Using ATL transformation services in the MDEForge collaborative modeling platform

Juri Di Rocco[1], Davide Di Ruscio[1], Alfonso Pierantonio[1 3], Jesús Sánchez Cuadrado[2], Juan de Lara[2], Esther Guerra[2]

[1] University of L'Aquila (Italy) - name.surname@univaq.it
[2] Universidad Autónoma de Madrid (Spain) - name.surname@uam.es
[3] Mälardalen University, Västerås (Sweden) - name.surname@mdh.se

Abstract. In the last years, the increasing complexity of Model-Driven Engineering (MDE) tools and techniques has led to higher demands in terms of computation, interoperability, and configuration management. Harnessing the software-as-a-service (SaaS) paradigm and shifting applications from local, mono-core implementations to cloud-based architectures is key to enhance scalability and flexibility. To this end, we propose *MDEForge*: an extensible, collaborative modeling platform that provides remote model management facilities and prevents the user from focussing on time-consuming, and less creative procedures. This demo paper illustrates the extensibility of MDEForge by integrating ATL services for the remote execution, automated testing, and static analysis of ATL transformations. The usefulness of their employment under the SaaS paradigm is demonstrated with a case-study showing a wide range of new application possibilities.

1 Introduction

Modeling and model management tools are commonly distributed as software packages that need to be downloaded and installed on client machines, and often on top of complex development IDEs, e.g., Eclipse[4]. Given the non-trivial implicit and explicit interdependencies of such tools, this can often be a burden, particularly for non-technical stakeholders (e.g., domain experts) with average IT skills. Moreover, the increasing complexity of the systems to be built and their high demands in terms of computation, memory and storage, requires more scalable and flexible MDE techniques.

A first attempt to deal with such challenges is the Modeling as a Service (MaaS) initiative [6], which proposed the idea of deploying and executing MDE services over the Internet. This is aligned with the software as-a-service (SaaS) paradigm, since consumers do not manage the underlying cloud infrastructure and deal mostly with end-user systems. Even though there are different projects (e.g., the EU MONDO project[5]) and approaches [1,9] related to the adoption of cloud infrastructures for MDE, the area is still at its infancy. In [4], MDEForge was proposed as an extensible platform enabling the adoption of model management tools as SaaS: advanced functionalities like unmanaged clustering of large metamodel repositories [3], and automated chaining of model transformations [5], are already part of the core services.

[4] http://www.eclipse.org
[5] http://www.mondo-project.org/

In this demo paper, we show how MDEForge has been extended to enable the remote execution and analysis of ATL transformations, their automated testing and static analysis. Section 2 presents an overview of MDEForge and its core services. The next section introduces the developed extensions to support ATL transformations. Section 4 presents how to use such services in practice by exploiting both the Web access and REST APIs. Finally, Section 5 draws conclusions and outlines future developments. Additional resources about this demo paper are available on line[6].

2 MDEForge

MDEForge is an extensible online modeling platform specifically conceived to foster a community-based modeling repository, which underpins the development, analysis and reuse of modeling artifacts. The MDEForge platform consists of a number of services that can be used by means of both Web-based and programmatic interfaces (APIs) that enable their adoption as SaaS (see Fig. 1). Core services are provided to manage users and modeling artifacts, e.g., models, and metamodels. Resembling functionalities of desktop IDEs, like Eclipse, registered users have the possibility to create modeling artifacts and organize them in projects that are, in turn, contained in workspaces. Projects and artifacts can be shared with users of the same system installation. Next, we describe the most relevant MDEForge services, shown in Fig. 2.

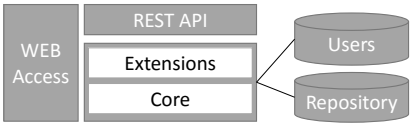

Fig. 1. Overview of the MDEForge architecture

CRUDArtifactService. It permits to create, update, query, and delete artifacts in the repository. An abstract implementation of the service is provided in order to have a default and common behavior, which can be parametrized by exploiting Java generics to handle specific kinds of artifacts (e.g., models, metamodels, and transformations).

CRUDRelationService. This service permits representing in a *megamodel* all the artifacts stored in the repository together with the relations among them. For instance, for each stored model in the repository, a conformance relation element exists in the megamodel to refer the corresponding metamodel. Similarly to *CRUDArtifactService*, a generic implementation of *CRUDRelationService* is given, which is then specialized to manage specific relations such as conformance (between models and metamodels) and domain conformance (between model transformations and corresponding metamodels).

UserService. This service provides authentication and authorization functionalities and underpins the management of workspaces, projects, and shared artifacts.

WorkspaceService. It provides CRUD operations to manage user workspaces, which are used to organize projects and artifacts.

ProjectService. It provides CRUD operations to manage projects with different kinds of artifacts. Differently to workspaces, projects can be shared between several users.

GridFileMediaService. In MDEForge we have defined a common layer to handle physical files. *GridFileMediaService* provides a set of functions that take as input artifacts and retrieve physical paths, input/output streams, etc.

[6] http://www.di.univaq.it/diruscio/ICMT2016-MDEForge-tool-demo-accompanying.pdf

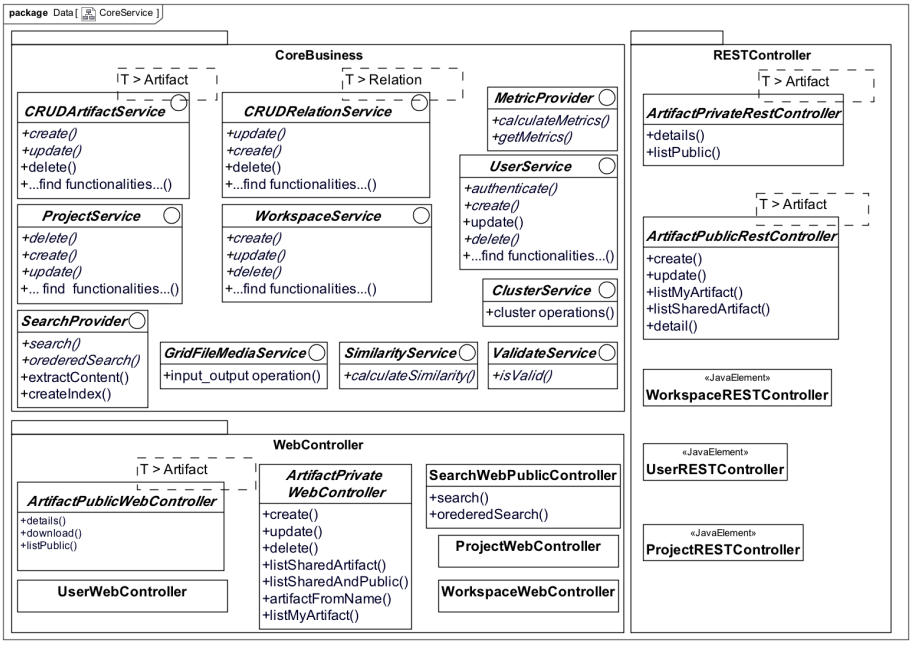

Fig. 2. MDEForge core services

ClusterService. In order to mitigate the difficulties related to manual categorization of artifacts, MDEForge provides a clustering technique to group together mutually similar artifacts depending on a proximity measure (implemented by *SimilarityService*), whose definition can be given according to specific search and browsing requirements [3].

MetricProvider. In order to assess the quality of the stored modelig artifacts, for each kind of artifact it is possible to define (by implementing the method *calculateMetrics*) the corresponding metrics to be calculated [7,8].

SearchProvider. MDEForge provides common methods to search artifacts. By implementing the *createIndex* method it is possible to customize the search.

3 MDEForge extensions for ATL

MDEForge has been extended with support for ATL transformations, including reuse, sharing, execution, analysis and testing. In Sec. 3.1 we identify the functional requirements that have been considered to implement the extensions, presented in Sec. 3.2.

3.1 Functional requirements

The functionalities provided by the MDEForge extensions presented in the next sections have been developed with the aim of fulfilling the requirements described below.

RQ1 - Create, read, update and delete ATL transformations: like any kind of artifacts handled by MDEForge, the extension has to permit CRUD operations on ATL transformations. Moreover, the system should manage transformations in *.xmi* and *.atl* formats. When transformations are uploaded the system should take care of compiling them and informing the user in case of errors.

RQ2 - Share ATL transformations: in order to promote reuse of existing transformations, the system has to provide sharing facilities similar to those of public storage services like Dropbox and Google Drive. Thus, when users upload transformations, they can decide if they have to be private, public or shared with other users.

RQ3 - Manage megamodeling relations: when new ATL transformations are uploaded, it is necessary to update the megamodel representing all the artifacts stored in the repository. Thus, specific *domainConformsTo* and *coDomainConformsTo* relations have to be introduced in order to relate the transformations being uploaded with the corresponding source and target metamodels, respectively.

RQ4 - Search ATL transformations: the default MDEForge search methods have to be extended in order to enable the specification of advanced queries, e.g., search all ATL transformations that produce models conforming to a specific metamodel.

RQ5 - Execution of ATL transformations: in line with the MaaS initiative, MDEForge has to enable the remote execution of ATL transformations. To this end, once an already stored transformation has been selected, it is necessary to upload the source models and validate them with respect to the source metamodels (*RQ5a*), execute the transformation (*RQ5b*), store and return back the result (*RQ5c*).

RQ6 - Analysis of ATL transformations: the system should enable the remote analysis of ATL transformations. According to [11], the evaluation of specific metrics can give relevant insights and support quality assessment transformations tasks. In particular, static analysis can efficiently reveal problems with no need for transformation execution [10]. Additionally, testing mechanisms able to generate large sets of test input models can play a key role for exercising transformations and detecting faults [2]. Thus, the MDEForge extensions required to analyse ATL transformations have to enable the calculation of metrics (*RQ6a*), and support their static analysis (*RQ6b*) and testing (*RQ6c*).

RQ7 - Remote access to the ATL transformation services: all the previously presented functionalities have to be implemented as services in order to enable their adoption by means of both specific APIs and the MDEForge Web interface.

3.2 ATL services in MDEForge

In order to fulfil the abovementioned requirements, MDEForge has been extended as depicted in Fig. 3. In particular, the added interface `ATLTransformationService` extends the core `CRUDArtifactService` in order to define ATL specific services i.e., executing and analysing transformations. The implementation of the added interface is given in the new `ATLTransformationServiceImpl` class, which extends the core `CRUDArtifactServiceImpl` class and implements also the core interfaces `SearchProvider` and `MetricProvider`. The static analysis and the testing services are also defined in `ATLTransformationServiceImpl` by implementing the new interfaces `AnATLyzerService` and `ATLTransformationTesterService`.

The analysis service uses *anATLyzer* [10], a static analyser for ATL able to detect over 40 types of errors statically (e.g., rule conflicts, unresolved bindings, uninitialized features). The testing service uses random testing, producing input models via constraint solving and checking for runtime errors and non-conforming target models.

In order to enable the use of ATL services in the MDEForge Web client, the core classes `ArtifactPublicWebController` and `ArtifactPrivateWebController`

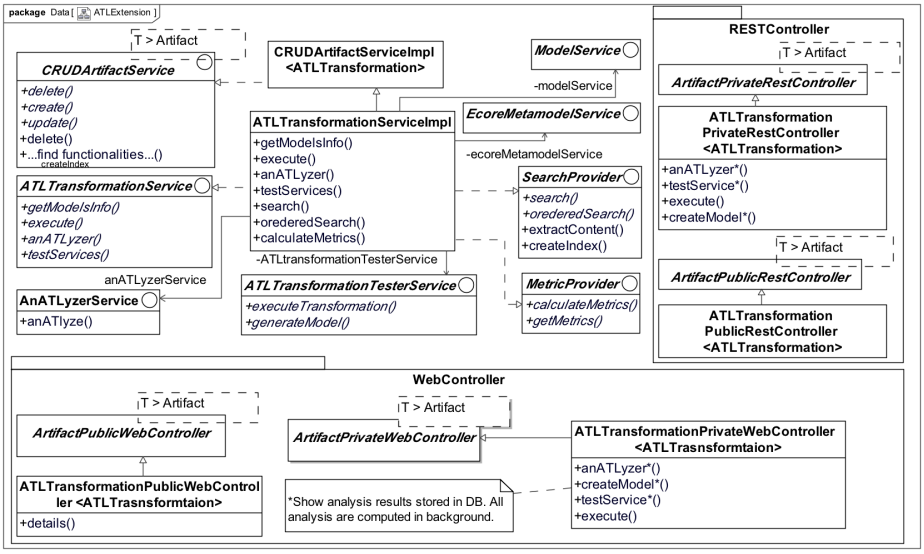

Fig. 3. MDEForge extensions for ATL

have been extended by new controllers. Similarly, the core REST controllers have been extended to enable the programmatic use of ATL services.

4 Use of ATL services

Next, we show how the added ATL services can be used. Fig. 4 shows the MDEForge Web page showing details about the *Families2Persons*[7] transformation stored in the repository. On the top of the page, general data of the transformation are shown i.e., the user who has imported it, when it was added to the repository, and the date of the last change. Moreover, the users the transformation has been shared with are also shown.

The outcome of the analysis services is shown in the sections *anATLyzer Transformations errors* and *Test service report*. In the specific example, the former shows an error that might occur at run-time because of the access to the `lastName` feature that can be undefined. This error has been confirmed by the test service, which has generated three test models that have raised the error at run-time. Test models can be downloaded and explored in order to figure out how to improve the transformations that raised the errors. Model transformations can be remotely executed from the *Execute the Transformation* section. From this section, users can select input models already available in the repository or can upload new ones. Once the input models are selected, the transformation can be executed, and the link to download the generated target model is given back to the user. On the bottom of the page, the system shows metrics calculated over the considered transformation, which can be used for quality assessment.

The ATL services can also be used in a programmatic way by means of a Java client, which makes use of specifically designed REST APIs. The execution of a given ATL transformation can be done as shown in line 23 by exploiting the `ATLTransformationService`, which has been initialized in lines 2-3. The input model to the transformation

[7] `http://www.eclipse.org/atl/atlTransformations/#Families2Persons`

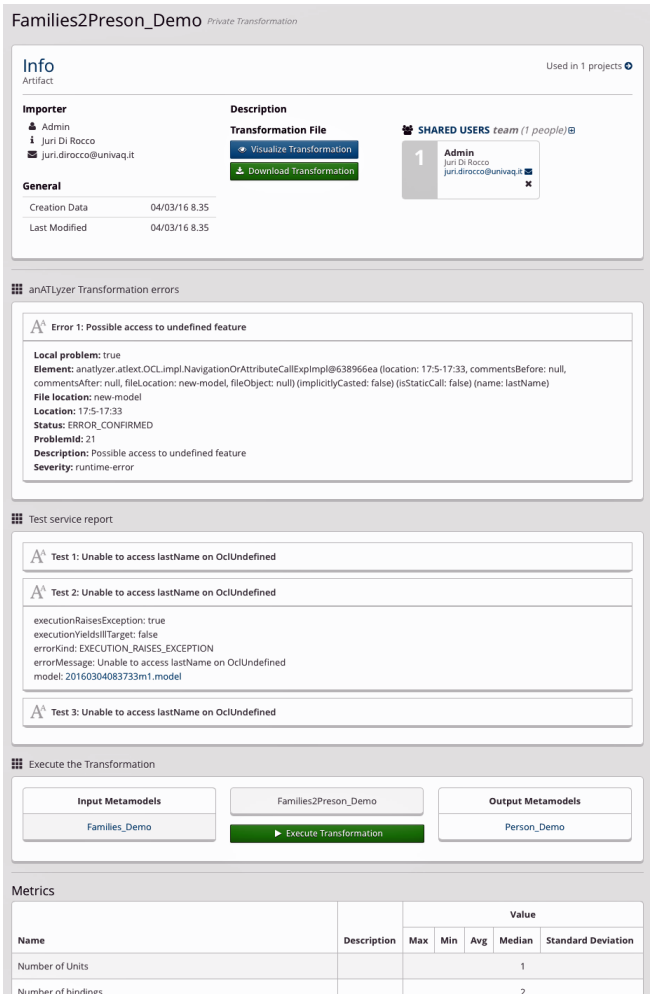

Fig. 4. ATL transformation details page

is retrieved in lines 9-11. To this end, the `EcoreMetamodelService` initialized in lines 4-5 is exploited. The analysis services can be applied on the loaded transformation as done in lines 18-20. It is important to remark that the execution of the analysis services can be time consuming, thus if the results are already available (because of previous executions) then they are given back immediately, otherwise the service executions are scheduled and the user is informed as soon as the results are available.

Listing 1.1. Use of ATL services in a programmatic way

```
1 //Init client services
2 ATLTransformationService atlClientService =
3    new ATLTransformationService("<server_url>", "<User", "<pw>");
4 EcoreMetamodelService ecoreClientService =
5    new EcoreMetamodelService("<server_url>", "<User", "<pw>");
6 EcoreMetamodel families= ecoreClientService.getEcoreMetamodelByName("Families");
7
```

```
//Create model to be transformed
Model simpleFamilyModel = new Model();
simpleFamilyModel.setName("simpleFamilies_Demo");
simpleFamilyModel.setFile(ModelService.setGridFileMedia("sample-Families.xmi"));
[...]
//Load Transformation
ArtifactList models = new ArtifactList();
ATLTransformation t=atlClientService.getATLTransformationByName("Families_Demo");

//Analyze transformation
List<ATLTransformationError> anATLyzerError = atlClientService.anATLyze(atl);
List<ATLTransformationTestServiceError> testServiceError =
   atlClientService.testerService(t);

//Execute transformation
models.add(simpleFamilyModel);
List<Model> result = atlClientService.executeATLTransformation(t, models);
```

5 Conclusions and future work

In this paper, we have shown how MDEForge has been extended to add support for executing and analysing ATL transformations according to the SaaS paradigm. In the future, we intend to implement further extensions for instance to support advanced queries on the repository and to support collaborative modeling activities. As future work we intend also to investigate issues that are typical in Cloud computing, e.g., scalability of the platform, and workload management.

Acknowledgements. Work supported by the Spanish MINECO (TIN2014-52129-R), the Madrid Region (S2013/ICE-3006), and the EU commission (#611125)

References

1. Acretoaie, V., Störrle, H.: Hypersonic-model analysis as a service. In: PSRC@ MoDELs. pp. 1–5 (2014)
2. Aranega, V., Mottu, J.M., Etien, A., Degueule, T., Baudry, B., Dekeyser, J.L.: Towards an automation of the mutation analysis dedicated to model transformation. Software Testing, Verification and Reliability 25(5-7), 653–683 (2015)
3. Basciani, F., Di Rocco, J., Di Ruscio, D., Iovino, L., Pierantonio, A.: Automated clustering of metamodel repositories. In: CAiSE (2016), to appear
4. Basciani, F., Rocco, J.D., Ruscio, D.D., Salle, A.D., Iovino, L., Pierantonio, A.: Mdeforge: an extensible web-based modeling platform. In: CloudMDE@MoDELS. pp. 66–75 (2014)
5. Basciani, F., Ruscio, D.D., Iovino, L., Pierantonio, A.: Automated chaining of model transformations with incompatible metamodels. In: MODELS. pp. 602–618 (2014)
6. Brunelière, H., Cabot, J., Jouault, F.: Combining Model-Driven Engineering and Cloud Computing. In: MDA4ServiceCloud@ECMFA. Paris, France (Jun 2010)
7. Di Rocco, J., Di Ruscio, D., Iovino, L., Pierantonio, A.: Mining metrics for understanding metamodel characteristics. In: MiSE@ICSE (2014)
8. Di Rocco, J., Di Ruscio, D., Iovino, L., Pierantonio, A.: Mining correlations of atl model transformation and metamodel metrics. In: MiSE@ICSE (2015)
9. Manzanares, C.C., Cuadrado, J.S., de Lara, J.: Building mde cloud services with distil. In: CloudMDE@MoDELS (2015)
10. Sanchez Cuadrado, J., Guerra, E., De Lara, J.: Uncovering errors in atl model transformations using static analysis and constraint solving. In: ISSRE. pp. 34–44. IEEE (2014)
11. Van Amstel, M.F., Van Den Brand, M.G.: Model transformation analysis: Staying ahead of the maintenance nightmare. In: ICMT, pp. 108–122. Springer (2011)