www.qualiMaster.eu

QualiMaster

A configurable real-time Data Processing Infrastructure mastering autonomous Quality Adaptation

Grant Agreement No. 619525

Deliverable D5.2

Work-package	WP5: The QualiMaster Adaptive Real-Time Stream Processing Infrastructure
Deliverable	D5.2: Basic QualiMaster Infrastructure
Deliverable Leader	TSI
Quality Assessors	Holger Arndt
Estimation of PM spent	13
Dissemination level	PU
Delivery date in Annex I	31.12.2014
Actual delivery date	30.12.2014
Revisions	5
Status	Final
Keywords:	Platform, infrastructure, adaptation, stream processing

Disclaimer

List of Authors

Partner Acronym	Authors
LUH	Mohammad Alrifai
	Claudia Niederée
SUH	Holger Eichelberger
	Cui Qin
TSI	Ekaterini Ioannou
	Evripidis Sotiriadis
	Gregory Chrysos
MAX	Steve Hutt
SPRING	Stefan Burkard

Table of Contents

Executive summary

This deliverable is a refinement of D5.1, which reported the set-up of the QualiMaster infrastructure, and it describes the stabilized design and initial version of the infrastructure. The current status of the infrastructure, as described in this deliverables, provides the (technical) foundation on which the various QualiMaster components and tools can be integrated.

More specifically, the D5.2 deliverable provides the architecture of the QualiMaster infrastructure, which has been updated for reflecting the additional/updated requirements and use cases reported in the D1.2 deliverable. This includes the detailed description of the layers composing the QualiMaster infrastructure as well as a discussion of communication between these layers. The deliverable also contains a plan for validating and evaluating the infrastructure relying on performance experiments and measurements. Furthermore, it reports the status of the priority pipeline, illustrating the current implementation and integration aspects of the QualiMaster infrastructure.

Note that D5.2 is the second deliverable of WP5 and it will be followed by two additional deliverables. In particular D5.3 "QualiMaster Infrastructure V1" and D5.4 "QualiMaster Infrastructure V2". These future WP5 deliverables will document our efforts on improving and extending the QualiMaster infrastructure.

1 Introduction

The D5.2 deliverable presents and discusses the stabilized design and initial version of the QualiMaster infrastructure. This in an important step for the QualiMaster project, since this version of the infrastructure forms the basis on which the work packages can create their components and tools. Thus, research and development in the other work packages can progress smoothly.

More specifically, this deliverable describes the QualiMaster infrastructure, which has been updated during the previous months in order to reflect the additional requirements and use cases arising in WP1. We follow an adapted version of the frequently applied 4+1 architecture view model of Kruchten [K95], i.e., we describe the logical view (mostly static architecture in terms of class diagram), the process view (also called dynamic architecture, i.e., the communication paths and patterns), the physical view (in terms of the deployment of the distributed architecture) and aspects of the development view (in particular the platform instantiation process as well as the testing and evaluation approach). The fifth view of Kruchten, i.e., use case or usage scenarios, are described in D1.2. For the static architecture, we describe each lifecycle phases (i.e., configuration, runtime, and startup) with their incorporated layers and the utilized repositories (i.e., pipeline repository and processing elements repository). For the dynamic architecture, i.e., communication inside the layers and components, we provide the set of possible communication cases as well as more concrete examples for the most important communication paths, for example communication during the normal pipeline processing. Furthermore, we describe the (distributed) deployment of the architecture as well as the infrastructure derivation process from the QualiMaster configuration.

In addition to the description of the QualiMaster infrastructure, we also report our plan for validating and evaluating the infrastructure. This relies on both performance experiments and measurements. Furthermore, we report the status of the priority pipeline, and through this illustrate the current status of the infrastructure with respect to the actual implementation and the integration of the various tools, applications, layers, and components composing it.

This deliverable is a refinement of the D5.1 deliverable that provided the set-up of the QualiMaster infrastructure. Thus, we do not provide the basic information related to the infrastructure since this has been included in D5.1, such as the benefits and functionalities we expect to have given the selected external systems. We primarily focus on providing the stabilized design of the infrastructure, discussing its current integration and implementation status, and outlining our plan for its validation and evaluation.

The description of the QualiMaster infrastructure, as provided in this deliverable, mainly depicts the current status that will evolve during the project. Thus, it could be that some of the layers, tool and external systems will change due to the constant monitoring of alternatives, e.g., decisions for execution systems. These extensions and modifications will be reported in deliverables D5.2, D5.3, and D5.4.

In the following paragraphs we discuss the relation to other deliverables that are submitted on the same times (Section 1.1) and then the topics that are presented and discussed in the following sections (Section 1.2).

1.1 Relation of other deliverables

The deliverable D5.2 is closely related to several other deliverables that are also submitted on the same time (i.e., month 12 of the project). The following paragraphs list these deliverables and denote the information they contain that is considered complementary to the one in D5.2.

Deliverable D1.2 - "Full Use Cases and Requirements"

Reports the requirements and use cases that QualiMaster needs to satisfy. D1.2 is actually a continuation of deliverable D1.1 "Initial Use Cases and Requirements" that was used for creating the initial version of the QualiMaster infrastructure. Similarly, we used the information in D1.2 for updating the architecture in order to satisfy additional/updated requirements.

Deliverable D2.1 - "Approach for Scalable, Quality-aware Data Processing"

Describes the foundations for quality-aware data processing as well as the families of algorithms that will be considered. The interactions between WP2 and WP5 ensured that these families of algorithms, and the incorporated algorithms, follow the developing guidelines, and that the infrastructure has all the required capabilities for executing these algorithms. Note that D5.2 simply refers to the current collection of families and algorithms, but D2.1 provides their description with the related implementation aspects.

Deliverable D3.1 - "Translation of Data Processing Algorithms to Hardware"

Describes the foundations for translating the processing algorithms to hardware. WP5 used this information during the design of the architecture, which was reported in the D5.1 deliverable. For completion, D5.2 provided an overview of the communication with the hardware, performed within the Execution Layer. However, the details for this communication along with the translation of the processing algorithms to hardware are provided in D3.1.

Deliverable D4.1 - "Quality-aware Processing Pipeline Modeling"

Provides the fundamental approach for modeling the quality-aware configuration and adaptation of data stream processing pipelines and of the processing elements. This is performed using the Monitoring, Coordination, and Adaptation Layers, and D5.2 explains the main processing performed in these layers as well as the communication between them. The actual quality-aware processing performed in QualiMaster is described and discussed in the D4.1 deliverable.

1.2 Structure of the deliverable

The remaining sections of this deliverable are as follows: Section 0 provides a detailed description of the updated QualiMaster infrastructure. This includes a discussion of the related requirements, overview of applications built on top of QualiMaster, description of the static architecture according to the lifecycle phases (i.e., configuration, runtime) and of the repositories (i.e., pipeline repository and processing elements repository). Section 3 focuses on the dynamic architecture, i.e., the communication between layers. It first discusses the possible communication cases (Section 3.1) and then provides a few communication examples, e.g., communication during the normal pipeline processing and during the starting of a pipeline (Section 3.2-0). Section 4 discusses the

QualiMaster platform instantiation process, an important part of the development view on the architecture. Then, Section 5 summarizes the architecture in terms of the (distributed) deployment of the components. Section 6 provides the status of the priority pipeline, and Section 7 our plan for validating and evaluating the infrastructure. Finally, Section 8 provides conclusions for this deliverable and gives an overview of the future plans.

2 The QualiMaster Infrastructure

This section describes the updated QualiMaster infrastructure. We consider this as an “updated version” in comparison to the infrastructure described in the D5.1 deliverable since it includes modifications focusing on reflecting the additional and modified requirements and use cases from in WP1 (reported in the D1.2. deliverable). For basic information related to the infrastructure, such as the benefits and functionalities we expect to have given the selected external systems, we refer the reader to the D5.1 deliverable.

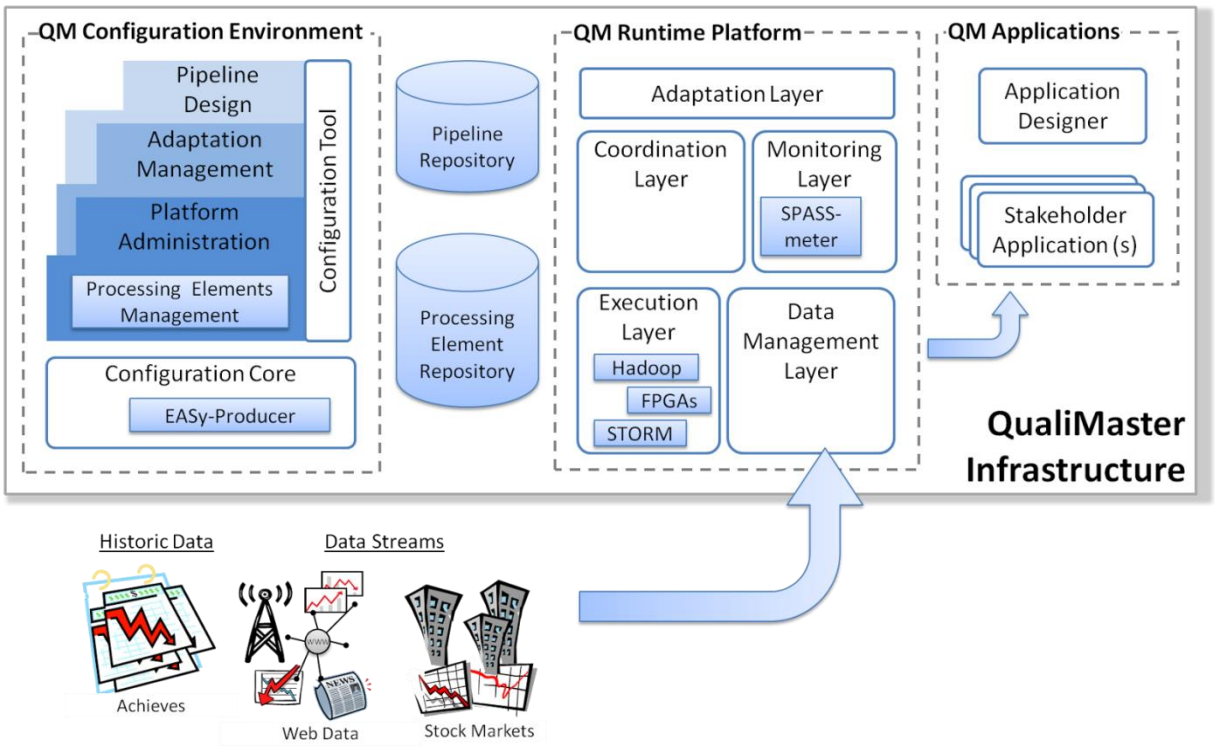

Figure 1: A graphical illustration of the architecture of the QualiMaster infrastructure.

A graphical illustration of the system’s architecture is provided in Figure 1. The QualiMaster infrastructure consists of the following four major parts, which also target different lifecycle phases (configuration, startup and runtime) of the infrastructure:

- **Configuration Environment:** The configuration part of the QualiMaster Infrastructure targets the customization and instantiation of the generic QualiMaster Infrastructure towards a domain-specific application-oriented infrastructure. Before starting the QualiMaster Infrastructure, it must be configured so that required algorithms, algorithmic families and pipelines are defined as described in the use cases of the D1.2 deliverable. This is supported by the QualiMaster Infrastructure Configuration Tool, which is based on the Configuration Meta Model (detailed in the D4.1 deliverable). As depicted in Figure 1, the Infrastructure Configuration Tool consists of three views corresponding to the three roles of Infrastructure Users, namely, Platform Administrator, Adaptation Manager and Pipeline Designer. Ultimately, the Platform Administrator utilizes the Infrastructure Configuration Tool to execute the platform instantiation process, which derives the specific software artifacts required to execute the designed pipelines from the configuration.

- **Repositories:** The repositories are used to store shared configuration-related information. More specifically, there are two repositories: the first is for the actual configuration of the infrastructure and the running pipelines in terms of models (the Pipeline Repository); the second repository is for the algorithm implementations.
- **Runtime Platform:** A configured QualiMaster Infrastructure, more precisely its runtime platform, can be started and executed in order to perform data processing as defined in the Configuration. This is supported by the runtime components of the QualiMaster infrastructure, consisting of the Execution Systems (e.g., Apache Storm), the common Data Management Layer, the Monitoring Layer for supervising the execution, the Coordination Layer for controlling and changing the execution in the Execution Systems as well as the Adaptation Layer, which is responsible for runtime decision-making and autonomously optimizing the execution of the pipelines at runtime. The runtime components utilize the Repositories to gain access to the actual configuration model and to the software artifacts implementing the pipelines.
- **Stakeholder Applications:** The QualiMaster applications utilize the results produced by certain pipelines at runtime, visualize the result information and, to a limited degree, support the user in communicating intended changes in the processing to the runtime components of the QualiMaster Infrastructure. Thus, the QualiMaster applications are domain- as well as pipeline-specific. The Application Designer is a specific program that helps the user customize the user interface of the Stakeholder Applications, e.g., to adjust the visualization so that it fits to the actual tasks at hand.

Please note that the QualiMaster Applications are mentioned here just for completeness and for providing an overview on the whole QualiMaster Infrastructure. Details on the design and realization of the QualiMaster Applications and the customization of their user interfaces will be discussed in the upcoming D6.1 deliverable.

In the following paragraphs, we elaborate on the lifecycle phases, tools, layers, and applications incorporated in the QualiMaster infrastructure. More specifically, Section 2.1 discusses the requirements that drove incorporating updates in the infrastructure defined in the D5.1 deliverable. Section 2.1 discussed the configuration environment, Section 2.3 runtime platform, and Section 2.4 presents the repositories used by these layers. Then, Section 2.5 introduces the external interfaces. Finally, Section 2.6 provides short overviews of the QualiMaster applications.

2.1 Requirements

The QualiMaster Infrastructure presented in this deliverable has modifications and updates with respect to the infrastructure described in the D5.1 deliverable. These modifications/updates were incorporated in the QualiMaster Infrastructure in order to reflect the requirements discussed in the D1.1 deliverable, which is actually a refinement of the requirements of the D1.1 deliverable that we used for the initial version of the QualiMaster Infrastructure.

The following list briefly reports the most important requirements that are captured by the updated/modified infrastructure, i.e., the one described in this deliverable:

- Modifications in the QualiMaster Configuration Platform, and more specifically integrating the three separate configuration tools into one single QualiMaster Infrastructure Configuration Tool.

- A larger collection of QualiMaster applications, since there is the need for providing sophisticated visualization that will allow the interpretation of the results.
- Modifications in the lifecycle phases, and especially the layers of the QualiMaster Runtime Platform in order to allow better communication and more efficient handling of volatilite and testing.

2.2 QualiMaster Configuration Environment

The configuration lifecycle phase is responsible for specifying the values of the configurable elements of the Configuration Meta Model for a certain application setting (e.g., financial data processing) as well as the pipelines to be executed. This basically creates an instantiated version of the QualiMaster platform that also contains the pipelines corresponding to the particular application. Specifying the configuration is performed using an application, which consists of the following views corresponding to the Infrastructure User groups defined in D1.2:

- **Pipeline Configuration**: This view allows the creation, modification, and deletion of a processing pipeline. The pipeline definitions perform validation of the syntax, semantics, and feasibility of the particular execution of the processing elements.
- **Adaptation Manager**: This view is responsible for configuring the adaptation aspects of the QualiMaster infrastructure. It allows defining the quality characteristics of processing elements as well as of pipelines, and defining the adaptation rules (including reactive and proactive) using the Adaptation Behavior Specification Language rt-VIL introduced in the D4.1 deliverable.
- **Platform Administration**: This view allows configuring, and modifying the incorporated hardware, such as field programmable gate arrays (FPGAs) or the general-purpose cluster nodes. For example, it allows handling the platform quality parameters, adding/modifying processing algorithms (including hardware-based algorithms), defining processing families, and starting/stopping pipelines.

Please note that in the initial version of the QualiMaster Infrastructure design we planned for three different applications according to the D1.1 deliverable. In the meantime, we decided to realize one integrated tool with different access-protected views, in order to support the Infrastructure Users in gaining a comprehensive overview on the running Infrastructure and the configuration.

Furthermore, the QualiMaster Infrastructure Configuration tool also realizes an initial **runtime view** (in direction of UC-PA3 - UC-PA9 and UC-AM5 in the D1.1 deliverable. Due to the current focus on the priority pipeline, this view actually allows the manual runtime selection of the alternative algorithms as well as a simple visualization of some monitored quality properties.

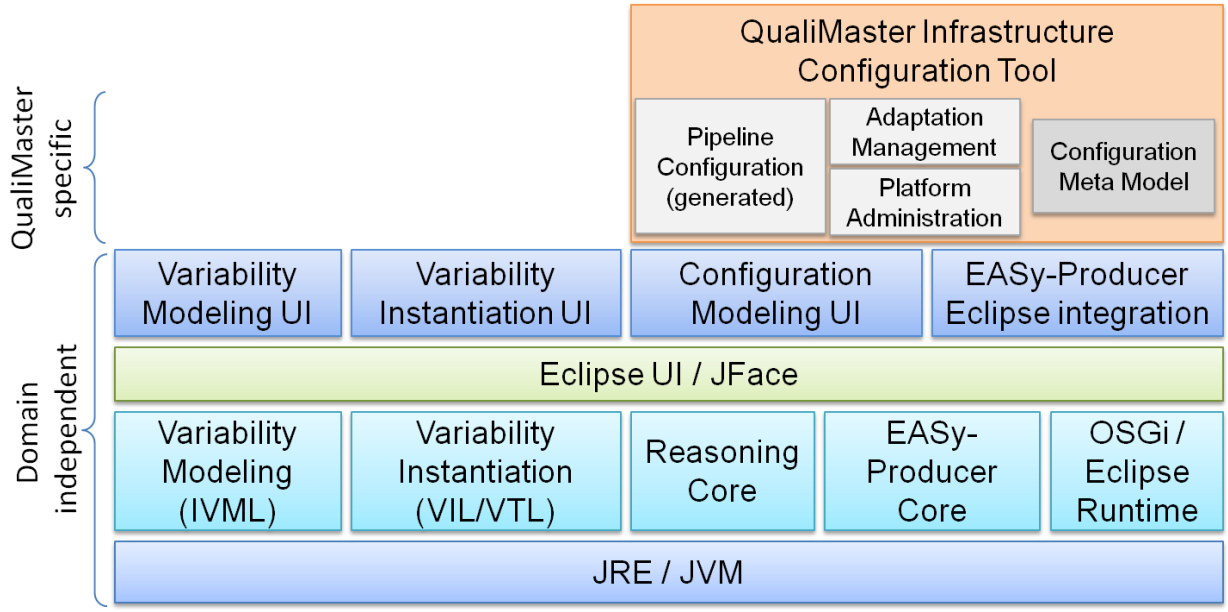

Figure 2: High-level architecture of the QualiMaster Infrastructure Configuration tool and the Configuration core.

During the configuration of the QualiMaster infrastructure, domain-specific or application-specific software components can be provided, in particular algorithm implementations, data source adapters or additional monitoring probes. These components are realized in terms of QualiMaster **extension packages**, i.e., a software archive with a QualiMaster specific manifest. The actual contents of the manifest supporting both, software-based and hardware-based algorithms is currently in development. Further versions of the QualiMaster Infrastructure Configuration tool will analyze the manifest of the provided extension package as part of the configuration activities for an individual algorithm, extract the configuration information and enter the information into the configuration. Thus, it will support the Infrastructure Users and, in particular, enable the Infrastructure Configuration Tool to determine whether algorithms fit into an algorithmic family (as described in the D4.1 deliverable, all algorithms must accept the same form of input data, provide the same form of output data and react on the same functional parameters) and whether a composed pipeline is structurally valid. This is determined in terms of a static pipeline analysis based on the input-output relation of the data sources, the algorithmic families and the data sinks. After validation, the QualiMaster Infrastructure Configuration tool will upload the provided extension package into the Processing Element repository. Please note that manifests can also contain runtime-related information such as quality profiles, which are then considered by the runtime part of the QualiMaster infrastructure.

As indicated in Figure 1, the architecture of the configuration part of the QualiMaster Infrastructure relies on the **Configuration Core**. The Configuration Core consists of EASy-Producer, a prototypical Software Product Line Engineering (SPLE) tool [EEK+14] as already introduced in the D5.1 deliverable. Recent improvements of EASy-Producer made during the QualiMaster project are detailed in the D4.1 deliverable. EASy-Producer is designed and realized to support domain engineers in modeling the configuration of a reusable software system as well as customizing the actual software, i.e., instantiating a software product line following according to an instantiation process. Actually, EASy-Producer provides generic support for SPLE, i.e., it is domain-independent. However, Infrastructure Users dealing with real-time data analysis are typically not

experts in SPLE, such as domain engineers or software architects. Thus, domain-independent capabilities are too generic for the QualiMaster Infrastructure. Therefore, the QualiMaster Infrastructure Configuration Tool realizes a domain-specific configuration frontend on top of EASy-Producer as illustrated in Figure 2[1]. The QualiMaster Infrastructure Configuration tool uses the generic SPLE tool as a library and adds specific editors and configuration logic in order to support the QualiMaster Infrastructure users. These editors allow the Infrastructure Users to configure the QualiMaster infrastructure according to the Configuration Meta Model in a domain-specific manner. One particular capability of the QualiMaster Infrastructure Configuration Tool is the graphical pipeline editor, which enables the Pipeline Designer to specify data analysis pipelines in a drag-and-drop fashion following the notion of a specialized data flow graph. This specific editor is derived from the Configuration Meta Model in a model-based fashion, i.e., an editor model is derived from the pipeline part of the Configuration Meta Model and turned into software artifacts of the QualiMaster Infrastructure Configuration Tool. As required by UC-PD1 and UC-PD2, the Pipeline Designer can select the processing elements (families) as well as the data sources and sinks to compose data analysis pipelines in an integrated way relying on the configuration entered through the views of the (roles of the) other Infrastructure Users.

For obtaining the actual Configuration and the Configuration Meta Model, the QualiMaster Infrastructure Configuration Tool is equipped with a repository connector to the Pipeline Repository, actually a Subversion Repository suitable for a textual variability model. The repository connector also supports the authentication of the actual user and, thus, can derive the access permissions to the different views. Upon realization of the Extension Package manifests, a second repository connector will allow the QualiMaster Infrastructure Configuration Tool to access the Processing Elements repository as described above.

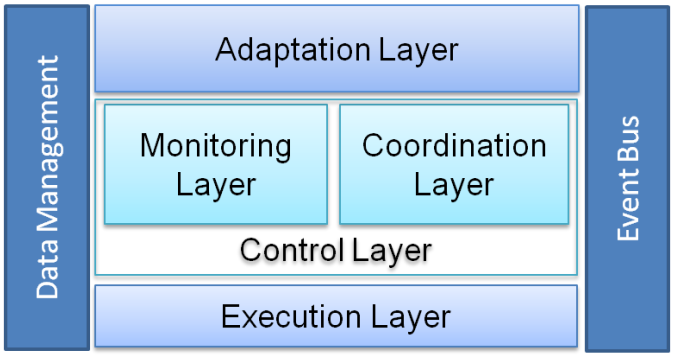

Figure 3: Architecture overview of the QualiMaster Runtime Platform

[1] We use high-level components for the illustration as the Configuration Core, i.e., EASy-Producer consists of more than 2000 classes.

2.3 QualiMaster Runtime Platform

The QualiMaster Runtime Platform controls, surveils and adapts the real-time data processing specified by the Configuration and executed by the Execution Systems. Basically, we designed a **layered architecture**, which consists of components realizing different aspects of the runtime execution. The runtime components of the QualiMaster architecture are classified in three logical layers shown in Figure 3.

We now provide an overview on the three logical layers, starting at the lowest level:

- The **Execution Layer** is responsible for the actual execution of processing tasks and includes Execution Systems, such as Storm, Hadoop and reconfigurable hardware.
- The **Control Layer** observes the actual execution while also modifying the execution on request in terms of the Monitoring Layer and the Coordination Layer introduced in the D5.1 deliverable. The Monitoring Layer obtains runtime information from the Execution Layer, aggregates, and analyzes the runtime information aiming at providing information and triggering the adaptation of the processing (if needed). The Coordination Layer is responsible for controlling the processing and for changing the processing in a coordinated way, e.g., to synchronize changes across multiple Execution Systems, such as affecting Storm and reconfigurable hardware in one step. Please note that both, Monitoring and Coordination Layer interact with the Execution Layer, can be called from the Adaptation Layer and can communicate among each other, in particular the Coordination Layer may cause changes in monitoring if requested by the Adaptation Layer.
- The **Adaptation Layer** is responsible for runtime-decision making in order to optimize the data processing. In particular, the adaptation behavior is specified in terms of a language as part of the configuration activities and, thus, supports domain-specific adaptation.

The **Data Management Layer** is a runtime component, that is available as a service to the three logical layers. More specifically, the Execution Layer must access information from the Data Management for persisting raw source data, processing results, and intermediary results; the Control Layer must be able to store quality profiles; and the Adaptation Layer must be able to access the impact and adaptation history. Due to performance reasons, the Data Management acts as a pillar as shown in Figure 3 that is available to the logical layers.

The logical layers described above must communicate to realize adaptive data processing in QualiMaster, i.e., they perform **internal communication**. As usual in a layered architecture, upper level layers can invoke lower level layers, e.g., the Adaptation Layer may call the Coordination Layer, but lower layers must not call upper layers. In addition, all logical layers can interact directly with the Data Management due to performance reasons. Furthermore, the three logical layers described above follow a strict layering, i.e., communication happens only among adjacent layers but not across layers, e.g., the Adaptation Layer does not affect directly the Execution Systems. However, Monitoring and Coordination Layer interact with the Execution Layer, can be called from the Adaptation Layer and, in particular, communicate among each other. Layering in general, and, in particular strict layering, supports separation of concerns among individual system parts, reduces dependencies, enables independent development of the individual components and even the replacement of layers if required. Please note that we discuss the internal communication among the layers in this section only for illustration of their functionality, while we detail the most important communication patterns in Section 3.

However, there is also an obvious need for communication in the other direction, e.g., the Monitoring Layer must be able to inform the Adaptation Layer about problematic situations, such as the violation of Service Level Agreements. Therefore, we introduce an **event bus**, which enables communication among layers in upwards direction without introducing dependencies. Akin to the Data Management Layer, the event bus is another pillar, all logical layers have access to as shown in Figure 3. Actually, the event bus can also be used for downwards communication, i.e., the Adaptation Layer may also send commands to the Coordination Layer in terms of events and can enable a physical distribution in the future if needed, e.g., to distribute processing load or to increase availability. Currently, the event bus is realized as an internal communication mechanism, but we will consider replacing it by a distributed event bus, such as a Java Messaging Service (JMS, e.g., ActiveMQ[2] or RabbitMQ[3]) if required.

In addition, the individual layers of the QualiMaster runtime platform also communicate with the repositories in order to obtain the actual Configuration, the implementing software artifacts or their manifests, e.g., to access quality profiles.

Finally, the runtime platform must also communicate with the environment, and, thus, perform **external communication**. In the overall QualiMaster architecture, there are two external communication partners:

- QualiMaster Infrastructure Configuration Tool: Basically, the Infrastructure Configuration Tool must communicate with the repositories in order to obtain or change the Configuration or upload extension packages. Further, the Infrastructure Configuration Tool shall commands to the runtime components, such as starting up a pipeline (UC-PA8 and UC-PA8 in D1.2). Ultimately, the Infrastructure Configuration Tool receives runtime information from the runtime components, in order support the monitoring of the execution or the adaptation (e.g., UC-AM5 in D1.2).
- Stakeholder applications: Basically, stakeholder applications receive processing results from the sinks of the running data processing pipelines. As described in the D1.1 deliverable, the Stakeholder Applications may also send user triggers to the platform in order to change the processing, e.g., to focus on a certain market. These user triggers are considered as adaptation triggers, but actually of lower priority compared with internal triggers such as SLA violation as described in the D4.1 deliverable.

In contrast to the Repositories and the Infrastructure Configuration Tool (which are external to the runtime components), Stakeholder Applications run on stakeholder side. Therefore, future versions of the QualiMaster Infrastructure will provide an external visible server, which acts as data access point to the Stakeholder Applications and enables access protection.

In the remainder of this section, we detail the static architecture of the individual layers, i.e., we provide class diagrams or hierarchy diagrams denoting the interfaces of the layers. We also discuss the layers from low-level to high-level layers. Thereby, we will already provide some details on the communication of the layers in order to explain the design of their interfaces. Note that in Section 3 we focus on the dynamic architecture, in particular the communication among the layers.

[2] http://activemq.apache.org/
[3] http://www.rabbitmq.com/

2.3.1 Data Management Layer

This layer manages the data streams given to the infrastructure for processing. This includes managing the sources and sinks of processing pipelines. Here, we distinguish between the management of raw data (input stream items given to pipelines) and the management of processed data (including final results). Raw data is used as input to several pipelines running on the QualiMaster platform. Thus, storing raw data is a common task in order to avoid repeated storage of the same information. However, storing all raw input data may not be feasible in all application cases. Thus, the QualiMaster infrastructure through the QualiMaster Configuration provides support to tailor the Data Management Layer in terms of configuration options, e.g., disable storage for a certain data source or apply data management strategies. More detailed information on the data management strategies is provided in deliverable D4.1 as part of the configuration of the Data Management Layer. We aim at typesafe compositions of data processing pipelines in the Configuration and, in particular, in the implementation. Here, domain-specific algorithms as well as data source and sink implementations are realized by potentially external Algorithm Providers. Thus, algorithm families, data sources and sinks are specified by interfaces using input and output types defined in the Configuration.

The Data Management Layer provides also mechanisms to store intermediate processing data produced by the processing elements of the data processing pipelines. Therefore, the Pipeline Designer is able to specify as part of the pipeline design, i.e., the Configuration, how and what data shall actually be stored in terms of a generic built-in processing element, which passes the related data tuples to the Data Management Layer for storing. In addition, the Data Management Layer also manages data produced by the infrastructure, in particular the quality profiles as well as the adaptation log indicating point in time and success of individual adaptation activities. Please note that internal and intermediate is handled internally by the QualiMaster infrastructure, and, thus, data management can be done in a more generic fashion.

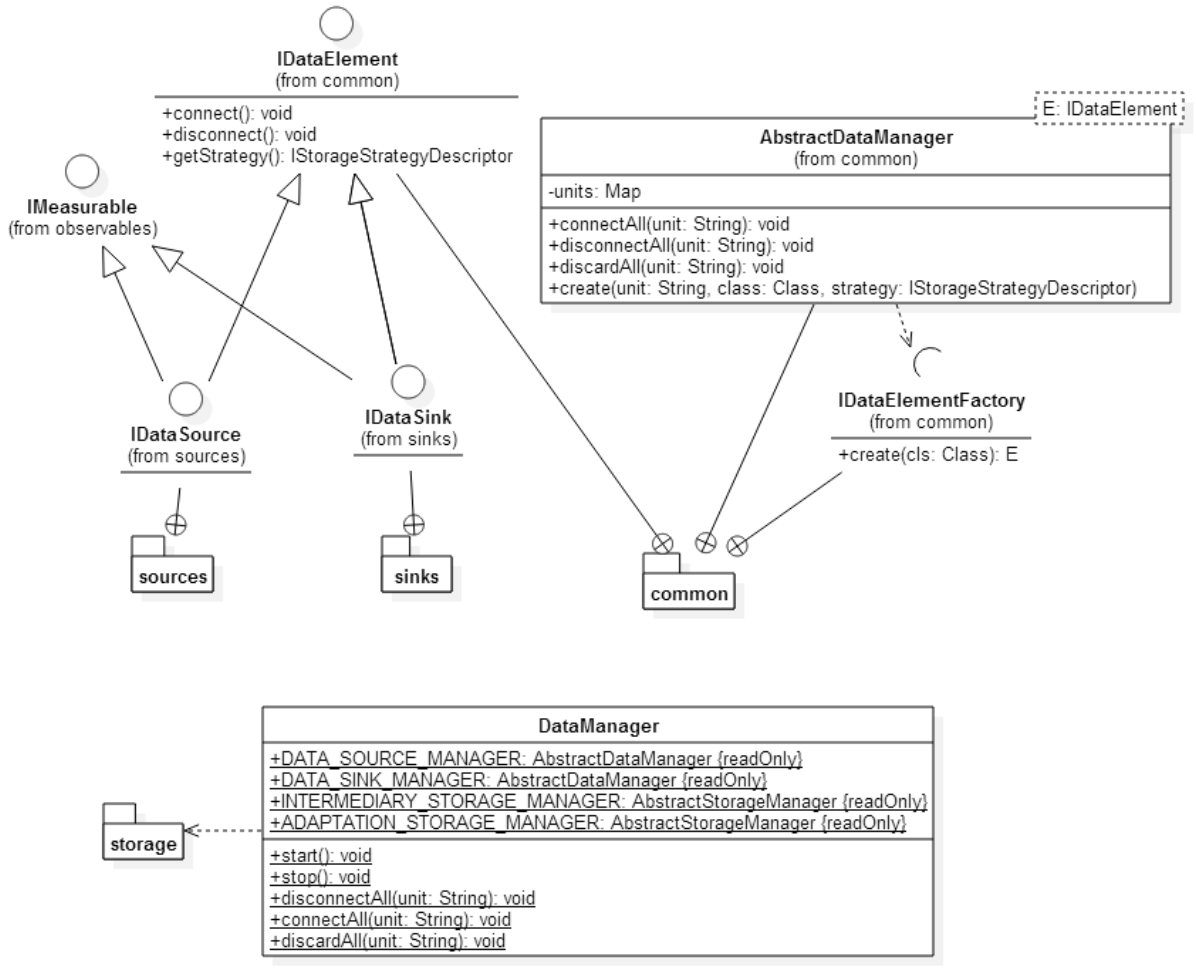

Figure 4: Overview class diagram on the Data Management Layer.

These different functionalities are reflected in the design of the Data Management Layer. An overview is illustrated in Figure 4. Basically, the Data Management Layer consists of several common interfaces and reusable classes, in particular `IDataElement`, representing a portion of data controlled by a storage strategy (more precisely a storage strategy descriptor, which can be used without knowing the underlying storage implementation; storage strategies are not detailed in Figure 4) and the `AbstractDataManager`, which provides access to instances of `IDataElement`. In particular, the `AbstractDataManager` allows managing (connect, disconnect, discard) `IDataElement`s for entire pipelines or other data processing "units". Basically, data elements such as sources and sinks can be created akin to an usual class. However, the Algorithm Provider shall not interfere with the transparent storage mechanisms, e.g., for data sources and sinks, and in particular not with future optimizations of the data management such as distributed buffering. This additional functionality is provided as an add-on to the implementations made by Algorithm Providers. Thus, client code does not create data elements rather than requesting an instance from a data manager via the `AbstractDataManager`, which utilizes a factory [GHJ+00] (`IDataElementFactory`) to create specific instances, e.g., through wrapping or (dynamic) proxying the original implementation [GHJ+00].

The common functionality described above is then specialized in terms of data sources (`IDataSource`) and data sinks (`IDataSink`). In particular, data sources and data sinks are subject to quality monitoring as described along with the quality taxonomy in D4.1. Thus, in contrast to the generic `IDataElement`, the interfaces for data sources and data sinks also implement a measurement interface (`IMeasurable`), which is detailed along with the Monitoring Layer in Section 2.3.4. Please note that `IDataSource` and `IDataSink` act as basis interface for the domain-specific data source and sink interfaces, i.e., they do not specify methods on how to handle data. These interfaces can be derived through the QualiMaster Infrastructure Configuration Tool based on the Configuration of the Data Management Layer.

From an external point of view, the Data Management Layer is represented by the `DataManager` class, which provides access to the specific data managers for data sources and sinks, and - in particular - allows to manage (connect, disconnect and discard) all data elements for one processing unit, e.g., a pipeline. Furthermore, the `DataManager` is also responsible for receiving lifecycle events of the platform, e.g., to disconnect and discard the data elements of a stopping pipeline.

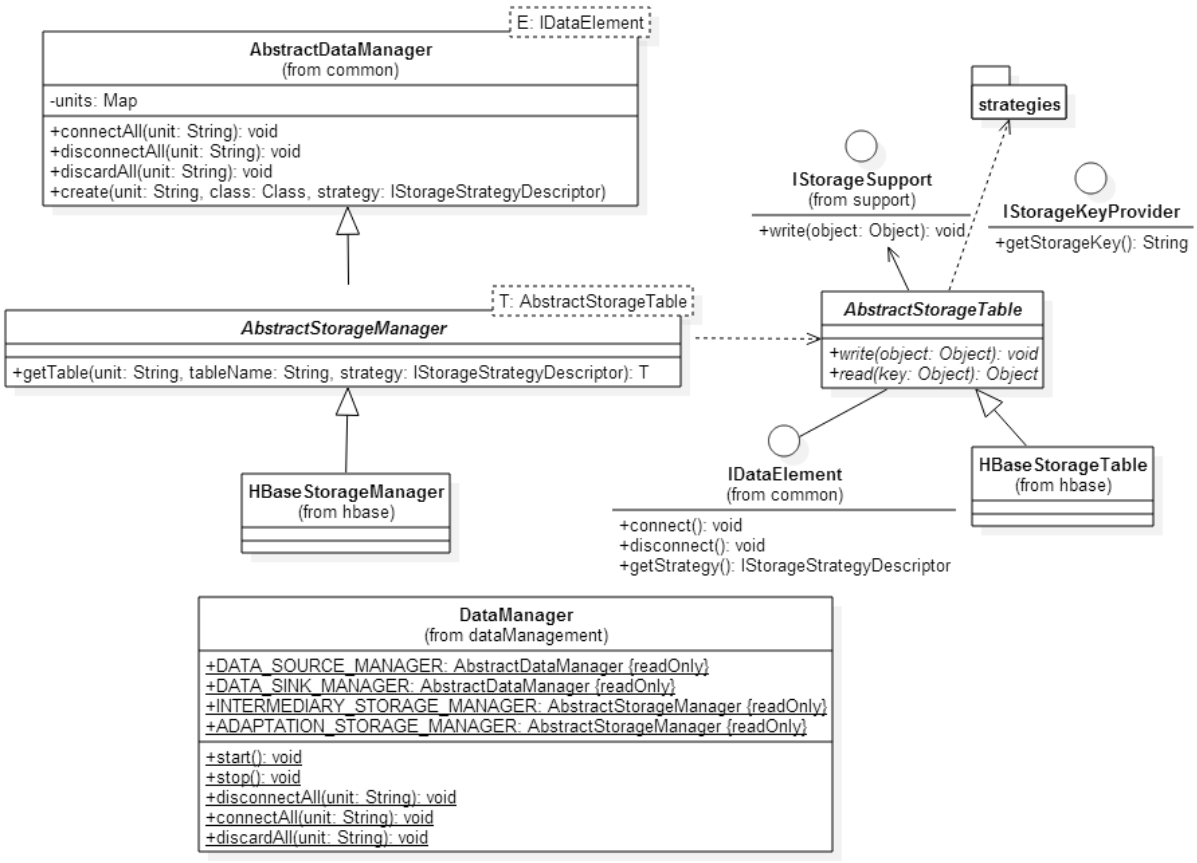

Figure 5: Detail class diagram on the storage part of the Data Management Layer.

The `DataManager` provides also access to the specific data managers for the internal storage, namely intermediary processing information and adaptation related data defined in the package `storage`. As this form of data is handled more generically (as internal), the Data Management Layer can provide a more specific implementation. An overview on the `storage` package is depicted in Figure 5.

The main classes of the storage package are the storage table (`AbstractStorageTable`) and the related data manager (`AbstractStorageManager`). The storage table represents the concept of a database table, in Big Data processing frequently following a key-value store and / or the idea of a BigTable[4]. A storage table is requested by the implementation, e.g., the Adaptation Layer or the realization of a Data Management Element in a pipeline using the name of the calling "unit", the table name and a storage strategy descriptor. A storage strategy descriptor is an abstract handle for a storage strategy, which is concretized by a specific implementation of the `AbstractStorageTable`. The storage table represents then the specified portion of data and allows writing and reading objects. By default, the actual timestamp is used as key for storing an object. However, by passing in an `IStorageKeyProvider`, the caller can define the key to be used. In data stream processing, storing intermediate data may impact the processing latency. In order to avoid this, an `AbstractStorageTable` provides access to an instance of `IStorageSupport`, which aims at low latency storage of data, e.g., via a producer-consumer pattern to separate stream processing from data management. Via `IStorageSupport` we provide an abstraction over the actual implementation, i.e., there may be even a specific realization for a certain storage strategy.

The classes and interfaces discussed above are currently realized by the default HBase[5] implementation in QualiMaster. This is indicated in Figure 5 in terms of the `HBaseStorageManager` and the `HBaseStorageTable`. Accordingly, the `DataManager` provides access to the intermediary and the adaptation stores in terms of the `AbstractStorageManager`, thus, abstracting over HBase.

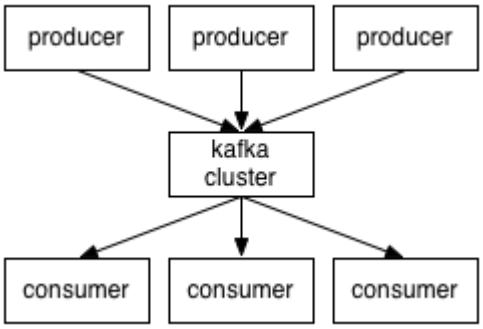

Figure 6: Message feeds publishers (producers) and subscribers (consumers) in the Apache Kafka model.

[4] http://en.wikipedia.org/wiki/BigTable
[5] http://hbase.apache.org/

In order to manage data streams of high load and high velocity in a scalable fashion, QualiMaster will employ Apache Kafka[6], which is a high-throughput distributed messaging system, i.e., it buffers and even stores messages to the underlying distributed file system in order to increase scalability and availability. Kafka maintains feeds of messages in categories called topics. The sources or publishers of the messages are called producers, while processes that subscribe to topics in order to process the feed of published messages are called consumers (see Figure 6).

In QualiMaster we plan to use Kafka to manage data streams (such as financial or social media streams) by acting as message (stream) broker between the data source implementations (`IDataSource`) and the consuming pipelines. Therefore, the `IDataElementFactory` will then provide specific instances of a data source implementation that transparently utilizes Kafka buffering and messaging capabilities in the data processing pipelines, or, if required, also other messaging and buffering frameworks. For instance, several instances of a sentiment analysis pipeline can then be used to process Twitter streams, such that each instance processes only tweets that mention a certain market player. Kafka can be used to connect to the Twitter stream and filter the streams by keywords (e.g., market player) and produce one topic for each market player. Processing pipelines can then subscribe to the respective topic as illustrated in Figure 7.

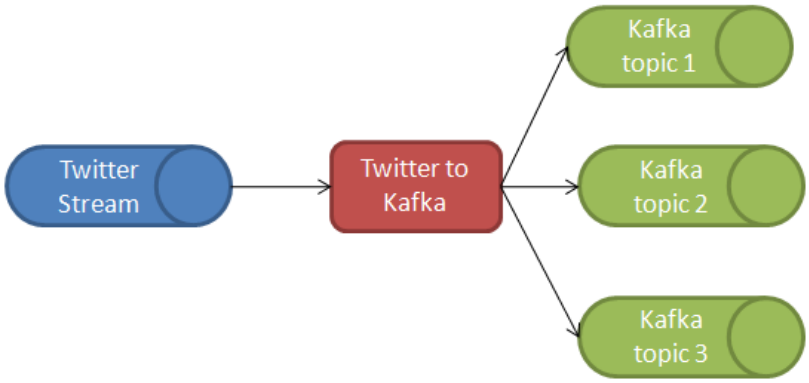

Figure 7: Example of using Apache Kafka to manage the Twitter stream and partition it into topics, one topic for each market player.

2.3.2 Execution Layer

The Execution Layer includes the systems that perform the actual execution of the data analysis algorithms, either in terms of a real-time stream processing topologies on Apache Storm, the execution of hardware-based algorithms on specialized hardware, such as MAX data flow engines, or the execution of data analysis algorithms on historical mass data in a batch manner using Apache Hadoop.

This layer is responsible of executing all the computational components that a pipeline needs for its execution. In particular, computational components include algorithms, but also data sources and sinks, or more precisely software adapters for accessing sources and sinks. The computational components are implemented in a certain format, i.e., following specific interfaces, so that they can be dynamically linked for execution, monitored by the Monitoring Layer, and

[6] http://kafka.apache.org/

execute adaptation commands originated by the Adaptation Layer. We rely on explicit interfaces, as the computational components are delivered by an Algorithm Provider, who may be external to the organization of the Infrastructure Administrator. Thus, the interfaces form a structural contract between the Infrastructure Users defining the computation, in particular the pipelines, and the computational components. Furthermore, the interfaces support a type-safe composition of the pipelines akin to higher-level frameworks in big data processing, such as (optionally) supported by Cascading[7] for Hadoop. By using adapters or wrappers, one core implementation of a computational component can be used to implement different interfaces.

In particular, (domain-specific) algorithms belong to a family of algorithms with similar functionality, i.e., implementing the same interface, but different quality tradeoffs. Components implementing algorithms of the same family are responsible for realizing the family interface so that switching dynamically among them at runtime can be done in a uniform and transparent manner. This is even true for source and sink interfaces, as the data management layer can then perform raw data storage transparently without changing the implementation of the pipelines, i.e., the Storm topologies.

Basically, dynamic changes within a Storm topology such as switching algorithms, changing functional parameters or switching from software to hardware can be realized in two ways, namely:

- Command / parameter streams, i.e., a pseudo data source in terms of an internal Spout emits a stream of infrequent command items such as parameter settings. Bolts react on this stream. However, using streams for commands or parameters requires the explicit layout of these (named) streams in a Storm topology.
- Signals sent through the Curator framework[8] to individual Spouts or Bolts. Signals represent a parallel information flow, which is orthogonal to the actual data processing. Signals can carry an arbitrary payload detailing the information to be sent.

In QualiMaster, we rely on Signals for realizing changes in the real-time data processing in Storm topologies. Based on runtime decisions made by the Adaptation Layer, the Coordination Layer sends appropriate signals to the involved Spouts and Bolts, which, in turn, implement the signal handling and either change or pass the information to the respective computational component.

The interfaces of families, data sources and data sinks follow the same construction principles. These interfaces contain:

- **Nested item interfaces** representing input and output item types. Input type interfaces declare getter methods to access the data, output type interfaces setter methods to define the results of the processing. Depending on the number of item types defined in the Configuration, algorithmic families define input and output interfaces, sources only output interfaces and sinks only input interfaces. Please note that different families cannot use the same input / output interfaces as this would prescribe a certain composition and impose unintended limitations to the Pipeline Designer.
- **Processing methods** depending on declared input and output types. Family interfaces declare calculate methods, data sources methods to read data from physical data sources and data sinks methods to write data to the physical endpoint such as a web server. The

[7] http://www.cascading.org/
[8] http://curator.apache.org/

processing methods are called from the respective elements of a Storm topology, i.e., Spouts and Bolts.
- Methods for **changing functional parameters** defined in the Configuration. These methods are called upon the reception of a Curator Signal indicating the need for changing the respective parameter.
-

As mentioned above, these interfaces of the computational components are defined in the configuration of the algorithmic families, data source and data sinks and can be derived automatically in terms of software artifacts, e.g., Java interfaces in order to ease the tasks of the Infrastructure Administrator.

In a pipeline implementation (a Storm topology), each computational component implementation is considered external, reusable and independent of Apache Storm. In particular, algorithms may be implemented in different forms, ranging from software-based algorithms to hardware-based algorithms. As also described in the D4.1 deliverable, we foresee currently four different technical types of algorithms, namely:

- Plain Java algorithms, i.e., Java classes implementing the family interface and performing the actual computation by producing output items for the given input items of the data stream, possibly relying on certain technical dependencies. An algorithm may receive functional parameters via the family interface.
- Distributed (Storm) algorithms in terms of an implemented Storm topology, actually a sub-Topology of the Storm implementation of a QualiMaster pipeline. This type of algorithm receives data streams complying with the input item types of the family interface, performs the calculation in a distributed manner and processes items on a data stream complying with the output types of the family interface. In contrast to a usual Storm algorithm implementation, the topology-building class of this algorithm also implements the family interface (calculating methods are left empty) in order to handle switching among algorithms in a uniform way. In particular, distributed algorithms can receive parameters via the family interface and may then either pass them on to the respective Bolts via the Storm signal mechanism, send them through a command data stream through the sub-topology or utilize local relevant mechanisms.
- Single hardware-based algorithms running on one or multiple Data Flow Engines (DFEs). Such algorithms receive an input stream over a network connection following the approach described in the D3.1 deliverable, perform that processing of the data stream in a massive-parallel hardware-based manner and produce output streams, again through a network connection. On the software side, a hardware-based algorithm is represented as an (implicit) member of the processing family realizing the family interface, but implementing the respective outgoing and incoming network streams. Akin to the types described above, not only data but also functional parameters can be sent to the hardware-based algorithm in a unified way via the family interface. Actually, the hardware-based algorithm may receive its functional parameters via the data stream or via a separate network connection for parameters.
- Multiple hardware-based algorithm, being part of a library of hardware-based algorithms loaded together into a set of DFEs. Due to performance reasons, it may be beneficial to execute algorithms that work closely together in a sub-pipeline directly on the reconfigurable hardware. Thus, such multiple hardware-based algorithms may represent a sub-topology, in which individual algorithms may be switched on and off, i.e., replaced by

software-based algorithms. Here, it is important to note that it typically makes only sense to execute connected sub-topologies, i.e., to enable or disable hardware-based algorithms at the beginning or the end of such a sub-pipeline, as arbitrarily mixing software- and hardware-based computing increases the communication overhead and, in summary, may destroy the benefits of hardware-based processing.

For example, an algorithmic family may include four different implementations of the same algorithm, such as two different Java implementations, one Apache Storm sub-Topology implementation, and one hardware-based MAX implementation. In order to distinguish the different types of implementations, an implementation is bundled by the Algorithm Provider into an Extension Bundle, which contains a QualiMaster-specific manifest as already mentioned in Section 2.1. This manifest contains information about the supported input and output item types, in particular for hardware-based implementations where an automated analysis may be difficult, the pipeline coverage in case of multiple hardware-based algorithms and, optionally, quality profiles for relevant quality parameters. During the configuration phase, the Extension Bundles are uploaded through the QualiMaster Infrastructure Configuration Tool, which analyzes the Manifest, takes relevant information over into the configuration and stores the Extension Bundle in the Processing Element Repository. As soon as a particular algorithm implementation is assigned to the configuration of a processing family, the QualiMaster Infrastructure Configuration Tool validates the validity of the interfaces and, in case of failures, informs the Infrastructure User and prevents that structurally invalid algorithms are composed together into a pipeline.

From a technical perspective, the Execution Layer also provides different forms of runtime information to the Coordination Layer and the Monitoring Layer:

- Execution failures, in case that pipeline parts fail unexpectedly at runtime. This information is passed to the Monitoring Layer, which informs the Adaptation Layer to take corrective actions.
- Adaptation enactment failures, in case that requested adaptation enactments (see D4.1 for a set of enactment patterns) cannot be performed for some reason. In general, the Coordination Layer, or in case of monitoring changes the Monitoring Layer inform the Adaptation Layer about the respective failure. The Adaptation Layer can then perform corrective actions such as a rollback to the last successful runtime configuration.
- Runtime execution information that is used for monitoring, in particular statistics collected by default by the Execution Systems. In case of Apache Storm, information such as the processing latency is available through the Thrift framework[9] and collected directly by the Monitoring Layer. Further, algorithms provide algorithm-specific information on the quality of the processing, in particular with respect to the Functional Suitability quality dimension of the QualiMaster quality taxonomy introduced in the D4.1 deliverable. This information is collected in the respective Bolts and passed on the Monitoring Layer. In addition, the Execution Layer indirectly also provides information about its Performance Efficiency parameters, either through generated monitoring probes in the Spouts / Bolts or due to the raw resource consumption monitoring probes of SPASS-meter [ES14] that are actually executed within the code running in the Execution Layer. Also this information is passed to the Monitoring Layer.

[9] https://thrift.apache.org/

2.3.3 Coordination Layer

The Coordination Layer is responsible for the entire pipeline lifecycle. This ranges from starting a pipeline to the enactment of adaptation decisions on the data analysis pipelines at runtime, such as selecting a specific data processing algorithm from an algorithmic family. Enacting one adjustment may imply further enactments, e.g., when switching the execution from software to hardware, two Execution Systems are involved. Enactment in general as well as efficient enactment across multiple Execution Systems in particular is handled by the Coordination Layer.

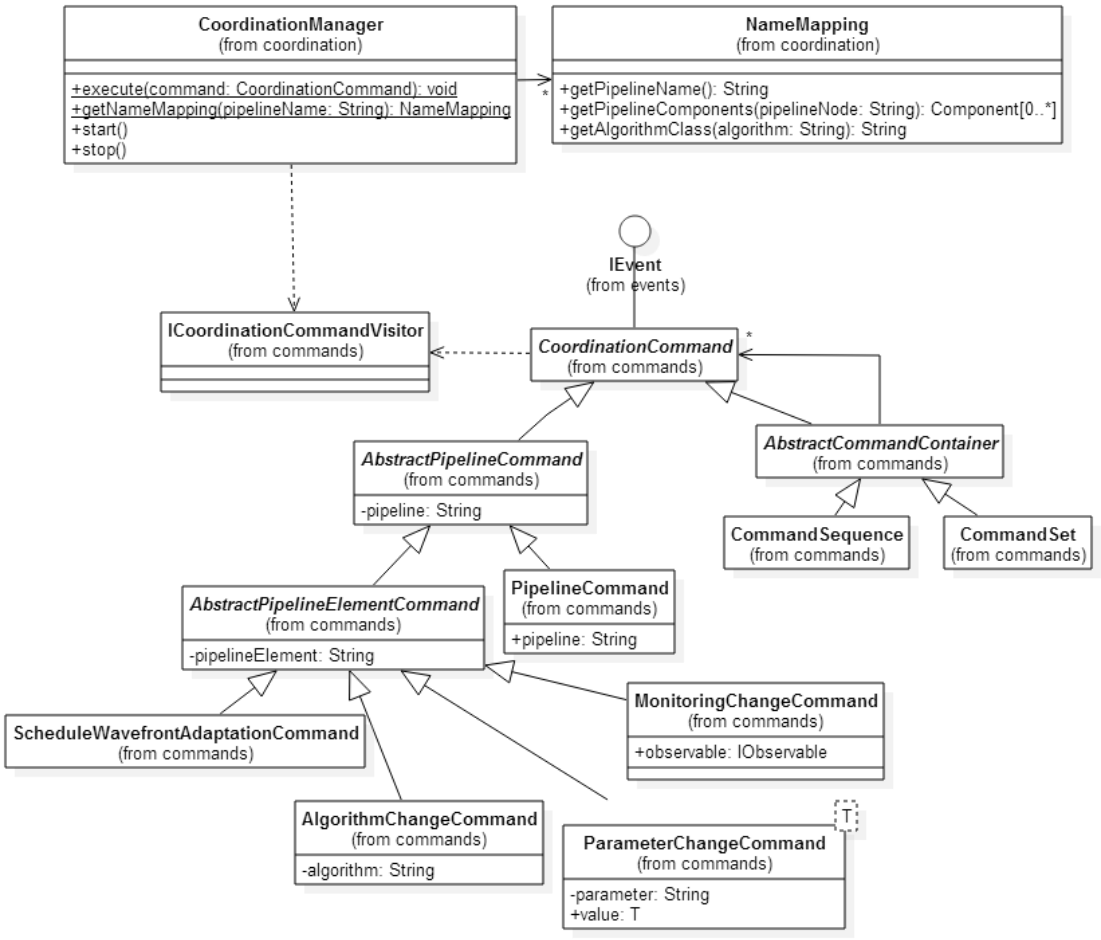

Figure 8: Class hierarchy for the Coordination Layer.

Figure 8 provides an overview on the class hierarchy of the Coordination Layer. Basically, the Coordination Layer consists of its frontend interface class `CoordinationManager`, which realizes the event handling in collaboration with the event bus and provides methods to be called during startup and shutdown of the QualiMaster runtime platform. Within the architecture of the runtime platform, the Coordination Layer acts as a mediator between the adaptation and the actual execution of the data processing. In fact, the Coordination Layer is located at the borderline between the logical design of a data processing pipeline done in the Coordination and the technical realization of the processing on the Execution Systems, e.g., using reconfigurable hardware or Apache Storm. Thus, for enactment of runtime changes, the Coordination Layer must translate the logical concepts of pipeline elements to the actual implementation and its

computational components. This is realized in terms of the `NameMapping` provided by the `CoordinationManager` for each running pipeline. The `NameMapping` is constructed from a descriptive XML artifact created along with the pipeline implementation during platform instantiation (cf. Section 4). In the opposite direction, implementation concepts must be translated back to logical concepts on the pipeline level in order to aggregate runtime information while monitoring to concepts defined by the Infrastructure Users. Also this backwards mapping is represented by the `NameMapping`, which, in turn, needs to be accessible also outside the Coordination Layer.

The major part of the interface of the Coordination Layer is realized in terms of executable commands. The command hierarchy is visualized in Figure 8. It contains:

- Commands issued by the Infrastructure User through the QualiMaster Infrastructure Configuration Tool in terms of the `PipelineCommand`:
  - Starting individual pipelines (through Apache Storm). Implicitly, the Coordination Layer connects the in- and output streams to the respective pipeline implementation and reprograms the reconfigurable hardware if required.
  - Stopping a given pipeline. Implicitly, the Coordination Layer disconnects the in- and output streams to the respective pipeline implementation.

  Upon execution of a `PipelineCommand`, the Coordination Layer issues lifecycle events in order to inform the other QualiMaster infrastructure layers about the changing state of individual pipelines. Examples are that the Coordination layer is about starting a pipeline, that a pipeline was started, that a pipeline is about to be stopped or that a pipeline was stopped.
- Commands issued by the Adaptation Layer for enacting adaptation decisions in a coordinated manner, i.e., executing the enactment patterns discussed in D4.1, such as setting a parameter (`ParameterChangeCommand`), switching an algorithm in a family (`AlgorithmChangeCommand`), potentially across software- and hardware-based execution or adapting the monitoring (`MonitoringChangeCommand`).
- Collections of commands to be executed in one step. Actually, we provide two types of collections in terms of the composite pattern [GHJ+00]: `CommandSequence` represents commands that must be executed in the specified order and `CommandSet` are commands that can be reordered during execution in order to optimize the coordination among the involved Execution Systems.

The execution of the individual commands is realized in terms of a visitor [GHJ+00] implementing the `ICoordinationCommandVisitor` interface based on the reception of a command in the `CoordinationManager`. Thereby, the implementation identifies the involved Execution Systems as well as the individual components within them, e.g., Storm Spouts or Bolts, translates the command based on the NameMapping to Execution System specific messages or commands (Curator Messages in case of Storm), and finally checks whether the command has been carried out successfully.

2.3.4 Monitoring Layer

The Monitoring Layer is responsible for the (distributed) surveillance of the Execution Layer in order to obtain readings of quality parameters for the actual execution of the data processing. As mentioned in Section 2.3.2, this includes information provided by the Execution Systems, the

running algorithms, the generated Spouts / Bolts as well as resource consumption information obtained from SPASS-meter [ES14]. The Monitoring Layer unifies and, if required, aggregates monitored information from different sources. In contrast to the Coordination Layer, the Monitoring Layer operates on the logical concepts of QualiMaster pipelines so that the aggregated information can be used for describing the adaptive behavior (see D4.1) or visualizing the actual execution using the right concepts. Therefore, the Monitoring Layer relies on the `NameMapping` provided by the Coordination Layer.

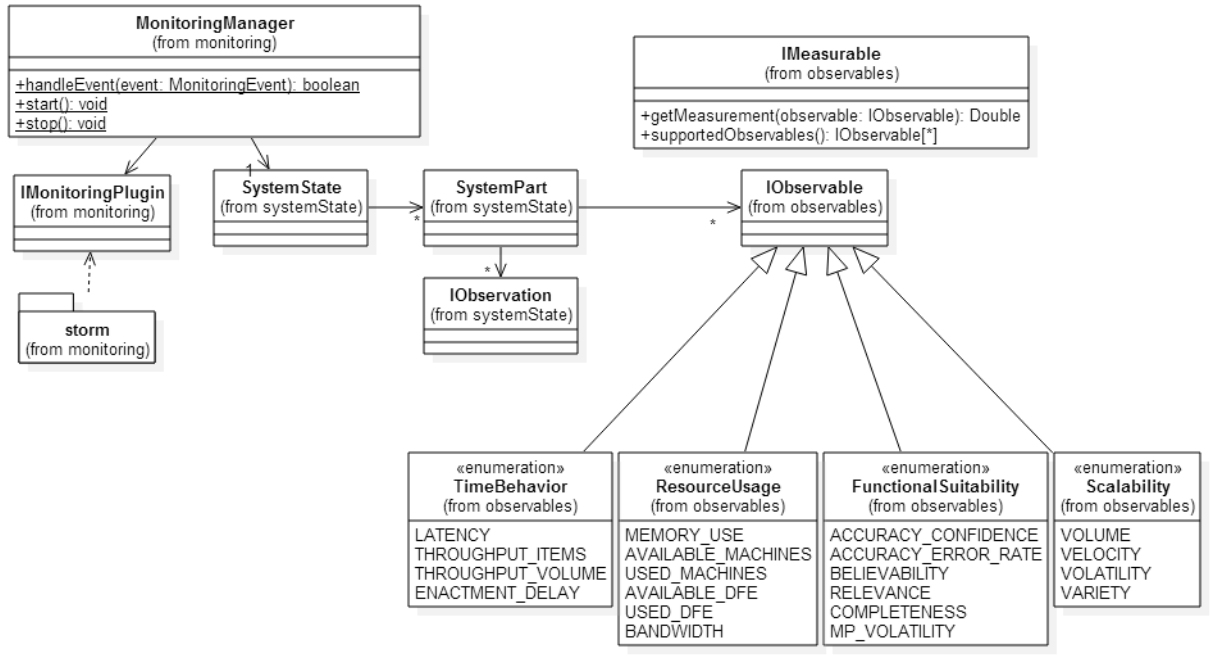

Figure 9: Overview of the static structure of the Monitoring Layer.

Figure 9 illustrates the class structure of the Monitoring Layer. The basic concepts of the Monitoring Layer are the following:

- The information that can be monitored. Akin to the Configuration Meta Model discussed in D4.1, this information is called `IObservable`. Actually, an `IObservable` is a descriptor of the information that is being monitored so that the information can be referenced, e.g., to aggregated actual values, retrieve the aggregated value or to change its monitoring. We define a set of enumerations, namely `TimeBehavior`, `ResourceUsage`, `FunctionalSuitability` and `Scalability`, representing the quality dimensions and parameters defined in the QualiMaster quality taxonomy in D5.1. Please note that through `IObservable` further information to be monitored can be defined.
- System entities that provide monitoring information through `IMeasurable`, in particular data sources, data sinks and algorithms. A Spout or Bolt can use the methods defined by `IMeasurable` to determine the actual values of the observables and to report them through a specific event to the Monitoring Layer for aggregation.
- `SystemState` containing the aggregated monitoring information for the runtime platform on different levels, namely pipelines, their processing elements and algorithms in terms of individual `SystemPart`s. Actually, a `SystemPart` links observations to observables. An `IObservation` represents the actual measurement of an `IObservable`. Specific

implementations of `IObservation` perform the aggregation in a polymorphic way. Note that the concept of a system state is extensible, i.e., additional `IObservable`s are concretized through their `IObservations` (as well as a monitoring probe to be deployed to the Execution Layer) and can be added to the `SystemPart`, thus, realizing an extensible Monitoring Layer (not detailed in Figure 9) supporting UC-AM2 and UC-PA1.

- Monitoring details of the actual execution happens in terms of monitoring events send through the event bus to the Monitoring Layer. Furthermore, Execution Systems may provide information on their processing, such as the number of worker nodes allocated by Apache Storm. This form of infrastructure level monitoring is realized in the Monitoring Layer in terms of plugins (`IMonitoringPlugin`). A monitoring plugin performs regular surveillance activities and aggregates the identified information in the `SystemState`. The `storm` package of the Monitoring Layer is an example for such a plugin for Apache Storm based on the Thrift framework.
- The `MonitoringManager` representing the frontend interface of the Monitoring Layer. The `MonitoringManager` implements the respective event handlers of the event bus in order to react on monitoring events from the Execution Systems, on change requests issued by the Coordination Layer and on starting / stopping a pipeline. In addition, the `MonitoringManager` provides methods to start and stop the entire layer used while powering up and shutting down the QualiMaster platform.

The Monitoring Layer is also responsible for controlling the actual monitoring, in particular what to monitor on which part and when, based on external requests (from the Adaptation Layer). Basically, Monitoring implies additional operations at runtime and, thus, some form of overhead. To a certain degree, the overhead can be controlled, e.g., by switching off raw monitoring probes and avoiding the further processing such as aggregation or analysis. Furthermore, in case of instrumented monitoring probes, a re-instrumentation at runtime may even remove monitoring probes at the cost of changing classes at runtime. While this may not be beneficial for (relatively) infrequently called monitoring probes, such as for network transfer, it can be crucial for expensive monitoring probes such as memory allocation or deallocation. In performance experiments on BenchMark suites we showed in [ES14] that about 7% of the execution time overhead incurred by SPASS-meter (around 10% using the naive mode in that experiment) can be avoided by dynamically instrumenting only relevant classes. Therefore, the Coordination Layer (based on a request from the Adaptation Layer) can request the change of monitoring for an `IObservable` on the different aggregation levels of the platform system state.

With respect to adaptation, the Monitoring Layer performs the first step in the so called MAPE-K cycle (monitor-analyze-plan-execute over a common knowledge base) [BSG+09, IBM06, KC03], which is frequently used as a blueprint for the architecture of adaptive systems. The next step, analyze, is responsible for determining adaptation triggers, e.g., due to the violation of Service Level Agreements (SLA), deviations from the expected quality, e.g., due to quality profiles stored in the Data Management Layer, or, in the case of QualiMaster, of quality constraints. Typically, this analysis is performed in a distinct component in order to separate concerns. In contrast, to reduce the communication among the layers of the QualiMaster infrastructure (and as the Monitoring Layer is mostly concerned with aggregation and control of monitoring), we decided to also locate also the analysis functionality of the MAPE-K cycle in this layer. In order to be able to perform analysis of the (built-in and user-defined) quality constraints, the Monitoring Layer must be aware of the Configuration (Meta) Model of the running pipelines. Therefore, during the startup process of

a pipeline, the Coordination Layer informs the other layers about changes of the pipeline lifecycle through a respective event. This enables the Monitoring Layer to obtain the Configuration (Meta) Model, to access the relevant constraints and to monitor their potential violation based on actual values of the `SystemState` for the quality parameters (`IObservable`). Therefore, the Monitoring Layer is based on the runtime library of EASy-Producer, which allows to read the models, access the individual model elements, in particular constraints, and to perform reasoning on the model, including the evaluation of individual constraints to improve for performance. Although the required events are implemented, linking `IObservable`s to the Configuration and the actual monitoring of constraint violations will be realized in the future.

According to the planning of the priority pipeline that was presented in the D5.1 deliverable, currently the Monitoring Layer sends updates of its `SystemState` to the Adaptation Layer, which is responsible for the external communication with the QualiMaster Infrastructure Configuration tool, here for visualizing monitoring information.

2.3.5 Adaptation Layer

The Adaptation Layer performs the decision making based on monitoring and analysis of the actual execution of pipelines (as provided by the Monitoring Layer). The behavior of the Adaptation Layer is defined in an Adaptation Behavior Specification, given in terms of the Runtime Variability Instantiation Language (rt-VIL) described in D4.1. The adaptation behavior is specified by the Adaptation Manager (e.g., based on existing specifications shipped with the generic QualiMaster infrastructure) in order to support domain-specific adaptation of the pipelines. As rt-VIL will be realized in a domain-independent form in EASy-Producer, also the Adaptation Layer will be based on the runtime library of EASy-Producer and, in particular, realize the QualiMaster-specific binding to rt-VIL, the event handling and the external communication with Stakeholder Applications in order to enable user triggers.

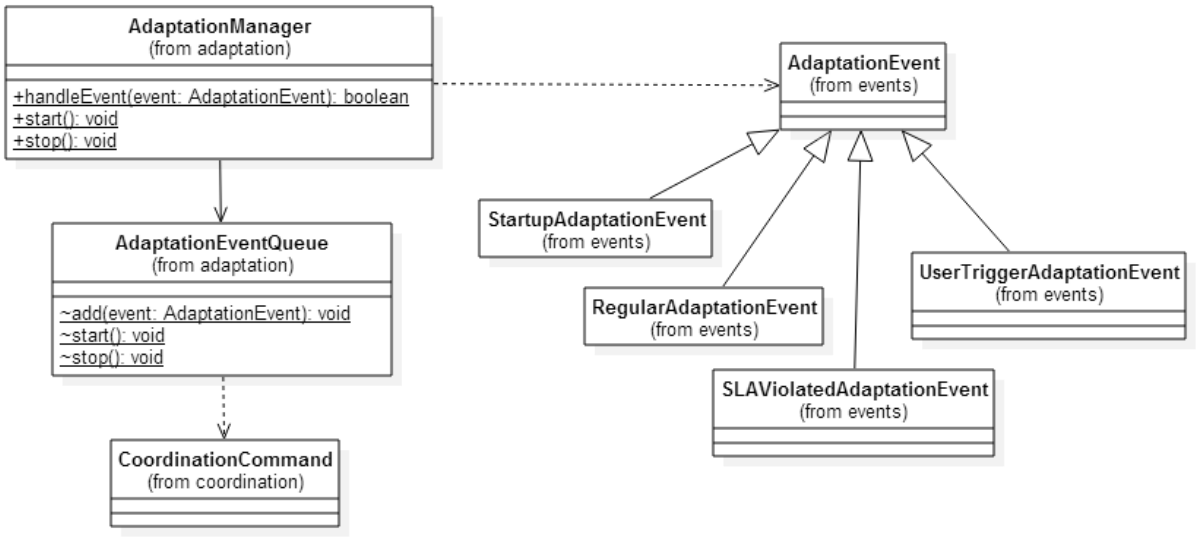

Figure 10: Class diagram for the Adaptation Layer.

Figure 10 depicts the static architecture of the Adaptation Layer. It consists of the `AdaptationManager`, which realizes the event handling as well as the integration into the QualiMaster platform lifecycle in terms of starting and stopping the platform.

At runtime, the Adaptation Layer only becomes active upon a trigger (technically an event), including the following:

- SLA constraint violation (`SLAViolatedAdaptationEvent`) issued by the Monitoring Layer.
- User triggers send by the Stakeholder Applications (`UserTriggerAdaptationEvent`).
- Regular adaptation schedule triggers as defined in the Configuration (`RegularAdaptationEvent`), actually an internal self-trigger of the Adaptation Layer.
- Pipeline startup (`StartupAdaptationEvent`), another internal trigger issued upon the reception of a pipeline lifecycle event by the `AdaptationManager`.

Note that further events due to triggers described in D4.1 will be defined in the future.

The reception of an `AdaptationEvent` causes the execution of the Adaptation Behavior Specification in terms of rt-VIL. Therefore, adaptation events are passed to the `AdaptationEventQueue`, which is responsible for the time-synchronous execution of the events by rt-VIL. Actually, the `AdaptationEventQueue` already prepares the later integration of rt-VIL into the Adaptation Layer. Please note that we will map QualiMaster architecture components such as adaptation events, observation descriptors or coordination commands into the type system of rt-VIL. Thus, rt-VIL will allow the Adaptation Manager to specify adaptation strategies, sub-strategies and tactics, which are selected dynamically upon failed objectives, the causing event type and weighting functions. The Execution of an adaptation strategies will change the runtime settings of the configuration, cause a runtime validation of the runtime configuration and, if successful, transform these settings into commands for the Coordination Layer also mapped into the rt-VIL type system. Please refer to D4.1 for more details on the concepts and semantics of rt-VIL.

According to the schedule implied by the priority pipeline described in D5.1, the Adaptation Layer currently acts as an endpoint for the communication with the Infrastructure Configuration Tool, which visualizes monitoring information for demonstration purposes and allows manual triggers of runtime algorithm changes. While monitoring information are passed through from the Monitoring Layer, manual algorithm change triggers are translated into commands of the Coordination Layer and send there for enactment.

2.4 Repositories

We now describe the two repositories used in the QualiMaster Infrastructure, i.e., the pipeline repository (Section 2.4.1) and the processing elements repository (Section 2.4.2).

2.4.1 Pipeline Repository

The pipeline repository stores the configuration information about the pipelines, more precisely about the actual Configuration Meta Model and the Configuration (see D4.1 for details) of the entire QualiMaster Infrastructure. As the Configuration Meta Model is specified in terms of the INDENICA Variability Modeling Language [I12, I14], the pipeline repository must in particular support storing and versioning of text files as well as access protection in order to separate the different QualiMaster Infrastructure User groups.

We selected Subversion[10] for this purpose and organized the repository for the Configuration into the following parts:

- Configuration Meta Model, defining the configuration space.
- Configuration, consisting of the following:
  - Infrastructure part corresponding to the configuration options of the Platform Administrator, i.e., software-based and hardware-based computing resources, algorithms, algorithmic families and global infrastructure settings, e.g., the pipelines selected for execution.
  - Adaptation part storing the settings of the Adaptation Manager regarding quality parameters and adaptation rules.
  - Pipeline part containing the configuration of individual pipelines in terms of their data flow topologies and references to the parts explained above.
- Pipeline derivation process description as we detail in Section 4.

Due to the capabilities of Subversion, read and write access to the individual parts can be defined and granted according to the roles of the QualiMaster Infrastructure Users.

For convenience, a higher level access interface to the Pipeline Repository has been defined. This interface is used for example by the QualiMaster Infrastructure Configuration Tool to obtain the Configuration Meta Model as well as the actual Configuration. Please note that instantiated pipelines are considered as software artifacts and stored along with the processing elements in the Processing Elements Repository.

2.4.2 Processing Elements Repository

The processing elements repository stores the software artifacts that realize the data processing in QualiMaster, in particular the computational components, the instantiated pipeline (in binary as well as in source format for analysis purposes) as well as further extension artifacts of the QualiMater platform and the Execution Systems. For easing the integration along the lines of the priority pipeline defined in D5.1, we decided to keep the current version of the processing elements repository simple and store the artifacts in a dedicated folder of the file system (synchronized for development purposes against the QualiMaster development Subversion repository).

As the deployment of Storm topologies heavily relies on the Maven[11] build process, we plan in the following months to utilize a local Maven repository in order to realize the QualiMaster processing elements repository. The QualiMaster Infrastructure Configuration Tool will then be extended with a second repository connector to write the provided extension packages after analysis into the Processing Elements Repository. Relying on Maven will also ease the platform derivation, in particular as the instantiation of the pipelines, can then be done through a derived Maven specification and, on success, written into the Processing Elements Repository using Maven functionality. Further, to ease the administration of the Processing Elements Repository and the permission-based access to it, we consider Sonatype Nexus[12], a software artifact repository management tool also used in industry.

[10] https://subversion.apache.org/
[11] http://maven.apache.org/
[12] http://www.sonatype.com/nexus

Further, we consider extending the Configuration Meta Model and the Infrastructure Configuration Tool with descriptive information for algorithms. We aim at easing the Infrastructure User's tasks by providing a search capability on the description of algorithms.

2.5 External Interface

The external interface of the QualiMaster runtime platform enables the QualiMaster Stakeholder Applications to send user triggers for adjusting the processing on demand (see D1.1). For providing the calculation results to the Stakeholders, the installation of the QualiMaster platform will utilize an external available server, called `Sink Server` in Figure S1. Basically, on the `Sink Server` the `ProtocolHandler` is running, a class which implements the application communication protocol currently specified by WP6. Thus, this section describes the conceptual design of the external interface. We discuss the communication process for a parameter-based user trigger, strictly separating external `Sink Server` and (internal) QualiMaster platform. If required, further kinds of user triggers can be realized in a similar way.

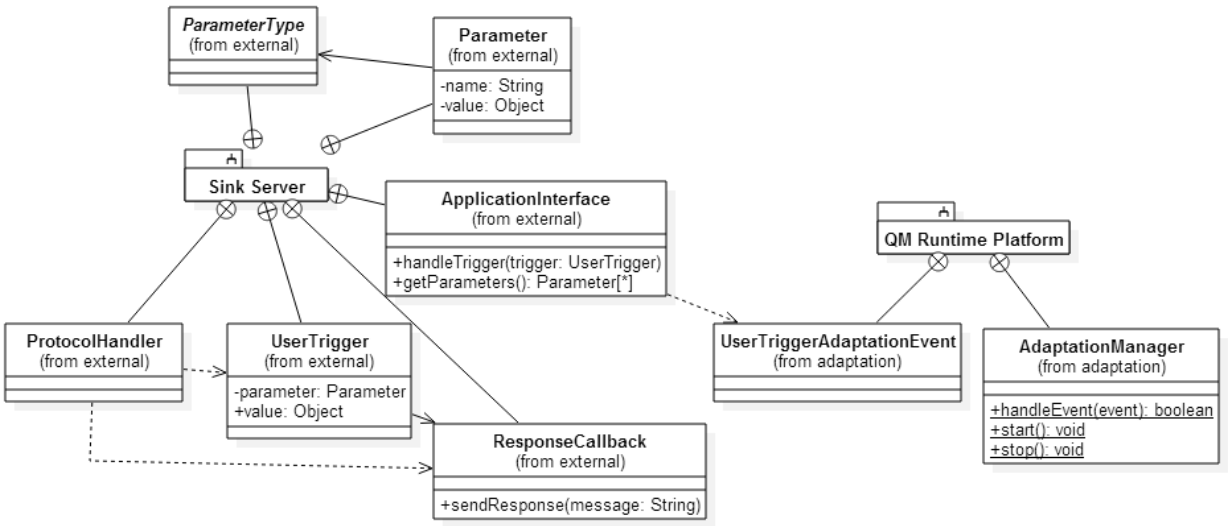

Figure 11: Class diagram of the external interface connecting the QualiMaster platform with the Stakeholder applications.

As part of the application communication protocol, the Stakeholder application is able to ask for runtime parameters a user can influence. This information is provided by the `ApplicationInterface` in terms of stakeholder-visible `Parameter`s obtained from the Configuration Meta Model (not displayed in Figure 11). A `Parameter` provides access to its name, its actual value and the value range a user can choose a value from. As (unlimited) primitive type ranges such as Integer as well as enumerations such as the actual market players are relevant, the value range will be a class hierarchy indicated in Figure 11 by `ParameterType`. For requesting an adaptive change by an user trigger, the `ProtocolHandler` creates an `UserTrigger` based on the actual input of the stakeholder as well as a `ResponseCallback` turning an asynchronous (fire & forget) trigger into a trigger, which leads to a result that can be displayed to the user. Finally, the `ProtocolHandler` calls the `handleTrigger` method of the `ApplicationInterface`. The `ApplicationInterface` acts as communication client on the perimeter of the QualiMaster infrastructure, i.e., it translates the user trigger into a

`UserTriggerAdaptationEvent` (linking the external `ResponseCallback` to an internal callback, not shown in Figure 11) and passes the event to the event bus. The Adaptation Layer receives the event, processes the rt-VIL specification and returns the result via the attached callback(s).

2.6 QualiMaster Applications

We now provide an overview on the Application Designer tool and the Stakeholder Applications, since these are the two major parts on the application side of QualiMaster. More detailed information regarding these application will be included in the D6.1 deliverable (due in month 19 of the project).

2.6.1 Application Designer

The Application Designer is used to design / compile stakeholder applications. It not only allows to define the application-side data processing of the data provided by the Qualimaster infrastructure, but also easily create / modify the visualisation of results. This makes the design of stakeholder application very flexible and allows different visualisation approaches within one stakeholder company. Also, applications can be modified in terms of client data processing and visualisation on demand.

2.6.2 Stakeholder Application(s)

Based on the processing and visualisation needs of each stakeholder, one or more applications are provided to the stakeholder. The application-side processing of the result data allows to further analyse, modify and summarize the results coming from the Qualimaster infrastructure. Visualisation includes several types of diagrams, graphics, tables and so on as well as the ability of viewing real time and historical data and applying several types of filters.

3 Communication between layers

An important aspect of the QualiMaster infrastructure is the communication between the layers. This section discusses this infrastructure aspect. We start by providing the possible communication cases (Section 3.1), followed by the presentation and discussion of the communication occurring in a few processing examples (Section 3.2-0).

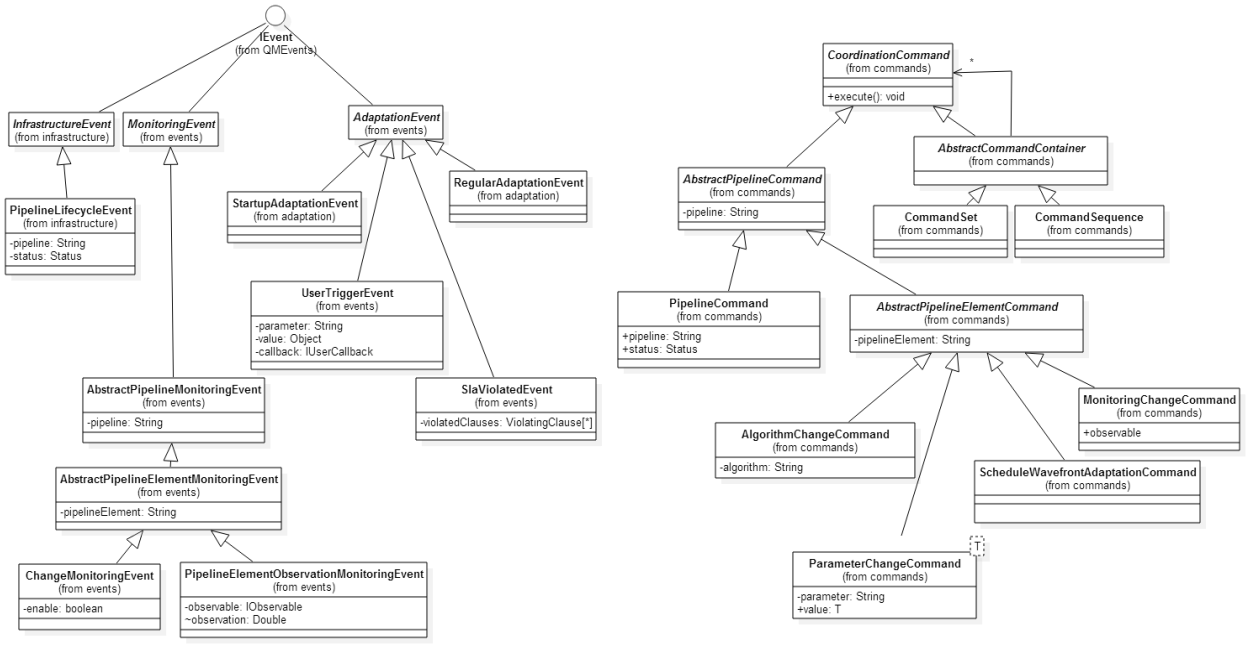

Figure 12: Class diagram for the events end commands used in QualiMaster.

3.1 Communication Cases

Consider again Figure 1, and in particular the layers composing the configuration environment and the runtime platform. According to the design principles introduced in Section 2, each layer can call the layers that are directly lower to it (in the figure) and send asynchronous events to the layers above. Figure 12 illustrates an overview on all events and commands defined by the QualiMaster platform.

The following list provides the layers that can communicate between them and provides an overview of the corresponding messages, i.e., the requested processes and the required parameters:

1. **Execution Layer with incorporated Execution Systems**: The Execution Layer is responsible for executing commands using the incorporated Execution Systems, which currently include Apache Storm, reconfigurable hardware, and Apache Hadoop. Basically, we utilize the mechanisms provided by the Execution Systems to send commands to one or multiple of these incorporated systems and to receive the results.
2. **Execution Layer with Data Management Layer**: The Execution Layer is also responsible for receiving the streaming data in order to provide it to the corresponding incorporated Execution Systems and for handling the final results. Both actions are performed through the Data Management Layer. Since QualiMaster deals with streaming data, the data

provided by Data Management Layer to the Execution layer is basically a stream. As discussed in D5.1, the Data Management Layer may transparently store raw input as well as result data according to a given storage strategy.

3. **Coordination Layer with Execution Layer**: One example for the communication between Coordination Layer and Execution Layer is to startup a pipeline. Therefore, the Coordination Layer processes and deploys the pipeline accordingly (e.g., a JAR file to Apache Storm or a MAX file to the data flow engines) and sends the appropriate commands via mechanisms mentioned in 1) to control the execution in the Execution Layer, e.g., starting the pipeline. Another example is to enact an adaptation decision, i.e., to cause changes to the deployed pipelines at runtime.
4. **Monitoring Layer with Execution Layer and incorporated Execution Systems**: The Monitoring Layer is constantly observing the processing performed by the Execution Layer as well as the performance of the systems it uses (e.g., reconfigurable hardware). As discussed in Section 2.3.4, the Monitoring Layer may actively query information, e.g., via Apache Thrift, or passively receive information, e.g., from the generated pipeline code or SPASS-meter. The Monitoring Layer collects statistics that are maintained locally in the layer.
5. **Monitoring Layer with Adaptation Layer**: As described in Section 2.3.4, the Monitoring Layer implements two phases of the MAPE-K cycle, namely monitoring and analysis. Upon unusual situations, e.g., SLA violations based on evaluating constraints of the Configuration Meta Model on actual data, the Monitoring Layer informs the Adaptation Layer about the need for an adaptation in terms of an adaptation event.
6. **Adaptation Layer with Coordination Layer**: Upon an adaptation event, the Adaptation Layer executes the adaptation behavior specification given in rt-VIL. During the execution, the Adaptation Layer may change the runtime configuration, validate the changes of the runtime configuration, create Coordination Layer commands and request the execution by the Coordination Layer.
7. **Coordination Layer with Monitoring Layer**: One adaptive decision may be to change the actual focus of the Monitoring. Then, the Adaptation Layer requests the enactment as described in 7), the Coordination Layer receives a monitoring change command (potentially as one of multiple commands), and informs the Monitoring Layer by sending a monitoring change event to the monitoring layer.

Below, we discuss the most prominent communication patterns in the QualiMaster platform, namely:

- Starting a pipeline
- Normal pipeline processing
- Adaptation through monitoring (SLA violation)
- User triggers from the stakeholder applications
- Stopping a pipeline

We discuss the communication patterns in terms of UML sequence diagrams. Please note that sequence diagrams typically do not show a complete interaction rather than an ideal, simplified representative example of a certain situation.

3.2 Starting a pipeline

Starting a pipeline is an essential Qualimaster command. It refers to the use case where the pipeline is created, fully configured, and the Platform Administrator wishes to start its execution on the Qualimaster infrastructure (UC-PA8).

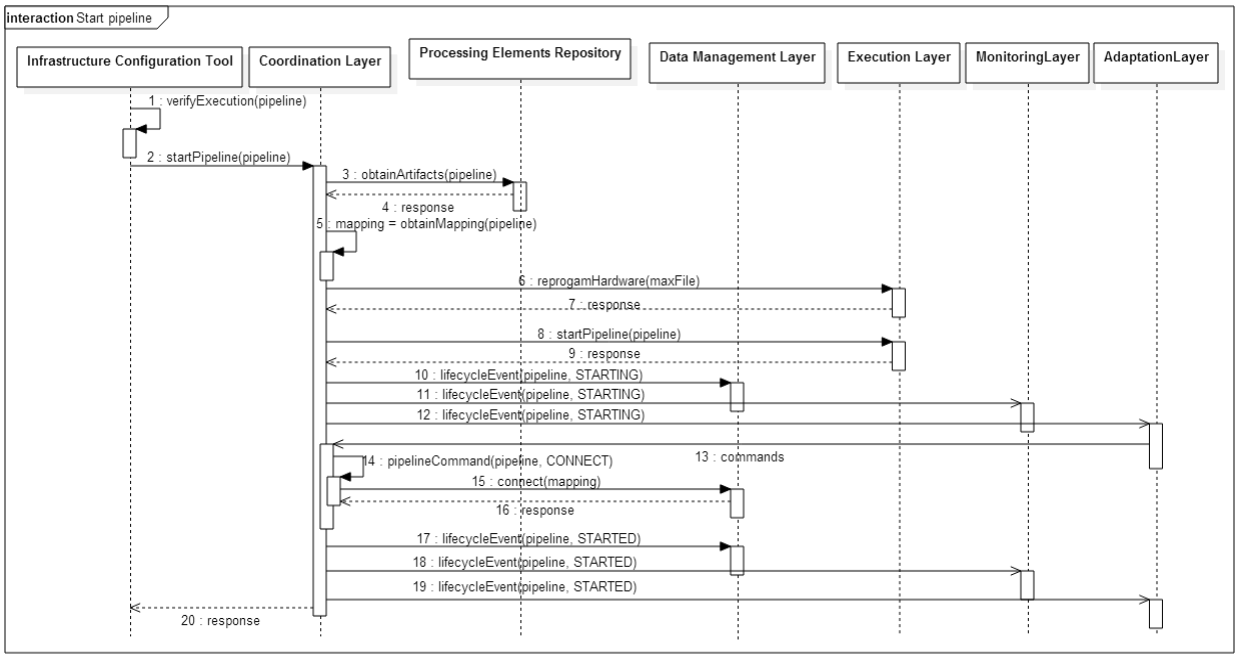

Figure 13: Sequence diagram for starting a pipeline.

Figure 13 depicts the sequence diagram for starting a pipeline. The `QualiMaster Infrastructure Configuration Tool` is responsible for checking whether the pipeline is correctly configured, i.e., it validates the constraints defined in the Configuration Meta Model and checks that it can be executed on the infrastructure based on a quality impact analysis (i.e., sequence number 1). Then the `Infrastructure Configuration Tool` verifies that the Platform Administrator indeed wants to execute the pipeline and, if he/she confirms, the Configuration Tool issues the start command to the coordination layer (i.e., sequence number 2). The `Coordination Layer` is responsible for interpreting the received command. Therefore, it obtains the actual implementation artifacts from the `Processing Elements Repository` (i.e., sequence number 3), obtains the mapping between logical and implementation names (i.e., sequence number 5), and issues the appropriate commands to the `Execution Layer`, namely reprogramming the reconfigurable hardware (i.e., sequence number 6) and starting the pipeline via the `Execution Layer` on Apache Storm (i.e., sequence number 8). Then the `Coordination Layer` notifies the QualiMaster platform that a pipeline is about to be started, i.e., the `Data Management Layer` (sequence number 10), the `Monitoring Layer` (sequence number 11) and the `Adaptation Layer` (sequence number 12) can prepare their activities, e.g., by loading the pipeline Configuration. This also causes a startup event in the `Adaptation Layer` to determine the initial algorithm selections and their parameters (sequence number 13) and causes a connect pipeline command indicating that the pipeline initialization is completed (processing of that event is indicated by sequence number 14). Due the asynchronous notion of events, we

enforce with that pipeline connect happens after the Adaptation Layer has determined and enacted the initial runtime configuration on the pipeline (through a command sequence). For turning the pipeline into operation, the `Coordination Layer` asks the `Data Management Layer` to connect all data elements (sources, sinks, intermediary result stores) for this pipeline (indicated by passing the mapping to the `Data Management Layer` in sequence number 15). Ultimately, the `Coordination Layer` informs the platform layers about the started pipeline (sequence numbers 17-19) as well as the `Infrastructure Configuration Tool` that the execution of the start command was successful (i.e., sequence number 19).

3.3 Normal Pipeline Processing

We now discuss the communication between the layers by considering a specific processing example and more specifically the communication occurring during normal pipeline processing with Apache Storm. Let us now consider that the pipeline is actually running. Note that all layers have been already initialized, since this happens during the configuration and startup lifecycle phases of the QualiMaster infrastructure. Thus, the initialization (as well as the configuration) of the tools and layers in not included in the provided sequence diagram. Figure 14 illustrates the interactions between the layers in the sequential order that those interactions occur.

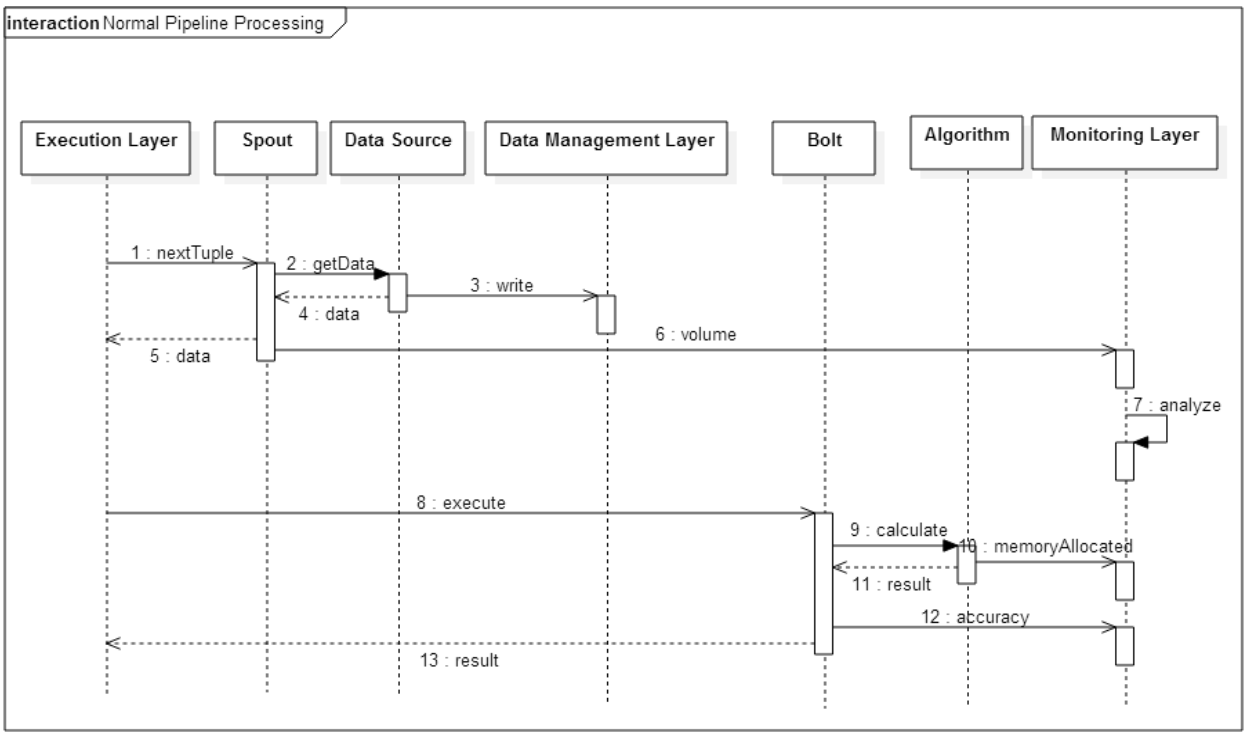

Figure 14: Sequence diagram for the normal pipeline processing.

The `Execution Layer`, more specifically, Apache Storm sends messages to the elements of the actual pipeline, for example, as shown in Figure 14 to a `Spout` for data ingestion (i.e., sequence number 1). The `Spout` asks its attached `Data Source` for the next data item(i.e., sequence number 2). The `Data Source` is querying the physical data source, for example a server at SPRING (not shown in Figure 14). Assuming that transparent data storage is configured for the `Data Source`, the obtained data item is passed through to the `Data Management`

Layer for storing the raw data in parallel (i.e., sequence number 3). As one of the implementation options, the Data Source may be a dynamic proxy, which obtains the item from the real Data Source implementation, returns the item and persists the data in parallel through the Data Management Layer (not shown in Figure 14). The obtained data item is then returned to the Spout (i.e., sequence number 4), which passes it to the Execution Layer (Storm) for processing it along the pipeline topology (i.e., sequence number 5). At the end of the Spout operation, the Monitoring Layer is informed about the actual volume (sequence number 6, based on querying the Data Source not shown in Figure 14). In the mean time, the Monitoring layer analyzes the constraints for validation, but does not detect a critical situation (i.e., sequence number 7).

Next, the Execution Layer (Storm) continues by passing the data item to the next Bolt (i.e., sequence number 8). Here, the control and the data item is passed to the actual Algorithm, which performs the calculation (i.e., sequence number 9). As part of the calculation, memory is allocated which is detected by SPASS-meter via an instrumented monitoring probe (not shown in Figure 14) and the Monitoring Layer is notified accordingly (i.e., sequence number 10). After ending the calculation by the algorithm (i.e., sequence number 11), the Bolt continues processing, informs the Monitoring Layer about the actual accuracy (generated monitoring probe, i.e., sequence number 12) and returns control and result to the Execution Layer (Storm, i.e., sequence number 13).

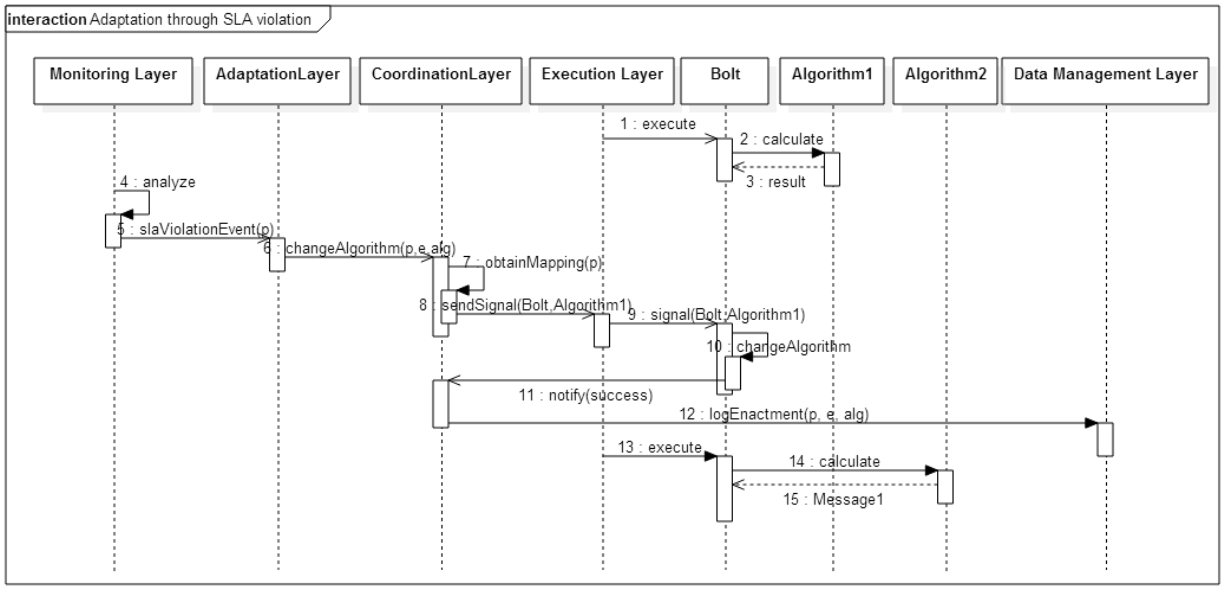

Figure 15: Sequence diagram for adapting a pipeline due to an SLA violation.

3.4 Adaptation Through Monitoring: SLA violation

Violation of quality constraints is a basic case where analysis in the Monitoring Layer leads to an adaptation of the data processing. This is illustrated in Figure 15.

Let us assume normal execution at the beginning of the sequence diagram, i.e., the `Execution Layer` (Storm) causes an execution of a `Bolt` (i.e., sequence number 1), which relies on the calculation in `Algorithm1` (i.e., sequence numbers 2 and 3). In the mean time, the `Monitoring Layer` analyzes the collected system state (i.e., sequence number 4) and detects an SLA violation through runtime reasoning. As a result, it issues an SLA violation event (i.e., sequence number 5) so that the `Adaptation Layer` can check whether an adaptation is required. By processing the rt-VIL specification of that specific event, the `Adaptation Layer` determines that algorithm attached to the processing element `e` (corresponding to `Bolt`) in pipeline `p` shall be switched to `alg` and sends a respective command to the Coordination Layer (i.e., sequence number 6). The coordination layer obtains the mapping from logical to implementation names and translates the given information to the Storm execution elements (i.e., `e` is translated to `Bolt`, `alg` to `Algorithm1` in sequence number 8). In order to cause the change of the algorithm, the `Coordination Layer` sends a Thrift signal to the `Execution Layer`, requesting that the algorithm in `Bolt` shall now be `Algorithm1` (i.e., sequence numbers 8 and 9). Upon reception of the signal, the signal receiver in `Bolt` now changes the processing algorithm from `Algorithm` to `Algorithm1`, i.e., considering the processing state it changes the actual instance of the family interface for `Bolt`. Now the `Bolt` notifies the `Coordination Layer`, which logs the enactment through the `Data Management Layer`. Upon the next tuple, the `Execution Layer` (Storm) will call `Bolt` again, which now utilizes the processing defined in `Algorithm1` (i.e., sequence numbers 13-15).

3.5 User Triggers from the Stakeholder Applications

As discussed in Sections 2.5-2.6, Stakeholder Applications shall be able to send user triggers for adjusting the processing. Figure 16 depicts a sequence diagram illustrating how user triggers are processed.

The Stakeholder user aims at changing the actual market players, let's say to "Apple". After entering the request, e.g., based on a graphical selection, the Stakeholder `Application` sends a user trigger request to the `Application Interface` of the QualiMaster runtime platform (i.e., sequence number 1, as discussed in Section 2.5). The Application Interface translates the request to an user trigger adaptation event also indicating the underlying running pipeline as well as the response callback (not shown in Figure 16) and sends it to the `AdaptationLayer` (i.e., sequence number 2). The `Adaptation Layer` executes the rt-VIL script of the running platform and decides that the parameter change shall be enacted on the pipeline source `source1` (due to the relation of the parameter to the data source). This happens in terms of a change parameter command send to the `Coordination Layer` (i.e., sequence number 4). The `Coordination Layer` obtains the logical-technical mapping for the given pipeline, translates the pipeline to its implementing topology as well as the `source1` to the actual `Spout` and sends the respective Thrift signal via the `Execution Layer` to the respective storm `Spout` (i.e., sequence numbers 5 and

6). The `Spout` passes the parameter to the actual `Data Source` implementation, which causes the change of the parameter. The `Spout` notifies the `Coordination Layer` of the success, which passes the information back to the `Adaptation Layer` and, finally, via the callback, to the `Application` (i.e., sequence numbers 8-11).

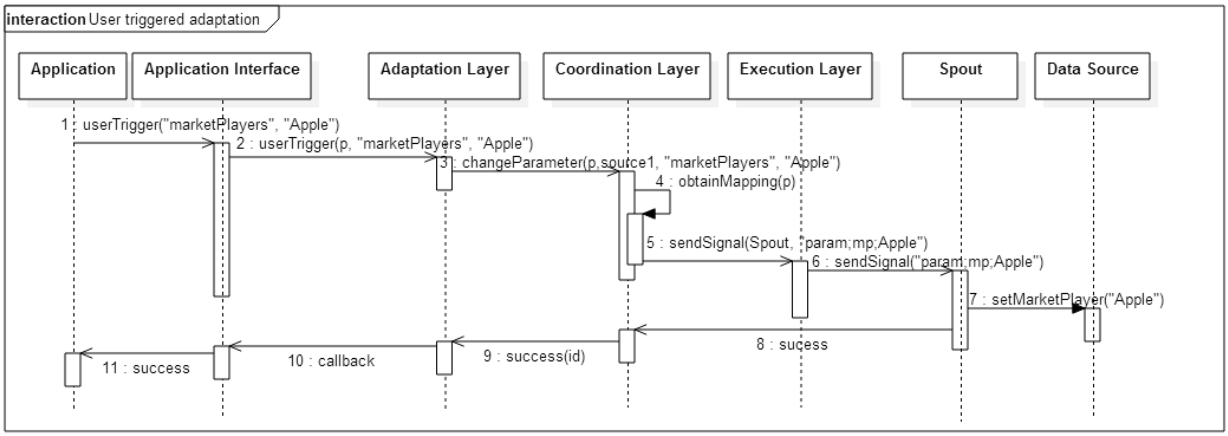

Figure 16: Sequence diagrams for adaptation due to an user trigger.

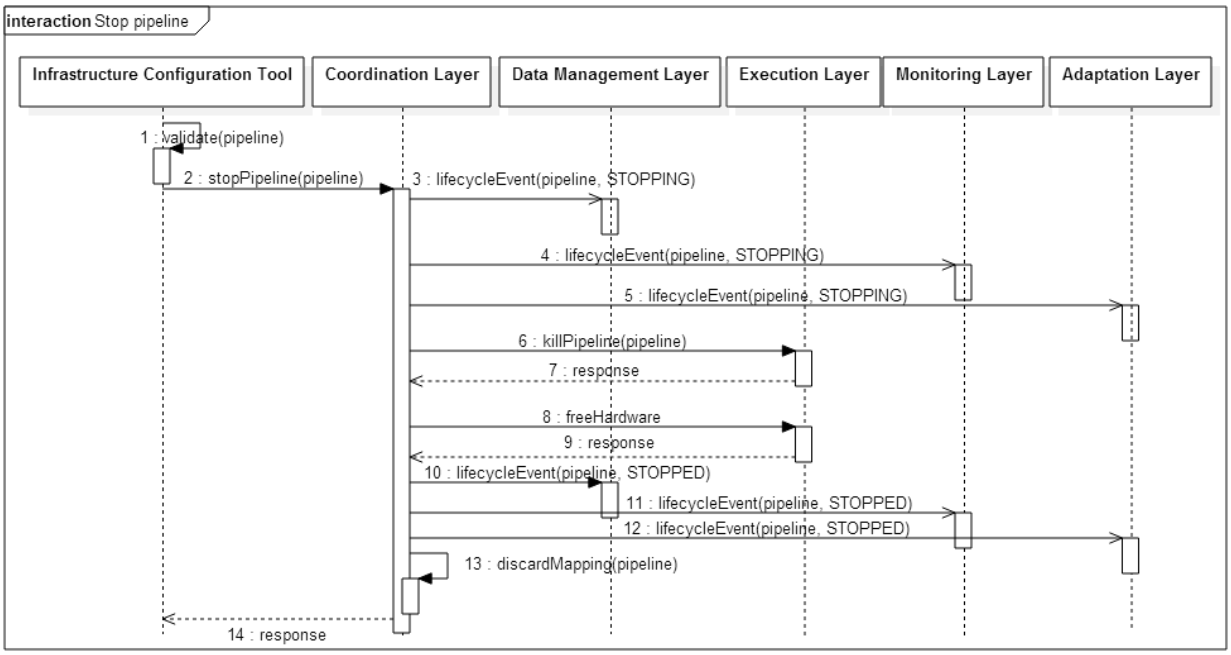

Figure 17: Sequence diagram for stopping a pipeline.

3.6 Stopping a pipeline

Stopping a pipeline is another essential Qualimaster command. It refers to the use case where the pipeline is already being executed on the Qualimaster infrastructure and the Platform Administrator wishes to stop its execution (UC-PA9).

The process for stopping a pipeline is similar to starting a pipeline, i.e., Figure 17, but, in contrast, the `Adaptation Layer` does not influence the pipeline to be stopped. The `Infrastructure Configuration Tool` is responsible for checking whether the pipeline can be safely stopped without affecting other pipelines (i.e., sequence number 1). Then the `Infrastructure Configuration Tool` verifies that the Platform Administrator indeed wants to stop the pipeline and, if he/she confirms, the `Infrastructure Configuration Tool` issues the stop command to the `Coordination Layer` (i.e., sequence number 2). First, the `Coordination Layer` issues lifecycle events that the pipeline is about to be stopped (sequence numbers 3-5), which causes the disconnection of the data sources, data sinks and intermediary storage exclusively used by the pipeline in the `Data Management Layer`. Further, the lifecycle event also informs `Monitoring Layer` and `Adaptation Layer` (sequence numbers 4-5) to stop their respective operations on the pipeline. Then the `Coordination Layer` stops the implementing Storm topology (via the name mapping not shown in Figure 17) by sending the kill pipeline command to Apache Storm (i.e., sequence number 6) and, afterwards, issuing a command to free the reconfigurable hardware (i.e., sequence number 8). Finally, the `Coordination Layer` issues the stopped pipeline lifecycle event (sequence numbers 10-12), discards the pipeline mapping (i.e., sequence number 13), and notifies the Infrastructure Configuration Tool about the successful execution (i.e., sequence number 14).

4 Infrastructure Instantiation Process

As discussed in the D4.1 deliverable, the configuration of the infrastructure includes in particular the hardware pool for software-based and hardware-based execution, the data management (data sources and sinks), the algorithms, the algorithmic families and the pipelines to be executed. The goal of configuring the infrastructure is to ease the consistent specification of data analysis pipelines by domain users and, in particular, the adoption to other software settings and domains. In order to be executed, a configuration must be turned into software artifacts such as a Storm topology. This can be achieved by applying Software Product Line Techniques for product instantiation (sometimes also called derivation), i.e., by automatically and consistently transforming a given configuration into software artifacts, thus, avoiding tedious and error-prone manual customization activities. In this section, we discuss the current state of automating the instantiating the QualiMaster platform based on a given configuration. This supports Platform Administrators in concentrating on their expertise rather than realizing adaptive Storm topologies or even manually integrating software- and hardware based algorithms into such a topology. However, as application settings differ in practice, automated instantiation must be flexible, customizable by experts and, ultimately, provide support for QualiMaster pipelines, i.e., topological configurations.

A valid configuration is a prerequisite for (successful) platform instantiation. A valid QualiMaster configuration determines values for the (pre-runtime) configurable elements defined by the QualiMaster Configuration Meta Model (cf. D4.1) and requires that the given values comply with all constraints of the Configuration Meta Model. The task of the QualiMaster platform instantiation process is to turn a valid configuration model into (software) artifacts and to prepare the instantiated QualiMaster infrastructure for deployment to the configured cluster consisting of general-purpose and (if configured) reconfigurable hardware.

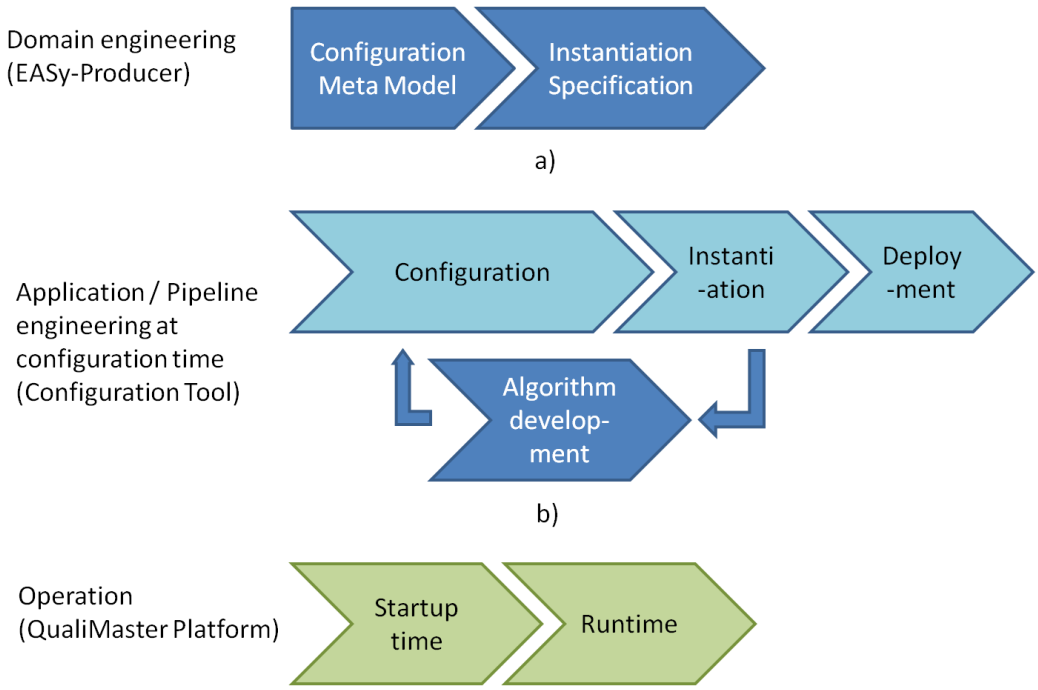

Figure 18: Overview on the QualiMaster lifecycle phases and the platform derivation process.

Figure 18 illustrates the overall process for deriving a QualiMaster platform. The Configuration Meta Model is given in terms of the INDENICA Variability Modeling Language (IVML) [I12, I14]. The instantiation process is given in terms of the Variability Instantiation Language (VIL) [I13, I14b], a domain-specific language for specifying and executing product line derivation processes. IVML and the corresponding VIL instantiation specification are developed during the course of the QualiMaster project as indicated in Figure 18 a). The QualiMaster Infrastructure Configuration Tool can execute this process through the Configuration Core. As part of an adoption of the QualiMaster approach, trained software professionals can use the EASy-Producer tool set [EEK+14] which realizes the Configuration Core to adjust the Configuration Meta Model and the instantiation process in order to adapt the QualiMaster, infrastructure for their specific needs, e.g., for including other kinds of reconfigurable hardware.

For deriving an instantiated QualiMaster platform, i.e., a platform that is tailored towards the configured resource pool and the specified data processing pipelines, the infrastructure users (cf. deliverable D1.2) define a valid configuration using the QualiMaster infrastructure configuration tool. As soon as data sources or algorithmic families are configured, the Platform Administrator can execute a variant of the platform instantiation process to obtain the programming interfaces (family, source, sink) that Algorithm Developers must comply with for proper execution. Algorithm Providers package their implementation components into extension packages. In turn, an extension package can be loaded / updated by the Platform Administrator to the Algorithm Repository through the configuration tool leading to the creation / update of the respective algorithm configuration. Thereby, the configuration tool analyzes the implementation (or in case hardware-based implementations a manifest), extracts the relevant configuration information and checks the implementation for compliance with the configured families and pipelines. This is indicated in Figure 18 b) as feedback from Instantiation to Configuration. After completion of the configuration activities, the Platform Administrator uses the valid configuration to derive the instantiated QualiMaster platform and deploy it to the compute cluster. We will discuss the detailed derivation process below in this section.

During operation, the instantiated QualiMaster platform (or the respective pipeline) is started as illustrated in Figure 18 c). While startup, the initial values of the runtime variables are determined, either from the configuration or the adaptation mechanism, i.e., initial algorithms are selected and their functional parameters are determined. Thereby, the actual hardware-based algorithms are loaded into the reconfigurable hardware as described in Deliverable D5.1. At runtime, the adaptation mechanism optimizes the execution of the pipelines.

The Instantiation Process of the QualiMaster platform is specified in terms of the Variability Instantiation Language (VIL) [I13, I14b, EEK+14]. In the remainder of this section, we provide an overview on the current state of the platform instantiation process and the artifacts that are derived. Although the related VIL specification can be seen as another type of model, we consider the details of the instantiation process on VIL level as out of scope of this deliverable. Please refer to the VIL language specification [I14b] for a detailed description of the actual version of VIL, its concepts and operations.

The QualiMaster instantiation process consists of two parts, one targeting the interfaces to be implemented by domain-specific components and one creating the artifacts to be executed, which

integrate the different forms of algorithms discussed in Section 2.3.2. We explain these two parts in more details in the following paragraphs.

The **creation of interfaces for domain-specific artifacts**, such as the interfaces for data sources and algorithmic families can be executed separately in order to support the Platform Administrator in the communication with the Algorithm Provider. Specific interfaces for pipeline sources and algorithmic families allow Algorithm Providers to focus on the realization of algorithms rather than constructing an overall topology or implementing the mechanisms to adapt a pipeline at runtime. According to the configuration of the algorithmic families, the interfaces are instantiated, including the input and output interfaces for the field types, methods to adjust functional parameters as well as algorithm behavior method(s) to be implemented by an Algorithm Provider. Similarly, interfaces for data sources (focusing on the pipeline input items) and data sinks (focusing on the pipeline output items) are derived.

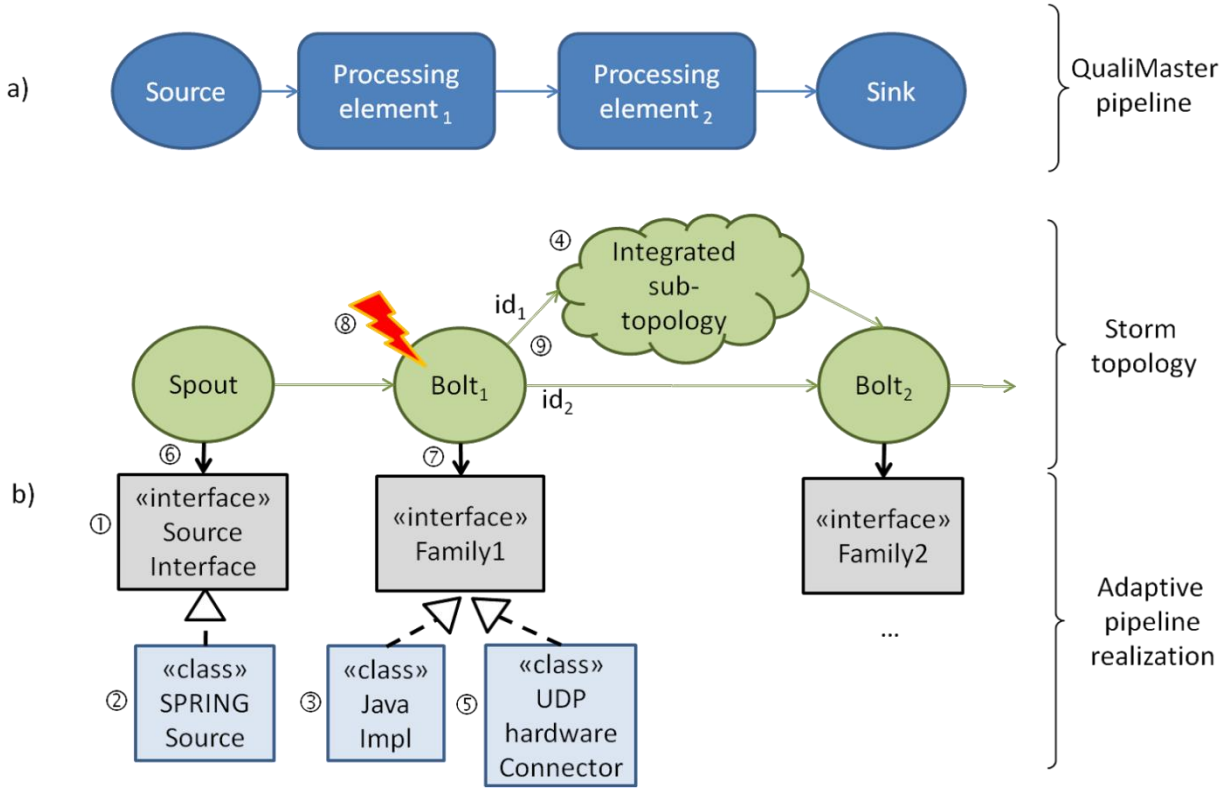

Figure 19: Overview of the VIL instantiation process for QualiMaster pipelines.

The **derivation of execution artifacts realizes** the integrated overall Storm topology implementing the configured data processing pipeline. We distinguish between the logical data processing pipeline defined in the QualiMaster configuration illustrated in Figure 19 a) and the realization in terms of an adaptive Storm topology shown in Figure 19 b). Please note that the processing pipeline abstracts over the processing families, while the realizing Storm topology implements the alternative family elements as well as the dynamic switching among them. Based on the already created interfaces for domain-specific or application-specific artifacts ① and the programmed component implementations for data sources ②, algorithms ③ and sub-topologies ④, the derivation of the Storm topology happens in two steps, namely, the creation of connector

implementations for hardware-based algorithms as well as the creation of the actual topology. In particular, we generate:

1) *Connector implementations* for the family interfaces ⑤. As QualiMaster unifies software- and hardware-based processing, specific connectors from Storm to hardware-based execution are needed. These connectors realize the communication with the Maxeler DFEs according to a generic communication protocol based on UDP (see D3.1), i.e., sending of data as well as receiving calculation results. A connector implementation is only instantiated if a family contains hardware-based algorithms. Please note that switching among hardware-based algorithms involves the Coordination Layer (see D5.1 and D5.2).
2) The *Storm topology* realizing the adaptive data processing pipeline as specified in the infrastructure configuration. The instantiation of the topology starts at the data sources (corresponding to Storm Spouts) and follows the configuration references to the processing elements (corresponding to Storm Bolts). Thereby, the instantiation generates recursively required Spout and Bolt classes for the entire pipeline until the pipeline sinks are reached:
    - A Spout ingests the data into the topology. Therefore, the pipeline instantiation refers to the data source interface as well as the configured implementation component realizing the interface ⑥. Please note that through the source interface parameters for changing the input data can be specified, e.g., to realize source-based data filtering.
    - A Bolt realizes the computation in a Storm topology. Therefore, the instantiation process produces code that refers to the respective family interface ⑧. In contrast to Spouts, the actual implementation of the family interface may differ, e.g., in terms of a Java-based algorithm or a hardware-based implementation. The instantiation process also generates a family-specific Storm signal receiver, to enable dynamic switching among the algorithms in a family, thus the different forms of implementation. The signal receiver handles algorithm change as well as the setting of functional parameter upon signals sent by the Coordination Layer through the Storm signal mechanism.

      As explained in Section 2.3.2, algorithms may also be implemented in terms of complete Storm sub-topologies for explicit distributed processing. In this case, the derivation process generates several streams with unique identification for the Storm filter mechanism in order to utilize or bypass the respective sub-topology ⑨. The current realization of the platform instantiation process aims at minimizing the signals to be sent in order to avoid additional synchronization among multiple Bolts. However, future evaluations may also show that separate bolts lead to better performance.

      For uniform handling of functional parameters, Storm sub-topologies do also implement the family-specific interface and realize the parameter setting accordingly.

      Furthermore, specific Bolts are generated for the Data Management Elements of a pipeline, i.e., the places where a data stream is transparently written to the Data Management Layer
    - Sinks are represented by Storm Bolts, which pass the data to the physical data access points for the QualiMaster applications.
3) The mapping specification for the pipeline, i.e., the relation between logical pipeline names and actual implementation names, for example relating a family element from the

Configuration to a specific Bolt. The mapping is stored in terms of an XML file along with the pipeline artifacts.

4) The deployable JAR file by executing the Maven Build specification from VIL.

Actually, the QualiMaster platform instantiation is able to derive the required artifacts for the Hello World Pipelines (D5.1) as well as for the more challenging priority pipeline (D5.1). In the future, we aim at realizing further artifacts, such as a dynamic integration of standalone sub-pipelines (either via Storm message mechanisms or network protocols akin to the hardware integration), the monitoring configuration for SPASS-meter [ES14], the Storm configuration files based on the general hardware configuration as well as the generation of further adaptation patterns (see D4.1) for the instantiated Storm topologies, Spouts and Bolts.

5 Architecture Deployment

QualiMaster performs stream and batch processing on a compute cluster in a distributed fashion, i.e., by utilizing Apache Storm, Hadoop, HBase, the Hadoop File System (HDFS) underneath as well as hardware-based processing on dedicated machines. Thus, the overall architecture is distributed due to data processing, data storage and management in order to support scalability and availability.

In addition, the QualiMaster layers and components work on top of these Execution System related components, communicate with them and even perform monitoring and changes at runtime. The overall view of the compute nodes, the deployed components and artifacts are depicted in terms of an UML deployment diagram in Figure 20, i.e., a high-level idealized graphical summary of the distribution and the deployment. Please note that that some parts of the diagram are replicated in a real world installation as indicated by the multiplicities, either in the nodes or at the communication links.

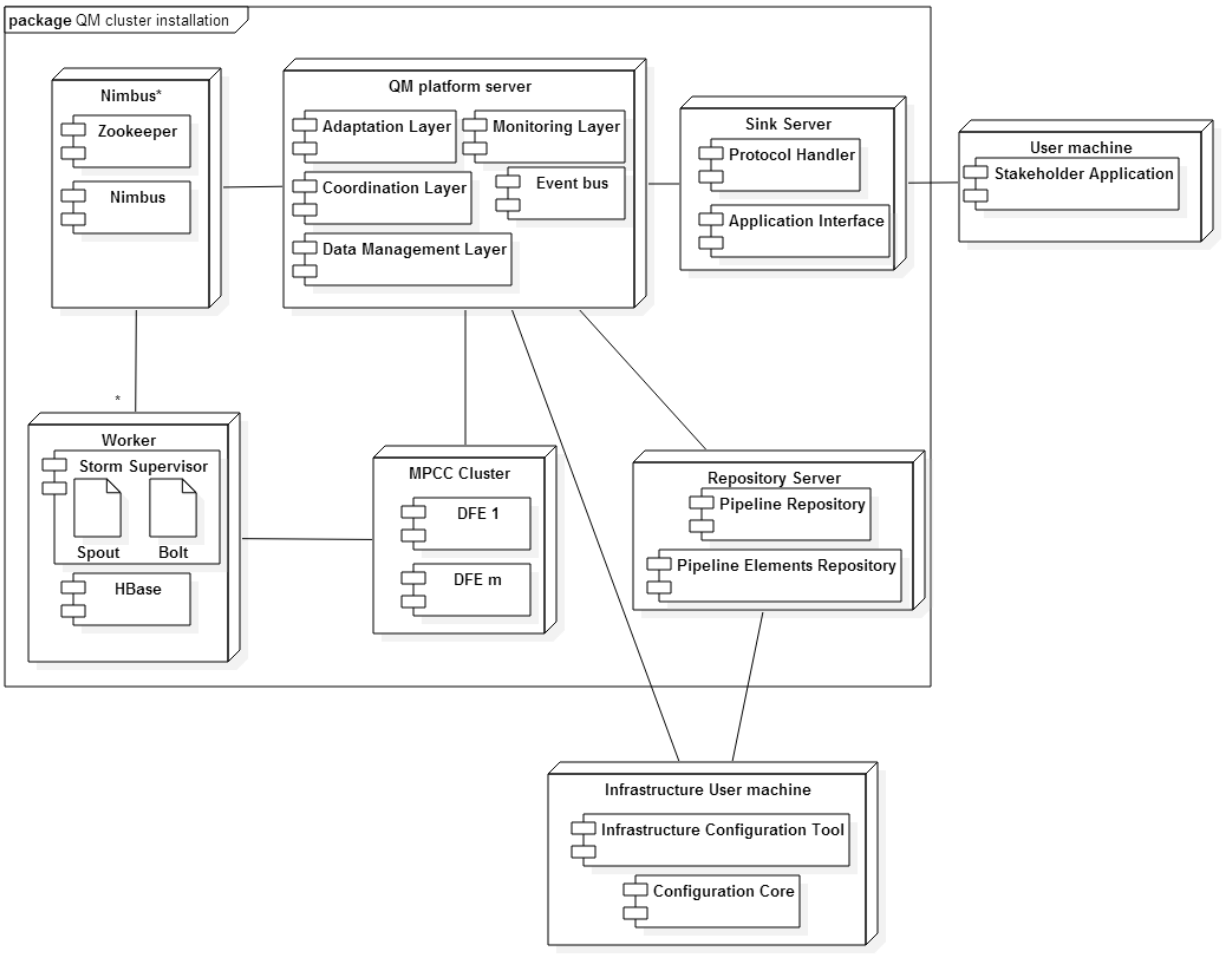

Figure 21: Logical deployment of the QualiMaster layers and components.

Basically, distributed data processing in QualiMaster is based on at least one coordinator node (Nimbus), several worker nodes (Worker) and the hardware processing in terms of at least one Maxeler cluster (MPCC Cluster). For a production cluster installation of Hadoop or Storm, typically several high-quality Nimbus machines are recommended in order to support failover and the more workers are available, the better the installed distributed database and the processing can be distributed and scale.

As described in this deliverable, the various layers of the QualiMaster platform will interact, currently in a local message passing fashion on the QM platform server. As pointed out in Section 2.3, we will consider a physical distribution of the QualiMaster layers if required in the future. Actually, the QM platform server communicates with Nimbus via Thrift Framework for Monitoring and the execution of the Coordination Layer via the Curator Framework. For executing pipelines, the QM platform server communicates with the (logical) Repository Server.

As discussed in Section 2.6, the Stakeholder Applications, which are external to the QualiMaster platform, communicate with the Sink Server for obtaining actual processing results and for sending user triggers to the QualiMaster platform. Further, the Infrastructure Users utilize the Infrastucture Configuration Tool on top of the Configuration Core on their local machines. These machines also communicate (possibly through a firewall) with the Repository Server for configuration with the QM platform to obtain runtime execution and adaptation information.

6 Status for the Priority Pipeline

As we explained in deliverable D5.1, we defined a QualiMaster Priority Pipeline for prioritizing the development of the infrastructure and jointly detecting and addressing challenges with respect to the design and implementation of the QualiMaster Infrastructre. The Priority Pipeline is now almost completed (we plan to finalize and verify everything by the end of December 2014) and helped the consortium in composing the basic infrastructure of QualiMaster, which primarily includes: (i) the implementation of the components, tools, and layers; (ii) the incorporation of the minimum set of functionalities (we expect to extend these functionalities in the following months); (iii) finalizing the interactions between tools and layers; (iv) creating a first family of algorithms with software- as well as hardware-based implementation of financial processing algorithms; (v) initial version of the dynamic pipeline adaptation using the incorporated software- and hardware-algorithms; and (vi) illustrating the abilities of combining data from the financial domain with data coming from the Social Web.

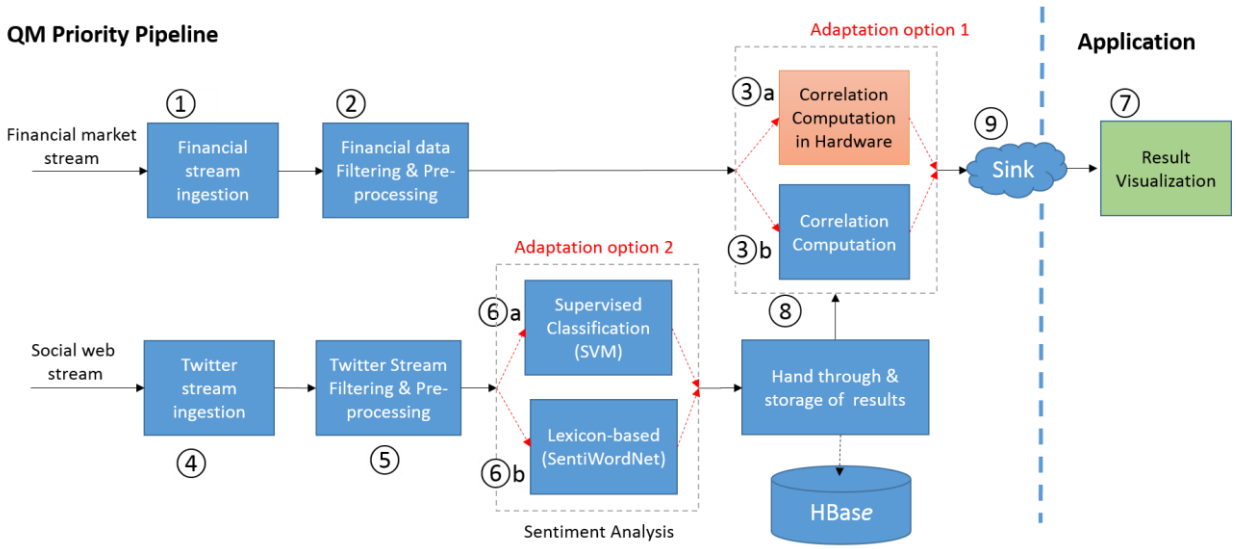

Figure 22: The QualiMaster Priority Pipeline.

Figure 22 illustrates the QualiMaster Priority Pipeline, and more specifically the steps composing this pipeline and their execution sequence. In the following paragraphs we present and briefly discuss each of these steps.

Step 1 - Financial stream ingestion

This step is responsible for fetching the financial data stream. It involves a Storm spout as a source. The source implementation involves the Spring API integration with the source interface. During the *initialization phase*, the implementation connects to Spring and performs a *loginAction*, a *getSymbolsAction* and a *startQuoteAllAction* through the Spring API. During the *execution phase*, the implementation waits for new data and for each new quote data, it returns the data to the spout so that it can be emitted to the rest of the pipeline.

Step 2 - Financial data Filtering & Pre-processing

This step is responsible for pre-processing the received data (from step 1) and to transform it into the necessary input of the algorithm. For the priority pipeline, the data is transformed from a tuple of the form {string *financialData*} to a tuple of the form {string *quoteID*, long *timestamp*, double *value*, int *volume*} and forwarded to step 3 to perform the actual computation.

Step 3 - Correlation Computation

This step is responsible for performing the actual computation over the received data, i.e. generating the pairwise correlation among all input elements (financial quotes or web quotes).
There are two distinct implementations of the *correlation* module; one implementation is *Hardware-based* while the other implementation is *Software-based* and more specifically it is implemented as a Storm topology. Both implementations realize the Hayashi-Yoshida correlation estimator and abide to the same implementation interface (i.e. the interface for this specific algorithmic family).

Step 3a - Correlation Computation in Hardware

This is the hardware-based implementation of the Hayashi-Yoshida correlation estimator. The module is mapped on a single FPGA device of a Maxeler C-Series server. The hardware-based implementation is connected via the TCP/IP protocol with a software-based unit that belongs to the Storm topology. The software-based unit is responsible for: (a) monitoring the input stream and sending the useful information to the FGPA-based unit; (b) setting the configuration of the reconfigurable module, e.g. the number of financial quotes; and (c) requesting and receiving the correlation estimator results from the hardware-based unit. The hardware-based system computes the correlation estimator over a time sliding window with step 1 second.

Step 3b - Correlation Computation in Software

Corresponds to the software-based implementation of the Hayashi-Yoshida correlation estimator is implemented as a Storm topology which implements the interface of the correlation algorithmic family (details are provided in the D2.1 deliverable). Hence, it can be considered as a processing element that can be interchanged with the hardware-based implementation at runtime.

More specifically, the implementation of the correlation matrix estimator consists of two main processing units; a *MapperBolt* and a *CorrelationBolt*. The *MapperBolt* is responsible for monitoring the input stream and evenly partitioning the workload amongst all the tasks (that are executed in parallel) of the *CorrelationBolt* and forward the appropriate input data to the responsible task. Each task of the *CorrelationBolt* performs the correlation computation, based on the user-defined window of interest, for the input chunk it receives and outputs the results for each pair it is responsible of monitoring.

Step 4 - Twitter stream ingestion

This step connects to the public Twitter Stream API to receive a sample of the posted tweets in the platform in real-time. There are several endpoints that can be used in Twitter's API. One endpoint sends a random sample (about 1%) of all tweets that are posted, while another endpoint sends a

filtered set of tweets based on user criteria. In the priority pipeline we use the filtered stream to get only tweets that mention certain keywords such as stock names of the market players that are considered in the analysis. Although both endpoints have the same limit on the volume of the tweets that can be received from the API (i.e., 1% of all tweets that are posted), the filtered stream provides more relevant tweets for the analysis. For example, if the limit is 200 tweets per second, both the filtered and unfiltered streams will contain up to 200 tweets per second, however, all tweets that come from the filtered stream are relevant, while only a small portion of the tweets that come from the unfiltered stream can be relevant. We use the Twitter4j library to connect to the API and convert the JSON format of the stream to Java objects that are then forwarded to the next bolt in the pipeline (i.e., step 5).

Step 5 - Twitter Stream Filtering & Pre-processing

Before performing sentiment analysis, some pre-processing of the tweets is required. For instance, the SVM-based sentiment classifier requires tokenizing the tweet text into bag of words, while the Lexicon-based sentiment classifier requires (in addition to the tokenization) Part-of-Speech tagging of the words. We use the IXA modules for tokenization and Part-of-Speech tagging as two subsequent steps in the pipeline in order to increase the concurrency of this step. The outcome of this step is an annotated text in the XML-based KAF representation model [BVS+09]. KAF is a language neutral annotation format representing both morpho-syntactic and semantic annotation in a structured format, which was originally designed in the Kyoto European project (http://kyoto-project.eu/), but it has since been in continuous development in the OpeNER European project (http://www.opener-project.org). We wrapped the IXA modules to run as bolts on a Storm topology and send the linguistic annotations to the sentiment classifier of step 6. For more details on this, please refer to D2.1.

Step 6 - Sentiment Analysis

In this step, the textual content of the tweets and their linguistic annotations from previous step are processed to evaluate the polarity and sentiment expressed in the text. This step uses the Sentiment Analysis family that is developed in WP2 and described in more details in D2.1. The outcome of this step is a time series of the sentiment scores. There are several parameters that are used to configure this component. One configuration parameter is the sentiment class to be evaluated, i.e. positive or negative sentiments. The user can select to monitor the development of the positive or negative sentiment towards a certain entity (e.g. stock/market player) over time. Another configuration parameter is the granularity of the produced time series (e.g. seconds, minutes, hours etc). In addition a functional configuration parameter can be used to set the threshold for the sentiment scores, which should be used by the classifiers to decide whether a tweet is positive, negative or neutral.

In the current version of the priority pipeline two alternative family members have been implemented for sentiment classification: a SVM-based and a lexicon-based sentiment classifier, which are described below. Both approaches have been implemented as a Storm sub-topology that is then integrated in the full stream processing topology.

Step 6a - SVM-based Sentiment Analysis

In step 6a, the SVM-based sentiment classifier belongs to the supervised approach of sentiment analysis. More details on this can be found in D2.1. We use three one-vs-all classifiers: positive-vs-all, negative-vs-all and objective-vs-all, whose results are then combined to decide (using the predicted score and the defined thresholds) whether a tweet belongs to the positive, negative or objective class, respectively.

Step 6b - Lexicon-based Sentiment Analysis

Alternatively, the SentiWordNet sentiment evaluator can be used in step 6 (step 6b). It belongs to the lexicon-based sentiment analysis, which follows an unsupervised approach. More details on this is provided in D2.1. Similar to the SVM-based sentiment classifier, the SentiWordNet sentiment classifier evaluates the (positive, negative and objective) sentiment scores of the tweet and combines them to decide on the most probable sentiment class. The results are then aggregated over time to produce the time series.

Step 7 - Result Visualization

This is an application in which the user will see the processing outcomes. The first version of the corresponding application will provide basic visualization, including the financial correlation matrix of several market player pairs, the social media correlation matrix of the same market players, the real time change of the correlation tables, the changes of the correlation of market player pairs over time as historical chart. It will also provide several filter sliders that allow zooming into the correlation tables for focusing on specific market player pairs and/or data range values.

Step 8 - Hand through & storage of results

As mentioned earlier, the results of the sentiment analysis step (step 6) is a time series showing the development of the sentiment over time. The format of the time series is similar to the format of the output of step 2. It includes an id representing the monitored entity (e.g. stock name), a timestamp, the number of tweets mentioning the entity in the given time window, and the percentage of positive or negative tweets in that window. The output has the form {string *entityID*, long *timestamp*, double *sentiment percentage*, int *volume of tweets*}. The output time series is then handed over to step 3 for computing the correlation between the different entities based on the results of the sentiment analysis in a similar way to the computation based on the financial stream. In addition, the output of step 6 is optionally stored locally in an HBase table for future analysis. Moreover, the produced time series can be sent directly to the sink in order to be accessed by the applications and the visualization tools in order to, for instance, monitor the development of the sentiments towards a specific market player in real-time.

Step 9 - Sink

This step is responsible is responsible for gathering the results from all previous processing elements (i.e. multiple *CorrelationBolt* tasks) and forwarding the results to the application for visualization. It consists of a bolt that interacts with an implementation of the *sink* interface. The

implementation is handles the communication of the Qualimaster infrastructure with the visualization application.

Note that the development of the priority pipeline is almost completed. This includes QualiMaster Configuration Environment including the domain-specific configuration frontend and the platform derivation process, the layers composing the QualiMaster Runtime Platform, and the steps composing the QualiMaster Priority Pipeline. We expect to finalize the development by end of December 2014, and in particular complete the final integration and verify the quality and efficiency of the overall processing. We plan to use this priority pipeline for creating early demonstrators of our work in QualiMaster, and thus allow the discussion of the QualiMaster capabilities with interested parties.

7 Plan for validating and evaluating the infrastructure

As presented and explained in the previous sections, the QualiMaster infrastructure is a complex system comprising a variety of layers as well as design and runtime components. This implies that also a more advanced validation and evaluation strategy is required, in order to assess the functionality, performance, and sound integration of the different components and layers that compose the infrastructure.

Firstly, the individual algorithms and processing elements have to be tested for correctness and evaluated for quality of results. This especially imposes challenges for algorithms, which have been implemented as Storm topologies or on the reconfigurable hardware. Part of these testing and evaluation activities are done in the individual WPs, where those algorithms are developed. Validation with respect to performance on this fine-granular level in also required for the processing elements (i.e., the algorithms in operation) to see if the algorithms perform as expected with the current data load and in the context of the QualiMaster platform and to collect quality profiles to be considered at runtime for adaptation. This activity is supported by the measuring tools foreseen in the QualiMaster infrastructure.

Going one granularity level up, full pipelines must be also tested for correctness and evaluated for performance, e.g., with respect to time behavior, in particular latency and quality. This again is partially supported by the measuring tools of the infrastructure. However, it also requires explicit testing and evaluation. The correctness of the pipelines also relies on the correct functioning of the QualiMaster configuration tools. Therefore, thorough testing of the design time tools, such as EASy-Producer, the domain-specific configuration frontend, the QualiMaster Infrastructure Configuration Tool, and the platform instantiation process is also crucial for ensuring the quality of the QualiMaster infrastructure. A further important building block for evaluating the performance of the pipelines in a systematic and repeatable way (as well as for scalability experiments) are simulations were streaming data are captured and thus replayed. Infrastructures such as Apache Kafka (cf. Section 2.3) will be evaluated and used for this purpose. The performance of pipelines also depends on the adequate implementation and integration of the QualiMaster platform, on which the pipelines will run. This also includes the adaptation mechanisms, which manipulate the pipeline at runtime. Thus, testing of the platform as well as evaluating and validating the adaptation mechanism are further building blocks of the QualiMaster infrastructure validation and evaluation strategy.

The remaining of this section, discusses the various testing, evaluation, and validation activities required for the systematic validation and evaluation of the QualiMaster Infrastructure. More specifically, Section 7.1 discusses the measuring and evaluation dimensions that we consider. Then, Section 7.2 presents the evaluation of the components and layers, Section 7.3 the evaluation of the individual algorithms, and Section 7.4 the evaluation of the pipelines. Finally, Section 7.5 introduces and explains our plans for the overall validations and testing of the infrastructure.

7.1 Measuring and Evaluation Dimensions

Measuring and (technically) evaluating the QualiMaster layers, data sources, sinks, and algorithms happens along the quality dimensions identified as the most important for the work performed in QualiMaster. As discussed in the D4.1 deliverable, and specialized for data processing algorithms in the D2.1 deliverable, these dimensions are the following:

- Time-behavior, in particular latency, throughput and the enactment delay.
- Resource utilization, in terms of memory consumption, the use of physical devices and bandwidth.
- Functional suitability, namely accuracy, believability, relevancy, and completeness.
- Scalability in the context of Big Data applications, in particular volume, volatility, and velocity using different data sources (variety) operationalized in terms of runtime measurements.

Furthermore, we also aim at measuring the overhead of the processing and the runtime monitoring. In addition to the enactment delay, an overhead caused by adaptation, we are in particular interested in the execution time of runtime decision making in the Adaptation Layer as well as the monitoring overhead caused by the Monitoring Layer. Understanding the characteristics of the overhead and the causes, we will optimize and tune the QualiMaster layers and components, i.e., we aim at reducing the overhead to its inevitable required for flexible and adaptive realtime, such as low-latency and data stream processing.

7.2 Evaluating the components and layers

Individual component evaluation will be performed each time the implementation of a specific component is modified. The evaluation of each QualiMaster component will be tested and validated with respect to correctness by simulating calls to the methods implementing the interface of the particular component. As long as the invoker of the implementation maintains the interface upon which the implementation was created, then the procedure is transparent for the implementation itself, and, thus it can be tested separately.

To assist this procedure, we have set up and already use a continuous integration and testing environment in terms of Jenkins[13]. Jenkins is a scalable server software that regularly obtains the most recent versions of the QualiMaster components from the central source code repository, compiles them according to their dependencies, and performs regression tests. For example, the Configuration Core, namely EASy-Producer, is subject to automated regression tests using several hundreds of test cases of different granularity, ranging from method unit tests up to scenario tests. Currently, we are preparing a testing environment for Storm-based regression tests in order to enable regression testing for the layers of the QualiMaster runtime platform (most of the layers actually rely on Storm functionality or require appropriate mockups) and the generated and integrated pipelines. In general, we aim at a test high coverage of the individual layers. Due to pragmatic reasons, in particular due to generated code parts such of the languages in the Configuration Core or the graphical pipeline editor we aim at least an average code coverage (C0 coverage) of 80% and higher coverage for individual components if applicable.

[13] http://jenkins-ci.org.

7.3 Evaluation of the individual algorithms

The individual data processing algorithms that are developed in WP2 for financial and social web data streams will be tested and evaluated independently within WP2. The quality and performance of each individual algorithm will be evaluated with appropriate data sets in order to create quality profiles for each algorithm. The quality profiles will describe the characteristics of the algorithms based on the quality dimensions and parameters taxonomy that was defined in WP4 and described in the D4.1 deliverable. A quality profile will be generated for each individual algorithmic family member to distinguish them based on their quality and performance trade-offs. The adaptation layer will then use these quality profiles to decide at run time on the selection of the family members for each step in the processing pipelines to fulfill the SLA or adapt to changes in the execution circumstances.

Data processing algorithms that have been implemented in hardware will be tested following the same methodology as the corresponding software-based algorithms. Thus, we will use the same data sets and references points, e.g., official software distribution, algorithms implemented in hardware will be tested for correct functionality and integrity. Hardware-based implementations will also use the same quality profiles as software-based implementations, proving that the hardware implementation is functional equivalent to the software one.

We will also perform a series of supplementary tests for evaluating the aspects that are related to the hardware implementation of the algorithms. For example, to evaluate hardware performance we will use specific representatives benchmarks and compare the execution time using hardware-based implementation with the execution time without it. The fraction of the two processing times will give the metric of the hardware performance against the software performance. The main drawback for hardware, is that usually even similar implementations need different hardware designs. This lack of flexibility, makes the design of several variations of the same algorithm, difficult. Designers usually implement the most common variations of an algorithm and that restricts the hardware design diversity. Following these two differences the quality profile for hardware-based implementation of algorithms is the same as the software one, with performance boosting but less flexible and sometimes with reduced functionality.

7.4 Evaluating the included pipelines

The evaluation of the components and layers (i.e., Section 7.2) and of the individual algorithms (i.e., Section 7.3) will be followed by the evaluation of the pipelines. In order to easily validate the pipelines incorporated in QualiMaster, we plan to define and use data sets along with a set of expected outcomes. The main idea is that each pipeline with be given a small collection of data sets, and for each data set we will map the returned results against the expected outcome. Note that defining the given data sets and especially the expected outcomes will be performed in close collaboration with the QualiMaster application partners.

In addition to the validation of the pipelines, we will also perform various evaluation tests focusing on performance (Section 7.1). For example, we will test the time-behavior of each running pipeline and its scalability with respect to the volume, volatility, and velocity of the given data streams.

7.5 Validation and Performance testing of the infrastructure

In Section 5, we described the QualiMaster Priority Pipeline. As we already explained, this pipeline contains the most challenging functionalities and integration tasks, such as the interconnection between hardware- and software-based stream processing algorithms, interaction between layers and tools, and the initial version of the dynamic pipeline adaptation. Therefore, testing and evaluating the priority pipeline will allow us to verify that the particular functionalities are correctly integrated and operate as expected. In addition, the evaluation results will provide initial insights to the capabilities of the QualiMaster, with respect to efficiency, effectiveness, and scalability.

The above (i.e., Section 7.2-7.4) are of course a simple methodology for performing an initial validation of the developed QualiMaster Infrastructure. In the future, we will work on a systematic methodology for validating QualiMaster. More specifically, we plan to create various collections of test cases. The first collection will contain use cases focusing on generic functionalities and capabilities provided by the QualiMaster Infrastructure. The other collections will focus on particular processing pipeline, i.e., the use cases in each collection will correspond to a particular pipeline. In addition to use cases, we will also plan to create data sets by collecting real world data, i.e., financial stream data coming either from our financial partner (Spring) or from Web data. The goal is to be able to test QualiMaster using the same data and thus monitor modifications in the produced results or the processing performance. Using these use case collections and data sets, we plan to define simulations, which we plan to execute and re-execute over the QualiMaster Infrastructure.

Please note that this deliverable will be followed the D5.3 “QualiMaster Infrastructure V1” and D5.4 “QualiMaster Infrastructure V2”. Both these deliverables will report and discuss the methodology that we will follow for the QualiMaster systematic evaluation methodology. They also provide the performance evaluation results for the individual processing analysis pipelines as well as for the entire infrastructure.

In order to enable the smooth validation of the infrastructure, we have already planned (and verified the availability) of a minimum set of hardware resources that we foresee will be required, or even useful, to have. This includes hardware resources providing a cluster infrastructure as well as a FPGA-based platform. The following paragraphs provide an overview to these resources.

(I) Cluster Infrastructure - Mikio Cluster

The Mikio Cluster corresponds to a cluster infrastructure that is relatively small. In particular it includes a cluster with 4 machines, each having 2 AMD cores, clocked at 2.1GHz, 8GB RAM and 80GB HDD. Our plan is to use the Mikio Cluster for performing initial experiments focusing on the validation aspect of our implementation.

(II) Cluster Infrastructure - SoftNet Cluster

It corresponds to a cluster with 24 machines, 18 machines with Dell PowerEdge R300 Quad Core Xeon X3323 2.5GHz and 6 machines with Dell PowerEdge R310 Quad Core Xeon X3440 2.53GHz. Each machine has 8GB RAM, 500GB HDD, and Gigabit Ethernet connection (1Gb/s).

Our plan is to use the SoftNet Cluster for performing the scalability experiments of the QualiMaster infrastructure, i.e., once the verification is validated using the QM-Mikio Cluster.

(III) FPGA-based platform - Maxeler PC-C series

The Maxeler MP-C series servers offers high speed stream-based processing, i.e., the type of processing which will mostly apply in the QualiMaster project. The servers provide the closest coupling of DFEs and CPUs for computations requiring the most data transfer. The server consists of multiple Intel Xeon cores with many GBs of RAM for the CPUs and multiple FPGA devices (DFEs, i.e. dataflow engines) closely coupled to the CPUs. Each dataflow engine is connected to the CPUs via PCI Express, and DFEs within the same server are directly connected with MaxRing interconnect. The servers also support optional Infiniband or 10GE interconnect for tight, cluster-level integration. In terms of design tools, the Maxeler card comes with a high-level compiler called MaxCompiler, which enables users to describe the hardware circuit using a high-level Java-based description. The Maxeler tools offer a dataflow model of the application, with the user-indicated parallelism being mapped to the hardware by the tools.

(IV) FPGA-based platform - Convey HC series

The Convey HC series systems are hybrid-core computers, which can be efficiently used as platforms on high performance computing (HPC) field. The HC systems allow for FPGA multiprocessing with a large-memory model. The Convey's infrastructure combines a 4-core Intel Xeon processor and a Convey-designed coprocessor based on four user-programmable Xilinx Virtex-5 XC5VLX330 Field Programmable Gate Arrays (FPGAs), with its own high bandwidth memory, i.e. 80 Gigabytes/sec, addressed and cache coherent memory subsystem. Last, the Convey systems accelerate computing by providing higher absolute performance, increased functionality, and improved efficiency. The Convey HC-2 systems increase application performance tens of times over commodity servers.

8 Conclusions

This deliverable presented the basic QualiMaster Infrastructure, and reflects the requirements and use cases collected and analyzed in WP1, i.e., as these have been reported in deliverables D1.1 and D1.2. With respect to the actual system, we have introduced and explained the configuration environment, the runtime platform, the repositories used by the runtime platform, the applications using QualiMaster, and the external interface of the runtime platform that enables these applications to send user triggers for adjusting the processing on demand. We have also discussed the communication inside the layers and components composing the infrastructure by providing a set of the possible communication cases and concrete communication examples. Furthermore, we described the process used for the instantiation of the infrastructure as well as the (distributed) deployment of the architecture. In addition, we reported the status of the priority pipeline that indicates the infrastructure's current status (i.e., implementation, integration, etc.), and discussed our plan for validating and evaluating the infrastructure.

The infrastructure reported in this deliverable forms the basis on which all QualiMaster work packages will develop algorithms, components, and tools. The future developments as well as integration results will be reported in following deliverables. More specifically, deliverable D5.3 will provide the first version of the integrated infrastructure (month 21) and deliverable D5.4 will provide the final version of the infrastructure (month 33). In addition to the integration work, in the upcoming months we will also extensively work on the validation and performance evaluation of the algorithms and components incorporated in QualiMaster and of the entire infrastructure.

References

[BSG+09] Y. Brun, G. Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Müller, M. Pezzè, M. Shaw. Engineering self-adaptive systems through feedback loops. Software Engineering for Self-Adaptive Systems, volume 5525, 48–70, 2009.

[BVS+09] W. Bosma, P. Vossen, A. Soroa, G. Rigau, M. Tesconi, A. Marchetti, M. Monachini, and C. Aliprandi. Kaf: A generic semantic annotation format. In Proceedings of the GL2009 Workshop on Semantic Annotation, 2009.

[EEK+14] H. Eichelberger, S. El-Sharkawy, C. Kröher, K. Schmid. EASy-producer: Product Line Development for Variant-rich Ecosystems. International Software Product Line Conference (SPLC) - Volume 2, 133–137, 2014.

[ES14] H. Eichelberger K. Schmid. Flexible resource monitoring of Java programs. Journal of Systems and Software, 93:163–186, 2014.

[GHJ+00] E. Gamma, R. Helm, R. Johnson, J. Vlissides. Design Patterns Elements of Reusable Object-Oriented Software. Addison-Wesley, 2000.

[I12] INDENICA Consortium. Open Variability Modeling Approach for Service Ecosystems. Technical Report Deliverable D2.1, 2012.
URL → http://www.indenica.eu

[I13] INDENICA Consortium. Variability Engineering Tool (final), 2013.
URL → http://www.indenica.eu

[I14] IVML team. Indenica variability modeling language: Language specification, version 1.22. Technical report, 2014.
URL → http://projects.sse.uni-hildesheim.de/easy/docs/ivml_spec.pdf.

[I14b] VIL team. INDENICA variability implementation language: Language specification, version 0.93. Technical report, 2014.
URL → http://projects.sse.uni-hildesheim.de/easy/docs/vil_spec.pdf.

[IBM06] IBM. An Architectural Blueprint for Autonomic Computing. Technical report, June 2006.
URL → http://www-01.ibm.com/software/tivoli/autonomic/pdfs/AC_Blueprint_White_Paper_4th.pdf

[K95] P. Kruchten. The 4+1 View Model of Architecture. IEEE Software 12 (6), 1995.

[KC03] J. Kephart D. Chess. The Vision of Autonomic Computing. IEEE Computer, 36(1):41–50, January 2003.