Project Number 611125

D6.8 – Integrated Platform - Final Version

Version 1.0
27 May 2016
Final

Public Distribution

University of York

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

Project Partner Contact Information

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

Contents

Document Control

Version	Status	Date
0.2	First draft	14 April 2016
0.9	Draft for internal review	28 April 2016
1.0	Version for submission	27 May 2016

Executive Summary

This deliverable is a final outcome of Task 6.1 (Platform Integration). It presents the final state of the MONDO platform, organized into cloud and client components, discusses how they interoperate with each other, and presents the final version at M30 of the web-service API of the cloud-based part of the platform.

1 Introduction

The aim of Work Package 6 of MONDO is to guide and synthesise the findings of Work Packages 2-5 into a platform that will enable developers to perform scalable model driven engineering. Following D6.1 which outlined the envisioned components of the platform, D6.2 which presented the initial state of the platform, and D6.3 which presented the interim state of the platform in M24, this deliverable presents the final state of the platform on M30 of the project.

Section 2 presents a general view of the platform and outlines how the components have been designed for integration. From the components, several cloud and client artifacts have been produced: their architecture and design are outlined in Sections 3 and 4, respectively. Section 5 presents the final version of the web-service API of the cloud-based component of the platform. Section 6 concludes this deliverable.

2 Platform overview

As outlined in D6.2 [18], the MONDO platform currently consists of the following components:

1. A framework (EMF-Splitter) for automated development of scalable Eclipse-based editors for Domain-Specific Languages (developed in WP2);
2. Reactive (ReactiveATL) and cloud-based (CloudATL) versions of the widely used ATL model-to-model transformation language (developed in WP3);
3. A new version of the VIATRA model transformation engine, built on a reactive virtual machine architecture supported by the EMF-INCQUERY incremental model query framework (developed in WP3);
4. A framework for online and offline collaborative modelling that supports query-based access control based on INCQUERY queries and VIATRA transformations (developed in WP4);
5. A framework for indexing of heterogeneous models stored in file-based version control repositories such as Git or SVN (Hawk - developed in WP5).

These components are then integrated into five classes of artifacts:

1. Cloud server nodes, which implement web service APIs (see Section 5) that expose the cloud-based tools in MONDO, and may host “golden” and “front” SVN/Git repositories for collaborative modelling (see D4.4 [15]).
2. Cloud worker nodes for CloudATL-based model transformations (see D3.3 [12]).
3. Eclipse workbenches, which include the Eclipse-based tools from MONDO. Some of these tools invoke the cloud API.

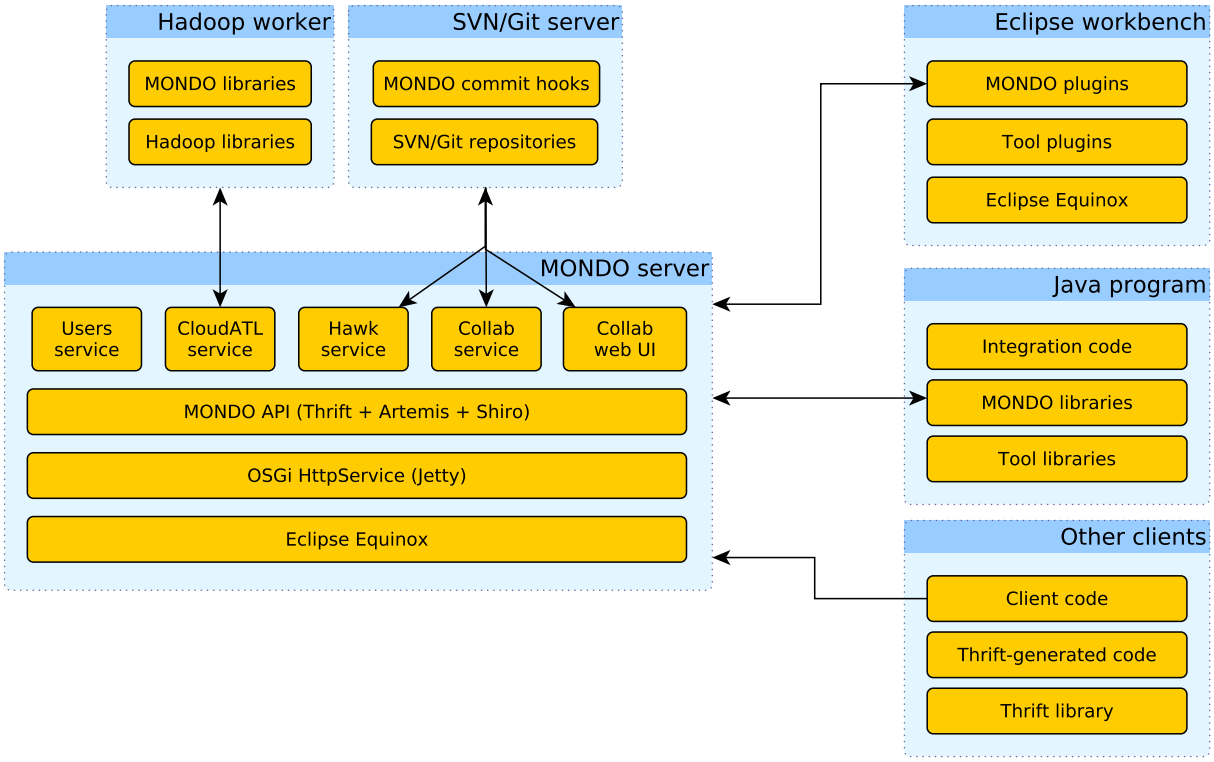

Figure 1: Architecture of the MONDO cloud platform

4. Standalone Java applications, which use the tools from MONDO as additional libraries that may invoke the cloud API.

5. Other clients, which invoke the cloud API or use the web UI of the collaboration framework.

Cloud frontends and Eclipse workbenches use OSGi to integrate the various components in MONDO in a controlled manner, ensuring they do not unintentionally interfere with each other and enabling their independent evolution beyond the lifetime of this project. The cloud API is largely implemented in Apache Thrift [4], an open source library for efficient cross-language Remote Procedure Calls and serialization (Section 5.1 outlines the technology stack used). Figure 1 illustrates how these components are interconnected and organised internally. In the following subsections, we focus on the various ways in which the tools enable these integrations into the platform.

2.1 Heterogeneous Model Indexing Framework

In M24, D5.5 [16] presented the final version of Hawk, a model indexing framework that can monitor large collections of models stored in regular file-based version control systems (e.g. Git or SVN) and maintain a high-performance graph database with a snapshot of the latest version of the models. Hawk can speed up advanced queries using its support for indexed and derived attributes, and the API is designed for managing its configuration and querying the indexed models.

Hawk has been integrated with the rest of the platform in two ways: by integrating new components into Hawk, and by developing compatibility layers so other tools can use Hawk with minimal changes. The industrial partners required several new features after M24 during their use cases, which are listed below.

The Hawk components dedicated to integration are:

- **Client components**: Hawk provides local and remote (Thrift-based) client components to manipulate Hawk indexes with the same UI.
- **Model parsers**: Hawk provides components that can parse Ecore XMI, Modelio XMI/EXML, IFC2x3 STEP/XML, IFC4 STEP/XML and BPMN models. The Ecore XMI component allows Hawk to index models developed using the scalable notations of WP2, produced by the transformations in WP3, or developed collaboratively with the framework in WP4.
  Changes since M24:
  - The Modelio EXML component was newly developed during the industrial evaluation by YORK at the request of SOFTEAM, since Modelio's XMI exports only covered the UML subset of Modelio's capabilities. It parses Modelio models in their native EXML format and is therefore faster and simpler to deploy than the Modelio XMI component, not requiring a local Modelio installation. It also provides a mapping of all the Modelio metamodels to Ecore, making it possible to use Modelio models from any EMF-based tools (such as CloudATL in WP3).
  - The EXML work during SOFTEAM's use case also produced various optimizations in Hawk for the "many small files" case, and introduced new challenges due to its use of unique identifiers for representing references between model elements and its sharing of objects across multiple files. The Hawk core was extended to address these challenges in a reusable way, to simplify the adoption of future formats with similar features.
- **Version control systems**: Hawk provides components for monitoring local folders, Git/SVN version control systems, workspaces and HTTP locations. The Git/SVN version control systems can be one of the "front" repositories maintained by the collaboration framework in WP4 (see D4.4 [15] for details).
  Changes since M24:
  - HTTP location support was added after M24 (during the industrial evaluation) by YORK at the request of SOFTEAM, for the integration of Hawk into their Constellation enterprise model management product.
  - Local folder monitoring was optimised to reduce memory requirements when handling folders with hundreds of thousands of files (as observed during SOFTEAM's use case), by using the MapDB library[1] to replace potentially large memory-backed data structures with file-backed data structures.
- **Backends**: Hawk can use Neo4j or OrientDB for storing its graphs. Neo4j is the *de facto* standard for graph databases, and OrientDB is another high-performance open source graph database with a more permissive license that simplifies redistribution in certain scenarios.
  Changes since M24: an alternate version of the OrientDB backend has been implemented, which supports standalone (i.e., not embedded) OrientDB installations that may reside in a

[1] `http://www.mapdb.org/`

different machine. This makes it possible to take advantage of the replication capabilities of OrientDB's multi-master mode for improved storage reliability, and to query the underlying OrientDB database natively outside Eclipse by using the OrientDB Studio web interface.

The above components allow Hawk to use the results of the other work packages. Hawk also provides abstractions that allow the tools developed in the other work packages to treat a local or a remote Hawk index as a standard model, without requiring any changes. CloudATL can use Hawk indexes as the source of a transformation, IncQuery can query Hawk indexes and update the query results as their contents change, and DSL-tao can use a Hawk index for faster discovery and editing of cross-file references. The design and implementation of these local and remote abstractions in M24 were presented in D5.5 [16] and D5.6 [17], respectively.

After M24, the abstractions were enhanced by YORK at the request of the industrial partners:

- **Local abstraction** (requested by SOFT-MAINT):
  - The abstraction was changed to make it possible to transform models that could not entirely fit into memory at once, by having it reference its model elements through Java "soft references" instead of the default "strong" references. If free memory is running low, the Java garbage collector will reclaim the memory of the least-recently used model elements that have not been changed and are no longer being used by the model transformation. With this change, the memory requirements of the model transformations are reduced to the total size of the largest subset that they need to use at once.
  - Two features from the remote abstraction were brought to the local abstraction in order to be compatible with the model transformations used by SOFT-MAINT. These were a flag to decide if the contents of the model should be split by file or not, and two fields to limit its contents to those of specific indexed repositories and/or files.
- **Remote abstraction** (requested by UNINOVA):
  - A simpler way to define various views on IFC models (e.g., "only plumbing" or "only building structure") without coding EOL queries was requested. Since the views were largely defined in terms of sets of IFC types that should be included or excluded, the remote abstraction was extended to support "effective metamodels": these are collections of rules that include or exclude metamodels, types and/or features within those types. The effective metamodel is provided to the MONDO server, limiting the information that will be sent back to the client. Effective metamodels can be defined by simply checking boxes in a new purpose-built UI.
  - Some of these views had millions of model elements, making it impractical to send the entire view in one response message. The remote abstraction was extended with the ability to perform "paged loads". If a page size is defined, loading will be divided into two stages: the first stage simply identifies the model elements that are to be loaded, and the second stage requests the actual model elements page by page, to reduce the memory requirements on the server. This is faster than using a lazy loading mode, since each request can retrieve one page of model elements instead of a single model element.

2.2 CloudATL

CloudATL (presented in D3.3 [12]) is an extended version of the ATL execution engine that can distribute large-scale transformations over a Hadoop cluster of worker nodes. The engine operates in a transparent manner to the user, who does not need to have any knowledge about distributed programming. The engine shares input data among the slaves, except for the the input model that is equally divided among them. The engine also relies on the MapReduce communication protocols to aggregate the intermediate results and provide the final output model.

CloudATL can be submitted to a standard Hadoop cluster through the Hadoop APIs, but in some cases it might be more convenient to use the console-based tools mentioned in Sections 4 and 4.4. These use a service-oriented API developed by ARMINES and YORK that turns a MONDO server into a frontend for a separate Hadoop cluster. The API provides services to launch, monitor and stop CloudATL transformations on the Hadoop cluster, and CloudATL jobs can read remote Hawk indexes through the above EMF resource abstractions (as discussed in D3.4 [13]). For validation and experimentation purposes, a virtualized Hadoop cluster (based on Docker[2]) has been made publicly available, and automated scripts for launching, stopping, and resizing the cluster are provided[3].

Initial versions of CloudATL ran into various scalability limitations of the standard XMI model serialization format, especially the lack of support for concurrent reads and writes. To solve this issue, a decentralized persistence backend (NeoEMF/HBase [9]) was developed and integrated with CloudATL. NeoEMF/HBase is transparent to standard EMF tools, since it exposes its persistence manager through the usual EMF interfaces. The persistence manager communicates with the underlying data nodes through a persistence driver, and supports a pluggable caching strategy. The design of the persistence manager decouples the high level EMF-based code from the low-level data structures and code accessing the database engine. Maintaining these uniform APIs between the different levels allows including additional functionality on top of the persistence driver by using the decorator pattern, such as different caching levels.

NeoEMF/HBase offers lightweight on-demand loading and efficient garbage collection. Model changes are automatically reflected in the underlying storage, making changes visible to all the clients. In the case of CloudATL, it guarantees that only the model elements needed to perform the partial transformation are loaded by each CloudATL worker. Moreover, NeoEMF/HBase has explicit support for ACID properties. This allows the use of NeoEMF/HBase as a scalable persistence backend for distributed and concurrent model transformations.

2.3 IncQuery and VIATRA

IncQuery [19, 7] is an incremental graph query engine over EMF models. It constructs RETE networks from the query and uses EMF notifications to efficiently update the results of a query after models change. It is part of the new VIATRA3 model transformation framework, which was presented in D3.2 in M15 and implemented eager reactive execution of transformations, in which all changes on the input models were processed as soon as possible.

[2] `https://www.docker.com/`
[3] `https://github.com/atlanmod/hadoop-cluster-docker/`

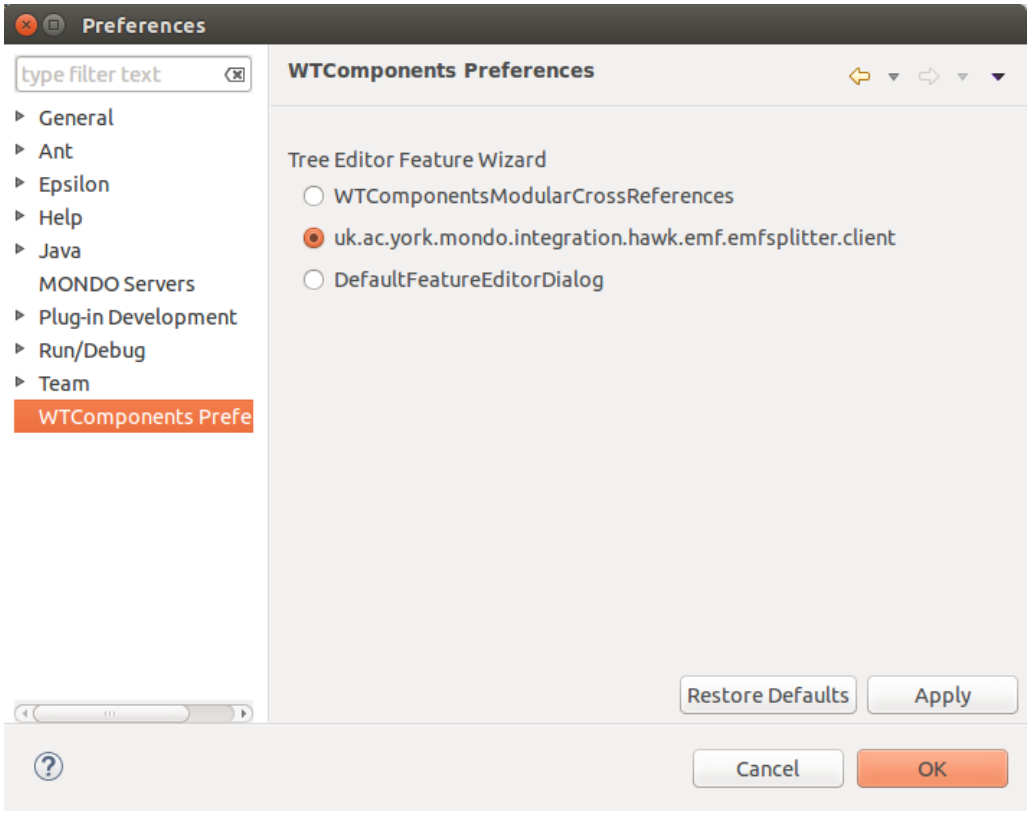

Figure 2: Screenshot of the EMFSplitter preferences page for selecting a component to find candidate values for non-containment references.

BME and YORK collaborated to allow IncQuery to query Hawk indexes directly, and through the EMF resource abstractions of Hawk. After an initial set of tests using the remote EMF resource abstraction with no optimisations on IncQuery's side, an optimised version of IncQuery that could leverage the underlying graph in Hawk was developed. This version of IncQuery can load only the part of the model that is required, accessing all the instances of a type using direct edge traversal instead of having to iterate through the entire model. The optimised version also passes the test suite used for the IncQuery implementation of the BME Train Benchmark. More details are available on D3.4 [13].

2.4 Scalable DSL Modelling Tools

The EMFSplitter scalable DSL modelling framework that was presented in D2.2 [11] can generate domain-specific Eclipse-based model editors that use model fragmentation strategies to improve scalability regarding the size of the models. As planned in D6.2 [18], the final version of Hawk can monitor the local Eclipse workspace and provide efficient local and global queries over the fragmented models. With the local EMF resource implementation, it is also possible to load a model fragment and a Hawk index into the same EMF resource set and use the Hawk index to find candidate values for non-containment references more quickly.

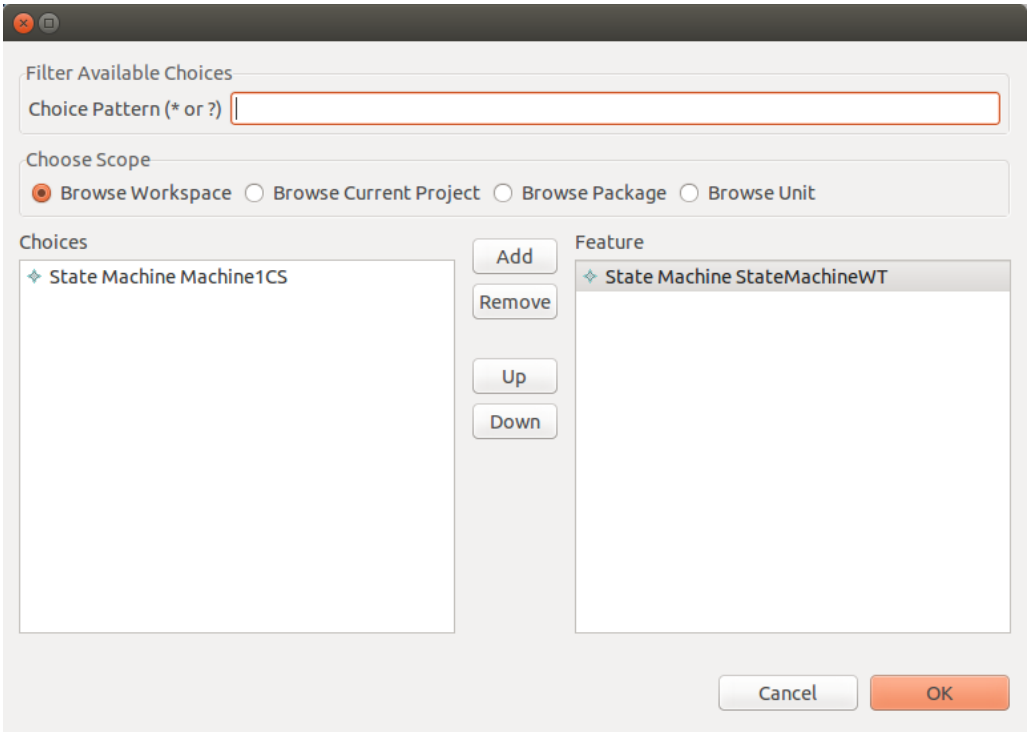

Figure 3: Screenshot of the EMFSplitter editor dialog for non-containment references.

After M24, UAM and YORK collaborated to complete the integration of Hawk into the EMFSplitter user interface. The user only needs to open the Eclipse preferences page produced by EMFSplitter (shown in Figure 2) and tell EMFSplitter to use Hawk to find candidate values for non-containment references. The next time that the EMFSplitter dialog for editing a non-containment reference is opened, the Hawk+EMFSplitter integration will create a Hawk index (backed by OrientDB) that monitors over all the models in the current workspace, and will use it to efficiently find the candidate values. The same index will be reused on later queries, and will be kept up to date automatically by Hawk.

The dialog is shown in Figure 3 and it is the same whether Hawk is used or not. Depending on the scope used (workspace, project, package, or unit), Hawk will be used in a slightly different way. In a first stage, Hawk will be queried to either find all the instances of the *EType* of the reference across the workspace, or find only the instances that are contained within a specific project, package, or unit. In the second (and optional) stage, the retrieved instances can be filtered by project nature.

2.5 Collaboration Framework

In the **offline collaboration** scenario of the MONDO Collaboration Framework, as described in [15], models are stored in one or more *gold* VCS repositories that are not directly accessible by actual users. The VCS server instead hosts a separate *front* repository dedicated to each user, that contains a copy of the gold repository, complete with version history, filtered according to the read access privileges of that user. Users are given access to their dedicated front repository so that they can

read the current or historical contents of the model files (up to their read privileges) and commit their changes (which may be denied based on write permissions).

In their normal day-to-day workflow, users interact with their front repository using standard VCS protocols, off-the-shelf VCS client software, and the repository itself is served by a standard VCS (specifically SVN) server. We provide MONDO-specific tools for special tasks only, such as users discovering their front repository for a given gold repository, or an administrative user managing the setup.

Offline collaboration server components include:

- MONDO-specific plugins called *hook scripts* that extend the behavior of the VCS server. They process VCS commits to enforce read and write access control (and property-based locking) of individual model elements.
- A *daemon* service to perform the actual *lens model transformation* described in [15]. The hook scripts delegate the actual processing of model files to this lens daemon, which is constantly loaded in server memory to improve performance.
- A servlet providing API endpoints (see section 5) for the remote control and management of the offline collaboration server by regular users and administrators.
- Additional *scripts* that can be executed by administrators logged in to the server machine to perform various configuration tasks, such as setting up a new VCS repository.

All these server components need to reside on the same host machine as the VCS server.

Offline collaboration client-side components include:

- A console-based client tool for invoking the remote management API outlined above.
- A graphical UI-based client tool for performing the same more conveniently.
- A graphical UI-based tool for merging versions of a model with conflicting changes (*DSE Merge*, see [14]).

Finally, as an alternative to offline collaboration, MONDO proposes a web-based **online collaborative modeling** tool. Using this solution, users can connect via their web browsers to open, view and edit models stored in the VCS backends. Multiple users can collaborate on the same model simultaneously, enjoying the same access control mechanism that underlies the offline collaboration framework as well. The editor is provided as an Eclipse RAP-based web application, which can optionally be co-deployed with the other server-side components. No special clients are required, as users can connect to this tool via any modern web browser; the access point URL is discoverable using either of the remote clients introduced above. Note that this online modeling tool is inherently dependent on the modeling language, and thus its implementation is only partially generic; the user interface has only been fully implemented for the *Wind Turbine* case study.

The remote client capabilities and their usage will be described later in this document. Server-side setup instructions for the collaboration server are explored in detail in the revised version of [15].

3 MONDO Platform Server

The MONDO integrated platform comprises many networked components, which include a Hadoop cluster, the Git and Subversion model repositories and the client machines. To integrate these components, a server component has been implemented. This server component acts as the front-end node to the Hadoop cluster, communicates with the Git and Subversion repositories and exposes the cloud-based tools to the clients through the API described in Section 5, as well as an online collaborative modeling environment via a web user interface. The following sections present its design and implementation and provide an administration guide.

3.1 Overall structure

The server component is implemented as an Eclipse application, based on the Eclipse Equinox OSGi runtime. Using Eclipse Equinox for the server allows for integrating the Eclipse-based tools with very few changes in their code, while reducing the chances of mutual interference. The OSGi class loading mechanisms ensure that each plugin only "sees" the classes that it declares as dependencies, avoiding common clashes such as requiring different versions of the same Java library or overriding a configuration file with an unexpected copy from another library.

To mitigate the risk of connectivity problems due to enterprise firewalls, the server uses for most of the API the standard HTTP and HTTPS protocols on their default ports (80 and 443). Optionally, the Hawk API can be exposed through raw TCP on port 2080, for increased performance[4]: however, security-conscious environments should leave it disabled as it does not support authentication. The embedded Apache Artemis[5] messaging queue required for remote change notifications in Hawk requires its own port, as it manages its own network connections. By default, this is port 61616. These notifications are made available through two protocols: Artemis Core (a lightweight replacement for the Java Message Service, for Java clients) and STOMP over WebSockets (a cross-language messaging protocol, for web-based clients).

Beyond the plugins of the cloud-based tools and the online collaboration web UI, the server includes plugins that use the standard OSGi *HttpService* facilities to register servlets and filters. Each service in Section 5.2 is implemented as one or more of these servlets. The currently implemented endpoints are listed in Table 1. Most services provide a JSON endpoint, since it is compatible across all languages supported by Thrift and works well with web-based clients. However, since Hawk is performance sensitive (as we might need to encode a large number of model elements in the results of a query), it also provides endpoints with the other Thrift protocols. Binary is the most portable after JSON, and Tuple is the most efficient but is only usable from Java clients. Having all four protocols allows Hawk clients to pick the most efficient protocol that is available for their language. For the sake of completeness, Table 1 also includes the Lens Daemon service, which is not a public API, but an internal communication point used by the offline collaboration framework; it is only accessible locally and only via the Tuple protocol, as it is meant to be invoked only by the version control hooks of the collaboration server, which must run on the same machine as the MONDO server.

[4]Measured to have 20% better throughput by YORK.
[5]`https://activemq.apache.org/artemis/`

Path within server	Service	Thrift protocol
`/thrift/hawk/binary`	Hawk	Binary
`/thrift/hawk/compact`	Hawk	Compact
`/thrift/hawk/json`	Hawk	JSON
`/thrift/hawk/tuple`	Hawk	Tuple
`/thrift/cloudatl`	CloudATL	JSON
`/thrift/users`	Users	JSON
`/thrift/offline-collaboration`	Offline collab. (public)	JSON
`/thrift-local/lens-daemon`	Offline collab. (only local host)	Tuple

Table 1: Available API endpoints

In Eclipse Equinox, the OSGi HttpService is implemented through the Eclipse Jetty [5] webserver. The MONDO server includes additional plugins that customize the default configuration of Jetty, configuring its logging filters and providing *gzip* compression on all HTTP responses. *gzip* compression has a considerable impact on the bandwidth used by Hawk when returning large result sets (more details available on D5.6 [17]).

The web-based user interface for the online part of the collaboration framework developed in WP4 (see Section 2.5 and D4.4 [15]) is integrated through a separate set of Eclipse plugins. These plugins run on top of the Eclipse Remote Application Platform (RAP), making it possible to reuse a considerable amount of code between the MONDO server and the web clients.

3.2 API access control

The industrial partners in MONDO expressed their interest in protecting the API from unaccounted use, as clients would have access to potentially sensitive information. In order to provide this access control, the Apache Shiro library [3] has been integrated transparently as a filter for all incoming requests to the endpoints under `/thrift`. `/thrift-local` endpoints are not password-protected, as they only answer requests from other processes in the machine hosting the MONDO Server.

Apache Shiro protects these `/thrift` endpoints using standard HTTP Basic authentication, which is transparent to Thrift, avoiding the need to pollute the web API with access tokens in every single method. Industrial partners will be instructed to always use the authentication layer in combination with SSL, since HTTP Basic by itself is insecure.

One important advantage of Shiro is its configurability through a single `.ini` file, as shown in Listing 1. Shiro is heavily componentized, making it easy to provide alternative implementations of certain pieces and reuse the default implementations for the rest. In the shown example, all requests to the `/thrift` endpoints go through the default `ssl` and `authcBasic` filters: when enabled, these filters enforce the use of SSL and HTTP Basic authentication respectively. Both filters should be enabled in production environments.

```
[main]
# Objects and their properties are defined here,
# Such as the securityManager, Realms and anything
# else needed to build the SecurityManager

# Note: this should be set to true in production!
ssl.enabled = true

# Toggle to enable/disable authentication completely
authcBasic.enabled = true

# Use MONDO realm
mondoRealm = uk.ac.york.mondo.integration.server.users.servlet.shiro.UsersRealm
securityManager.realms = $mondoRealm

# We're using SHA-512 for passwords, with 10k iterations
credentialsMatcher = org.apache.shiro.authc.credential.Sha512CredentialsMatcher
credentialsMatcher.hashIterations = 10000
mondoRealm.credentialsMatcher = $credentialsMatcher

[urls]

/thrift/** = ssl, authcBasic
```

Listing 1: Example `shiro.ini` file

For the HTTP Basic authentication, the MONDO server component provides its own implementation of a Shiro security realm, which is dedicated to storing and retrieving user details. The MONDO security realm uses an embedded MapDB [10] database to persist these user details, which are managed through the Users service (Section 5.2.4). An embedded database was used in order to prevent end users from having to set up a database just to store a small set of users. MapDB is distributed as a single `.jar` file, making it very simple to integrate. In any case, industrial partners would be able to entirely replace the realm used to a different one if desired by editing `shiro.ini` on their installation.

Passwords for the MONDO realm are stored in a hashed and salted form, using 10 000 iterations of SHA-512 and a random per-password salt.

As for the client side, the command-line based clients accept optional arguments for the required credentials when connecting to the Thrift endpoints. If the password is omitted, the command-line based clients will require it in a separate "silent" prompt that does not show the characters that are typed, preventing shoulder surfing attacks. Due to limitations in the Eclipse graphical user interface, these silent prompts are only available when running the command-line based clients from a proper terminal window and not from the Eclipse "Console" view.

The graphical clients connect to the Thrift endpoints using "lazy" credential providers: if authentication is required, they will attempt to retrieve previously used credentials from the Eclipse secure store and if no such credentials exist, they will show an authentication dialog asking for the username and password to be used. The Eclipse secure storage takes advantage of the access control and encryption capabilities of the underlying operating system as much as possible, and makes it possible to

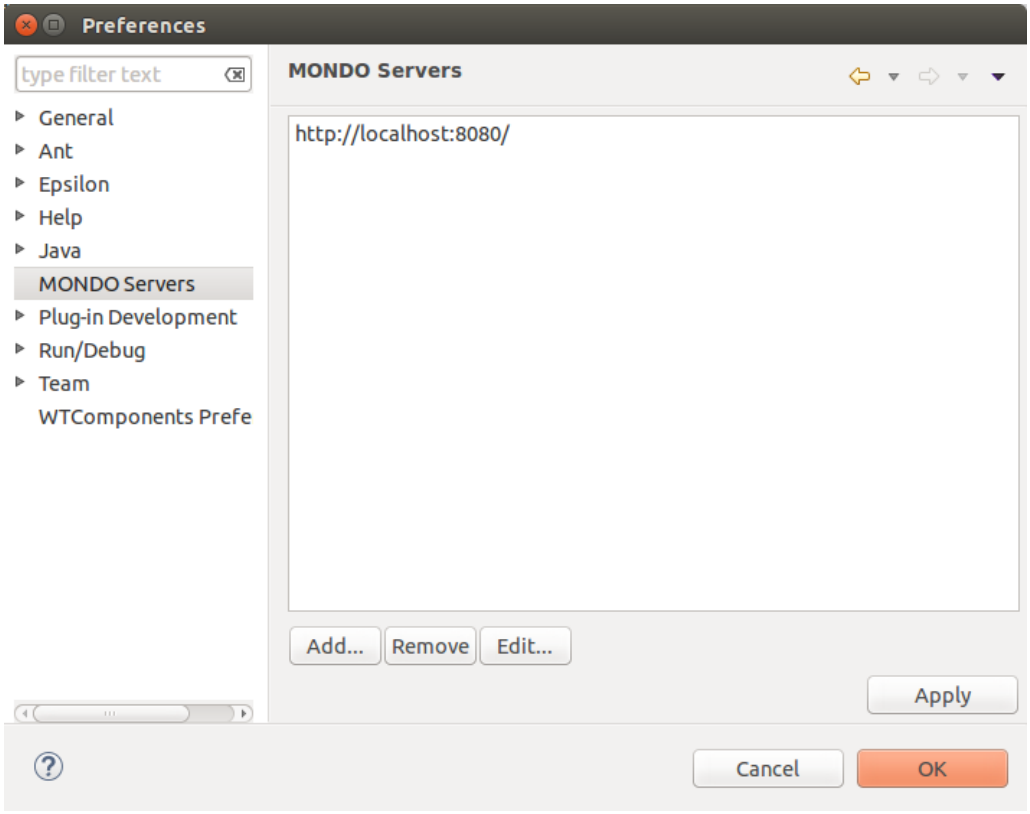

Figure 4: Eclipse preferences page for managing stored credentials to access MONDO servers

store passwords safely and conveniently. These stored MONDO server credentials can be managed from the "MONDO Servers" page in the MONDO workbench product (shown in Figure 4).

Regarding the Artemis messaging queue, it has been secured with the same Shiro realm as the Thrift endpoints. The remote Hawk EMF abstraction (the only component that uses Artemis within the MONDO platform) will connect to Artemis with the same credentials that were used to connect to Thrift, if authentication was required.

3.3 Storage of VCS credentials

The server hosts a copy of the Hawk model indexer, which may need to access remote Git and Subversion repositories. To access password-protected repositories, the server will need to store the proper credentials in a secure way that will not expose them to other users in the same machine.

To achieve this goal, the MONDO server uses the Eclipse secure storage facilities to save the password in an encrypted form. Users need to prepare the secure storage by following these two steps:

1. The secure store must be placed in a place no other program will try to access concurrently. This can be done by editing the `mondo-server.ini` server configuration file and adding `-eclipse.keyring <newline> /path/to/keyringfile`. That path should be only readable by the user running the server, for added security.

2. An encryption password must be set. For Windows and Mac, the available OS integration should be enough. For Linux environments, two lines have to be added at the beginning of the `mondo-server.ini` file, specifying the path to a password file with `-eclipse.password <newline> /path/to/passwordfile`.

   On Linux, creating a password file from 100 bytes of random data that is only readable by the current user can be done with these commands:

   ```
   $ head -c 100 /dev/random | base64 > /path/to/password
   $ chmod 400 /path/to/password
   ```

The MONDO server tests on startup that the secure store has been set properly, warning users if encryption is not available and urging them to revise their setup.

3.4 Integration and distribution

Being an Eclipse product, the server is developed using the standard Eclipse toolset as a collection of plugins. However, these plugins come from many different sources: some are developed by Eclipse, while others are developed by third party developers or the academic partners within MONDO.

To bring all these different plugins together and ensure that all participants in the integration efforts have a consistent experience, the integration plugins and the MONDO server are developed on top of a shared "target platform": a predefined collection of plugins from a set of Eclipse `p2` repositories[6]. The Eclipse Plug-in Development Environment provides the required functionality to ensure that target platforms are consistent (no version conflicts between plugins) and complete (no missing plugins).

The target platform definition is kept up to date by YORK, and integrates Eclipse update sites managed independently by each of the research and user partners. Some of these update sites include the Eclipse Luna, Eclipse Epsilon, Eclipse Orbit and Eclipse Xtext update sites and the Hawk and EMF-IncQuery update site.

However, some of the dependencies in the integration project are not available through Eclipse update sites. For instance, the Artemis libraries need to be fetched using Apache Ivy [2], and some of the dependencies of the ATL/MapReduce framework need to be downloaded from Github. Most dependencies go through the target platform, nevertheless.

Industrial partners are shielded from this complexity, since the server is distributed as an Eclipse application that can be simply unpacked from its `.zip` file and executed. Two classes of server distributions are available for 32- and 64-bit Windows systems, 64-bit Linux systems and 64-bit MacOS X systems: one with the EPL-incompatible components for internal evaluation among MONDO partners, and one without the EPL-incompatible components for public distribution.

[6] `http://www.eclipse.org/equinox/p2/`

4 MONDO Platform Clients

As shown in Figure 1, the MONDO platform can accommodate three different types of clients: Eclipse workbenches with the MONDO tools installed in them, Java programs using the MONDO tools as libraries (e.g., the MONDO SVN/Git commit hooks or the CloudATL Hadoop jobs), and web-based clients using the collaboration UI described in Section 3 and the APIs described in Section 5.

In this section, we present the various types of clients that have been developed on top of the MONDO Platform. The first is a set of console-based tools that allow users to directly invoke the operations in the remote API, and the second is an extension for Hawk that allows managing remote indexes with the same user interface as local ones. Additionally, fully-fledged Eclipse environments with all the MONDO tools and remote clients pre-installed have been created.

4.1 Console-based tools

In order to validate the first versions of the remote API in terms of functionality and fit for purpose, a set of console-based client tools were developed. These tools can be run from an environment without a graphical display, by simply entering text-based commands.

The tools are packaged as bundles for an OSGi environment (such as Eclipse Equinox) and extend the OSGi console with additional commands, by registering an OSGi *CommandProvider* service. An excerpt of the *CommandProvider* for the Hawk API is shown in Listing 2. The *CommandProvider* Java interface requires implementing `getHelp` to integrate into the standard `help` command, and then each `_name` method is made available as "name" (ignoring case). Arguments passed by the user can be obtained from the supplied *CommandInterpreter* object.

```
1  public class CommandProvider implements CommandProvider {
2    // Thrift-generated client stub
3    private Hawk.Client client;
4
5    // Available through the console as "hawkhelp" or "hawkHelp"
6    public Object _hawkHelp(CommandInterpreter intp) {
7      return getHelp();
8    }
9
10   // Available through the console as "hawkconnect" or "hawkConnect"
11   public Object _hawkConnect(CommandInterpreter intp) throws Exception {
12     // ...
13   }
14
15   // Required by the CommandProvider interface
16   @Override
17   public String getHelp() {
18     return /* ... */;
19   }
20 }
```

Listing 2: Excerpt of the Hawk *CommandProvider*

These console-based clients are currently available:

- `fr.inria.atlanmod.mondo.integration.cloudatl.cli` for the CloudATL API in Section 5.2.1, providing the commands in Table 2.
- `uk.ac.york.mondo.integration.hawk.cli` for the Hawk API in Section 5.2.2, providing the commands in Table 3.
- `uk.ac.york.mondo.integration.server.users.cli` for the Users API in Section 5.2.4, providing the commands in Table 4.
- `org.mondo.collaboration.offline.management.cli` for the Offline Collaboration API in Section 5.2.3, providing the commands in Table 5.

The console-based clients have been packaged into a single Eclipse product that can be unpackaged and run from the command line, without requiring a graphical environment. This product is intended to be used as a client in cloud deployments, where graphical environments are usually not available.

Table 2: Console commands for the CloudATL API: names are case-insensitive, <> are required arguments, [] are optional arguments.

Command	Description
cloudAtlHelp	Lists all the available commands for CloudATL
cloudAtlConnect <url> [username] [password]	Connects to a Thrift endpoint
cloudAtlDisconnect	Disconnects from the current Thrift endpoint
cloudAtlLaunch <transformation> <source-mm> <target-mm> <input> <output>	Launches an ATL transformation (all arguments are hdfs:// or hawk+http:// URLs)
cloudAtlStatus <id>	Shows the status of the specified transformation job
cloudAtlList	Lists all the transformation jobs tracked by this endpoint
cloudAtlKill <id>	Kills the transformation identified by <id>

Table 3: Console commands for the Hawk API: names are case-insensitive, <> are required arguments, [] are optional arguments.

Command	Description
hawkHelp	Lists all the available commands for Hawk
hawkConnect <url> [username] [password]	Connects to a Thrift endpoint (guesses the protocol from the URL)
hawkDisconnect	Disconnects from the current Thrift endpoint

Continued on next page

Table 3 – continued from previous page

Command	Description
hawkAddInstance <name> <backend> [minDelay] [maxDelay|0]	Adds an instance with the provided name (if maxDelay = 0, periodic updates are disabled)
hawkListBackends	Lists the available Hawk backends
hawkListInstances	Lists the available Hawk instances
hawkRemoveInstance <name>	Removes an instance with the provided name, if it exists
hawkSelectInstance <name>	Selects the instance with the provided name
hawkStartInstance <name>	Starts the instance with the provided name
hawkStopInstance <name>	Stops the instance with the provided name
hawkSyncInstance <name> [waitForSync:true|false]	Requests an immediate sync on the instance with the provided name
hawkListMetamodels	Lists all registered metamodels in this instance
hawkRegisterMetamodel <files...>	Registers one or more metamodels
hawkUnregisterMetamodel <uri>	Unregisters the metamodel with the specified URI
hawkAddRepository <url> <type> [user] [pwd]	Adds a repository
hawkListFiles <url> [filepatterns...]	Lists files within a repository
hawkListRepositories	Lists all registered metamodels in this instance
hawkListRepositoryTypes	Lists available repository types
hawkRemoveRepository <url>	Removes the repository with the specified URL
hawkUpdateRepositoryCredentials <url> <user> <pwd>	Changes the user/password used to monitor a repository
hawkGetModel <repo> [filepatterns...]	Returns all the model elements of the specified files within the repo
hawkGetRoots <repo> [filepatterns...]	Returns only the root model elements of the specified files within the repo
hawkListQueryLanguages	Lists all available query languages
hawkQuery <query> <language> [repo] [files]	Queries the index
hawkResolveProxies <ids...>	Retrieves model elements by ID
hawkAddDerivedAttribute <mmURI> <mmType> <name> <type> <lang> <expr> [many|ordered|unique]*	Adds a derived attribute
hawkListDerivedAttributes	Lists all available derived attributes
hawkRemoveDerivedAttribute <mmURI> <mmType> <name>	Removes a derived attribute, if it exists
hawkAddIndexedAttribute <mmURI> <mmType> <name>	Adds an indexed attribute
hawkListIndexedAttributes	Lists all available indexed attributes
hawkRemoveIndexedAttribute <mmURI> <mmType> <name>	Removes an indexed attribute, if it exists

Continued on next page

Table 3 – continued from previous page

Command	Description
hawkWatchModelChanges [default|temporary|durable] [client ID] [repo] [files...]	Watches an Artemis message queue with detected model changes

Table 4: Console commands for the Users API: names are case-insensitive, <> are required arguments, [] are optional arguments.

Command	Description
usersHelp	Lists all the available commands for Users
usersConnect <url> [username] [password]	Connects to a Thrift endpoint
usersDisconnect	Disconnects from the current Thrift endpoint
usersAdd <username> <realname> <isAdmin: true|false> [password]	Adds the user to the database
usersUpdateProfile <username> <realname> <isAdmin: true|false>	Changes the personal information of a user
usersUpdatePassword <username> [password]	Changes the password of a user
usersRemove <username>	Removes a user
usersCheck <username> [password]	Validates credentials

Table 5: Console commands for the offline collaboration API: names are case-insensitive, <> are required arguments, [] are optional arguments.

Command	Description
collabHelp	Lists all the available commands for the Offline Collaboration Management Interface.
collabHelp <mgmtURL> <user> <pass> <goldRepoURL>	Regenerates all front repositories based on the gold repository.
collabGetMyFront <mgmtURL> <user> <pass> <goldRepoURL>	Retrieves the front repository URL for the given user.
collabListGolds <mgmtURL> <user> <pass>	Lists the URLs of all gold repositories maintained by the given MONDO server.

Continued on next page

Table 5 – continued from previous page

Command	Description
collabGetOnline <mgmtURL> <user> <pass>	Retrieves the URL of the login screen of the online collaboration tool.

4.2 Graphical clients for remote Hawk indexes

The Hawk model indexer in WP5 provides its own Eclipse-based graphical user interface for managing local indexes. Within WP6, Hawk was extended with additional plugins that enable it to query and manage remote indexes through the Thrift API in Section 5.2.2:

- `uk.ac.york.mondo.integration.hawk.remote.thrift` contributes a new type of *IHawkFactory* whose *IHawk* instances operate over the remote API. See Section 4.2.1 for details.
- `uk.ac.york.mondo.integration.hawk.emf` extends Hawk with an EMF resource abstraction over remote indexes. See D5.6 for details about this abstraction.
- `uk.ac.york.mondo.integration.hawk.emf.dt` provides an Eclipse-based editor for the `.hawkmodel` model access descriptors used by the remote EMF resource abstraction. See Section 4.2.2 for details.

4.2.1 IHawkFactory for remote indexes

When creating a Hawk instance for the first time (using the dialog shown in Figure 5), users can specify which factory will be used. The name of the selected factory will be saved into the configuration of the instance, allowing Hawk to recreate the instance in later executions without asking again. Hawk provides a default *LocalHawkFactory* whose *LocalHawk* instances operate in the current Java virtual machine. Users can also specify which Hawk components should be enabled: for instance, if SOFTEAM does not need to index IFC models, they can leave the IFC components disabled to reduce the size of the graph database.

A factory can also be used to "import" instances that already exist but Hawk does not know about. For the local case, these would be instances that were previously removed from Eclipse but whose folders were not deleted. The Eclipse import dialog is shown in Figure 6.

WP6 provides a plugin that contributes a new indexer factory, *ThriftRemoteHawkFactory*, which produces *ThriftRemoteHawk* instances that use *ThriftRemoteModelIndexer* indexers. When creating a new instance, the factory will use the *createInstance* operation to add the instance to the server. When used to "import", the remote factory retrieves the list of Hawk instances available on the server through the *listInstances* operation of the Thrift API. Management actions (such as starting or stopping the instance) and their results are likewise translated between the user interface and the Thrift API.

The Hawk user interface provides live updates on the current state of each indexer, with short status messages and an indication of whether the indexer is stopped, running or updating. Management

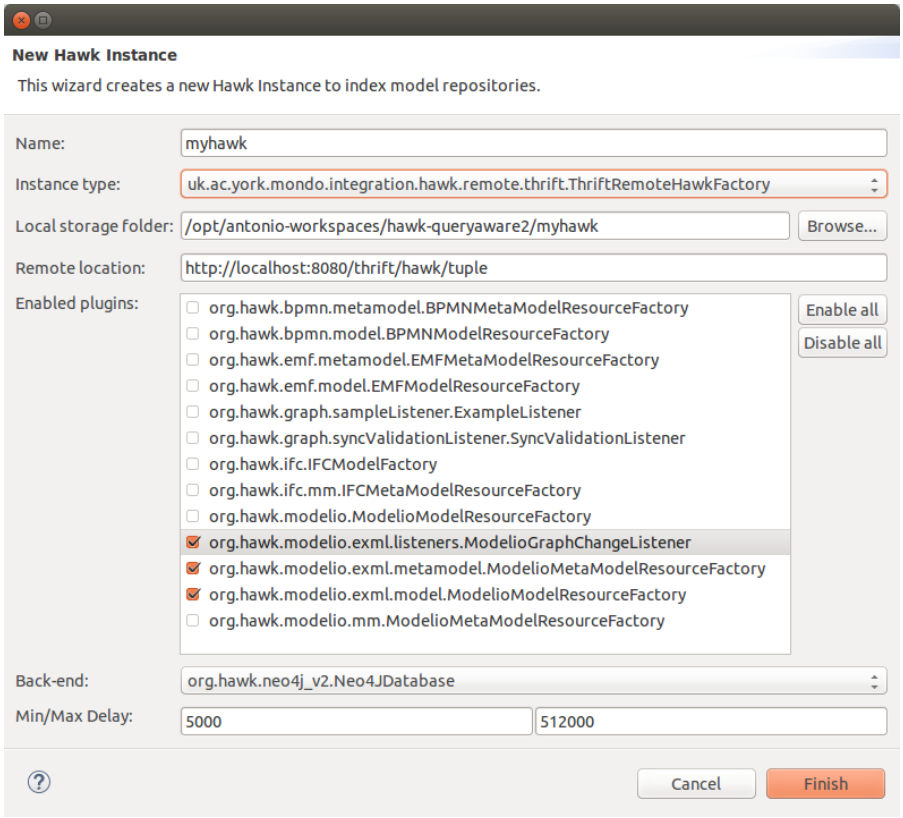

Figure 5: Creation of a new remote Hawk instance

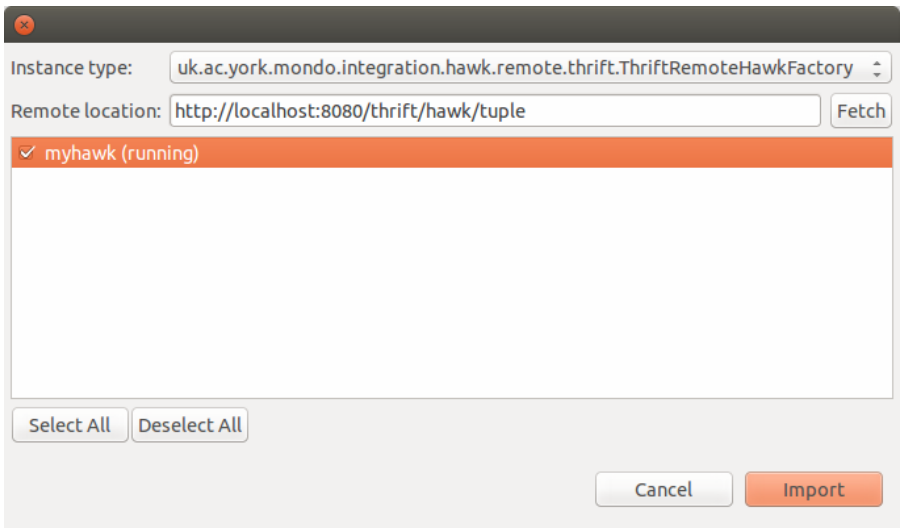

Figure 6: Importing an existing remote Hawk instance

actions and queries are disabled during an update, to prevent data consistency issues. This was simple to implement in the local case with regular Java method calls, but the remote case required that the server propagated these live updates to all its clients. The simplest option would have been to make the clients repeatedly poll the server for its latest status, but this would have severely impacted

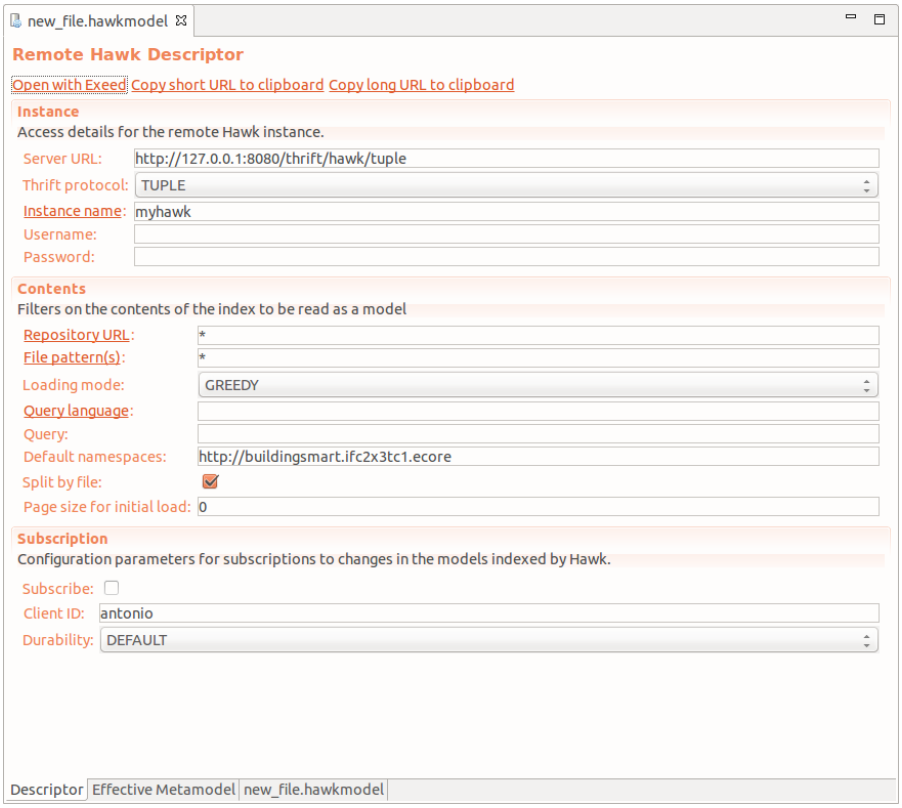

Figure 7: Eclipse-based editor for remote Hawk model access descriptors: main tab

the scalability of the platform as the number of clients increased, and it would not have guaranteed that the connected clients were always kept up to date. Instead, the server itself sends the live updates directly to the clients through temporary Apache Artemis queues (which are not persisted to disk), as Thrift *HawkStateEvent* structures.

All these aspects are transparent to the user: the only difference is selecting the appropriate "Instance type" in the new instance or import dialogs and entering the URL to the Hawk Thrift endpoint. If the remote instance type is chosen, Hawk will only list the Hawk components that are installed in the server, which may differ from those installed in the client.

4.2.2 Editor for remote model access descriptors

As described in D5.6, the remote EMF resource abstractions are highly configurable according to the needs of the user. These configuration options are provided through a model access descriptor, which is serialized as a `.hawkmodel` file containing a set of name-value pairs.

To simplify the creation and maintenance of these `.hawkmodel` files, an Eclipse-based editor has been provided in WP6. The editor is divided into three tabs: a form-based tab for editing most aspects of the descriptor in a controlled manner, another form-based tab for editing the effective metamodel to limit the contents of the model, and a text-based tab for editing the descriptor directly.

The main form-based tab (shown in Figure 7) is divided into three sections:

- The "Instance" section provides connection details for the remote Hawk instance: the URL of the Thrift endpoint, the Thrift protocol to use (more details in D5.6) and the name of the Hawk instance within the server. "Instance name" can be clicked to open a selection dialog with all the available instances.
  The "Username" and "Password" fields only need to be filled in if using the `.hawkmodel` file outside Eclipse. When using the `.hawkmodel` inside Eclipse, the remote EMF abstraction will fall back on the credentials stored in the Eclipse secure store if needed (see the Section 3.2 on access control in the MONDO platform).
- The "Contents" section allows for filtering the contents of the Hawk index to be read and changing how they should be loaded:
  - By default, the entire index is retrieved (repository URL is "*", file pattern is "*" and no query is used). The "Repository URL", "File pattern(s)" and "Query language" labels can be clicked to open selection dialogs with the appropriate options.
  - The default loading mode is "GREEDY" (send the entire contents of the model in one message), but various lazy loading modes are available. Readers are encouraged to read D5.6 for details.
  - Finally, the contents of the index can be split over the different source files or not. While splitting by file is useful for browsing and is required by the EMF-Splitter integration, some EMF-based tools (such as CloudATL) are not compatible with it.
  - The "Default namespaces" field (added at the request of UNINOVA after M24) makes it possible to resolve ambiguous type names. For instance, both the IFC2x3 and the IFC4 metamodels have a type called *IfcActor*. Without this field, the query would need to specify which one of the two metamodels should be used on every reference to *IfcActor*, which is unwieldy and prone to mistakes. With this field filled in as in Figure 7, the query will be told to resolve ambiguous type references to those of the IFC2x3 metamodel.
  - The "Page size for initial load" field (also added at the request of UNINOVA after M24) can be set to a value other than 0, indicating that during the initial load of the model, its contents should not be sent in one response message, but rather divided into "pages" of a certain size. UNINOVA has observed that a GREEDY loading mode with an adequate page size can be faster to load than a lazy loading mode, while still keeping server memory and bandwidth requirements under control.
- The "Subscription" section allows users to enable live updates in the opened model through the *watchGraphChanges* operation and an Apache Artemis queue of a certain durability (more information about Artemis is available on Section 5.1.2). In order to allow the server to recognize users that reconnect after a connection loss, a unique client ID should be provided.

The effective metamodel editor tab (shown in Figure 8) presents a table that lists all the metamodels registered in the selected remote Hawk instance, their types, and their features (called "slots" by the Hawk API). It is structured as a tree with three levels, with the metamodels at the root level, the types inside the metamodels, and their slots inside the types.

The implicit default is that all metamodels are completely included, but users can manually include or exclude certain metamodels, types or slots within the types. This can be done through drop-down selection lists on the "State" column of the table, or through the buttons on the right of the table:

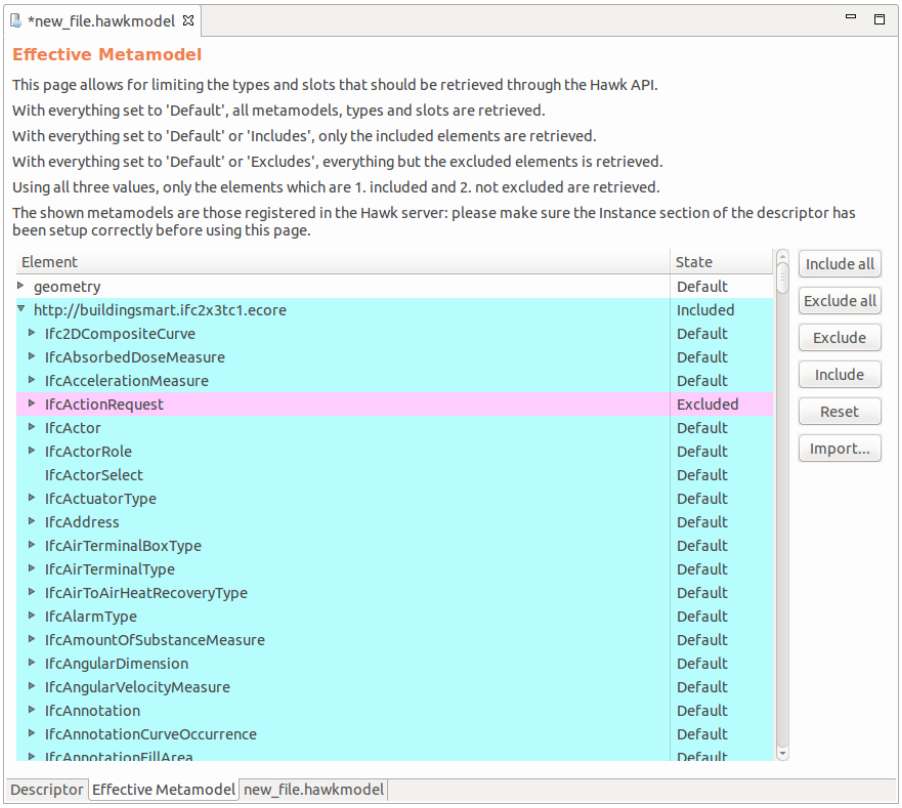

Figure 8: Eclipse-based editor for remote Hawk model access descriptors: effective metamodel tab

- "Include all" resets the entire table to the default state of implicitly including everything.
- "Exclude all" resets the entire table to excluding all metamodels.
- "Exclude" and "Include" only change the state of the currently selected element.
- "Reset" return the currently selected element to the "Default" state.

The effective metamodel is saved as part of the `.hawkmodel` file, and uses both inclusion and exclusion rules to remain as compact as possible (as it will need to be sent over the network). The rules work as follows:

- A metamodel is included if it is "Included", or if it has the "Default" state and no metamodels are explicitly "Included".
- A type is included if it is not "Excluded" and its metamodel is included.
- A slot is included if it is not "Excluded" and its type is included.

4.3 Graphical client for remote Collaboration Servers

As discussed in Section 2.5, users primarily interact with the MONDO Collaboration Server via standard VCS clients and web browsers. For performing additional tasks not covered by these channels, the Eclipse-based MONDO client product contains an additional client UI. Note that setting up and configuring such a collaboration server is not possible through this interface; it requires remote shell access to the server itself and is described in [15].

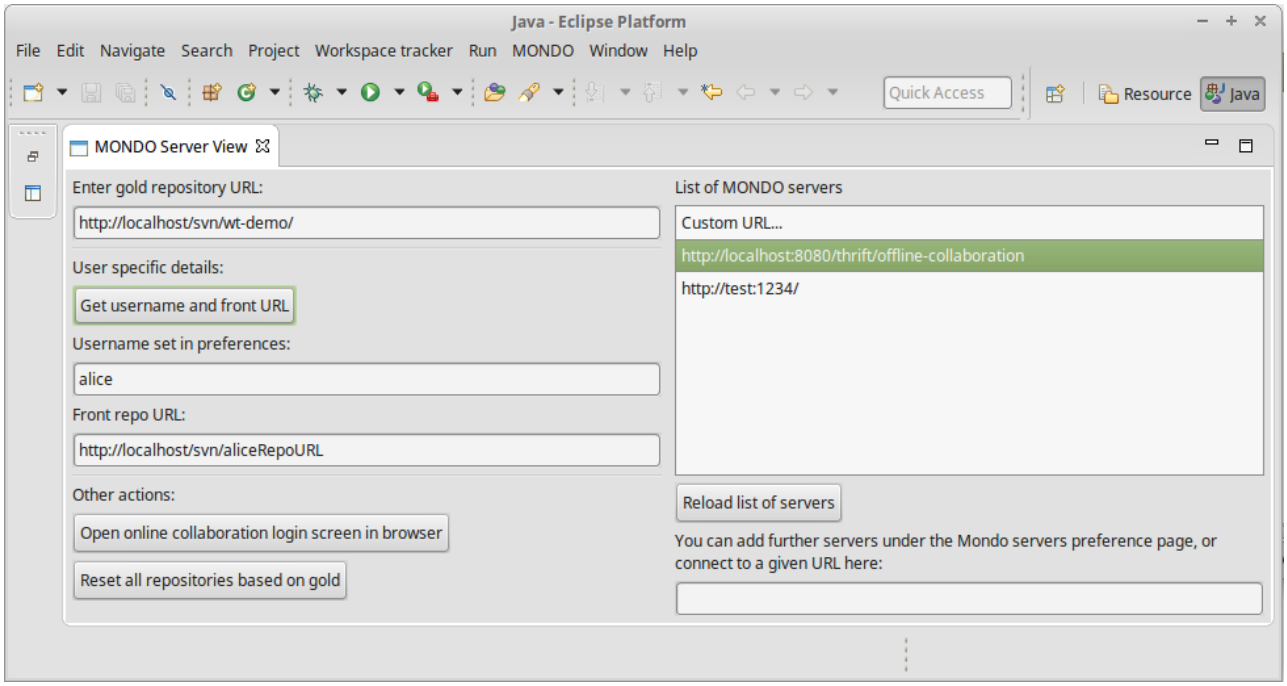

Figure 9: Remote Client UI for Collaboration Servers

As shown before in Figure 4, users can maintain a list of known MONDO Servers and their associated credentials in an Eclipse Preferences page. The remote client UI for Collaboration consists of an Eclipse View (shown by Figure 9), where the following actions are available for a selected server:

- For the given collaboration server, and a (gold) VCS model repository hosted on that server (which is not directly accessible), one can request the SVN URL for the user-specific front repository that they can access.
- For the given collaboration server, one can open the online collaboration interface in a web browser.
- In case the front VCS repositories of the offline collaboration server are somehow corrupted (e.g. due to power outage), it possible to remotely instruct a collaboration server to reset / reconstruct the front repositories of a given gold repository, returning the system to a healthy state. Privileges to perform this operation are typically restricted to a subset of users (using the API access control outlined in Section 3.2).

4.4 Integrated Eclipse environment

Due to the diversity of the tools developed within the MONDO project, the first-time setup for users would require installing many plugins into their Eclipse IDEs, having to deal repeatedly with possible version mismatches, unexpected interactions between tools and undesirable delays.

To avoid these issues and accelerate the training and initial adoption of the tools by the industrial evaluators, special distributions of Eclipse with all the Eclipse-based MONDO tools have been created. These include:

- DSL-tao and EMF-Splitter, from WP2.
- VIATRA3, CloudATL and ReactiveATL, from WP3.
- DSE Merge, MONDO Collaboration and tools for the Wind Turbine case study, from WP4.
- Hawk (WP5).
- Graphical clients for the remote APIs, for most users.
- Console-based clients for the remote APIs, for advanced users. See Section 4.1 for details.

With these distributions, the industrial evaluators will be able to test the MONDO tools by unpacking the appropriate `.zip` for their operating system and architecture and running a single executable. The Eclipse distributions come preconfigured with references to the various Eclipse update sites maintained by the academic partners, allowing industrial partners to upgrade the MONDO tools through the regular Eclipse UI for managing software updates.

The Eclipse distributions are built through Eclipse Tycho [6] and are based on the latest stable release of Eclipse at the moment, Mars SR.1. All distributions are based on the same set of Eclipse plug-ins or "target platform" from a collection of update sites managed by the Eclipse Foundation and the MONDO academic partners. Two classes of distributions are available for 32- and 64-bit Windows systems, 64-bit Linux systems and 64-bit MacOS X systems, in the same way as the server distributions: one with the EPL-incompatible components for internal evaluation among MONDO partners, and one without the EPL-incompatible components for public distribution.

5 MONDO Platform API

In this section we present the services, service operations and entities of the API of MONDO's integrated cloud platform. Section 5.1 introduces some of the technologies used to implement the API. Section 5.2 lists the services that comprise the API. For each service, its constituent operations and their parameters are presented. The reader should note that the description of some of these operations is not self-contained (e.g. services related to *indexed* and *derived* attributes in Hawk); where this is the case, we provide references to deliverables that provide an extended discussion on the concepts behind these services. Section 5.3 then lists the structured entities, instances of which can act as input/output of the service operations discussed in Section 5.2, and discusses their roles and fields. Section 5.4 lists the enumeration types that are used by the previous services and entities, and Section 5.5 describes the exceptions that can be sent from services when they fail.

5.1 Technology stack

The remote API is based on several state-of-the-art technologies for serialization, RPC and message queuing. To aid readers in understanding some of the technical details underlying the API, in this section we will provide short introductions for each of them.

5.1.1 Serialization and RPC: Apache Thrift

As indicated in D6.1, Apache Thrift was selected as the common communications framework for the API, as it was more scalable than competing WSDL/XML approaches and could easily generate server and client stubs in multiple programming languages from the same Thrift interface descriptor file. The Thrift web service/RPC architecture is illustrated in Figure 10, and is configured for the MONDO platform as follows:

- Client App: these are the clients manually written on top of the remote API, as described in Section 4. All MONDO clients connect to the API using the methods of the *APIUtils* class provided by the shared `uk.ac.york.mondo.integration.api` plugin, shielding them from the details of how to connect to Thrift and how to provide adequate credentials.
- Service Handler: these are the manually written subclasses of the Thrift *TServlet* class that implement the *Iface* interfaces generated by Thrift. More information about the servlets is available in Section 3.
- Client and Server Stubs: this is the code that Thrift generates from the interface description language file for the clients. The IDL file itself was generated from an annotated Ecore metamodel using Ecore2Thrift [8], a set of model-to-text transformations developed as part of WP6 and published under the Eclipse Public License.
- User Types: the service stubs are defined using standard Thrift scalar and collection types, which are implemented by Thrift as libraries for each of the supported languages (e.g. Java). The MONDO platform is using the 0.9.3 version of the Java library for Thrift.
- Protocols: Thrift protocols encode the predefined user types as a sequence of bytes. The MONDO platform uses the four most common protocols (JSON, binary, compact and tuple): JSON is the most compatible across languages and least efficient, and tuple is the least compatible but most efficient (only targets Java as of 0.9.3).
  As indicated in 3.1, the Hawk services are available in all four protocols as they are critical to the performance of the MONDO platform. The Users, OfflineCollaboration and CloudATL APIs only use the JSON protocol, as they do not require the same level of performance (their messages are always small).
- Transports: transports move the bytes across devices, and server transports in particular receive remote connections and turn them into regular transports. The MONDO platform implicitly uses HTTP connection-based transports through the Thrift *TServlet* class, and can also use raw TCP-based transports through the Thrift *TServerSocket* class.

Beyond its application as a web service implementation framework, Apache Thrift has been used as a serialization library for messages sent through Apache Artemis queues. More details are provided in Section 5.1.2.

5.1.2 Message queues: Apache Artemis

Apache Thrift allows for easily implementing request-response and request-only communication schemes, but it does not provide any facilities for publish-subscribe communications, in which a client subscribes to updates sent by the server. In this case, it is the server that initiates the sending of a message, not the client.

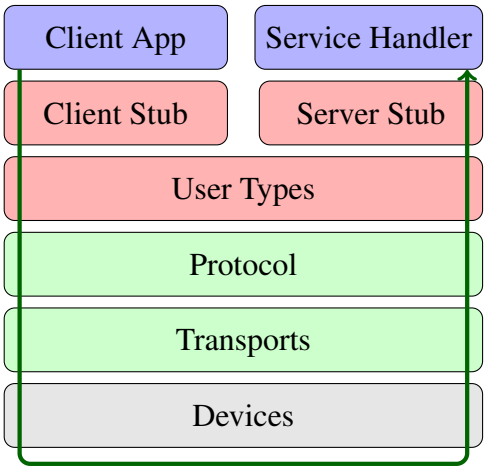

Figure 10: Thrift RPC architecture, based on [1]

This type of communication was found to be necessary while implementing the *watchModelChanges* service operation in the Hawk API, which had to return a "stream of change events". Thrift did not provide any alternatives for sending a "stream" of messages from the server to the client, so an alternative solution for asynchronous messaging had to be found.

If possible, the solution had to be Java-based, so it could be embedded directly in the same Java virtual machine as the MONDO server and simplify deployment while reducing the overhead of producing messages to a few method calls. Several messaging systems were considered:

- Apache ActiveMQ is most mature and feature-rich of all, but it was found to be rather heavyweight and cumbersome to embed within a Java application, due to its long development history as a standalone application.
- Apache Apollo is a more lightweight rewrite of ActiveMQ, but after reviewing the project's source code repository and interacting with the community, it was found to be mostly inactive.
- Finally, Apache Artemis (based on HornetQ[7]) was selected, as it is currently in active development, it is designed to be simple to embed and it is poised to receive the higher core performance of HornetQ and the advanced features of ActiveMQ[8].

Since Apache Artemis is not OSGi-enabled yet, all its libraries (24 `.jar` files in total, including transitive dependencies) have been combined into a single OSGi bundle. These libraries are downloaded automatically from the Maven Central repository hosted by Apache, using the Apache Ivy dependency manager tool. To avoid interfering with other plugins, the OSGi bundle only exports the Artemis public API and core packages and two internal packages of its own:

- The first internal package exports *Consumer*, a utility class that abstracts away the details of implementing a consumer for an Artemis queue. A single Artemis address can feed several queues: normally, producers send messages to an address, which then copies the message to all the queues associated with that address. To readers familiar with JMS: an Artemis address is very similar to a JMS topic.

[7] `http://activemq.apache.org/apache-activemq-board-report-201504-april.html`
[8] `http://hornetq.blogspot.co.uk/2015/06/hornetq-apache-donation-and-apache.html`

- The second internal package exports *Server*, a utility class that abstracts away the details of embedding an Artemis server into a Java application. Some of these details include setting the various storage directories using OSGi data directories, configuring disk paging for large message queues or fine tuning redelivery options, for instance.
  The embedded Artemis servers are configured to enable the Artemis Core (more lightweight than JMS) and STOMP over WebSockets (compatible with web clients) protocols on the standard Artemis port (61616).

Apache Artemis only provides very basic functionality for storing data within messages: strings or byte sequences can be sent, and key-value pairs of strings can be attached to them, but it does not implement any advanced data structures or forwards/backwards compatibility mechanisms between old and new clients and servers.

To cover for this, Thrift has been combined with Artemis. The MONDO platform serializes Thrift data types into sequences of bytes, which are then sent as Artemis messages and deserialized back into Thrift data types at the client side. A specialized Thrift transport has been implemented that reads and writes bytes from ActiveMQ message body buffers (*ActiveMQBufferTransport*). This combination implements the desired publish-subscribe communication scheme while ensuring that messages are interpreted in a well-defined manner across clients and servers written in different programming languages.

Regarding the integration of Apache Artemis queues into the API, the general approach for has been as follows:

1. The client invokes the appropriate *watch** operation indicating that they wish to subscribe to events of a certain type.

2. The server then ensures that an Artemis message producer has been set up to send the relevant events to an Artemis address, and returns a Thrift *Subscription* structure with the connection details to an Artemis queue mapped to the address and the client.

   Artemis producers for Hawk model changes are set up only when at least one client has invoked *watchModelChanges*, as they incur in some memory and processing overhead on updates. However, Artemis producers for Hawk state changes are set up as soon as the indexer is loaded, since their overhead is minimal.

3. The client registers itself as a consumer of the provided *Subscription* using the *APIUtils.connectToArtemis* method of the shared API plugin, which returns a *Consumer* object. This object is used to open an Artemis session and set up a *MessageHandler* that will react to the received messages as they arrive. As an example, an excerpt of the code that listens to the JSON-encoded Hawk state changes is shown in Listing 3.

```
1 protected void connectToArtemis() {
2   // ...
3   Subscription subState = client.watchStateChanges(name);
4   Consumer c = APIUtils.connectToArtemis(subState, SubscriptionDurability.TEMPORARY);
5   c.openSession();
6   c.processChangesAsync(new StatePropagationConsumer(...));
```

```
7     //...
8   }
9
10  class StatePropagationConsumer implements MessageHandler {
11    //...
12    @Override
13    public void onMessage(ClientMessage message) {
14      final TProtocol proto = ThriftProtocol.JSON.getProtocolFactory()
15        .getProtocol(new ActiveMQBufferTransport(message.getBodyBuffer()));
16      final HawkStateEvent change = new HawkStateEvent();
17      change.read(proto);
18      //... process 'change' ...
19    }
20    //...
21  }
```

Listing 3: Excerpt of the code used to subscribe to Hawk state changes through Artemis

5.2 Services

The present section lists the services offered by the current version of the MONDO platform. For each service, the available operations are presented, along with the parameters they take, their return type and the exceptions they can throw. Readers familiar with Thrift may notice that some types inherit from others, which is not supported by Thrift itself: this is implemented through the Ecore2Thrift [8] model-to-text transformation developed in WP6, which takes into account inherited fields when transforming an *EClass* into a Thrift data type.

In addition to their explicitly listed exceptions, operations requiring authentication may produce replies with HTTP 4xx status codes if the user fails to provide valid credentials or does not have the required permissions. In its current version, the MONDO platform assumes that authenticated users will have the necessary permissions.

5.2.1 CloudATL

The following service operations expose the capabilities of the cloud-enabled version of the ATL transformation language which is presented in D3.3 [12].

Table 6: Operation CloudATL.launch

	Name	**Type**	**Documentation**
Operation	launch	string	Invokes a cloud-based transformation in a batch non-blocking mode. Returns a token that can be used to check the status of the transformation.
Parameters			
	transformation	string	The ATL source-code of the transformation.

	source	ModelSpec	The input models of the transformation.
	target	ModelSpec	The target models of the transformation.
Exceptions			
	Invalid-Transformation	The transformation is not valid: it is unparsable or inconsistent.	
	InvalidModelSpec	The model specification is not valid: the model or the metamodels are inaccessible or invalid.	

Table 7: Operation CloudATL.getJobs

	Name	**Type**	**Documentation**
Operation	getJobs	list<string>	Lists the identifiers of the transformation jobs tracked by this server.
Parameters	None.		

Table 8: Operation CloudATL.getStatus

	Name	**Type**	**Documentation**
Operation	getStatus	Transformation-Status	Returns the status of a previously invoked transformation.
Parameters			
	token	string	A valid token returned by a previous call to launch().
Exceptions			
	Transformation-TokenNotFound	The specified transformation token does not exist within the invokved MONDO instance.	

Table 9: Operation CloudATL.kill

	Name	**Type**	**Documentation**
Operation	kill	void	Kills a previously invoked transformation.
Parameters			
	token	string	A valid token returned by a previous call to launch().
Exceptions			
	Transformation-TokenNotFound	The specified transformation token does not exist within the invokved MONDO instance.	

5.2.2 Hawk

The following service operations expose the capabilities of the Hawk heterogeneous model indexing framework developed in Work Package 5. The framework is discussed in detail in D5.2 and D5.3.

Table 10: Operation Hawk.createInstance

	Name	Type	Documentation
Operation	createInstance	void	Creates a new Hawk instance (stopped).
Parameters			
	name	string	The unique name of the new Hawk instance.
	backend	string	The name of the backend to be used, as returned by listBackends().
	minimumDelayMillis	i32	Minimum delay between periodic synchronization in milliseconds.
	maximumDelayMillis	i32	Maximum delay between periodic synchronization in milliseconds (0 to disable periodic synchronization).
	enabledPlugins (optional)	list<string>	List of plugins to be enabled: if not set, all plugins are enabled.

Table 11: Operation Hawk.listBackends

	Name	Type	Documentation
Operation	listBackends	list<string>	Lists the names of the available storage backends.
Parameters	None.		

Table 12: Operation Hawk.listPlugins

	Name	Type	Documentation
Operation	listPlugins	list<string>	Lists all the Hawk plugins that can be enabled or disabled: metamodel parsers, model parsers and graph change listeners.
Parameters	None.		

Table 13: Operation Hawk.listInstances

	Name	Type	Documentation

Operation	listInstances	list<Hawk-Instance>	Lists the details of all Hawk instances.
Parameters	None.		

Table 14: Operation Hawk.removeInstance

	Name	**Type**	**Documentation**
Operation	removeInstance	void	Removes an existing Hawk instance.
Parameters			
	name	string	The name of the Hawk instance to remove.
Exceptions			
	HawkInstanceNotFound	No Hawk instance exists with that name.	

Table 15: Operation Hawk.startInstance

	Name	**Type**	**Documentation**
Operation	startInstance	void	Starts a stopped Hawk instance.
Parameters			
	name	string	The name of the Hawk instance to start.
Exceptions			
	HawkInstanceNotFound	No Hawk instance exists with that name.	

Table 16: Operation Hawk.stopInstance

	Name	**Type**	**Documentation**
Operation	stopInstance	void	Stops a running Hawk instance.
Parameters			
	name	string	The name of the Hawk instance to stop.
Exceptions			
	HawkInstanceNotFound	No Hawk instance exists with that name.	
	HawkInstanceNotRunning	The selected Hawk instance is not running.	

Table 17: Operation Hawk.syncInstance

	Name	Type	Documentation
Operation	syncInstance	void	Forces an immediate synchronization on a Hawk instance.
Parameters			
	name	string	The name of the Hawk instance to stop.
	blockUntilDone (optional)	bool	If true, blocks the call until the synchronisation completes. False by default.
Exceptions			
	HawkInstanceNotFound	No Hawk instance exists with that name.	
	HawkInstanceNotRunning	The selected Hawk instance is not running.	

Table 18: Operation Hawk.registerMetamodels

	Name	Type	Documentation
Operation	registerMetamodels	void	Registers a set of file-based metamodels with a Hawk instance.
Parameters			
	name	string	The name of the Hawk instance.
	metamodel	list<File>	The metamodels to register. More than one metamodel file can be provided in one request, to accomodate fragmented metamodels.
Exceptions			
	HawkInstanceNotFound	No Hawk instance exists with that name.	
	InvalidMetamodel	The provided metamodel is not valid (e.g. unparsable or inconsistent).	
	HawkInstanceNotRunning	The selected Hawk instance is not running.	

Table 19: Operation Hawk.unregisterMetamodels

	Name	Type	Documentation
Operation	unregisterMetamodels	void	Unregisters a metamodel from a Hawk instance.
Parameters			
	name	string	The name of the Hawk instance.
	metamodel	list<string>	The URIs of the metamodels.
Exceptions			

	HawkInstanceNot-Found	No Hawk instance exists with that name.
	HawkInstanceNot-Running	The selected Hawk instance is not running.

Table 20: Operation Hawk.listMetamodels

	Name	Type	Documentation
Operation	listMetamodels	list<string>	Lists the URIs of the registered meta-models of a Hawk instance.
Parameters			
	name	string	The name of the Hawk instance.
Exceptions			
	HawkInstanceNot-Found	No Hawk instance exists with that name.	
	HawkInstanceNot-Running	The selected Hawk instance is not running.	

Table 21: Operation Hawk.listQueryLanguages

	Name	Type	Documentation
Operation	listQueryLanguages	list<string>	Lists the supported query languages and their status.
Parameters			
	name	string	The name of the Hawk instance.

Table 22: Operation Hawk.query

	Name	Type	Documentation
Operation	query	list<Query-Result>	Runs a query on a Hawk instance and returns a sequence of scalar values and/or model elements.
Parameters			
	name	string	The name of the Hawk instance.
	query	string	The query to be executed.
	language	string	The name of the query language used (e.g. EOL, OCL).
	options	HawkQuery-Options	Options for the query.
Exceptions			
	HawkInstanceNot-Found	No Hawk instance exists with that name.	

	HawkInstanceNot-Running	The selected Hawk instance is not running.
	UnknownQuery-Language	The specified query language is not supported by the operation.
	InvalidQuery	The specified query is not valid.
	FailedQuery	The specified query failed to complete its execution.

Table 23: Operation Hawk.resolveProxies

	Name	**Type**	**Documentation**
Operation	resolveProxies	list<Model-Element>	Returns populated model elements for the provided proxies.
Parameters			
	name	string	The name of the Hawk instance.
	ids	list<string>	Proxy model element IDs to be resolved.
	options	HawkQuery-Options	Options for the query.
Exceptions			
	HawkInstanceNot-Found	No Hawk instance exists with that name.	
	HawkInstanceNot-Running	The selected Hawk instance is not running.	

Table 24: Operation Hawk.addRepository

	Name	**Type**	**Documentation**
Operation	addRepository	void	Asks a Hawk instance to start monitoring a repository.
Parameters			
	name	string	The name of the Hawk instance.
	repo	Repository	The repository to monitor.
	credentials (optional)	Credentials	A valid set of credentials that has read-access to the repository.
Exceptions			
	HawkInstanceNot-Found	No Hawk instance exists with that name.	
	HawkInstanceNot-Running	The selected Hawk instance is not running.	
	Unknown-RepositoryType	The specified repository type is not supported by the operation.	

	VCSAuthentication-Failed	The client failed to prove its identity in the VCS.

Table 25: Operation Hawk.isFrozen

	Name	Type	Documentation
Operation	isFrozen	bool	Returns true if a repository is frozen, false otherwise.
Parameters			
	name	string	The name of the Hawk instance.
	uri	string	The URI of the repository to query.
Exceptions			
	HawkInstanceNot-Found	No Hawk instance exists with that name.	
	HawkInstanceNot-Running	The selected Hawk instance is not running.	

Table 26: Operation Hawk.setFrozen

	Name	Type	Documentation
Operation	setFrozen	void	Changes the 'frozen' state of a repository.
Parameters			
	name	string	The name of the Hawk instance.
	uri	string	The URI of the repository to be changed.
	isFrozen	bool	Whether the repository should be frozen (true) or not (false).
Exceptions			
	HawkInstanceNot-Found	No Hawk instance exists with that name.	
	HawkInstanceNot-Running	The selected Hawk instance is not running.	

Table 27: Operation Hawk.removeRepository

	Name	Type	Documentation
Operation	removeRepository	void	Asks a Hawk instance to stop monitoring a repository and remove its elements from the graph.
Parameters			

	name	string	The name of the Hawk instance.
	uri	string	The URI of the repository to stop monitoring.
Exceptions			
	HawkInstanceNotFound	No Hawk instance exists with that name.	
	HawkInstanceNotRunning	The selected Hawk instance is not running.	

Table 28: Operation Hawk.updateRepositoryCredentials

	Name	**Type**	**Documentation**
Operation	updateRepositoryCredentials	void	Changes the credentials used to monitor a repository.
Parameters			
	name	string	The name of the Hawk instance.
	uri	string	The URI of the repository to update.
	cred	Credentials	The new credentials to be used.
Exceptions			
	HawkInstanceNotFound	No Hawk instance exists with that name.	
	HawkInstanceNotRunning	The selected Hawk instance is not running.	

Table 29: Operation Hawk.listRepositories

	Name	**Type**	**Documentation**
Operation	listRepositories	list<-Repository>	Lists the repositories monitored by a Hawk instance.
Parameters			
	name	string	The name of the Hawk instance.
Exceptions			
	HawkInstanceNotFound	No Hawk instance exists with that name.	
	HawkInstanceNotRunning	The selected Hawk instance is not running.	

Table 30: Operation Hawk.listRepositoryTypes

	Name	**Type**	**Documentation**

Operation	listRepositoryTypes	list<string>	Lists the available repository types in this installation.
Parameters	None.		

Table 31: Operation Hawk.listFiles

	Name	**Type**	**Documentation**
Operation	listFiles	list<string>	Lists the paths of the files of the indexed repository.
Parameters			
	name	string	The name of the Hawk instance.
	repository	list<string>	The URI of the indexed repository.
	filePatterns	list<string>	File name patterns to search for (* lists all files).
Exceptions			
	HawkInstanceNotFound	No Hawk instance exists with that name.	
	HawkInstanceNotRunning	The selected Hawk instance is not running.	

Table 32: Operation Hawk.configurePolling

	Name	**Type**	**Documentation**
Operation	configurePolling	void	Sets the base polling period and max interval of a Hawk instance.
Parameters			
	name	string	The name of the Hawk instance.
	base	i32	The base polling period (in seconds).
	max	i32	The maximum polling interval (in seconds).
Exceptions			
	HawkInstanceNotFound	No Hawk instance exists with that name.	
	HawkInstanceNotRunning	The selected Hawk instance is not running.	
	InvalidPollingConfiguration	The polling configuration is not valid.	

Table 33: Operation Hawk.addDerivedAttribute

	Name	**Type**	**Documentation**

Operation	addDerivedAttribute	void	Add a new derived attribute to a Hawk instance.
Parameters			
	name	string	The name of the Hawk instance.
	spec	Derived-AttributeSpec	The details of the new derived attribute.
Exceptions			
	HawkInstanceNot-Found	No Hawk instance exists with that name.	
	HawkInstanceNot-Running	The selected Hawk instance is not running.	
	InvalidDerived-AttributeSpec	The derived attribute specification is not valid.	

Table 34: Operation Hawk.removeDerivedAttribute

	Name	**Type**	**Documentation**
Operation	removeDerived-Attribute	void	Remove a derived attribute from a Hawk instance.
Parameters			
	name	string	The name of the Hawk instance.
	spec	Derived-AttributeSpec	The details of the derived attribute to be removed. Only the first three fields of the spec need to be populated.
Exceptions			
	HawkInstanceNot-Found	No Hawk instance exists with that name.	
	HawkInstanceNot-Running	The selected Hawk instance is not running.	

Table 35: Operation Hawk.listDerivedAttributes

	Name	**Type**	**Documentation**
Operation	listDerived-Attributes	list<Derived-AttributeSpec->	Lists the derived attributes of a Hawk instance. Only the first three fields of the spec are currently populated.
Parameters			
	name	string	The name of the Hawk instance.
Exceptions			
	HawkInstanceNot-Found	No Hawk instance exists with that name.	

	HawkInstanceNot-Running	The selected Hawk instance is not running.

Table 36: Operation Hawk.addIndexedAttribute

	Name	Type	Documentation
Operation	addIndexedAttribute	void	Add a new indexed attribute to a Hawk instance.
Parameters			
	name	string	The name of the Hawk instance.
	spec	Indexed-AttributeSpec	The details of the new indexed attribute.
Exceptions			
	HawkInstanceNot-Found	No Hawk instance exists with that name.	
	HawkInstanceNot-Running	The selected Hawk instance is not running.	
	InvalidIndexed-AttributeSpec	The indexed attribute specification is not valid.	

Table 37: Operation Hawk.removeIndexedAttribute

	Name	Type	Documentation
Operation	removeIndexed-Attribute	void	Remove a indexed attribute from a Hawk instance.
Parameters			
	name	string	The name of the Hawk instance.
	spec	Indexed-AttributeSpec	The details of the indexed attribute to be removed.
Exceptions			
	HawkInstanceNot-Found	No Hawk instance exists with that name.	
	HawkInstanceNot-Running	The selected Hawk instance is not running.	

Table 38: Operation Hawk.listIndexedAttributes

	Name	Type	Documentation
Operation	listIndexed-Attributes	list<Indexed-AttributeSpec->	Lists the indexed attributes of a Hawk instance.
Parameters			

	name	string	The name of the Hawk instance.
Exceptions			
	HawkInstanceNot-Found	No Hawk instance exists with that name.	
	HawkInstanceNot-Running	The selected Hawk instance is not running.	

Table 39: Operation Hawk.getModel

	Name	**Type**	**Documentation**
Operation	getModel	list<Model-Element>	Returns the contents of one or more models indexed in a Hawk instance. Cross-model references are also resolved, and contained objects are always sent.
Parameters			
	name	string	The name of the Hawk instance.
	options	HawkQuery-Options	Options to limit the contents to be sent.
Exceptions			
	HawkInstanceNot-Found	No Hawk instance exists with that name.	
	HawkInstanceNot-Running	The selected Hawk instance is not running.	

Table 40: Operation Hawk.getRootElements

	Name	**Type**	**Documentation**
Operation	getRootElements	list<Model-Element>	Returns the root objects of one or more models indexed in a Hawk instance. Node IDs are always sent, and contained objects are never sent.
Parameters			
	name	string	The name of the Hawk instance.
	options	HawkQuery-Options	Options to limit the contents to be sent.

Table 41: Operation Hawk.watchStateChanges

	Name	**Type**	**Documentation**

Operation	watchStateChanges	Subscription	Returns subscription details to a queue of HawkStateEvents with notifications about changes in the indexer's state.
Parameters			
	name	string	The name of the Hawk instance.
Exceptions			
	HawkInstanceNot-Found	No Hawk instance exists with that name.	
	HawkInstanceNot-Running	The selected Hawk instance is not running.	

Table 42: Operation Hawk.watchModelChanges

	Name	**Type**	**Documentation**
Operation	watchModel-Changes	Subscription	Returns subscription details to a queue of HawkChangeEvents with notifications about changes to a set of indexed models.
Parameters			
	name	string	The name of the Hawk instance.
	repositoryUri	string	The URI of the repository in which the model is contained.
	filePath	list<string>	The pattern(s) for the model file(s) in the repository.
	clientID	string	Unique client ID (used as suffix for the queue name).
	durableEvents	Subscription-Durability	Durability of the subscription.
Exceptions			
	HawkInstanceNot-Found	No Hawk instance exists with that name.	
	HawkInstanceNot-Running	The selected Hawk instance is not running.	

5.2.3 OfflineCollaboration

The offline collaboration tool is realized in the MONDO platform through the MONDO Offline Collaboration Server, as mentioned in D4.4 [15]. It extends an off-the-shelf version control server with "hooks" that enforce access control and maintain the lens relationship between the "gold" repository and the "front" repositories. This allows users to continue using their preferred tools for interacting with the version control systems in their day-to-day modelling activities. The access control rules to be used by the security lens are themselves described in the repository.

Nevertheless, managing the operation of the hooks and the lens relationship requires its own API, as this is not covered by traditional VCS protocols. The rest of the section describes an API for managing these access rules.

Table 43: Operation OfflineCollaboration.listGoldRepositories

	Name	Type	Documentation
Operation	listGoldRepositories	list<string>	Retrieve the list of all managed gold repositories.
Parameters	None.		
Exceptions			
	Unauthorized-Repository-Operation	Authenticated user is not permitted to carry out the requested operation on the specified repository.	
	Offline-Collaboration-InternalError	An internal error occurred on the collaboration server. See details in server log.	

Table 44: Operation OfflineCollaboration.regenerateFrontRepositories

	Name	Type	Documentation
Operation	regenerateFront-Repositories	void	Regenerate all front repositories based on the gold repository. Requires superuser privileges.
Parameters			
	goldRepoURL	string	URL of the gold repository.
Exceptions			
	GoldRepoNotFound	No gold repository is configured at the specified URL.	
	Unauthorized-Repository-Operation	Authenticated user is not permitted to carry out the requested operation on the specified repository.	
	Offline-Collaboration-InternalError	An internal error occurred on the collaboration server. See details in server log.	

Table 45: Operation OfflineCollaboration.getMyFrontRepositoryURL

	Name	Type	Documentation

Operation	getMyFront-RepositoryURL	string	Retrieve the front repository URL for the current user.
Parameters			
	goldRepoURL	string	URL of the gold repository.
Exceptions			
	GoldRepoNotFound	No gold repository is configured at the specified URL.	
	Unauthorized-Repository-Operation	Authenticated user is not permitted to carry out the requested operation on the specified repository.	
	Offline-Collaboration-InternalError	An internal error occurred on the collaboration server. See details in server log.	

Table 46: Operation OfflineCollaboration.getOnlineCollaborationURL

	Name	**Type**	**Documentation**
Operation	getOnline-CollaborationURL	string	Retrieve the online collaboration access point URL for the current user.
Parameters			
	goldRepoURL	string	URL of the gold repository.
Exceptions			
	GoldRepoNotFound	No gold repository is configured at the specified URL.	
	Unauthorized-Repository-Operation	Authenticated user is not permitted to carry out the requested operation on the specified repository.	
	Offline-Collaboration-InternalError	An internal error occurred on the collaboration server. See details in server log.	

5.2.4 Users

The majority of service operations provided by the MONDO platform require user authentication (indicated in the top-left cell of each operation table) to prevent unaccountable use. As such, the platform needs to provide basic user management service operations for creating, updating and deleting user accounts.

Table 47: Operation Users.createUser

	Name	**Type**	**Documentation**
Operation	createUser	void	Creates a new platform user.

Parameters			
	username	string	A unique identifier for the user.
	password	string	The desired password.
	profile	UserProfile	The profile of the user.
Exceptions			
	UserExists	The specified username already exists.	

Table 48: Operation Users.updateProfile

	Name	**Type**	**Documentation**
Operation	updateProfile	void	Updates the profile of a platform user.
Parameters			
	username	string	The name of the user to update the profile of.
	profile	UserProfile	The updated profile of the user.
Exceptions			
	UserNotFound	The specified username does not exist.	

Table 49: Operation Users.updatePassword

	Name	**Type**	**Documentation**
Operation	updatePassword	void	Updates the password of a platform user.
Parameters			
	username	string	The name of the user to update the profile of.
	newPassword	string	New password to be set.
Exceptions			
	UserNotFound	The specified username does not exist.	

Table 50: Operation Users.deleteUser

	Name	**Type**	**Documentation**
Operation	deleteUser	void	Deletes a platform user.
Parameters			
	username	string	The name of the user to delete.
Exceptions			
	UserNotFound	The specified username does not exist.	

5.3 Entities

5.3.1 AttributeSlot

Represents a slot that can store the value(s) of an attribute of a model element.

Inherits from: Slot.

Table 51: Structure AttributeSlot

Field	Type	Documentation
name (inherited)	string	The name of the model element property the value of which is stored in this slot.
value	SlotValue (optional)	Value of the slot (if set).
Used in: ModelElement		

5.3.2 CommitItem

Simplified entry within a commit of a repository.

Table 52: Structure CommitItem

Field	Type	Documentation
path	string	Path within the repository, using / as separator.
repoURL	string	URL of the repository.
revision	string	Unique identifier of the revision of the repository.
type	CommitItemChange-Type	Type of change within the commit.
Used in: HawkModelElementAdditionEvent, HawkModelElementRemovalEvent, HawkAttributeUpdateEvent, HawkAttributeRemovalEvent, HawkReferenceAdditionEvent, HawkReferenceRemovalEvent, HawkFileAdditionEvent, HawkFileRemovalEvent		

5.3.3 ContainerSlot

Represents a slot that can store other model elements within a model element.

Inherits from: Slot.

Table 53: Structure ContainerSlot

Field	Type	Documentation
elements	list<ModelElement>	Contained elements for this slot.
name (inherited)	string	The name of the model element property the value of which is stored in this slot.

Used in: ModelElement

5.3.4 Credentials

Credentials of the client in the target VCS.

Table 54: Structure Credentials

Field	**Type**	**Documentation**
password	string	Password for logging into the VCS.
username	string	Username for logging into the VCS.
Used in: Hawk.addRepository, Hawk.updateRepositoryCredentials		

5.3.5 DerivedAttributeSpec

Used to configure Hawk's derived attributes (discussed in D5.3).

Table 55: Structure DerivedAttributeSpec

Field	**Type**	**Documentation**
attributeName	string	The name of the derived attribute.
attributeType	string (optional)	The (primitive) type of the derived attribute.
derivationLanguage	string (optional)	The language used to express the derivation logic.
derivationLogic	string (optional)	An executable expression of the derivation logic in the language above.
isMany	bool (optional)	The multiplicity of the derived attribute.
isOrdered	bool (optional)	A flag specifying whether the order of the values of the derived attribute is significant (only makes sense when isMany=true).
isUnique	bool (optional)	A flag specifying whether the the values of the derived attribute are unique (only makes sense when isMany=true).
metamodelUri	string	The URI of the metamodel to which the derived attribute belongs.
typeName	string	The name of the type to which the derived attribute belongs.
Used in: Hawk.addDerivedAttribute, Hawk.removeDerivedAttribute, Hawk.listDerivedAttributes		

5.3.6 EffectiveMetamodel

Representation of a set of rules for either including or excluding certain types and/or slots within a metamodel.

Table 56: Structure EffectiveMetamodel

Field	Type	Documentation
slots	set<string>	Slots within the type that should be included or excluded: empty means 'all slots'.
type	string	Type that should be included or excluded.
Used in: EffectiveMetamodelMap		

5.3.7 EffectiveMetamodelMap

Representation of a set of rules for either including or excluding metamodels, types or slots.

Table 57: Structure EffectiveMetamodelMap

Field	Type	Documentation
metamodel	map<string,set<string>>	Types and slots within the metamodel that should be included or excluded: empty means 'all types and slots'.
uri	string	Namespace URI of the metamodel.
Used in: HawkQueryOptions		

5.3.8 File

A file to be sent through the network.

Table 58: Structure File

Field	Type	Documentation
contents	binary	Sequence of bytes with the contents of the file.
name	string	Name of the file.
Used in: Hawk.registerMetamodels		

5.3.9 HawkAttributeRemovalEvent

Serialized form of an attribute removal event.

Table 59: Structure HawkAttributeRemovalEvent

Field	Type	Documentation
attribute	string	Name of the attribute that was removed.
id	string	Identifier of the model element that was changed.
vcsItem	CommitItem	Entry within the commit that produced the changes.
Used in: HawkChangeEvent		

5.3.10 HawkAttributeUpdateEvent

Serialized form of an attribute update event.

Table 60: Structure HawkAttributeUpdateEvent

Field	Type	Documentation
attribute	string	Name of the attribute that was changed.
id	string	Identifier of the model element that was changed.
value	SlotValue	New value for the attribute.
vcsItem	CommitItem	Entry within the commit that produced the changes.
Used in: HawkChangeEvent		

5.3.11 HawkChangeEvent

Serialized form of a change in the indexed models of a Hawk instance.

Table 61: Union HawkChangeEvent

Field	Type	Documentation
fileAddition	HawkFileAddition-Event	A file was added.
fileRemoval	HawkFileRemoval-Event	A file was removed.
modelElementAddition	HawkModelElement-AdditionEvent	A model element was added.
modelElement-AttributeRemoval	HawkAttribute-RemovalEvent	An attribute was removed.
modelElement-AttributeUpdate	HawkAttributeUpdate-Event	An attribute was updated.
modelElementRemoval	HawkModelElement-RemovalEvent	A model element was removed.

referenceAddition	HawkReference-AdditionEvent	A reference was added.
referenceRemoval	HawkReference-RemovalEvent	A reference was removed.
syncEnd	HawkSynchronization-EndEvent	Synchronization ended.
syncStart	HawkSynchronization-StartEvent	Synchronization started.

5.3.12 HawkFileAdditionEvent

Serialized form of a file addition event.

Table 62: Structure HawkFileAdditionEvent

Field	Type	Documentation
vcsItem	CommitItem	Reference to file that was added, including VCS metadata.
Used in: HawkChangeEvent		

5.3.13 HawkFileRemovalEvent

A file was removed.

Table 63: Structure HawkFileRemovalEvent

Field	Type	Documentation
vcsItem	CommitItem	Reference to file that was removed, including VCS metadata.
Used in: HawkChangeEvent		

5.3.14 HawkInstance

Status of a Hawk instance.

Table 64: Structure HawkInstance

Field	Type	Documentation
message	string	Last info message from the instance.
name	string	The name of the instance.
state	HawkState	Current state of the instance.
Used in: Hawk.listInstances		

5.3.15 HawkModelElementAdditionEvent

Serialized form of a model element addition event.

Table 65: Structure HawkModelElementAdditionEvent

Field	Type	Documentation
id	string	Identifier of the model element that was added.
metamodelURI	string	Metamodel URI of the type of the model element.
typeName	string	Name of the type of the model element.
vcsItem	CommitItem	Entry within the commit that produced the changes.
Used in: HawkChangeEvent		

5.3.16 HawkModelElementRemovalEvent

Serialized form of a model element removal event.

Table 66: Structure HawkModelElementRemovalEvent

Field	Type	Documentation
id	string	Identifier of the model element that was removed.
vcsItem	CommitItem	Entry within the commit that produced the changes.
Used in: HawkChangeEvent		

5.3.17 HawkQueryOptions

Options for a Hawk query.

Table 67: Structure HawkQueryOptions

Field	Type	Documentation
defaultNamespaces	string (optional)	The default namespaces to be used to resolve ambiguous unqualified types.
effectiveMetamodel-Excludes	EffectiveMetamodel-Map (optional)	If set and not empty, the mentioned metamodels, types and features will not be fetched. The string '*' can be used to refer to all types within a metamodel or all fields within a type.

effectiveMetamodel-Includes	EffectiveMetamodel-Map (optional)	If set and not empty, only the specified metamodels, types and features will be fetched. Otherwise, everything that is not excluded will be fetched. The string '*' can be used to refer to all types within a meta-model or all fields within a type.
filePatterns	list<string> (optional)	The file patterns for the query (e.g. *.uml).
includeAttributes	bool (optional)	Whether to include attributes (true) or not (false) in model element results.
includeContained	bool (optional)	Whether to include all the child elements of the model element results (true) or not (false).
includeNodeIDs	bool (optional)	Whether to include node IDs (true) or not (false) in model element results.
includeReferences	bool (optional)	Whether to include references (true) or not (false) in model element results.
repositoryPattern	string (optional)	The repository for the query (or * for all repositories).
Used in: Hawk.query, Hawk.resolveProxies, Hawk.getModel, Hawk.getRootElements		

5.3.18 HawkReferenceAdditionEvent

Serialized form of a reference addition event.

Table 68: Structure HawkReferenceAdditionEvent

Field	Type	Documentation
refName	string	Name of the reference that was added.
sourceId	string	Identifier of the source model element.
targetId	string	Identifier of the target model element.
vcsItem	CommitItem	Entry within the commit that produced the changes.
Used in: HawkChangeEvent		

5.3.19 HawkReferenceRemovalEvent

Serialized form of a reference removal event.

Table 69: Structure HawkReferenceRemovalEvent

Field	Type	Documentation
refName	string	Name of the reference that was removed.
sourceId	string	Identifier of the source model element.

targetId	string	Identifier of the target model element.
vcsItem	CommitItem	Entry within the commit that produced the changes.
Used in: HawkChangeEvent		

5.3.20 HawkStateEvent

Serialized form of a change in the state of a Hawk instance.

Table 70: Structure HawkStateEvent

Field	Type	Documentation
message	string	Short message about the current status of the server.
state	HawkState	State of the Hawk instance.
timestamp	i64	Timestamp for this state change.

5.3.21 HawkSynchronizationEndEvent

Serialized form of a sync end event.

Table 71: Structure HawkSynchronizationEndEvent

Field	Type	Documentation
timestampNanos	i64	Local timestamp, measured in nanoseconds. Only meant to be used to compute synchronization cost.
Used in: HawkChangeEvent		

5.3.22 HawkSynchronizationStartEvent

Serialized form of a sync start event.

Table 72: Structure HawkSynchronizationStartEvent

Field	Type	Documentation
timestampNanos	i64	Local timestamp, measured in nanoseconds. Only meant to be used to compute synchronization cost.
Used in: HawkChangeEvent		

5.3.23 IndexedAttributeSpec

Used to configure Hawk's indexed attributes (discussed in D5.3).

Table 73: Structure IndexedAttributeSpec

Field	Type	Documentation
attributeName	string	The name of the indexed attribute.
metamodelUri	string	The URI of the metamodel to which the indexed attribute belongs.
typeName	string	The name of the type to which the indexed attribute belongs.
Used in: Hawk.addIndexedAttribute, Hawk.removeIndexedAttribute, Hawk.listIndexedAttributes		

5.3.24 MixedReference

Represents a reference to a model element: it can be an identifier or a position. Only used when the same ReferenceSlot has both identifier-based and position-based references. This may be the case if we are retrieving a subset of the model which has references between its elements and with elements outside the subset at the same time.

Table 74: Union MixedReference

Field	Type	Documentation
id	string	Identifier-based reference to a model element.
position	i32	Position-based reference to a model element.
Used in: ReferenceSlot		

5.3.25 ModelElement

Represents a model element.

Table 75: Structure ModelElement

Field	Type	Documentation
attributes	list<AttributeSlot> (optional)	Slots holding the values of the model element's attributes, if any have been set.
containers	list<ContainerSlot> (optional)	Slots holding contained model elements, if any have been set.

file	string (optional)	Name of the file to which the element belongs (not set if equal to that of the previous model element).
id	string (optional)	Unique ID of the model element (not set if using position-based references).
metamodelUri	string (optional)	URI of the metamodel to which the type of the element belongs (not set if equal to that of the previous model element).
references	list<ReferenceSlot> (optional)	Slots holding the values of the model element's references, if any have been set.
repositoryURL	string (optional)	URI of the repository to which the element belongs (not set if equal to that of the previous model element).
typeName	string (optional)	Name of the type that the model element is an instance of (not set if equal to that of the previous model element).
Used in: Hawk.resolveProxies, Hawk.getModel, Hawk.getRootElements, ContainerSlot, QueryResult		

5.3.26 ModelElementType

Represents a type of model element.

Table 76: Structure ModelElementType

Field	Type	Documentation
attributes	list<SlotMetadata> (optional)	Metadata for the attribute slots.
id	string	Unique ID of the model element type.
metamodelUri	string	URI of the metamodel to which the type belongs.
references	list<SlotMetadata> (optional)	Metadata for the reference slots.
typeName	string	Name of the type.
Used in: QueryResult		

5.3.27 ModelSpec

Captures information about source/target models of ATL transformations.

Table 77: Structure ModelSpec

Field	Type	Documentation

metamodelUris	list<string>	The URIs of the metamodels to which elements of the model conform.
uri	string	The URI from which the model will be loaded or to which it will be persisted.
Used in: CloudATL.launch, InvalidModelSpec		

5.3.28 QueryResult

Union type for a scalar value or a reference to a model element. Useful for heterogeneous collections.

Inherits from: Value.

Table 78: Union QueryResult

Field	Type	Documentation
vBoolean (inherited)	bool	Boolean (true/false) value.
vByte (inherited)	byte	8-bit signed integer value.
vDouble (inherited)	double	64-bit floating point value.
vInteger (inherited)	i32	32-bit signed integer value.
vLong (inherited)	i64	64-bit signed integer value.
vModelElement	ModelElement	Encoded model element.
vModelElementType	ModelElementType	Encoded model element type.
vShort (inherited)	i16	16-bit signed integer value.
vString (inherited)	string	Sequence of UTF8 characters.
Used in: Hawk.query		

5.3.29 ReferenceSlot

Represents a slot that can store the value(s) of a reference of a model element. References can be expressed as positions within a result tree (using pre-order traversal) or IDs. id, ids, position, positions and mixed are all mutually exclusive. At least one position or one ID must be given.

Inherits from: Slot.

Table 79: Structure ReferenceSlot

Field	Type	Documentation
id	string (optional)	Unique identifier of the referenced element (if there is only one ID based reference in this slot).
ids	list<string> (optional)	Unique identifiers of the referenced elements (if more than one).

mixed	list<MixedReference> (optional)	Mix of identifier- and position-bsaed references (if there is at least one position and one ID).
name (inherited)	string	The name of the model element property the value of which is stored in this slot.
position	i32 (optional)	Position of the referenced element (if there is only one position-based reference in this slot).
positions	list<i32> (optional)	Positions of the referenced elements (if more than one).
Used in: ModelElement		

5.3.30 Repository

Entity that represents a model repository.

Table 80: Structure Repository

Field	Type	Documentation
isFrozen	bool (optional)	True if the repository is frozen, false otherwise.
type	string	The type of repository.
uri	string	The URI to the repository.
Used in: Hawk.addRepository, Hawk.listRepositories		

5.3.31 Slot

Represents a slot that can store the value(s) of a property of a model element.

Inherited by: AttributeSlot, ReferenceSlot, ContainerSlot.

Table 81: Structure Slot

Field	Type	Documentation
name	string	The name of the model element property the value of which is stored in this slot.

5.3.32 SlotMetadata

Represents the metadata of a slot in a model element type.

Table 82: Structure SlotMetadata

Field	Type	Documentation

isMany	bool	True if this slot holds a collection of values instead of a single value.
isOrdered	bool	True if the values in this slot are ordered.
isUnique	bool	True if the value of this slot must be unique within its containing model.
name	string	The name of the model element property that is stored in this slot.
type	string	The type of the values in this slot.
Used in: ModelElementType		

5.3.33 SlotValue

Union type for a single scalar value or a homogeneous collection of scalar values.

Inherits from: Value.

Table 83: Union SlotValue

Field	Type	Documentation
vBoolean (inherited)	bool	Boolean (true/false) value.
vBooleans	list<bool>	List of true/false values.
vByte (inherited)	byte	8-bit signed integer value.
vBytes	binary	List of 8-bit signed integers.
vDouble (inherited)	double	64-bit floating point value.
vDoubles	list<double>	List of 64-bit floating point values.
vInteger (inherited)	i32	32-bit signed integer value.
vIntegers	list<i32>	List of 32-bit signed integers.
vLong (inherited)	i64	64-bit signed integer value.
vLongs	list<i64>	List of 64-bit signed integers.
vShort (inherited)	i16	16-bit signed integer value.
vShorts	list<i16>	List of 16-bit signed integers.
vString (inherited)	string	Sequence of UTF8 characters.
vStrings	list<string>	List of sequences of UTF8 characters.
Used in: HawkAttributeUpdateEvent, AttributeSlot		

5.3.34 Subscription

Details about a subscription to a topic queue.

Table 84: Structure Subscription

Field	Type	Documentation
host	string	Host name of the message queue server.

port	i32	Port in which the message queue server is listening.
queueAddress	string	Address of the topic queue.
queueName	string	Name of the topic queue.
sslRequired	bool	Whether SSL is required or not.
Used in: Hawk.watchStateChanges, Hawk.watchModelChanges		

5.3.35 TransformationStatus

Used to report the status of a long-running transformation by CloudATL.

Table 85: Structure TransformationStatus

Field	Type	Documentation
elapsed	i64	Time passed since the start of execution, in milliseconds.
error	string	Description of the error that caused the transformation to fail.
state	TransformationState	State of the tranformation.
Used in: CloudATL.getStatus		

5.3.36 UserProfile

Minimal details about registered users.

Table 86: Structure UserProfile

Field	Type	Documentation
admin	bool	Whether the user has admin rights (i.e. so that they can create new users, change the status of admin users etc).
realName	string	The real name of the user.
Used in: Users.createUser, Users.updateProfile		

5.3.37 Value

Union type for a single scalar value.

Inherited by: QueryResult, SlotValue.

Table 87: Union Value

Field	Type	Documentation
vBoolean	bool	Boolean (true/false) value.

vByte	byte	8-bit signed integer value.
vDouble	double	64-bit floating point value.
vInteger	i32	32-bit signed integer value.
vLong	i64	64-bit signed integer value.
vShort	i16	16-bit signed integer value.
vString	string	Sequence of UTF8 characters.

5.4 Enumerations

5.4.1 CommitItemChangeType

Table 88: Enumeration CommitItemChangeType

Name	Documentation
ADDED	File was added.
DELETED	File was removed.
REPLACED	File was removed.
UNKNOWN	Unknown type of change.
UPDATED	File was updated.
Used in: CommitItem	

5.4.2 HawkState

Table 89: Enumeration HawkState

Name	Documentation
RUNNING	The instance is running and monitoring the indexed locations.
STOPPED	The instance is stopped and is not monitoring any indexed locations.
UPDATING	The instance is updating its contents from the indexed locations.
Used in: HawkStateEvent, HawkInstance	

5.4.3 SubscriptionDurability

Table 90: Enumeration SubscriptionDurability

Name	Documentation
DEFAULT	Subscription survives client disconnections but not server restarts.
DURABLE	Subscription survives client disconnections and server restarts.

TEMPORARY	Subscription removed after disconnecting.
Used in: Hawk.watchModelChanges	

5.4.4 TransformationState

Table 91: Enumeration TransformationState

Name	Documentation
FAILED	The transformation has failed.
KILLED	The transformation was interrupted by a user (i.e. killed).
PREP	The transformation is in preparation.
RUNNING	The transformation is running.
SUCCEEDED	The transformation has completed successfully.
Used in: TransformationStatus	

5.5 Exceptions

5.5.1 FailedQuery

The specified query failed to complete its execution.

Table 92: Exception FailedQuery

Field	Type	Documentation
reason	string	Reason for the query failing to complete its execution.
Used in: Hawk.query		

5.5.2 GoldRepoNotFound

No gold repository is configured at the specified URL.

Table 93: Exception GoldRepoNotFound

Field	Type	Documentation
—	—	—
Used in: OfflineCollaboration.regenerateFrontRepositories, OfflineCollaboration.getMyFrontRepositoryURL, OfflineCollaboration.getOnlineCollaborationURL		

5.5.3 HawkInstanceNotFound

No Hawk instance exists with that name.

Table 94: Exception HawkInstanceNotFound

Field	Type	Documentation
—	—	—
Used in: Hawk.removeInstance, Hawk.startInstance, Hawk.stopInstance, Hawk.syncInstance, Hawk.registerMetamodels, Hawk.unregisterMetamodels, Hawk.listMetamodels, Hawk.query, Hawk.resolveProxies, Hawk.addRepository, Hawk.isFrozen, Hawk.setFrozen, Hawk.removeRepository, Hawk.updateRepositoryCredentials, Hawk.listRepositories, Hawk.listFiles, Hawk.configurePolling, Hawk.addDerivedAttribute, Hawk.removeDerivedAttribute, Hawk.listDerivedAttributes, Hawk.addIndexedAttribute, Hawk.removeIndexedAttribute, Hawk.listIndexedAttributes, Hawk.getModel, Hawk.watchStateChanges, Hawk.watchModelChanges		

5.5.4 HawkInstanceNotRunning

The selected Hawk instance is not running.

Table 95: Exception HawkInstanceNotRunning

Field	Type	Documentation
—	—	—
Used in: Hawk.stopInstance, Hawk.syncInstance, Hawk.registerMetamodels, Hawk.unregisterMetamodels, Hawk.listMetamodels, Hawk.query, Hawk.resolveProxies, Hawk.addRepository, Hawk.isFrozen, Hawk.setFrozen, Hawk.removeRepository, Hawk.updateRepositoryCredentials, Hawk.listRepositories, Hawk.listFiles, Hawk.configurePolling, Hawk.addDerivedAttribute, Hawk.removeDerivedAttribute, Hawk.listDerivedAttributes, Hawk.addIndexedAttribute, Hawk.removeIndexedAttribute, Hawk.listIndexedAttributes, Hawk.getModel, Hawk.watchStateChanges, Hawk.watchModelChanges		

5.5.5 InvalidDerivedAttributeSpec

The derived attribute specification is not valid.

Table 96: Exception InvalidDerivedAttributeSpec

Field	Type	Documentation
reason	string	Reason for the spec not being valid.
Used in: Hawk.addDerivedAttribute		

5.5.6 InvalidIndexedAttributeSpec

The indexed attribute specification is not valid.

Table 97: Exception InvalidIndexedAttributeSpec

Field	Type	Documentation
reason	string	Reason for the spec not being valid.
Used in: Hawk.addIndexedAttribute		

5.5.7 InvalidMetamodel

The provided metamodel is not valid (e.g. unparsable or inconsistent).

Table 98: Exception InvalidMetamodel

Field	Type	Documentation
reason	string	Reason for the metamodel not being valid.
Used in: Hawk.registerMetamodels		

5.5.8 InvalidModelSpec

The model specification is not valid: the model or the metamodels are inaccessible or invalid.

Table 99: Exception InvalidModelSpec

Field	Type	Documentation
reason	string	Reason for the spec not being valid.
spec	ModelSpec	A copy of the invalid model specification.
Used in: CloudATL.launch		

5.5.9 InvalidPollingConfiguration

The polling configuration is not valid.

Table 100: Exception InvalidPollingConfiguration

Field	Type	Documentation
reason	string	Reason for the spec not being valid.
Used in: Hawk.configurePolling		

5.5.10 InvalidQuery

The specified query is not valid.

Table 101: Exception InvalidQuery

Field	Type	Documentation
reason	string	Reason for the query not being valid.
Used in: Hawk.query		

5.5.11 InvalidTransformation

The transformation is not valid: it is unparsable or inconsistent.

Table 102: Exception InvalidTransformation

Field	Type	Documentation
location	string	Location of the problem, if applicable. Usually a combination of line and column numbers.
reason	string	Reason for the transformation not being valid.
Used in: CloudATL.launch		

5.5.12 OfflineCollaborationInternalError

An internal error occurred on the collaboration server. See details in server log.

Table 103: Exception OfflineCollaborationInternalError

Field	Type	Documentation
errorMessage	string	Summary description of the error. See details in server log.
Used in: OfflineCollaboration.listGoldRepositories, OfflineCollaboration.regenerateFrontRepositories, OfflineCollaboration.getMyFrontRepositoryURL, OfflineCollaboration.getOnlineCollaborationURL		

5.5.13 TransformationTokenNotFound

The specified transformation token does not exist within the invokved MONDO instance.

Table 104: Exception TransformationTokenNotFound

Field	Type	Documentation
token	string	Transformation token which was not found within the invoked MONDO instance.
Used in: CloudATL.getStatus, CloudATL.kill		

5.5.14 UnauthorizedRepositoryOperation

Authenticated user is not permitted to carry out the requested operation on the specified repository.

Table 105: Exception UnauthorizedRepositoryOperation

Field	Type	Documentation
—	—	—
Used in: OfflineCollaboration.listGoldRepositories, OfflineCollaboration.regenerateFrontRepositories, OfflineCollaboration.getMyFrontRepositoryURL, OfflineCollaboration.getOnlineCollaborationURL		

5.5.15 UnknownQueryLanguage

The specified query language is not supported by the operation.

Table 106: Exception UnknownQueryLanguage

Field	Type	Documentation
—	—	—
Used in: Hawk.query		

5.5.16 UnknownRepositoryType

The specified repository type is not supported by the operation.

Table 107: Exception UnknownRepositoryType

Field	Type	Documentation
—	—	—
Used in: Hawk.addRepository		

5.5.17 UserExists

The specified username already exists.

Table 108: Exception UserExists

Field	Type	Documentation
—	—	—
Used in: Users.createUser		

5.5.18 UserNotFound

The specified username does not exist.

Table 109: Exception UserNotFound

Field	Type	Documentation
—	—	—
Used in: Users.updateProfile, Users.updatePassword, Users.deleteUser		

5.5.19 VCSAuthenticationFailed

The client failed to prove its identity in the VCS.

Table 110: Exception VCSAuthenticationFailed

Field	Type	Documentation
—	—	—
Used in: Hawk.addRepository		

6 Conclusions and Future Work

This deliverable presented the final state of the components of the MONDO platform (namely, the scalable modelling tools developed in WP2, the reactive and cloud-based transformation engines developed in WP3, the online and offline collaborative modelling frameworks developed in WP4 and the heterogeneous model indexing framework developed in WP5) as of M30, discussed how they interoperate with each other, and outlined the web-service API of the cloud-based part of the platform. All components are fully developed and the platform has been used by the industrial use case providers to conduct their case studies.

A Research and Development Requirement Status

Req. No.	Requirement	Priority	Satisfied?	Justification
D1	The system shall include optimization patterns for the design of the abstract syntax.	SHALL	Yes	DSL-tao contains has a modularity pattern for fragmentation.
D2	Optimizations shall be directed to address scalability aspects related to efficient modularity of the expected instance models.	SHALL	Yes	DSL-tao contains has a modularity pattern for fragmentation.
D3	The system shall provide modularity mechanisms, so that instance models can be fragmented, and composed.	SHALL	Yes	Decomposition and composition mechanisms are supported by the tools generated from DSL-tao.
D4	The system shall enable the construction of meta-models by means of libraries of reusable patterns.	SHALL	Yes	DSL-tao offers pattern-based meta-model construction, and is equiped with a library of patterns.
D5	The system shall provide abstraction mechanisms to facilitate the visualization and exploration of large-scale models.	SHALL	Yes	The SAMPLER tools permits defining abstractions and provides graphical visualization. It also supports the fragmentation as defined in DSL-tao.
D6	Abstraction mechanisms shall be reusable, across different modelling languages.	SHALL	Yes	SAMPLER is meta-model independent.
D7	The system shall provide a library of concrete syntax styles, so that a coarse-grained reutilization can be achieved.	SHALL	Yes	DSL-tao provides domain patterns, which then can be attached secondary patterns (e.g., for concrete syntax) with different styles.
D8	The system shall provide mechanisms to semi-automatically generate default concrete syntaxes for meta-models lacking a visualization to facilitate the visualization and exploration of large-scale models.	SHALL	Yes	In two ways: (1) SAMPLER can visualize models in a graph- based way. (2) DSL-tao/EMFSplitter provides a wizard to automatically produce a concrete syntax for a meta-model.

Continued on next page

Continued from previous page

Req. No.	Requirement	Priority	Satisfied?	Justification
D9	A methodology shall be provided which considers the design of the abstract and concrete syntaxes of a DSL and integrates these within the tool.	SHALL	Yes	Methodological guidelines were described in the deliverable D2.4.
D10	The system should provide a means to partition diagrams according to a set of customisable parameters.	SHOULD	Partially	DSL-tao provides capabilities to divide diagrams into multiple views. These views allow to "drill down" into the contents of an object. Currently, the division can only be done through containment references.
D11	The system should consider models with no defined visualization, and provide one according to what is provided for the associated metamodel.	SHOULD	Yes	SAMPLER, EMFSPLITTER wizard.
D12	The system should be able to integrate EMF-based DSLs with IFC heterogeneous technologies.	SHOULD	Yes	It can be done by using the BIMserver Ecore metamodel for IFC.
D13	The system should be able to integrate EMF-based DSLs with UML heterogeneous technologies.	SHOULD	Yes	It can be done by using the EMF UML2 metamodel.
D14	The system should be able to visualize views extracted through model queries.	SHOULD	No	Note: this is possible using VIATRA Viewers (`https://wiki.eclipse.org/VIATRA/Addon/VIATRA_Viewers`).
D15	The system may include an intelligent recommender system suggesting the applicability of certain patterns. For example, recommendations of patterns related to optimization on the basis of the expected sizes, queries and model usages.	MAY	Yes	The recommender is on DSL-maps.

Continued on next page

Continued from previous page

Req. No.	Requirement	Priority	Satisfied?	Justification
D16	The system may be able to integrate EMF-based DSLs with Matlab/Simulink heterogeneous technologies.	MAY	Yes	It can be done using the Matlab/Simulink EMF integration of the Massif project.
D17	The system may include textual, visual or tabular support for hybrid concrete syntaxes.	MAY	No	The pattern for tabular visualization is created, but no support has been implemented yet.
D18	Different DSL development methodologies may be supported by the tool (e.g. top-down or bottom-up).	MAY	Yes	
D19	Optimizations shall be available to address different scalability aspects of the expected instance models for efficient traversal and efficient queries.	SHALL	Yes	Reassigned from WP3 to WP6. EMF-Splitter can use Hawk to perform efficient querying of candidates for a reference, according to certain scoping rules.
D20	A set of query and transformation benchmarks shall be provided in a publicly available benchmark repository.	SHALL	Yes	4 use cases are made available on Github.
D21	Each benchmark shall include a set of real-world models or a way of generating reproducible models.	SHALL	Yes	Models coming from partners, or generated (we provide seeds), or reverse engineered from real world projects.
D22	Languages for designing reactive model query / transformation networks shall be provided.	SHALL	Yes	Reactive ATL supports transformation chains with incremental propagation and lazy evaluation. VIATRA3 supports event-driven/reactive transformation chains, streaming transformations and complex event processing.
D23	A virtual machine to generate and lazily recompute the elements of a set of evolving models shall be provided.	SHALL	Yes	ATL supports lazy computation and modification (invalidate lazy recompute).

Continued on next page

Continued from previous page

Req. No.	Requirement	Priority	Satisfied?	Justification
D24	The virtual machine language shall be designed to allow for automated parallelization over existing frameworks (e.g. HADOOP).	SHALL	Yes	ATL-MR.
D25	The virtual machine language shall be designed to allow for task distribution over existing frameworks (e.g. GIRAPH).	SHALL	Yes	ATL-MR with type-based assignment.
D26	It shall be possible to use existing MDE applications over the models in the reactive system.	SHALL	Yes	EMF API can be used together with Reactive ATL. Reactive models can be opened with the EMF reflective editor. VIATRA3 has full EMF support.
D27	A network communication protocol for model transformation in distributed/cloud environments shall be provided.	SHALL	Yes	For invoking the transformation, the CloudATL Thrift API can be used. The transformation itself is coordinated through the Hadoop protocols.
D28	The user of the virtual machine shall be able to define if the execution environment is local, distributed or cloud-based.	SHALL	Yes	The ATLMR VM is designed to run both in local and distributed modes.
D29	The use of a resource (e.g. model) in the Cloud shall be transparent (i.e. the same API as for local resources shall be used).	SHALL	Yes	Hawk and NeoEMF resources extend the EMF resource.
D30	The query/transformation layer shall support EMF, Express and Modelio models.	SHALL	Yes	ATL-MR supports Hawk-indexed models, VIATRA3 supports Hawk-indexed models via IncQuery integration.
D31	A scalable model transformation and query framework shall be provided.	SHALL	Yes	ATL-MR IncQuery-D.

Continued on next page

Continued from previous page

Req. No.	Requirement	Priority	Satisfied?	Justification
D32	The query and transformation languages should allow users to specify transformations over finite models.	SHOULD	Yes	ATL-MR IncQuery-D.
D33	The query and transformation languages should allow users to specify transformations over infinite or streaming models.	SHOULD	Yes	Reactive ATL supports infinite models through its features of incremental propagation and lazy evaluation. Lazy evaluation also allows the generation of infinite models. VIATRA3 supports streaming transformations via reactive transformations and complex event processing.
D34	If the query and transformation languages are developed as extensions of existing languages, backward compatibility should be maintained in terms of concrete syntax.	SHOULD	Yes	No distribution constructs are added which guarantees backward compatibility. Reactive ATL does not change the concrete syntax. However, it does support only a subset of the language.
D35	The query and transformation languages should be computationally complete.	SHOULD	Yes	
D36	The virtual machine should support the execution of several high-level model query languages.	SHOULD	Yes	VIATRA3's EVM has architectural support for this, however no implementations other than IQPL (VQL) exist as of today. The EVM has been used with multiple query languages in D37.
D37	The virtual machine should support the execution of several model transformation languages.	SHOULD	Yes	e.g. SimpleGT.
D38	The virtual machine language should be designed to allow for data distribution over existing frameworks (e.g. HADOOP).	SHOULD	Yes	By distributing model elements over the cluster.

Continued on next page

Continued from previous page

Req. No.	Requirement	Priority	Satisfied?	Justification
D39	Modifications to existing MDE applications should be necessary only for exploiting advanced features, i.e. additional to the standard model-access API.	SHOULD	Yes	Hawk exposes standard EMF interfaces with additional methods for advanced features.
D40	It should be possible to modify the queries and transformations of the reactive engine at runtime and have the system adapt by triggering minimal computation (e.g. live adaptation).	SHOULD	Yes	Reactive ATL can react to modifications of the transformation itself at runtime.
D41	The query/transformation layer should interact with the persistence layer by providing information on the usage patterns of models.	SHOULD	No	Lower priority requirement.
D42	It may be possible to select independently the query and transformation language for different parts of the reactive transformations engine and have them seamlessly interact.	MAY	No	Lower priority requirement. This would require writing a new compiler that could combine multiple languages in the same transformation.
D43	Collaboration features (e.g. push, commit, and access control) shall be available as a service, so that they can be used from MONDO editors.	SHALL	Yes	Management APIs accessible through MONDO Thrift API, rest of operations are delegated to underlying VCS protocols (offline) or via the web UI (online).
D44	A collaboration framework shall support the collaboration of multiple users when using modelling tools having support for MONDO features.	SHALL	Yes	The requirements are more relaxed, as non-MONDO tools can also participate.
D45	The collaboration framework shall provide offline collaboration (i.e. like Git/SVN) primitives such as push, pull, commit and merge.	SHALL	Yes	Core functionality.

Continued on next page

Continued from previous page

Req. No.	Requirement	Priority	Satisfied?	Justification
D46	The collaboration framework shall provide online collaboration (i.e. like Google Documents) primitives such as session start/end, join, leave.	SHALL	Yes	Core functionality.
D47	The collaboration framework shall provide secure and fine-grained locking and access control services where rules are specified at the model element level.	SHALL	Yes	Core functionality.
D48	The collaboration framework shall provide secure and fine-grained locking and access control services where the set of rules can be changed at runtime.	SHALL	Yes	Core functionality.
D49	The collaboration framework shall support domain-specific conflict resolution strategies in model merging.	SHALL	Yes	DSE Merge add-on.
D50	The collaboration framework shall support multi-device scenarios where at least two different types of devices can participate in online collaboration sessions.	SHALL	Yes	Tested with online framework.
D51	The collaboration framework shall integrate with the Eclipse Platform and the Eclipse Modelling Framework.	SHALL	Yes	Core functionality.
D52	The collaboration framework should integrate with the Eclipse Team API.	SHOULD	Yes	only for offline (Team API doesn't have support for offline).
D53	The collaboration framework should integrate with RDF models/triple stores and property graph models/graph databases.	SHOULD	No	Prototype is EMF only: supporting other modelling technologies was a lower priority requirement.

Continued on next page

Continued from previous page

Req. No.	Requirement	Priority	Satisfied?	Justification
D54	The collaboration framework should tolerate network connectivity loss gracefully in online collaboration sessions.	SHOULD	Partially	Graceful connection loss handling depends on the underlying technologies chosen for the implementation.
D55	The collaboration framework should scale to support at least 10 real or 50 simulated concurrent users.	SHOULD	Yes	Confirmed through experiments between R&D partner and use case partner.
D56	The collaboration framework may integrate with Microsoft Word or MatLab Simulink models.	MAY	Yes	Possible via Massif.
D57	Model indexing features shall be available as a service, so that they can be used from MONDO editors.	SHALL	Yes	MONDO Server exposes a service-oriented API, which includes Hawk.
D58	Optimizations shall be available to address different scalability aspects of the expected instance models for storage.	SHALL	Yes	Hawk provides abstractions that allow using fragmented models that do not fit into memory all at once, and the derived/indexed attributes speed up lookups.
D59	A scalable model indexing framework shall be provided.	SHALL	Yes	Hawk is the scalable model indexing framework.
D60	The model indexing framework shall provide a model access layer for elementary queries.	SHALL	Yes	Hawk has three model access layers: one based on Epsilon for local indices, two based on EMF for local and remote indices.
D61	The model indexing framework and the query and transformation framework shall provide transaction management.	SHALL	Yes	Hawk relies on the underlying DB transactional semantics for consistency. VIATRA3 has built-in support for transaction management.
D62	A framework shall be provided that supports indexing, access and querying of large-scale models stored in remote VCS repositories.	SHALL	Yes	Hawk includes components for accessing SVN and Git repositories.

Continued on next page

Continued from previous page

Req. No.	Requirement	Priority	Satisfied?	Justification
D63	The indexing framework shall provide support for indexing models stored in SVN repositories.	SHALL	Yes	See D62.
D64	The indexing framework shall provide support for indexing models stored in Git repositories.	SHALL	Yes	See D62.
D65	The indexing framework shall only require read-access to VCS repositories.	SHALL	Yes	Hawk only uses read-only APIs for SVN repositories, and it accesses Git clones simply by reading their files.
D66	The indexing framework shall support indexing, access and querying of large-scale models stored in a local Eclipse workspace.	SHALL	Yes	Hawk includes a Workspace connector that accesses the Eclipse workspace and reacts immediately to any changes.
D67	The indexing framework shall expose index querying and configuration capabilities as a web service API.	SHALL	Yes	Hawk includes a Thrift API that can not only query, but also configure the indexing. The user interface for managing remote Hawk indexes is the same as the local one, and all operations are backed by the API.
D68	The indexing framework API shall provide support for authentication.	SHALL	Yes	All the HTTP-based Thrift APIs in the MONDO Server can be secured through Apache Shiro, by enabling a configuration option.
D69	The indexing framework API shall enable clients to modify the configured set of monitored VCS repositories.	SHALL	Yes	The Thrift API includes methods for adding/removing repositories.
D70	The indexing framework API shall enable clients to query the index.	SHALL	Yes	The Thrift API includes methods for querying indices.

Continued on next page

Continued from previous page

Req. No.	Requirement	Priority	Satisfied?	Justification
D71	The indexing framework API shall enable clients to receive file-level notification events when indexed models change.	SHALL	Yes	The Thrift API has a subscribeToChanges method, which will produce an Artemis message queue that clients can subscribe to. This queue produces not only file-based changes, but also model-level changes (add/remove element, set/unset attribute and so forth).
D72	The indexing framework shall provide support for models captured in EMF-compatible XMI.	SHALL	Yes	Hawk can index XMI files by reusing the standard EMF parser.
D73	The indexing framework shall provide support for models captured in Express.	SHALL	Yes	Hawk can index IFC STEP/-EXPRESS files by reusing the BIMserver EMF-compatible parser.
D74	The indexing framework shall provide support for models captured using Modelio.	SHALL	Yes	Hawk can index Modelio files in two files: by reusing the Modelio XMI export, or by parsing directly the .exml files.
D75	A high-performance model persistence format alternative to EMF XMI shall be provided.	SHALL	Yes	Three alternatives have been provided: NeoEMF provides a high-performance alternative storage layer for EMF based on different persistence backends (for different intentions), the Hawk Thrift API defines an efficient network encoding, and SmartSAX provides a more efficient model parser for read-only operations.
D76	The high-performance format shall be as expressive as EMF XMI.	SHALL	Yes	All options are fully EMF compatible.

Continued on next page

Continued from previous page

Req. No.	Requirement	Priority	Satisfied?	Justification
D77	The high-performance format shall support cross-references between elements contained in different files.	SHALL	Yes	See D77.
D78	The high-performance format shall support lazy model element loading.	SHALL	Yes	NeoEMF generates EClass implementations that use lazy loading for going through references. The Hawk Thrift API supports lazy loading modes.
D79	Clients shall be able to migrate between EMF XMI and the high-performance format in a lossless manner.	SHALL	Yes	Hawk EMF resource and NeoEMF resources can be saved back to XMI. The SmartXMI loader always works with the original XMI file.
D80	The scalable model indexing framework should accept information about the usage of the models by the query/transformation layer.	SHOULD	Yes	Users of Hawk can specify derived/indexed attributes based on their queries/transformations and have them sped up.
D81	The indexing framework should respect access control rules of the underlying VCS repositories.	SHOULD	Yes	The SVN repository uses the same HTTP API as any regular client and can be provided with a username and password. Git repositories are accessed through local clones, which are assumed to be managed by the user normally.
D82	Locally deployed instances of the indexing framework should be able to communicate with remote instances to expand the scope of queries.	SHOULD	No	Lower priority requirement.
D83	The indexing framework API should enable clients to receive model-element-level notification events when indexed models change.	SHOULD	Yes	See D71.

Continued on next page

Continued from previous page

Req. No.	Requirement	Priority	Satisfied?	Justification
D84	The high-performance format should enable models to be persisted as a single file.	SHOULD	Yes	Using the MapDB backend, NeoEMF could save the model to a single file. SmartSAX can operate with the original XMI file.
D85	Clients should be able to query/modify models captured using the high-performance format using EMF's Resource interface.	SHOULD	Yes	NeoEMF allows for querying and modifying models through a Resource interface. Both the Hawk Thrift API and SmartSAX support querying through the EMF Resource interface, but not modification (indexes and partial loads are read-only).
D86	Architectural guidelines for technical workpackages to follow shall be provided.	SHALL	Yes	The integration approach was defined and provided in WP6 deliverables (D6.2 and onwards).
D87	A common integration technology for intra-process communication shall be specified.	SHALL	Yes	The Equinox OSGi implementation has been used to integrate all the components in a single JVM and have them talk to each other.
D88	A common integration technology for network-based communication of remote components shall be specified.	SHALL	Yes	Components are integrated through a set of Thrift APIs and Apache Artemis messaging queues.
D89	An integration plan will be specified and enforced.	SHALL	Yes	An iterative process was agreed between the partners, and multiple visits were conducted across R&D partners to initiate the integrations.
D90	Third party libraries used in workpackages 2-5 shall be monitored for compliance with EPL.	SHALL	Yes	Most components are EPL-compatible, and all the EPL-incompatible components can be replaced by compatible ones (e.g. Hawk can use OrientDB instead of Neo4j for its backend, with a performance hit).

Continued on next page

Continued from previous page

Req. No.	Requirement	Priority	Satisfied?	Justification
D91	An integrated server-side platform that supports distributed model management activities shall be provided.	SHALL	Yes	A MONDO Server product was provided to all partners, based on a headless Eclipse Mars application.
D92	An integrated client-side workbench to access the server-side platform, using an API shall be provided.	SHALL	Yes	A MONDO Eclipse workbench product was provided to all partners, based on an Eclipse Mars application.

B Mondix: a generalized indexer access API prototype

As a generalization of the integration efforts between Hawk and IncQuery (as mentioned in Section 2.3), BME has drafted an indexer access API called *Mondix*, intended as a motivator for future research. The key motivating factor for this development is to provide general incremental index service to both persisted and in-memory models. Furthermore, Mondix employs a *relational data representation model*, which is more suitable for scenarios where the contents of the indexer are not presented directly to end-users, but are processed by a high-level incremental query engine that internally works with relational operations (such as IncQuery).

B.1 Key notions of Mondix

The **Mondix** API is a **uniform interface** for publishing knowledge in a **read-only** way. The Mondix interface is intended to be implemented by **model indexers** and other sources of knowledge in a scalable model management context; such Mondix backends are sometimes called **Mondixers**.

The Mondix interface is intended to be used by **Mondix clients** that extract and process knowledge published by Mondixers; a typical Mondix client could be a **complex model query engine** that performs computations and provides results to users based on information (such as knowledge about models) provided by one or more Mondixers.

B.2 Overview of data model and operations of Mondix

A Mondixer publishes knowledge in a **relational format**, as a set of named **base relations**. The names of published relations is exposed through a special **catalog relation** (with an empty string for name). Each base relation defines an ordered list of named columns. A base relation is conceptually said to contain rows, each of which is a tuple containing a value for each column of the relation.

The client can open **relational views** from a base relation. A view indicates a specific intention of the client to obtain knowledge from a base relation, and may be (should be) disposed if no longer relevant for the client. Each view specifies a *filtered relational projection* on the base relation that

1. considers only those rows of the relation that take given values at given columns (while taking arbitrary values at the other columns);
2. exposes only the given subset of relation columns in query results;
3. is a relation, i.e. without duplicate rows.

Having specified views, the client can invoke at least two **query operations** to actually obtain results: retrieving all (filtered, projected) rows of the view, and counting the number of such rows (without enumerating them all).

B.3 Guarantees and responsibilities

Mondixers publish knowledge through the Mondix API. Thus they are the components responsible for the **comprehension & uniformization** of some data source. In a scalable model management context, there may be many model repositories avaliable with heterogenous model representations, so it is the responsibility of the Mondixer to expose all this knowledge through a single API, so that the complex query engine can treat model uniformly. This might require a conceptual transformation of the data, such as ORM (Object-Relational Mapping).

Mondixers guarantee fast answers for the core query operations of the Mondix interface. This means that the execution time of such an operation can grow at most proportionally to the size of the result or return value of that operation (“big O" bound), regardless of the size of the irrelevant parts of knowledge. As an exception, a **one-time initialization cost** exceeding this bound is allowed upon the creation of a view, e.g. for constructing the appropriate lookup structures. This guarantee is backed by the intention that Mondixer are typically model indexers that are designed to provide such operations efficiently, and is mandatory so that complex query engines can operate without having to guess the internal efficiency of Mondixers.

The complex query engines access knowledge published by one or more Mondixers (potentially at different locations in a distributed system) through the Mondix interface, and process it to compute query results. It is the responsibility of such Mondix clients to allow the **formulation and interpretation and execution of complex queries** that refer to knowledge provided by the Mondixers, where the query execution phase may consult the Mondixers through the Mondix API. It is the responsibility of the query engine to plan, distribute and schedule computation steps and intermediate results; complex query processing is not expected of Mondixers.

B.4 Modularity

The Mondix API is envisioned as modular: not all Mondixers are required to provide every capability, as not all clients would need them all. The following feature packs are foreseen at the moment:

- (Core query interface)
- Change-aware Extensions
- Multi-threading Extensions
- Access Control Extensions
- Transactional Extensions
- Configuration & Lifecycle Extensions
- Metadata Extensions
- Distributed Processing Extensions

Currently, the draft API is available for the core functionality and the change-aware extensions. The rest of the extensions are to be designed in the future.

B.5 Change-aware extensions

The core functionality assumes that the knowledge indexed by the Mondixer does not change during the time frame of the activities of the client (more precisely, during the lifespan of a view). There are cases when this assumption cannot be made. The indexed model, or at least some base relations provided by the Mondixers, may experience change. In this case, it can often be useful to notify clients of this change, so that the complex query results computed from the output of the Mondixer can be kept up to date.

If a Mondix base relation is change-aware, its views are **live views**. A live view accepts **change listeners**, which will be notified of rows added to and removed from of the results of the view (updated rows are represented as a removal and a reinsertion). If a complex query engine registers such change listeners, it can update its own complex query results, thereby realizing **incremental query evaluation**.

A meaningful modification of the indexed model may involve more than one row-level change in the relational representation. Thus the client may have an **inconsistent picture** of the published knowledge when it has only partially received change notifications from the various live queries of the various change-aware base relations. Therefore change-aware relations must be published by **change-aware Mondix instance**s that provide a **consistency listener** facility. When the indexed model reaches a consistent state after a sequence of changes (of which clients have been notified via the change listeners registered at live queries), the change aware Mondix instance notifies the consistency listeners of this fact. The exact guarantees of the consistency notification (described in detail in the Javadoc) take into account the case of multiple consistency listeners that may change the model.

B.6 Code Resources

B.6.1 Mondix core repository

The following resources are available from the GitHub code repository at `https://github.com/FTSRG/mondo-Mondix`:

- *eu.mondo.mondix.interfaces*: the Mondix interfaces themselves
- *eu.mondo.mondix.client*: a JUnit test case for the default “hashmap" Mondixer
- *eu.mondo.mondix.incquery*: runtime component of Mondix bindings for IncQuery
- *eu.mondo.mondix.incquery.test*: JUnit test for the IncQuery binding

In addition to the draft API and implementation above, the GitHub code repository at `https://github.com/FTSRG/mondo-mondix-teiiddemo` contains a fully functional Mondix implementation for the *PostgreSQL* relational database, and the *JBoss Teeid* data integration platform. The PostgreSQL-specific Mondix interface also supports the change-aware extensions, based on PostgreSQL database triggers.

References

[1] R. Abernethy. *The Programmer's Guide to Apache Thrift*. Manning Publications, 2015. ISBN 9781617291814.

[2] Apache Software Foundation. Apache Ivy project site. `http://ant.apache.org/ivy/`, December 2014. Last checked: November 12, 2015.

[3] Apache Software Foundation. Apache Shiro project site. `http://shiro.apache.org/`, July 2015. Last checked: November 12, 2015.

[4] Apache Software Foundation. Apache Thrift project site. `https://thrift.apache.org/`, 2015. Last checked: November 12, 2015.

[5] Eclipse Foundation. Eclipse Jetty project site. `http://www.eclipse.org/jetty/`, October 2015. Last checked: November 12, 2015.

[6] Eclipse Foundation. Eclipse Tycho project site. `http://www.eclipse.org/tycho/`, November 2015. Last checked: November 12, 2015.

[7] Eclipse.org. EMF-IncQuery. `http://eclipse.org/incquery/`.

[8] A. Garcia-Dominguez and N. van Doorn. Ecore2Thrift project site. `https://github.com/bluezio/ecore2thrift`, November 2015.

[9] Abel Gómez, Amine Benelallam, and Massimo Tisi. Decentralized Model Persistence for Distributed Computing. In *3rd BigMDE Workshop*, L'Aquila, Italy, July 2015.

[10] Jan Kotek. MapDB project site. `http://www.mapdb.org/index.html`, October 2015. Last checked: November 12, 2015.

[11] MONDO Partners. Work Package 2: Scalable Domain Specific Languages. Deliverable 2.2: Prototype tool supporting scalable concrete visual syntax.

[12] MONDO Partners. Work Package 3: Scalable Queries and Transformations. Deliverable 3.3: Cloud-enabled query and transformation engine.

[13] MONDO Partners. Work Package 3: Scalable Queries and Transformations. Deliverable 3.4: Integration with Model Management Languages Report.

[14] MONDO Partners. Work Package 4: Scalable Collaborative Modelling. Deliverable 4.3: Prototype Tool for Collaborative Modeling - Interim Version.

[15] MONDO Partners. Work Package 4: Scalable Collaborative Modelling. Deliverable 4.4: Prototype Tool for Collaborative Modeling.

[16] MONDO Partners. Work Package 5: Scalable Model Persistence. Deliverable 5.5: Model Indexing Framework - Final Version.

[17] MONDO Partners. Work Package 5: Scalable Model Persistence. Deliverable 5.6: High-Performance Model Persistence Format.

[18] MONDO Partners. Work Package 6: Platform Integration and Evaluation. Deliverable 6.2: Integrated Platform - Interim Version.

[19] Zoltán Ujhelyi, Gábor Bergmann, Ábel Hegedüs, Ákos Horváth, Benedek Izsó, István Ráth, Zoltán Szatmári, and Dániel Varró. Emf-incquery: An integrated development environment for live model queries. *Science of Computer Programming*, 98, 02/2015 2015.