D4.5.2-Megamodel for Transformations prototype

MoScript – Models Scripting Language

	NAME	PARTNER	DATE
WRITTEN BY	KLING W.	ATLANMOD	28/11/2011
REVIEWED BY	BERNARD Y.	Airbus	

©Galaxy consortium, 2010. ALL RIGHTS RESERVED. CONFIDENTIAL AND PROPRIETARY DOCUMENT.

RECORD OF REVISIONS

ISSUE	DATE	EFFECT ON		REASONS FOR REVISION
		PAGE	PARA	
1.0	28/10/2011			Document creation

©Galaxy consortium, 2010. ALL RIGHTS RESERVED. CONFIDENTIAL AND PROPRIETARY DOCUMENT.

TABLE OF CONTENTS

©Galaxy consortium, 2010. ALL RIGHTS RESERVED. CONFIDENTIAL AND PROPRIETARY DOCUMENT.

TABLE OF APPLICABLE DOCUMENTS

N°	TITLE	REFERENCE	ISSUE	DATE	SOURCE	
					SIGLUM	NAME
A1						
A2						
A3						
A4						

TABLE OF REFERENCED DOCUMENTS

N°	TITLE	REFERENCE	ISSUE
R1	D1.2.2 Galaxy glossary		
R2	D3.2 Megamodel for Transformations Architecture		
R3			
R4			

ACRONYMS AND DEFINITIONS

Except if explicitly stated otherwise the definition of all terms and acronyms provided in [R1] is applicable in this document. If any, additional and/or specific definitions applicable only in this document are listed in the two tables below.

Acronymes

ACRONYM	DESCRIPTION

Definitions

TERMS	DESCRIPTION

©Galaxy consortium, 2010. ALL RIGHTS RESERVED. CONFIDENTIAL AND PROPRIETARY DOCUMENT.

1. INTRODUCTION

Along the Galaxy project extensive research was carried out centred on the scalability problem when using large numbers, big sized and heterogeneous models with complex interrelations, and on how to best provide solutions to it. The results of this research work have been described in D.3.1.2 in terms of a Domain Specific Language architecture called MoScript. As a next step, this architecture has been materialized in a prototype, which serves as proof of concept for the research and whose description is the main purpose of this document.

The developed prototype has been conceived as a complement for the AtlanMod MegaModel Management (AM3) prototype, which is part of the Modisco MDT project and has been committed to eclipse.

This document describes the MoScript prototype. It explains the role the different plug-ins play in the language, its installation process and describes the concrete syntax of the language.

©Galaxy consortium, 2010. ALL RIGHTS RESERVED. CONFIDENTIAL AND PROPRIETARY DOCUMENT.

2. INSTALLATION PROCESS

The MoScript prototype is available from the Eclipse-MDT MoDisco SVN [1](sources only). The steps to install MoScript are the following:

- Download the Eclipse Modeling Tools from here: Eclipse Modeling Tools (Indigo)
- Install ATL from sources:
  - Download the ATL source code project set file (.psf) from here [1].
  - Import it into Eclipse with File->Import->Team->Team Project Set.
  - Download the MoScript patch for ATL from Bugzilla – Bug 361688.
  - Click right click on any ATL plugin project and select Team->Apply Patch ... and select the patch to apply it to ATL.
- Install subclipse and subversion if you have not done it yet.
- Install AM3 and MoScript from sources:
  - Open the SVN perspective by selecting Window -> Open Perspective-> SVN Repository Exploring perspective.
  - Add a new repository location by selecting File->New->Repository Location. The required parameters are the followings:
    - Url: https://dev.eclipse.org/svnroot/modeling/org.eclipse.mdt.modisco/incubation/trunk/am3
    - User: anonymous
  - Browse the just created repository location until /plugins/trunk and checkout all the plugins

[1] http://www.eclipse.org/downloads/packages/eclipse-modeling-tools/indigosr1

©Galaxy consortium, 2010. ALL RIGHTS RESERVED. CONFIDENTIAL AND PROPRIETARY DOCUMENT.

3. PROTOTYPE ARCHITECTURE

This section presents the concrete architecture of the prototype. It describes the internal structure of the prototype in terms of plug-ins, details its core metamodel and explains the extension mechanism provided. As mentioned in deliverable D3.2 (Megamodel for Transformations Architecture), MoScript depends on AM3 and ATL eclipse plugins as shown in Figure 1.

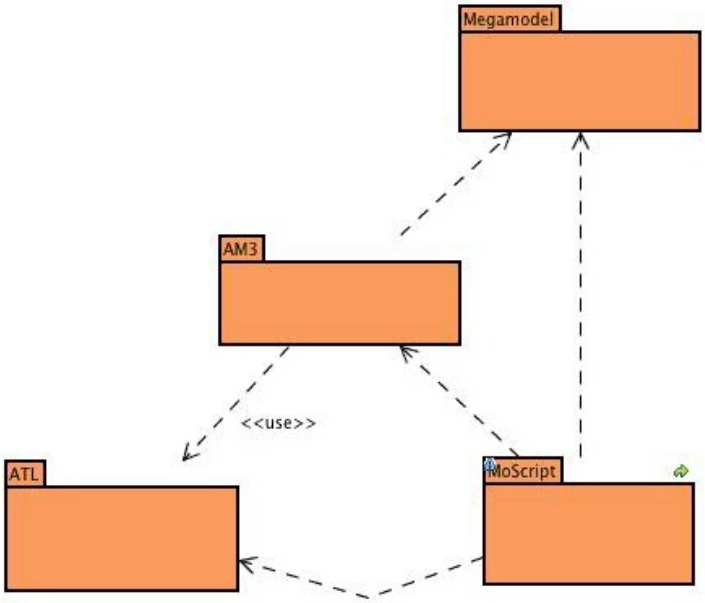


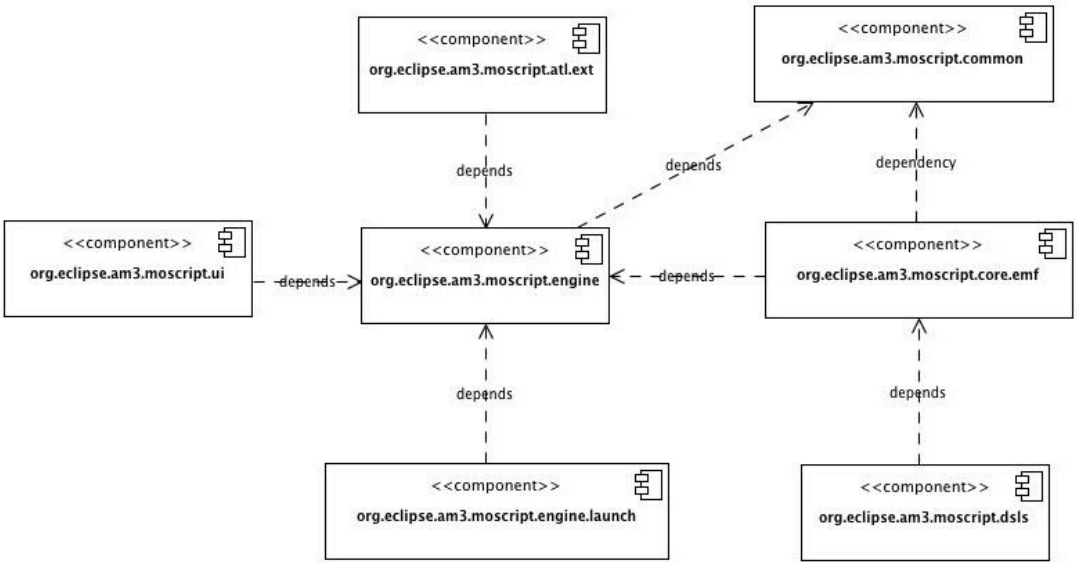
Figure 1 MoScript dependencies

3.1 PLUG-INS

As an Eclipse component, the prototype is implemented by a set of Eclipse plugins, each of them having a specific role. **Erreur ! Source du renvoi introuvable.** shows the MoScript components and their dependencies.

- **org.eclipse.am3.common**: Contains the MoScript abstract syntax in the form of an Ecore metamodel and provides means for retrieving the resource.
- **org.eclipse.am3.moscript.engine:** Contains the MoScript to ATL binary code compiler.
- **org.eclipse.am3.moscript.dsls:** Contains the MoScript parser and lexer classes
- **org.eclipse.am3.moscript.core.emf**: Contains the EMF factory for instantiating the MoScript EMF based engine.
- **org.eclipse.am3.moscript.ui**: Contains the classes related with MoScript project creation wizard, nature and builder.
- **org.eclipse.am3.moscript.launch:** Contains the classes related to the launcher of MoScript scripts.
- **org.eclipse.am3.moscript.atl.ext:** Contains the classes which extend the ATL OCL engine with custom side effects free operations.

©Galaxy consortium, 2010. ALL RIGHTS RESERVED. CONFIDENTIAL AND PROPRIETARY DOCUMENT.



©Galaxy consortium, 2010. ALL RIGHTS RESERVED. CONFIDENTIAL AND PROPRIETARY DOCUMENT.

4. MOSCRIPT LANGUAGE

MoScript reuses a big part of the ATL syntax and semantics such as the OCL expressions and data types, the control flow statements and the helpers. Any doubt about MoScript may be usually solved imitating the ATL syntax. MoScript syntax specifics will be explained next.

4.1 LANGUAGE GENERAL STRUCTURE

A MoScript program has the following general structure:

```
program program_name

uses library ...

[using {
  -- Comments
  ...
  -- Variable declarations
  variable :type = OclExpr ...
}]

do {
  -- Value assignements
  variable <- OclExpr;

  -- Operations invocations
  ... save(...);
  ... remove(OclExpr);
  ... register (...);

  -- Control flow statements
  if...
  for...
}

helper context OclAny def: helper_name( params ) :return_type
  ...
;
```

The using section is optional, and is used for declaring variables and assigning their initial value. The do section is mandatory and is the core of the program. In it, operations are used in combination with control flow statements and OCL queries to perform modelling artefacts manipulations.

4.2 MODEL ELEMENT DATA TYPE

The OCL specification introduces de model element data type. This type corresponds to the classes contained in the metamodel of the model is being queried. In the case of MoScript, the model which is being queried, is a megamodel, so the model elements allowed in MoScript

©Galaxy consortium, 2010. ALL RIGHTS RESERVED. CONFIDENTIAL AND PROPRIETARY DOCUMENT.

correspond to classes of the metamodel of the megamodel. MoScript uses the implementation of the megamodel provided by AM3 which conforms to the following metamodels among others:

- **AM3Core**: Is the top hierachy metamodel and can be found in */org.eclipse.gmt.am3.platform.runtime.core/model/AM3Core.ecore*
- **GMM**: Which extends AM3Core and can be found in*/org.eclipse.gmt.am3.platform.extension.globalmodelmanagement/model/GlobalModelManagement.ecore*
- **GMM4ATL**: A megamodel extension specific for ATL M2M transformations, which extends AM3Core and GMM and can be found in*/org.eclipse.gmt.am3.platform.extension.gmm4atl/model/GMM4ATL.ecore*

Model element variables are referred to by means of the notation `!Class`. For instance, `!Model`, `!TerminalModel`, `!ATLTransformation` etc.

4.3 OPERATIONS

```
Model :: allContentInstancesOf(elementType :String): Collection(OclAny)
```

This operation dereferences and load the physical model represented by the Model element. Then it queries the model and return a collection of OCL elements of type ***elementType***. The elements of the resulting collection are used as entry points to the model, from where the rest of the elements may be reached. Subsequent queries to the model content are made with standard OCL expressions.

```
Model :: inject(injectorName: String, modelElement: TupleType(...)) : Model
```

This operation make a projection of model expressed in a textual syntax to a model in XMI conforming to a given metamodel. ***injectorName*** is the name of the injector that is going to be used to make the projection of the model and ***modelElement*** represents the metadata information that will describe the new model. The structure of the tuple is explained along with the register operation.

```
Model :: save(location :String)
```

This operation stores the model in ***location***

```
ATLTransformation :: applyTo(Map{(key :String, model :Model),
                             ...,(..., ...)}) :Map(key :String, model :Model)
```

This operation applies the a transformation to one or more models. The ***key*** is the alias of that identifies the model inside an ATL transformation module and ***model*** is a model obtained from the megamodel.

©Galaxy consortium, 2010. ALL RIGHTS RESERVED. CONFIDENTIAL AND PROPRIETARY DOCUMENT.

```
register(TupleType(concreteType :String, value :Set(TupleType(...))))
```

This operation does the registration of a model in the megamodel i.e. it creates a new element in the megamodel of type ***concreteType*** where:

- **concreteType**: Is a string with the concrete type of the model element to be created. The string must be in the form of*Package::ConcreteType*. For instance GlobalModelManagement::URI, GlobalModelManagement::ReferenceModel etc.
- **value**: Corresponds to a set of attributes and references of the model element, each of them described again by a tuple as follows:
-
    - **Attribute tuple:**

```
TupleType(attributeName :String, primitiveType :String, attributeValue :String)
```

    - **Reference tuple:**

```
TupleType(creationId :String, referenceName :String, concreteType :String,
          referenceValue :Set(TupleType(...)))

TupleType(referenceName: String, concreteType :String, creationId: String)
```

There are two ways for declaring reference tuples. The first one creates the element to be referenced and then references it, the second one asumes that the elements have been already created (in the same register operation), thus it uses the ***creationId*** of the element to be referenced to find it and reference it.

The ***creationId*** is just a kind name that must be given to a model element instance when it is created, if we want it to be referenced by another élément in the same register operation. Otherwise it is not mandatory.

Note that the Tuples are used as a recursive way of expressing the creation of model elements, the model elements referenced by this elements and so on, and also the attributes contained in each model element.

```
IdentifiedElement :: remove()
```

This operation allows removing any megamodel element, which extends from the Core::IdentifiedElement element.

```
ReferenceModel :: registerInEMFpkgReg()
```

©Galaxy consortium, 2010. ALL RIGHTS RESERVED. CONFIDENTIAL AND PROPRIETARY DOCUMENT.

This operation registers the reference model in the EMF packages registry

```
collectWrkSpaceFileNames() :Collection(String)
```

The *collectWrkSpaceFileNames* operation queries the eclipse workspace to get all the workspace filenames.

4.4 EXAMPLES

The followings are a set of examples that demonstrate the use of some of the operations. Select all the models in the megamodel, get the first one and get a collection of all its elements of type EClass.

```
!Model.allInstances()->first().allContentInstancesOf('EClass');
```

Select all the models in the megamodel, get the first one and remove it from the megamodel.

```
!Model.allInstances()->first().remove();
```

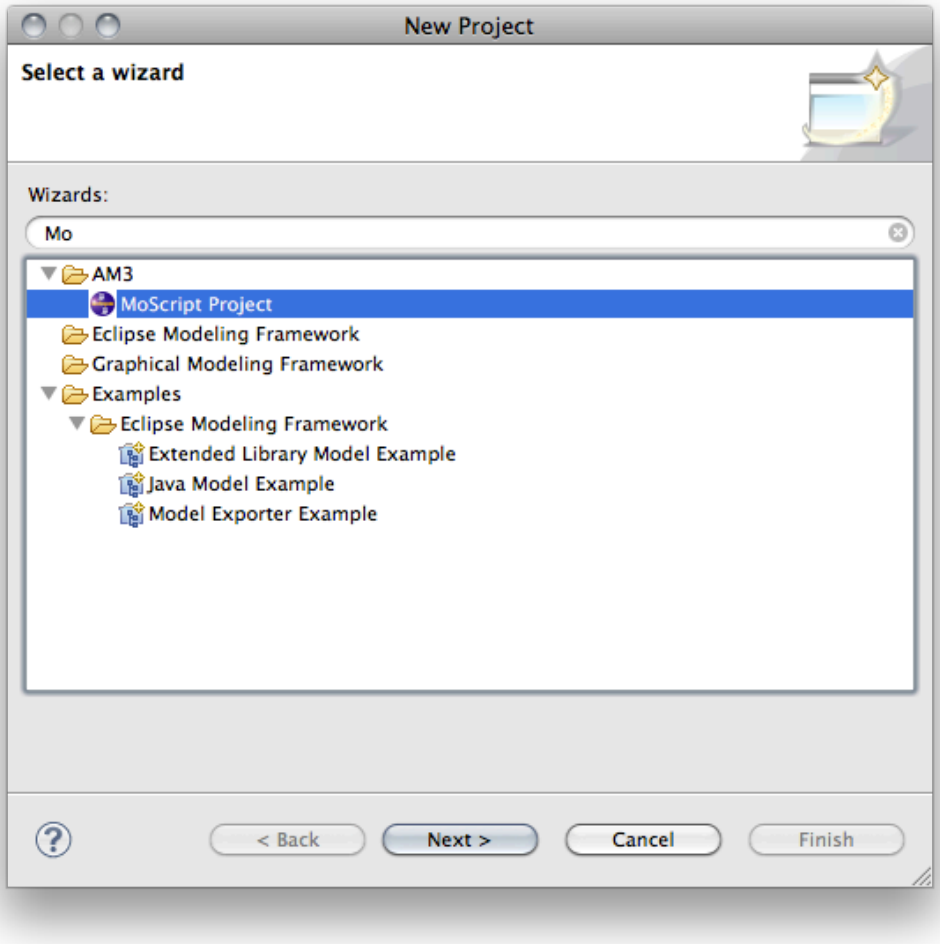
Select the models in textual syntax, which conform to a given *xmlRefernceModel* (grammar), get the first one and then inject it with the XML injector to obtain an XML model in XMI format conforming to an XML metamodel. The *xmlEcoreModelTuple* contains the metadata information for the creation of the new model element like, identifier, locator, referenceModel etc.

```
xml <- !TerminalModel.allInstances()->select(m | m.conformsTo = xmlReferenceModel)
                                    ->first();
xmlModel <- xml.inject('XML', xmlEcoreModelTuple);
```

4.4.1 MoScript Hello World!

Create a new MoScript project by clicking on File-> New-> Project and selecting the MoScript project type under the AM3 Folder.

©Galaxy consortium, 2010. ALL RIGHTS RESERVED. CONFIDENTIAL AND PROPRIETARY DOCUMENT.



Then create a MoScript script by clicking in File-> New-> File. Give it the name helloWorld.mscr. When the file is open, fill it with the following script code:

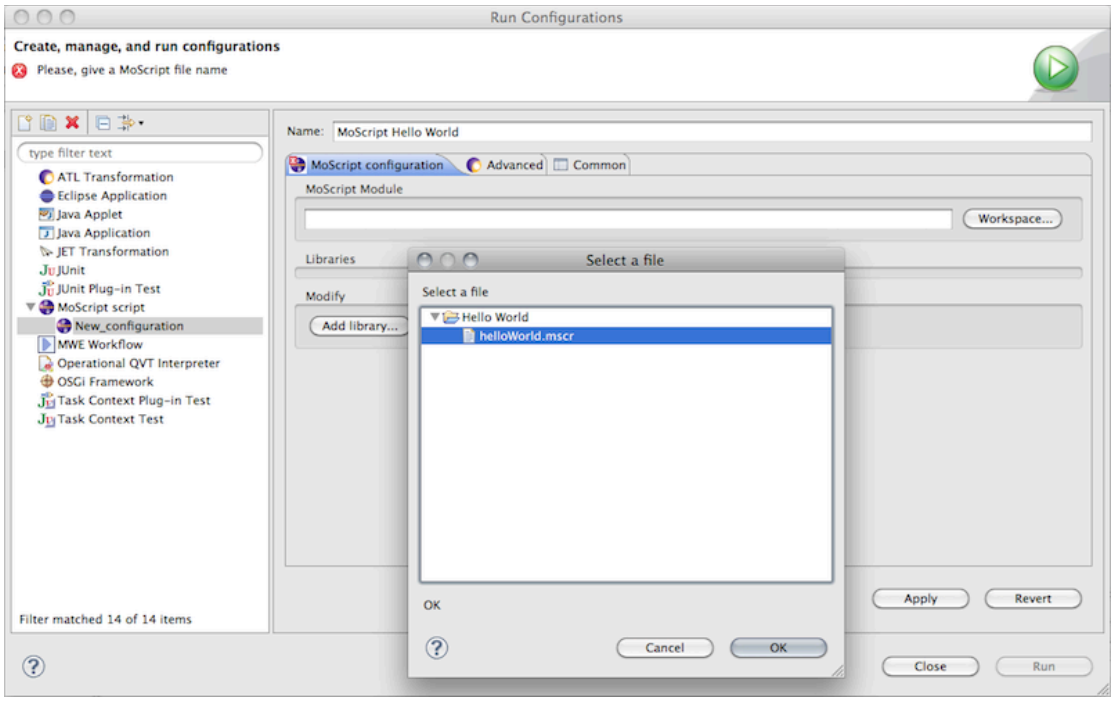
```
program helloWorld

do{
  'HelloWorld'.debug();
}
```

After saving the file, if there are no errors, you should see a new binary file called helloWorld.asm in the project explorer or navigator.

Now that the code is ready, create a MoScript launcher to run the script, by clicking in

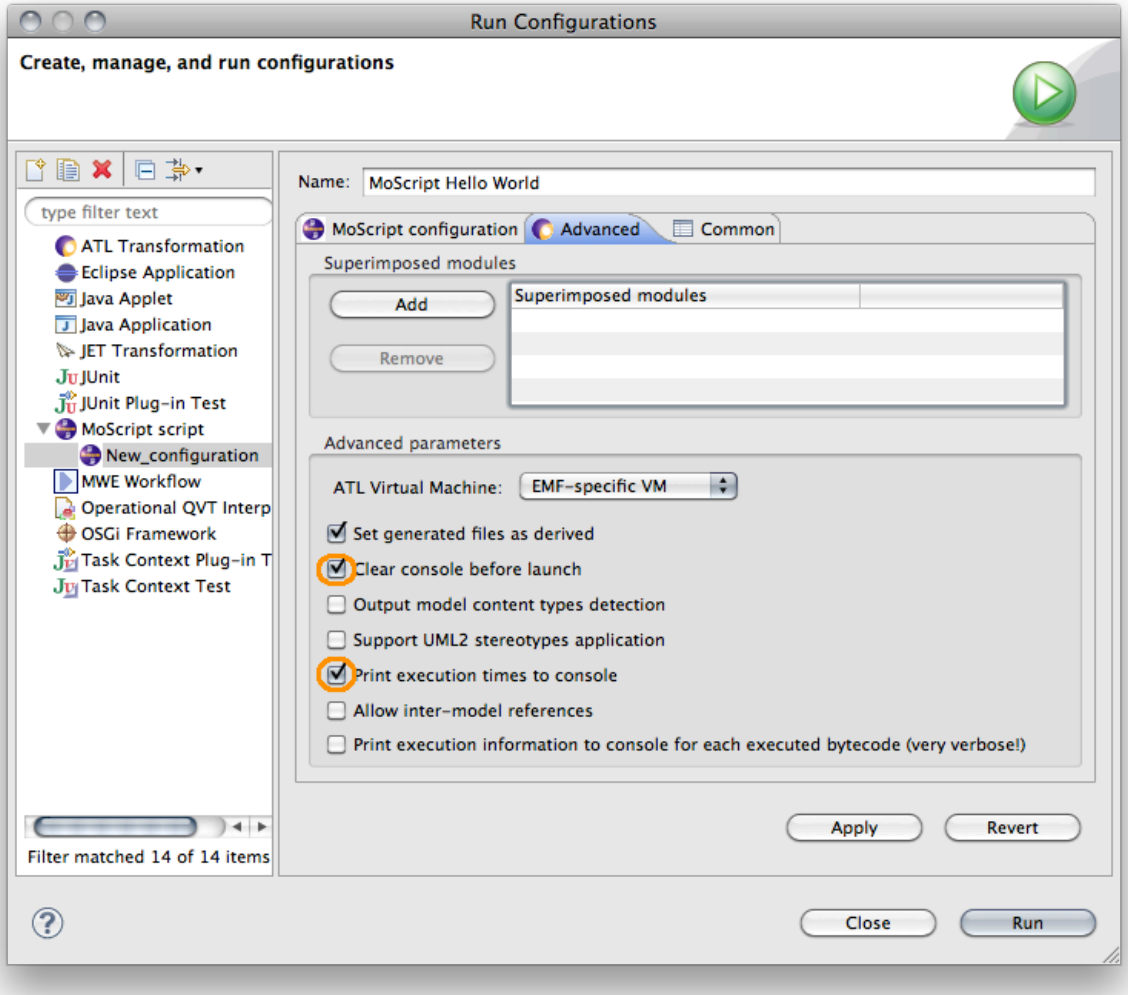
Run-> Run Configurations...

©Galaxy consortium, 2010. ALL RIGHTS RESERVED. CONFIDENTIAL AND PROPRIETARY DOCUMENT.



Browse in the workspace for the HelloWorld.mscr file and select it. Then, in the advanced tab, check the options Clear console before launch and Print execution times to console, so that you can clearly see when de script finishes.

©Galaxy consortium, 2010. ALL RIGHTS RESERVED. CONFIDENTIAL AND PROPRIETARY DOCUMENT.

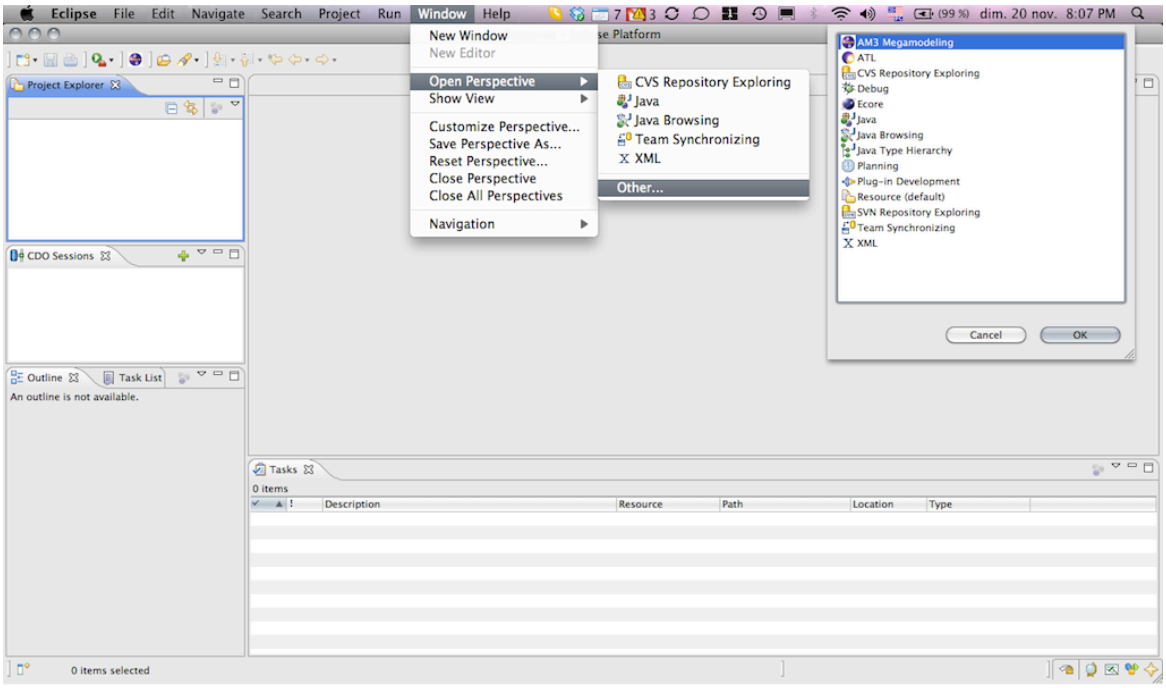


Then press the Apply button and the Run button. In the console window you will see an error like the following:

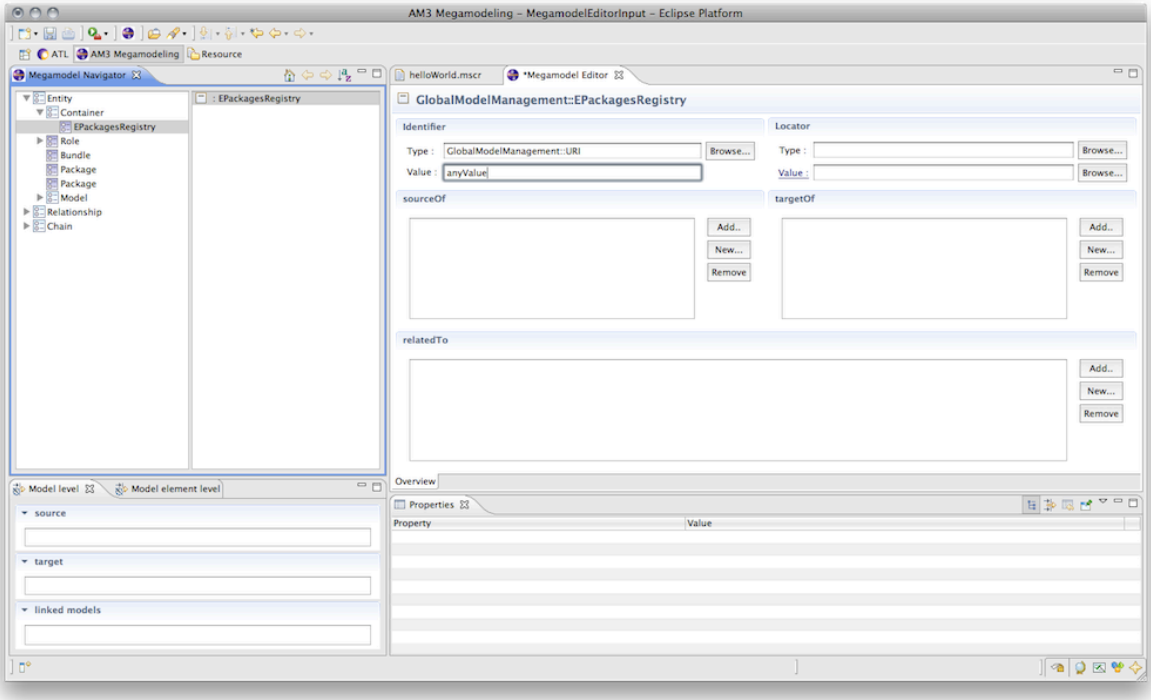
```
Error loading file:...workspacePath/.am3/megamodel.xmi: java.io.FileNotFoundException:
... (No such file or directory)
```

This is because the megamodel file has not been created yet. For creating the megamodel file, open the AM3 Perspective.

©Galaxy consortium, 2010. ALL RIGHTS RESERVED. CONFIDENTIAL AND PROPRIETARY DOCUMENT.

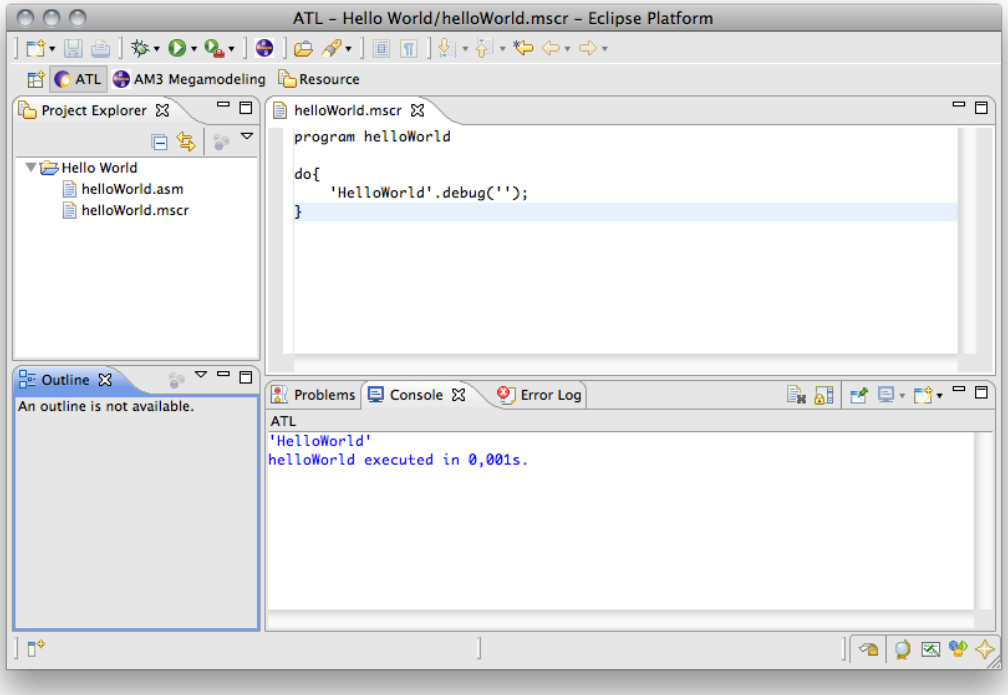


Create any model element in the megamodel left clicking in any element in the left panel, selecting new [element]. Give any value to the element and then use File->Save option to save the Megamodel. With this operation the Megamodel file is created in the filesystem.



Now try again to run the script by clicking on Run->Run History->Hello World and you should see the output in the console.

©Galaxy consortium, 2010. ALL RIGHTS RESERVED. CONFIDENTIAL AND PROPRIETARY DOCUMENT.



©Galaxy consortium, 2010. ALL RIGHTS RESERVED. CONFIDENTIAL AND PROPRIETARY DOCUMENT.