
Proceedings Paper:
Garcia-Dominguez, Antonio, Barmpis, Konstantinos, Kolovos, Dimitrios S. et al. (2016) Integration of a graph-based model indexer in commercial modelling tools. In: Proceedings - 19th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS 2016. 19th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS 2016, 02-07 Oct 2016 ACM, FRA, pp. 340-350.

https://doi.org/10.1145/2976767.2976809

Integration of a Graph-Based Model Indexer in Commercial Modelling Tools

Antonio Garcia-Dominguez
University of York
Deramore Lane, York
YO10 5GH, United Kingdom
antonio.garcia-dominguez@york.ac.uk

Konstantinos Barmpis
University of York
Deramore Lane, York
YO10 5GH, United Kingdom
konstantinos.barmpis@york.ac.uk

Dimitrios S Kolovos
University of York
Deramore Lane, York
YO10 5GH, United Kingdom
dimitris.kolovos@york.ac.uk

Marcos Aurelio Almeida da Silva
Softeam Cadextan
21 Avenue Victor Hugo, Paris
75016 France
marcos.almeida@softeam.fr

Antonin Abherve
Softeam Cadextan
21 Avenue Victor Hugo, Paris
75016 France
antonin.abherve@softeam.fr

Alessandra Bagnato
Softeam Cadextan
21 Avenue Victor Hugo, Paris
75016 France
alessandra.bagnato@softeam.fr

ABSTRACT

Softeam has over 20 years of experience providing UML-based modelling solutions, such as its Modelio modelling tool, and its Constellation enterprise model management and collaboration environment. Due to the increasing number and size of the models used by Softeam's clients, Softeam joined the MONDO FP7 EU research project, which worked on solutions for these scalability challenges and produced the Hawk model indexer among other results. This paper presents the technical details and several case studies on the integration of Hawk into Softeam's toolset. The first case study measured the performance of Hawk's Modelio support using varying amounts of memory for the Neo4j backend. In another case study, Hawk was integrated into Constellation to provide scalable global querying of model repositories. Finally, the combination of Hawk and the Epsilon Generation Language was compared against Modelio for document generation: for the largest model, Hawk was two orders of magnitude faster.

CCS Concepts

•**Software and its engineering** → **Unified Modeling Language (UML);** *Software performance;* Interoperability; •**Information systems** → *Specialized information retrieval;*

Keywords

scalability; model querying; model-driven engineering; model fragmentation

1. INTRODUCTION

Softeam is a French consulting and technology service company with over 20 years of experience in building modelling environments, over 800 employees and operations in London, Singapore and Paris. Two of Softeam's products are the open source Modelio modelling environment, and the recently developed Constellation enterprise model management and collaboration solution. Softeam noticed that their clients were producing increasingly larger models and collaborating over larger teams, and decided to participate in the MONDO EU FP7 project on scalable model-driven engineering to improve their products in this regard.

The MONDO EU FP7 project was motivated by the identification of scalability as a major issue that prevented wider use of MDE [10, 13] and its well-documented benefits of improved productivity and reuse [12]. The MONDO project consisted of multiple work packages dedicated to tackle scalability on its various perspectives: notations, queries and transformations, collaborative modelling, and model persistence. Within the scalable model persistence work package, the Hawk model indexer was developed by the University of York [2]. Hawk can monitor repositories containing file-based models and mirror their contents into a graph that can be leveraged to perform efficient querying.

Within the Softeam use case in MONDO, it was decided to integrate Hawk into the Softeam toolset and evaluate its advantages over its existing approaches. This paper discusses the challenges involved in this integration, how they were addressed and the results that were produced. It also introduces several new features in Hawk, some of which were motivated by these integration efforts.

The rest of this paper is structured as follows: Section 2 provides the necessary background on Modelio, Constellation and Hawk. Section 3 discusses the various steps involved in the integration process. Section 4 describes three case studies in which the costs and benefits of using Hawk were evaluated, and Hawk was integrated into the Constellation administration server and web-based user interface. Section 5 mentions related works, and Section 6 presents conclusions and future lines of work.

2. BACKGROUND

This section introduces the main concepts discussed throughout the rest of the work. It presents the Modelio toolset

developed by Softeam and the Hawk model indexer, and discusses certain technical details that are important for the presented integration case studies.

2.1 Modelio

Modelio is a commercial open-source modelling environment developed by Softeam Cadextan[1]. It allows modellers to create and manage models in various notations, including UML, BPMN, SysML, TOGAF and SoaML, among others. It is licensed under the GNU General Public License version 3 and is written in Java, being implemented as an Eclipse Rich-Client Platform application. Modelio includes its own scripting environment for various modeling tasks (e.g. code generation), based on the Jython implementation of the Python programming language.

Unlike many tools in the MDE research community, Modelio is not based on the Eclipse Modelling Framework (EMF): users can extend it with new modules that provide additional UML profiles. Models are handled through Modelio's bespoke modelling infrastructure. XMI is only used for model interchange within certain metamodels: for instance, UML models can be imported from/exported to EMF/MOF XMI.

Modelio users work within one *project* at a time. In the local system, a Modelio project consists of a folder with a certain structure. The model elements themselves are contained within `.exml` and `.ramc` files, but projects also include various configuration files, small binary files with management metadata and an *ad hoc* binary index to speed up certain queries.

Conceptually, a Modelio project can be thought of as a single large model. However, projects can get quite large and it may be needed to reuse certain parts across projects (e.g. reverse engineered models from Java libraries). To cope with this, projects are divided into *model fragments* according to a predefined strategy. A *model fragment* is a collection of *model elements* with a single root object and a subtree of objects contained within. *.exml* files contain model fragments, which may be shared across projects, and *.ramc* files are ZIP archives that contain `.exml` files with some additional metadata.

An example of a simple `.exml` file containing the model fragment for a UML *Component* class is shown in Listing 1. This class is within the *Architecture* package and inherits from the *Element* class. The contents of this file are organized as follows:

- Line 1 indicates that this file conforms to version 3 of the EXML format, and lines 2–5 list all the fragments that are required to resolve this fragment by UID (Unique IDentifier).

- The "OBJECT" element in line 6 is the root object within this fragment: the *Component* class itself. The "ID" element in line 7 indicates that the metaclass or *MClass* (equivalent to the EMF *EClass*) of the object is *Class* (a UML class), and that the object has the UID (Unique IDentifier) "2d7b...". The "PID" (parent ID) element in line 8 indicates that *Component* is contained within the *Architecture* package with the UID "ea87...".

  Modelio UIDs are 16 bytes long and are assigned randomly to new objects: they are guaranteed to be unique

[1]https://www.modelio.org

Listing 1: Example Modelio `.exml` file with the model fragment for the *Component* UML class

```
1  <EXT object="Component" version="3">
2   <DEPS>
3    <ID name="Component" mc="Class" uid="2d7b..."/>
4    <EXTID name="Element" mc="Class" uid="4ed7..."/>
5   </DEPS>
6   <OBJECT>
7    <ID name="Component" mc="Class" uid="2d7b..."/>
8    <PID name="Architecture" mc="Package" uid="ea87..."/>
9    <ATTRIBUTES>
10    <ATT name="Name">Component</ATT>...
11   </ATTRIBUTES>
12   <DEPENDENCIES>
13    <COMP relation="OwnedOperation">
14     <OBJECT>
15      <ID name="execute" mc="Operation" uid="795e..."/>
16      <PID name="Component" mc="Class" uid="2d7b..."/>
17      <ATTRIBUTES>
18       <ATT name="Name">execute</ATT>...
19      </ATTRIBUTES>
20      <DEPENDENCIES>...</DEPENDENCIES>
21     </OBJECT>
22    </COMP>
23    <COMP relation="Parent">
24     <OBJECT>
25      <ID name="..." mc="Generalization" uid="58e6..."/>
26      <PID name="Component" mc="Class" uid="2d7b..."/>
27      <ATTRIBUTES>...</ATTRIBUTES>
28      <DEPENDENCIES>
29       <LINK relation="SuperType">
30       <ID name="Element" mc="Class" uid="4ed7..."/>
31       </LINK>
32      </DEPENDENCIES>
33     </OBJECT>
34    </COMP>
35   </DEPENDENCIES>
36  </OBJECT>
37 </EXT>
```

within a Modelio project. Across projects, Modelio assumes that two "OBJECT"s with the same UID are conceptually the same model element (e.g. during merging). This is very common when reusing model libraries (`.ramc` files) that provide read-only model fragments that were reverse-engineered from popular Java libraries, for instance.

- The "ATTRIBUTES" element in line 9 provides a list of name-value pairs for the *MAttribute*s of *Component* (similar to EMF *EAttribute*s).
- The "DEPENDENCIES" element in line 12 provides another list of name-value pairs for the *MDependency*s of *Component* (again, similar to EMF *EReference*s). A "COMP" child element (such as those in lines 13 or 23) represents a composition (equivalent to a containment *EReference*). A "LINK" child element represents a simple association and does not imply containment.

  Interestingly, even if "COMP" is used, the nested objects might be contained in a different object if they have an explicit "PID" pointing to something else. In this case the two nested objects have explicit PIDs, but they are still pointing to *Component*. This is important in certain corner cases (e.g. UML associations and association ends, in which containers are assigned differently based on the navigability of the association). This is different from EMF XMI, in which element nesting is always equivalent to containment.

  An example of "LINK" is shown in line 30. In this case, it points to the UID of the *Element* superclass of *Component*, which is defined in a different model fragment. Again, this purely UID-based reference is different from EMF, whose references combine a location and a "fragment" within the location. In fact, "COMP" elements can also use ID-based references through "COMPID" elements (not shown here).

In summary, Modelio has many similarities with EMF, but it has three major differences: container relationships are preferred to containment relationships, associations between model elements are entirely based on unique identifiers, and the same model element may be contained in multiple model fragments at once. These two differences will be important to the integration efforts in Section 3.

2.2 Constellation

Constellation[2] is a client/server-based solution developed by Softeam which allows users of Modelio to collaborate with each other. It can host model fragments, allowing their collaborative use across teams, and can organise these fragments into catalogues and monitor their evolution.

A deployment of Constellation is organised as shown on Figure 1 and involves two types of server nodes: the *Administration Server* and one or more *agent*s. The administration server coordinates all the agents and provides a centralised view of all their information. The agents host the model fragments themselves and may be of various types depending on the underlying access mechanism. At the time of writing, two agent types exist: SVN model agents that access a standard Subversion version control system and HTTP model agents that access a HTTP server hosted elsewhere by the user.

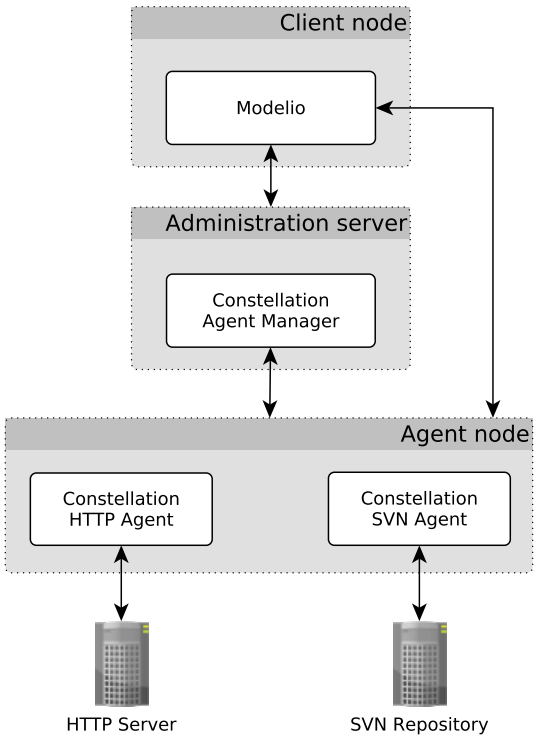

Figure 1: Typical deployment of Constellation for collaborative modelling

Constellation provides its own web interface for browsing through the projects and teams and an API that Modelio can use. However, before the integration described in this paper it did not have any model query capabilities of its own. Users wishing to search for collections of model elements of interest would have to load the fragments in Constellation into their Modelio instance and use the local search or scripting capabilities.

2.3 Hawk

Hawk [2] 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. Using a database makes it possible to load only the part of the model that is needed at a time, and the join-free adjacency provided by modern graph databases can speed up many common queries. Hawk consists of multiple types of components:

- *Backend* components integrate Hawk with a specific graph database technology: implementations exist for OrientDB[3] and Neo4j[4].
- *Repository connector* components allow Hawk to read files from local folders, SVN/Git repositories, Eclipse workspaces, among others.

[2]https://www.modeliosoft.com/en/products/modelio-constellation.html

[3]http://orientdb.com

[4]http://neo4j.com

- The files are parsed by *model drivers* that understand specified modelling technologies and metamodels (e.g. generic EMF models in XMI format or IFC models in the STEP format).
- The *updater* takes the detected changes in the model files and turns them into graphs.
- The *query language* component provides a convenient way to perform fast and efficient queries that make the most out of the underlying graph database: the current implementation is based on the Epsilon Object Language [9].
- The *model view* components expose the Hawk index itself as a model. Currently, it is possible to expose an entire index or parts of it as Epsilon or EMF models.

Figure 2 shows the graph Hawk would build from a simple Library model based on EMF:

- The original Library metamodel would be turned into one *metamodel node* for the original *EPackage* and two *type nodes* for the *Author* and *Book EClass*es.
- The model would be turned into a *file node* that would contain two *element nodes*: one for the *Book* "b1" and one for the *Author* "a1". The original association from the *Book* to the *Author* would be mapped to an edge, and additional edges would represent their types.
- Two indices would be built: a *metamodel index* from namespace URIs to metamodel nodes, and a *file index* from the full location of the file (repository and path) to the file node.

Later versions of Hawk have added support for indexed and derived attributes, further speeding up queries [4]. The EOL query engine in Hawk is aware of when an attribute has been indexed and will replace filtered iteration with direct lookups when using the OCL-inspired *select* operation and other similar ones. Derived attributes allow Hawk to precompute partial results and reuse them in later queries, possibly as direct lookups as well.

Hawk can be used in many different ways: as a Java library, as an Eclipse plugin, as an Epsilon or EMF model, or as a network service through its Thrift API. More details will be provided about these access methods throughout the rest of this work as necessary.

3. INTEGRATION

As mentioned during the introduction, Softeam intended to evaluate Hawk in two ways: to extend their products with efficient global search capabilities through a concise query language, and to speed up code generation through the use of graph databases. This required completing several tasks:

- Creating a standalone representation of the Modelio metamodels.
- Developing a standalone parser for `.exml` and `.ramc` files.
- Integrating the metamodels and the parser into a new Modelio model driver for Hawk.
- Generating an EMF-compatible metamodel from the Modelio metamodels.

The rest of this section will explain how these tasks were completed and present several new features of Hawk that were necessary for the case study.

3.1 ModelioMetamodelLib

The first hurdle to solve when integrating Modelio and Hawk was a legal one: since most of Modelio is under the GPLv3 strong copyleft license, integrating any of these components directly would require releasing Hawk under the GPLv3 and not under the EPL. This would have hindered future integration efforts with other EPL-compatible tools, which was undesirable.

Instead, the developers from Softeam and Hawk agreed on an alternative solution that would not require licensing changes: since the Modelio metamodels are developed as Modelio models themselves, Softeam created a new model-to-text transformation that produced ModelioMetamodelLib[5] (MML). MML is a small Java library (around 4KLOC) released under an EPL-compatible license (the Apache Software License) that exposes the Modelio metamodel as a set of Java classes. The metamodel is accessible through instances of the *MMetamodel* class through their *getMPackages*, *getClassByName*, and *getDataTypeByName* methods, exposing all the *MPackage*s, *MClass*es, *MAttribute*s, *MDependency*s and *MDataType*s.

3.2 EXML parser

While recent versions of Modelio had cleanly separated the EXML parser as a reusable component, its GPLv3 license presented the same legal issues as above. In this case, the Hawk developers created a clean-room description in natural language of the Modelio EXML file format and validated it with Softeam. Based on this clean-room description, the Hawk developers created a Java StAX parser for EXML.

Java StAX parsers operate on a "pull streaming" basis: they only keep the last parsing event in memory, and produce further events when instructed by the user. This makes it possible to process the file without having it entirely in memory at once. In order to keep the parsing algorithm as robust as possible (since the Modelio EXML format may evolve slightly over time), it focuses on the specific elements mentioned in Section 2.1 and ignores the rest.

The algorithm is shown in pseudocode in Listing 2. It is a recursive algorithm that takes advantage of the simplicity that StAX parsers provide in comparison to SAX parsers ("push streaming" parsers that send all events to a callback). The parser works with only 2 Java classes: *ExmlReference*s that consist of a name, *MClass* and UID triplet, and *ExmlObject*s that extend *ExmlReference*s with the ability to contain attributes, links and compositions.

The parser also includes a component for iterating through all the model fragments in a `.ramc` archive, decompressing in memory and ensuring that only one model fragment is in memory at each time. The last detail is especially important, as `.ramc` files can grow to large sizes: a 30MB `.ramc` file might expand to 200MB+ in memory if all model fragments are loaded at once.

3.3 Modelio model driver

[5]https://github.com/aabherve/modelio-metamodel-lib

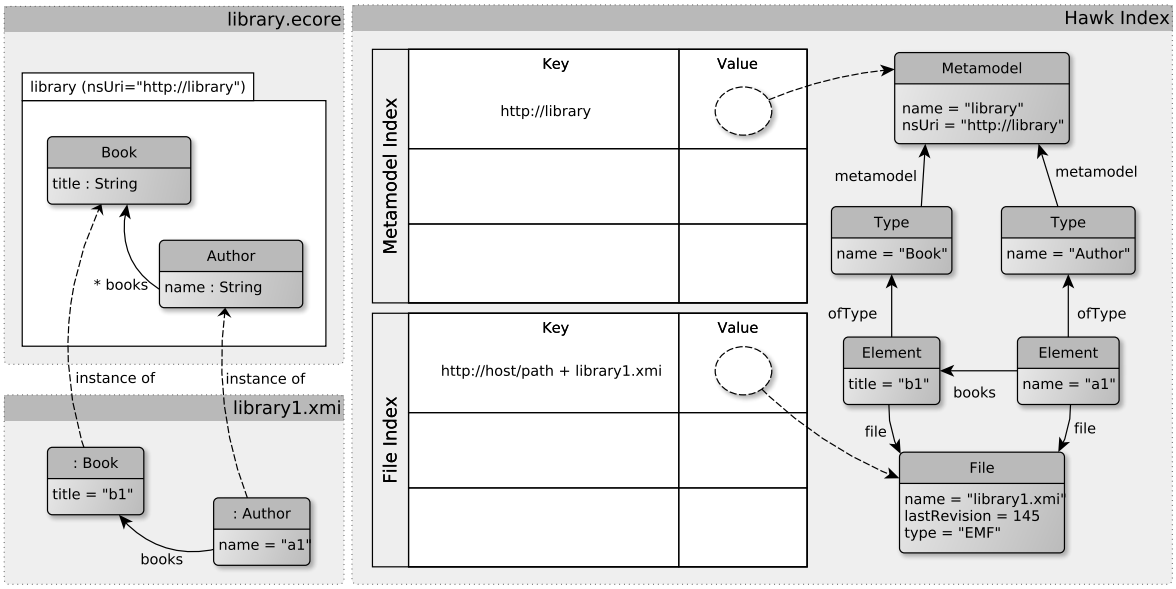

Figure 2: Example of a Hawk model index

To make Hawk understand Modelio `.exml` and `.ramc` files, ModelioMetamodelLib and the EXML parser were integrated into a new Modelio model driver component for Hawk. The model driver component includes three components: a metamodel resource factory that allows Hawk to generate metamodel and type nodes for the *MPackage*s in ModelioMetamodelLib, a model resource factory that produces Hawk objects from the *ExmlObject*s, and a graph change listener that reacts to key events happening inside a Hawk index.

Hawk needs to deal with the special features of Modelio mentioned in Section 2.1:

- UID-based references: Modelio metamodel elements and model elements indicate to the Hawk updater that their URI fragments are globally unique. In this case, Hawk will associate the URI fragment to the graph node in an *object-by-ID index*, and will use it to resolve references flagged to be UID-based.

- A model element can be contained in multiple model fragments: the Modelio model resource indicates that it may contain model elements that are shared with other resources. Hawk will then reuse the same object-by-ID index to check if the model element is already indexed from another resource, and if so it will reuse the node and simply record the fact that it is contained in one more file.

- Explicit container references that override containment references: the Modelio model elements report explicit container (from child to parent) references as a new *hawkParent* reference in all Modelio types, and do not report any explicit containment (from parent to child) references. This is enough for querying the graph with EOL and for using the index as an Epsilon model: when using *eContents* from Epsilon, the children elements will be computed on the fly.

  However, when exposing a remote Hawk index as an EMF model over the network, explicit containment references need to be sent in advance to the client. These containment references must also have been computed in advance as well. In this case, an additional component in the Modelio model driver can be optionally enabled: a graph change listener that will extend all Modelio types with a *hawkChildren* derived reference. These derived references are an extension of previous work on derived attributes [4].

  The derived reference is defined through an EOL query ("return self.revRefNav_hawkChildren;"). This simple query retrieves the source nodes of all the incoming *hawkChildren* references: the "revRefNav_" tells Hawk to perform reverse reference navigation.

3.4 Ecore Metamodel Generation

When exposing a local or remote Hawk index as a standard EMF model, it is necessary to associate an *EClass* to every *EObject* created from each model element node. This *EClass* must be an accurate mapping of the original *MClass* to avoid losing information when using the Modelio models from EMF.

These *EClass*es have been generated through a Java-based transformation that maps every *MPackage* to an *EPackage* and so forth. *EClass*es are further extended with the *hawkChildren* and *hawkParent* references mentioned in Section 3.3, in order to allow EMF to recreate the original containment tree. The resulting metamodel has 289 *EClass*es divided over 53 packages.

With this metamodel, the EMF drivers that expose local and remote Hawk indexes can access the type node of the model element node, obtain the metamodel URI and type name and retrieve the equivalent *EClass*. An *EObject* can be produced from the *EClass* and filled in with the information from the model element node.

4. CASE STUDIES

Listing 2: Pseudocode for the EXML parser

```
1  main(file) {
2    open file with StAX parser
3    advance to first OBJECT
4    exmlObj = new ExmlObject(file)
5    fillObject(exmlObj)
6    close file
7  }
8
9  fillObject(exmlObj) {
10   currentLink = null
11   currentComposition = null
12   do {
13     advance to next XML element start
14     if event is element start {
15      if element is ID, FOREIGNID or EXTID {
16          if currentLink is null {
17            fill in exmlObj with mc, name and UID
18          } else {
19            newRef = new ExmlReference(file)
20            fill in newRef with mc, name and UID attributes
21            add newRef to currentLink
22          }
23        } else if element is PID {
24          fill in parent of exmlObj with mc, name and UID
25        } else if element is COMPID {
26          newRef = new ExmlReference()
27          fill in newRef with mc, name and UID attributes
28          add newRef to composition currentComp
29        } else if element is ATT {
30          fill in attribute of exmlObj with name and value
31        } else if element is LINK {
32          currentLink = value of "name" attribute
33        } else if element is COMP {
34          currentComp = value of "name" attribute
35        } else if element is OBJECT {
36          nestedObj = new ExmlObject(file)
37          fillObject(nestedObj)
38          add nestedObj to currentComposition
39        }
40   } while (not at end of OBJECT)
41 }
```

File type	`.exml`	`.ramc`
Minimum	229B	23kB
Q1	953B	1.88MB
Median	1.51kB	8.45MB
Mean	8.41kB	11.59MB
Q3	3.00kB	22.38MB
Maximum	2.77MB	30.98MB
Count	201 526	23

Table 1: Distribution of `.exml` and `.ramc` sizes in Modelio workspace with 15 internal projects

Having completed the initial technical work on integrating Hawk into Modelio and Constellation, Softeam and the Hawk developers conducted a series of case studies to evaluate the performance of Modelio. These case studies used two types of models:

- The first type (*InternalWorkspace*) consisted of 15 internal projects, which took up 3.9GB in total including indices: 2.0GB were `.exml` and `.ramc` files that contained over 1.23M unique model elements. These were intended to test if Hawk could scale to realistic collections of Modelio projects.

  File sizes were distributed as shown in Table 1: 75% of the `.exml` files were within 3kB, while 75% of the `.ramc` files were within 22.38MB.

- The second type (*SyntheticWorkspace*) was a collection of synthetic Modelio projects with UML models of various sizes, which were randomly generated with a Jython script inside Modelio, written by Softeam. These were intended to evaluate the effect of model size on Hawk's memory usage and execution time.

  The script took a multiplier parameter m and produced a random number of packages (uniformly distributed between $15m$ and $20m$), with each package having between 40 and 65 classes, and each class having between 5 and 15 attributes and between 15 and 25 operations. Metrics for the generated projects are shown in Table 2.

The rest of this section shows how these two types of models were used in three case studies: ensuring Hawk could process real-world collections of Modelio projects, integrating Hawk into the Constellation product and using Hawk to speed up code and document generation from Modelio models. All these case studies were conducted on a laptop with an Intel i7-5600 CPU, 16GiB of RAM, and a SanDisk SATA3 256GB SSD running Ubuntu 15.10 and Linux 4.2.0-35-generic, using Oracle Java 8u60 and the latest version of Hawk at the time of writing (commit "e329048" on Github).

4.1 Testing Real-World Scalability

The first question asked by Softeam was if Hawk could deal with the large models that their clients managed in their day-to-day work, and which would be the system requirements. To answer these questions, the *InternalWorkspace* models were indexed by Hawk using the Neo4j backend, giving the Java VM different amounts of memory through the `-XmsNg -XmxNg` options (where N is the number of GB to allocate). The G1 garbage collector was used by passing the `-XX:+UseG1GC` option, as recommended by Neo4j.

Each of the generated Neo4j databases took up 2.5GB of disk space. The results are shown on Table 3, which includes the times of the three main stages in which Hawk processes the Modelio model fragments are : adding each model fragment to the graph, connecting the fragments together and optionally deriving the *hawkChildren* containment references for EMF. An additional row shows the total time required, which is slightly higher than the sum due to other minor operations before and after these main stages.

Hawk was also run with only 1GB of RAM, but in that case containment derivation had to be disabled in order to complete the process (which took 3146s), by disabling the

Metric	$m = 1$	$m = 2$	$m = 4$	$m = 8$	$m = 16$
Project size	80MB	150MB	279MB	814MB	1.8GB
Size of `.exml` files	58MB	104MB	179MB	422MB	752MB
Number of `.exml` files	1255	2087	3430	7849	13790
Number of model elements	47981	88451	154281	367840	657228

Table 2: Metrics for the randomly generated Modelio projects, by multiplier generator parameter m

Main stages	2GB	4GB	8GB
Fragment insertion	2554s	2502s	2371s
Fragment connection	728s	541s	443s
Containment derivation	1851s	1234s	1123s
Sum + minor ops	5171s	4287s	3958s

Table 3: Indexing times by stage and memory amount for the *InternalWorkspace* projects (Neo4j backend)

ModelioGraphChangeListener when creating the Hawk index through the UI. This is because of the memory overhead incurred by the Lucene indexes used to store the derived references. While disabling containment derivation results in flat Hawk EMF resources, EOL queries and the Epsilon model driver can still reproduce the containment tree on the fly. Future versions of the Hawk EMF resource could do this on-the-fly computation as well to overcome this issue.

These results confirm that Hawk can index the collections of projects that Softeam's customers deal with on a day-to-day basis, being able to index a 2GB folder with 1GB of RAM without derived references or 2GB with derived references. Being able to use only 1GB of RAM is important for cloud deployments (e.g. AWS *t2.micro* instances), and additional memory can be used to speed up the process, with diminishing returns. One of the key factors that enable this low memory usage is the fact that Hawk only keeps in memory one model fragment at a time, even when processing a `.ramc` archive: as shown in Table 1, 75% of `.exml` files are within 3kB and the largest file is only 2.77MB.

4.2 Providing Global Remote Queries

After verifying that Hawk could index the usual collections of projects that Softeam's clients use, Softeam asked the Hawk developers to produce a distribution of Hawk that they could integrate into their Constellation product as a regular Java library. It should be redistributable under open source and commercial licenses without additional licensing costs and be able to index both SVN repositories and individual HTTP locations. While Hawk was already usable as a Java library, the other two requirements required additional work from the Hawk developers.

At the time, Hawk only supported Neo4j as a backend as it produced the best results in previous benchmarks [3]. However, Neo4j is only freely redistributable under the GPL, while Hawk is under the EPL: to avoid a licensing conflict, Hawk had to avoid redistributing Neo4j in any form, so users always had to download Neo4j and compile Hawk on their own. In order to be able to produce redistributable releases, an alternative graph database technology with a more permissive license would have to be integrated. OrientDB was selected as it was based on pure Java and was available under the EPL-compatible ASL 2.0 license. Initial implementations used the OrientDB Graph API, but after some benchmarks the low-level Document API was used instead. Likewise, initial versions used the OrientDB Lucene integration for internal indexes, but its limited features required moving to regular SB-Trees, which supported all the operations required by Hawk.

Another limitation was that Hawk could not directly read models from HTTP locations. To solve this problem, a new repository connector component was implemented, which could monitor a specific HTTP location for changes using various HTTP headers (e.g. *ETag*, *Last-Modified*, or *Content-Length*). Due to limitations in the HTTP protocol it would not be able to browse HTTP-hosted directories recursively, but at the time Softeam deemed this functionality to be sufficient for their purposes.

Having lifted these two limitations, a binary distribution of Hawk using the OrientDB backend was given to Softeam for integration. Softeam dedicated less than 3 months to produce a first version of the integration into the Constellation administration server and its web UI, and then continued improving it iteratively. A screenshot of the first release of the web-based model query UI is shown in Figure 3: users can enter an EOL query and the results are displayed in paginated form on a table below.

While our prior benchmarks indicate that OrientDB does result in a performance hit, execution times and memory usage have shown to be acceptable during internal testing at Softeam, as a long-running service that processes changes in model fragments incrementally. Nevertheless, the design of Hawk could allow Softeam to offer a Neo4j-based version in the future without any additional development work: the commercial Neo4j license would be included in the price for Constellation.

4.3 Speeding Up Document Generation

Having been successful in integrating Hawk into Constellation, Softeam decided to look into other applications for their Hawk indices. Since Softeam intended to extend Constellation in the future with document/code generation agents, it was decided to evaluate if the combination of Hawk (exposed as an Epsilon model) and the Epsilon Generation Language (EGL [15]) could generate code and documents faster than the Modelio Jython scripting environment they used at the moment.

To test this, the *SyntheticWorkspace* projects were indexed separately by Hawk, using the OrientDB backend and `-Xmx1g` to emulate what could happen within an AWS *t2.micro* instance running Constellation, and disabling derived containment since the indices would be used only from Epsilon. The results are shown on Figure 4: both indexing time and OrientDB index scaled linearly over project size (measured in model elements). For the largest project with 657 228 model elements, Hawk with the OrientDB backend

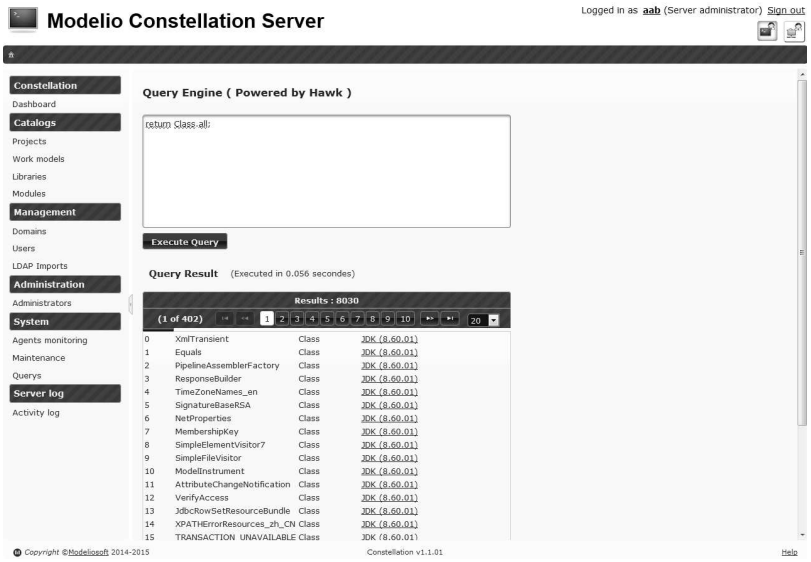

Figure 3: Web-based model query UI in Constellation, powered by Hawk

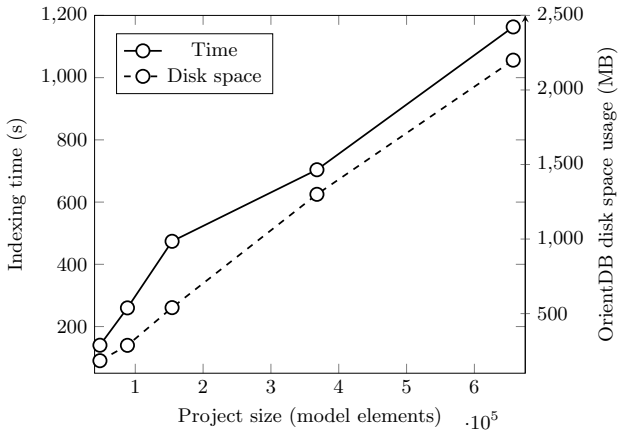

Figure 4: Indexing times and index sizes for the *SyntheticWorkspace* projects (OrientDB backend)

required 1 163s to index and produced a 2.2GB OrientDB database.

Two versions of a model-to-text transformation that produced a list of packages, classes, attributes and operations were developed: the "MT" transformation used the Modelio Jython scripting environment, and the "HT" transformation combined Hawk with EGL using the Hawk Epsilon driver. Within the same execution of Modelio, MT was run 5 times, and within the same execution of Eclipse, HT was run 5 times. Both tools were given the same amount of memory (`-Xmx1g`) to run their transformation. Since the execution times for Modelio largely changed after the second run (as it seems to leave the relevant part of the model in memory), the times for the last 4 runs of MT and HT were displayed separately and averaged together.

Execution times are shown on Figure 5. In general, it can be seen that Hawk on its first run is markedly faster than Modelio on its first run (two orders of magnitude for the largest project), and that both tools are similar in later runs

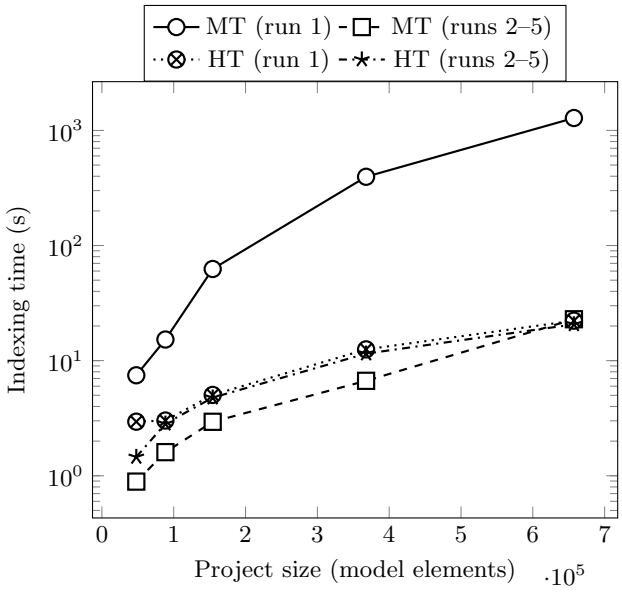

Figure 5: Document generation times by Modelio (MT) and Hawk+EGL (HT)

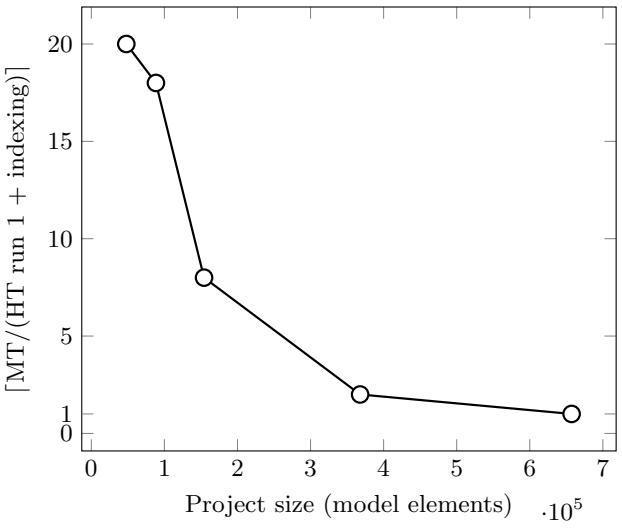

Figure 6: Number of first-time runs of MT required to amortise Hawk (OrientDB backend)

(with a slight edge to Modelio except for the largest project). For the largest project with 657 228 model elements, MT required 1 281s on its first run and 22.89s on average on later runs, and HT required 22s on its first run and 20.66s on its later runs. Even including the 1 163s of indexing, HT would have been faster for this size, and this cost only needs to be paid once per project, instead of once per Modelio execution. Our prior results suggest that using the Neo4j backend would have reduced indexing times even further.

Figure 6 plots how many times MT would have to be run for the first time since Modelio was launched to exceed the total time required by the very first execution of HT, including Hawk indexing time (HIT) and execution (HET). Values quickly drop below 10 and for the largest model element, it is below 1: the cost of using Hawk is immediately amortised on the first run. In any case, the cost itself may not be that important if the indexing process is left up to a constantly running server (e.g. Constellation): in most cases, by the time the developer needs to run their model-to-text transformation, it will have already been indexed.

5. RELATED WORK

The Constellation server into which Hawk was integrated is a versioned model store that divides models into fragments in order to improve scalability. The literature has various other works dealing with model fragmentation and model versioning: in this section we will discuss the most relevant works in these areas.

Regarding model fragmentation, modeling frameworks like EMF offer the capability of physically separating a model into several smaller interconnected model fragments. There are two forms of fragmentation:

- With total fragmentation, each model element is inside its own fragment. This is commonly used when models are to be stored in databases using tools such as Morsa [14] or NeoEMF [5, 7].

- With partial fragmentation, models are broken into fragments according to a certain heuristic, which is usually based on certain types (e.g. "fragment by class"). This is the approach followed by tools such as EMF-Fragments [16] or DSL-Tao [1]. DSL-Tao in particular is based on Fragmenta, a theory of fragmentation within MDE that provides both bottom-up and top-down fragmentation that has been verified and validated through the Isabelle theorem prover.

  This approach is favored by models are stored in traditional file-based version control systems, and it is the approach followed by Modelio as well: each `.exml` file represents a model fragment, which may contain multiple model elements. `.ramc` archives contain a collection of model fragments, which can be processed separately by Hawk.

As for model versioning, there are two options:

- Using a traditional file-based version control system (VCS): ModelCVS [11] stores EMF models and offers a model element versioning abstraction built on top of an SVN[6] VCS.

  This is the approach used by Constellation: it keeps track of file-based models stored in SVN repositories and hosted in HTTP servers. Users generally feel more confident about using a mature VCS that is independent of their tool vendor.

- Using a specialized model store: Eclipse CDO [6] stores model versions in a relational or non-relational database: the default configuration and the most popular one is an H2 database with the *DB Store* object/relational mapping. EMFStore [8] uses a MongoEMF-based backend[7] to implement its versioning model store.

6. CONCLUSIONS AND FUTURE WORK

The present work has shown how Hawk was integrated with the Modelio modelling tool and Constellation enterprise model management and collaboration environment developed by Softeam. This entailed a collaboration between Softeam and the developers of Hawk at the University of York: Softeam provided a Java library that exposed the Modelio metamodels and reused Hawk as a library inside Constellation, and York extended Hawk with the required features to support Modelio models. These extensions have been published under an open source license on Github[8]. Hawk was extended with a clean-room parser for Modelio's `.exml` and `.ramc` file formats, and several new features have been introduced to support Modelio's UID-based references, reusable model libraries and containment representation. The Modelio metamodels have been mapped to Ecore, and Hawk provides the required functionality to expose Modelio models as regular EMF models.

After the integration, several case studies have been conducted to evaluate if Hawk could scale up to Softeam's needs, and which would be the costs and benefits of using Hawk. On a first case study using the Neo4j backend, Hawk could index a collection of 15 Modelio projects containing 1.23M unique model elements in 3958s with 8GB of RAM and 5171s with 2GB of RAM: 1GB of RAM could still be used if derived

[6]http://subversion.apache.org/
[7]https://github.com/BryanHunt/mongo-emf/wiki
[8]https://github.com/mondo-project/mondo-hawk

containment references (only needed for EMF compatibility) were disabled, requiring 3146s.

The next case study required integrating Hawk into Constellation. A new OrientDB backend was developed to enable the redistribution of Hawk within Constellation, and Hawk was extended with the ability to read models stored in HTTP servers. While using OrientDB resulted in a slight performance hit, Hawk's design would allow Softeam to offer a Neo4j-based edition that bundled the Neo4j license.

The final case study compared the Modelio Jython scripting environment against the combination of Hawk (exposed as an Epsilon model) and the Epsilon Generation Language for generating documents from a collection of synthetic models. Hawk showed much stronger performance even with the OrientDB backend, being 2 orders of magnitude faster than Modelio for the largest model. Further calculations showed that the cost of indexing with Hawk could be amortised after generating 20 documents for the smallest model and only 1 document for the largest model.

These studies point to several future lines for both Softeam and the Hawk developers at York. Softeam is currently working on reusing Hawk queries beyond the expert-oriented EOL-based query console, integrating dashboards with useful visualisations about the models stored within the Constellation server. Softeam is also considering adding document generation facilities to Constellation based on Hawk and EGL. The Hawk developers have observed that increasing the memory given to Hawk had diminishing returns (going from 2GB to 4GB was much more noticeable than going from 4GB to 8GB), and plan to study if multiple coordinated Hawk instances could scale up better than increasing the available memory. It is also planned to extend Hawk with transparent support for "on-the-fly" derived features that are not computed and stored in advance, but rather computed on demand. This would make it possible to skip the precomputation of the optional derived containment references required for EMF compatibility, which was costly in terms of time and memory.

7. ACKNOWLEDGMENTS

This research was part supported by the EPSRC, through the Large-Scale Complex IT Systems project (EP/F001096/1) and by the EU, through the MONDO FP7 STREP project (#611125).

8. REFERENCES

[1] N. Amálio, J. de Lara, and E. Guerra. Fragmenta: A theory of fragmentation for MDE. In *Proc. 16th Conf. on Model-Driven Engineering Languages and Systems*, MODELS'15, pages 106–115. IEEE, 2015.

[2] K. Barmpis and D. S. Kolovos. Hawk: towards a scalable model indexing architecture. In *Proc. Workshop on Scalability in Model Driven Engineering*, BigMDE '13, pages 6:1–6:9, New York, NY, USA, June 2013. ACM.

[3] K. Barmpis and D. S. Kolovos. Evaluation of contemporary graph databases for efficient persistence of large-scale models. *Journal of Object Technology*, 13-3:3:1–26, July 2014. DOI 10.5381/jot.2014.13.3.a3.

[4] K. Barmpis, S. Shah, and D. S. Kolovos. Towards incremental updates in large-scale model indexes. In *Proc. 11th European Conference on Modelling Foundations and Applications*, ECMFA'15, pages 35–50, July 2015.

[5] A. Benelallam, A. Gómez, G. Sunyé, M. Tisi, and D. Launay. Neo4EMF, a scalable persistence layer for EMF models. In *Modelling Foundations and Applications*, pages 230–241. Springer, 2014.

[6] Eclipse Connected Data Objects. Available at http://wiki.eclipse.org/CDO, 2016. Last checked: 9 April 2016.

[7] A. Gómez, M. Tisi, G. Sunyé, and J. Cabot. Map-based transparent persistence for very large models. In A. Egyed and I. Schaefer, editors, *Fundamental Approaches to Software Engineering*, volume 9033 of *Lecture Notes in Computer Science*, pages 19–34. Springer Berlin Heidelberg, 2015.

[8] M. Koegel and J. Helming. EMFStore: a model repository for EMF models. In *Proc. 32nd ACM/IEEE International Conference on Software Engineering*, volume 2, pages 307–308. ACM, 2010.

[9] D. Kolovos, R. Paige, and F. Polack. The Epsilon Object Language (EOL). In A. Rensink and J. Warmer, editors, *Model Driven Architecture Foundations and Applications*, volume 4066 of *Lecture Notes in Computer Science*, pages 128–142. Springer Berlin / Heidelberg, 2006. 10.1007/11787044_11.

[10] D. S. Kolovos, R. F. Paige, and F. A. Polack. Scalability: The Holy Grail of Model Driven Engineering. In *Proc. Workshop on Challenges in MDE, collocated with MoDELS '08*, Toulouse, France, 2008.

[11] G. Kramler, G. Kappel, T. Reiter, E. Kapsammer, W. Retschitzegger, and W. Schwinger. Towards a semantic infrastructure supporting model-based tool integration. In *Proc. 2006 International Workshop on Global Integrated Model Management*, GaMMa '06, pages 43–46, New York, NY, USA, 2006. ACM.

[12] P. Mohagheghi, M. Fernandez, J. Martell, M. Fritzsche, and W. Gilani. MDE Adoption in Industry: Challenges and Success Criteria. In *Models in Software Engineering*, volume 5421 of *Lecture Notes in Computer Science*, pages 54–59. Springer, 2009.

[13] A. Mougenot, A. Darrasse, X. Blanc, and M. Soria. Uniform Random Generation of Huge Metamodel Instances. In *Proc. 5th European Conference on Model-Driven Architecture Foundations and Applications*, ECMDA-FA'09, pages 130–145, Berlin, Heidelberg, 2009. Springer-Verlag.

[14] J. E. Pagán, J. S. Cuadrado, and J. G. Molina. Morsa: a scalable approach for persisting and accessing large models. In *Proc. of 14th International Conference on Model Driven Engineering Languages and Systems*, MODELS'11, pages 77–92, Berlin, Heidelberg, 2011. Springer-Verlag.

[15] L. M. Rose, R. F. Paige, D. S. Kolovos, and F. A. C. Polack. The Epsilon Generation Language. In I. Schieferdecker and A. Hartman, editors, *Model Driven Architecture - Foundations and Applications*, number 5095 in Lecture Notes in Computer Science, pages 1–16. Springer Berlin Heidelberg, 2008.

[16] M. Scheidgen and A. Zubow. Map/reduce on emf models. In *Proc. 1st International Workshop on Model-Driven Engineering for High Performance and Cloud Computing*, MDHPCL '12, pages 7:1–7:5, New York, NY, USA, 2012. ACM.