Integration of Hawk for Model Metrics in the MEASURE Platform

Orjuwan Al-Wadeai[1], Antonio Garcia-Dominguez[1], Alessandra Bagnato[2], Antonin Abherve[2] and Konstantinos Barmpis[3]

[1]*SARI, School of Engineering and Applied Science, Aston University, Birmingham, U.K.*

[2]*Softeam, Research and Development Department, Paris, France*

[3]*Department of Computer Science, University of York, York, U.K.*

Keywords: Specialized Information Retrieval, Unified Modeling Language (UML), Model Repositories, Big Data, Scalable Model Querying.

Abstract: The MEASURE project aims to integrate metrics across all phases of the software development lifecycle into a single decision support platform. For the earlier phases, metrics can be derived from models. Industrial use of model-driven engineering produces large model repositories, and high-performance querying is key to keep their metrics up to date. This paper presents an integration between the MEASURE metrics platform and the Hawk model indexing tool. Hawk was improved in several ways, such as adding support for the new Modelio metamodelling framework, or allowing Hawk servers to be provisioned through configuration files rather than through its web services. MEASURE and Hawk were then combined successfully to extract metrics from Modelio models of various domains, and Hawk was able to index and efficiently answer queries about the 2GB collection of models used by Softeam to develop Modelio.

1 INTRODUCTION

Growing global competition and system complexity in the software industry means that companies need to meet ever increasing demands without compromising on quality and delivery times (Shields, 2014). Many companies have taken to automation and placed a strong emphasis on nimble iteration in their processes to address these challenges. However, this automation and iteration need to be guided by up-to-date and accurate metrics, and traditional approaches are not up to the task.

The MEASURE ITEA3 consortium (Softeam R&D, 2017) aims to cover this gap by developing a comprehensive set of tools for automated and continuous measurement over all stages of the software development lifecycle. It includes the development of better metrics and ways to analyse the big data produced by continuous measurements, the validation of those metrics by the integration of the metrics and tools into running processes in various industrial partners, and the creation of decision support tools for project managers through the visualisation of the collected data. As shown in Figure 1, MEASURE revolves around a central data collection and analysis platform (the "MEASURE platform" from now on), a web application that integrates all other efforts.

Unlike other metric platforms, MEASURE aims to collect metrics about more than just code, as it covers the entire software lifecycle and not just those stages related to coding or testing. Particularly, one of the aims is to collect metrics about models. One of the MEASURE project partners in particular (Softeam Cadextan) is the lead developer of the commercial open-source Modelio (Softeam Cadextan, 2017) modelling tool. Softeam wanted to integrate metrics about Modelio business, requirements and design models into the MEASURE platform. This would require a technology that provided high-performance querying from potentially very large models, as those seen in the field by Modelio. For instance, it is common to have millions of elements for models reverse engineered from large code bases (e.g. for software modernisation). Distributed teams producing models concurrently can also create large collections of model elements as time goes on. Rather than develop their own from scratch, Softeam decided to reuse Hawk, a scalable model indexing and querying framework that had been successfully integrated with their Constellation product for collaborative model-

Al-Wadeai, O., Garcia-Domínguez, A., Bagnato, A., Abherve, A. and Barmpis, K.
Integration of Hawk for Model Metrics in the MEASURE Platform.
DOI: 10.5220/0006732207190730
In *Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2018)*, pages 719-730
ISBN: 978-989-758-283-7

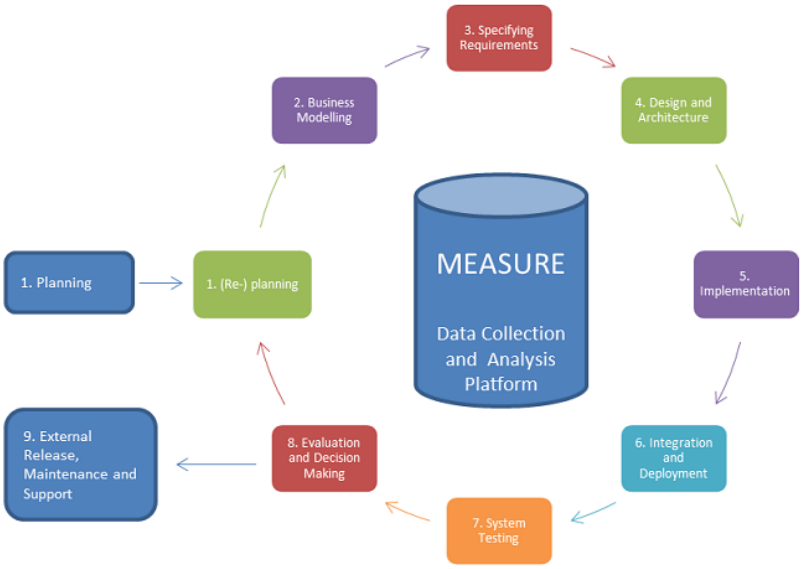


Figure 1: General approach for the MEASURE ITEA3 project.

ling (Garcia-Dominguez et al., 2016).

Hawk could not be used as-is, however. The MEASURE platform imposed new requirements for a seamless integration, and recent releases of Modelio had fundamentally changed the way the structure of the models could be described to Hawk. The overall practicality and scalability of Hawk for this domain had to be re-evaluated as well. This paper presents the work that was conducted to clarify these new requirements and re-engineer Hawk for this new application, and the evaluations that were conducted to validate Hawk as the model metrics component for the MEASURE platform.

The rest of this paper is organized as follows: Section 2 provides a background about MEASURE, Modelio and Hawk. Section 3 summarizes the integration efforts of Hawk into MEASURE. Section 4 shows our initial validation of the flexibility of this approach, followed by an evaluation of its scalability. Section 6 provides some conclusions and future lines of work.

2 BACKGROUND

This section will introduce some basic concepts required to understand the rest of the work: the MEASURE platform, the Modelio modelling environment, and the Hawk model indexing framework.

2.1 MEASURE Platform

The MEASURE platform is an open-source web application (Abherve et al., 2017) which allows collecting, calculating and visualizing data collected by executing *measures* defined according to the Structured Metrics Meta-Model (SMM) (Object Management Group, 2016). A measure is a method for assigning *measurements* (numerical or symbolic values) to entities (*measurands*). An *observation* applies a set of measures to a certain *scope*, obtaining specific measurements of the attributes of interest.

Within the MEASURE platform, this observation scope is defined through *projects* defined by the user, and *phases* that they may go through. Observations and their measurements are collected into dashboards like the one in Figure 2. Particularly, observations are known as *measure instances* in the MEASURE platform: they configure the measure itself in some way and provide a scope of what should be the measurand. Figure 3 shows the current user interface for managing measure instances.

There are two types of SMM measures: *direct* measures that are taken from a measurand through some process (e.g. lines of code), and *derived* measures whose values are computed from others (e.g. a ratio or a sum). Measures are contributed by users as Java classes written against the interfaces of the SMM Measure API library (Abherve, 2017), with some additional XML metadata.

As an example, one of these direct measures implemented in the MEASURE platform is an integration with the SonarQube code-centric continuous eva-

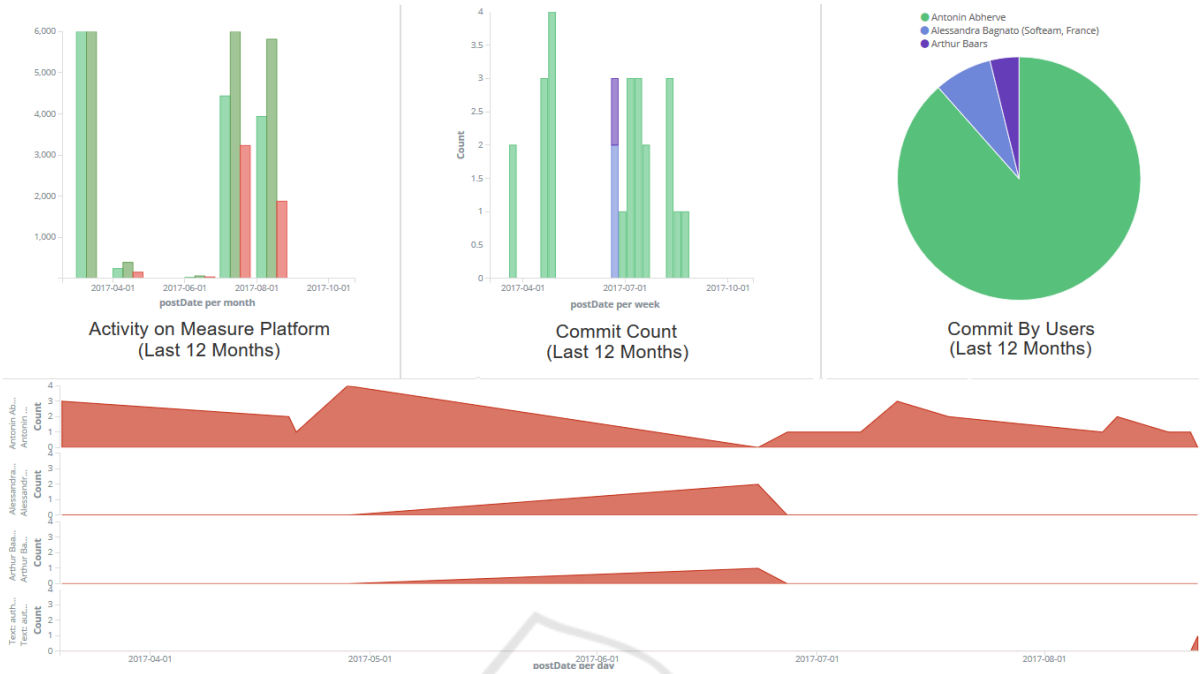
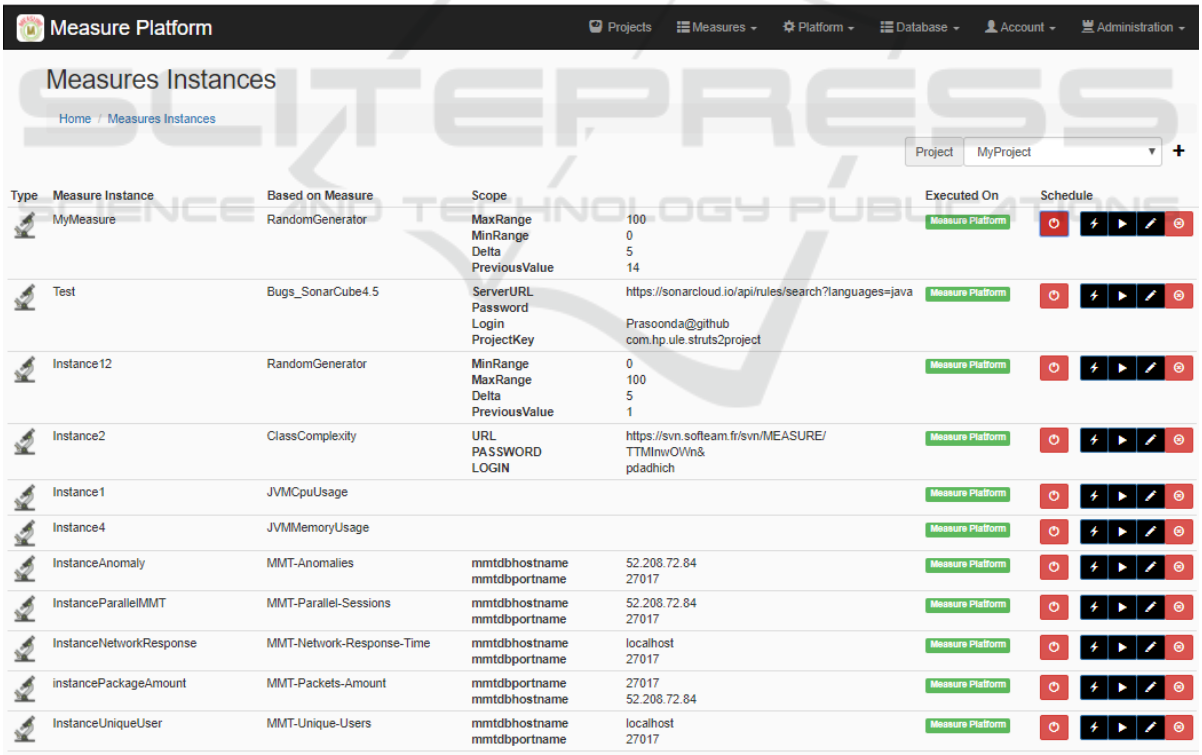


Figure 2: Screenshot of a sample dashboard in the MEASURE platform.



Type	Measure Instance	Based on Measure	Scope		Executed On	Schedule
	MyMeasure	RandomGenerator	MaxRange MinRange Delta PreviousValue	100 0 5 14	Measure Platform	
	Test	Bugs_SonarCube4.5	ServerURL Password Login ProjectKey	https://sonarcloud.io/api/rules/search?languages=java Prasonda@github com.hp.ule.struts2project	Measure Platform	
	Instance12	RandomGenerator	MinRange MaxRange Delta PreviousValue	0 100 5 1	Measure Platform	
	Instance2	ClassComplexity	URL PASSWORD LOGIN	https://svn.softeam.fr/svn/MEASURE/ TTMInwOWn& pdadhich	Measure Platform	
	Instance1	JVMCpuUsage			Measure Platform	
	Instance4	JVMMemoryUsage			Measure Platform	
	InstanceAnomaly	MMT-Anomalies	mmtdbhostname mmtdbportname	52.208.72.84 27017	Measure Platform	
	InstanceParallelMMT	MMT-Parallel-Sessions	mmtdbhostname mmtdbportname	52.208.72.84 27017	Measure Platform	
	InstanceNetworkResponse	MMT-Network-Response-Time	mmtdbhostname mmtdbportname	localhost 27017	Measure Platform	
	instancePackageAmount	MMT-Packets-Amount	mmtdbportname mmtdbhostname	27017 52.208.72.84	Measure Platform	
	InstanceUniqueUser	MMT-Unique-Users	mmtdbhostname mmtdbportname	localhost 27017	Measure Platform	

Figure 3: Screenshot of measure instances in the MEASURE platform.

luation platform[1]. The implementation of the SMM Measure API interfaces invokes the service of the SonarQube platform, feeding its information to MEASURE to be combined with other metrics not related to code.

2.2 Modelio

Modelio (Softeam Cadextan, 2017) is a commercial open-source modelling environment developed by Softeam. It supports multiple notations (UML, BPMN, and SysML among others). While Modelio is developed with the Eclipse Rich-Client platform, it is not based on the popular Eclipse Modelling Framework (EMF). Instead, it uses its own metamodelling infrastructure and file format (EXML). The standard OMG XMI format is only available as an export format. More details on the EXML format are available from our prior work in integrating Hawk and Modelio (Garcia-Dominguez et al., 2016).

One important detail is that while it is open-source, the exact license changes depending on the component. The core runtime is under the Apache License, but most of the other components are under the GNU General Public License version 3 (GPLv3). The incompatibility of the GPLv3 license with other popular open licenses was already an issue in our prior work: we will need to refer back to this later on.

Other distinguishing features for Modelio are the availability of a full-fledged scripting environment for various modeling tasks (e.g. model transformation or code generation), and the strong support for reverse engineering of models from existing codebases. Scripting has been useful to test its scalability, as it is quite convenient for generating large synthetic models.

2.3 Hawk

As mentioned before, Hawk was selected to provide model metrics for MEASURE as it had been successfully integrated with other Modelio-based products from Softeam in the past, delivering the desired functionality and performance for the Constellation collaboration tool. This section will provide a high-level description of some of the details behind Hawk.

Hawk (Barmpis and Kolovos, 2013) is a heterogeneous model indexing framework that keeps track of collections of file-based models and maintains a NoSQL model-element-level graph database with their latest versions, in order to provide efficient and scalable model querying. Hawk is distributed as open-source software under the Eclipse Public License (EPL) 1.0.

[1]https://github.com/ITEA3-Measure/Measures

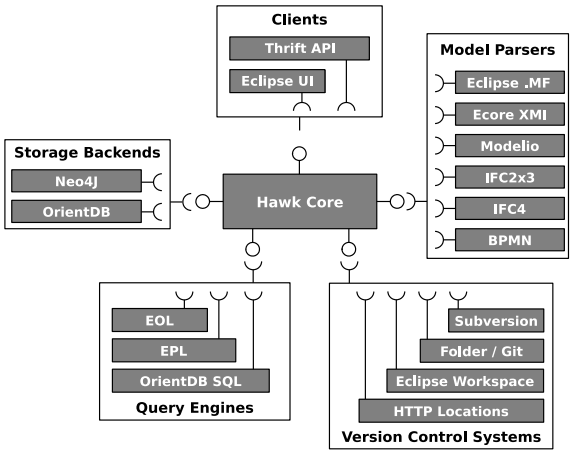


Figure 4: Component-based architecture of the Hawk model indexing framework.

2.3.1 Architecture

As shown in Figure 4, Hawk follows a component-based architecture. These are the most important component types:

Model Parsers: understand various file formats for storing models and metamodels (descriptions of the structure of a set of models). The Ecore XMI parser understands models created by tools based on the Eclipse Modelling Framework, for instance. There was also a model parser component for Modelio 3.4, which was developed before this paper (Garcia-Dominguez et al., 2016).

Backends: integrate Hawk with different database technologies. At the moment, two backends are supported: OrientDB and Neo4j. OrientDB is preferred by Softeam, since its Apache licensing is compatible with Hawk's EPL license without requiring separate arrangements.

Version Control System Connectors: allow Hawk to read models in various types of locations. At the moment, Hawk supports local folders, Subversion or Git VCS, arbitrary HTTP locations, and Eclipse workspaces.

Query Engines: answer questions written in certain query languages for any type of model indexed by Hawk. Three languages are available currently: the Epsilon Object Language and Epsilon Pattern Languages (Kolovos et al., 2017) can be used with the Neo4j or OrientDB backends, whereas the OrientDB SQL-like dialect is only available for that backend.

2.3.2 Using Hawk

Hawk can be used as a library, as a set of plugins for the Eclipse IDE, or as a network service. Regardless of the method, using Hawk generally consists of these steps:

1. Creating an *index* with a particular backend in a certain directory.
2. Registering the relevant metamodels describing the structure of the models we will index.
3. Registering the locations where our models are stored (e.g. folders or VCS).
4. Optionally, defining *indexed* attributes and *derived* attributes for faster searching. Indexed attributes make it possible to quickly find model elements by the value of their attributes. Derived attributes extend certain model element types with new properties derived from their regular attributes, and can be used for fast searching and performing expensive computations in advance.
5. Waiting for Hawk to index all files mentioned, and then querying through one of the available engines. Hawk will detect changes in the model files and trigger updates as needed.

As a standalone server, Hawk exposes its capabilities through a set of HTTP(S) web services on top of the Apache Thrift (Apache Software Foundation, 2017) messaging library. Thrift makes it possible to support multiple messaging formats (*protocols* in Thrift terminology) in the same API, which cover different tradeoffs between language compatibility (JSON is the most compatible) and performance (the "tuple" binary format produces the smallest messages).

3 INTEGRATION OF HAWK INTO MEASURE

The previous section presented the MEASURE platform, the Modelio modelling tool and the Hawk model indexing framework separately. This section will discuss the new requirements that arose from the need to use Hawk for model metrics in MEASURE, and how these were met.

3.1 New Requirements

Softeam had prior experience with Hawk from the MONDO EU project in scalable modelling (Kolovos et al., 2016), and its high-performance and flexibility made it the first choice for MEASURE. Since the MONDO project ended in 2016, however, Modelio had continued to evolve, and MEASURE presented new challenges. Further talks between Softeam and the Hawk developers extracted these requirements:

R1. Since Modelio 3.6, metamodels were no longer embedded in the source code of Modelio, but instead provided by *metamodel descriptor files*. Hawk needed to be able to understand these.

R2. Hawk would need to run as a standalone service from the MEASURE platform, unlike in the Constellation integration where it was used as a library. This was to keep the MEASURE platform simple and to separate the high resource demands of Hawk from it.

R3. Softeam needed to be able to automatically provision new Hawk servers in a cloud environment with standard tools (e.g. Docker, Puppet or Chef). These tools usually operate by filling in configuration file templates, rather than invoking web services.

R4. Model metrics would still be written as queries in the Epsilon Object Language, much like those in the Constellation work. Softeam had found this language simple enough to use during prior experiments in the MONDO project.

R5. Hawk would need a component that could be deployed as a measure in the MEASURE platform, reading the SMM-based configuration to invoke Hawk and relay the results back to the platform.

These requirements meant that while most of Hawk could be reused (R2 and R4 could leverage existing components), it was necessary to rewrite some components (R1), expand others with new features (R3) and create a new one (R5). The integrated architecture was envisioned as in Figure 5, where the MEASURE platform would live in one machine, the Hawk server in another machine, and the modeller's Modelio installation in a third machine (the workstation). The yellow components would be the new pieces in the puzzle: a version of Hawk with a revised Modelio model parser component, a file-based Hawk server configuration engine, and a measure implementation based on Hawk queries for the MEASURE platform. The following sections will expand on the work involved for each of these.

3.2 Metamodel Descriptor Support

Before Modelio 3.6.0, the Modelio metamodels were embedded in the source code of the tool and were not customisable by users. Thanks to this, it was possible to adopt a simple approach in which Hawk was compiled against an Apache-licensed library produced by

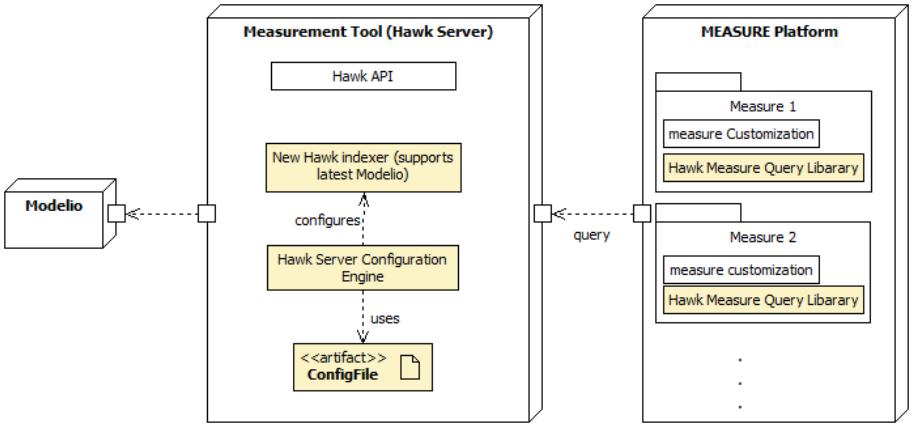


Figure 5: UML deployment diagram of the integration between Hawk and the MEASURE platform.

Listing 1: Sample contents of a `mmversion.dat` file

```
modelio.kernel          1
0.1.00                  2
Standard                3
1.0.00                  4
```

Softeam for each release of Modelio. Unfortunately, this meant that a particular installation of Hawk could not index models developed with multiple versions of Modelio, and that supporting a different version required replacing `.jar` files manually.

Modelio 3.6.0 made that approach unfeasible: users could now define their own metamodels through *metamodel descriptor files*. We could not expect metamodels to be stable for a release: instead, Hawk had to understand those descriptions and then use them to understand models conforming to this structure.

Luckily, this is a very similar approach to what was done for EMF-based models, whose structure is described through ECore metamodels. Since it was not possible to bring any GPLv3 code from Modelio into Hawk, it was decided to implement a metamodel parser following a clean-room approach: instead of looking at the code, the format would be reverse engineered from a sample, with some answers from the original developers. This is the same approach that was taken when implementing the original EXML parser in Hawk (Garcia-Dominguez et al., 2016).

Now that there could be multiple Modelio metamodels registered at a time in Hawk, it was necessary to create a centralised registry of metamodels. Metamodels can be looked up by the name and a version. Unlike EMF metamodels, where identifiers tend to include the versions themselves, Modelio metamodels keep the version separate. This provides a small degree of flexibility: if a model conforms to a version of a metamodel we don't have an exact match for, we use the latest version available.

Finding out which version of a metamodel to use for a specific model file requires some thought as well. This information is present in a `mmversion.dat` file in the project within the user's Modelio workspace, and not as part of the model file or the model element themselves. Listing 1 shows the typical contents of these files, formed by a sequence of line pairs with the name of the metamodel (lines 1 and 3) and the version of the metamodel (lines 2 and 4).

3.3 Automated Server Provision

In order to use Hawk effectively in cloud environments, it should be possible to quickly bring up an index for a certain location in an automated manner. Most cloud deployment tools expect to be able to clone an image, make some small changes to the files and rerun things. Being able to configure Hawk through files is also much more familiar to server administrators, and it is easier to teach with typical copy-and-paste instructions.

To accommodate the scenarios, the Hawk server was modified so it would look upon startup for configuration files in a predefined folder, and set up new indices for those it does not have a match for. It will not make any changes to indices for which we do not have configuration files (since they may have been created manually through the API), and for now, it will not update the configuration of existing indices either.

An example of a configuration file is shown in Listing 2. Lines 1–2 mention that the OrientDB backend should be used, and provide a name for the instance. Line 3 mentions the minimum and maximum delays in milliseconds for repository polling (0 means it is disabled). Lines 4–8 specify which components should be enabled. Lines 9–12 register the metamodels des-

Listing 2: Sample contents of a server config file.

```
1  <hawk backend="...OrientDatabase"
2        name="instance_36">
3    <delay max="0" min="0"/>
4    <plugins>
5      <plugin
6        name="...ModelioGraphChangeListener"/>...
7    </plugins>
8    <metamodels>
9      <metamodel uri="" location=".../mm36.xml"/>
10   </metamodels>
11   <derivedAttributes>...</derivedAttributes>
12   <indexedAttributes>
13     <indexedAttribute attributeName="Name"
14       metamodelUri="modelio://..."
15       typeName="Class"/>
16   </indexedAttributes>
17   <repositories>
18     <repository frozen="false"
19       location="file:///.../ArchiChocolate/"
20       pass="" type="...LocalFolder" user=""/>
21   </repositories>
22 </hawk>
```

cribing the structure of the models to be indexed. Line 13 would be used to registered derived attributes, and lines 14-18 register an indexed attribute for quickly finding classes by name. Finally, lines 19-23 mention where the models are stored.

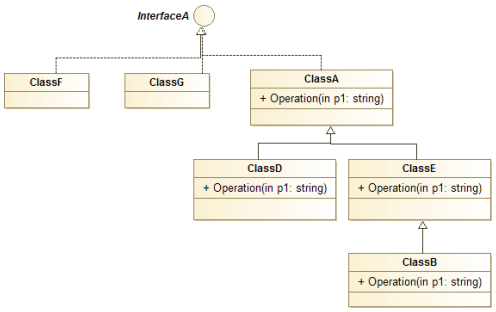
3.4 Hawk-based Measures

In order to create direct measures on models using Hawk for the MEASURE platform, a new implementation of the SMM Measure API (§2.1) interfaces was developed: the Hawk Query measure library (HawkM from now on) (Al-wadeai, 2017).

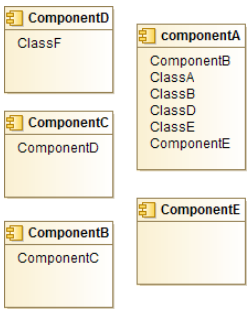
HawkM can be used in two ways:

- As a regular component for MEASURE, which takes in a configuration file with the server details and the EOL query to be run, and relays back to the MEASURE platform.
- As a library for developing more advanced metrics, which require orchestrating multiple EOL queries or automatically generating the EOL source altogether.

HawkM has provisions for the fact that the Epsilon Object Language is dynamically typed, and therefore it is not clearly known what will be the return type of the query in the configuration file. The Hawk server will provide a response with the EOL native type and the raw value, and HawkM will map it into the MEASURE platform API's type system. HawkM will also timestamp the moment when the measurement was taken.



(a) Relations between interfaces and classes.



(b) Division of classes across components.

Figure 6: UML class diagrams for the example Modelio project used to develop MEASURE queries.

4 EVALUATION

The work in the previous section allowed Hawk to become another piece of the MEASURE platform. This section will study whether this integration can implement the model measures required by the MEASURE ITEA3 project, and whether it can scale to models in the large sizes expected by its industrial case studies.

4.1 Flexibility

After completing the work in Section 3, several case studies were run to validate if Hawk could accommodate the wider variety of models in Modelio 3.6, and run the desired queries.

4.1.1 MEASURE Queries

The first test was done by creating a sample Modelio 3.6 project with various containment and inheritance/implementation relationships (as shown in Figure 6). The Hawk server was configured to index this sample project, using the OrientDB backend and the Modelio-specific model and metamodel parsing components. The Modelio 3.6 metamodel descriptor (part of the open source source code release) was parsed successfully by Hawk: this sample project requi-

Listing 3: Excerpt of the EOL source code for the "Number of overridden methods" query.

```
1  var nOverriden = 0;
2  for (myClass in Class.all) {
3    nOverriden += getNOverridden(myClass);
4  }
5  return nOverridden;
6
7  operation getNOverriden(myClass) {
8    ...
9    var ops = getAllOpsOfSubTypesOf(myClass);
10   opsNames.addAll(ops.Name);
11   for (myop in myClass.OwnedOp) {
12     if (opsNames.includes(myop.Name)) {
13       var sameName
14         = ops.select(t|t.Name = myop.Name);
15       if (sameName.size > 0) {
16         var sameNameReturn = /* ... */;
17         var sameNameReturnArgs = /* ... */;
18         if (sameNameReturnArgs.size > 0) {
19           nOverridden = nOverridden + 1;
20   }}}}
21   return nOverridden;
22 }
```

red the "Infrastructure" 2.0.00, "Standard" 2.0.00 and "modelio.kernel" 1.0.00 fragments in particular.

After indexing these models, a subset of the metrics requested by the MEASURE industrial partners was selected for implementation (as shown in Table 1). These metrics were defined and prioritised along with their required metadata and measurement tools. The goal was to have basic metrics that could be recombined into higher-level indicators. The table shows examples for the two ways of implementing queries:

- Deploying the generic measure implemented in Section 3.4 directly, specifying an EOL expression in the "query" part of its configuration. This does not require any Java coding, but it does require knowing the structure of the models to be queried quite well. This is good for advanced users or very specialised queries. The first two queries were like this.
- Writing Java code on top of the generic measure, which provides the EOL query to be run. This can be useful to distribute "canned" queries about known metrics for known metamodels to users that may not know how to write EOL code. The other queries were written in this way. The last two queries in particular were written to generate EOL queries on the fly for a specific component, whose name was given during deployment.

As an example of how a query looks like in EOL, Listing 3 shows an excerpt of the "Number of overridden methods" query. EOL is a very flexible language, and it is possible to write complex queries that check if a method has been overridden within the proper subtypes of a class. Particularly, here we check if across the subtypes, there is at least one operation with the same name, return type and argument types.

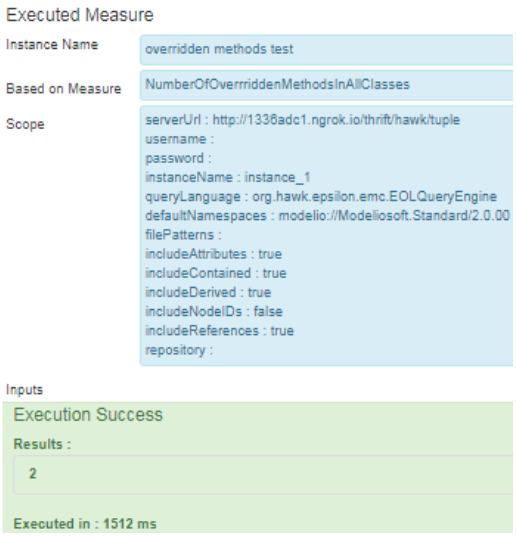


Figure 7: Query report from MEASURE platform.

Listing 4: Example query on Archimate models for retrieving the top 5 most related concepts.

```
return Concept.all.collect(c | Sequence{                    1
    c, c.closure(c2|c2.relatedTo.to).size                   2
  }).sortBy(c | -c.second).collect(c | Sequence{            3
    c.first.getTypeName() + ' ' + c.first.Name,             4
    c.second}).subList(0,5);                                5
```

The measure containing this query as part of its Java code was then deployed within the MEASURE platform, producing a report upon execution as shown in Figure 7. The execution time in this case was slightly higher than one would expect, but mostly it was due to the fact that MEASURE was on temporary infrastructure, while querying a server running off a laptop on a different country altogether. As we will see later, queries run faster given the right infrastructure.

4.1.2 Archimate Models

The second case study used to evaluate the flexibility of this integration was a set of sample models developed with the new support in Modelio 3.6 for creating enterprise architecture models written in the Archimate notation and metamodel. Specifically, the model shown in Figure 8 was indexed. This type of model combines software components (ERP, SCADA) with buildings (factories) and descriptions of the processes within the enterprise and the various business actors.

This model was indexed by Hawk successfully using the same Modelio 3.6 metamodel descriptor

Table 1: Listing of queries implemented on Hawk from MEASURE industrial partners.

Implementation	Parameters	Metric name	Test result
Generic measure with custom EOL query in config.	serverUrl instanceName queryLanguage query	Number of interfaces	1
		Number of attributes in a component	1
Custom measure with EOL query generated from Java code	serverUrl instanceName	Average classes per component	2
		Average subcomponents per component	1
		Number of overridden methods	2
		Number of overriding methods	3
	serverUrl instanceName componentName	Number of subcomponents in a component	4 (cmp. A)
		Number of classes in a component	6 (cmp. A)

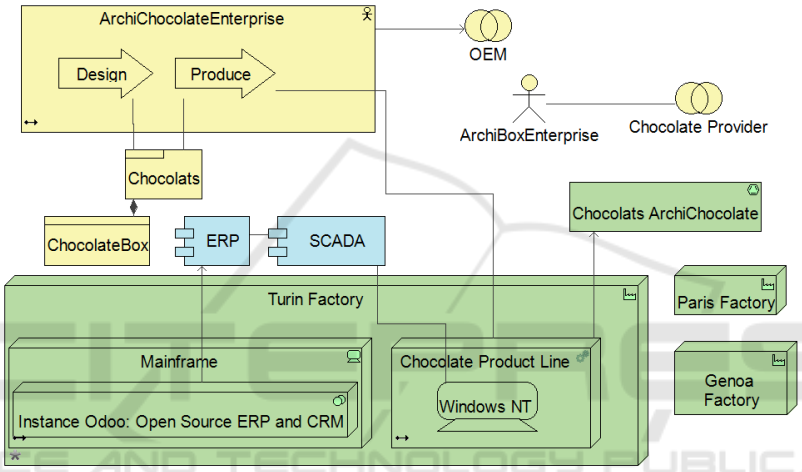


Figure 8: Archimate model of the enteprise architecture of the fictional "ArchiChocolate" company.

as before. The model contains 634 elements (as reported by `return Model.allInstances.size;` through Hawk) and was indexed in 8 seconds and 48 milliseconds with a recent laptop. Particularly, a Thinkpad X1 with an i7-6600U CPU, 16GiB of RAM, and a solid-state disk, running Ubuntu 16.04.3 and Linux 4.4.0-98, using Oracle Java 8u102 and the latest version of Hawk at the time of writing (commit "66edee" on Github). The model is not very large, but architectural models are important for early high-level analysis. Listing 4 shows an example of a query which, for each Concept in the model:

- Annotates each concept with the number of concepts that it is related to, transitively.
- Sorts them from the ones with the most related concepts to the least.
- Collects the type name, name and number of "related to" concepts for each.
- Returns the top 5 of those elements.

This type of query could be used for impact analysis of a potential change in the model: the top 5 elements would be the ones that would have the largest effect if they were removed.

Listing 5: Response from Thrift for Archimate query.

```
1 QueryReport(result:[
2   [BusinessActor ArchiChocolateEnterprise, 13],
3   [BusinessProcess Produce, 10], [Facility Turin, 7],
4   [Facility Paris, 5], [Equipment ChocoProdLine, 4]
5 ], wallMillis: 43)
```

When executed through the Thrift API, this query produces an output like that shown on Listing 5. The query itself took 54ms from the client to the server while running in a local instance (as this was a proof of concept), but it only took 43ms to run within the server itself: the other 11ms were network overhead.

4.2 Scalability

The next part was to index a large collection of Modelio 3.7 models with this new version of Hawk. This evaluation was conducted by Softeam, with the help

Table 2: Indexing times per project and processing step for the scalability study, in seconds.

Modelio project	`analyst`	`app`	`archimate`	`bpmn`	`uml`	**Total**
Indexable files	1215	4543	2226	763	2305	11052
Fragment insertion	79	412	64	48	151	754
Fragment connection	105	301	167	140	248	960
Children derivation	103	388	105	66	235	897

Listing 6: EOL source code for the "average classes per component" query.

```
1  var components = Component.all;
2  return components.collect(c|c.countClasses()).sum() / components.size;
3
4  operation Component countClasses() {
5    return self.OwnedElement.select(c|c.isTypeOf(Class)).size
6    + self.OwnedElement.select(p|p.isTypeOf(Package)).collect(p|p.countClasses()).sum()
7    + self.OwnedElement.select(subc|subc.isTypeOf(Component)).collect(subc|subc.countClasses()).sum();
8  }
9  operation Package countClasses() {
10   return self.OwnedElement.select(c|c.isTypeOf(Class)).size
11   + self.OwnedElement.select(p|p.isTypeOf(Package)).collect(p|p.countClasses()).sum();
12 }
```

of the Hawk developers. The models were those used for the internal development of Modelio itself: these change for every release of Modelio, and are used to generate over a million lines of code. The Modelio projects added up to 3.7GB on disk, where 2.16GB was from the 11 052 `.exml` files to be indexed. These had 452 084 model elements, according to Hawk.

The indexing process was done on a laptop with an Intel i7-6500U CPU, 8GiB of RAM, and an SSD running Oracle Java 8u60 over Windows 10, and the same version of Hawk as above. The OrientDB backend was used. Indexing took 2686 seconds in total (45 minutes), which is slightly higher than the breakdown shown on Table 2 (2611s). The table has the times needed to index the `.exml` fragments, connect them, and (optionally) derive their parent-children relationships.

While it may seem expensive at first glance, it is important to note that this high upfront cost only needs to be paid once: later changes to the models will be processed incrementally by Hawk, with a cost roughly proportional to the change of the model. In our previous study, this cost was quickly amortised as queries were faster than with just Modelio (Garcia-Dominguez et al., 2016).

As for the queries, we first ran some simpler examples in this case to see how quickly we could count all instances of a certain type. This would give us a rough estimate of how quickly we could find certain subsets of the entire collection of models. These were queries of the form `return X.all.size;`, except for the one counting all model elements, which was `return Model.allInstances.size;`. The results are shown on Table 3: it is possible to iterate over all instances and count them in 15 seconds, and we can find all 8545 classes across all projects in less than two seconds.

Table 3: Times required to count all instances.

Type	Time (s)	Count
(All types)	15.50	452084
Attribute	0.73	7403
Class	1.93	8545
Component	0.20	118
Interface	0.20	370
Operation	1.45	52502

Next, we decided to run one of the MEASURE industrial queries, particularly the "average classes per component" query from Table 1 shown in Listing 6. This query ran across the indexed 2.16GB of models in only 1666ms, producing the end result of 59.49 classes on average per component. Results like these show that Hawk can produce answers in seconds for queries over very large models, by taking advantage of the efficient navigation of references in graph databases.

5 RELATED WORK

The increasing size of industrial models has given rise to several other high-performance model persistence and model querying technologies. If the model is stored in a database in the first place, it may be faster

to query than if it is stored on files: this is possible with NeoEMF (Gómez et al., 2015), an alternative model persistence layer for EMF models with support for Neo4j and MapDB among other technologies. Another similar option is MongoEMF[2], which uses the MongoDB document database.

Beyond database-backed single models, database storage of entire collections is also possible with model repository technologies such as Eclipse Connected Data Objects (CDO)[3] or Morsa (Pagán et al., 2013). CDO in particular is very mature and supports both relational and document-oriented databases.

Unfortunately, none of these technologies would have been of much use for MEASURE straight away, as Modelio models are not based on EMF. Even if a mapping to EMF were implemented, we would be left with two options: either replace the persistence technology in the original models (which is non-trivial and intrusive on the user experience), or implement an incremental synchronisation approach between the original models and our EMF-based copy for indexing. The second alternative would have been roughly equivalent to what was already in Hawk, without its other benefits of incremental/derived attributes and a web service API.

6 CONCLUSION AND FUTURE WORK

The MEASURE project is developing a platform for collecting metrics across the entire software development lifecycle. In model-driven processes, the specification and design phases operate on models rather than on code: it is necessary to extract measurements from models as well. It can be very expensive to collect metrics across large collections of models, or from very large models. This paper has shown the first version of an integration between the MEASURE platform and the Hawk model indexing framework, with positive results in terms of flexibility and performance. Hawk makes it possible to use "big data"-class NoSQL technologies for efficient querying of existing models with little technical risk.

Hawk can now index any model supported by current and future versions of Modelio without changes in its code. It has been used in this paper to index both Modelio 3.6 and 3.7 models, some of them from the enteprise architecture domain and some from the object-oriented software design domain. Hawk servers can be provisioned in an automated way without involving the use of its API, and it is possible to deploy both custom and predefined EOL queries in the MEASURE platform. Hawk was also used to index 2GB of industrial Modelio models: while there is some upfront cost in the indexing, it is quickly amortised through the faster execution of the queries and the incremental updates for later changes in the models.

Regarding future work, the MEASURE platform will be extended with analysis capabilities that combine multiple metrics, possibly from different artefacts and phases in the software development lifecycle. Hawk will be further validated through the implementation of additional queries from the MEASURE industrial partners, and some of the queries will be optimised with the use of derived and indexed attributes.

[2]https://github.com/BryanHun/mongo-emf

[3]http://wiki.eclipse.org/CDO

ACKNOWLEDGEMENTS

The research leading to these results was partially funded by the ITEA3 project 14009, MEASURE.

REFERENCES

Abherve, A. (2017). Github project for the SMM Measure API library. https://github.com/ITEA3-Measure/SMMMeasureApi. Last accessed on 2017-11-01.

Abherve, A., Bagnato, A., Stefanescu, A., and Baars, A. (2017). Github project for the MEASURE platform. https://github.com/ITEA3-Measure/MeasurePlatform/graphs/contributors. Last accessed on 2017-11-01.

Al-wadeai, O. (2017). Github project for the Hawk query SMM measure library. https://github.com/Orjuwan-alwadeai/HawkQuerySMMMMeasureLib. Last accessed on 2017-11-01.

Apache Software Foundation (2017). Apache Thrift project website. http://thrift.apache.org/. Last accessed on 2017-11-01.

Barmpis, K. and Kolovos, D. S. (2013). Hawk: towards a scalable model indexing architecture. In *Proceedings of the Workshop on Scalability in Model Driven Engineering*, BigMDE '13, pages 6:1–6:9, New York, NY, USA. ACM.

Garcia-Dominguez, A., Barmpis, K., Kolovos, D. S., da Silva, M. A. A., Abherve, A., and Bagnato, A. (2016). Integration of a graph-based model indexer in commercial modelling tools. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*, pages 340–350, Saint Malo, France. ACM Press.

Gómez, A., Tisi, M., Sunyé, G., and Cabot, J. (2015). Map-based transparent persistence for very large models. In Egyed, A. and Schaefer, I., editors, *Fundamental Approaches to Software Engineering*, volume 9033

of *Lecture Notes in Computer Science*, pages 19–34. Springer Berlin Heidelberg.

Kolovos, D. S., Garcia-Dominguez, A., Paige, R. F., Guerra, E., de Lara, J., Rath, I., Varr, D., Suny, G., and Tisi, M. (2016). MONDO: Scalable Modelling and Model Management on the Cloud. In *Joint Proceedings of the Doctoral Symposium and Projects Showcase Held as Part of STAF 2016 co-located with Software Technologies: Applications and Foundations (STAF 2016)*, Vienna, Austria.

Kolovos, D. S., Rose, L., Garcia, A., and Paige, R. (2017). The Epsilon book. http://www.eclipse.org/epsilon/doc/book/. Last accessed on 2017-11-01.

Object Management Group (2016). The Software Metrics Meta-Model Specification 1.1.1. http://www.omg.org/spec/SMM/1.1.1/. Last accessed on 2017-11-01.

Pagán, J. E., Cuadrado, J. S., and Molina, J. G. (2013). A repository for scalable model management. *Software & Systems Modeling*, 14(1):219–239.

Shields, A. (2014). Must-know: an overview of the software industry - Market Realist. http://marketrealist.com/2014/07/must-know-overview-software-industry-2/.

Softeam Cadextan (2017). Modelio project website. https://www.modelio.org/. Last accessed on 2017-11-01.

Softeam R&D (2017). MEASURE project website. http://measure.softeam-rd.eu/. Last accessed on 2017-11-01.