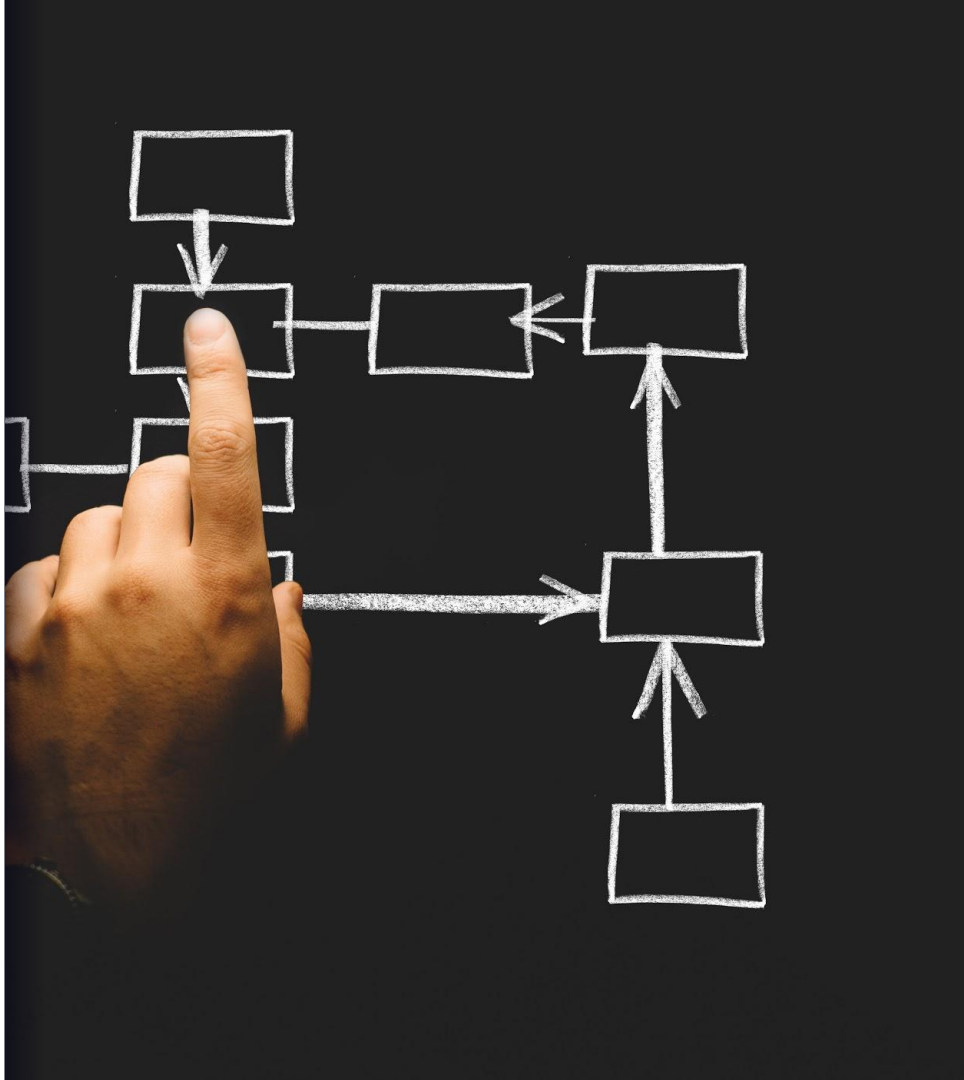
GRAN SASSO SCIENCE INSTITUTE
CENTER FOR ADVANCED STUDIES
INFN

Exploring model repositories by means of megamodel-aware search operators

F. Basciani, J. Di Rocco, D. Di Ruscio, L. Iovino, A. Pierantonio

International Workshop on Analytics and Mining of Model Repositories (AMMoRe) @ MODELS '18
16 October 2018 - Copenhagen, Denmark



Premise

MODEL REPOSITORIES

The pervasiveness of modelling techniques in everyday software practice has escalated the importance of reliable model **repositories**

FOR THIS REASON

The availability of efficient and accurate ways to retrieve artifacts is becoming of high **relevance**.

Thus, relying on sound and well-formed models for **discovering and reusing** existing artifacts is **key** to preserving productivity benefits.

*A contributory factor **limiting** the use of current search engines is the **poor** alignment between the **query** languages and the lattice of **relations** among the different and heterogeneous artifacts in the repository.*

Problem

The diversity and numerosity of models stored in a repository require query mechanisms based on a finer-grained level of understanding of the repository.

For instance, in order to locate an artifact, it might be useful to be able to predicate over repository-wide attributes:

- artifact types
- metamodels
- domain types
- maturity levels

and...

- metamodel/model elements (internally)

CONTRIBUTION

We present a novel approach to model search that leverages the repository structure into a **megamodel**

USABILITY

The approach provides designers with dedicated operators to explore the model repository **without requiring the knowledge** of low-level details about the underlying platforms to formulate the queries.

Outline

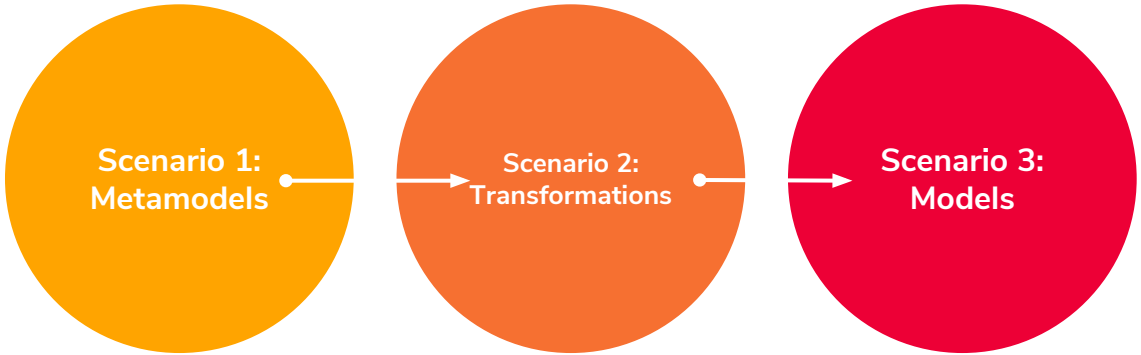
Structure of the presentation

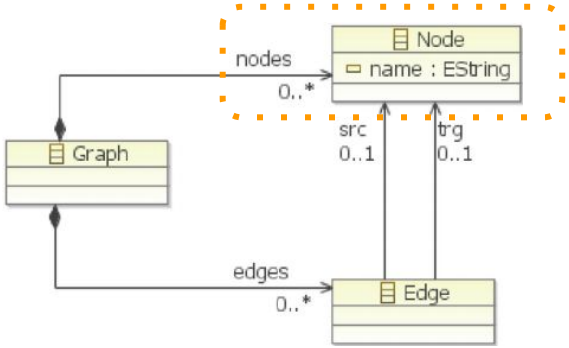
1. Motivating scenarios
2. Requirements
3. Presented Approach
4. Running Example
5. Conclusions

Motivating Scenario

We discuss exploratory scenarios with the goal:

- highlighting **the need** for proper methods and tools supporting the exploration of model repositories managing different kinds of interrelated modelling artifacts;
- showing that even the implementation of **simple search** scenarios can be **error-prone and time-consuming** if not adequately supported.





1.

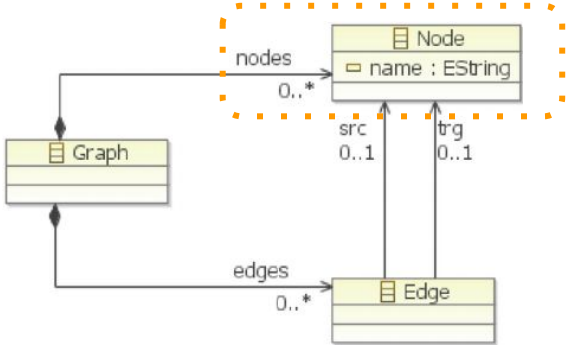
Metamodels

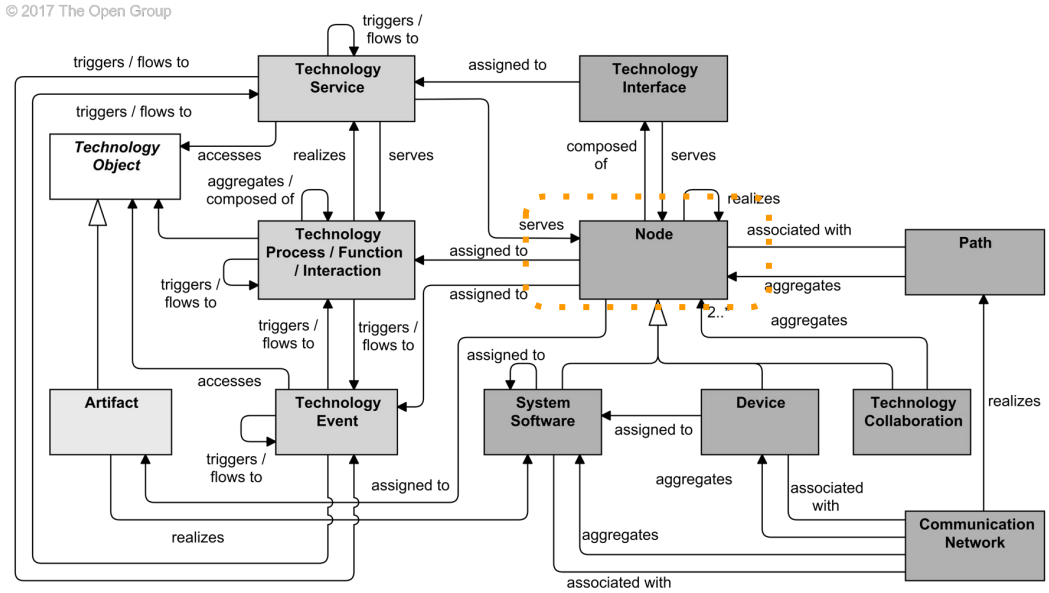
*Find the metamodels that contain a specific **metaclass** defined in terms of its **name***

1.

Metamodels

Find the metamodels that contain a specific metaclass defined in terms of its ***name***





1.

Metamodels

*Find the metamodels that contain a specific metaclass defined in terms of its **name** and structural **features***

```
1 public  List<File> search(String folderString, String className
      , String attrName, String refName) {
2  File folder = new File(folderString);
3  List<File> results = new ArrayList<Artifact>();
4  String query = "EClass.allInstances()->exists(e | e.name='"+
       className+"') and EAttribute.allInstances()->exists(e | 
       e.name='"+attrName+"') and EReference.allInstances()->
       exists(e | e.name='"+refName+"')"
5  for (final File fileName : folder.listFiles()) {
6   if (getFileExtension(fileName).equals("ecore")) {
7    List<EPackage> epList = getEPackages(filename)
8    for (EPackages package : epList)
9     if(checkConstraint(package, query))
10     results.add(fileName);
11  }
12 }
13 return result,
14}
```

2.

Transformations

*Find transformations able to generate specific elements **out** of source models of a given type*

```
1 List<File> search(String folderString, String outPatternName,
      String inPatternName) {
2  File folder = new File(folderString);
3  List<File> result = new ArrayList<File>();
4  String query = "SimpleOutPatternElement.allInstances()->exists
       (e␣|␣e.type.name␣=␣'" + outPatternName+"')␣and␣
       SimpleInPatternElement.allInstances()->exists(e␣|␣e.type
       .name␣=␣'" + inPatternName+"')"
5  for (final File fileName : folder.listFiles())
6   if(getFileExtension(fileName).equals("atl")){
7    ATLModel atlModel = injectTrasformation(fileName.getName());
8    if (checkConstraint(atlModel.getRoot(), query))
9     result.add(fileName);
10  }
11 }
```

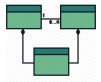
3.

Models

Find models conforming to a specific metamodel

```
1 public  List<File> search(String folderString, String nsUri) {
2  File folder = new File(folderString);
3  List<File> results = new ArrayList<Artifact>();
4
5  for (final File fileName : folder.listFiles()) {
6   List<EPackage> epList = getNsURiFromModels(filename);
7   if(containsUri(epList, nsUri))
8    results.add(fileName);
9  }
10 return result,
11}
12f.eClass().eResource().getURI()
```

Requirements



Supported modeling artifact

It refers to the kinds of artifacts that the considered approach is able to manage

Indexing supported

In order to make searches more efficient, approaches may rely on indexing mechanisms

Query mechanism

It refers to how query are specified; e.g., there are approaches that allow users to specify queries with query strings; others adopt more structured languages like OCL

Megamodel-awareness

Some approaches consider also relations among different kinds of artifacts, where relations give place to joins that traverse the repository;

Considering the artifact relations requires the approach to be aware of the repository structure typically represented by means of megamodels

“

*Only few approaches leverage the **relationships** among the artifacts as first-class entities to be used when exploring an existing model repository.*

“

*Most of the existing approaches are based on indexing techniques for the sake of better **performance**.*

*Some approaches have recognized the importance of being **generic** and supporting the management of **any kind** of modeling artifacts.*

Inspirational Examples

Very simple requirements may lead to complex queries embedded in components that involve several languages and tools

Gmail

provides users with a list of domain-specific operators that can be used when searching throughout mails

Operators

Multiple operators can be combined in complex search string. For instance:

has: attachment

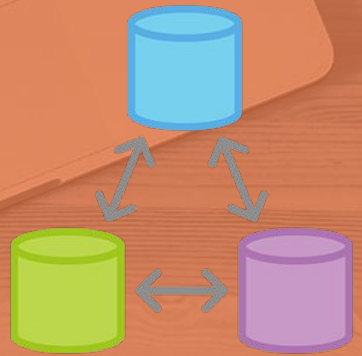
To: David has: youtube

Inspirational Examples

The interesting idea about this is that it is based on a very simple syntax that does scale well

- It is **platform agnostic** as it does not require specific expertise to be used
- The notation remains **concise** despite searches can get complex

Thus, the same mechanism has been adopted for searching through model repositories



Example

Let us assume we are interested in retrieving all transformations that consumes models conforming to a ***Family*** *metamodel*

The expression fromMM:Family returns them.

Now, if we restrict our attention to only those transformations that return Person models, we could write...

fromMM:Family and toMM:Person

Available Operators in our approach

An excerpt of the available operators together with descriptions and typing requirements*

Operator name	Artifact	Description
`name:`	Any	It returns all the artifacts matching the name provided by the query tag value
`author:`	Any	It returns all the artifacts provided by a specific author
`conformToMM:`	Metamodel	It returns the models that conform to the metamodel named as the provided value
`eClass:`	Metamodel	It returns the metamodels containing at least one metaclass named as the provided value
`eAttribute:`	Metamodel	It returns the metamodels containing at least one metaclass having an attribute named as the provided value
`eReference:`	Metamodel	It returns the metamodels containing at least one metaclass having a reference named as the provided value
`fromMM:` / `toMM:`	Transformation	It returns all the transformations having as source metamodel the provided value for the fromMM tag and as destination the provided value for the toMM tag
`fromMC:` / `toMC:`	Transformation	It returns all the transformations having a rule transforming the metaclass specified in the value for the first tag into the metaclass specified as value for the last tag

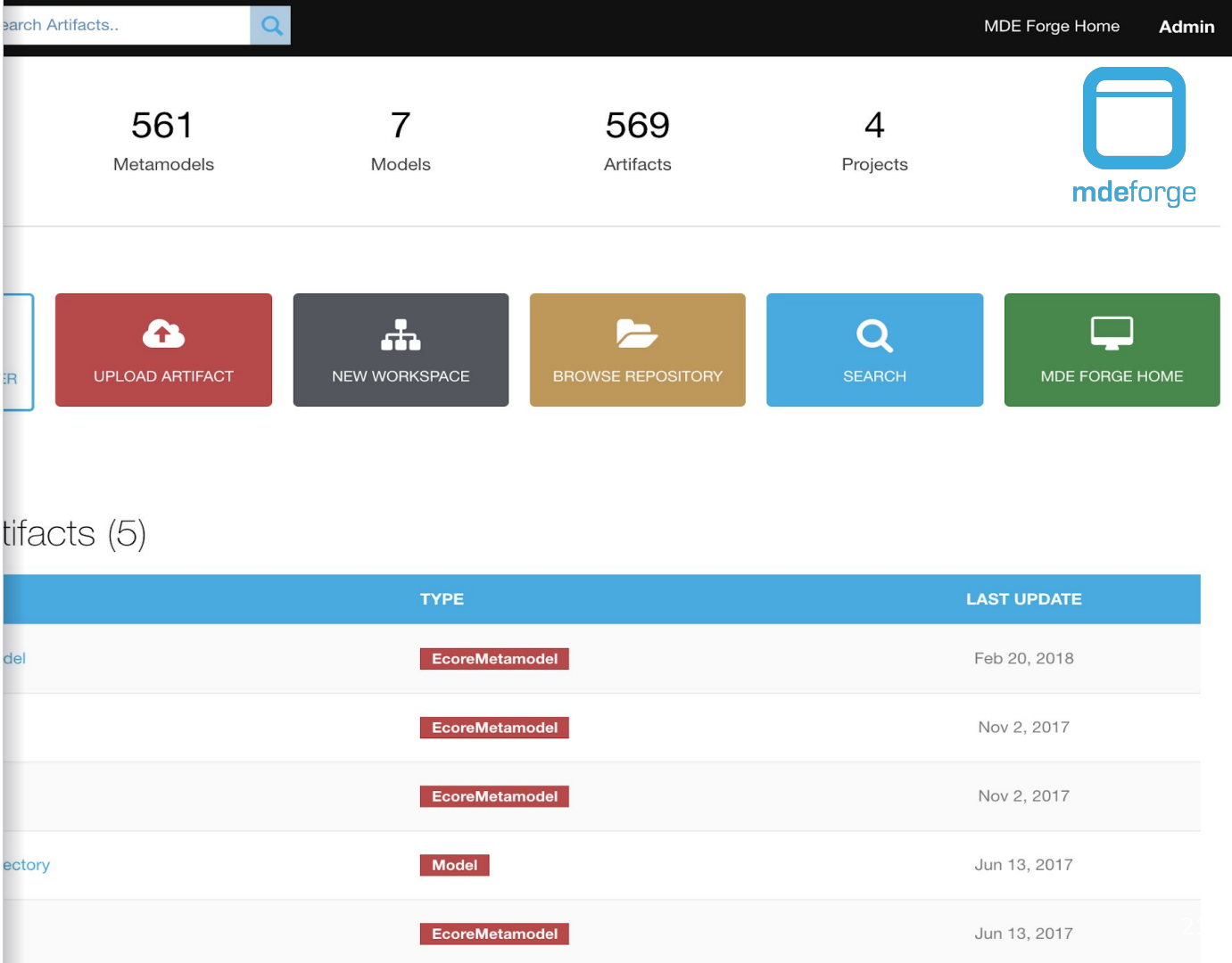
**Operators are still under development*

Overview of the technology baseline

The presented framework is developed on top of the MDEForge Platform.

MDEForge is an extensible modeling framework that consists of services for storing, managing, analysing any kind of modeling artefacts

http://mdeforge.org



Lucene

Lucene is a simple but powerful Java-based and open source search library.

The library provides the core operations, which are typically required by any search application.

The main **operations** the engine provides can be summarized as:

- collecting the content
- analyzing the artifacts
- indexing the documents
- providing a search interface
- query building, query execution
- showing results.

Lucene™ Features

Lucene offers powerful features through a simple API:

Scalable, High-Performance Indexing

- over 150GB/hour on modern hardware
- small RAM requirements -- only 1MB heap
- incremental indexing as fast as batch indexing
- index size roughly 20–30% the size of text indexed

Powerful, Accurate and Efficient Search Algorithms

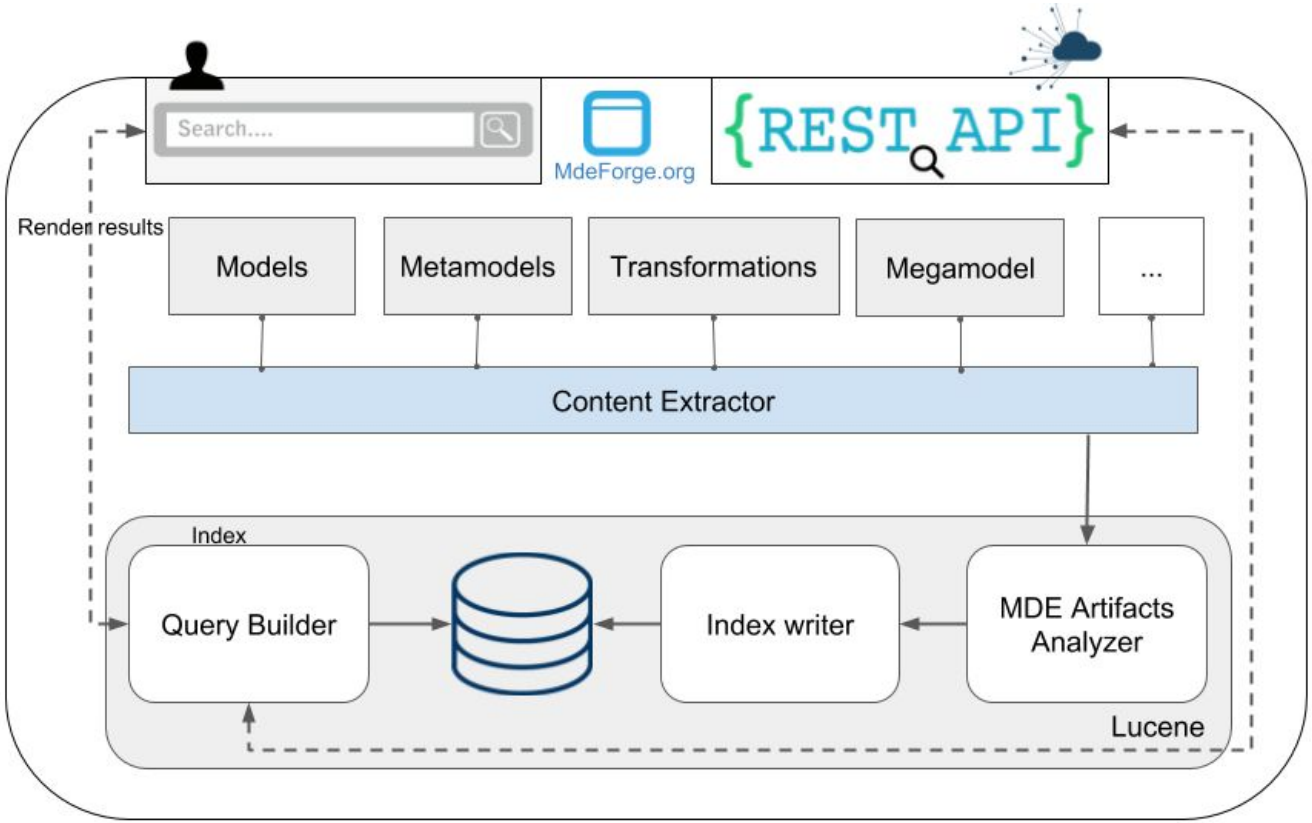
- ranked searching -- best results returned first
- many powerful query types: phrase queries, wildcard queries, proximity queries, range queries and more
- fielded searching (e.g. title, author, contents)
- sorting by any field
- multiple-index searching with merged results
- allows simultaneous update and searching
- flexible faceting, highlighting, joins and result grouping
- fast, memory-efficient and typo-tolerant suggesters
- pluggable ranking models, including the Vector Space Model and Okapi BM25
- configurable storage engine (codecs)

Cross-Platform Solution

- Available as Open Source software under the Apache License which lets you use Lucene in both commercial and Open Source programs
- 100%-pure Java
- Implementations in other programming languages available that are index-compatible

Artifact search infrastructure

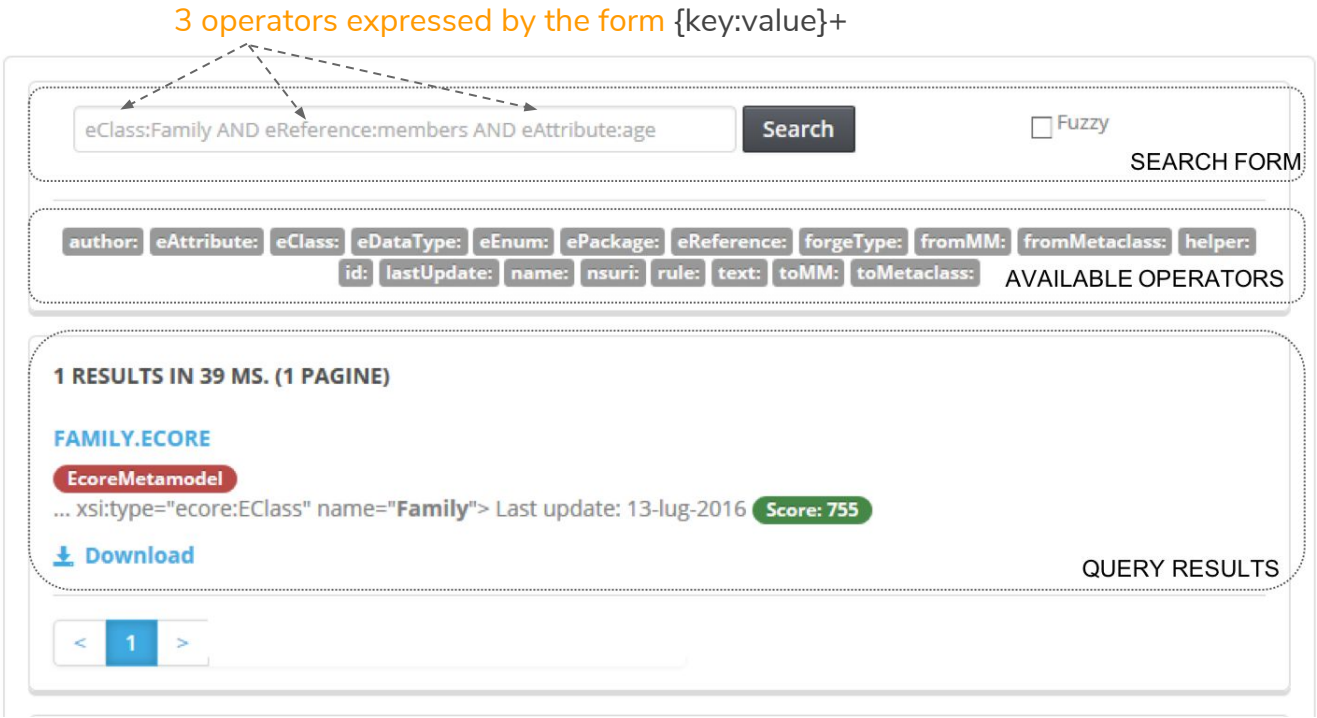
The integration of MDEForge and Lucene



Artifact search web interface

Web-based search page, consists of three main parts:

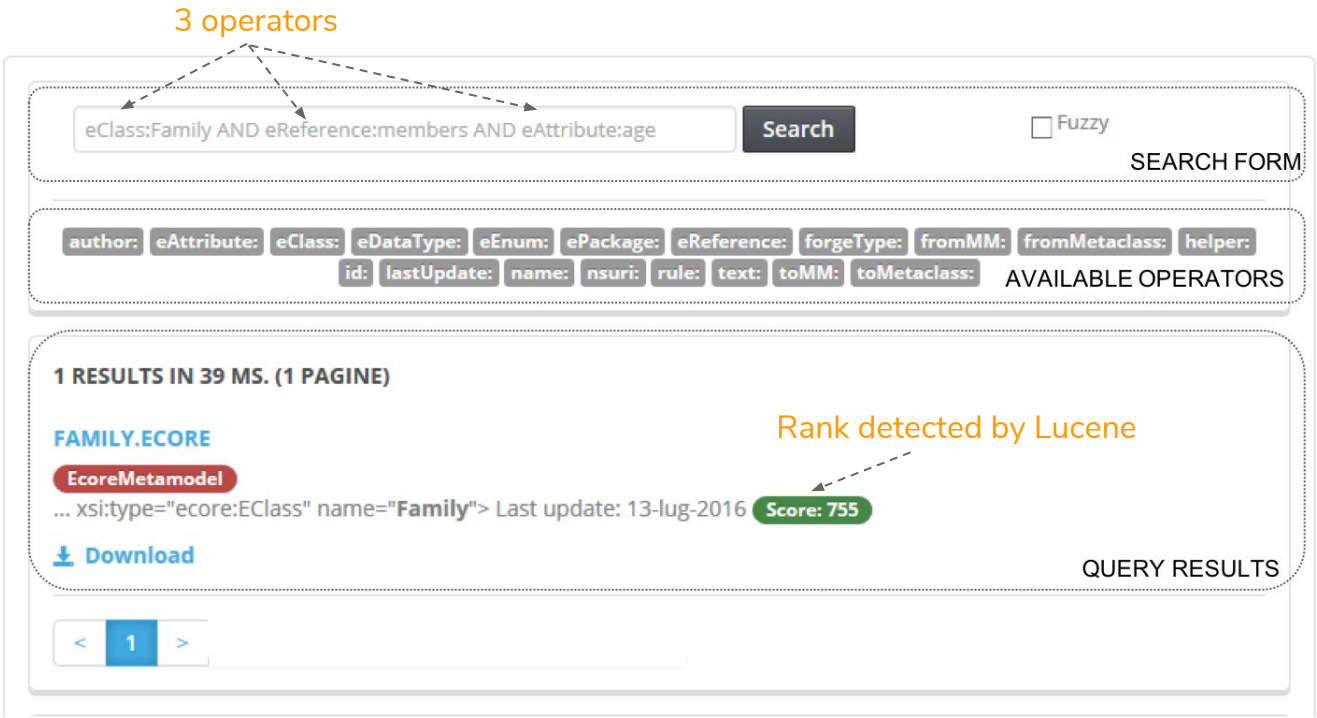
1. the search **form**
2. the list of available search **operators** that can be used for specifying the query
3. and the query **results**.



Artifact search web interface

Web-based search page, consists of three main parts:

1. the search **form**
2. the list of available search **operators** that can be used for specifying the query
3. and the query **results**.



Experiments

Q1

Get all the **metamodels** having a metaclass named *className*, which in turn contains an attribute named *attrName*, and a reference named *refName*

Q2

Get all the model **transformations** transforming metaclasses named *metaclassName1* to generate target *metaclassName2* elements

Q3

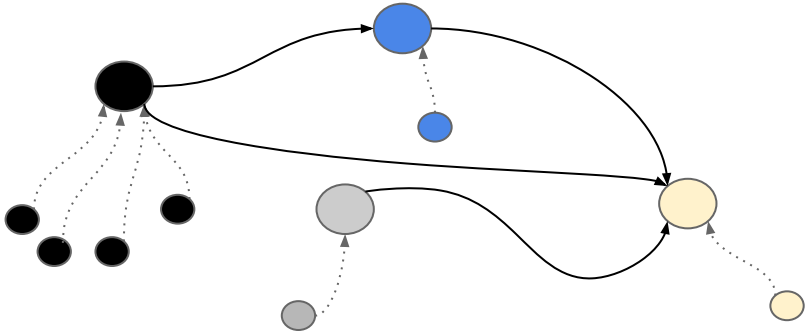
Get all the **models** conforming to the metamodel named *metamodelName*

The queries are written with the proposed approach and in OCL in order to provide a comparison

Experiments

Q1

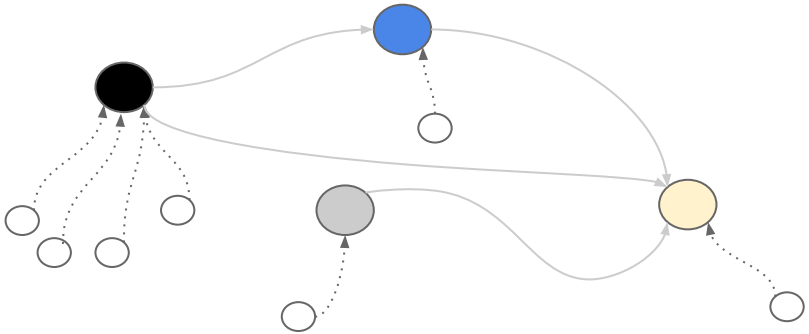
Get all the **metamodels** having a metaclass named *className*, which in turn contains an attribute named *attrName*, and a reference named *refName*



Experiments

Q1

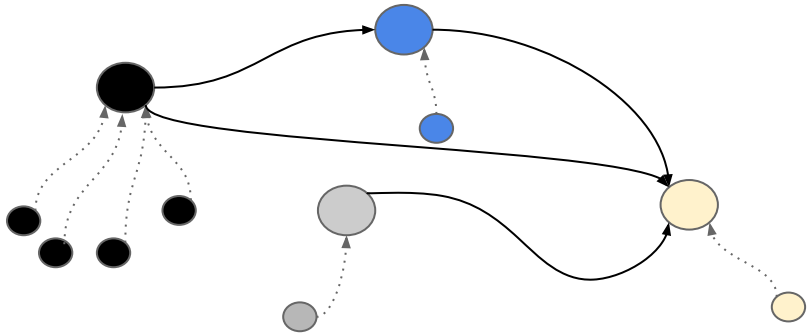
Get all the **metamodels** having a metaclass named *className*, which in turn contains an attribute named *attrName*, and a reference named *refName*



Experiments

Q2

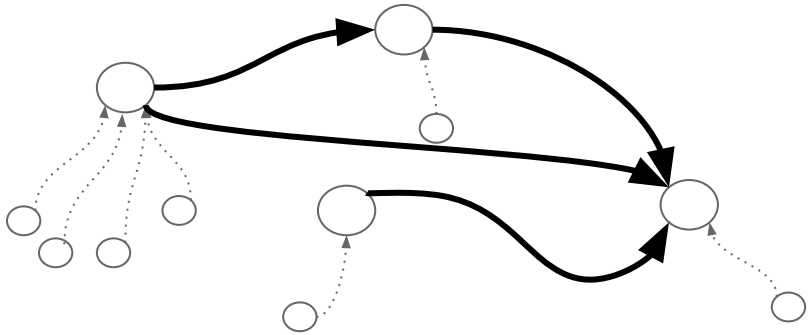
Get all the model **transformations** transforming metaclasses named *metaclassName1* to generate target *metaclassName2* elements



Experiments

Q2

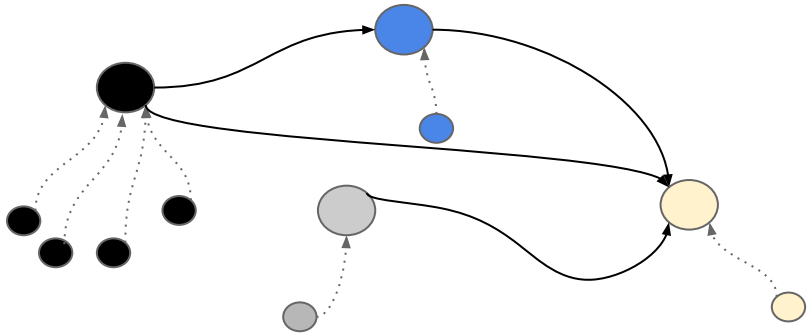
Get all the model **transformations** transforming metaclasses named *metaclassName1* to generate target *metaclassName2* elements



Experiments

Q3

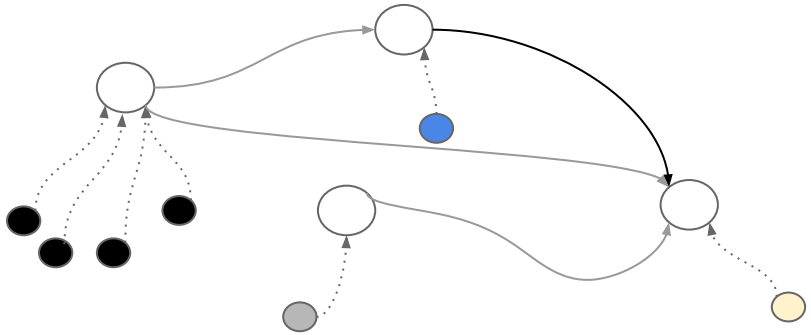
Get all the **models** conforming to the metamodel named *metamodelName*



Experiments

Q3

Get all the **models** conforming to the metamodel named *metamodelName*



Experiments

Q1

Get all the **metamodels** having a metaclass named *className*, which in turn contains an attribute named *attrName*, and a reference named *refName*

Q2

Get all the model **transformations** transforming metaclasses named *metaclassName1* to generate target *metaclassName2* elements

Q3

Get all the **models** conforming to the metamodel named *metamodelName*

	Proposed approach		OCL based approach	
	exec.time (ms)	query string	exec.time (ms)	#loc
Q1	39	`eClass:`*`className`* `AND eAttribute:`*`attrName`* `AND eReference:`*`refName`*	12641	≈70
Q2	59	`fromMetaclass:`*`metaclassName1`* `AND toMetaclass:`*`metaclassName2`*	7701	≈40
Q3	24	`conformToMM:`*`metamodelName`*	8102	≈60

Highlights

It emerges that the execution time of the OCL-based queries is always considerably higher than that of the proposed approach

Indexes are typically created off-line by means of batch processes, which do not interfere with the actual execution of queries

Each time the indexes must be updated because new artifacts are added, this is done incrementally

Conclusion and Future Work

Modern modelling tools are becoming more and more distributed platforms where artifacts can be persistently stored and coherently dealt with.

Being able to conveniently search throughout the repository according to specific criteria is key to any reuse practice

The presented approach proposes simple yet powerful operators to perform complex repository searches

The corresponding search infrastructure permits modelers to explore model repositories in an efficient way by abstracting from the specific platforms and tools used to formulate the queries

More accurate comparison criteria and metrics have to be properly defined.

The adoption of alternative frameworks for model query, like Hawk will be also investigated

Thanks for the attention!

.

.

.

.

Question?

Contact:

- ludovico.iovino@gssi.it
- http://cs.gssi.infn.it/people/postdoc/iovino