ATL: Eclipse Support for Model Transformation

Freddy Allilaire, Jean Bézivin, Frédéric Jouault, Ivan Kurtev
ATLAS (INRIA & LINA)
University of Nantes
2, rue de la Houssinière BP 92208
44322, Nantes, France
{freddy.allilaire | jean.bezivin | frederic.jouault |ivan.kurtev}@univ-nantes.fr

Abstract

In the context of Model Driven Engineering models are the main development artifacts and model transformations are among the most important operations applied to models. A number of specialized languages have been proposed, aimed at specifying model transformations. Apart from the software engineering properties of transformation languages the availability of high quality tool support is also of a key importance for the industrial adoption and ultimate success of MDE. In this paper we present ATL: a model transformation language and its execution environment based on the Eclipse framework. ATL tools provide support for the major tasks involved in using a language: editing, compiling, executing, and debugging.

1. Introduction

Model transformations play an important role in the Model Driven Engineering (MDE) approach. Developing model transformation definitions is expected to become a common task in model driven software development. Software engineers should be supported in performing this task by mature MDE tools and techniques in the same way as they are presently supported by classical IDEs, compilers, and debuggers in their everyday programming work.

One direction for providing such a support is to develop domain-specific languages designed to solve common model transformation tasks. Indeed, this is the approach that has been taken recently by the research community and software industry. As a result a number of model transformation languages have been proposed [[1], [9], [11]].

In this paper we describe ATL (ATLAS Transformation Language): a transformation language developed as a part of the AMMA (ATLAS Model Management Architecture) platform [2]. In the AMMA platform, in addition to ATL, other projects are being developed. One is the ATLAS model Weaver (AMW) [14]. Another one is the ATLAS MegaModel Management Tool (AM3) [13]. The last one is the ATP (ATLAS Technical Projectors), a set of injectors and extractors to/from other technical spaces [17]. ATL, AMW, AM3 and ATP are presently the essential part of the AMMA platform. All these tools are built on top of the Eclipse Modeling Framework (EMF). Documentation and open source software related to various components of the AMMA platform will be regularly announced and made available. Number of documentation documents and examples are already available for the ATL component in the ATL Eclipse/GMT subproject [12].

ATL is inspired from the OMG QVT standard recommendation [9] and builds upon the OCL formalism [10]. It is a hybrid language, i.e. it provides a mix of declarative and imperative constructs. The major language features and the supporting development tools are presented. We informally explain the syntax and semantics of ATL by using a simple case study as example.

The paper is organized as follows. Section 2 explains the basic concepts related to model transformations and to the operational context of ATL. Section 3 presents the language. Section 4 describes the tool support available for ATL: the ATL virtual machine, the ATL compiler, the IDE based on Eclipse, and the debugger. Section 5 presents a brief comparison with other approaches for model transformations. Section 6 gives conclusions.

2. Models and Model Transformations

In MDE, models are considered as the unifying concept in IT engineering. Traditionally, models have been used as initial design sketches mainly aimed for communicating ideas among developers. MDE promotes models to primary artifacts that drive the whole development process. The notion of model goes beyond the narrow view of semi-formal diagram thus requiring much more precise definitions and modeling languages. In this section we introduce the basic terminology needed for the further discussion.

The MDE community has been using the concepts of *model*, *metamodel*, and *metametamodel* for quite some time. A model is a representation of a system. It captures some characteristics of the system and provides knowledge about it. In MDE we are interested in models expressed in precise languages. The conceptual foundation of a modeling language, when expressed as a model, is called *metamodel*. Often, metamodels are considered as definitions of the abstract syntax of modeling languages. The relation between a model expressed in a language and the metamodel of this language is called *conformsTo* (or *C2* for brevity). Metamodels are in turn expressed in a modeling language called metamodeling language. Its conceptual foundation is captured in a model called *metametamodel*. Models, metamodels, and metametamodel form a three-level architecture with levels named M1, M2, and M3 respectively. For a formal definition of these concepts based on multi-graphs the reader is referred to [6].

The principles of MDE may be implemented in several standards. For example, OMG proposes a standard metametamodel called Meta Object Facility (MOF) [8]. An example of a metamodel in the context of OMG standards is the UML metamodel.

An MDE development process is a series of transformations over models. Usually, transformations are refinement steps over models that decrease the level of abstraction but other forms may also be found in transformation chains. The goal is to produce a model that contains enough details for automatic generation of executable code. In general, model transformations may be implemented in different ways, for example, by using a general purpose programming language. In this paper we focus on transformations expressed in a specialized model transformation language.

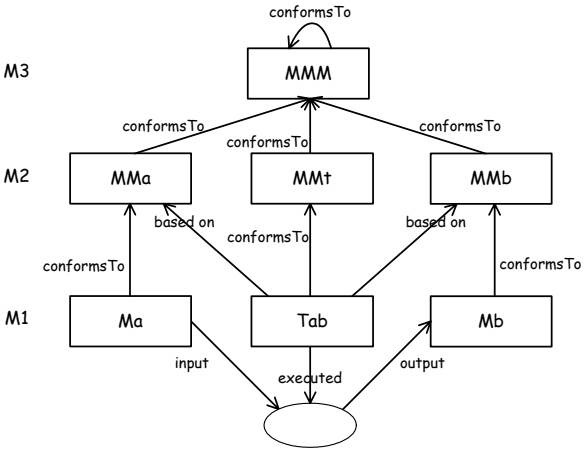

Figure 1 Model transformation pattern

Model transformations in MDE follow a common pattern known as *model transformation pattern* shown in Figure 1. *Tab* is a transformation program which execution results in automatic creation of model *Mb* from *Ma*. These three entities (*Tab*, *Mb*, and *Ma*) are all models conforming to *MMt*, *MMb*, and *MMa* metamodels respectively. *MMt* corresponds to the abstract syntax of the transformation language. The three metamodels conform to a metametamodel named *MMM*. In the context of OMG standards, the metametamodel *MMM* is the MOF.

3. ATLAS Transformation Language

ATL is applied in the context of the transformation pattern shown in Figure 2. In this pattern a source model *Ma* is transformed into a target model *Mb* according to a transformation definition *mma2mmb.atl* written in the ATL language. The transformation definition is a model conforming to the *ATL* metamodel. All metamodels conform to the *MOF*.

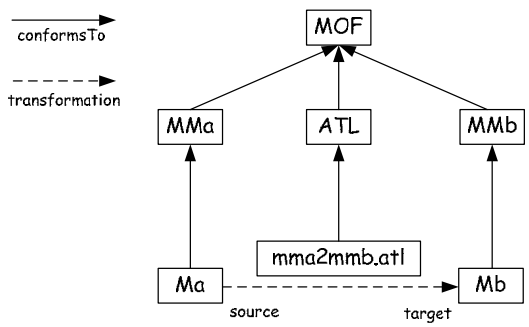

Figure 2 Overview of ATL transformational approach

ATL is a hybrid transformation language. It contains a mixture of declarative and imperative constructs. We encourage a declarative style of specifying transformations. However, it is sometimes difficult to provide a complete declarative solution for a given transformational problem. In that case developers may resort to the imperative features of the language.

ATL transformations are unidirectional, operating on read-only source models and producing write-only target models. During the execution of a transformation the source model may be navigated but changes are not allowed to it. Target model cannot be navigated. A bidirectional transformation is implemented as a couple of transformations: one for each direction.

Source and target models for ATL may be expressed in the XMI OMG serialization format. Source and target metamodels may be expressed also in XMI or in the more convenient KM3 notation [6].

An ATL transformation can be decomposed into three parts: a header, helpers and rules. The header is used to declare general information such as the module name (it is the transformation name: it must match the file name), the input and output meta-model and potential import of needed libraries. Helpers are subroutines that are used to avoid code redundancy. We can imagine a helper for translating UML visibility kind to MOF visibility kind in a UML to MOF transformation. It is easy to guess the utility of this helper when we know all UML and MOF elements that define a visibility. Rules are the heart of ATL transformations because they describe how output elements (based on the output meta-model) are produced from input elements (based on the input meta-model). They are made up of bindings, each one expressing a mapping between an input element and an output element. A more detail presentation of the language syntax accompanied with examples can be found in [[16], [12]]. A more detailed and formal specification of the ATL semantics can be found in [4].

4. ATL Development Tools

ATL Development Tools (ADT) is developed under the ATL Eclipse/GMT subproject [[12], [15]].

Figure 3 ATL Home Page in GMT

ATL is accompanied by a set of tools built on top of the Eclipse platform. Figure 4 represents the overall architecture of the ATL Development Tools (ADT). ADT is composed of the ATL transformation engine (*Engine* block) and the ATL Integrated

Development Environment (IDE: *Editor*, *Builder* and *Debug* blocks). Complete usage documentation is provided in the ATL user manual. Each block is described in the remaining part of this section.

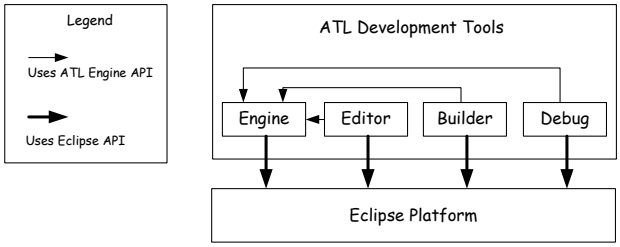

Figure 4 ATL Development Tools architecture

4.1 Engine

The ATL engine is responsible for dealing with core ATL tasks: compilation and execution. ATL transformations are compiled to programs in specialized byte-code. Byte-code is executed by the ATL Virtual Machine (VM). The VM is specialized in handling models and provides a set of instructions for model manipulation.

The architecture of ATL execution engine is shown in Figure 5.

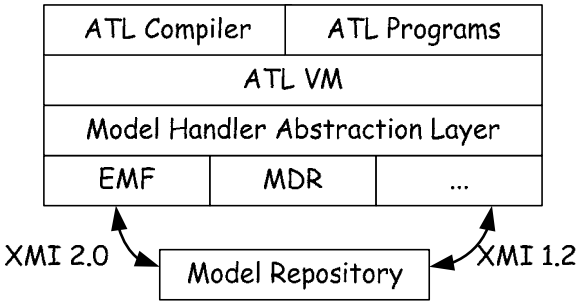

Figure 5 The architecture of the ATL execution engine

The VM may run on top of various model management systems. To isolate the VM from their specifics an intermediate level is introduced called *Model Handler Abstraction Layer*. This layer translates the instructions of the VM for model manipulation to the instructions of a specific model handler. Model handlers are components that provide programming interface for model manipulation. Some examples are Eclipse Modeling Framework (EMF) [3] and MDR [7]. Model repository provides storage facilities for models. Within Eclipse, it corresponds to the workspace file system.

4.2 Editing

The ATL editor supports syntax highlighting, error reporting, and outline view (i.e. tree-based representation of the ATL program). The bottom left part of Figure 6 shows how the *Class2Relational.atl* example is represented under the editor. The bottom right part of the figure shows the corresponding outline.

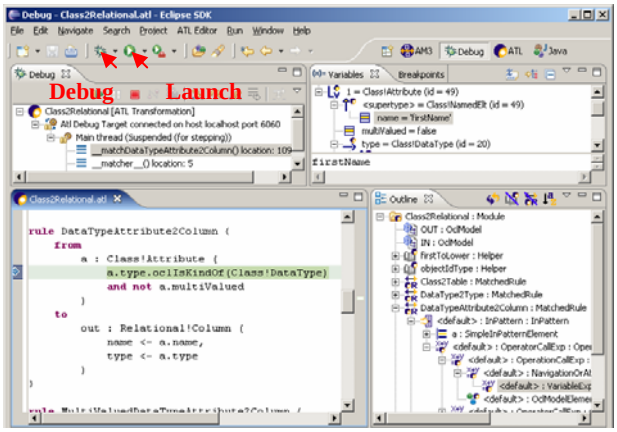

Figure 6 ATL editor and debugger screenshot

4.3 Building and Launching ATL Transformations

ATL compiler is automatically called on each ATL file in each ATL project during the Eclipse build process. By default, this process is triggered when a file is modified (e.g. saved).

Executing an ATL transformation requires the declared source and target models and metamodels to be bound to actual models (i.e. XMI files typically ending in *.xmi* or *.ecore*). This is done in the launch configuration wizard. Figure 7 gives a screenshot of this wizard and shows the correspondence between the user interface and the operational context of the transformation given in Figure 1. Red arrows map models and metamodels to their declarations whereas blue arrows map the declarations to the corresponding files. For instance, the source model *Ma* is declared as model *IN* with metamodel *Class*. The file corresponding to model *IN* is *sample-Class.ecore*. The first tab that can be seen on the top left of Figure 7 is entitled “ATL Configuration” and is used to specify the location of the transformation to execute. This wizard is accessible from Eclipse *Launch* button (see Figure 6).

The ATL engine delegates reading and writing models to the underlying model handler. When EMF is used, for instance, source and target models must be in EMF XMI 2.0. They can be edited with EMF reflective editor, which represents models as trees. More complex transformation scenarios can use other kinds of formats (e.g. XML, textual). This is illustrated by the CPL2SPL example provided with ATL Development Tools. However, dealing with such formats involves more than model-to-model transformation for which ATL is designed. It is consequently out of the scope of this paper.

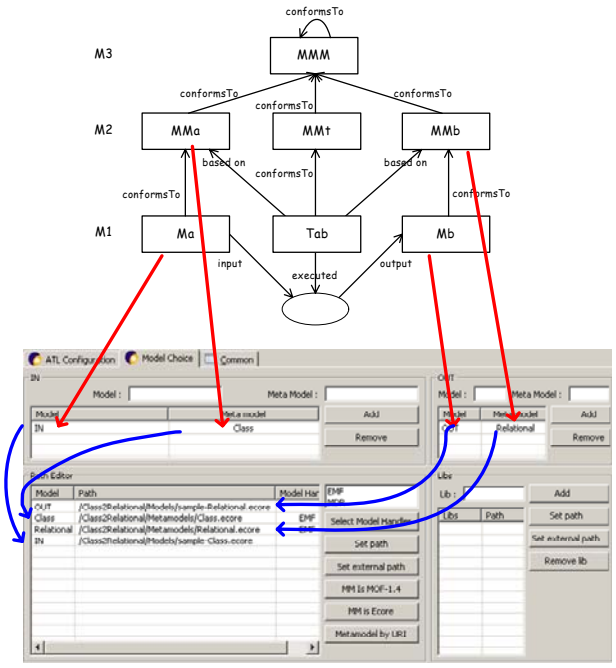

Figure 7 ATL launch configuration

4.4 Debugging

ATL transformations may be debugged using the same launch configuration used for launching. The only difference is that we now use the *Debug* button (see Figure 6) instead of *Launch*. Transformations can be executed step-by-step or run normally. In this case, execution stops when an error occurs or a breakpoint is reached. The current context (i.e. values of variables) may be analyzed using the variable view (see the top right part of Figure 6). It enables simple navigation in source and target models from the current context (rule or helper).

5. Related Work

In the last couple of years we observed a number of proposals for model transformation languages. Some of them are a response to the QVT RFP issued by OMG [9]. ATL is applicable in QVT transformation scenarios where transformation definitions are specified on the base of MOF metamodels. ATL is capable to perform other transformation scenarios going beyond QVT context where source and target models are artifacts created with various technologies such as databases, XML documents, etc. For this purpose ATL relies on AMMA platform features that are beyond the scope of this paper. A comparison between ATL and the last QVT proposal may be found in [5].

An important class of transformation approaches relies on graph transformations techniques [[1], [11]]. ATL is not directly based on the mathematical foundation of these approaches. An interesting direction for future research is to formalize the ATL semantics in terms of graph transformation theory. The declarative part of ATL is especially suitable for this.

6. Conclusions

In this paper we presented ATL: a hybrid model transformation language developed as a part of the ATLAS Model Management Architecture. ATL is supported by a set of development tools built on top of the Eclipse environment: a compiler, a virtual machine, an editor, and a debugger. ATL allows both imperative and declarative approaches to be used in transformation definitions depending on the problem at hand.

ATL is currently used or evaluated on more than 100 sites, academic or industrial. There is an initial library of ATL transformations and number of documentation (a user manual, an installation guide and a starter guide) available in open source from the GMT Eclipse project. The current state of ATL tools already allows solving non-trivial problems. This is demonstrated by the increasing number of implemented examples and the interest shown by the rapidly growing ATL user community that provides a valuable feedback.

7. Acknowledgements

This work is partially supported by ModelWare, IST European project 511731 and by System@tic Paris-region, the french cluster in complex system design and management.

8. References

[1] Agrawal A., Karsai G., Kalmar Z., Neema S., Shi F., Vizhanyo A.The Design of a Simple Language for Graph Transformations, Journal in Software and System Modeling, in review, 2005

[2] Bézivin, J., Jouault, F., Rosenthal, P., Valduriez, P. Modeling in the Large and Modeling in the Small. MDAFA'2004, Springer-Verlag LNCS 3599, pages 33-46.

[3] Budinsky, F., Steinberg, D., Raymond Ellersick, R., Ed Merks, E., Brodsky, S. A., Grose, T. J. Eclipse Modeling Framework, Addison Wesley, 2003

[4] Di Ruscio, D., Jouault, F., Kurtev, I., Bezivin, J., Piearantonio, A. Extending AMMA for Supporting Dynamic Semantics Specifications of DSLs. LINA Research Report number 06.02, University of Nantes, April, 2006

[5] Jouault, F, Kurtev, I. On the Architectural Alignment of ATL and QVT. In: Proceedings of ACM Symposium on Applied Computing (SAC 06), Model Transformation Track, Dijon, Bourgogne, France, April 2006

[6] Jouault, F., Bézivin, J. KM3: a DSL for Metamodel Specification FMOODS 2006, Bologna, Italy, 14-16 June 2006

[7] Netbeans Meta Data Repository (MDR). http://mdr.netbeans.org

[8] OMG/MOF Meta Object Facility (MOF) Specification. OMG Document AD/97-08-14, September 1997. Available from www.omg.org

[9] OMG/RFP/QVT MOF 2.0 Query/Views/Transformations RFP, OMG document ad/2002-04-10. Available from www.omg.org

[10] OMG. Object Constraint Language (OCL). (2003) OMG Document ptc/03-10-14

[11] Varró, D., Varró, G., Pataricza, A. Designing the automatic transformation of visual languages. Journal of Science of Computer Programming, vol. 44, pp. 205-227, Elsevier, 2002

[12] ATL official site (Eclipse/GMT subproject): http://www.eclipse.org/gmt/atl/

[13] AM3 official site (Eclipse/GMT subproject): http://www.eclipse.org/gmt/am3/

[14] AMW official site (Eclipse/GMT subproject): http://www.eclipse.org/gmt/amw

[15] Eclipse GMT Official site: http://www.eclipse.org/gmt

[16] Jouault, F, and Kurtev, I : Transforming Models with ATL. In: Proceedings of the Model Transformations in Practice Workshop at MoDELS 2005, Montego Bay, Jamaica.

[17] Bezivin, J, and Kurtev, I : Model-based Technology Integration with the Technical Space Concept. In: Metainformatics Symposium. Springer-Verlag.