Melodic
Big data cloud

Multi-cloud Execution-ware
for Large-scale Optimised
Data-Intensive Computing

H2020-ICT-2016-2017
Leadership in Enabling and Industrial Technologies; Information and Communication Technologies

Grant Agreement No.:
731664

Duration:
1 December 2016 -
30 November 2019

www.melodic.cloud

Deliverable reference:
D3.2

Date:
4 February 2019

Responsible partner:
Simula Research Laboratory

Editor(s):
Feroz Zahid

Author(s):
Feroz Zahid, Dipesh Pradhan

Approved by:
Ernst Gunnar Gran

ISBN number:
N/A

Document URL:
http://www.melodic.cloud/deliverables/D3.2 Business logic for supporting the complete data and data-intensive application life-cycle management.pdf

Title:

D3.2 Business logic for supporting the complete data and data-intensive application life-cycle management

Abstract:

This is a report accompanying the initial software release of the Melodic Data Life-cycle Management System (DLMS). The DLMS is an Upperware component which enables holistic management of data in Cross-Cloud environments and assists in achieving data-aware application deployments in the Melodic platform. The main functionality DLMS offers includes the management of data sources in Cross-Cloud environments on behalf of Melodic users, the ability to run data life-cycle management events on commissioned nodes, and the assignment of utility values to all Cross-Cloud deployment solutions proposed by the Melodic solvers. The utility values, based on various DLMS algorithms such as affinity-aware utility calculation between application components and datasets, and inter-data centre network performance based utility calculation, are used in the utility function to optimize Cross-Cloud application deployments with respect to the data-awareness enabled by the DLMS.

In this document, we give an overview of the implementation of the DLMS and its sub-components. For technical details about the DLMS design, readers are referred to the Melodic deliverable *D2.5 Report on Data Placement and Migration Methodologies* [1].

Document	
Period Covered	M6-24
Deliverable No.	D3.2
Deliverable Title	Business logic for supporting the complete data and data-intensive application life-cycle management
Editor(s)	Feroz Zahid
Author(s)	Feroz Zahid, Dipesh Pradhan
Reviewer(s)	Jörg Domaschka, Ernst Gunnar Gran
Work Package No.	3
Work Package Title	Upper ware
Lead Beneficiary	Simula Research Laboratory
Distribution	PU
Version	1.0
Draft/Final	Final
Total No. of Pages	20

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Table of Contents

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

List of Tables

List of Figures

1 Introduction

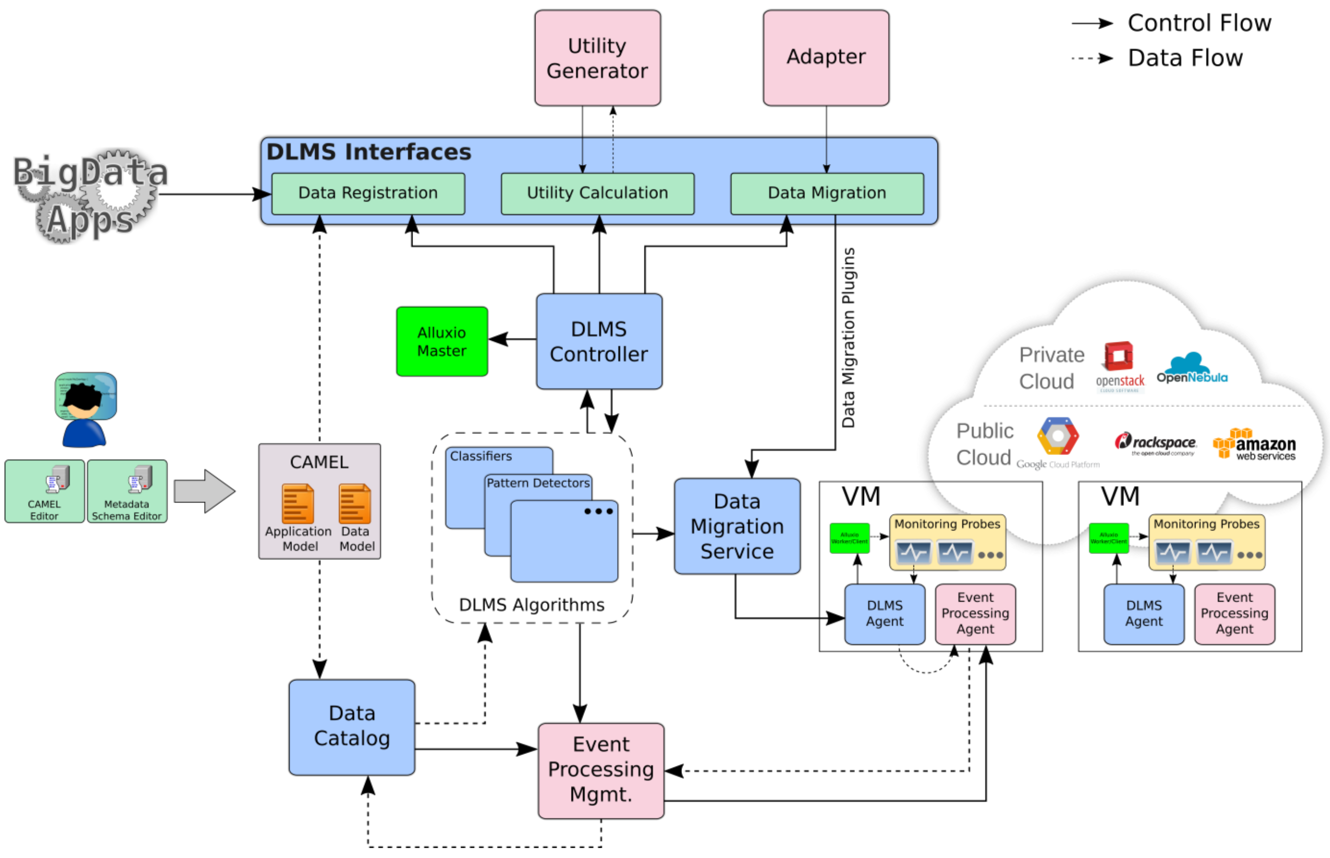

Figure 1: An overview of the DLMS Architecture

The Data Life-cycle Management System (DLMS) is a Melodic Upperware component which empowers the Melodic platform with the ability of data-aware application deployments and optimisations in Cross-Cloud environments. An overview of the DLMS architecture is presented in Figure 1. As shown in the figure, the DLMS interacts with the *Utility Generator* and the *Adapter* components of the Upperware. For each solution proposed by the Melodic Optimisation *Solvers*, the Utility Generator consults the DLMS *Controller*, which in turn employs DLMS *algorithms, to* assign utility values based on the logic implemented in the algorithm [1]. To elaborate further, the utility value represents the degree to which a solution is favourable by each DLMS algorithm, considering any data migrations required and impact on the application performance by the prescribed placement of application components and data sources.

As described above, in Melodic the data-aware application deployments and adaptations follow a two-step approach: In the first step, the CAMEL modelling language [2], is used to formally describe all the data sources and application components to be deployed by the Melodic in the Cloud, as well as their requirements and constraints, including any external data sources that the user has access to. The modelling enables the Melodic Upperware *Solvers* to consider constraints related to the data components in the calculation of candidate deployment solutions. In the second step, each candidate deployment solution is evaluated by the DLMS and is assigned utility values based on various DLMS algorithms. The DLMS algorithms assign utility values using information provided in the CAMEL DataModel [3], as well as keep in an internal knowledge-base containing information about historical data access patterns by the application components, network performance of data transfers and access costs, among others. The utility values, returned by the Utility Calculation component, are then used by the Utility Generator in the utility function for the selection of the optimal solution among the proposed candidate deployment solutions.

The current implementation of the DLMS computes utility values based on historical data access patterns reflecting affinities between application components and data sources, dataset characteristics, average network latencies and throughput between data centres, Cloud provider costs, and predictions from past DLMS decisions, as implemented by the DLMS algorithms. However, new DLMS algorithms can also be implemented and hooked to provide additional functionality if needed. Moreover, in the utility function employed by the user, the utility values returned by the DLMS are used as deem fit by the specific use-case. Further, the Adapter, which is the Upperware component responsible for analysing and validating a new deployment model and deciding on reconfiguration action tasks to be executed in a specific order, consults DLMS for specific data migrations and configurations needed for a deployment reconfiguration. This is done using the interface provided by the DLMS migration service, which is a part of the DLMS.

1.1 Scope of the Document

This document is an implementation report accompanying the initial software release of the Data Life-cycle Management System (DLMS). Intended audience are developers and end users who want a reference to the implementation details of the DLMS and its source code repositories. Most parts of the document requires knowledge about the DLMS architecture and design as described in the Melodic deliverable *D2.5* [1]. Note that the DLMS is an integrated Melodic Upperware component and comes pre-installed starting from Release 2.0 of the Melodic platform.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

2 Functionality

In the following Table 1, we highlight the main functionality offered by the DLMS.

Table 1: Summary of main DLMS functionalities

Functionality	Description
Holistic Data Management	*Data Management* The DLMS keeps track of the current locations and characteristics of each data source registered in Melodic through CAMEL. As Melodic is an optimisation platform for Cross-Cloud applications and data deployments, the data source locations (Cloud data centres) can be changed over the life-cycle of a Cloud application, in accordance with user-defined requirements and constraints. *Virtualized Data Storage Access and Data Copying / Migrations* DLMS offers virtualised data access through a storage middleware enabling applications to access data transparently across multiple data sources that may be using different storage technologies. Through the DLMS migration service, DLMS offers interfaces for data movement across data sources. For example, the data migration service makes it possible to move data transparently from an HDFS data directory to an S3 bucket. *Unified Namespace* DLMS offers a unified namespace across all internal and external file-based data sources, such as distributed file systems and key-stores. The unified namespace makes it possible to access files from any resource in a Cross-Cloud deployment using a constant unified resource identifier irrespective of the actual location of the files and the applications accessing them.
Data-aware Optimisations	*DLMS Algorithms* DLMS employs various algorithms to calculate utility values, which assists in reaching optimal data-aware deployments in Melodic. The current implementation of the DLMS algorithms compute utility values based on historical data access patterns reflecting affinities between

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

	application components and data sources, dataset characteristics, average network latencies and throughput between data centres, Cloud provider costs, and predictions from past DLMS decisions. The DLMS algorithms are discussed in detail in deliverable *D2.5* [1].
Data Access Monitoring	*DLMS Agents and Data Access Patterns* DLMS agents are deployed on each VM commissioned by the Melodic platform. Data accesses are monitored through monitors installed by the DLMS agents and reported to the event monitoring system in Melodic. Data access monitoring empowers the DLMS algorithms to assign utilities based on the observed historical data access patterns of the application components.
Cloud Network Performance Monitoring	*Latency and Bandwidth* Latency and bandwidth among Cloud data centres is measured through network performance agents installed by the DLMS agents on a selected node in each Cloud data centre used by in the application deployment.
Life-cycle Events	*Arbitrary life-cycle events* DLMS agents offer a REST interface to register arbitrary life-cycle events that can be triggered through the API. The API is also exposed to the user applications.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

3 Components and their Interaction

DLMS offers a modular extensible architecture with each component doing a designated job under the control of the *DLMSController*. The DLMS component diagram is depicted in Figure 2, where component interaction through the provided interfaces is shown. All interfaces are exposed via RESTful APIs. However, in order to reduce the overhead involved in invocation of the DLMS Controller by the Utility Generator, taking into consideration a large number of candidate solutions proposed by the Solvers, the DLMS Controller and Utility Generator interface has now also been implemented as a java library. The java library also includes an intelligent caching mechanism to further improve performance.

The *DataRegistrar* interface allows to register the data sources in the DLMS after they have been modelled, which follows *mounting* of a given data source under the unified namespace and storing the information in the DLMS database sub-component. This is automatically done by dedicated functionality in the DLMSController, which is invoked by the Melodic Upperware whenever a CAMEL model is available.

UtilityCalculationInterface offers a single method that is called by the Utility Generator with the proposed solution, in the form of *Node Candidates,* that is evaluated by the DLMS algorithms for cost assignments. Individual utility values, based on each algorithm, are then returned to the Utility Generator. This is an update from the initial DLMS design where the utility values returned by the individual DLMS algorithms were combined through a weighted function by the DLMS Controller and a single utility value was being returned to the Utility Generator. In order to provide more flexibility to the end-users so the individual DLMS algorithms can be employed through application-level utility functions, DLMS now returns separate utility values calculated by each DLMS algorithm.

A *DataMigrationInterface* interface is provided through the REST APIs with *copy/move/delete* methods dealing with the data migration between one data source to another and data purging.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

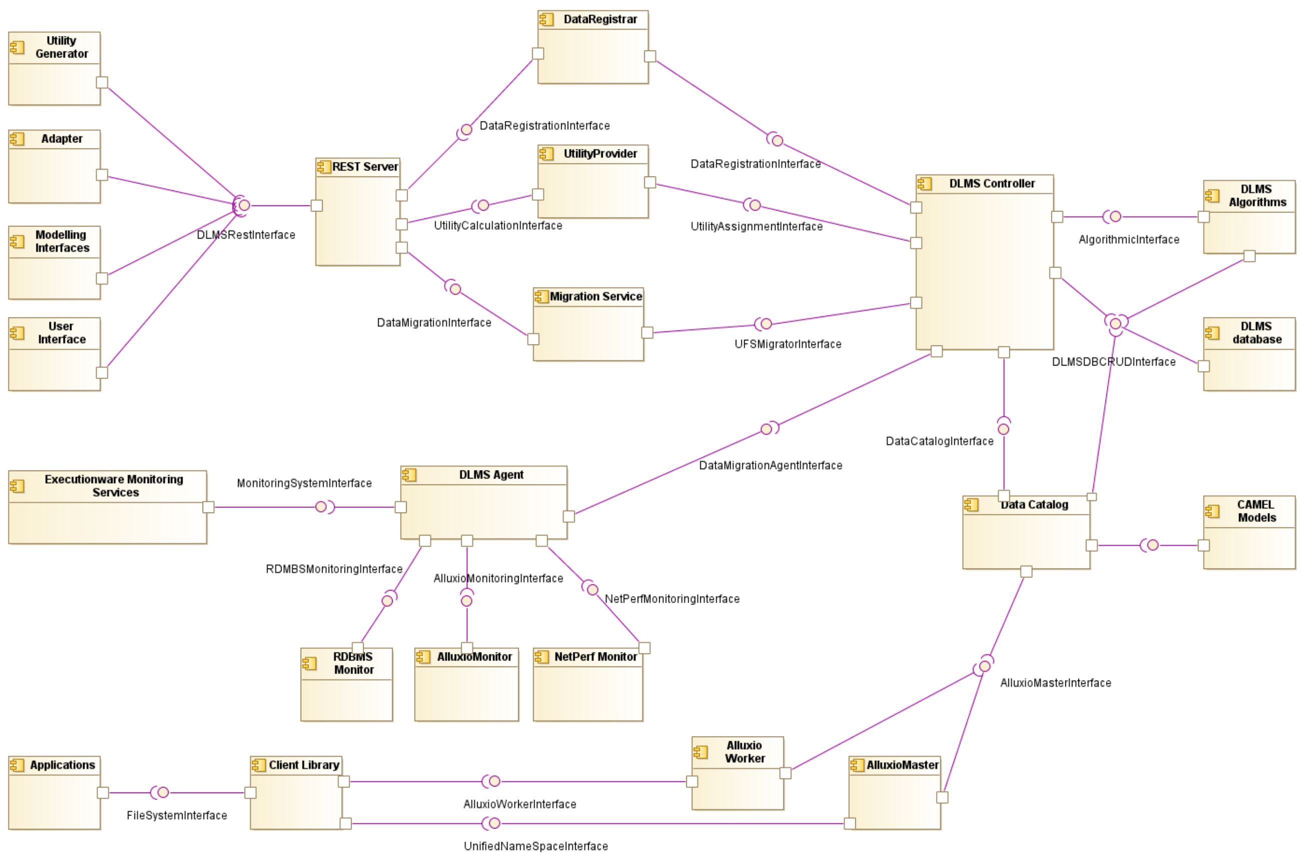

Figure 2: The DLMS component diagram

The DLMS Controller has access to the information stored in the CAMEL models, as well as historical information gathered by the DLMS agents, through the *Data Catalog*, which is also utilised by the DLMS algorithms. The monitoring information is gathered through the Executionware Monitoring Services [4], whereas the event processing agents deployed on each node process monitoring information as events and forward them to the Data Catalog which subscribes to those events [5]. The DLMS Controller also runs data life-cycle event tasks on VMs through the DLMS agents upon receiving a request through the designated REST interface.

The DLMS use Alluxio[1] (formerly Tachyon [6]) as the middleware for storage technologies. Alluxio is a rapidly growing open source memory speed virtual distributed storage system enabling big data applications to interact with data from a variety of storage systems and technologies. Applications interact with Alluxio-enabled data stores through a specialised client library which provides access to an Alluxio-based global file system. However, a client library is installed on the

[1] Alluxio – Open Source Memory Speed Virtual Distributed Storage – https://alluxio.org/

This project has received funding from the European Union's Horizon 2020
research and innovation programme under grant agreement No 731664

commissioned nodes during the deployment of the application frameworks, such as Spark, so no change in the application code is need when using popular big data frameworks.

3.1 DLMS Agents

The DLMS agents are implemented as Spring[2] boot server applications developed in Java.

For all file based data sources, monitors are based on *AlluxioMonitor* that records read/write operations to the underlying file systems. In addition, an RDBMS-specific monitor is also available based on MySQL databases, in accordance with Melodic use-case requirements. However, new monitors can be added if an unsupported data storage system is used. There is also one specialised *NetPerfMonitor,* which is a server/client script that runs on both ends of a communication channel on the selected VMs in the Clouds, to gather metrics related to the available latency and network throughput between Cloud data centres. One VM deployed in each distinct Cloud data centres is used to periodically run NetPerfMonitor and is selected as the first VM of the corresponding data centre where an application / data component is deployed by Melodic.

Alluxio Monitor

The Alluxio monitors are implemented as an integrated Java client that is called by the DLMS agent periodically.

NetPerf Monitor

The Network Performance (NetPerf) [7], [8] monitor is based on a network performance metering tool, *NetPerfMeter,* developed by the consortium partner SRL. NetPerfMeter is an open source, multi-platform transport protocol performance evaluation software. The source code of NetPerfMeter is available at git[3]. Furthermore, the open source tool iPerf[4] will potentially be integrated with DLMS, making it possible to select one of the tools for network monitoring.

RDBMS Monitor

We have implemented a reference RDBMS monitor for the MySQL databases. The MySQL monitor reports data access metrics using MySQL performance schema tables[5].

[2] http://spring.io/
[3] https://github.com/dreibh/netperfmeter
[4] https://iperf.fr/
[5] https://dev.mysql.com/doc/refman/5.7/en/performance-schema-table-descriptions.html

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

3.2 DLMS Controller

The DLMS Controller is implemented as a Spring boot Java application. Table 2 outlines classes implementing the current DLMS algorithms and their configurable parameters. Parameters are configured through a file (.properties).

Table 2: The DLMS algorithms

Class	Description	Configurable Parameters
Algo_CalculateCouplet	Calculate affinity between application component and data sources.	*Number of historical records* considered Weight of data READ and WRITE is assigned based on the method employed. Currently, *average weight* or *latest higher weight or real-time prediction* methods are implemented.
Algo_CombineValSelectedRecords	Calculate utility value based on the cloud network performance	Update can be based on *time* or *number of records* *Number of historical records* and *amount of time* considered The weight of data can be calculated based on *average weight* or *latest higher weight* methods
Algo_ClusterDataCenters	Cluster data centres to zones that will be used in computing graph similarity.	Two different algorithms to choose from: *AffinityPropagation* or *PAMClustering*. The number of clusters need to be specified for *PAMClustering*.

Algo_ComputeCost	Calculate the total cost of solutions.	
Algo_TotalUtilty	Calculates expected total utility of a proposed solution based on previous utility histories. Uses Algo_ClusterDataCenters too.	

4 DLMS and Applications

As the underlying unified file system available to the applications is based on Alluxio, applications can use client libraries provided by Alluxio for file system access. We use the convention that all file-based data sources are mounted at the time of the data source registration (which is automatically done reading the CAMEL model provided by the user), under a global namespace prefixed by **`/melodic/`**. So, a data source modelled in CAMEL under the name **`dsource1`** is mounted at the path **`/melodic/dsource1.`**

The paths in Alluxio follow **`alluxio://server:port/`** semantics. For example, a fully qualified URI of a file **`file.txt,`** which is placed on the root of the file system of data source **`dsource1`**, will be **`alluxio://127.0.0.1:19998/melodic/dsource1/file.txt.`**

However, when the Alluxio master is configured via a properties file in the Client libraries, expansion of paths take place automatically, such that **`/melodic/dsource1/file.txt`** would be resolved to the fully qualified URI automatically.

Alluxio provides several different Filesystem APIs and client libraries implemented for various programming languages. Applications mainly interact with Alluxio through the Alluxio FileSystem API for which various clients are provided (Java Client, Python, REST API client, Go Client). A Hadoop compatible FileSystem API is also provided to make sure that applications using HDFS can be used as is without the need of any change in the application code. Moreover, Alluxio also provides a FUSE-based POSIX API so applications implemented in any language like C, C++, Ruby, Perl among others, can interact with Allxuio using standard POSIX APIs like *open, write, read* and so on without any Alluxio client integration or setup.

In order to allow for transparent access to the mounted underlying storage systems by the application developed to run on big data frameworks, such as Spark and Hadoop MapReduce, some configuration needs to be done when the frameworks are being configured on the commissioned VMs by the Executionware. These configurations are done by the Executionware, and the user applications do not need to make changes to the application code written to run under these frameworks. As Melodic supports integration with Hadoop MapReduce and Spark, we show configuration examples using HDFS. However, a large number of file systems and frameworks are supported by Alluxio and can be configured through appropriate initialization scripts ran be the Executionware.

4.1 Hadoop MapReduce and Spark

- Add the following properties to the **`core-site.xml`** file for the Hadoop installation

```
<property>
  <name>fs.alluxio.impl</name>
  <value>alluxio.hadoop.FileSystem</value>
  <description>Melodic Alluxio based File System</description>
</property>
```

- Make sure that Alluxio client library is available at all the nodes
- For Spark, the Alluxio client library path needs to be given in **`conf/spark-defaults.conf`**

```
spark.driver.extraClassPath=/<PATH_TO_ALLUXIO>/client/alluxio-1.8.0-SNAPSHOT-client.jar
spark.executor.extraClassPath=/<PATH_TO_ALLUXIO>/client/alluxio-1.8.0-SNAPSHOT-client.jar
```

- Note that the above steps are automatically done through initialization scripts run by the Executionware. Once the big data frameworks are configured, no changes to the application code are necessary. However, applications can also use standard interfaces to access Allxuio if required:

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

```
FileSystem fs = FileSystem.Factory.get();
AlluxioURI path = new AlluxioURI("/myFile");
// Create a file and get its output stream
FileOutStream out = fs.createFile(path);
// Write data
out.write(...);
// Close and complete file
out.close();
```

4.2 Other Applications

Alluxio-FUSE is a feature that allows to mount the distributed Alluxio File System as a standard file system on most operating systems. By using Alluxio-FUSE, applications can access Alluxio without the need of any client integration or setup. Aluxio-FUSE provides standard POSIX based file system APIs. Any Alluxio path can be mounted using Alluxio-FUSE, as shown below. Again, such mounting can be done as part of the node setup and initialization scripts run by the Executionware.

```
integration/fuse/bin/alluxio-fuse mount mount_point [alluxio_path]
```

5 Implementation

This section discusses topics related to the code implementing the aforementioned DLMS functionalities. The DLMS consists of four main artefacts: DLMSController, DLMSWebService, DLMSUtility java library, and DLMSAgent.

5.1 License

All the source code is released under MPL 2.0[6] (Mozilla Public License), which has been approved by the Open Source Initiative in January 2012. The choice of license has resulted from deliberation over its compatibility with the GNU General Public License and licenses used by the Apache Software Foundation.

[6] https://www.mozilla.org/en-US/MPL/2.0/

5.2 Third-Party Dependencies

DLMS relies on multiple open source projects. Table 3 provides an overview of the DLMS dependencies.

Table 3: Main dependencies of DLMS

Name	Description	Link
Spring Boot	Spring boot is a framework to built stand-alone, production-grade Spring based applications.	https://spring.io/projects/spring-boot
Apache Maven	Apache Maven is a project dependency management software for Java. It is used for resolving dependencies and building Java artefacts.	https://maven.apache.org/
Lombok	Lombok is a Java library that automatically plugs into the build process to avoid repetitive code. It autogenerates Java bytecode based on annotations used in the code.	https://projectlombok.org/
Hibernate	Hibernate is an object relational mapper for Java. It is used to abstract the database access logic from an individual database.	http://hibernate.org/
Apache Commons	Apache Commons is an Apache project focused on providing reusable, open source Java software.	https://commons.apache.org/
Gson	Gson is a Java library that can be used to convert Java objects into their JSON representation.	https://github.com/google/gson
Java Message Service (JMS)	JMS API provides a common way for Java programs to create, send, receive, and read an enterprise messaging system's messages.	https://docs.oracle.com/javaee/6/api/javax/jms/package-summary.html
Apache ActiveMQ	Apache ActiveMQ is a powerful open source messaging and integration patterns server. It is written in Java together with a full JMS.	http://activemq.apache.org/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

MySQL	MySQL is an open source relational database management system.	https://www.mysql.com/
JUnit	JUnit is a unit testing framework for Java. It uses annotation to identify methods that specify a test.	https://junit.org/
Mockito	Mockito is an open source testing framework for Java. It allows the user to create and configure mock objectives, and it can be used in conjunction with JUnit.	https://site.mockito.org/
Cloudiator	Cloudiator is a multi-tenant, cross-cloud orchestration framework. The Cloudiator component installs DLMS agents on the nodes provisioned in the clouds.	http://cloudiator.org/
Metrics	Metrics is a java library, which gives unparalleled insight into what the code does in production.	https://metrics.dropwizard.io/
Alluxio	Alluxio is a virtual distributed storage system. It bridges the gap between computation frameworks and storage systems, enabling applications to connect to numerous storage systems through a common interface.	https://www.alluxio.org/
H2 Database	H2 Database is an in-memory relational database management system used by DLMS agents to store life-cycle events and corresponding commands to run on the call.	http://www.h2database.com/html/main.html
Eclipse Modelling Framework	The Eclipse Modelling Framework (EMF) is used to provide the Object Constraint Language implementation required in the matchmaking agent.	https://www.eclipse.org/modeling/emf/
Javax.xml.bind	Javax.xml.bind provides a runtime-binding framework for client applications including unmarshalling, marshalling, and validation capabilities.	https://docs.oracle.com/javase/7/docs/api/javax/xml/bind/package-summary.html

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

5.3 Source Code Repositories

The source code of DLMS is hosted on bitbucket[7] under the organisation dlms and dlmsAgent. Table 4 provides pointers to the repositories.

Table 4: Repositories

Repository Folder	Artefact	Link
dlms	DLMSController DLMSWebService DLMSUtility	https://bitbucket.7bulls.eu/projects/MEL/repos/upperware/browse/dlms
dlmsAgent	DLMSAgent	https://bitbucket.7bulls.eu/projects/MEL/repos/upperware/browse/dlmsAgent

5.4 Documentation

Pointers to all Melodic documentation are available through the Melodic website[8]. The technical details about the DLMS design are described in deliverable *D2.5* [1].

5.5 Installation and Packaging

The DLMS comes bundled with the Melodic installation package. To ease the installation process, DLMS uses Docker for both DLMSWebService and DLMSController. The DLMSWebService also encapsulates interfaces related to the DLMS Migration Service. The DLMSController docker encapsulates all the required DLMS algorithms.

The DLMS agents are deployed by Cloudiator. For details about the Melodic Executionware, please consult [9] .

[7] https://bitbucket.7bulls.eu/projects/MEL/repos
[8] https://melodic.cloud/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

5.6 Continuous Integration Platform

The DLMS uses the continuous integration platform implemented for Melodic [9].

6 Summary and Outlook

As the DLMS integration with the rest of the Melodic components has been completed, this opens up the possibility of comprehensive real-world testing of the DLMS algorithms in the coming months as use-case applications make use of Melodic Release 2.0. Moreover, new DLMS algorithms will be added if needed, while the existing ones will be tuned according to the evaluation.

For data sources that are mounted under Melodic, we also plan to provide a wrapper file system interface that allows automatic translation of the file system URIs to the unified global namespace available to the system. A similar proposal is also under consideration at the Alluxio[9].

We are also considering a functionality for the final Melodic platform release that will exploit the data sanitization support offered as a service by the used cloud providers (in case they support it) through life-cycle events for purging and deleting data securely from cloud providers.

[9] https://alluxio.atlassian.net/browse/ALLUXIO-3287

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

References

[1] K. Kritikos, F. Zahid, S. Mazumdar, D. Seybold, and Y. Verginadis, “D2.5 Report on Data Placement and Migration Methodologies.”, The Melodic H2020 Project Deliverable D2.5, 2018.

[2] A. Rossini *et al.*, “The cloud application modelling and execution language (CAMEL),” (10.18725/OPARU-4339) May 2017.

[3] Y. Verginadis *et al.*, “D2.2 Architecture and Initial Feature Definitions.” 2018, The Melodic H2020 Project Deliverable D2.2.

[4] D. Baur and D. Seybold, “D4.1 Provider agnostic interface definition & mapping cycle.” The Melodic H2020 Project Deliverable D4.1, 2018.

[5] C. Chalaris *et al.*, “D3.4 Workload optimisation recommendation and adaptation enactment.”, The Melodic H2020 Project Deliverable D3.4, 2019.

[6] H. Li, A. Ghodsi, M. Zaharia, S. Shenker, and I. Stoica, “Tachyon: Reliable, Memory Speed Storage for Cluster Computing Frameworks,” in *Proceedings of the ACM Symposium on Cloud Computing*, New York, NY, USA, 2014, pp. 6:1–6:15.

[7] T. Dreibholz, “NetPerfMeter: A Network Performance Metering Tool,” *Multipath TCP Blog*, Sep. 2015.

[8] T. Dreibholz, M. Becke, H. Adhari, and E. P. Rathgeb, “Evaluation of A New Multipath Congestion Control Scheme using the NetPerfMeter Tool-Chain,” in *Proceedings of the 19th IEEE International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, Hvar, Dalmacija/Croatia, 2011, pp. 1–6.

[9] P. Skrzypek, “D5.05 Continuous Integration Platform and Guidelines.”, The Melodic H2020 Project Deliverable D5.05, 2018.

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 731664