An Eclipse-Based IDE for the ATL Model Transformation Language

Jean BÉZIVIN [1], Frédéric JOUAULT[1] [2] and Patrick VALDURIEZ[1]

[1] ATLAS Group (INRIA & LINA)

[2] TNI-Valiosys Company

— ATLAS GDD —

RAPPORT DE RECHERCHE

N° 04.08

Novembre 2004

LINA, Université de Nantes – 2, rue de la Houssinière – BP 92208 – 44322 NANTES CEDEX 3
Tél. : 02 51 12 58 00 – Fax. : 02 51 12 58 12 – http://www.sciences.univ-nantes.fr/lina/

Jean BÉZIVIN, Frédéric JOUAULT and Patrick VALDURIEZ

An Eclipse-Based IDE for the ATL Model Transformation Language

10 p.

An Eclipse-Based IDE for the ATL Model Transformation Language

Jean BÉZIVIN, Frédéric JOUAULT
and Patrick VALDURIEZ

Résumé

We are currently building a model management platform on top of Eclipse/EMF. The basic prin-ciple on which this platform is based is the con-sideration of models as first-class entities. One of the main elements of this platform is an engine for the ATL model transformation language. This paper presents some characteristics of the IDE developed in support of ATL within the Eclipse GMT (General Model Transformer) project [7].

Catégories et descripteurs de sujets : D.2.12 [**Software**]: Software Engineering - Data Mapping

Termes généraux : MDA, model engineering, model transformation

1 Introduction

ATL (ATLAS Transformation language) is a metamodel-based transformation DSL (Domain Specific Language) intended to be compliant with the OMG/QVT recommendation when this is accepted. The scope of ATL is however not limited to the OMG set of recommendations as the intention is to cover other technical spaces as well.

A program in ATL is considered as a model, taking a model as input and producing a model as output. Multiple input and output parameters are accepted, but will not be considered here for the sake of simplicity in the presentation.

This paper describes the current status of the project and presents some basic principles on which it is built. A more detailed report will be available later at [2].

2 Model Engineering

In November 2000 the OMG proposed a new approach to interoperability named MDA™ (Model-Driven Architecture) [12]. MDA is one example of a much broader approach known as Model Driven Development (MDD), encompassing many popular research trends like generative programming, domain specific languages, model-integrated computing, model management and much more.

The basic assumption in MDD is the consideration of models as first class entities. A model is an artifact that conforms to a metamodel and that represents a given aspect of a system. These relations of conformance and representation are central to model development [4]. A model is composed of model elements. The metamodel describes the various kinds of contained model elements and the way they are arranged, related and constrained. A language intended to define metamodels and models is called a metametamodel. The OMG/MDA proposes the MOF (Meta Object Facility) as such a language. The Eclipse/EMF metametamodel is called Ecore and is compatible with MOF 2.0. This language has the power of UML class diagrams complemented by the OCL assertion and navigation language.

There are other representation systems that may also offer, outside the MDA strict boundaries, similar model engineering facilities. We call them technical spaces [9]. They are often based on a three level organization like the metametamodel, metamodel and model of the MDA. One example is grammarware [8] with EBNF, grammars and programs, but we could also consider XML documents, Semantic Web, DBMS, ontology engineering, etc. A Java program may be viewed as a model conforming to the Java grammar. As a consequence we may consider strict (MDA)-models, i.e. MOF-based like a UML model but also more general models like a source Java program, an XML document, a relational DBMS schema, etc.

3 Open Platforms for MDD

The basic principle of model as first class entities has not only the advantage of conceptual simplicity. It also leads to clear architecture, efficient implementation, high scalability and good flexibility. As part of several projects, open source platforms are being built with the intention to provide high interoperability not only between recent model-based tools like ATL, but also between other legacy tools.

Connected to such an open platform, tools will exchange models. But tools will also be handled as models. A tool implements a number of operations. Each operation is also represented as a model. Furthermore an operation may also have input and output parameters, each parameter being itself considered as a model. The interoperability platform may be organized as an intelligent model exchange system according to several principles and protocols like the classical software bus or even more advanced P2P architectures. To facilitate this exchange, the platform may use open standards like XMI (XML model Interchange), CMI (CORBA Model Interchange), JMI (Java Model Interchange), etc.

The advantage of using a model-based platform is that it allows many economies of scale. For example model and metamodel repositories may handle efficient and uniform access to these models, metamodels and their elements in serialized or any other mode. Transactional access, versioning and many other facilities may be also offered, whatever the kind of considered model: executable or not, product or process, transformation, business or platform, etc.

To identify a given model we could use a three level scheme ***model:a/b/c*** where *a* is the metametamodel, *b* the metamodel and *c* the model. This could be seen either as an extension of

MIME types or as a basis for a URI-naming style. This has the advantage of allowing several representation systems to be considered, each being identified by its own metametamodel. An example could be ***model:Ecore/UML/Banking***.

To be handled by the platform, a model component should be named and typed. Another issue for these model components is packaging. A model component should be encapsulated in a standard way so that the various tools may easily interoperate by sending and receiving these components. The current encapsulation standard for the ATL tool is based on the OMG RAS recommendation (Reusable Asset Specification) [11].

Finally the platform will also need to know the available tools and services. For this we use the concept of a megamodel, i.e. a model which elements represent models, metamodels, metametamodels, services, tools, etc. As any model, the megamodel is based on a precise metamodel. It records the main relations between its different entities and also the metadata that applies on them. One typical relation is the association of a metamodel to a given model or the fact that a metamodel is an extension or a version of another one. Applying a given transformation to a model may imply finding the metamodel for this transformation and an available transformation tool compatible with this metamodel. If none is found, one possibility would be to transform the transformation model itself, if this is possible.

Each time a tool is plugged into a given platform, it registers for itself and for its supported services. If this is a legacy tool, there is an adapter to the platform in charge of interfacing and handling these tasks. The local megamodel records the local context. Global views of local megamodels may also be used by local platforms.

4 Model Transformation

One of the most important operations in MDD is model transformation. Since a transformation program *Mt* is considered as a model, it must conform to a given metamodel *MMt*. The transformation *let Mt Ma:MMa* $\rightarrow$ *Mb:MMb* produces a model *Mb* conforming to metamodel *MMb* from a model *Ma* conforming to metamodel *MMa*. As we can see the metamodels act as types.

A transformation is a typed operation, but it has also some invariants, pre conditions and post conditions. All these are expressed in term of the metamodels, of constraints applying to the metamodels or to the models.

As a side effect, a transformation may also produce a trace model of the transformation execution. Since there is no standard trace, this model is also based on some trace metamodel, either generic or specific to a given transformation. There are many different usages for traces (code synchronization, incremental transformations, etc.) and different tools will deal with these traceability data in our model management platform.

5 Tool Integration

Model transformation has an applicability scope much larger than mere UML refactoring and transformation from UML to Java or EJB. We have the conviction that metamodel-based technology could be most useful in other areas as well, for example in facilitating the bridging between legacy tools.

Let us take one example to illustrate this. In [3] we describe a transformation experiment where a UML activity diagram is used to drive the Microsoft Project tool. In order to achieve this, we have first to define a source metamodel, then a target metamodel and then a transformation from source to target. The source metamodel is part of the standard OMG UML 1.x metamodel so there is no difficulty in getting access to it. By itself Microsoft Project is obviously not delivered with a metamodel and so we have to design one from scratch. Then the concepts from the source have to be matched to the concepts of the target. Now we have the final task of building injectors and extractors, i.e. drivers from source and to target tools. For the source tool, this should be straightforward because the source model naturally lies inside the MDA technical space. Unfortunately this is not so simple because activity diagrams representation may be based on UML profiles in some UML tools and we need to do a conversion, which is an additional complication. On the target side, the MS-Project metamodel contains concepts like *Resource*, *Task*, *Project*, *Assignment*, etc. Creating an MSProject planning from a UML activity model corresponds to writing a transformation which may be complex because it has for example to handle ordering dependencies. The ultimate step consists in writing an MSProject extractor driver, for example from XMI to a proprietary or XML-based format. There are several generic techniques to design such drivers and to

produce the corresponding components. Injector and extractor to various technical spaces are intended to be made available in the ATL library of components together with a wide collection of metamodels and ATL standard transformations.

6 The ATL environment

ATL is a language, an engine, but more than that, it is a global environment for preparing, checking, testing, executing, debugging, etc. transformation operations.

6.1 The OMG Recommendation

At the time of this writing the OMG has not yet finalized the result of the MOF/QVT request for proposal [10] . The architecture of the ATL engine and IDE has been made sufficiently flexible to accommodate possible forthcoming changes in the OMG final MOF/QVT recommendation.

6.2 The Language

A complete description of the ATL language is out of scope in this paper. A short sample is provided in Appendix A. This is a fragment of a typical program transforming a UML model into a Java program.

6.3 The Environment

Besides the ATL engine we need tools to prepare transformations, to browse libraries of existing transformations, to check transformations, etc.

We have therefore implemented an ATL specific code editor with outline and a source-level transformation debugger [1].

6.4 The Architecture

The architecture of the ATL framework has been defined to allow maximum upper and lower adaptability. Lower level adaptability has already been proven by porting the engine from the Sun MDR/NetBeans to the Eclipse/EMF environment. Taking into account other underlying platforms should not be a big challenge. This means that ATL may be easily integrated in various open MDD platforms. Upper level adaptability was also essential because ATL was developed before the OMG MOF/QVT recommendation was finalized. We give below some information on the organization of the ATL IDE.

6.4.1 Transformation Virtual Machine

The concept of virtual machine has been generally useful for portability. We have used it here with the additional advantages of defining generic access operation to model elements. In the same way the Java Virtual Machine (JVM) instruction set can directly deal with objects, the ATL Transformation Virtual Machine (TVM) directly handles model elements. The present prototype version is a stack machine with less than twenty instructions. It is built on top of a Java-based model repository abstraction layer, which is the key to the current version's lower level adaptability.

7 Related Work

There are currently plenty of activities going on in the domain of model transformation. Several other MOF/QVT compliant languages are being defined, with the hope that the fact that they are based on some common metamodel may facilitate "internal" interoperability between languages written in these different languages. An important distinction between these proposals is how they deal with "external" interoperability i.e. with languages defined outside the scope of MOF/QVT like XSLT, XQuery, Perl, etc. In some cases it would be convenient to use a composite transformation made for example of an XSLT part and an ATL part.

8 Conclusion

We have tried to summarize in this paper the main characteristics of the project. The ATL IDE is now ready in its first version on the Eclipse GMT project [7]. It will be improved for performance and functionality in the coming months. We see it as a tentative contribution to set up an experimental platform in model engineering. Exploring the promises and potentialities of MDA is important. How can we encapsulate and reuse knowledge and expertise on transformation patterns for example is a currently open question.

In addition to the ATL IDE we are building other Eclipse-based model engineering tools that will enrich our model management platform named AMMA (ATLAS Model Management Architecture). One of the most important is the "model weaver" that may be used as a front-end to the ATL model transformer. Other tools envi-

sioned are model-editors and model-checkers for example. However, the main objective is to set up rich libraries of transformation components together with their metamodels and the corresponding extractors and injectors. The wide availability of such components is a prerequisite for demonstrating potential reusability in the domain of model transformation.

Acknowledgements

This work has been supported by an IBM Eclipse grant and by the TNI-Valiosys company. Many contributors have helped to develop the IDE for ATL: Freddy Allilaire, Tarik Idrissi, Gaetan Le Colleter, Xavier Méhaut, Christophe Monti and many others.

About the Authors

Patrick Valduriez is director of research at INRIA, heading the newly created ATLAS group at the University of Nantes. Jean Bézivin is professor at the University of Nantes, leading the model management project in the ATLAS group. Frédéric Jouault is a PhD student at the University of Nantes and the main designer of the ATL engine and environment. Frédéric is supported in this work by the TNI-Valiosys company, Brest, France. Authors can be reached at the following address: LINA, Université de Nantes, 2, rue de la Houssinière, 44072 Nantes cedex 03 and by e-mail at {bezivin, jouault, valduriez}@lina.univ-nantes.fr.

References

[1] Allilaire, F., Idrissi, T. ADT: Eclipse Development Tools for ATL. EWMDA-2 , Canterbury, England, 2004 http://www.cs.kent.ac.uk/projects/kmf/mdaworkshop/submissions/Allilaire.pdf .

[2] ATL, ATLAS Transformation Language Reference site http://www.sciences.univ-nantes.fr/lina/atl/

[3] Bézivin, J. & Breton, E. Applying the basic principles of model engineering to the field of process engineering. Novatica N° 171, Software Process Technologies http://www.upgrade-cepis.org/issues/2004/5/up5-5Bezivin.pdf

[4] Bézivin, J. In search of a Basic Principle for Model Driven Engineering, Novatica/Upgrade, Vol. V, N°2, (April 2004), pp. 21-24, http://www.upgrade-cepis.org/issues/2004/2/up5-2Presentation.pdf

[5] Bézivin, J., Gérard, S. Muller, P.A., Rioux, L. MDA Components: Challenges and Opportunities, Metamodelling for MDA, First International Workshop, York, UK, (November 2003), http://www.cs.york.ac.uk/metamodel4mda/onlineProceedingsFinal.pdf

[6] Booch G., Brown A., Iyengar S., Rumbaugh J., Selic B. The IBM MDA Manifesto The MDA Journal, May 2004, http://www.bptrends.com/publicationfiles/05-04%20COL%20IBM%20Manifesto%20-%20Frankel%20-3.pdf

[7] GMT, General Model Transformer Eclipse Project, http://www.eclipse.org/gmt/

[8] Klint, P., Lämmel, R. Kort, J., Klusener, S., Verhoef, C., Verhoeven, E.J. Engineering of Grammarware. http://www.cs.vu.nl/grammarware/

[9] Kurtev, I., Bézivin, J., Aksit, M. Technical Spaces: An Initial Appraisal. CoopIS, DOA'2002 Federated Conferences, Industrial track, Irvine, 2002 http://www.sciences.univ-nantes.fr/lina/atl/www/papers/PositionPaperKurtev.pdf

[10] OMG The MOF/QVT Queries, Views, Transformations request for proposal www.omg.org

[11] OMG, Reusable Asset Specification, Final Adopted Specification, PTC/04/06/06, www.omg.org, (June 2004)

[12] Soley, R. & the OMG staff MDA, Model-Driven Architecture, (November 2000), http://www.omg.org/mda/presentations.htm

Appendix A: ATL sample

The following is a sample ATL program. For a more complete description, an ATL tutorial will be available at [1].

```
module UML2JAVA;
create OUT :  JAVA from IN : UML;

uses strings;

helper context UML!ModelElement def: isPublic() : Boolean =
        self.visibility = #vk_public;

helper context UML!Feature def: isStatic() : Boolean =
        self.ownerScope = #sk_static;

helper context UML!Attribute def: isFinal() : Boolean =
        self.changeability = #ck_frozen;

helper context UML!Namespace def: getExtendedName() : String =
        if self.namespace.oclIsUndefined() then
                ''
        else if self.namespace.oclIsKindOf(UML!Model) then
                ''
        else
                self.namespace.getExtendedName() + '.'
        endif endif + self.name;

rule P2P {
        from e : UML!Package (e.oclIsTypeOf(UML!Package))
        to out : JAVA!Package (
                name <- e.getExtendedName()
        )
}

rule C2C {
        from e : UML!Class
        to out : JAVA!JavaClass (
                name <- e.name,
                isAbstract <- e.isAbstract,
                isPublic <- e.isPublic(),
                package <- e.namespace
        )
}

rule D2P {
        from e : UML!DataType
        to out : JAVA!PrimitiveType (
                name <- e.name,
                package <- e.namespace
        )
}

rule A2F {
        from e : UML!Attribute
        to out : JAVA!Field (
                name <- e.name,
                isStatic <- e.isStatic(),
                isPublic <- e.isPublic(),
                isFinal <- e.isFinal(),
                owner <- e.owner,
                type <- e.type
        )
}

rule O2M {
        from e : UML!Operation
        to out : JAVA!Method (
                name <- e.name,
                isStatic <- e.isStatic(),
                isPublic <- e.isPublic(),
                owner <- e.owner,
                type <- e.parameter->select(x|x.kind=#pdk_return)->asSequence()->first().type,
                parameters <- e.parameter->select(x|x.kind<>#pdk_return)->asSequence()
        )
}

rule P2F {
        from e : UML!Parameter (e.kind <> #pdk_return)
        to out : JAVA!FeatureParameter (
                name <- e.name,
                type <- e.type
        )
}
```

An Eclipse-Based IDE for the ATL Model Transformation Language

Jean BÉZIVIN, Frédéric JOUAULT and Patrick VALDURIEZ

Résumé

We are currently building a model management platform on top of Eclipse/EMF. The basic prin-ciple on which this platform is based is the con-sideration of models as first-class entities. One of the main elements of this platform is an engine for the ATL model transformation language. This paper presents some characteristics of the IDE developed in support of ATL within the Eclipse GMT (General Model Transformer) project [7].

Catégories et descripteurs de sujets : D.2.12 [**Software**]: Software Engineering - Data Mapping

Termes généraux : MDA, model engineering, model transformation

LINA, Université de Nantes
2, rue de la Houssinière
B.P. 92208 — F-44322 NANTES CEDEX 3