Using a Virtual-Machine Strategy to Enhance Model Transformation Experiences

E. Victor Sánchez Rebull[1], José Luis Roda García[2],
Adolfo Sánchez-Barbudo Herrera[1], and Antonio Estévez García[1]

[1] Open Canarias, S.L., C/. Elías Ramos González, 4 - Oficina 304
38001 Santa Cruz de Tenerife, Spain
{vsanchez,adolfosbh,aestevez}@opencanarias.com
http://www.opencanarias.com
[2] Universidad de La Laguna, La Laguna, Spain
jlroda@ull.es

Abstract. Among the cornerstones in Model-Driven Engineering we find model transformations. QVT being one of the most relevant alternatives, its consolidation has lasted longer than expected. Since support to QVT execution has been among the goals of this doctoral thesis from the beginning, a virtual-machine strategy is being adopted as a robustness measure to decouple variations of its specification from implementation. This virtual machine comprises a middle layer supported by a byte-code model transformation language named Atomic Transformation Code (ATC). This decision motivates a set of hypotheses: the isolating capabilities of adopting a virtual machine perspective might prove substantially helpful for interoperability, flexibility, simplification, maintainability and simultaneous availability of model transformation engine implementations. In this paper, some of these hypotheses will be discussed in terms of the ATC language's design and implementation principles, and its application on industrial case-study experiments.

1 Introduction

Model transformations are of vital importance for embracing Model-Driven Engineering (MDE). Practitioners tend to look at the MDA framework's MOF 2.0 QVT as the official model transformation specification. But QVT has released version 1.0 only recently [OMG08] in a process that has taken longer than expected. This delay may have possibly hindered the adoption rate of the MDA set of standards. In this respect other available, proprietary or open, languages may potentially have played a valuable fulfilling role.

The purpose of this doctoral thesis was to experiment and find novel ways of approaching model transformations. Soon enough the focus had centered around QVT. But the QVT specification has been steadily changing its contents in successive updates. It will probably continue to do so, even now that the 1.0 status has been reached. Directly supporting any such an unstable language seems too rigid and hard to maintain. As a better alternative, a Virtual Machine (VM) architecture has been considered instead. That is, a middleware layer to decouple

the supported language definition from the underlying implementation, aimed at improving architectural flexibility and robustness against language evolution.

This intermediate layer comprises a model transformation engine, driven by a byte-code model transformation language. This language, named Atomic Transformation Code (ATC), will remain independent from the above indirectly supported languages, thus opening the spectrum of (simultaneous) available languages. In order to be executable in the VM-based environment, mechanisms that translate them into the equivalent byte-code must be provided.

In this paper the deployment of a VM model transformation approach in terms of ATC will be discussed. We will describe how this strategy can achieve execution of the QVT specification, and identify additional usages by examining real translation experiences. The paper is structured as follows: Sect. 2 introduces the Virtual Machine concept applied on model transformation languages, and offers an overview of the ATC byte-code language, Sect. 3 describes two practical cases of languages supported via this VM, Sect. 4 covers the related work, Sect. 5 closes with a summarized survey on present achievements and future work.

2 Introducing a Virtual Machine Approach

This doctoral thesis originated from the company Open Canarias's interest to produce a running model transformation engine implementation. A strong commitment to standards lead us to evaluate the QVT specification. But what the specification would finally look like was unguessable and full of uncertainty back in 2005. As a robustness measure, a Virtual-Machine (VM) strategy was adopted to isolate efforts put in the execution engine implementation from potential updates in the QVT definition. Building a VM on top of existing language and engine solutions risked from being too constrained by their own limitations.

The idea of using a VM intermediate layer has been recently gaining momentum [SRER08]. Portability may be one of the main reasons behind this trend. Also, the coding of a particular domain-specific language might be faster and easier than building complete support from scratch.

Additionally, this abstraction decoupling favors the assignment of overall responsibilities and independent evolution of the layers involved. The engine has therefore been built from scratch in a bottom-up approach that connects with a metamodeling infrastructure to abstract up model transformation fundamental principles[3] in a modular fashion. The engine would be driven by a byte-code model transformation language, called ATC, that would encode these fundamental activities in its instruction set. The VM layering is illustrated in Fig. 1.

2.1 Atomic Transformation Code

ATC is an imperative, structured, dynamically typed language, with certain object-orientation traits. ATC's instruction set basically comprises the semantics of a general-purpose programming language enriched with a complement of

[3] Which are the *creation*, *modification* and *elimination* of models and model elements.

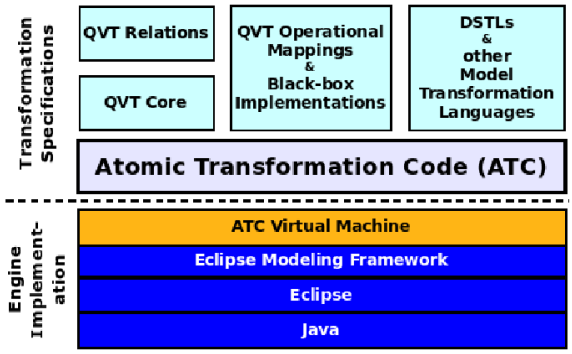

Fig. 1. Layered architecture of the ATC-based VM implementation. Above ATC lie the types of languages suitable to receive indirect support. Below, the engine implementation layers are stacked. Java permeates the whole engine implementation implicitly.

indivisible, combinable primitives (called atoms) carefully designed to provide model transformation functionality at a low abstraction level.

Among these, we find `AtcGetFeature` to query the value of a model element's feature, or `AtcCreateElement` to instantiate a desired type defined in a metamodel. Many of these instructions are explained in [EPSR06]. Implementation of constructs specific to model transformations are based on the API of the well known Eclipse's metamodeling infrastructure, Eclipse Modeling Framework.

Obviously, any formal model transformation language, not just a standard specification, should in principle be translatable into the ATC language. This imposes some fundamental requirements upon the design and capabilities of ATC. In particular, ATC must be *exhaustive* to cover all possible transformation mechanisms. It must also be *efficient* enough to be of any practical use.

Above the byte-code language, mechanisms to translate higher-level languages (including QVT) into ATC must be provided for this whole model transformation infrastructure to be capable of executing their instances. Translation approaches attempted in this doctoral thesis include textual parsing and generation of translated ATC model transformations, or creation of model to model transformations to translate between languages, or a mixture of both.

3 Model Transformation Translation Real Cases

The two cases of automated model transformation translation into ATC illustrated here correspond to the QVT suite of languages, and to a lightweight DSTL developed internally at Open Canarias named Model-Template Transformation Language (MTTL) [AGF07]. Both approaches involve model transformations, in what is called Platform-Independent to Platform-Specific Transformations (PIT-PST) in [BFJ+03]. This is possible because ATC transformation instances are themselves represented as models. Both cases successfully certify the feasibility of the adopted ATC-based VM approach (see [RAGRG07]).

The QVT Specification The MOF 2.0 QVT specification [OMG08] defines three languages, two declarative, *Relations* (QVTr) and *Core* (QVTc), and an imperative third one, *Operational Mappings* (QVTo). The QVT specification extends upon the Object Constraint Language (OCL).

The translation procedure is explained as follows. Textual transformations of QVTc and QVTo are parsed to generate equivalent QVT models. These are then translated into one or several ATC equivalent models via a model transformation. Support for QVTr via automated translation into QVTc will come next.

Although ATC is not meant for direct human use, the *QVTo2ATC* and *QVTc2ATC* model transformations that transparently convert QVT models into ATC (see Fig. 2) have both been written as a set of coordinated ATC transformations. The *OCL2ATC* module, along with support to common MOF constructs, are being reused among both QVTo and QVTc.

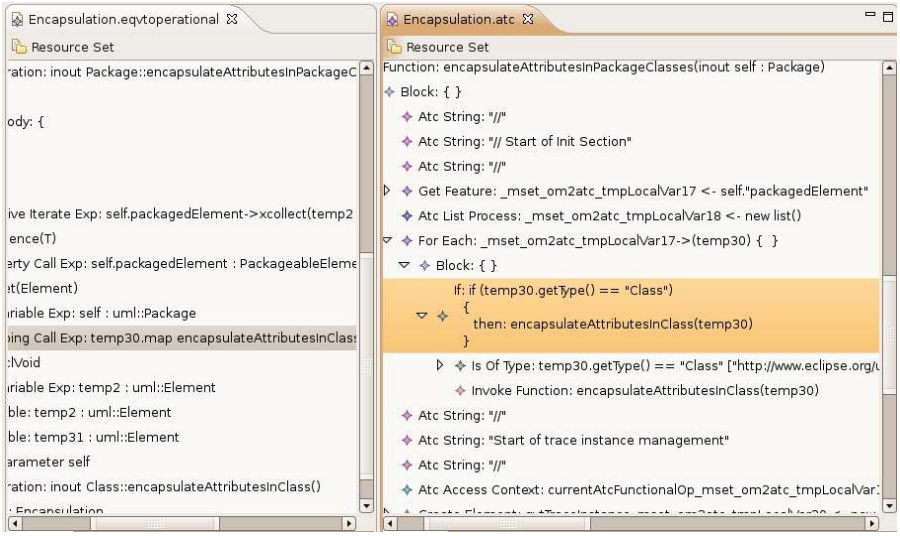

Fig. 2. A QVTo sample (on the left) and its translated ATC model (on the right).

This scenario has led to improvements on the ATC language in terms of scalability. Imports between transformation components has been installed and blind virtual invocations have been incorporated to the language. It is worth mentioning the semantics of OCL's `PropertyCallExp` which refers to a pair *model element - property* without considering it for query (read) or value change (write) beforehand. It is unexpectedly more atomic and thus composable than ATC's imperative `GetFeature` and `SetFeature` complementary instructions. Thus, the context-free principle cannot be preserved when translating such an expression.

MTTL as a Lightweight DSTL The second translation support example corresponds to Model-Template Transformation Language (MTTL), a lightweight language composed of five instructions, such as `select` or `remove`. It is specifically designed to handle feature models that define Software Product Lines' variability. More information can be found at [AGGRG07].

The distinctive nature of MTTL is that its instances are programmed as models. Then, a direct translation of MTTL models into ATC via a model transformation (again written in ATC), is enough, thus eluding having to maintain any parsing module. This approach is important because it provides a way to create personalized DSTLs in a considerably short time.

MTTL motivated the inclusion of the *feature provider* concept in ATC, a flexible mechanism that is capable of mimicking the semantics of the aforementioned OCL's `PropertyCallExp` construct. Support for MTTL has reached completion.

4 Related Work

The only other byte-code model transformation language we know of, is the Atlas Group's ATL-VM [JK06], which has been conceived during the same period as ATC. The main difference with ATC is that ATL-VM incorporates OCL in its core to specify model queries and constraints. OCL being declarative, it is independently translated into the imperative ATC code in our case.

5 Conclusions: Past, Current and Future Work

A Virtual Machine implementation strategy has been adopted to give execution support to model transformation languages. Driven by a byte-code named Atomic Transformation Code, it is successfully progressing towards becoming a viable industrial solution. A lot of functionality is encoded in the ATC instruction set (which includes instructions that tap on a metamodeling infrastructure), so it pays off to consider only to provide means to translate into such an executable byte-code language than to start an independent language implementation, above all with lightweight DSTLs with reduced instruction sets such as MTTL, but also applicable to any language specification, such as MDA's QVT.

A VM solution delivers an ideal environment in which to experiment with model transformation principles, techniques, paradigms or languages, in academic or professional environments. It succeeds in easing the development and maintenance of model transformation language executions, and to enhance their usage experience by sharing and reusing the metamodeling infrastructure, along with other facilities, related with debugging, textual editors, etc. Also simultaneous execution available for different languages may reduce the risks derived from sticking to a particular technology and promote interoperability.

For the two crucial principles surrounding ATC, which are exhaustivity and efficiency, the first is being dealt with incrementally, which is possible thanks to the VM layout, and the second depends highly on the engine performance. This is being boosted by compiling ATC models to native Java for fastest execution.

No development about indirect debugging support has started, although several approaches have been considered. The fact that ATC is imperative means that languages whose definition is based on declarative paradigms will be initially harder to receive support, although the design and implementation of a Pattern-Matching Virtual-Machine language built on top of ATC could be worth trying.

ATC is currently in a very advanced development state, as is its execution support for QVT. As for the future, ATC must still improve in its scalability capabilities, perhaps by enhancing the language with inheritance principles.

6 Acknowledgements

Research co-funded by the *MEC* (PTR1995-0928-OP) and the *Plan Nacional de Investigación Científica, Desarrollo e Innovación Tecnológica* of the *Ministerio de Industria, Turismo y Comercio* (FIT-340000-2007-162).

Thanks to Orlando Avila-García and Víctor Roldán for their selfless deep contributions to this thesis.

References

[AGF07] Orlando Avila-García and Marcos Didonet Del Fabro. AMW use case: Mapping features to models. Apr 2007. URL `http://www.eclipse.org/gmt/amw/usecases/softwareproductline/`.

[AGGRG07] Orlando Avila-García, Antonio Estévez García, E. Victor Sánchez Rebull, and Jose Luis Roda García. Using software product lines to manage model families in model-driven engineering. In *SAC 2007: Proceedings of the 2007 ACM Symposium on Applied Computing, track on Model Transformation*, pages 1006–1011. ACM Press, Mar 2007. ISBN 1-59593-480-4.

[BFJ+03] Jean Bézivin, Nicolas Farcet, Jean-Marc Jézéquel, Benoit Langlois, and Damien Pollet. Reflective Model Driven Engineering, 2003. Available at http:/www.lina.sciences.univ-nantes.fr/Publications/2003/BFJLP03.

[EPSR06] Antonio Estévez, Javier Padrón, E. Victor Sánchez, and José Luis Roda. ATC: A low-level model transformation language. In *MDEIS 2006: Proceedings of the 2nd International Workshop on Model Driven Enterprise Information Systems*, May 2006.

[JK06] Fréderic Jouault and Ivan Kurtev. On the architectural alignment of ATL and QVT. In *1st track on MT - SAC 2006: Proceedings of the Symposium on Applied Computing*. ACM Press, Apr 2006.

[OMG08] OMG. MOF 2.0 Query/View/Transformation specification. Technical Report formal/2008-04-03, Apr 2008.

[RAGRG07] E. Victor Sánchez Rebull, Orlando Avila-García, José Luis Roda, and Antonio Estévez García. Applying a Model-Driven Approach to Model Transformation Development. In *Proceedings of the 3rd International Workshop on Model Driven Enterprise Information Systems, ICEIS*, Jun 2007.

[SRER08] E. Victor Sánchez, Víctor Roldán, Antonio Estévez, and José Luis Roda. Making a case for supporting a byte-code model transformation approach. In *Symposium on Eclipse Open Source Software and OMG Open Specifications*, Jun 2008.