The AMMA Platform

The ATL project has allowed broadening the view of MDE. Model transformations are absolutely necessary to any application of MDE. However, they are probably not sufficient. We need other operations as well. **AMMA** (Atlas Model Management Architecture) is a model management platform developed by the Atlas group (INRIA & LINA). In the AMMA platform, in addition to ATL, some new components are being developed. One is **AMW** (Atlas model Weaver). Another one is **AM3** (Atlas MegaModel Management tool). A last one is **ATP** (Atlas Technical Projectors), a set of injectors and extractors to/from other technical spaces. ATL, AMW, AM3 and ATP are presently the essential part of the AMMA platform. Three of these components: AM3, AMW, and ATL are available as GMT components. All these tools are built on top of the Eclipse Modeling Framework (EMF).

AM3

The goal of AM3 (Atlas MegaModel Management) is to provide a practical support for **modeling in the large**, i.e. managing global resources in the field of MDE. These global resources are usually heterogeneous and distributed. To access them without increasing the accidental complexity of MDE, we need to invent new ways to create, store, view, access, and modify the global entities that may be involved in developing a solution. To this end, the notion of **megamodel** (i.e. a model which elements are themselves models) is being used. In order to achieve this overall goal, AM3 provides a set of tools and artifacts that implement our **Global Model Management** (GMM) approach which is based on the concept of "megamodel".

AMW

The AMW (Atlas Model Weaver) component supports the creation of different kinds of **links between model elements**. The links are saved in a weaving model. This weaving model conforms to an **extensible weaving metamodel**. Weaving models can be used in different application scenarios, such as tool interoperability, transformation specification, traceability, model merging.

TCS

TCS (Textual Concrete Syntax) is a DSL (Domain Specific Language) for the specification of **Textual Concrete Syntaxes** in MDE.

KM3

KM3 (Kernel Meta MetaModel) is a neutral language to **write metamodels**. There is an evolutive **library of open source metamodels** written in KM3 called: AtlanticZoo. This library contains over 230 KM3 metamodels. This collection of KM3 metamodels is intended for experimental purposes. On the same site there are also a number of "**mirror zoos**" containing metamodels written in other languages like MOF, UML, Prolog, VB, SQL, Microsoft DSL Tools, GME, ASM, etc.

Additional ATL Resources

- **GMT/ATL Web Site**: http://www.eclipse.org/gmt/atl/
- **ATL Reference manuals**: User Manual, Starter Guide, Installation Guide and the ATL Virtual Machine Specification (http://www.eclipse.org/gmt/atl/doc/).
- **Atlas Publications**: http://www.sciences.univ-nantes.fr/lina/atl/publications/
- **ATL Wiki:** http://wiki.eclipse.org/index.php/ATL (FAQ, troubleshooter, etc.)
- **ATL Download**: http://www.eclipse.org/gmt/atl/download/
- **ATL Transformations Zoo**: http://www.eclipse.org/gmt/atl/atlTransformations/
- **ATL Mailing List**: http://groups.yahoo.com/group/atl_discussion/
- **ATL Source Code**: http://dev.eclipse.org/viewcvs/indextech.cgi/org.eclipse.gmt/ATL
- **AMMA Wiki**: http://wiki.eclipse.org/index.php/AMMA

Contact information

Send a mail to atl-contact@univ-nantes.fr to request ATL mailing-list membership or any additional information. The **AMMA/ATL web site** can be found at http://www.sciences.univ-nantes.fr/lina/atl/ and the **GMT/ATL web site**, on which ATL is released (source, binaries, documentation, examples, wiki, bugzilla, etc.), is located at http://www.eclipse.org/gmt/atl/.

The Atlas Transformation Language (ATL) project

Transforming models with ATL

ATLAS Group
INRIA & LINA (University of Nantes)

GMT/ATL project website: http://www.eclipse.org/gmt/atl/

Contacts: Frédéric Jouault – frederic.jouault@univ-nantes.fr
Freddy Allilaire - freddy.allilaire@univ-nantes.fr

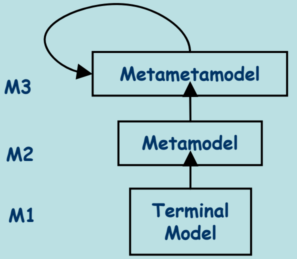
ATL Project Goals

ATL (Atlas Transformation Language) is a model transformation language and toolkit developed by the Atlas group (INRIA & LINA). In the field of Model-Driven Engineering (MDE), ATL provides ways to produce a set of target models from a set of source models.

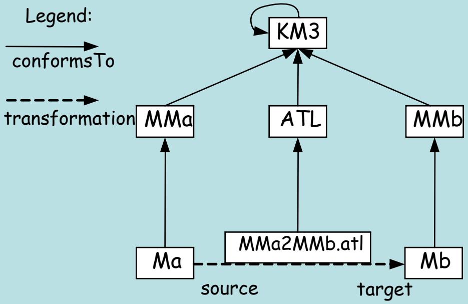
Developed on top of the Eclipse platform, the ATL Integrated Development Environment (IDE) provides a number of standard development tools (syntax highlighting, debugger, etc.) that aim to ease the development of ATL transformations.

Principles

- A **model transformation** is the automatic creation of target models from source models.
- Model transformation is not only about M1 to M1 transformations (e.g. promotion from M1 to M2).



Operational Context of ATL



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 metametamodel (e.g. KM3, MOF, Ecore).

ATL Language Overview

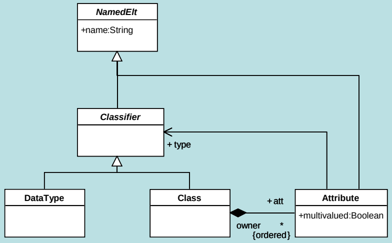
ATL is a **hybrid** transformation language. It contains a mixture of **declarative** and **imperative** constructs. The encouraged style is declarative.

ATL transformations are **unidirectional**, operating on read-only source models and producing write-only target models. A bidirectional transformation is implemented as a couple of transformations: one for each direction. During the execution of a transformation the source model may be navigated but changes are not allowed to it. The target model cannot be navigated. Source and target models for ATL may be expressed in the XMI OMG serialization format. Source and target metamodels may also be expressed in XMI or in the more convenient KM3 notation.

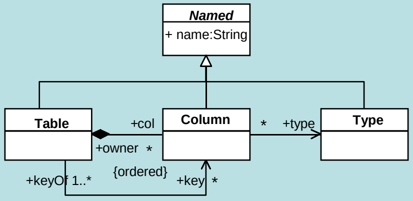
An ATL transformation can be decomposed into three parts: a header, helpers and rules. The **header** is used to declare general information such as the module name (this is the transformation name: it must match the file name), the source and target metamodels and imported libraries. **Helpers** are subroutines (based on OCL) that are used to avoid code redundancy. **Rules** are the heart of ATL transformations because they describe how target elements (based on the target metamodel) are produced from source elements (based on the source metamodel). They are made up of bindings, each one expressing a mapping between a source element and a target element.

ATL Transformation Example: Class to Relational

The transformation creates tables and columns from classes and attributes.



The source metamodel *Class* is a simplification of class diagrams.



The target metamodel *Relational* is a simplification of the relational model.

ATL declaration of the transformation:

module Class2Relational;
create Mout : Relational **from** Min : Class;

ATL Rule Sample:

```
rule Class2Table {
        from
                c : Class!Class
        to
                t : Relational!Table (
                        name <- c.name,
                        col <- c.attr->select(e |
                                not e.multiValued
                        )->union(Sequence {key}),
                        key <- Set {key}
                ),
                key : Relational!Column (
                        name <- 'Id'
                )
}
```

Metamodel name

A Table is created for each Class

The name of the Table is the name of the Class

Model Element Name

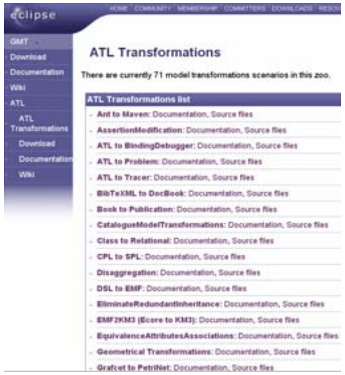
The columns of the table correspond to the single-valued attributes of the class

Each Table owns a key containing a unique identifier

ATL Transformations Zoo

The ATL component hosts an initial experiment for building a collection of model transformations. This collection of transformations already offers an interesting landscape of application domains of model transformations. It represents an interesting starting point for studying what should be the content of **transformation libraries** and helps identifying some of the issues large scale transformation libraries will have to face. More than 70 scenarios are available. A **classification** of the ATL Transformations has been created:

- Semantic Bridges
- Bridges to Graphical Representations
- From Tool to Tool Transformations
- Calculation Transformations
- Miscellaneous Transformations



ATL Transformations

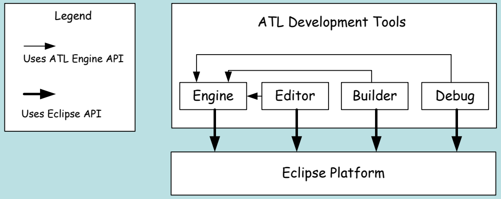
There are currently 71 model transformations scenarios in this zoo.

ATL Transformations list

- **Ant to Maven**: Documentation, Source files
- **AssertionModification**: Documentation, Source files
- **ATL to BindingDebugger**: Documentation, Source files
- **ATL to Problem**: Documentation, Source files
- **ATL to Tracer**: Documentation, Source files
- **BibTeXML to DocBook**: Documentation, Source files
- **Book to Publication**: Documentation, Source files
- **CatalogueModelTransformations**: Documentation, Source files
- **Class to Relational**: Documentation, Source files
- **CPL to SPL**: Documentation, Source files
- **Disaggregation**: Documentation, Source files
- **DSL to EMF**: Documentation, Source files
- **EliminateRedundantInheritance**: Documentation, Source files
- **EMF2KM3 (Ecore to KM3)**: Documentation, Source files
- **EquivalenceAttributesAssociations**: Documentation, Source files
- **Geometrical Transformations**: Documentation, Source files
- **Grafcet to PetriNet**: Documentation, Source files

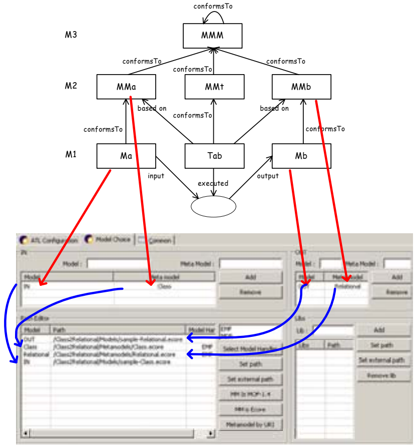
ATL Toolkit

ATL is accompanied by a set of tools built on top of the **Eclipse** platform. **ADT** (ATL Development Toolkit) is composed of the **ATL transformation engine** (Engine block) and the **ATL Integrated Development Environment** (IDE: Editor, Builder and Debug blocks).



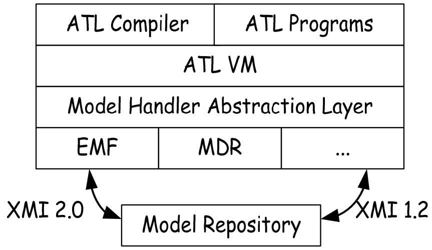
ATL Engine

The **ATL engine** (see right) is responsible for dealing with core ATL tasks: **compilation** and **execution**. ATL transformations are compiled to programs in a specific byte-code. The 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 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.



Debugging

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

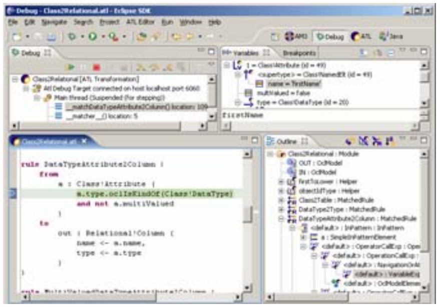


Editing

The **ATL editor** (see below) supports syntax highlighting, error reporting, and outline view (i.e. tree-based representation of the ATL program).

Building and Launching ATL Transformations

The **ATL compiler** is automatically called on each ATL file in all ATL projects during the Eclipse build process. By default, this process is triggered when a file is modified (e.g. saved). **Executing** an ATL transformation requires that 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 (see left). The ATL engine delegates reading and writing models to the underlying model handler. For instance, when EMF is used source and target models must be in EMF XMI 2.0. More complex transformation scenarios can use other kinds of formats (e.g. XML, textual).



Class2Relational [ATL Transformation]
Atl Debug Target connected on host localhost port 6060
Main thread (Suspended (for stepping))
__matchDataTypeAttribute2Column() location: 109
__matcher__() location: 5

```
rule DataTypeAttribute2Column {
    from
        a : Class!Attribute (
            a.type.oclIsKindOf(Class!DataType)
            and not a.multiValued
        )
    to
        out : Relational!Column (
            name <- a.name,
            type <- a.type
        )
}
```