Project Number 611125

D3.1 – Public set of transformation benchmarks

Version 1.0
26 March 2014
Final

Public Distribution

ARMINES, BME, Soft-Maint, UNINOVA

Project Partners: **ARMINES**, **Autonomous University of Madrid**, **BME**, **IKERLAN**, **Soft-Maint**, **SOFTEAM**, **The Open Group**, **UNINOVA**, **University of York**

Project Partner Contact Information

ARMINES Massimo Tisi Rue Alfred Kastler 4 44070 Nantes Cedex, France Tel: +33 2 51 85 82 09 E-mail: massimo.tisi@mines-nantes.fr	**Autonomous University of Madrid** Juan de Lara Calle Einstein 3 28049 Madrid, Spain Tel: +34 91 497 22 77 E-mail: juan.delara@uam.es
BME Daniel Varro Magyar Tudosok korutja 2 1117 Budapest, Hungary Tel: +36 146 33598 E-mail: varro@mit.bme.hu	**IKERLAN** Salvador Trujillo Paseo J.M. Arizmendiarrieta 2 20500 Mondragon, Spain Tel: +34 943 712 400 E-mail: strujillo@ikerlan.es
Soft-Maint Vincent Hanniet Rue du Chateau de L'Eraudiere 4 44300 Nantes, France Tel: +33 149 931 345 E-mail: vhanniet@sodifrance.fr	**SOFTEAM** Alessandra Bagnato Avenue Victor Hugo 21 75016 Paris, France Tel: +33 1 30 12 16 60 E-mail: alessandra.bagnato@softeam.fr
The Open Group Scott Hansen Avenue du Parc de Woluwe 56 1160 Brussels, Belgium Tel: +32 2 675 1136 E-mail: s.hansen@opengroup.org	**UNINOVA** Pedro Maló Campus da FCT/UNL, Monte de Caparica 2829-516 Caparica, Portugal Tel: +351 212 947883 E-mail: pmm@uninova.pt
University of York Dimitris Kolovos Deramore Lane York YO10 5GH, United Kingdom Tel: +44 1904 32516 E-mail: dimitris.kolovos@york.ac.uk	

Contents

Document Control

Version	Status	Date
0.1	Document outline	30 November 2013
0.2	First draft	17 January 2014
0.3	Second draft	06 March 2014
0.5	Third draft	10 March 2014
1.0	Final version	26 March 2014

Executive Summary

The scope of this document is to report on the results of Task 3.1 of MONDO and in particular, to present to present the benchmarks and their corresponding artifacts that will be used to evaluate the MONDO query and transformation engines. The deliverable introduces the benchmarks, their originating context and their representative value in the query and transformation landscape.

The artifacts presented in this document are publicly available in a so-called MDE Benchmark Repository. This document describes the repository structure and the way to access its content. The document also provides guidelines for the evolution and maintenance of the repository in the context of the project.

1 Introduction

1.1 Scope of this document

The scope of this deliverable is (i) to describe the different benchmarks that will be used to evaluate and validate the MONDO Query/Transformation engines, and (ii) to discuss how they have been made publicly available. These benchmarks comprise the base benchmarks that will be used to evaluate the scalable query and transformation engine to be released within deliverable D3.2 (M15) and the Cloud-enabled query and transformation engine to be released within deliverable D3.3 (M24).

1.2 Acronyms and abbreviations

DSL	Domain-Specific Language
EMF	Eclipse Modeling Framework
IDE	Integrated Development Environment
MDE	Model-Driven Engineering
MDRE	Model-Driven Reverse Engineering
OMG	Object Management Group
OSP	OpenSourceProjects.eu
RCS	Revision Control System
SVN	Apache Subversion
KDM	Knowledge
GT	Graph Transformations
CAD	Computer Aided Design
AECO	Architecture, Engineering, Construction, and Operations
BIM	Building Information Model
IFC	Industry Foundation Classes
MVD	Model View Definition

2 Context and objectives

Model Driven Engineering (MDE) is a software engineering paradigm that advocates for the rigorous use of (software) models and model manipulation operations (known as model transformations) as the main artifacts in all software engineering activities. This comes from an industrial need to have a regular and homogeneous organization where different facets of a software system may be easily separated or combined when appropriate. The basic assumption of MDE is that models provide a better abstraction level than the third generation programs to manage the complexity of software development (and, in general, any other software-related task). When needed, executable code can be semi-automatically generated from (low-level) models of the system. Adoption of MDE can increase productivity and quality of the software engineering process.

As MDE is increasingly applied to larger and more complex systems, the current generation of modeling and model management technologies are being pushed to their limits in terms of capacity and efficiency. As such, additional research and development is imperative in order to enable MDE to remain relevant with industrial practice and to continue delivering its widely-recognized productivity, quality, and maintainability benefits.

In order to cope with these challenges in MDE, in the scope of this project, we aim to develop the theoretical foundations and an open-source cloud-based platform for scalable modeling and model management.

This section introduces the objectives of the MONDO Project, and more specifically the objectives of the first task of the Work Package 3.

2.1 Objectives of Work Package 3 (Scalable Queries and Transformations)

Any non-trivial MDE project involves querying and manipulation of a substantial number of models. Model manipulation operations are usually implemented as model-to-model transformations that take as input one or more source models and generate as output one or more target models, where target and source models can conform to the same or to different metamodels. Model queries are primary means to extract views and to formally capture and validate well-formedness constraints, design rules and guidelines on the fly. Therefore, scalability of model queries and transformations is a key element in any scalable MDE solution.

Our experience with industrial benchmarks is that current transformation technologies do not scale, which discourages some potential adopters from using MDE. This work package aims at creating a new generation of model querying and transformation technologies that can solve this problem. We propose to build a reactive transformation engine by combining incremental change propagation with lazy computation. We plan to provide the engine with strong parallelization properties, to be able to fully exploit distributed/cloud execution environments.

The goals of this work package can be achieved by the completion of the tasks shown in table 2.

Task name	Month
Task 3.1: Definition of a set of transformations benchmarks	3
Task 3.2: Incremental propagation of model changes	15
Task 3.3: Lazy / on-the-fly / on-demand creation of target models	24
Task 3.4: Queries and Transformations in the cloud: parallel/distributed execution	24
Task 3.5: Infinite/streaming transformations	24
Task 3.6: Integration with scalable persistence mechanisms	24

Table 1: Summary of tasks of Work Package 3

2.2 Objectives of Task 3.1 (Definition of a set of transformations benchmarks)

In order to evaluate the results of the project (and to compare the efficiency of different transformation languages and engines, though this is not our focus right now) we need to define a set of (public) benchmarks for model transformations.

For the completion of this task we have released a web site with a set of public benchmarks that partners and external users can download and execute. Each benchmark consists of a set of models and transformations exercising the scalability properties of the transformation engine that is being implemented in the forthcoming tasks of WP3.

2.3 Objectives of this document

This document serves as a reference for project partners for locating, accessing and executing the transformation benchmarks. The transformation benchmarks are the basic artifacts (mainly models and transformations) used to evaluate the tools supporting efficient, incremental and on-demand model queries and transformations.

3 Existing benchmarks

A few works in literature [36, 37, 21] proposed benchmarks to assist developers in selecting the most suitable query/transformation technology for their application scenarios. However only one of the existing case studies is explicitly dedicated to the manipulation of very large models ([21]) and none is based on benchmarks queries and transformations based on real-world applications. In this section we give a short overview of the related works, while in the next section we introduce the real-world benchmarks proposed by the MONDO project.

3.1 F. Jouault et al. [21]

One of the widely used benchmarks in MDE is Grabats'09 [21]. This case study consists of the definition of program comprehension operators (i.e, queries over source code) using a graph or model transformation tool. One of the main challenges in the definition of program comprehension operators as transformations is scalability with respect to input size.

This case study is divided into two independent tasks:

- a simple filtering query that selects a subgraph of its input according to a given condition
- a complex query that computes a control flow graph and a program dependence graph

These queries are performed over the JDTAST metamodel, the previous Java metamodel used in MoDisco to discover Java projects. It is an updated version of the metamodel used in the *SharenGO Java Legacy Reverse-Engineering* MoDisco use case[1], and closely resembles the class model behind the Eclipse Java Development Tools (JDT)[2] Materials for that case study can be found in the following link [3].

3.2 G. Varro et al. [37]

This work [37] is considered one of the early systematic MDE benchmarks dedicated to Graph Transformations (GT). It proposes a methodology for quantitative benchmarking in order to assess the performance of GT tools. The overall aim is to uniformly measure the performance of a system under a deterministic, parametric, and especially reproducible environment. Four tools participated in the experimentation: AGG, PROGRES, FUJABA and DB. Every benchmarking set is tuned according to some features related on one side to the graph transformation paradigms, and on the other side to the surveyed tools. Thus, a benchmarking set is characterized by turning on/off these features. The following table shows the list of considered features.

[1] `http://www.eclipse.org/gmt/modisco/useCases/JavaLegacyRE/`
[2] `http://www.eclipse.org/jdt/core/index.php`
[3] `http://www.emn.fr/z-info/atlanmod/index.php/GraBaTs_2009_Case_Study`

Paradigm features	Tool dependent features
Pattern size	Parameter Passing
Fan-out	Multiplicities
Matchings size	Parallel execution
Transformation sequence length	ALAP (As Long As Possible)

Table 2: Paradigm and tool features [37]

3.3 M. Amstel et al. [36]

This experiment [36] compares a performance of three model transformation engines: ATL, QVT-R, and QVT-O. This comparison is based on two different transformation examples, targeting meta-models with different structural representations: linear representation (Class2RDBMS) and tree-like representation (RSS2ATOM). The benchmarking set involves randomly generated input models of increasing numbers of elements (up to hundreds of thousands). Like the previous work [37], the benchmark sets are also tuned according to a particular feature such as the size of input models, their complexity (complex interconnection structure) and transformation strategies. In order to study the impact of different implementation strategies in ATL, the Class2RDBMS transformation was implemented in different programming styles. The first one promotes expressing input models navigation in the in the right-hand side of the bindings, the second use ATL attribute helpers, and third uses the imperative part of ATL.

4 MONDO benchmarks

The MONDO benchmarks have the purpose to evaluate and validate the proposed query and transformation engine. This set of benchmarks is made public, with the intention to also help projects beyond MONDO to assess their solutions.

In this section we describe the source, context and properties of the benchmarks proposed by WP3 of the MONDO project. The set of four benchmarks is designed to cover the main use cases for queries and transformations in model-driven applications. Table 3 summarizes the characteristics of the four benchmarks in terms of type of computation (query/transformation) and computational complexity (high/low).

Table 3: Summary of the MONDO WP3 benchmarks

Benchmark	Type	Computational complexity
Train benchmark	query	high
Open-BIM	query/transformation	low
ITM Factory	transformation	high
Transformation zoo	transformation	low

Each one of the benchmarks either includes concrete source models, or a model generator that can be used to produce models of different sizes in a deterministic manner. In the latter case, models of different sizes can be generated, but seeds are provided to drive the deterministic generators in producing the same models for each user.

Each benchmark in the set is given a reference implementation that has to be considered as a specification of the case semantics. Languages and technologies used for each reference implementation may vary, including MDE-specific and general-purpose technologies. One of the future objectives of WP3 is to provide a set of equivalent implementations for each benchmark, to compare the execution performance before/after MONDO.

Finally, while each benchmark defines the source/target relation for each query or transformation, other aspects of the transformation runtime semantics are left open. For instance high-complexity benchmarks can be run in batch or incremental mode, to test different execution properties of the tool under study.

4.1 Train Benchmark (BME)

Scalability issues in model-driven engineering arise due to the increasing complexity of modeling workloads. This complexity comes from two main factors: (i) *instance model sizes* can quickly grow as the complexity of systems-under-design is increasing, (ii) increasing *feature sophistication* in toolchains, such as complex model validation or transformations.

One of the the most computationally expensive tasks in modeling applications is the evaluation of *model queries*. While there are a number of existing benchmarks for queries over relational databases

[34] or graph stores [9, 31], modeling tool workloads are significantly different. Specifically, modeling tools use much more complex queries than typical transactional systems [19], and the real world performance is affected by response time (i.e. execution time for a specific operation such as validation or transformation) than throughput (i.e. the amount of parallel transactions).

To address this challenge, the Train Benchmark [35, 1] is a macro benchmark that aims to measure batch and incremental query evaluation performance, in a scenario that is specifically modeled after *model validation* in (domain-specific) modeling tools: at first, the entire model is validated, then after each model manipulation (e.g., the deletion of a reference) is followed by an immediate re-validation. The benchmark records execution times for four phases:

1. During the *read* phase, the instance model is loaded from hard drive to memory. This includes the parsing of the input as well as initializing data structures of the tool.

2. In the *check* phase, the instance model is queried to identify invalid elements. This can be as simple as reading the results from cache, or the model can be traversed based on some index. By the end of this phase, erroneous objects need to made available in a list.

3. In the *edit* phase, the model is modified to simulate effects of manual user edits. Here the size of the change set can be adjusted to correspond to small manual edits as well as large model transformations.

4. The re-validation of the model is carried out in the *re-check* phase similarly to the *check* phase.

The Train Benchmark computes two derived results based on the recorded data: (1) *batch validation time* (the sum of the *read* and check phases) represents the time that the user must wait to start to use the tool; (2) *incremental validation time* consists of the *edit* and *re-check* phases performed 100 times, representing the time that the user spent waiting for the tool validation.

4.1.1 Queries and transformations

Queries are defined informally in plain text (in a tool independent way) and also formalized using the standard OCL language as a reference implementation (available on the benchmark website [1]). The queries range from simple attribute value checks to complex navigation operations consisting of several (4+) joins.

The functionally equivalent variants of these queries are formalized using the query language of different tools applying tool based optimizations. As a result, all query implementations must return (the same set of) invalid instance model elements.

In the *edit* phase, the model is modified to change the result set to be returned by the query in the re-check phase. For simulating manual modifications, the benchmark always performs a hundred random edits (fixed low constant) which increases the number of erroneous elements. An edit operation only modifies one model element at a time - more complex model manipulation is modelled as a series of edits.

4.1.2 Instance models

The Train Benchmark uses a domain-specific model of a railway system that originates from the MOGENTES EU FP7 project, where both the metamodel and the well-formedness rules were defined by railway domain experts. This domain enables the definition of both simple and more complex model queries while it is uncomplicated enough to incorporate solutions from other technological spaces (e.g. ontologies, relational databases and RDF). This allows the comparison of the performance aspects of wider range of query tools from a constraint validation viewpoint.

The instance models are systematically and reproducibly generated based on the metamodel and the defined complex model queries: small instance model fragments are generated based on the queries, and then they are placed, randomized and connected to each other. The methodology takes care of controlling the number of matches of all defined model queries. To break symmetry, the exact number of elements and cardinalities are randomized (with a fixed seed to ensure deterministic reproducibility). In the benchmark measurements, model sizes ranging from a few elements to 13 million elements (objects and references combined) are considered.

This brings artificially generated models *closer to real world instances*, and *prevents query tools from efficiently storing* or caching of instance models. This is important in order to reduce the sampling bias of the experiments. During the generation of the railway system model, errors are injected at random positions. These errors can be found in the check phase of the benchmark, which are reported, and can be corrected during the edit phase. The initial number of constraint violating elements is low (<1% of total elements).

4.1.3 Evaluation of measurements

The Train Benchmark defines a Java-based framework and application programming interface that enables the integration of additional metamodels, instance models, query implementations and even new benchmark scenarios (that may be different from the original 4-phase concept). The default implementation contains a benchmark suite for queries implemented in Java, Eclipse OCL and EMF-IncQuery.

Measurements are recorded automatically in a machine-processable format (CSV) that is automatically processed by R [3] scripts. An extended version of the Train Benchmark [19] features several (instance model, query-specific and combined) *metrics* that can be used to characterize the "difficulty" of benchmark cases numerically, and – since they can be evaluated automatically for other domain/model/query combinations – allow to compare the benchmark cases with other real-world workloads.

4.2 Open-BIM (UNINOVA)

The construction industry has traditionally communicated building construction information (sites, buildings, floors, spaces, and equipment and their attributes) through drawings with notes and specifications. BIM (Building Information Model), a CAD (Computer Aided Design) method, came to automate that process and enhance its operability according to different tools, actors, etc. within the

AECO (Architecture, Engineering, Construction, and Operations) industry. A BIM model is a multi-disciplinary data specific model instance which describes all the information pertinent to a building and its components. It is described using the IFC (Industry Foundation Classes) specification, a freely available format to describe, exchange, and share information typically used within the building and facility management industry sector.

Intrinsically, the IFC model is expressed using the EXPRESS data definition language, defined as ISO10303-11 by the ISO TC184/SC4 committee. EXPRESS representations are known to be compact and well suited to include data validation rules within the data specification.

4.2.1 Queries and transformations

The Open-BIM use case includes a query benchmark and a transformation benchmark:

IFC well-formedness rules The IFC format describes, using the EXPRESS language, the set of well-formed IFC models. The EXPRESS notation includes, in a single specification, 1) the set of element types and properties allowed in the data file, 2) the set of well-formedness constraints that have to be globally satisfied. When representing an IFC model in an EMF format these two parts of the specification translate to 1) an Ecore metamodel defining element and property types, 2) a set of constraints encoding the well-formedness rules.

This benchmark involves validating the set of well-formedness rules (2) over a given model, model that conforms to the IFC Ecore metamodel (1). An Ecore metamodel is provided, coming from the open-source BIMServer [4] project. The well-formedness rules are given in EXPRESS format and are meant to be translated to the query technology under evaluation.

IFC2BIMXML BIMXML [5] is an XML format describing building data in a simplified spatial building model. The BIMXML XML Schema was developed as an alternative to full scale IFC models to simplify data exchanges between various AEC applications and to connect Building Information Models through Web Services. It is currently used by several primary actors in the CAD construction domain, including Onuma System (Onuma, Inc.), DDS Viewer (Data Design System), vROC, Tokmo, BIM Connect, and various plugins for CAD Applications (Revit, SketchUp, ArchiCAD). The BIMXML specification includes an XML Schema[6] and documents the translation rules from the full IFC specification.

This benchmark involves performing the translation of a full IFC model into the BIMXML format. Ecore metamodels for the source and target models are provided.

4.2.2 Instance models

UNINOVA has contributed 4 real-world IFC data files with size ranging from 30MB to 2GB. All the files represent real construction projects and were used in production context. They contain a precise

[4] https://github.com/opensourceBIM/BIMserver
[5] http://bimxml.org/
[6] http://www.bimxml.org/xsd/001/bimxml-001.xsd

and detailed information about actors, approvals, buildings etc. The data files, in EXPRESS format, are translated into EMF models so that they can be used by EMF-based query and transformation tools.

4.3 ITM Factory (Soft-Maint)

MoDisco (Model Discovery), the open-source Model Driven Reverse Engineering project lead by Soft-Maint. It uses a two steps approach **model discovery** and **model understanding**. The initial step consists in obtaining a model representation of a specific view on the legacy system, whereas, the second involves extracting useful information from the discovered model. So far, MoDisco is facing two main limitations, (i) efficiency in handling large models, especially these involved in reverse engineering legacy systems, (ii) participation scenarios of such large models(e.g. long transformation chains). The lack of collaboration capabilities also hampers the use of MoDisco in large scale projects.

To overcome these issues Soft-Maint is designing a new ITM (IT Modernization) Factory on top of MoDisco containing a set of customizable cartridges. The ITM Factory framework (IDE) will be contributed to the Eclipse Platform under the Eclipse Public License [33].

Soft-Maint has contributed three benchmarks, each addressing a different phase in a model-driven reverse engineering process.

4.3.1 Queries and transformations

Java2KDM This transformation takes place at beginning of almost every modernization process of a Java legacy system, it comes just after the discovery of the Java model from Java projects (plugins) using the MoDisco Java Discoverer (see figure 1). This transformation generates a KDM [28] (Knowledge Discovery Metamodel) model that defines common metadata required for deep semantic integration of application life-cycle management tools. Java2KDM transformation is useful when the only available information on a Java source code is contained in a Java model, even without the source code it is then possible to get a KDM representation. This intermediate model provides useful and precise information that can be used to produce additional types of models.

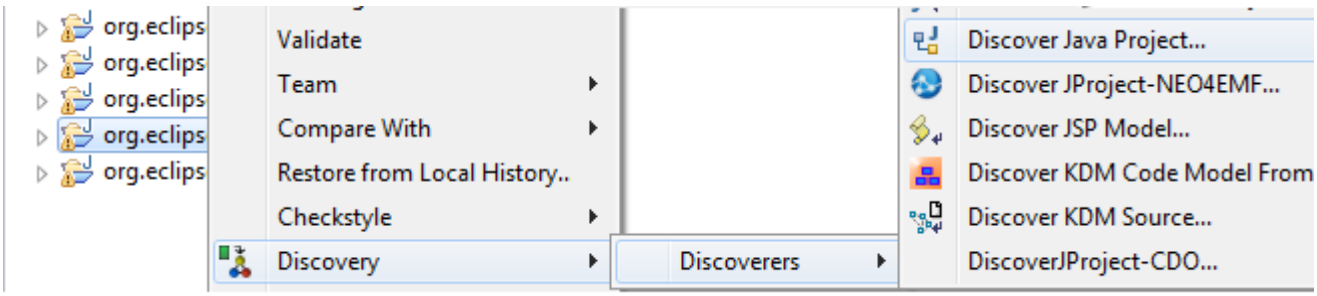


Figure 1: Java model discovery using MoDisco

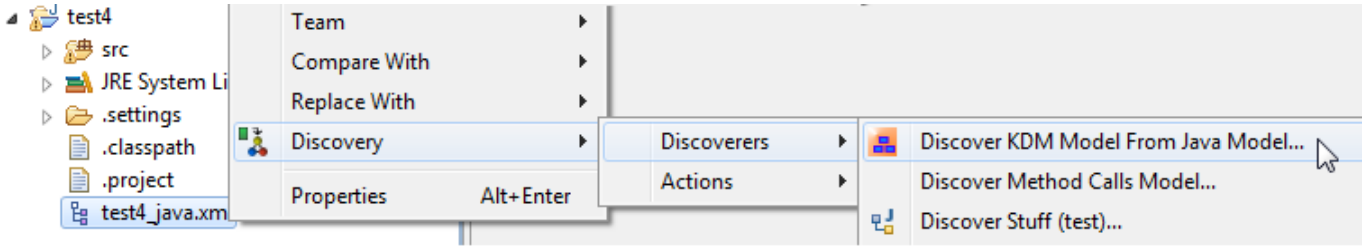


Figure 2: KDM model discovery using MoDisco

KDM2UML Based on the previously generated model, this transformation generates a UML diagram in order to allow integrating KDM-compliant tools (i.e. discoverers) with UML-compliant tools (e.g. modelers, model transformation tools, code generators, etc.).

Java code quality This set of code quality transformations identify well-known anti-patterns in Java source code and fix the corresponding issues by a model transformation. The input format of the transformations is a model conforming to the Java metamodel. For a specification for the transformations we refer the reader to the implementations of these fixes in well-known code-analysis tools like CheckStyle and PMD. Table 4 contains the list of fixes and for each one the link to the corresponding documentation.

4.3.2 Instance models

Thanks to the MoDisco Java discoverer, we are able to extract Java models up to 1.3GB, that conform to the Java metamodel [23] defined in MoDisco (refinement of the JDTAST metamodel) as depicted in Fig. 1. Those models are the input of the Java2KDM and Java code quality transformations, while, KDM output models are inputs for the KDM2UML transformation. It is also possible to retrieve directly KDM models using MoDisco, please see Fig. 2. Because of confidentiality agreements, Soft-Maint is not able to publish instance models derived from their commercial projects. For this reason we choose to derive instance models from the source code of open-source projects, specifically from the Eclipse JDT plugins (org.eclipse.jdt.*). This does not affect the relevance of the benchmark, as these plugins are written by experienced developers with a quality standard that is comparable to commercial projects.

Table 5 depicts the different models we recovered against the discovered plugins.

4.4 ATL Transformation Zoo (ARMINES)

The ATL project maintains a repository of ATL transformations produced in industrial and academic contexts (ATL Transformation Zoo [32]). These transformations are representative of the use of model transformations for low-complexity tasks (i.e., low number of transformation rules, lack of recursion, etc. . .).

Table 4: List of Java code quality fixes

Rule	Documentation
ConstantName	http://checkstyle.sourceforge.net/config_naming.html#ConstantName
LocalFinalVariableName	http://checkstyle.sourceforge.net/config_naming.html#LocalFinalVariableName
LocalVariableName	http://checkstyle.sourceforge.net/config_naming.html#LocalVariableName
MemberName	http://checkstyle.sourceforge.net/config_naming.html#MemberName
MethodName	http://checkstyle.sourceforge.net/config_naming.html#MethodName
PackageName	http://checkstyle.sourceforge.net/config_naming.html#PackageName
ParameterName	http://checkstyle.sourceforge.net/config_naming.html#ParameterName
StaticVariableName	http://checkstyle.sourceforge.net/config_naming.html#StaticVariableName
TypeName	http://checkstyle.sourceforge.net/config_naming.html#TypeName
AvoidStarImport	http://checkstyle.sourceforge.net/config_imports.html#AvoidStarImport
UnusedImports	http://checkstyle.sourceforge.net/config_imports.html#UnusedImports
RedundantImport	http://checkstyle.sourceforge.net/config_imports.html#RedundantImport
ParameterNumber	http://checkstyle.sourceforge.net/config_sizes.html#ParameterNumber
ModifierOrder	http://checkstyle.sourceforge.net/config_modifier.html#ModifierOrder
RedundantModifier	http://checkstyle.sourceforge.net/config_modifier.html#RedundantModifier
AvoidInlineConditionals	http://checkstyle.sourceforge.net/config_coding.html#AvoidInlineConditionals
EqualsHashCode	http://checkstyle.sourceforge.net/config_coding.html#EqualsHashCode
HiddenField	http://checkstyle.sourceforge.net/config_coding.html#HiddenField
MissingSwitchDefault	http://checkstyle.sourceforge.net/config_coding.html#MissingSwitchDefault
RedundantThrows	http://checkstyle.sourceforge.net/config_coding.html#RedundantThrows
SimplifyBooleanExpression	http://checkstyle.sourceforge.net/config_coding.html#SimplifyBooleanExpression
SimplifyBooleanReturn	http://checkstyle.sourceforge.net/config_coding.html#SimplifyBooleanReturn
FinalClass	http://checkstyle.sourceforge.net/config_design.html#FinalClass
InterfaceIsType	http://checkstyle.sourceforge.net/config_design.html#InterfaceIsType
VisibilityModifier	http://checkstyle.sourceforge.net/config_design.html#VisibilityModifier
FinalParameters	http://checkstyle.sourceforge.net/config_misc.html#FinalParameters
LooseCoupling	http://pmd.sourceforge.net/pmd-5.1.0/rules/java/typeresolution.html#LooseCoupling
SignatureDeclareThrowsException	http://pmd.sourceforge.net/pmd-5.1.0/rules/java/typeresolution.html#SignatureDeclareThrowsException
DefaultLabelNotLastInSwitchStmt	http://pmd.sourceforge.net/pmd-5.1.0/rules/java/design.html#DefaultLabelNotLastInSwitchStmt
EqualsNull	http://pmd.sourceforge.net/pmd-5.1.0/rules/java/design.html#EqualsNull
CompareObjectsWithEquals	http://pmd.sourceforge.net/pmd-5.1.0/rules/java/design.html#CompareObjectsWithEquals
PositionLiteralsFirstInComparisons	http://pmd.sourceforge.net/pmd-5.1.0/rules/java/design.html#PositionLiteralsFirstInComparisons
UseEqualsToCompareStrings	http://pmd.sourceforge.net/pmd-5.1.0/rules/java/strings.html#UseEqualsToCompareStrings
IntegerInstantiation	http://pmd.sourceforge.net/pmd-5.1.0/rules/java/migrating.html#IntegerInstantiation
ByteInstantiation	http://pmd.sourceforge.net/pmd-5.1.0/rules/java/migrating.html#ByteInstantiation
ShortInstantiation	http://pmd.sourceforge.net/pmd-5.1.0/rules/java/migrating.html#ShortInstantiation
LongInstantiation	http://pmd.sourceforge.net/pmd-5.1.0/rules/java/migrating.html#LongInstantiation
BooleanInstantiation	http://pmd.sourceforge.net/pmd-5.1.0/rules/java/migrating.html#BooleanInstantiation
SimplifyStartsWith	http://pmd.sourceforge.net/pmd-5.1.0/rules/java/optimizations.html#SimplifyStartsWith
UnnecessaryReturn	http://pmd.sourceforge.net/pmd-5.1.0/rules/java/unnecessary.html#UnnecessaryReturn
UnconditionalIfStatement	http://pmd.sourceforge.net/pmd-5.1.0/rules/java/basic.html#UnconditionalIfStatement
UnnecessaryFinalModifier	http://pmd.sourceforge.net/pmd-5.1.0/rules/java/unnecessary.html#UnnecessaryFinalModifier

Set1	org.eclipse.jdt.apt.pluggable.core
Set2	Set1 + org.eclipse.jdt.apt.pluggable.core
Set3	Set2 + org.eclipse.jdt.core + org.eclipse.jdt.compiler.* + org.eclipse.jdt.apt.core
Set4	Set3 + org.eclipse.jdt.core.manipulation + org.eclipse.jdt.launching + org.eclipse.jdt.ui + org.eclipse.jdt.debug
Set5	org.eclipse.jdt.* (all jdt plugins)

Table 5: Discovered plugins per set

In this benchmark we select a subset of the transformations in the ATL Transformation Zoo based on their quality level. We specifically include only transformations that may be realistically used in production environments. We automatize the sequential execution of this subset and the generation of performance analysis data.

4.4.1 Queries and transformations

Ant to Maven Ant [5] is an open source build tool (a tool dedicated to the assembly of the different pieces of a program) from the Apache Software Foundation. Ant is the most commonly used build tool for Java programs. Maven [6] is another build tool created by the Apache Software Foundation. It is an extension of Ant because ant Tasks can be used in Maven. The difference from Ant is that a project can be reusable. This transformation [13] generates a file for the build tool Maven starting from a file corresponding to the build tool Ant.

CPL2SPL CPL (Call Processing Language) is a standard scripting language for the SIP (Session Initiation Protocol) protocol. It offers a limited set of language constructs. CPL is supposed to be simple enough so that it is safe to execute untrusted scripts on public servers [20]. SPL programs are used to control telephony agents (e.g. clients, proxies) implementing the SIP (Session Initiation Protocol) protocol. Whereas, the CPL has an XML-based syntax, the *CPL2SPL* transformation [15], provides an implementation of CPL semantics by translating CPL concepts into their SPL equivalent concepts.

Graphcet2PetriNet This transformation[16] establishes a bridge between grafcet [10], and petri nets [27]. It provides an overview of the whole transformation sequence that enables to produce an XML petri net representation from a textual definition of a grafcet in a PNML format, and the other way around.

IEEE1471 to MoDAF This transformation example [4] realizes the transformation between IEEE1471-2000 [22] and MoDAF-AV [8]. The IEEE1471 committee prescribes a recommended practice for the design and the analysis of Architecture of Software Intensive Systems. It fixes a terminology for System, Architecture, Architectural Description, Stakeholder, Concerns, View ,and Viewpoints concepts. MoDAF (Ministry of Defense Architecture Framework) is based on the DoDAF

(Department of Defense Architecture Framework). DoDAF is a framework to design C4ISR systems. MoDAF-AV (Architecture View) used several concepts defined in the IEEE1471.

Make2Ant Make (the most common build tool) is based on a particular shell or command interface and is therefore limited to the type of operating systems that use that shell. Ant uses Java classes rather than shell-based commands. Developers use XML to describe the modules in their program build. This benchmark [17] describes a transformation from a Makefile to an Ant file.

MOF2UML The MOF (Meta Object Facility) citemof is an OMG standard enabling the definition of metamodels through common semantics. The UML (Unified Modeling Language) Core standard is the OMG common modeling language. Although, MOF is primarily designed for metamodel definitions and UML Core for the design of models, the two standards define very close notions. This example [18] describes a transformation enabling to pass from the MOF to the UML semantics. The transformation is based on the UML Profile for MOF OMG specification.

OCL2R2ML The OCL to R2ML transformation scenario [25] describes a transformation from OCL (Object Constraint Language) [29] metamodel (with EMOF metamodel) into a R2ML (REWERSE I1 Rule Markup Language) metamodel. The Object Constraint Language (OCL) is a language that enables one to describe expressions and constraints on object-oriented (UML and MOF) models and other object modeling artifacts. An expression is an indication or specification of a value. A constraint is a restriction on one or more values of (part of) an object-oriented model or system. REWERSE I1 Rule Markup Language (R2ML) is a general web rule markup language, which can represent different rule types: integrity, reaction, derivation and production. It is used as pivotal metamodel to enable sharing rules between different rule languages, in this case with the OCL.

UML2OWL This scenario [26] presents an implementation of the OMG's ODM specification. This transformation is used to produce an OWL ontology, and its *OWL Individuals* from an UML Model, and its *UML Instances*.

BibTeXML to DocBook The *BibTeXML to DocBook* example [14] describes a transformation of a BibTeXML [30] model to a DocBook [38] composed document. BibTeXML is an XML-based format for the BibTeX bibliographic tool. DocBook, as for it, is an XML-based format for document composition.

DSL to EMF This example [7] provides a complete overview of a transformation chain example between two technical spaces: Microsoft DSL Tools [24] and EMF. The aim of this example is to demonstrate the possibility to exchange models defined under different technologies. In particular, the described bridges demonstrate that it should be possible to define metamodels and models using both Microsoft DSL Tools and Eclipse EMF technologies. The bridge between MS/DSL and EMF spans two levels: the metamodel and model levels. At the level of metamodels, it allows to transform MS/DSL domain models to EMF metamodels. At the level of models, the bridge allows transforming

MS/DSL models conforming to domain models to EMF models conforming to EMF metamodels. At both levels, the bridge operates in both directions. A chain of ATL-based transformations is used to implement the bridge at these two levels. The benefit of using such a bridge is the ability to transpose MS/DSL work in EMF platform, and inversely.

4.4.2 Instance models

For the aforementioned transformations, we do not have large enough models that conform to the respective metamodels, and as such we make use of a probabilistic model instantiator. This instantiator takes as parameter a generation configuration specified by the user. A generation configuration holds information such as 1) meta-classes that should (not) be involved in the generation, 2) probability distributions to establish how many instances should be generated for each metaclass, and which values should be assigned to structural features. We provide a default generation configuration, using uniform probability distributions for each meta-class and structural feature. For some transformations we provide ad-hoc probability distributions, exploiting domain knowledge over the instances of the corresponding metamodel.

A generation configuration may come also with a seed that makes the generation deterministic and reproducible. For each one of the built-in generation configurations we provide a seed, producing the exact set of models we used during our experimentation.

5 The MDE Benchmark repository

The *MDE Benchmark repository* is the central storage area where the artifacts of the benchmarks are archived for public access. These artifacts, mainly text files, comprise large models and metamodels – typically represented in their XMI serialization – and model transformations. To increase the visibility of these files we have chosen to make them publicly available through the OpenSourceProjects.eu (OSP) platform. The OSP platform is a software forge dedicated to hosting Open Source projects created within EU research projects.

The OSP platform provides, among other tools, a Git revision control system (RCS). Git is a distributed system which allows maintaining current and historical versions of files. Moreover, Git repositories hosted in the OSP platform can be easily navigated with a standard web browser.

The site of the MONDO project within the OSP platform is `http://opensourceprojects.eu/p/mondo`.

5.1 Repository structure

The public deliverables of the MONDO project are published in the MONDO Git of the OSP Platform (i.e. `http://opensourceprojects.eu/p/mondo/_list/git`). Each deliverable is available as a dedicated repository (set of repositories) in the MONDO Git.

The *MDE Benchmark repository* corresponds to deliverable D3.1, and is located in the URL `http://opensourceprojects.eu/p/mondo/d31-transformation-benchmarks/`. Inside this repository every top level resource corresponds to a git submodule, each, representing a different case study held in a separate git repository.

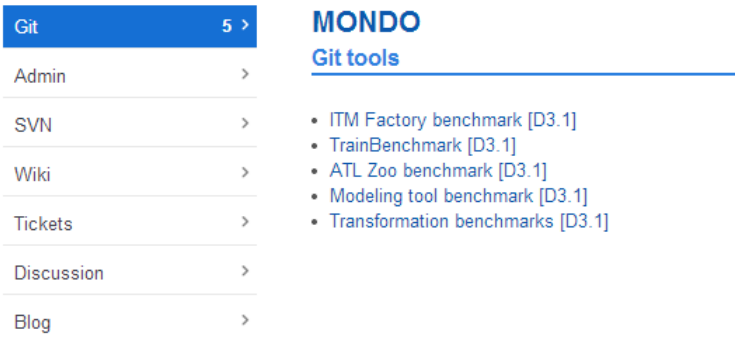


Figure 3: Organization of the repositories

Related resources for benchmarking a specific feature of a transformation engine are grouped in *projects*. A *project* is a self-contained entity, and can be considered as the basic benchmarking unit. *Projects* are structured as follows[7]:

`/README` (Mandatory) — This is a human-readable file describing the details of the test case, the file and directory structure, and any other important information (e.g. test cases can evolve and

[7]Entries ending with an asterisk (⋆) represent directories

additional information not considered at the point of writing this document may be needed for executing the benchmark).

`/doc/*` (Optional)— This directory stores the documentation about the test case. The documentation of a test case may include, among other information, a detailed description of the test case, the foundations of the feature under testing, the building and execution instructions, etc.

`/transformations/*` (Optional) — This directory stores the transformation(s), in source code form, that stress the feature under testing.

`/models/*` (Optional) — This directory contains the model and metamodel descriptions involved in the test transformation(s).

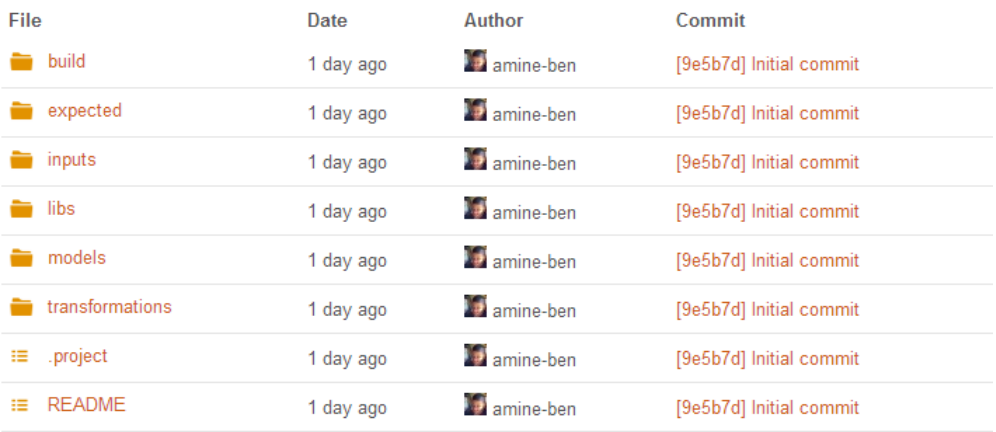
`/data/inputs/*` (Optional) — This directory contains the input data to be used by the test case(s).

`/data/expected/*` (Optional) — In this directory we store the files that contain the expected values that must be returned by the transformation. The expected data are compared with the actual output of the transformation to determine if the test execution has been successful or not.

`/src/*` (Mandatory) — In some situations, test cases may require additional code (such as Java code) to be executed. For example, test cases may be automatically launched with the help of third party libraries (such as JUnit), or test cases may execute external code following a black-box scheme. In this situations the additional code should be placed inside the `/src` directory.

`/libs/*` (Optional) — This directory is used to store any additional third party library (usually a binary file) required by the test case.

`/build/*` (Optional) — Build and execution scripts should be placed under the `/build` directory. Examples of such scripts are Ant files [5], Maven files [6], Makefiles [12], bash shell scripts [11], etc.



File	Date	Author	Commit
build	1 day ago	amine-ben	[9e5b7d] Initial commit
expected	1 day ago	amine-ben	[9e5b7d] Initial commit
inputs	1 day ago	amine-ben	[9e5b7d] Initial commit
libs	1 day ago	amine-ben	[9e5b7d] Initial commit
models	1 day ago	amine-ben	[9e5b7d] Initial commit
transformations	1 day ago	amine-ben	[9e5b7d] Initial commit
.project	1 day ago	amine-ben	[9e5b7d] Initial commit
README	1 day ago	amine-ben	[9e5b7d] Initial commit

Figure 4: Structure of the ITMFactory benchmark

5.2 Using the repository

The benchmarks can be accessed through a web browser through the URL `http://opensourceprojects.eu/p/mondo/_list/git`.

Besides the web interface, it is possible to use Git to get a local copy of a benchmark. To download a benchmark it is only necessary to clone the repository. A command line example to clone a case study to a local directory called `case-study` is:

```
1 git clone https://opensourceprojects.eu/git/p/mondo/case-study-path
    local-folder-name
```

In order to download the full benchmarks, you can execute this command line :

```
1 git clone --recursive https://opensourceprojects.eu/git/p/mondo/d31-
    transformation-benchmarks local-folder-name
```

More configuration options for Git can be accessed via the OSP Web interface at `http://opensourceprojects.eu/p/mondo/admin/tools` within the available options from Git tool's block (Admin -> tools).

A full description of Git commands can be found in [2]. It is also possible to clone a benchmark repository from eclipse using EGit[8]. To do so, please follow the next steps :

1. Click on **File** → **Import**, and then **Projects from Git**, see Fig. 5
2. Click on URI
3. Enter the repository URI as depicted in Fig. 6
4. Enter the destination directory (see Fig. 7). If you are cloning the parent repository please check *Clone submodules*
5. Finally, select the projects you want to import as illustrated by Fig. 8. If you are cloning the parent repository please check *Search for nested projects*

5.3 Submission guidelines

In order to increase the quality and soundness of the test cases available in the *MDE Benchmark repository*, we plan to keep it open to further submissions from the MDE community.

We have defined a simple set of guidelines that must be followed when contributing a new case study to guarantee that the quality of the repository is maintained. Specifically:

- New contributions must include a comprehensive description of the case study. A rationale for its inclusion must be provided, specially focusing on the differential aspects of the proposed case study, compared to the already included benchmarks.

[8] `https://www.eclipse.org/egit/`

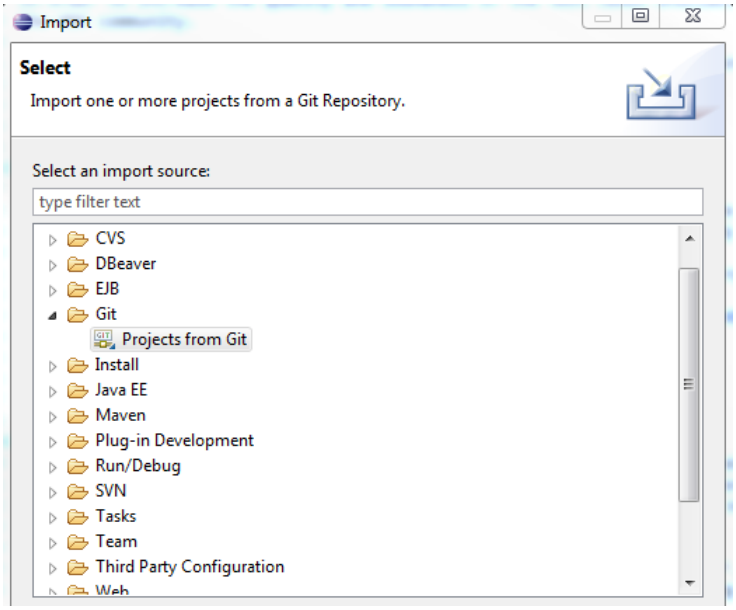


Figure 5: Import git projects

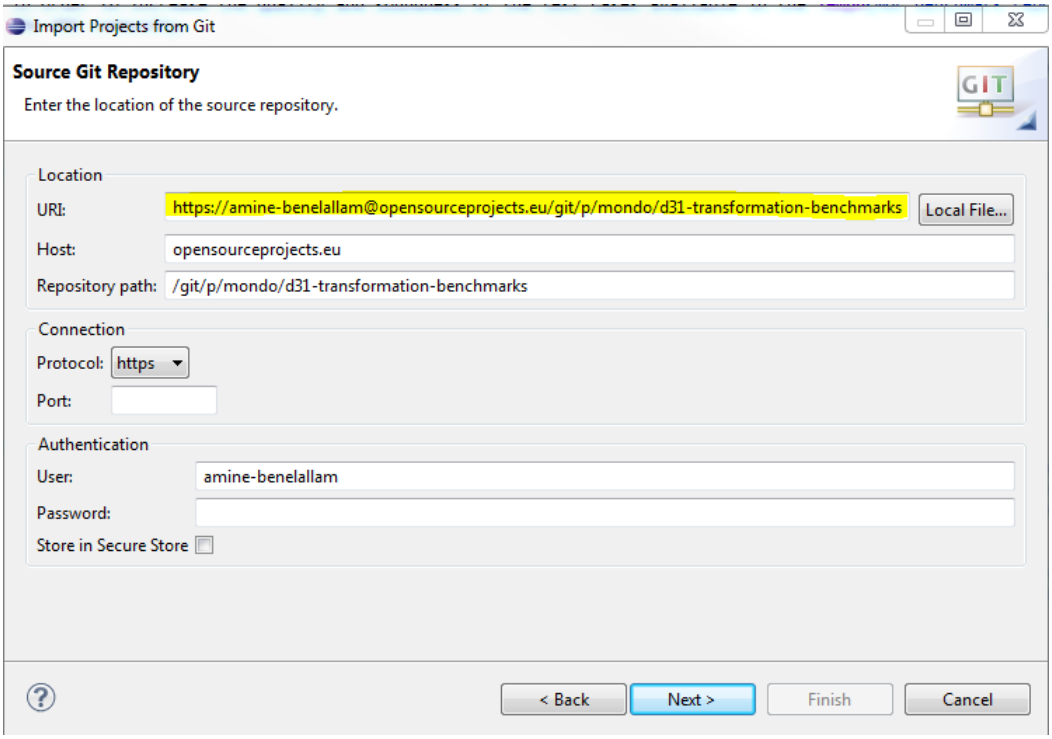


Figure 6: Enter repository URI

- The sources, models, documentation and utility scripts must be organized as described in section 5.1.
- Contributions must be sent to the address `mondo_team@opengroup.org` for their evaluation and approval.

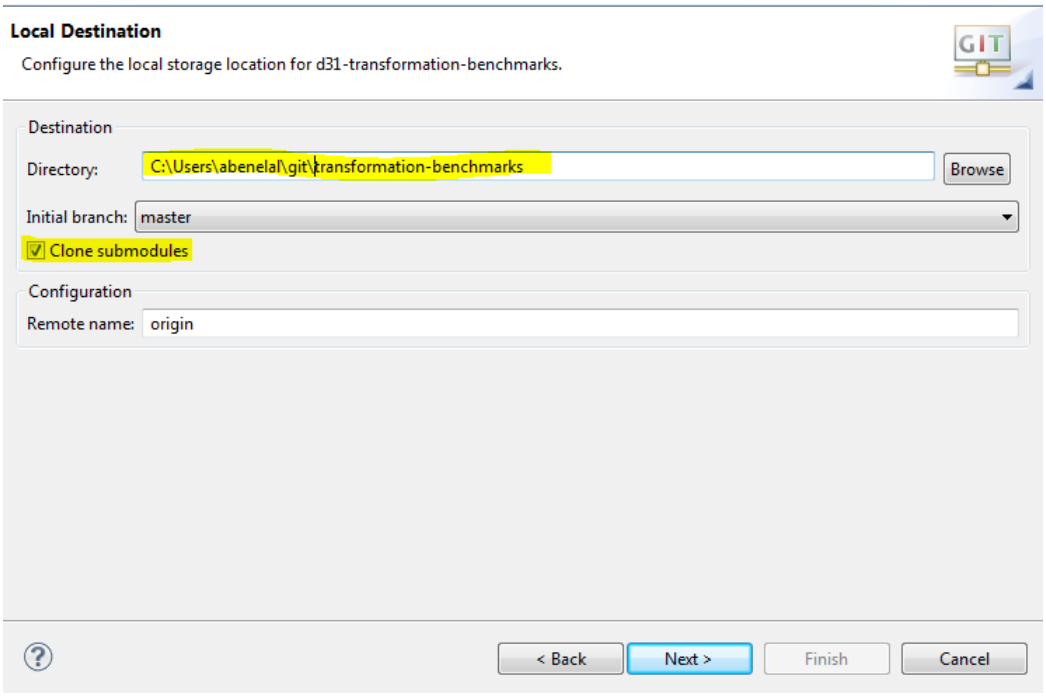


Figure 7: Enter destination folder

6 Conclusion

This document introduces the benchmark set that has been selected for evaluating the performance of the querying/transformation engines envisioned in the context of WP3, and for comparing them to state-of-the-art solutions. Two of the four benchmarks included come from industrial contexts and two from successful open-source projects in the MDE community. Their different characteristics make the set representative of a wide set of applications of model queries and transformations in MDE today.

This set represents the initial baseline for experimentation in MONDO WP3, and it is publicly available to the MDE community, to foster collaboration with external solution providers. The repository is finally open to extension, allowing the community to submit proposals for additional benchmarks, to be evaluated for inclusion by the MONDO consortium.

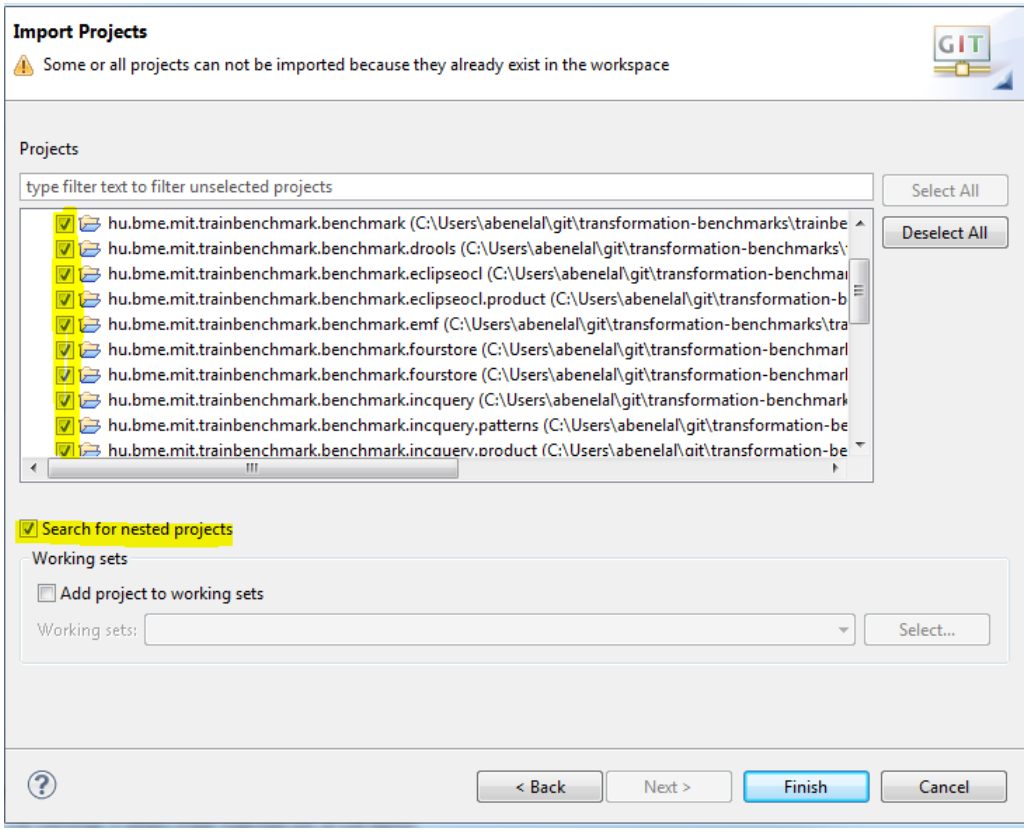


Figure 8: Select the projects to be imported

References

[1] The train benchmark website. https://incquery.net/publications/trainbenchmark/full-results, 2013.

[2] Git - fast version control system, 2014. URL: http://git-scm.com/.

[3] The R project for statistical computing. http://www.r-project.org/, 2014.

[4] Albin Jossic. ATL Transformation Example: IEEE1471 to MoDAF, 2005. URL: http://www.eclipse.org/atl/atlTransformations/IEEE1471_2_MoDAF/IEEE1471_2_MoDAF.doc.

[5] Apache. Apache ant, 2014. URL: http://ant.apache.org/.

[6] Apache. Apache maven project, 2014. URL: http://maven.apache.org/.

[7] ATLAS group – LINA & INRIA. The Microsoft DSL to EMF ATL transformation , 2005. URL: http://www.eclipse.org/atl/atlTransformations/DSL2EMF/ExampleDSL2EMF%5Bv00.01%5D.pdf.

[8] Bill Biggs. Ministry of defence architectural framework (modaf). 2005.

[9] Christian Bizer and Andreas Schultz. The Berlin SPARQL benchmark. *International Journal on Semantic Web & Information Systems*, 5(2):1–24, 2009.

[10] René David. Grafcet: A powerful tool for specification of logic controllers. *Control Systems Technology, IEEE Transactions on*, 3(3):253–268, 1995.

[11] GNU. Bourne-Again SHell manual, 2014. URL: http://www.gnu.org/software/bash/manual/.

[12] GNU. GNU 'make', 2014. URL: http://www.gnu.org/software/make/manual/.

[13] INRIA. ATL Transformation Example: Ant to Maven, 2005. URL: http://www.eclipse.org/atl/atlTransformations/Ant2Maven/ExampleAnt2Maven%5Bv00.01%5D.pdf.

[14] INRIA. ATL Transformation Example: BibTeXML to DocBook, 2005. URL: http://www.eclipse.org/atl/atlTransformations/BibTeXML2DocBook/ExampleBibTeXML2DocBook%5Bv00.01%5D.pdf.

[15] INRIA. ATL Transformation Example: CPL to SPL, 2005. URL: http://www.eclipse.org/atl/atlTransformations/CPL2SPL/README.txt.

[16] INRIA. ATL Transformation Example: Grafcet to Petri Net, 2005. URL: http://www.eclipse.org/atl/atlTransformations/Grafcet2PetriNet/ExampleGrafcet2PetriNet[v00.01].pdf.

[17] INRIA. ATL Transformation Example: Make to Ant, 2005. URL: http://www.eclipse.org/atl/atlTransformations/Make2Ant/ExampleMake2Ant[v00.01].pdf.

[18] INRIA. ATL Transformation Example: MOF to UML, 2005. URL: http://www.eclipse.org/atl/atlTransformations/MOF2UML/ExampleMOF2UML[v00.01].pdf.

[19] Benedek Izsó, Zoltán Szatmári, Gábor Bergmann, Ákos Horváth, and István Ráth. Towards precise metrics for predicting graph query performance. In *2013 IEEE/ACM 28th International Conference on Automated Software Engineering (ASE)*, pages 412–431, Silicon Valley, CA, USA, 11/2013 2013. IEEE.

[20] Frédéric Jouault, Jean Bézivin, Charles Consel, Ivan Kurtev, Fabien Latry, et al. Building dsls with amma/atl, a case study on spl and cpl telephony languages. In *ECOOP Workshop on Domain-Specific Program Development*, 2006.

[21] Frédéric Jouault, J Sottet, et al. An amma/atl solution for the grabats 2009 reverse engineering case study. In *5th International Workshop on Graph-Based Tools, Zurich, Switzerland*, 2009.

[22] Eric Jouenne and Véronique Normand. Tailoring ieee 1471 for mde support. In *UML Modeling Languages and Applications*, pages 163–174. Springer, 2005.

[23] MIA-Software. Modiscojava metamodel (knowledge discovery metamodel) version 1.3, 2012. URL: http://dev.eclipse.org/svnroot/modeling/org.eclipse.mdt.modisco/main/branches/0_11/org.eclipse.gmt.modisco.java/model/java.ecore.

[24] Microsoft Corp. The DSL tools, 2014. URL: http://msdn.microsoft.com/vstudio/DSLTools/.

[25] Milan Milanovic. ATL Transformation Example: OCL to R2ML, 2005. URL: http://www.eclipse.org/atl/atlTransformations/OCL2R2ML/README.txt.

[26] Milan Milanovic. ATL Transformation Example: UML to OWL, 2005. URL: http://www.eclipse.org/atl/atlTransformations/UML2OWL/README.txt.

[27] Tadao Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[28] OMG (Object Management Group). Kdm (knowledge discovery metamodel) version 1.3, 2011. URL: http://www.omg.org/spec/KDM/1.3/.

[29] OMG (Object Management Group). Ocl (object constraint language) v2.0, 2011. URL: http://www.omg.org/spec/OCL/2.0/PDF.

[30] Luca Previtali, Brenno Lurati, and Erik Wilde. Bibtexml: An xml representation of bibtex. In Vincent Y. Shen, Nobuo Saito, Michael R. Lyu, and Mary Ellen Zurko, editors, *WWW Posters*, 2001.

[31] Michael Schmidt, Thomas Hornung, Georg Lausen, and Christoph Pinkel. SP2Bench: A SPARQL performance benchmark. In *Proc. of the 25th International Conference on Data Engineering*, pages 222–233, Shanghai, China, 2009. IEEE.

[32] The Eclipse Foundation. ATL Transformations, 2014. URL: `http://www.eclipse.org/atl/atlTransformations/`.

[33] The Eclipse Foundation. Eclipse Public License - v1.0, 2014. URL: `http://www.eclipse.org/legal/epl-v10.html`.

[34] Transaction Processing Performance Council (TPC). TPC-C Benchmark, 2010.

[35] Zoltán Ujhelyi, Gábor Bergmann, Ábel Hegedüs, Ákos Horváth, Benedek Izsó, István Ráth, Zoltán Szatmári, and Dániel Varró. EMF-IncQuery: An integrated development environment for live model queries. *Science of Computer Programming*, 2014.

[36] Marcel Van Amstel, Steven Bosems, Ivan Kurtev, and Luís Ferreira Pires. Performance in model transformations: experiments with atl and qvt. In *Theory and Practice of Model Transformations*, pages 198–212. Springer, 2011.

[37] Gergely Varro, Andy Schurr, and Daniel Varro. Benchmarking for graph transformation. In *Visual Languages and Human-Centric Computing, 2005 IEEE Symposium on*, pages 79–88. IEEE, 2005.

[38] N. Walsh and R.L. Hamilton. *DocBook 5: The Definitive Guide*. Definitive Guide Series. O'Reilly Media, 2010.