Industrialization of Research Tools: the ATL Case

Hugo Brunelière, Jordi Cabot, Frédéric Jouault, Massimo Tisi, Jean Bévizin

AtlanMod, INRIA RBA Center & EMN, 4 rue Alfred Kastler, 44307 Nantes, France
{hugo.bruneliere,jordi.cabot,frederic.jouault,massimo.tisi,jean.bezivin}@inria.fr

Abstract

Research groups develop plenty of tools aimed at solving real industrial problems. Unfortunately, most of these tools remain as simple proof-of-concept tools that companies consider too risky to use due to their lack of proper user interface, documentation, completeness, support, etc that companies expect from commercial-quality level tools.

Based on our tool development experience in the AtlanMod research team, specially regarding the evolution of our ATL model transformation tool, we argue in this paper that the best solution for research teams aiming to create high-quality and widely-used tools is to *industrialize* their research prototypes through a partnership with a technology provider.

Keywords: tool, ATL, prototype, industrialization, model-driven

1. Introduction

In software engineering, many research groups work on topics directly linked to (and sometimes also inspired by) real industrial problems. This research activity usually end ups implemented in a tool that serves as proof of concept for the developed techniques.

These tools are technically good and solve real problems but unfortunately most times they are largely ignored by software practitioners. Technical quality is just one of the many factors that companies analyze when deciding whether to take the risk of adopting a new tool. Examples of other factors that influence this decision are: user support, good documentation or usability aspects. The problem is that resources of research groups are very limited and cannot be usually devoted to work on this non-core research activities (also because funding for tool development is not exactly a priority for funding agencies). Because of this, research groups risk missing the opportunity of having a large user base for their tools along with the benefits that this brings to the table (e.g. empirical validation of their research, feedback, visibility, collaboration opportunities and so on). This is specially true in emerging software engineering areas with growing industrial interest but without a dominant tool/s monopolizing the market and where some of our research tools could make a difference.

We believe that Model Driven Engineering (MDE) is one of these areas. MDE is a software engineering paradigm based on the use of models as primary artifacts of all software engineering activities. Many of these activities imply model manipulations, usually implemented as model transformations. Therefore, availability of tools for specifying and executing model transforma-

1

tions is crucial. Right now, the most popular[1] model transformation language and toolset is ATL (AtlanMod Transformation Language [1]). Regardless the technical quality of ATL (comparing technical aspects of ATL with respect to other alternative tools, as graph transformation tools [2] or QVT-based tools [3] is not the topic of this paper) we believe its non-technical features have been one of the key factors of its success.

In this sense, the goal of this paper is present our experience in the development of a long-term sustainable model for ATL and the choices (both at the technical and business model level) we have made during this process. In particular, we will emphasize our industrialization approach for ATL (through a partnership with the technology provider Obeo [4]) that has permitted us to meet the needs of our large user base while at the same time committing our resources only to the research aspects of the language. This *business model* has proven to be feasible and beneficial both for the company and for ourselves as a research group.

The rest of the paper is structured as follows. The next section introduces the ATL tooling. Section 3 comments on the challenges of developing a commercial-quality level tool while Section 4 and 5 explain our best practices to overcome these challenges at the technical and business model level, respectively. Section 6 generalizes these results to other tools and, finally, Section 7 presents some conclusions and further work.

2. Overview of ATL

This section provides some background on the history (Section 2.1) and characteristics (Section 2.2) of the ATL model-to-model transformation language [1] and highlights the complexity of the tool as it stands today 2.3.

2.1. History of ATL

ATL is a model-to-model transformation language created by AtlanMod [5], a joint research team between INRIA and Ecole des Mines de Nantes. ATL was first developed as part of the Phd Thesis of Frédéric Jouault [6] started in 2003.

Simultaneously to the definition of the language, a toolset with the same name to specify and execute ATL transformations was also implemented. This allowed the first concrete use of ATL on industrial scenarios that took place in 2004 in the context of the French project CAR-ROLL/MOTOR, in collaboration with CEA and Thales. Then, the development of the language and tooling has continued, mainly as part of two consecutive European integrated projects called MODELWARE (October 2004 - September 2006) and MODELPLEX (September 2006 - February 2010) which provided many concrete industrial use cases of ATL in various and varied contexts. These use cases provided new research challenges that helped us to improve the language.

ATL is integrated in the Eclipse open source community. It joined the Eclipse-Modeling GMT (Generative Modeling Technologies) project in 2004 and then moved to Eclipse-Modeling M2M (Model-to-Model Transformation) project [7] in 2006. This was the effective recognition of ATL as one of the de-facto modeling standards in Eclipse. We believe this integration in the Eclipse foundation has contributed a lot to the development of the community and of the tool in general.

---

[1]At least among researchers, based on the number of works referencing and/or using ATL; there are plenty of industrial users as well but this is more difficult to quantify and compare against other tools

Finally, as we will explain in the next sections, the ATL ecosystem and the user community became too large to manage only with the resources available in the AtlanMod team. Therefore, in 2007, it was decided to start a collaboration with the Obeo company with the purpose of industrializing ATL. This situation is still going on today and has proved to be very beneficial for both AtlanMod and Obeo.

2.2. Transforming models with ATL

ATL is a model-to-model (M2M) transformation language. M2M transformations specify the production of target models from a number of source models. ATL enables developers to define how source model elements must be matched and navigated in order to create and initialize the target model elements. Roughly speaking, an ATL transformation is defined as a set of declarative rules. Each rule matches a subset of the source model and creates an excerpt of the target model. Pattern matching conditions are expressed in the OCL language [8].

Source and target models can be instance of different metamodels. A simple ATL transformation Mt (see Fig. 1 transforms a model Ma, which conforms to a metamodel MMa, into a model Mb, conforming to a metamodel MMb. Since in MDE everything should be considered, as far as possible, as a model, Mt itself is defined as a model conforming to the ATL model transformation metamodel MMt. All three metamodels conform to a metametamodel MMM such as MOF or Ecore.
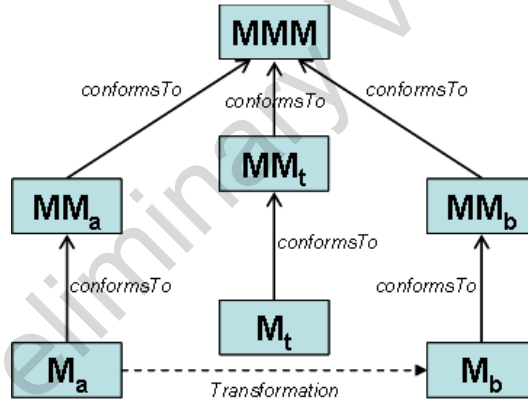

Figure 1: Model-to-Model transformation principle

As an example, Figure 2 shows an excerpt of the classic *ClassToRelational* model-to-model transformation example. The objective of this transformation is to transform a class diagram into a relational database schema, with the corresponding tables, columns, etc. In this example, the source metamodel is *Class* and the target metamodel is *Relational*. The actual transformation defines several rules (as *Class2Table* and *DataType2Type*) defining the mappings between the concepts from the two metamodels, and one helper method (helpers factorize ATL code that can be called from different points of an ATL transformation) navigating the input model to retrieve some data needed during the mappings.

Fig. 2 shows part of the *ATL Perspective*, with the *Project Explorer* presenting the different resources involved in the transformation, an excerpt of the ATL transformation displayed using the *ATL Editor* and the *Outline* of the transformation displaying all its important information.
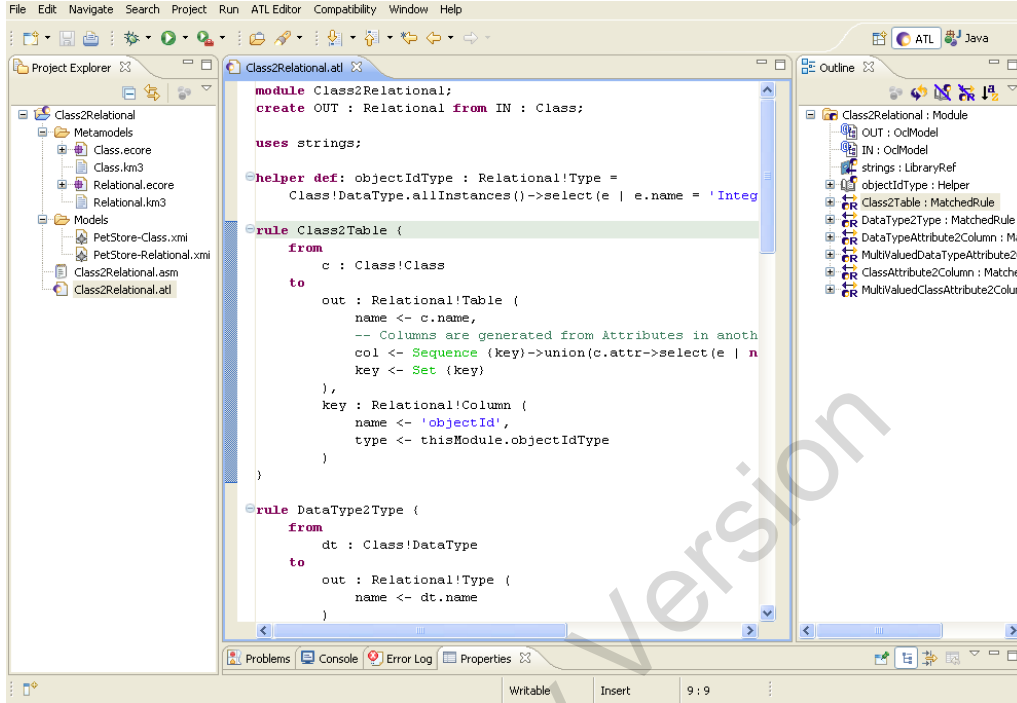
Figure 2: The ATL perspective with a simple transformation example

2.3. Current status: the ATL project

More than a simple model transformation language, ATL has evolved into is a full Eclipes project [9] directly under the umbrella of the Eclipse *Modeling* top-level project

As an Eclipse project, ATL proposes to the community a complete Eclipse IDE (Integrated Development Environment) coming along with the ATL language, core components and many different kinds of resources (see Figure 3). The ATL project is implemented as a set of specific Eclipse plug-ins and features. ATL has a dedicated Eclipse perspective, its own project nature and builders (and corresponding creation wizards) for ATL projects and files. An editor with extended capabilities (syntax highlighting, runtime parsing, compilation and error detection, code completion, etc) and corresponding outline, a debugger, profiler, launcher and ATL Ant tasks have also been implemented.

Apart from the regular plug-in downloads that can be get from the various update sites the project offers a large range of online information: conceptual overview, user guide, developer guide, technical specifications, etc. Many training resources like simple examples, tutorials or complete use cases are also available. Moreover, the project also provides a library of more than one hundred open source model-to-model transformations, all written in ATL and developed by various contributors, concerning many different metamodels and domains. These transformations can of course be taken as examples, but also reused as is or refined/extended for specific transformation scenarios.

Besides this, the ATL community is continuously expanding. There are already several external tools using ATL as their standard model-to-model transformation technology and, in addition

4

Figure 3: The ATL project on Eclipse.org

to the various publications and presentations related to ATL in major events, project committers and contributors regularly give support to ATL users via the dedicated Eclipse forum, Eclipse Wiki or Eclipse Bugzilla. For instance, around 40 different topics (i.e. discussion threads) directly concerning ATL have been active in the Eclipse-M2M forum just in the two first weeks of July 2010.

3. Challenges in the Development of a Commercial-Quality ATL

As ATL was growing into the nowadays complex ecosystem (see previous section) it was clear that ATL had the chance to become the reference model-transformation solution for companies starting to adopt MDE as their development paradigm. As a research group, this was very appealing since it would allow us, among many other benefits, to: 1 - Empirically validate our transformation language and, in general, our research ideas; 2 - to provide good on-the-field experimental data; and 3 - to increase the AtlanMod team visibility, increasing the number of companies/groups that could discover (and become users of) our other works in the MDE field and opening opportunities for future collaborations (e.g. technology transfer contracts or joint participation in international projects).

However, we quickly realize that it would be very difficult to maintain all the components of the ATL project and give proper support to users only with the resources available in the team.

5

Furthermore, focusing on providing a good technical solution would not be enough to keep and expand our user base, for potential users (specially companies) technical quality is only one of the many aspects they take into account when selecting a tool. ATL users started to ask for:

- better user support (e.g. responsiveness to questions, bugs and feature requests)

- up-to-date and complete tool documentation

- good user experience (ergonomy)

- long-term perspectives for the tool

- frequent upgrades and interoperability with the latest versions of other common used tools

- extensibility, adaptability, and all the other typical -ities

- backward compatibility (which many times is difficult to combine with innovative research)

In fact, we agree that all these aspects are important (as first users of the tool we would also benefit from them, e.g. when hiring a new team member a proper documentation set would ease his/her integration in the team activities) and they have been always in the wish-list of ATL though hardly ever made it to the actual to-do list. The main reason is that the low availability of funding for pure tool development forces us (and, in general, all research groups) to focus our resources on the most innovative and research aspects of ATL neglecting all these other issues. This is a direct consequence of the fact that most (all?) government agencies evaluate research groups (and their members individually) exclusively based on the number of papers published by the group, ignoring the impact in the community of the tools developed by them. This makes extremely difficult to convince young researchers pursuing a permanent position to dedicate time to any task that does not immediately and directly lead to new research results.

To overcome this situation, we have put in practice two kinds of strategies during the development of ATL. First, the design of the tool itself simplifies some of these problematic aspects and, more importantly, tries to facilitate the collaboration and technological transfer among all external actors contributing to the ATL development. The second strategy deals with the problem of convincing external people to contribute to ATL. In fact, given the open source license[2] of ATL, one could have the (naive) initial assumption that it would be easy to find external contributors ready to jump in and start working on all aspects a research group cannot cover. However, this assumption fails because few projects are succesful involving external contributors [10] and most times, these external users also prefer to work on the most challenging tool features and not, for instance, write documentation or improve the graphical user interface. For instance, many important ATL contributors are members of other research groups that extend or improve ATL as part of their research interests.[3] Therefore, we realized that, apart from having a good tool design, we needed to define a *business model* for ATL to guarantee a continuous influx of resources to satisfy the requests of our users while allowing us to keep focusing on the core research aspects of the tool. Section 4 and Section 5 elaborate on these two kind of strategies.

---

[2]More specifically, ATL follows the Eclipse Public License

[3]See, for instance, the program of the two ATL workshops organized so far: http://www.emn.fr/z-info/atlanmod/index.php/MtATL2009:Program and http://www.emn.fr/z-info/atlanmod/index.php/MtATL2010:Program

4. The ATL Solution: Technical Aspects

ATL tool design has been influenced by the principles of modularity, interoperability and use of standard (de facto) technologies as a way of facilitating contributions from independent actors taking part in the ATL development. Apart from this, becoming intensive users of your own tool is also a *best practice* to test if the previous design priciples have been properly followed.

4.1. Modularity

The ATL framework has a modular structure based on the extension mechanism of the Eclipse platform [11], that in turn is built upon the OSGi open modular system. The *core modules* of ATL are shown in Figure 4. These modules are in charge of the operations that lead to the transformation execution: code parsing and static checking, compilation and bytecode interpretation.
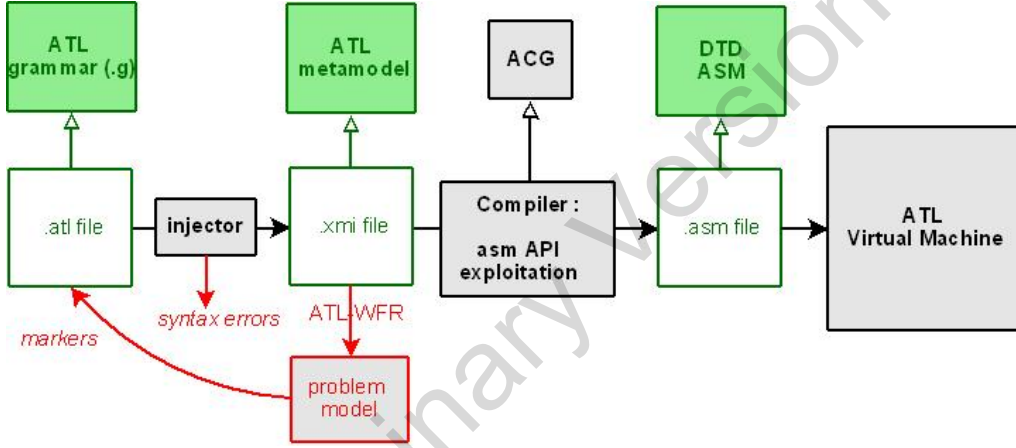

Figure 4: Components involved in the execution of an ATL transformation

The first component is the ATL parser (i.e. the *injector* in the figure) which takes an ATL source file as input and produces a model that conforms to the ATL metamodel as output plus a problem model storing all errors detected in the code. This error information is then used by the ATL editor to display the corresponding error markers in the ATL source file. The second component is the ATL compiler in charge of converting the ATL model into bytecode (ASM format) interpretable by the ATL Virtual Machine (VM). It has been implemented using ACG, a compiler-oriented DSL [12]. The last core component, the ATL VM, takes the compiled transformation and actually executes it on the specified input model(s) to generate as a result the output one(s). Additional ATL modules are implemented on top of these core ones to provide the main functionalities of the ATL development environment (e.g., editor, debugger, profiler, launcher). Communicaton between the modules is done thanks to the standard Eclipse extension interfaces. These same interfaces can be used by external developers wanting to make use of ATL in their own Eclipse plug-ins (e.g., AMW [13]).

This modularization of the ATL core allows for a clear separation of concerns among the basic components in charge of the transformation processing. This decoupling has proven to be a valuable asset for collaborative work over the ATL core. For instance, it is common for independent research groups to locate a research problem in only one of these components. Thanks

7

to the ATL modularization the group only needs to get familiar with a limited set of technologies (e.g. ACG for the compiler) and can provide an updated interchangeable version of the module addressing that specific research issue. Examples of this are the optimization of the collection handling approach of ATL by updating only the virtual machine [14] or the development of an incremental execution mode for ATL by changing only the ATL compiler [15].

The modularization of ATL has shown its importance in several projects during the last years. As a further step in this direction, we plan to introduce a new fine-grained extension system for the ATL core components by defining a new set of extension-points in the core, i.e. of functionalities that can be delegated to externally provided plug-ins.

4.2. Standard technologies

An important driver in the success of ATL has been the selection of the right core technologies for grounding the tool. The search for an open framework, supported by a solid open-source community, and that could become the de-facto standard platform for open software engineering tools, lead us to choose the Eclipse platform. The growing popularity of Eclipse and the increasing number of tools migrating to this platform proves our decision right and increases the opportunities of collaborations between ATL and the other Eclipse projects.

A first benefit of integrating ATL in Eclipse is that Eclipse relieves the ATL project from implementing most of the components of the IDE. The ATL perspective reuses many of the standard IDE components provided by Eclipse and adapts them to the specific needs of ATL. Eclipse also provides an extensibility mechanism that facilitates the definition of ATL extensions. Eclipse plug-ins can define *extension-points*. An extension point indicates that the plug-in is able to delegate that specific functionality to a different plug-in. *Extension-points* can be answered by *extensions*, that implement an interface and descriptor matching the extension-point. Availability of a standard extension mechanism is a key factor in enabling the creation of re-usable components from a complex and heterogeneous community.

Finally, the success of a model transformation technology like ATL has necessarily to rely on the success of the underlying modeling framework it uses (i.e. if models to be used as input/output of ATL transformations had to be conformant to a modeling framework nobody uses then, obviously, few people would use ATL). While the modular architecture of ATL favors interoperability and adaptation to different frameworks (see next subsection), we decided to privilege[4] the Eclipse Modeling Framework (EMF) since it soon appeared that among several possible alternatives, EMF stood out as the de facto standard solution for model handling. The fact that today most model-based tools in Eclipse work with EMF facilitates the selection of ATL as a model transformation language for their model manipulation activities.

4.3. Interoperability

While the early choice of Eclipse and EMF has certainly influenced the success of ATL, the design of its architecture has followed the principle of maximum independence from specific technologies so that we can change these technological choices in the future. For instance, as a transformation language, ATL should be independent on the particular way a model is represented and handled in the modeling environment.

---

[4]We say that EMF is a privileged modeling framework in the ATL toolkit because there is an optimized version of the virtual machine available for EMF that guarantees the maximum efficiency when working with EMF models

On the one hand, this criterion guarantees that the longevity of ATL will not be influenced by a particular technology (e.g., EMF) going out of fashion. This is particularly important in an industrial context, where the preservation of investments is a central requirement. On the other hand this independence enables a straightforward path for exporting ATL to other modeling environments (e.g., Microsoft DSL Tools [16]), with the opportunity of remarkably expand the user base.

As technical solution for remaining independent from the modeling layer we introduced the notion of virtual machine to execute the ATL compiled bytecode. The ATL Virtual Machine is an abstract computing machine, associated with its own instruction set and specialized in model manipulation. Within the virtual machine, an intermediate component called Model Handler Abstraction Layer deals with the specifics of the modeling layer. This component translates the instructions of the VM for model manipulation into instructions for a specific model handler. By implementing an ad-hoc model handler it is possible to use the same VM on top of various model management systems (e.g., Eclipse EMF, Netbeans MDR, Microsoft DSL Tools) [17].

Moreover, porting the VM itself to other runtime environments (e.g. from Java to .NET) opens the door to immediately execute ATL transformations in them since any compiled ATL transformation can be indifferently executed on all implementations of the ATL VM that conform to the provided specification, independently of the underlying technology [17].

4.4. Eating your own dog food

We believe it is important that research groups are the first users of the tool and that the tool is bootstrapped when possible. For instance, the ATL compiler towards ASM bytecode is written in the ACG DSL. The ACG compiler is bootstrapped, being itself written in ACG and being parsed, compiled towards ASM and executed using the same technology (the TCS parser, ACG compiler, ASM virtual machine) used to parse user defined ATL transformations. In the same way the execution of ATL transformations is partially performed using other ATL transformations (e.g. to generate the problem model).

Apart from being a non-trivial test for the tool, the main benefit of this is that any improvement on the tool components improve not only the user-defined transformations but also the ATL engine itself.

5. The ATL Solution: Business Model

The software engineering best-practices that have been followed since the first years enabled, in a first phase, the construction of an initial open-source community around the tool under the guidance of the research group AtlanMod. However, as argued in Section 3, once the user base started to grow and the ATL project became more complex this was not enough to guarantee a durable and expanding environment for ATL. A novel business model was required, to coordinate needs and efforts of the actors operating around ATL.

In 2007, the industrialization process for ATL started. AtlanMod partnered with Obeo [4] to ensure the existence of an open source but commercial-quality version of ATL. As part of this agreement, Obeo started committing resources on the non-core aspects of ATL (the amount of resources is more or less equivalent to the ones AtlanMod devotes to ATL research topics).

From the point of view of the research group, the main goal of the agreement is allowing AtlanMod to focus its resources on new research activities, and letting Obeo take over traditional software development and maintenance tasks, including performance and usability improvements, bug fixing and user support. Thanks to this agreement, ATL technology reaches an

industrial quality, becoming a very attractive tool for company users and a stable base for new research directions.

Obeo, on the other hand, gets to strenghten its presence and visibility within the Eclipse community and among industries using its open-source tecnologies. This strategic positioning is used to create a network of key partner companies that help to expand the Obeo market. As part of the agreement, Obeo also becomes the lead ATL training company (AtlanMod redirects to Obeo all ATL-specific course requests coming from companies) and can deliver adapted versions of ATL to specific companies with special needs. Moreover, ATL complements well the other MDE technologies in the the Obeo offering (that already include, for instance, the Acceleo tool for model-to-text transformations) and it is already been used as a key component in Obeo's new tool releases. This in-house development of ATL and the privileged relationship with the creators and lead research contributors of ATL (ourselves) give Obeo a competitive advantage over other companies that also propose products based on ATL.

More specifically, and according to the agreement with AtlanMod, Obeo takes responsibility over a precise set of activities around the ATL technology:

Quality assurance. The main responsibility of Obeo is in maintaining the ATL software at a quality level that is acceptable for wide industrial use. This task is twofold, since it includes a *reactive* part, mainly related to managing bug reporting and correction, and a *proactive* part, for the optimization of the ATL codebase, with a special attention to scalability.

Interoperability. The ATL tool is intended as a part of a complex modeling system, composed both by modules built within Obeo itself (e.g., Acceleo) and tools provided by third parties (e.g., modelers, modeling frameworks). Obeo takes care of the issues connected with the interoperability among ATL and the related modules, and implements new solutions to improve this interoperability.

Continuity. In an industrial context it is important to guarantee that the evolution of development tools will not unexpectedly break existing projects. The integration in ATL of new technical solutions, coming from Obeo or academia, has to be performed in an incremental way, carefully preserving backward compatibility with previous versions.

User experience. One of the priorities of the industrialization of ATL is the improvement of the general user experience in the development environment. Obeo makes use of ergonomy principles for the improvement of the IDE, and adds typical facilities such as wizards and hints. The company also takes care of the internationalization of the user interface.

Release management. Obeo takes charge of the management of ATL releases, by defining milestones, building, testing, packaging and distributing new versions.

User support. Finally Obeo offers support to the user base, in both the forms of free and subscription-based support. Casual users are assisted for free by managing the newsgroup and the mailing list on the Eclipse website. Obeo also commits in extending and updating the documentation of the ATL components. On the other hand Obeo has an immediate income from offering industrial users a personalized support and training programs.

At the same time Obeo took the lead of the official (stable) version of ATL, a new branch of ATL was formed, *ATL Research*, to host the experimental ATL versions, under the supervision of AtlanMod. Figure 5 illustrates the mechanism for technological exchange between the two

branches of ATL. The official versions of ATL, maintained by Obeo, are publicly distributed and constitute the baselines upon which research prototypes are built. These experimental *ATL Research* prototypes exhibit novel solutions for particular research projects. In the cases in which these solution are considered interesting for the general public, they are integrated in the subsequent official version of ATL. Depending on the stability of the prototype and on its compliance with a set of quality criteria (e.g., scalability, backward compatibility) the module in ATL Research can be directly imported in ATL, or, more commonly, its functionality is re-engineered by Obeo before being integrated in the stable branch. We believe this two-branch strategy is a good solution to avoid any restrictions on the research activities around ATL while not damaging the stability and backward compatibily properties desired by industrail users.
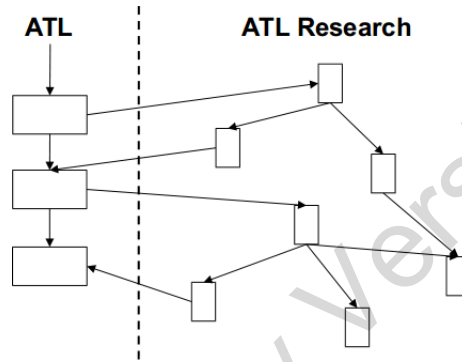

Figure 5: ATL and ATL Research

We remark that the initial choice of licencing every part of ATL under an open-source licence (this includes both the ATL research branch and the official branch maintained by Obeo) has remarkably simplified the creation of this new business model. The open source licence automatically guarantees the possibility to transfer every ATL update between the company and the research lab. Moreover this open source licence also assures to any third party, especially other research groups, that this code exchange will not benefit only AtlanMod, but the whole research community . This guarantee has the effect to stimulate the contribution of new code from ATL users and researchers, with a return gain for AtlanMod and Obeo themselves.

6. A Sustainable Open Source Business Model for Industrializing Research Tools

We are now replicating this industrialization process with two other MDE technologies from the group: the software modernization tool MoDisco [18] (together with Mia-Software [19]) and, more recently, our Megamodeling model management approach [20] (with ProDevelop [21]), with the same levels of success and satisfaction for the results of the collaboration achieved so far. Therefore we plan to generalize this strategy in most of the new projects of the team.

In this sense, this section sketches our *sustainable open source business model for industrializing research tools* triangle that summarizes the general industrialization schema we envision. There are three main actors in this triangle: the research lab, the technology provider and a big company/user community (see Figure 6). The triangle allows connecting research groups and industrial users and match their interests.
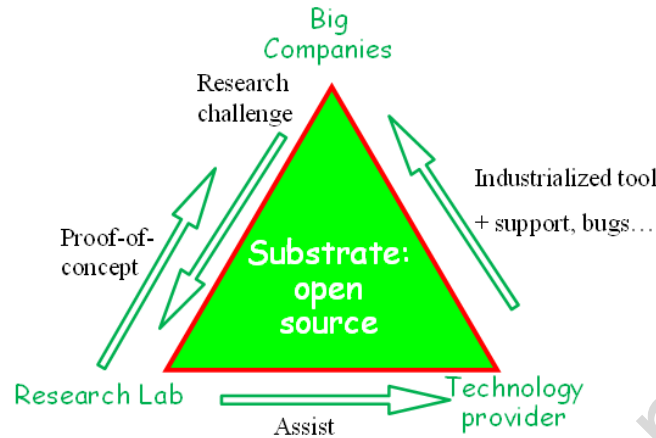
The triangle works as follows:

Figure 6: Open source triangle for the successful industrialization of research tools

1. Big companies (or big user communities) describe a challenging problem they face in their domain and that they would like to see solved (e.g. the industrial use cases that helped to drive the development of ATL in its early beginnings)
2. The lab evaluates the problem and decides whether it is a relevant research problem
3. If it is, the lab conducts the research (and publishes papers about it) and implements a proof-of-concept to present the results to the big company
4. The big company evaluates these results and decides whether to go forward and request an industrialized version of the tool to use in its day-to-day practice
5. In that case, the big company, with the help of the research lab, selects a technology provider.
6. The research lab assists the technology provider during the industrialization of the tool
7. The technology provider releases the tool

The fact that this is an application-driven research (i.e. the starting point is a real problem that a big company wants to be solved) ensures the return of investment for the technology provider. Adopting open source as the common denominator in all these activities is not absolutely mandatory but facilitates a lot the communication between the different actors and maximizes the benefits of the relationship (e.g. for the research group it is easier to publish papers about it and the technology provider could commercialize services and adaptations on top of the tool for other big companies sharing the same problem). This triangle can also be used to characterize many other interesting scenarios between the actors. For instance, when the research group is not interested in the problem expressed by the big company, the company can still search for a technology provider to solve that problem. Detailing all possible relationships cannot be done here due to space limitations.

This idea of intertwining research groups and industry partners has been already praised for its benefits [22]. Similar to the *industry-as-laboratory* approach presented there, we believe that continuous communication and technology transfer, between research lab and industry, is very beneficial for both partners. However, we believe that our triangular structure can help overcoming the limitations identified in [22] (such as the short-term/long-term and evolutionary/revolutionary different interests between the lab and the industry), since it gives to the re-

12

search lab the flexibility to combine the benefits of working with real end-users and technology providers at the same time in a coherent global schema.

We strongly believe that this schema can be successfully adopted by other research groups as a strategy to develop industrial quality level tools with all the benefits that this brings to the group.

7. Conclusion

We have presented our strategy for developing high-quality tools that can become widely used by the software engineering community: industrialization of the research prototypes thanks to the partnership with a technology provider. We believe this is the best solution to ensure that somebody external to the team (in our case, this technology provider) is in charge of all non-core research aspects of the tool development (documentation, usability, user support, etc). These aspects are key to convince external users (specially companies) to adopt our tools but cannot be taken in charge by the research teams themselves due to their limited resources. We have validated this strategy during the development of our ATL tool and we are now replicating it with other tools of the group.

We believe our experience can be useful to other research groups willing to go from proof-of-concept tools to real industrial-level tools, with all the benefits this may bring to the team (in terms of visibility, feedback, project opportunities and so on). We also hope our example helps to convince as well technology providers and development companies about the advantages of participating in such joint initiatives; there is a viable business model behind it, as companies like Obeo or MIA-Software have demonstrated.

As further work, we would like to formalize a protocol for the establishment and execution of this kind of partnerships so that roles, resources and responsibilities of each participant are clear from the beginning. This will facilitate explaining this open source industrialization concept and developing new partnerships in the future. This is being investigated as part of our contribution to the OPEES European project [23] aimed to ensure long-term availability of innovative engineering technologies in the domain of dependable / critical software-intensive embedded systems.

Additionally, we plan to create a web-based collaboration platform for ATL extensions (we tag this project *ATL labs* due to its similarity with existing initiatives as the recent Eclipse-Labs.org proposal) to facilitate the autonomous experimentation and contribution by other people beside our main partner. This project will provide a sandbox where researchers can experiment with ATL and extend it for specific purposes. Such specialized extensions cannot be done on the official ATL branch since there the contribution policy is quite rigid and they don't fit in the tree structure of the ATL Research branch. ATL Labs is meant to host a number of orthogonal ATL extensions for specific needs that potential users can select and compose to get a specific ATL "flavor" adapted to their needs.

Finally, we plan to start using social networking tools to build a community of ATL users that can easily share their experience, help each other and participate in discussions about the future of ATL; the limited participative mechanisms in Eclipse (mailing lists, bugzilla, forums, ...) are not enough for this.

References

[1] F. Jouault, F. Allilaire, J. Bézivin, I. Kurtev, Atl: A model transformation tool, Sci. Comput. Program. 72 (1-2) (2008) 31–39.

[2] J. de Lara, H. Vangheluwe, Defining visual notations and their manipulation through meta-modelling and graph transformation, J. Vis. Lang. Comput. 15 (3-4) (2004) 309–330.

[3] OMG, MOF 2.0 Query/View/Transformation specification (2007).

[4] The Obeo company, `http://www.obeo.fr`.

[5] The AtlanMod team (Ecole des Mines de Nantes & INRIA), `http://www.emn.fr/z-info/atlanmod`.

[6] F. Jouault, Contribution to the study of model transformation languages, Ph.D. thesis, University of Nantes (September 2006).

[7] The Eclipse Model-to-Model (M2M) project, `http://www.eclipse.org/m2m`.

[8] OMG, UML 2.0 Object Constraint Language specification (2006).

[9] The Eclipse ATL project, `http://www.eclipse.org/atl`.

[10] S. Krishnamurthy, Cave or community? an empirical examination of 100 mature open source projects, First Monday 7 (6).

[11] Eclipse Helios Simultaneous Release, `http://www.eclipse.org/helios`.

[12] ATL VM Code Generator (ACG), `http://wiki.eclipse.org/ACG`.

[13] M. Del Fabro, J. Bezivin, F. Jouault, E. Breton, G. Gueltas, AMW: a generic model weaver, in: Proc. of the 1ères Journées sur l'Ingénierie Dirigée par les Modèles, 2005.
URL `http://idm.imag.fr/idm05/documents/11/P11.pdf`

[14] J. S. Cuadrado, A proposal to improve performance of ATL collections, in: MtATL2010, 2010.

[15] F. Jouault, M. Tisi, Towards Incremental Execution of ATL Transformations, in: International Conference on Model Transformation, ICMT2010, 2010.

[16] Microsoft Domain-Specific Language (DSL) Tools, `http://msdn.microsoft.com/fr-fr/library/bb126235.aspx`.

[17] H. Bruneliere, J. Cabot, C. Clasen, F. Jouault, J. Bézivin, Towards model driven tool interoperability: Bridging eclipse and microsoft modeling tools, in: T. Kühne, B. Selic, M.-P. Gervais, F. Terrier (Eds.), ECMFA, Vol. 6138 of Lecture Notes in Computer Science, Springer, 2010, pp. 32–47.

[18] MoDisco, `http://www.eclipse.org/MoDisco/`.

[19] Mia-Software, `http://www.mia-software.com/`.

[20] M. Fritzsche, H. Bruneliere, B. Vanhooff, Y. Berbers, F. Jouault, W. Gilani, Applying megamodelling to model driven performance engineering, in: ECBS, IEEE Computer Society, 2009, pp. 244–253.

[21] ProDevelop, `http://www.prodevelop.es/`.

[22] C. Potts, Software-engineering research revisited, IEEE software (1993) 19–28.
URL `http://www.computer.org/portal/web/csdl/doi/10.1109/52.232392`

[23] Open Platform for the Engineering of Embedded Systems, `http://opees.org/`.