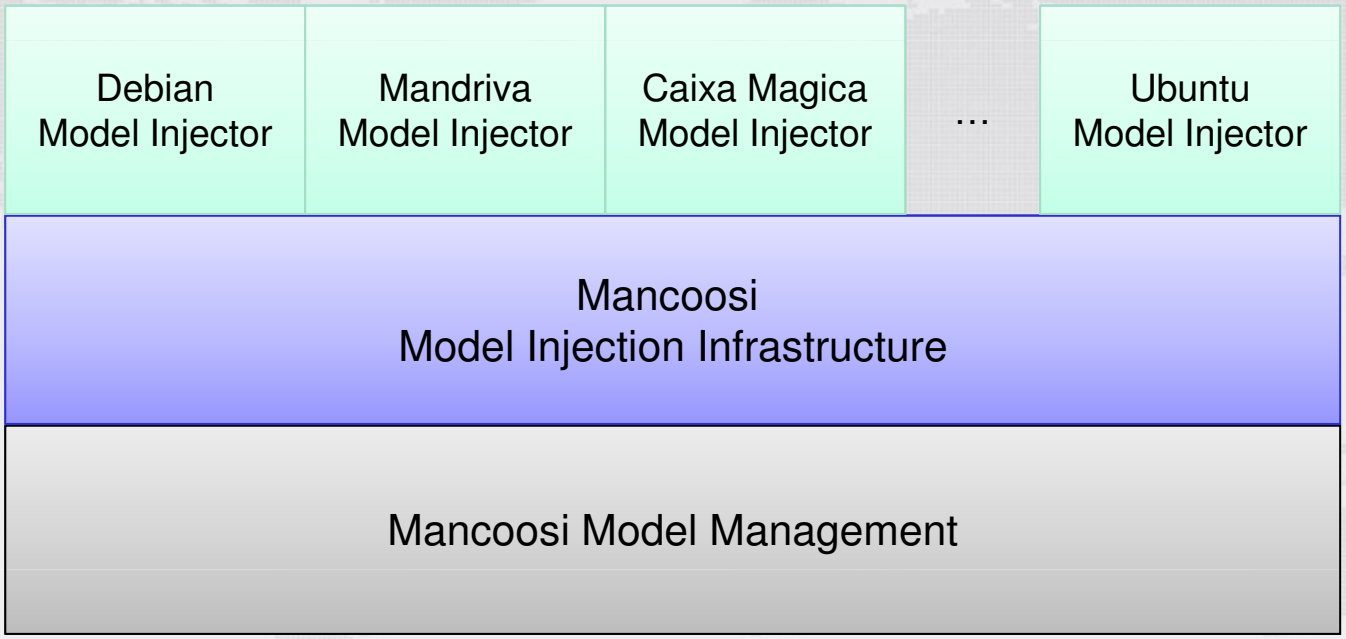
Using EMF in MANCOOSI



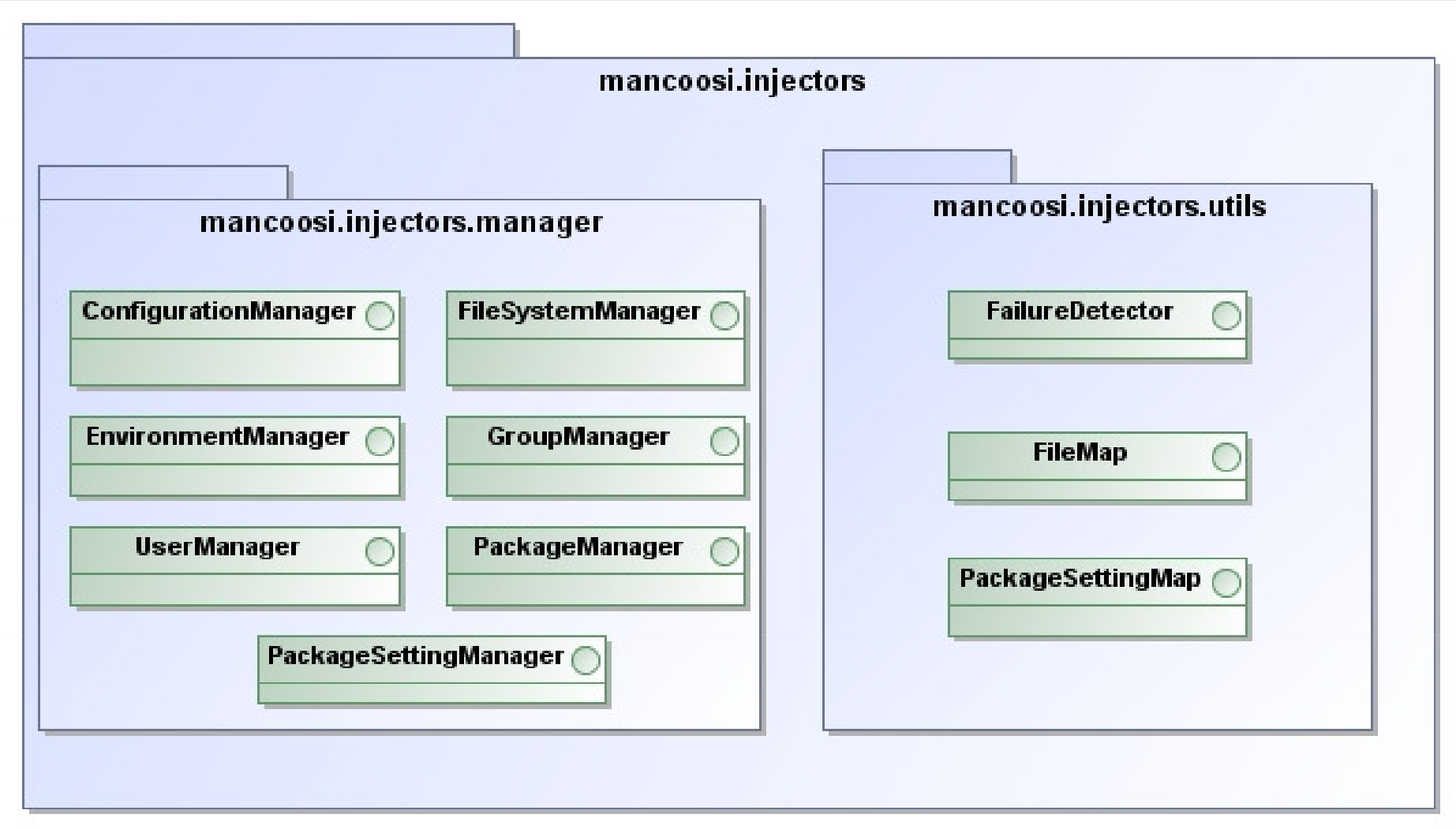
Used to:

- manage the persistence of Mancoosi models
- manipulate Mancoosi models
- interoperate with other EMF based tools like those used for calculating the differences between different system models

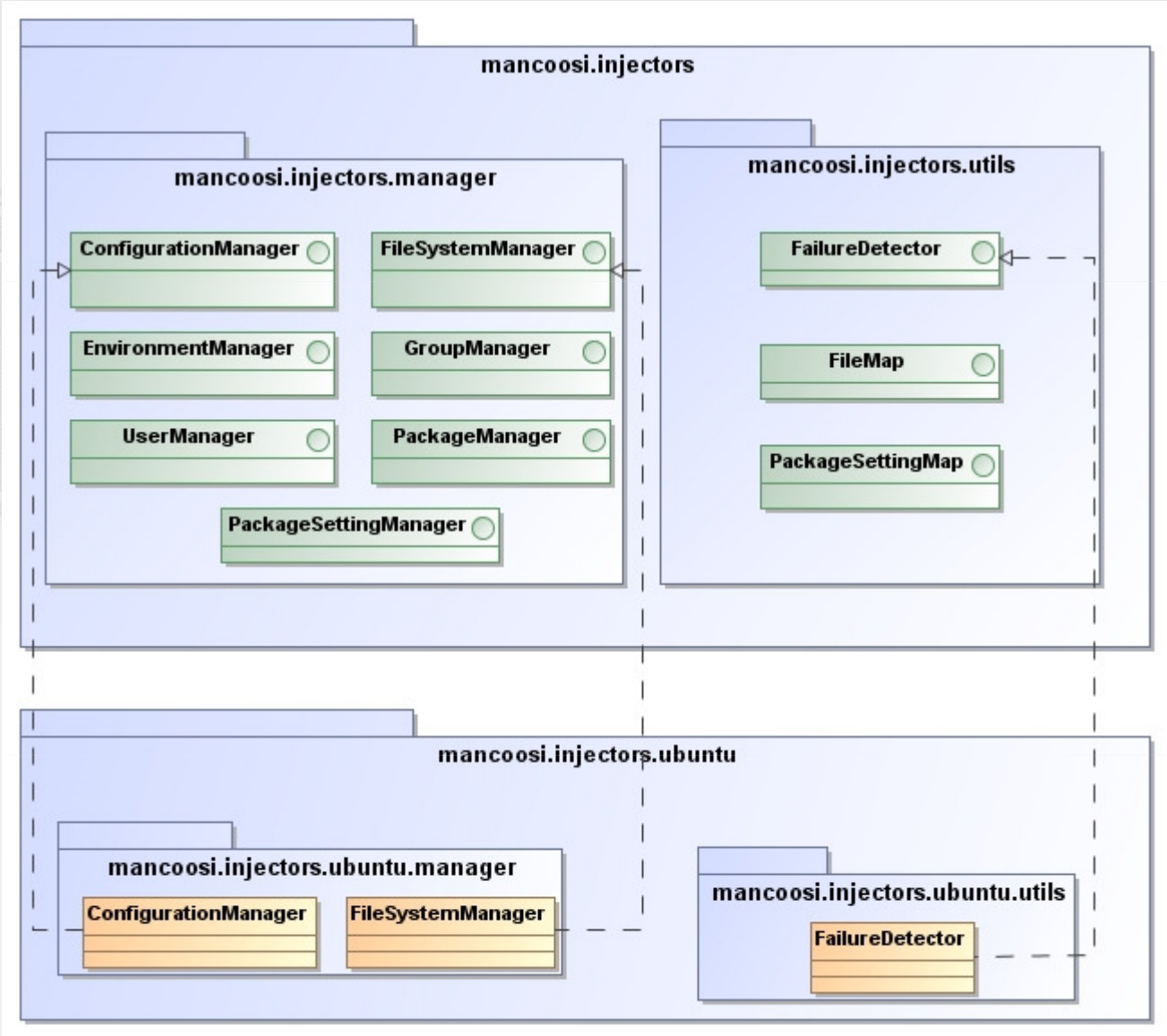
Model Injection



Model Injection Infrastructure



Ubuntu Model Injector

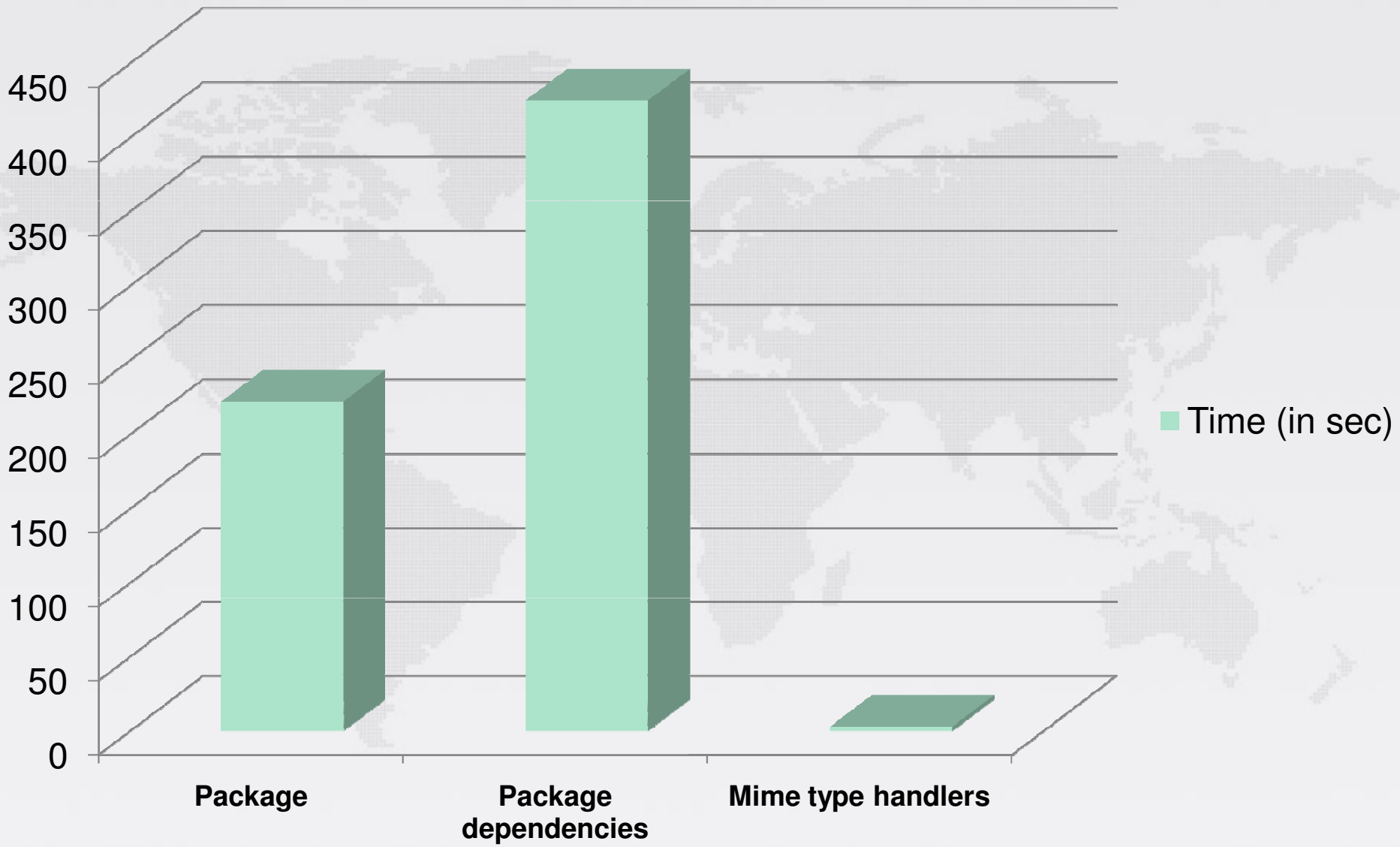


Ubuntu Model Injector

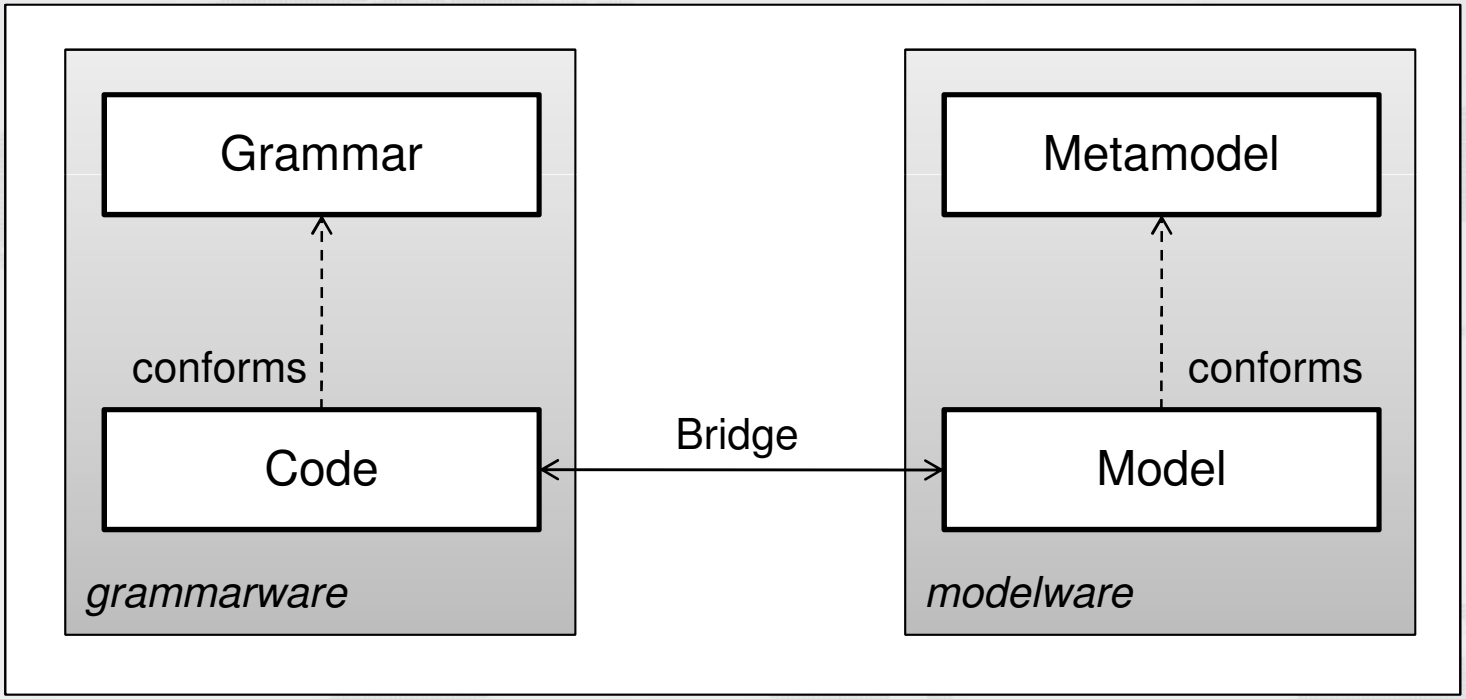
» Demo

- Injection of an Ubuntu system
- Static analysis
  - ✓ Detection of missing files
  - ✓ Missing package dependencies
  - ✓ Missing mime type handlers
  - ✓ …

Running the Ubuntu injector

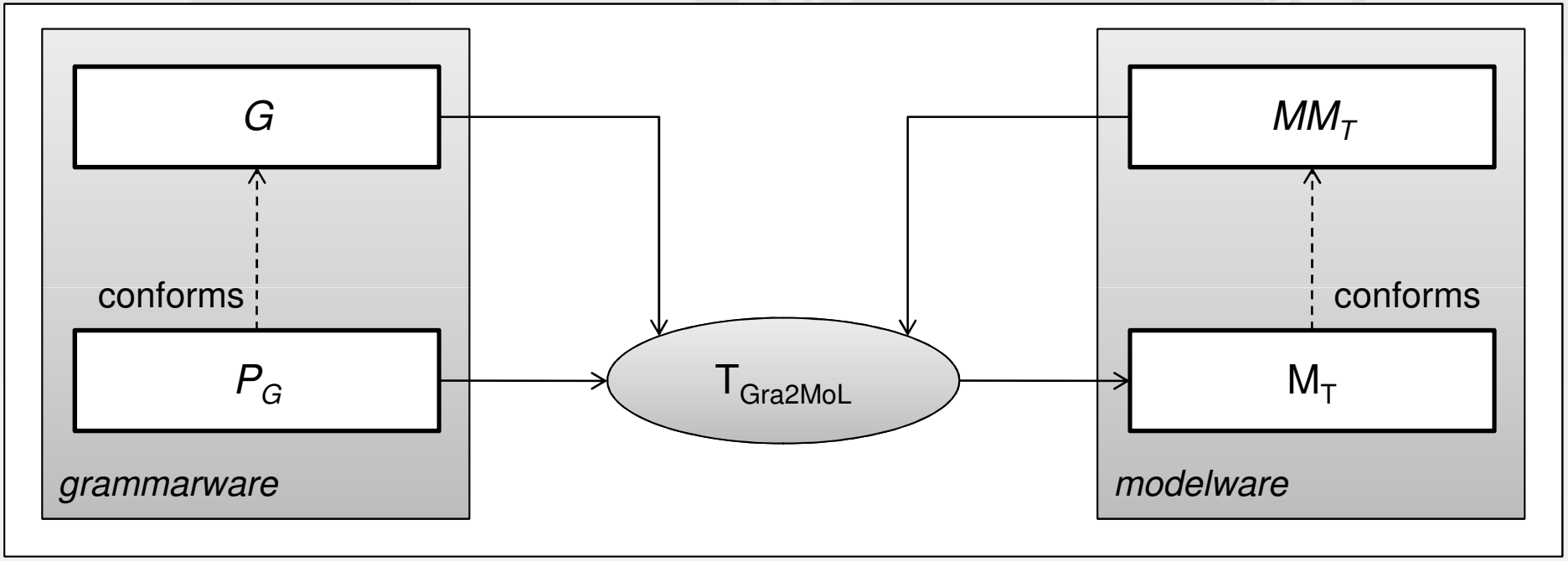


Model Injection: Maintainer Scripts



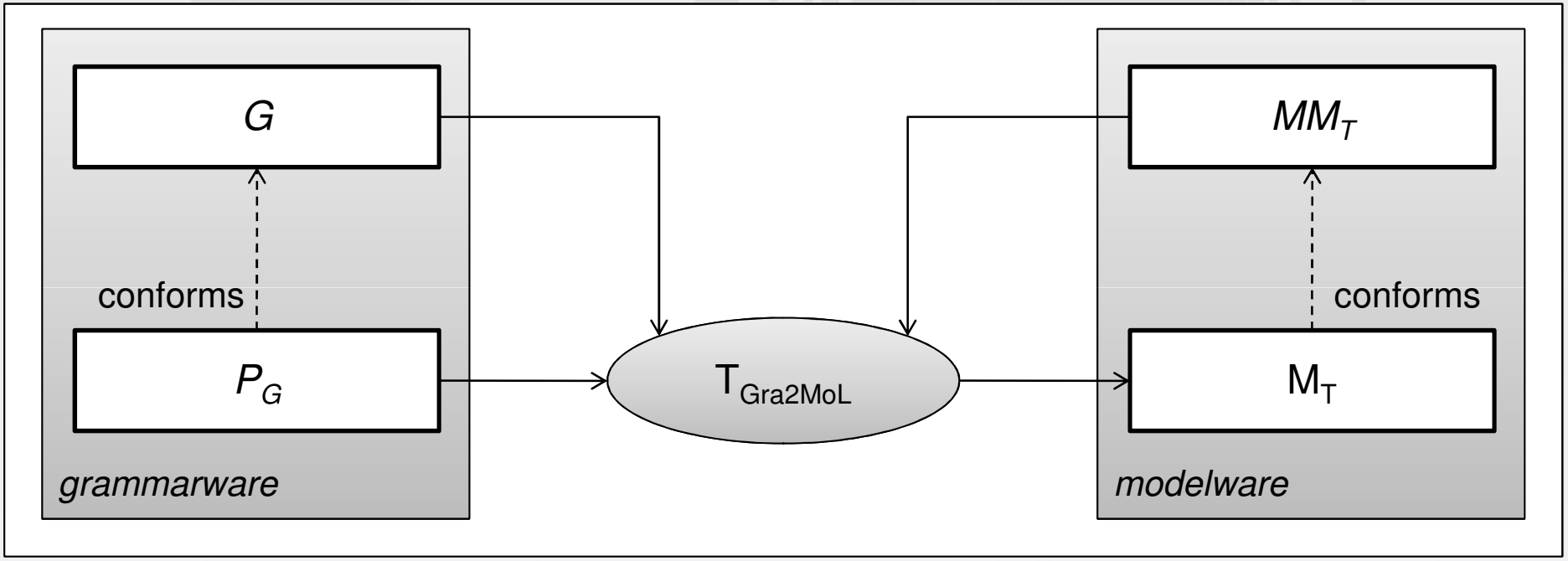
Model Injection: Maintainer Scripts > Gra2MoL

- Gra2Mol is from the University of Murcia
- It is based on the definition of a grammar-to-model transformation language which is specially tailored to address the grammarware-modelware bridge
- It promotes grammar reuse, and provides domain-specific features such as a query language to traverse syntax trees

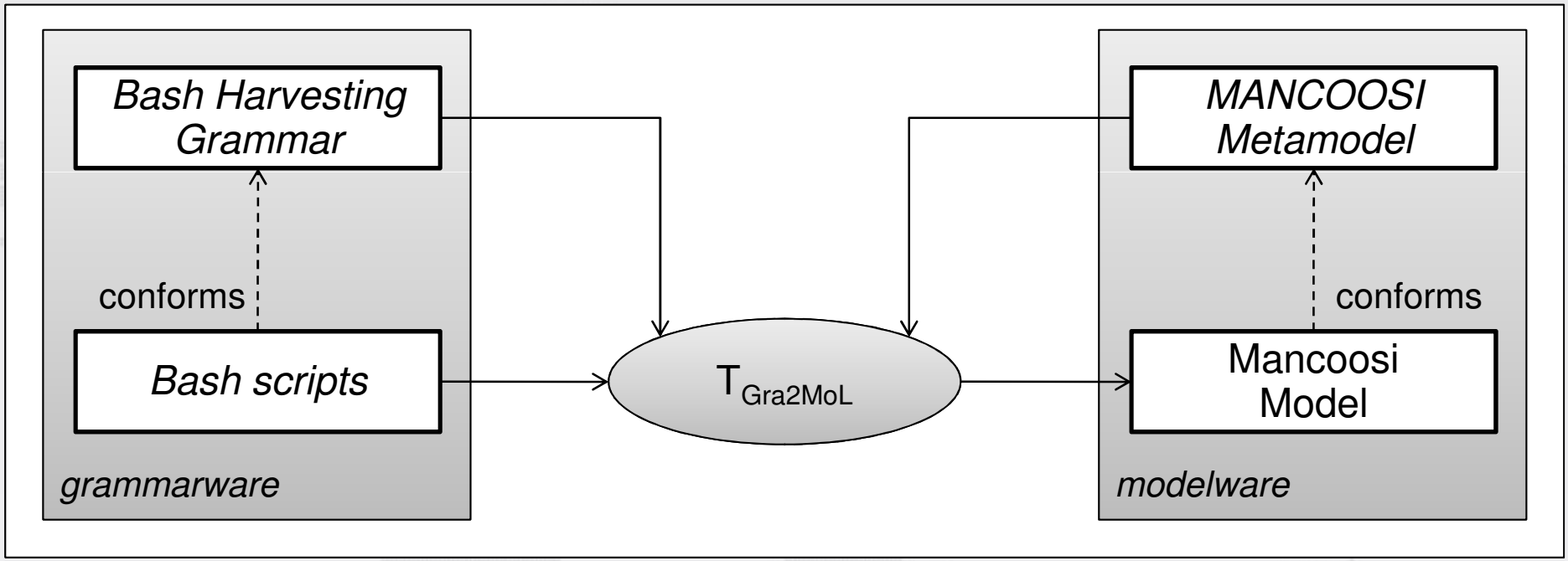


Model Injection: Maintainer Scripts > Gra2MoL

- Given a source program (P_G) conforming to the grammar (G) of a programming language, the objective is to generate a model conforming to a target metamodel MM_T
- The Gra2MoL language is used to explicitly specifying the relationships between source grammar elements and metamodel elements



Model Injection: Maintainer Scripts > Gra2MoL



Model Injection: Maintainer Scripts

» Some rules of the developed Bash Harvesting Grammar

```
mainRule
        :header (command)*
        ;
command
        :templates
        |shell_command
        |…
        ;
templates
        :UpdateMimeTypeCache
        |PostinstUdev
        |…
        ;
UpdateMimeTypeCache
        : 'if' ' [ "$1" = "configure" ] && [ -x "`which update-mime-database 2>/dev/null`" ];'
        'then' 'update-mime-database' '/usr/share/mime' 'fi'
        ;
…
shell_command
        :
…
```

Model Injection: Maintainer Scripts

» Fragment of the developed code-to-model transformation

```
rule 'mapPackage'
        from file f
        to InstalledPackage
        queries
        mains : //#mainRule;
          mappings
         PreinstScript = mains;
end_rule

…

rule 'mapUpdateMimeTypeCacheTemplate'
        from command/templates//template1 st
        to UpdateMimeTypeCache
        queries
        mappings
        location = st.location
end_rule
```

Model Injection: Maintainer Scripts > Example

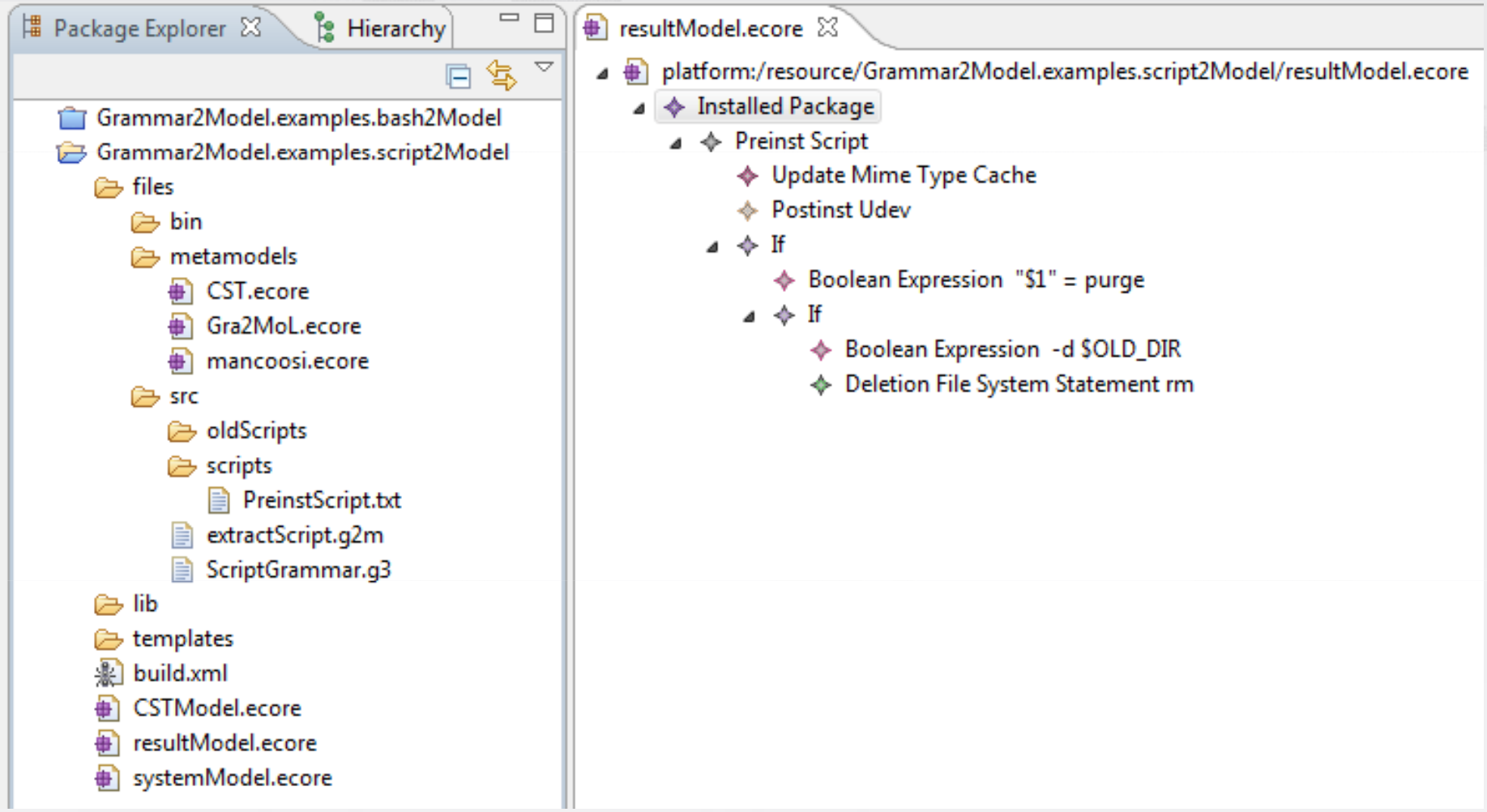
» Source bash script

```
#!/bin/sh

if [ "$1" = "configure" ] && [ -x "`which update-mime-database 2>/dev/null`" ];  then
        update-mime-database /usr/share/mime
fi
if [ "$1" = configure ]; then
        if [ -e "#OLD#" ]; then
                echo "Preserving user changes to #RULE# ..."
                if [ -e "#RULE#" ]; then
                        mv -f "#RULE#" "#RULE#.dpkg-new"
                fi
                mv -f "#OLD#" "#RULE#"
        fi
fi
if [ "$1" = purge ];  then
   if [ -d $OLD_DIR ];  then
       rm  -f  $OLD_DIR/$SCHEMA
   fi
fi
```

Model Injection: Maintainer Scripts > Example

» Generated model



Next steps

Short term

- Complete the implementation of the system injection
- Deliverable D2.2
- Integration of the model driven approach with the rollback components in WP3

Medium/Long term

- Integration of the model driven approach with the rollback components in WP3
- Complete the injection of the maintainer scripts
- Model-based framework for managing the complexity and the state of the GNU/Linux instantiation
- This is due at T0+36 and it will consist of
  - Simulator
  - Failure detector