AsmEE: an Eclipse plug-in in a metamodel based framework for the Abstract State Machines

Angelo Gargantini
Università di Bergamo
angelo.gargantini@unibg.it

Elvinia Riccobene
Università di Milano
riccobene@dti.unimi.it

Patrizia Scandurra
Università di Milano
scandurra@dti.unimi.it

Abstract. The Abstract State Machines are a formal method successfully employed as system engineering method that guides the development of complex systems seamlessly from requirements capture to their implementation. Several tools supporting the ASMs have been developed in the past. To integrate these different tools and technologies we have proposed a metamodel based framework, called ASMETA, for the ASM. ASMETA includes a set of integrated tools and libraries, and can be integrated itself with the Eclipse project. In this paper we present the initial results of such integration, which consists in a Eclipse plug-in, called AsmEE, for editing and simulating ASM specifications. We present the AsmEE architecture, its features and capability and how they have been realized. Moreover we present the future work in the direction of a stronger integration between ASMETA and Eclipse.

1 Introduction

The Abstract State Machines (ASMs) [AsmBook] are nowadays acknowledged as a formal method successfully employed as systems engineering method that guides the development of complex systems seamlessly from requirements capture to their implementation.

The increasing application of the ASM formal method in academic and industrial projects has caused a rapid development of tools around ASMs of various complexity and goals: tools for mechanically verifying properties using theorem provers or model checkers, and execution engines for simulation and testing purposes. However, ASM tools are loosely coupled and their integration is hard to accomplish, so preventing ASMs from being used in an efficient and tool supported manner during the system development life-cycle.

To achieve the goals of developing a *unified abstract notation* for ASM, a notation independent from any specific implementation syntax and allowing a more direct encoding of the ASM mathematical concepts and constructs, and developing a general framework for a wide interoperability and integration of tools around ASMs, we exploited the metamodelling approach suggested by the Model-Driven Engineering (MDE) [MDE].

Exploiting the advantages offered by the metamodelling approach, we have developed an ASM metamodelling (ASMETA) framework which allows ASM tools (new and existing ones) interoperability by the combination of standards like MOF, XMI, and JMIs. This set of standards provides a global infrastructure for interoperability of ASM application tools including ASM model editors, ASM model repositories, ASM model validators, ASM model verifiers, ASM simulators, ASM-to-Any code generators, etc.

The ASMETA framework helps developers to build new tools by providing an XMI-based interchange format, standard JMI APIs, and several MOF-related facilities which supply standard projections toward other technical spaces. A developer who is interested in developing a new tool for the ASMs can completely base the tool development on the ASMETA framework and exploit all technologies provided by ASMETA in terms of specification language, abstract storage (i.e. the MOF-based model repository), APIs, interchange format, etc.

In order to provide the tools in the ASMETA framework with a user friendly graphical interface (e.g. an IDE for editing) we have investigated the integration of ASMETA with external IDEs or tools and we found that the best solution available nowadays is the Eclipse framework. Eclipse is a *open development platform comprised of extensible frameworks, tools and runtimes for building, deploying and managing software across the lifecycle.*

A large and vibrant ecosystem of major technology vendors, innovative start-ups, universities, research institutions and individuals extend, complement and support the Eclipse platform.

Moreover, providing a graphical interface to our tools is only the first step in the direction of the integration of Eclipse with ASMETA. Indeed Eclipse itself adopts in many its projects a Model Driven Approach (even if the technologies differ from those of ASMETA), and a better integration of ASEMTA and Eclipse at several level may be possible and desirable.

In this paper we present the initial results of the effort of the integration of ASMETA with Eclipse, in particular the development of an Eclipse plug-in for the ASMETA framework.

2 The Abstract State Machines

Abstract State Machines are an extension of Finite State Machines, where unstructured “internal” control states are replaced by states comprising arbitrary complex data. Even if the ASM method comes with a rigorous scientific foundation [AsmBook], the practitioner needs no special training to use it since they can be understood correctly as pseudo-code or Virtual Machines working over abstract data structures.

A complete mathematical definition of the ASM method can be found in [AsmBook], together with a presentation of the great variety of its successful application in different fields such as: definition of industrial standards for programming and modeling languages, design and re-engineering of industrial control systems, modeling e-commerce and web services, design and analysis of protocols, architectural design, language design, verification of compilation schemas and compiler back-ends, etc.

For our purposes, we quote here only the essential definitions of the ASMs. The states of an ASM are multi-sorted first-order structures, i.e. domains of objects with functions and predicates defined on them. The transition relation is specified by “rules” describing the modification of the functions from one state to the next. The basic form of a transition rule is the following guarded update:

if Condition **then** Updates

where Updates are a set of function updates of the form $f(t_1, \ldots, t_n) := t$ which are simultaneously executed when Condition is true. An ASM M is a finite set of rules for such guarded multiple function updates. State transitions of M may be influenced in two ways: internally, through the transition rules, or externally through the modifications of the environment. A computation of M is a finite or infinite sequence $S_0, S_1, \ldots, S_n, \ldots$ of states of M, where S_0 is an initial state and each S_{n+1} is obtained from S_n by firing simultaneously all of the transition rules which are enabled in S_n.

The notion of ASMs moves from a definition which formalizes simultaneous parallel actions of a single agent, either in an atomic way, Basic ASMs, and in a structured and recursive way, Structured or Turbo ASMs, to a generalization where multiple agents interact in a synchronous way, or asynchronous agents proceed in parallel at their own speed and whose communications may provide the only logical ordering between their actions, Synchronous/Asynchronous Multi-agent ASMs. Appropriate rule constructors also allow non-determinism (choose or existential quantification) and unrestricted synchronous parallelism (universal quantification forall).

3 The Asmeta Framework

ASMETA is an instantiation of the OMG metamodelling framework for the ASMs, and it has been developed to create and handle ASM models exploiting the advantages offered by the metamodelling approach and its related facilities (in terms of derivatives, libraries, APIs, etc.).

Figure 1 shows the ASMETA tool set which consists of the following components (those visualized in gray are still under development).

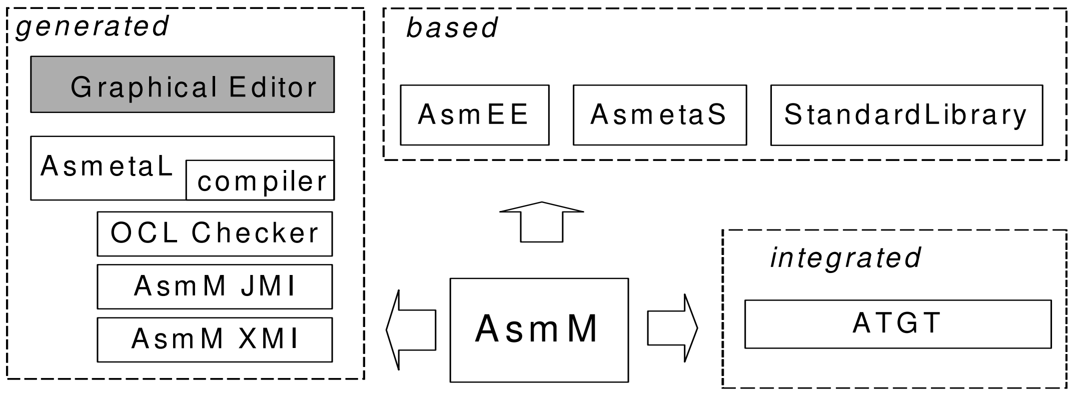

Figure 1 The Asmeta Framework

- The **AsmM metamodel**, based on MOF 1.4, representing in an abstract way concepts and constructs of the ASM formalism as described in [AsmBook]. A detailed description of AsmM can be found in [ASMM].

- The **AsmM OCL checker**, based on the OCLE tool and used to check if a given model is well-formed or not with respect to the OCL constraints defined over the AsmM metamodel.

- The **AsmM Java Metadata Interfaces** (JMIs) to manage the creation, storage, access, discovery, and exchange of ASM models (either at design time or runtime) in terms of Java objects.

- The **AsmM XMI** format which is XMI 1.2 complaint and is provided in terms of an XML Document Type Definition (DTD) automatically generated from the AsmM, for the interchange of ASM models among tools by XML serialization.

- The **AsmetaL** (ASMETA Language) textual notation for the AsmM, provided in terms of an EBNF (extended Backus-Naur Form) grammar generated from the AsmM (the abstract syntax) as a concrete syntax to be used by modelers to effectively write ASM models in a textual form. Details on AsmetaL can be found in [ASMETA_WEBSITE] [ASM05], while the process of derivation is described in [3M4MDA].

- A text-to-model **compiler** to parse the ASM models written in the ASMETAL notation, check for their consistency with respect to the OCL constraints of the metamodel, and translate information about concrete models into AsmM instances in a MOF-based repository by using the AsmM JMIs.
- A **standard library**, namely a declarative collection of predefined ASM domains (basic domains for primitive data values like Boolean, Natural, Integer, Real, etc., and structured domains over other domains like finite sets, sequences, bags, maps and cartesian products) and functions which implement a set of canonical operations on domains.
- An **graphical notation**, generated from the AsmM (the abstract syntax) as an alternative concrete syntax to be used by modelers to effectively write ASM models in a graphical form.
- The **AsmetaS**, a simulator to make AsmM models executable; essentially, it is an interpreter which navigates through a model repository where ASM specifications are stored (as instances of the AsmM metamodel) to make its computations.
- The ASM Tests Generation Tool (**ATGT**) [ATGT], an existing tool for test case generation from models, which has been made AsmM-compliant.
- A graphical front-end called **AsmEE** (ASM Eclipse Environment) which acts as IDE to edit, manipulate, and export ASM models by using all tools/artifacts listed above. This environment is implemented as an Eclipse plug-in and it is the main subject of this paper.

All available material on the ASMETA tool set (including source code, binaries, documentation and a great variety of ASM specifications) can be found in [ASMETA_WEBSITE], under GNU General Public License (GPL).

4 AsmEE: the ASMeta Eclipse Environment

AsmEE is a graphical front-end for the ASMETA: it is based on Eclipse and can be used as IDE to edit, manipulate, and export ASM models by using all tools/artifacts listed above. Figure 2 show the architecture of AsmEE with respect to all the other components of the ASMETA framework (the external components are in yellow). AsmEE has three main components: **utils** which contains a set of utilities (e.g. AsmEE preferences), **Editor** which provides editing facilities and **Simulator** which controls the Asmeta Simulator. On one hand, AsmEE is based on the Eclipse Plug-in Development Environment (PDE), while on the other hand it is built on the top of the Asmeta simulator and the AsmetaL compiler, which are normally distributed and used as command line tools. AsmEE calls some APIs exported by AsmetaL compiler and AsmetaS to perform the tasks of parsing and simulating ASM through a graphical interface provided by Eclipse. In this way we did not modify the layers under AsmEE and all the components below it still do not require Eclipse to run.

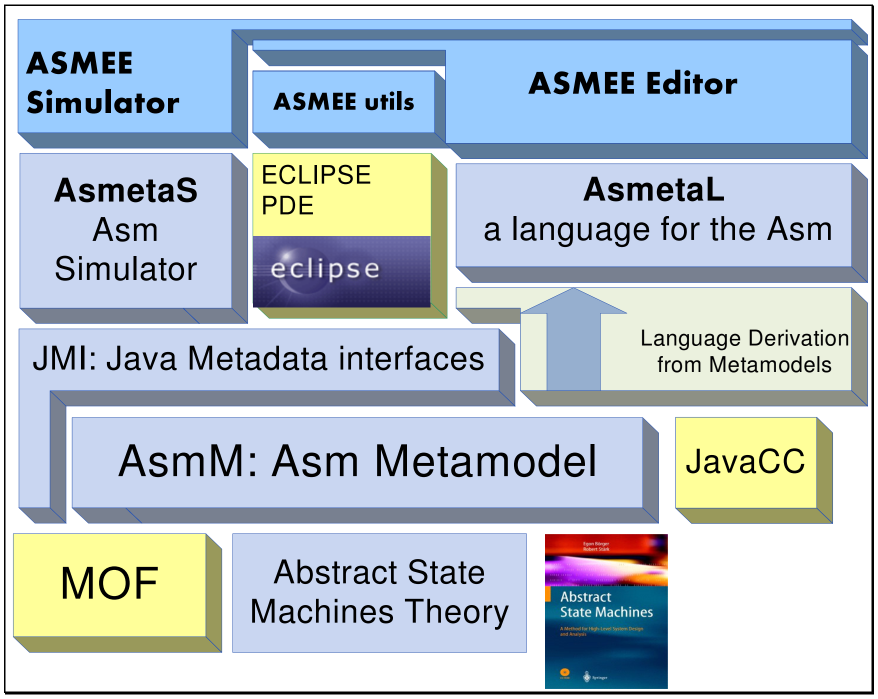

Figure 2 The AsmEE stack

5 AsmEE Features

AsmEE extends the Eclipse IDE by adding new capabilities to Eclipse. This has been done in the classical and well known way proposed by the Plug-in Development Environment (PDE). The definition of a plug-in mainly consists in defining several Eclipse extensions, i.e. introducing new classes which extend or implement Eclipse classes and registering them in the plugin.xml file in appropriate extension points. In the following we present the main extensions we have so far realized for AsmEE.

5.1 AsmetaL Editor

Implementing the basic features of an editor for specifications written in AsmetaL in AsmEE has required to extend the editors supported by Eclipse by introducing a new editor for AsmetaL files:

```
public class AsmetaLEditor extends TextEditor {…}
```

and then adding the new editor as an extension point of the editors. This can be easily done by the plug-in editor, which will modify the file plugin.xml as follows :

```
   <extension point="org.Eclipse.ui.editors">
      <editor
            name="AsmetaL Editor"
            extensions="asm"
            class="org.asmeta.Eclipse.editor.AsmetaLEditor"
            …
```

Since `AsmetaLEditor` extends the basic Eclipse TextEditor, it contains already all the basic functionalities. We have added only a ColorManager to ménage the syntax highlighting and a method `updateMarkers` To display the errors in the text window

5.2 Syntax Highlighting

The syntax highlighting is obtained by introducing a new class:

```
public class AsmScanner extends RuleBasedScanner
```

which builds all the rules necessary to recognize special tokens to be highlighted by a particular color.

The tokens are identified by means of rules of several subclasses of the interface `org.Eclipse.jface.text.rules.IRule`:

- A `WordRule` which identifies a token if is equals to a particular keyword. We add to the word rule all the keywords of the language by reading them from the grammar file (written in JavaCC) and then by invoking the method `addWord`
- A `EndOfLineRule` to recognize lines which start with a particular sequence of characters. For example single line comments are recognized by a `new EndOfLineRule ("--", commentToken)`
- A `MultiLineRule` to recognize a region with starts and ends with a particular sequence of characters. For example the comments in C++ style are recognized by a `new MultiLineRule("/*", "*/", commentToken),`
- `A PatternRule` to recognize words starting with a particular pattern. For example variables, which always start with a $, are recognized by a `new PatternRule("$", " ", varsToken,'$',true);`

5.3 New File Wizard

A new AsmetaL specification wizard has been developed by using the PDE wizards. We had only to implement a class for the wizard (the Eclipse wizard helped us)

```
public class AsmmNewWizard extends Wizard implements INewWizard
```

the class which asks some information about the new file (name and container project) is implemented in the

```
public class AsmmNewWizardPage extends WizardPage
```

The wizard is registered as extension point

```
<extension point="org.Eclipse.ui.newWizards">

      <wizard category="AsmM"

          class="org.asmeta.Eclipse.editor.wizards.AsmmNewWizard"

...
```

5.4 Preferences

Some preferences for the editor (e.g. colors for some kind of tokens), has been developed by using the Eclipse wizard. The result is a class which contains several fields (like `ColorFieldEditor)`:

```
public  class  AsmEEPreferencePage  extends  FieldEditorPreferencePage  implements
IWorkbenchPreferencePage
```

The preference page is then registered as extension

```
   <extension point="org.Eclipse.ui.preferencePages">
      <page
            class="org.asmeta.Eclipse.editor.preferences.AsmEEPreferencePage"
...
```

5.5 Console

To display the messages coming from the parser or form the interpreter we had to introduce a new consoled for AsmEE. This can be done by extending (by using the extension point) Eclipse with a class

```
public class AsmEEConsole extends IOConsole{
```

A new console of this kind can be added to the consoles managed by the `IConsoleManager`.

Our console contains also some buttons, defined as `IConsolePageParticipant`.

5.6 Actions

One of most classical extensions is the introduction of actions activated by buttons or/and menus. We have introduced the following actions:

✓ parse and check the specification	▶ run the simulator
run the simulator using a random environment	run the simulator until the update set is empty
⇨ resume the computation	⏸ pause the computation
🟥 stop (abort) the computation	

5.7 Jobs

Since normal actions on any resources would lock the entire workspace, we had to use the Job API to perform the parsing e simulation of AsmetaL specifications. Thanks to the Job apis of Eclipse we are able now to start, schedule and cancel the jobs of paring and simulating. This has required the definition of several classes like this one for parsing:

```
public class ParseJob extends Job
```

These classes actually defines the actions to be performed when the user requestes a specific tasks, and the rules for pausing, resuming and cancelling the task.

6 Deploying AsmEE

To distribute AsmEE, we have developed a plug-in site as Eclipse project. However, since AsmEE requires several external libraries (like log4j) and the AsmetaL compiler and the simulator are normally packaged as jar file, we were unable to package everything in an unique jar file containing all the classes and resources. The AsmEE jar file is still a jar file (to allow the installation via the update site), but it contains several jar files itself and it needs to be unpacked when the AsmEE is installed. Details how to distribute a plug-in which uses a set of external libraries can be found at [PDE_JARS].

AsmEE can be installed via the update site [ASMETA_WEBSITE]. When installed, the user will be able to edit and simulate AsmetaL specifications inside the Eclipse IDE.

7 Future Work

Besides improving the AsmEE plug-in by adding new advanced features (e.g. auto completion), we plan to work in two directions to better integrate ASMETA with Eclipse:

1. *Porting the ASM metamodel to the EMF/Ecore technology*. The EMF project is an Eclipse subproject providing a modeling framework and code generation facility for building tools and other applications based on a structured data model. Although EMF and MOF (the metamodelling technology used by ASMETA) are similar, some differences still exist. Initial attempt to translate AsmM to EMF by means of the ATL tool [ATL] can be found at [ASMETA_WEBSITE].

2. *Developing a graphical editor for Abstract state Machines by the GMF approach*. Note that concrete notations (i.e. notations used by the developers to actually write models) derived from metamodels can be graphical too. The Eclipse Graphical Modeling Framework (GMF) provides a generative component and runtime infrastructure for developing graphical editors based on Eclipse Modelling Framework (EMF) and the Eclipse Graphical Editing Framework (GEF). The GMF follows a novel approach which suggests to derive modelling tools, like model editors, directly from metamodels. Completed the porting of AsmM to EMF, we plan to use GMF to develop a graphical editor for ASMs.

3. *Integrating AsmEE as Eclipse subproject.* In the future AsmEE could become an Eclipse subprojects by adopting all the standard technologies and methodologies promoted by the Eclipse community. AsmEE itself could became a framework in the Eclipse style by defining a set of public APIs and extension points.

In this case a developer could define new plug-in for AsmEE by using the PDE technology or develop a new application for the ASMs by using the RCP and AsmEE.

8 Bibliography

[3M4MDA] Angelo Gargantini, Elvinia Riccobene, Patrizia Scandurra *Deriving a textual notation from a meta-model: an experience on bridging Modelware and Grammarware* in 3M4MDA 2006 - - European Conference on Model Driven Architecture, in Bilbao, Spain from July 10th-13th 2006.

[ASM05] Scandurra, Gargantini, Genovese, Genovese, and Riccobene. *A Concrete Syntax derived from the Abstract State Machine Metamodel*. **ASM'05**

[AsmBook] E. Boerger and R. Staerk. Abstract State Machines: A Method for High-Level System Design and Analysis. Springer Verlag, 2003.

[ASMETA_WEBSITE] http://asmeta.sourceforge.net/

[ASMM] A. Gargantini, E. Riccobene, and P. Scandurra. *Metamodelling a Formal Method: Applying MDE to Abstract State Machines*. Technical Report 97, DTI Dept., University of Milan, November 2006.

[ATGT] ATGT: ASM tests generation tool. http://cs.unibg.it/gargantini/project/atgt/

[ATL] The AMMA Platform. http://www.sciences.univ-nantes.fr/lina/atl/

[MDE] Stuart Kent: Model Driven Engineering. IFM 2002: 286-298

[PDE_JARS] http://cs.unibg.it/gargantini/didattica/Eclipse/plugin_libs.html