www.qualiMaster.eu

QualiMaster

Quality-aware Processing Pipeline Modeling

Grant Agreement No. 619525

Deliverable D4.1

Work-package	WP4: Quality-aware Configuration and Adaptation of Stream Processing Pipelines
Deliverable	D4.1: Quality-aware Processing Pipeline Modeling
Deliverable Leader	SUH
Quality Assessor	Ekaterini Ioannou
Estimation of PM spent	10
Dissemination level	PU
Delivery date in Annex I	31.12.2014
Actual delivery date	31.12.2014
Revisions	8
Status	Final
Keywords:	Quality-aware configuration, platform instantiation, quality-aware adaptation, enactment patterns, quality taxonomy, tool support

List of Authors

Partner Acronym	Authors
MAX	-
LUH	Mohammad Alrifai
SUH	Holger Eichelberger, Cui Qui, Roman Sizonenko
SPRING	Stefan Burkhard
TSI	Gregory Chrysos

Table of Contents

Table of Contents 4
Executive summary 5
1 Introduction 6
2 Terminology and Basic Concepts 8
2.1 Terminology 8
2.2 QualiMaster Pipeline Execution Model 10
3 Challenges and Approach 11
4 Related work 14
4.1 Quality Parameters and Quality Analysis 14
4.1.1 Software Quality 14
4.1.2 Data Stream Processing Quality 15
4.1.3 Data Quality Analysis 18
4.2 Configuration 19
4.3 Adaptation 21
5 Requirements 24
5.1 Requirements for the QualiMaster Configuration Model 24
5.2 Requirements for the QualiMaster Adaptation Model 26
6 QualiMaster Quality Dimensions and Quality Analysis 27
6.1 Quality Survey 27
6.2 Quality Taxonomy 29
6.2.1 Performance Efficiency 29
6.2.2 Functional Suitability 32
6.2.3 Scalability 33
6.2.4 Summary 34
6.3 Pipeline Quality Analysis 35
7 QualiMaster Configuration Model 36
7.1 Configuration Modeling Overview 36
7.2 QualiMaster Configuration Meta Model 38
7.2.1 Configuration Meta Model Overview 38
7.2.2 Observables 40
7.2.3 Resource Pool (Execution Layer) 41
7.2.4 Data Management 43
7.2.5 Data Processing Algorithms 46
7.2.6 Algorithm Families 47
7.2.7 Data Processing Pipelines 48
7.2.8 Adaptivity 52
7.2.9 Infrastructure 53
7.3 Tool Support 53
8 QualiMaster Adaptation Model 57
8.1 Approach Overview 57
8.2 Enactment Patterns 59
8.2.1 Pattern Structure 59
8.2.2 Primary Enactment Patterns 60
8.2.3 Secondary Enactment Patterns 64
8.2.4 Application of Enactment Patterns 70
8.3 Flexible Adaptation Behavior Specification 70
8.3.1 Concept Overview 71
8.3.2 VIL Core Concepts 74
8.3.3 rt-VIL Language Elements 75
8.4 Tool Support 84
9 Conclusion and Future Work 86
10 References 89
Appendix A: QualiMaster Configuration Meta Model in IVML 98
Appendix B: rt-VIL Grammar 111

Executive summary

Quality-aware configuration and runtime adaptation provides flexibility before, and, at runtime in the QualiMaster project. Configuration targets flexibility before runtime in terms of a consistent customization of a real-time data analysis platform in order to support users and developers in applying, extending and adopting the QualiMaster approach. Runtime adaptation aims at flexibility at execution time, i.e., to cope with suddenly occurring or even unforeseen changes of the environment, e.g., to handle a massive increase of data volume due to hectic market times while keeping the quality of the data analysis. Configuration and adaptation are closely related, as the configuration determines the adaptation opportunities, i.e., the adaptation space, and the adaptation needs to take configuration settings into account in order to act according to the preferences of the users.

This deliverable reports on the concepts, models and tools developed in the project to realize quality-aware configuration and adaptation. Thus, the main topics of this deliverable are quality, configuration and adaptation in QualiMaster. As a basis for configuration and adaptation, we discuss a common quality taxonomy derived from a quality survey we conducted. Regarding configuration, we provide detailed insights into the quality-aware Configuration Meta Model, the automated instantiation of the configuration for customization of software artifacts and the tooling, which allows infrastructure users perform the configuration including the resource pool, the algorithm families and the data analysis pipelines. Finally, we discuss our approach to quality-aware integration in terms of an adaptation specification language integrated with the configuration approach as well as enactment patterns to realize dynamic modifications in the running QualiMaster platform.

1 Introduction

In this deliverable, we discuss the fundamental concepts for modeling the quality-aware configuration of the QualiMaster infrastructure before runtime as well as the dynamic adaptation and optimization of the QualiMaster platform at runtime. In particular, this deliverable describes our approach on how to unify these two distinct phases in the software lifecycle of the QualiMaster platform and, thus, respective approaches in these two fields of research and development.

In QualiMaster, *configuration* refers to the activities for consistently customizing and tailoring the capabilities of a certain infrastructure instance for a given setting. One particular example for such a setting is the application of the QualiMaster infrastructure to the analysis of systemic risk in the financial domain on the cluster hardware for example the one used in the project at the Technical University of Crete. Performing this customization includes the definition of the physical (general-purpose as well as reconfigurable) hardware used for computation in terms of a heterogeneous resource pool, the algorithms and algorithm families, the data analysis pipelines as well as quality constraints, e.g., formalizing external service level agreements (SLAs). Ultimately, the configuration activities lead to a consistent configuration, which allows the automatic customization, packaging and deployment of the artifacts constituting the QualiMaster infrastructure for the particular setting. These configuration activities aim at high degree of flexibility in order to support the application and adoption of the QualiMaster solution by domain experts, and, from a wider perspective, also at easing application and adoption of the project results even in other application domains.

In contrast, *adaptation* refers to the (autonomous) modification of the configured and running platform in order to cope with unforeseen changes in the environment. In QualiMaster, runtime adaptation allows, e.g., to react appropriately on input load changes to efficiently use the computational resources or to automatically focus the risk calculation in hectic market times on dynamically selected markets to provide analysis results of high relevance to the user. Therefore, we rely on the concept of an adaptive data analysis pipeline, i.e., the specification of the data flow and the data processing in terms of algorithm families. An algorithm family groups algorithms with the same analysis semantics, but different quality, performance or resource tradeoffs. Selecting the most appropriate algorithms in a data analysis pipeline in order to optimize the processing for the actual situation is one of the most important tasks of the adaptation in QualiMaster.

In general, we aim at a *unified quality-aware configuration and adaptation model* for real-time (low-latency) data stream processing. Thereby, we understand the term "unified" in two flavors. First, we aim at the unification of configuration and adaptation, in the sense that the configuration activities before runtime determine the adaptation space at runtime. Some examples that detail and limit the adaptation space are the heterogeneous resource pool, the algorithm families and their member algorithms, the specified processing pipelines as well as the quality constraints. Second, we aim at a unified treatment of software- and hardware-based execution, in particular to configure the presence of reconfigurable hardware and to dynamically utilize software as well as hardware-based execution in an opportunistic way. Achieving both unification flavors imposes several research challenges that we discuss in this deliverable and focus on in the work of WP4.

To achieve the overall goal, we rely on proven concepts and technologies from the area of Software Product Line Engineering (SPLE) [31, 96, 111]. SPLE is a successful approach to mass customization, configuration, tailoring, and reuse of software systems. Thereby, a software system is not considered as a single system, but rather as software product line (SPL), a family of similar, but differently configured systems. SPLE is successfully applied in different industrial settings, such as embedded systems or information systems and enables to manage complex configurations while reducing time-to-market as well as development efforts. Deriving a specific member from a product family (in QualiMaster the configured infrastructure instance) includes the configuration of the product (the QualiMaster platform), it's validation, and, finally, its derivation, i.e., turning the configuration into (software) artifacts. While traditional SPLE focuses on pre-runtime customization, Dynamic Software Product Lines (DSPL) [67, 71] aim at runtime (re-)configuration as one approach to realize adaptive software systems. However, current DSPL approaches typically utilize concepts and techniques from SPLE, but focus exclusively on the runtime aspects rather than linking configuration and runtime. Unifying aspects of SPL and DSPL and exploiting their common aspects and mutual benefits in the domain of adaptive real-time data stream processing is at the heart of the work in WP4.

This deliverable is structured as follows. In Section 2, we introduce terminology and basic concepts, e.g., the execution model for adaptive QualiMaster data analysis pipelines. Based on this terminology, we discuss in Section 3 the (research and development) challenges for our work in WP4 and introduce our overall approach. In Section 4, we provide an overview on the state of the art for configuration and adaptation approaches for software systems in general and for data stream processing infrastructures in particular. In Section 5, we analyze the requirements for our work on configuration and adaptation in QualiMaster drawn from the Description of Work (DoW) and other related deliverables such as D1.2 and D5.1. In Section 6, we detail the quality dimensions relevant for the QualiMaster project. The remainder of the deliverable focuses then on the configuration model and the adaptation model. Although the adaptation behavior is (initially) specified along with the configuration before runtime and relies on the QualiMaster Configuration (Meta) Model, we discuss configuration and adaptation in two sections according to their point in time when they mainly affect the QualiMaster infrastructure, i.e., configuration time and runtime. Thus, in Section 7, we discuss the Configuration (Meta) Model, its configurable elements and constraints. Further, we give an insight into the actual tool support for configuration used in QualiMaster. In Section 8, we discuss the QualiMaster Adaptation Model, in terms of the adaptation cycle, the concepts of the adaptation specification language as well as the state of the related tool support. In Section 9, we conclude this deliverable and give an outlook on future work in WP4. In the appendix, we provide an overview on the actual specification of the configuration meta model from Section 7 as well as the (initial) grammar of the adaptation specification language discussed in Section 8.

Relation to other deliverables:

- D4.1 is based on the use cases and requirements collected in D1. 2. A brief analysis of the requirements relevant to WP4 is given in Section 5. Further, WP4 conducted a consortium internal quality survey in order to create a common quality taxonomy for the project. Although we report on the quality taxonomy in Section 6, this is a form of requirements elicitation that indicated further requirements in D1.2.
- D2.1 relies on the algorithm-related aspects of the quality taxonomy discussed in Section 6 of this deliverable.
- D3.1 and D4.1 are loosely linked through the configuration of the hardware-based processing, in particular to reflect the Maxeler hardware used in the QualiMaster project, the generation of hardware connectors during platform instantiation, the opportunistic execution of analysis algorithms on reconfigurable hardware as well as common understanding of the quality of the data processing.
- D4.1 relies on D5.1, in particular the organization of the Configuration Meta Model as well as the design of the Adaptation Model in terms of the architecture of the QualiMaster infrastructure. In turn, the work on realizing the concepts of D4.1, the underlying monitoring, the infrastructure configuration tool support and the infrastructure derivation process links D4.1 to D5.2.

2 Terminology and Basic Concepts

This section defines the basic terminology used in this deliverable in Section 2.1 and the basic concepts on adaptive data stream analysis pipelines in Section 2.2.

2.1 Terminology

We now provide an overview on the basic terminology for this deliverable. This section is based on the DoW and terms defined in D1.2, i.e., terms are partly repeated here for convenience, but explained in the context of the project and the work in this work package.

- The *QualiMaster infrastructure* consists of an environment for adaptively executing data stream pipelines as well as related tools for configuring and managing that environment. We will call the execution environment the *QualiMaster platform*, or *platform* for short.
- The *Configuration Meta Model* (Variability Model[1] in SPLE) defines the valid configuration space, i.e., the configurations that describe meaningful and executable systems. This configuration space is defined in terms of configurable elements (called “variabilities” in SPLE), which may represent optional, alternative or multiple selections from related value domains. Further, constraints among the configurable elements express interdependencies, such as inclusions or mutual exclusions and impose further limitations of the validity of a configuration. The Configuration Meta Model is defined by the QualiMaster consortium using a textual modeling language and can be adapted or extended by expert users in order to apply the QualiMaster infrastructure to other settings or domains.
- A *Configuration* determines the actual settings of the configurable elements defined by the underlying Configuration Meta Model for a specific instance of the QualiMaster infrastructure for a given analysis setting. In particular, this includes the definition of the domain-specific analysis algorithms or data processing pipelines. A configuration is *valid* if it structurally complies to and fulfills all constraints defined by the Configuration Meta Model. The configuration can be defined and validated by an Infrastructure User (cf. D1.2) through the QualiMaster infrastructure configuration tool.
- The *derivation* or *instantiation* of the QualiMaster platform is the activity of tailoring the generic (template) infrastructure towards the application setting described by a valid Configuration. The derivation of the platform happens before its runtime, in particular before any adaptation activities are carried out. We focus on an automatic derivation of the platform by creating, modifying and deleting infrastructure artifacts as specified by the Configuration. In particular, this includes the derivation of the implementation of the adaptive data analysis pipeline for the actual execution system (Apache Storm) as well as the integration of the domain-specific (manually programmed) data analysis algorithms. As we describe in more detail in the following sections, the platform derivation is defined in terms of a Domain-Specific Language (DSL) so that modifications or extensions of derivation process or the Configuration Meta Model can be turned into software artifacts without modifying the underlying tool support.
- As introduced in D1.2, a quality parameter [101] is a measurable and quantifiable property of a computational element (also other terms are used in literature, e.g., quality dimension or quality attribute or quality property depending on the community [56]). A computational element can be a data processing algorithm, a data flow, a data analysis pipeline or a physical compute resource. Examples for quality parameters are numbers of items per time unit (data flow or data pipeline level), execution time or memory usage (compute resource level). We refer to a quality parameter as a single measurement. *Monitoring* is the activity of performing such measurements. We distinguish between primitive quality parameters

[1] Typically, this kind of model is called “Variability Model” in SPLE. Due to a possible mismatch with the terminology of the Data Stream community, we use in the QualiMaster project the term “Configuration Meta Model” instead.

directly measured by the infrastructure or the hardware and derived quality parameters defined in the Configuration (such as a domain-specific kind of throughput).

- As in D1.2, we call (the capture of) the evolution of a quality parameter over time in a certain setting a *quality characteristic*, e.g., the behavior of a stream-based correlation calculation processing algorithm under high load. Typically, quality parameters cannot be considered as constant, so that mathematical or statistical means must be applied to capture quality characteristics appropriately. Quality characteristics are defined for individual algorithms, propagated to algorithm families (through the selection of the actual algorithm) and, ultimately, to the end-to-end quality characteristics of a data processing pipeline.
- *Quality of Service* (QoS) refers to measurable and quantifiable quality parameters of a software product [37]. As we discuss in Section 4.1, standards are in place that aim at characterizing typical QoS parameters. Typically, the negotiated QoS a system must provide is captured in terms of *Service Level Agreements* (SLA).
- *Adaptation* (or *adaptive execution* of a pipeline) refers to autonomous changes determined and carried out by the QualiMaster platform in order to maintain the actual QoS of the data analysis and the efficiency of the use of physical computing resources. In contrast to the configuration activities before startup, adaptation happens at runtime of the QualiMaster platform. In turn, the adaptive execution relies on the configuration that implicitly defines the boundaries and the validity of the autonomous activities.
- *Enactment* or *execution of an adaptation decision* refers to the implementation of a decision made by the QualiMaster platform at runtime, e.g., by changing individual algorithms or algorithm parameters.
- *Reactive adaptation* is triggered by monitoring of the actual system state, e.g., to identify violations of SLAs. Reactive adaptation is performed when monitored values are outside the boundaries (given by the Configuration) and an immediate reaction is required. Reactive adaptation aims at a nearly immediate reaction on a situation and does not include planning or prediction of quality properties.
- *Proactive adaptation* aims at anticipating the short-term future development of the environment. This includes planning activities, the prediction of immediate quality properties or characteristics, the anticipation of unwanted situations or the prediction of the short-term evolution of the adaptation model along with impact records or recorded quality characteristics.
- *Reflective adaptation* aims at improving the quality of the configuration and the adaptation on meta-level on a large time scale. Therefore, the reactive adaptation in QualiMaster analyzes the overall operation of the infrastructure and the effects / impacts of reactive and proactive adaptation over time. Reactive adaptation aims at providing insights for the improvement of the configuration, the use of adaptation strategies or their refinement so that human Infrastructure Users (Platform Administrator, Adaptation Manager, and Pipeline Designer as introduced in D1.2) can improve and optimize the behavior of a running QualiMaster platform. Please note that we will not discuss the concepts for reflective adaptation in this deliverable, as the related task 4.4. is scheduled to start in month 18 of the project.
- *Cross-pipeline* adaptation aims at optimizing the pipeline execution across multiple running pipelines, in particular to free resources for value-added resource-consuming (offline) analysis tasks. Although we actually prepare the configuration and the platform instantiation for cross-pipeline execution as we will discuss in Section 7, the related task 4.5 is scheduled to start in month 18 of the project. Thus, we will not detail concepts for cross-pipeline adaptation in this deliverable.

2.2 QualiMaster Pipeline Execution Model

Pipelining is a well-known implementation technique for high performance architectures [75]. The idea is to break down the computation logic of a computing structure into sub-components and arranging them as a sequential chain, in analogy to a physical pipeline, so that the output of each element is the input of the next. Pipelining increases the concurrency in the computation by allowing processing *k* items at a time in a *k*-stages pipeline, thus increasing throughput.

In QualiMaster, a real-time data processing pipeline is a directed graph composed of algorithms (more precisely sources, processing families and sinks as we discuss below) that are put together to accomplish a specific data analysis task. Example tasks are:

- Analyzing the co-dependency matrix of a set of market players using financial streams of stock tick data.
- Analyzing the sentiment of user posts in the Twitter stream with respect to a market player.

Each pipeline receives a stream of (structured or unstructured) raw data as input (e.g., Twitter or stock market streams) and produces one or more streams of analysis results as output (e.g., time series of the sentiment analysis, or correlation measurement of a pair of market players).

In QualiMaster, a **data processing pipeline** consists of the following types of components:

- Data sources, which provide the raw input data. As discussed in D4.1, the data sources are handled by the Data Management Layer, in particular to avoid storing the same raw input data multiple times.
- Processing elements, which perform the actual data processing. Processing elements may
    - Represent an algorithm family and, thus, facilitate the execution of a dynamically selected algorithm from the respective families.
    - Interact with the Data Management Layer (see D5.1) and provide functionality to store or access intermediary processing data in terms of historical data. The actual data to be stored is determined during the design of the containing pipeline.
- Data sinks represent the conceptual endpoints of a pipeline and provide analysis results to the applications via the Data Management Layer.
- Data flows link sources with processing elements and sinks.

There can be different pipeline structures/compositions that fulfill the same task, known as execution paths. Typically, execution paths are implicit, i.e., by exchanging algorithms within their respective family, different execution paths are realized. Furthermore, there may be explicit alternatives designed into the pipeline in order to be decided at runtime, e.g., depending on quality metrics used in a pipeline constraint. The actual realization can be done using one concrete algorithm (or a combination of several algorithms) from the same family. Hence, there can be several implementations of the abstract pipeline depending on the selected execution path and the actual methods used in for each step. Thus, a QualiMaster data processing pipeline can be seen as a family of execution paths, i.e., dynamically instantiated pipelines.

3 Challenges and Approach

Achieving the overall goal of realizing a unified configuration and adaptation model for real-time data stream processing imposes several challenges. We describe the most relevant challenges for the QualiMaster project in this section, provide further detailed relations to related work in Section 4 and to the model concepts in Section 7 and 8, respectively.

C1 Unification of configuration and adaptation: Various approaches are described in literature that either focus on the (pre-runtime) configuration of systems or on (runtime) adaptation. However, approaches that unify both aspects, i.e., transfer configuration knowledge to runtime to support valid adaptation, contribute to a recent research topic as encouraged in keynotes such as [8, 68]. Please note that we aim at changing properties of the QualiMaster infrastructure at runtime rather than enumerating the potential alternative artifacts and selecting them at runtime as done in some related approaches, such as [6, 119]. In QualiMaster, this unification happens in terms of the Configuration (Meta) Model and the Adaptation Behavior Specification Model. On the one side, the Configuration Meta Model defines the space of valid infrastructure configurations. In particular, the QualiMaster Configuration Meta Model allows capturing runtime configurable elements (e.g., the actual selected algorithms for the processing families or individual quality parameters) as well as constraints on them such as SLAs. The values of these configurable elements are determined at runtime and, in turn, the runtime constraints become active at runtime, possibly leading to constraint validations adaptation must cope with. On the other side, the QualiMaster Adaptation Behavior Specification Model allows defining the adaptation behavior of the QualiMaster platform. This model relies on the (traditional and runtime) configurable elements and determines the adaptation behavior for different triggers, in particular violations of the runtime configuration constraints defined in the Configuration.

C2 Unification of software and hardware-based execution: Opportunistic usage of specialized hardware resources is a core concept of the QualiMaster project. Thus, a configuration of the QualiMaster infrastructure defines the available resource pool in terms of both, general purpose and specialized hardware. Please note that also the absence of specialized hardware is a configuration and, thus, limits the resource pool and the runtime adaptation opportunities. The configuration of the resource pool is used to opportunistically allocate analysis algorithms to the physical resources at runtime. In addition, data analysis algorithms (regardless whether they are designed for software- or hardware-based execution) as well as their quality parameters are treated in a unified way, i.e., they can be assigned to the same family in the Configuration and they are selected dynamically at runtime based on needs and available resources.

C3 Topology configuration: Adaptive data analysis pipelines are a further core concept of the QualiMaster project. Actually, a data analysis pipeline as described in Section 2.2 (or, more generally, a data analysis network or graph) can be represented by a topology. In SPLE, a topology is a set of interconnected configurable elements [16], which possibly form a hierarchy, a graph, etc. Configuring topologies in an integrated (rather than hybrid) way has been identified as an actual research challenge in SPLE [16], in particular as most configuration modeling approaches do not support (theoretically infinite configurations through) topologies at all. However, topological configuration needs also a domain-specific form of (graphical) tool support as noted in [16].

C4 Flexible automated instantiation: One particular goal of configuring a QualiMaster infrastructure (more precisely the QualiMaster platform) is to ease the specification of data analysis pipelines by domain users and, in particular, the adoption to other application settings and domains. In order to be executed, a configuration must be turned into software artifacts such as a Storm implementation[2]. Some of the benefits of SPLE are achieved by turning the configuration automatically into software artifacts, thus, avoiding tedious and error-prone manual customization activities. Following these experiences, we perform in

[2] A laid out distributed Storm processing algorithm is also called a topology. We will always use the term "Storm topology" for this form of implementation and "topology" for connected configuration structures.

QualiMaster an automated instantiation of the platform based on a given configuration. This supports Platform Administrators in concentrating on their expertise rather than realizing adaptive 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 topology configurations. Although improved concepts for flexible automated instantiation are designed in WP4, the actual QualiMaster infrastructure instantiation process is developed and realized in WP5 and, thus, discussed in D5.2.

C5 Flexible adaptation specification: Akin to a fixed instantiation process, a fixed realization of the adaptation is an obstacle for practical adoption. Thus, we aim at a flexible configuration and specification of the adaptive behavior of the QualiMaster platform for a certain application setting. In particular, the specification approach shall support the Adaptation Manager to adjust the adaptation behavior as needed. The research for such approaches is one particular topic on the roadmap for software engineering for adaptive software systems [27]. Moreover, in QualiMaster we aim at providing different forms of support for different user groups ranging from "knobs" on the user interface to details settings in the tool support for the (expert) Adaptation Manager.

C6 Combination of data and technical quality: In close collaboration with WP2 (cf. D2.1), WP4 aims at a unified treatment of different quality parameters ranging from technical and resource properties over generic data quality parameters up to user-defined domain-specific quality parameters. In contrast to related work (as we discuss in Section 4.1), we aim at characterizing, analyzing and aggregating these properties for adaptive data analysis pipelines consisting of complex user-defined analysis algorithms (rather than a fixed set of generic, but parameterizable operations as usual in literature).

C7 Adaptation for real-time big data analysis: QualiMaster aims at the analysis of real-time big data streams. While this is already a challenge for the design of the data analysis algorithms in WP2 and WP3, it is also a challenge for adaptation. In particular reactive and proactive adaptation aim at coping with changes in the environment in short time frames. While determining the adaptation decision shall be fast, it must not necessarily happen in real time. In contrast, the enactments of the adaptation decisions on the QualiMaster platform shall as far as possible not taint the real-time (low-latency) properties of the data streams.

These challenges motivate the approach that we take for configuration and adaptation. An overview of our approach is depicted in Figure 1. The Configuration Meta Model defines the configuration space for QualiMaster infrastructures in terms of configurable elements and constraints. Constraints restrict the values that can be assigned to the configurable elements. Please note that constraints can link several configurable elements, for example selecting the financial data source of SPRING implies that successor algorithm can process the output of that data source. The Configuration defines the settings for a specific application domain and a given infrastructure. If we refer to both, the definition of the configuration space and at least one specific configuration we use the term "Configuration (Meta) Model", as done for example in Figure 1. The Adaptation Model specifies the behavior of the (reactive and proactive) adaptation at runtime based on the Configuration Meta Model and the event / command types of the platform (an implicit complementing "architecture model"). The Adaptation Model can, for example, be formulated in terms of a domain-independent and a domain-specific part. Furthermore, the reflective adaptation may indicate changes to the Configuration Meta Model, the actual Configuration as well as the Adaptation Model.

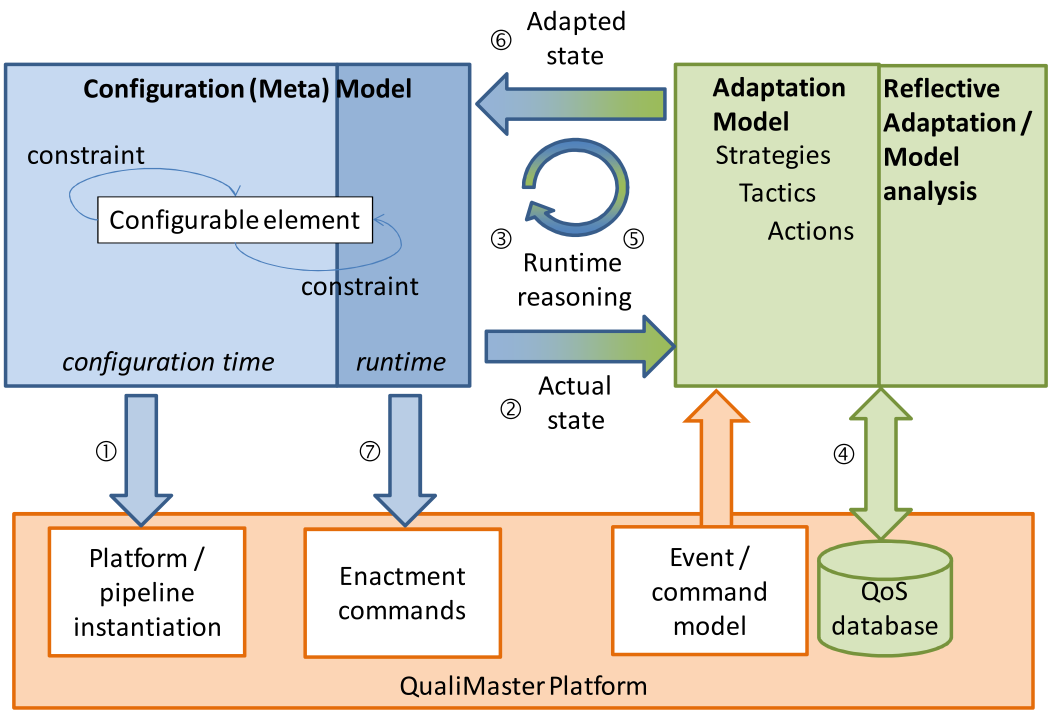

Figure 1: Configuration and adaptation approach overview.

Most of the configurable elements must be determined before startup of the QualiMaster platform, in particular the resource pool, the algorithms, the families and the processing pipelines. The instantiation of the QualiMaster platform denoted as ① in Figure 1 is driven by the configurable elements that are known prior to runtime. As part of the platform instantiation, (software) artifacts are created, modified, deleted, packaged, or deployed. As discussed in Challenge C1, some configurable elements reflect runtime information, such as actual measurements for quality parameters. This is indicated in Figure 1 by the graphical separation of the box representing the Configuration Meta Model. In particular, these runtime configurable elements and the constraints defined on them determine the adaptation space, actually a subspace of the space of valid configurations.

At startup time or runtime of the QualiMaster platform, the actual state of the platform ②, is represented in terms of the runtime configurable elements, e.g., the selected algorithms or measured quality parameters. This state is still subject to the respective constraints (e.g., SLAs). Violations of these constraints detected by monitoring and runtime reasoning ③ will cause a specific form of adaptation trigger, and, thus, an execution of the adaptation model. Examples for further adaptation triggers are regular schedules or user triggers issued by the QualiMaster applications. The adaptation model specifies the adaptive behavior based on (runtime) configurable elements. This may involve the QoS database ④, which stores quality characteristics, enaction impact statistics, adaptation execution logs and, possibly, a runtime model of the running infrastructure for proactive adaptation. As part of its execution, the adaptation model changes the actual configuration to reflect the desired state after adaptation. The desired state must comply with the constraints of the Configuration Meta Model as analyzed by the runtime reasoning ⑤. In case of a valid (runtime) model, the state described in the runtime configuration ⑥ is finally enacted ⑦ and leads to a modification of the QualiMaster platform at runtime. Reflective adaptation may lead to changes of the Configuration or the Adaptation Model.

4 Related work

In this section, we discuss existing work and approaches related to the quality-aware configuration and adaptation required in QualiMaster. Thereby, we follow the main topics of this deliverable reflected in the overall structure of this deliverable, i.e., quality parameters, configuration and adaptation. Thus, we start in Section 4.1 with a discussion of quality parameters in software systems and, in particular, in data stream analysis systems as well as an overview on (aggregating) quality analysis approaches in that field. In Section 4.2, we provide an overview on related work in the configuration of software systems. Finally, in Section 4.3, we discuss related work on adaptation, in particular on adaptive software systems and runtime adaptation in data stream analysis systems.

4.1 Quality Parameters and Quality Analysis

In the following paragraphs we review quality parameters that have been proposed in standards for software in general, and, in particular, data stream processing systems. It is important to note that we aim at quality parameters that describe the processing at runtime, i.e., for the actual data processing rather than the development or product quality of the data processing infrastructure. As a result, this section provides the basic terminology for discussing the quality properties and characteristics to be considered in QualiMaster. These properties and characteristics will be discussed in Section 6.

We structure our discussion into *software (product) quality* as a basis for the terminology (Section 4.1.1) and quality parameters in data stream processing (Section 4.1.2). Typically, the quality parameters referring to technical or data quality focus on the quality at a certain point during data processing, e.g., at a data source, an operator or a data sink. For overall optimization of the data processing and, in particular, for later cross-pipeline execution, it is important to derive the (aggregated) quality parameters for a complex data stream analysis consisting of multiple processing steps, i.e., in QualiMaster the aggregated quality of a data processing (sub-)pipeline. We discuss approaches to determine the aggregated quality in Section 4.1.3.

4.1.1 Software Quality

The basic terminology for software (product) quality is defined in standards, such as ISO/IEC 25010 [80][3] and the recent successor of ISO/IEC 9126 [79]. ISO/IEC 25010, defines eight characteristics of software (product) quality, namely functional suitability, reliability, performance efficiency, operability, security, compatibility, maintainability, and transferability. Each characteristic consists of further sub-characteristics. Note that, most of these characteristics focus on software development and product quality rather than on runtime quality (of the data processing). Thus, for QualiMaster the following characteristics and sub-characteristics are relevant:

- Functional suitability, in particular **accuracy**, i.e., the degree to which the software provides the right or specified results for the required degree of precision.
- Reliability in all defined sub-characteristics, namely
    - **Availability**, i.e., the degree to which software is operational and available when required for usage.
    - **Fault tolerance**, i.e., the degree to which the software can maintain a specified level of performance in case of software faults or of infringement of its specified interface.
    - **Recoverability**, i.e., the degree to which the software product can re-establish a specified level of performance and recover the data directly affected in the case of a failure.
- Performance efficiency, namely

[3] See also http://iso25000.com/index.php/en/iso-25000-standards/iso-25010

- **Time-behavior**, i.e., the degree to which the software product provides appropriate response times, processing times and throughput rates when performing its function under stated conditions.
- **Resource utilization**, i.e., the degree to which the software uses appropriate amounts and types of resources when performing its function under stated conditions.

As agreed in the DoW (Section B1.1.1), security / data protection is not in the focus of QualiMaster. Functional suitability, reliability and performance efficiency will be considered as a basis for the quality taxonomy in Section 6. Furthermore, the consortium aims at complying with the non-runtime characteristics of ISO/IEC 25010, namely operability, compatibility, maintainability, and transferability as part of the tool, infrastructure and application development activities.

In addition to taxonomies, generic (even tailorable) quality description approaches are known from the representation of quality parameters in software architectures (e.g., OMG MARTE [109]) or from a measurement-perspective in terms of the OMG Software Metrics Metamodel [110]. However, regarding specific parameters, we do not analyze software engineering approaches here, but discuss an overview on quality properties in data stream processing in the next section.

4.1.2 Data Stream Processing Quality

We now discuss typical measures of quality properties from the data analysis perspective and in particular from the perspective of data stream processing. In general, several classifications and taxonomies for data quality are published, including the Total Data Quality Management (TDQM) classification [133, 143], the Redman classification [116], the Data Warehouse Quality (DWQ) classification [83], the classification of Naumann [106], the classification of Liu & Chi [98], the Accessibility-Interpretability-Relevance-Credibility (AIRC) classification in [19] or the data stream mining classification in [55]. A more comprehensive overview on a basic set of parameters, namely accuracy, completeness, consistency and timeliness, and their definitions can be found in [11]. However, the terminology used in these publications is often quite diverse. Thus, this section contributes to a common terminology for quality dimensions and properties in QualiMaster. First, we discuss the concepts of these classifications and taxonomies. Then, we review the quality parameters we found during an (informal) literature survey on data stream processing approaches in terms of the ISO/IEC 25010 and show also weaknesses of the new ISO/IEC standard for the field of data (stream) processing.

Below, we categorize the quality properties from the classifications discussed above in terms of the ISO/IEC 25010 standard. We also indicate the data stream processing approaches we identified during our survey that actually target these quality parameters. For completeness, we will provide a mapping to all ISO/IEC 25010 characteristics although only functional suitability, reliability and performance efficiency are relevant to the runtime adaptation in QualiMaster.

- *Functional suitability.* This characteristic is generally addressed by the related software implementation functionality dimension in DQW [83], the methodological quality of [55] as well as partly by the collection / application dimensions in [98]. In particular, the sub-characteristic accuracy is frequently targeted by data stream processing approaches (also listed in the TDQM, the DQW classification as well as [19, 98, 106, 116]). However, additional and partially more specific data quality sub-characteristics as identified in the classifications above are not defined by the ISO/IEC 25010, probably as the characteristics in the standard terminology of ISO/IEC does not apply to all kinds of software systems. We discuss these specific quality parameters based on the classifications and the individual approaches below.
- *Reliability.* This characteristic is (partly) addressed by the related software implementation quality aspect as well as by data usage accessibility in the DQW classification [83], the intellectual dimension in [106] or partly by the Collection dimension in [98]. For example, the TelegraphCQ system in [26] provides a reliability-based service “knob” for data flows.
- *Performance efficiency.* This corresponds to the software efficiency aspect of the software implementation dimension in DQW [83], to the timeliness and amount of data properties as

part of contextual quality in TDWM, the technical dimension in [106], the application and organization dimensions in [98], the relevance-timeliness dimension in [19] or to the reaction time aspect of the temporal quality in [55]. Please note that in these classifications, timeliness is frequently defined in terms of the age of the data or the information instability (volatility) (see also [11]) rather than processing time or real-time aspects.

  - Several approaches actually target the *time-behavior* sub-characteristics of ISO/IEC 25010, for example in terms of timeliness (processing time or punctuality) [59, 87, 93, 103, 145], item latency [25, 122], response time [122] or throughput [1, 2, 25] or bandwidth [87] (akin to the QoS dimension in [106] defined in terms of transmission rates).
  - Some approaches also consider *resource utilization* in terms of memory usage [25, 87, 122] or (battery) power [87]. It is interesting to note that the classification of Liu and Chi [98] explicitly considers storage efficiency in terms of utilized space.

- *Operability* (the degree to which the software product can be understood or learned). This ISO/IEC 25010 characteristic corresponds to the accessibility category and the representational category in TDQM [143] as well as the usability aspect of the software implementation quality characteristic and the data usage quality (except for availability and security) in DQM [83].
- *Security* (the protection of system items from accidental or malicious access, use, modification, destruction or disclosure). This ISO/IEC 25010 characteristic corresponds to the access security in TDQM [143], the security aspect of the data usage quality in DQM [83] and partly to the application dimension in [98]. Please note that we mention this dimension here only for completeness, because security and privacy are not in the scope of QualiMaster as agreed in the DoW (Section B1.1.1).
- *Compatibility* (the ability to exchange information or to share the same hardware/software environment) is targeted by the data interpretability aspect (in particular for the communication with legacy systems) of the data quality dimension of DQM [83], the interpretability dimension in [19], the interpretability aspect of the content dimension in [106] and partial aspects of the presentation dimension in [98].
- *Maintainability* (the degree to which the software can be modified in order to maintain it). This ISO/IEC 25010 characteristic corresponds to the maintainability aspect of the software implementation quality dimension in DQW [83].
- *Transferability* (the degree to which the software product can be transferred among environments). This ISO/IEC 25010 characteristic corresponds to the portability aspect of the software implementation quality dimension in DQW [83], but also to the format portability aspect of [116].

As discussed for the functional suitability characteristic, several quality parameters that are in particular relevant for data stream processing are not explicitly mentioned in the ISO/IEC 25010 standard. Actually, they fit however into the “accuracy” dimension of ISO/IEC 25010 as this aims at the degree to which the software provides the right or specified results within the needed degree of precision [80]. We provide now an overview on these quality parameters we identified either as part of the taxonomies or as part of reviewing further research approaches. Several of the properties are (at least to some degree) application specific as stated in [59].

- *Accuracy* refers to the integrity of the data, i.e., the extent to which data is correct, reliable and certified free of error [143]. Further synonyms listed in [143] are accurate, correct, flawless or precise. In DQW [83] and also in [98], accuracy of the data is only related to the data entry process at the sources. In [19], accuracy is one aspect of credibility (see also below), while in [106] accuracy is an aspect of the content dimension. Confidence is one possible measurement for reliability [59]. Some approaches specifically targeting accuracy are [25, 59, 93, 103].

- *Objectivity* is the extent to which data is unbiased (unprejudiced) and impartial [143]. Objectivity is also listed as part of the classifications in [98, 106].
- *Believability* is the extent to which data is accepted or regarded as true, real and credible in [106, 143] (called *credibility* in DQW [83], or trustworthiness in [98]). Please note that in [19], creditability is an own dimension consisting of accuracy, completeness and consistency. Some approaches targeting confidence are [93, 103].
- *Reputation* is the extent to which data is trusted or highly regarded in terms of their source or content [143]. Reputation is also listed in the classification in [106].
- *Relevancy* is the degree to which data is applicable or helpful for the task at hand [143]. Relevancy is also listed in [98, 106].
- *Value-added* is the extent to which data is beneficial and provides advantage from its use [143]. This parameter is also listed in [106].
- *Completeness* is the extent to which data is sufficient breadth, depth and scope for the task at hand [143] and also listed in [19, 98, 106, 116]. More technically, this is frequently measured as the number of delivered vs. expected items [83, 59]. This quality property is considered by several approaches, in particular in terms of item drops [1, 2, 59, 93, 103, 145] or miss ratios [89, 145].
- *Consistency* is the degree to which a value adheres to defined (application or task-specific) constraints, in particular other data [59, 103, 116]. This parameter is also listed in [19, 98, 106]
- *Volume* to be processed, e.g., in terms of processed items [59, 89, 93, 103, 122].
- *Volatility* of the data stream, i.e., the frequency of change, either of the data [11, 19, 103, 116] or of the data stream, considered as smoothness / burstiness of the input data in [25] or as signal frequency in [122]. Some definitions such as [19] also relate volatility to the age of the data.
- *Item characteristics* such as value range, distribution and size [87].
- *Temporal quality*, e.g., in terms of applied time range or time granularity [55].

Several approaches on quality parameters in data stream processing discussed above focus exclusively on monitoring and adaptation based on technical or resource quality aspects as summarized in Table 1. More recent approaches target combinations of technical / resource quality and generic data quality aspects such as item drop or throughput as also indicated in Table 1. Recently, the integration of user- or application-defined data quality was identified as a challenge [58, 86, 87, 122], which requires capabilities to flexibly define such quality parameters.

In QualiMaster, we also aim at considering a reasonable combination of relevant quality parameters ranging from technical / resource properties over generic stream properties up to application-defined data quality parameters, in particular for the financial (use case) domain. In contrast to existing systems discussed above such as Aurora [2], TelegraphCq [26], Borealis [1], Rtstream [145], QStream [122], MavStream [25], PIPES [93], CluStream [87], QualiMaster aims at complex user-defined data stream analysis algorithms rather than a fixed set of (parameterizable) stream analysis operator such as aggregate, join, filter or sample (challenge C6). Actually, the work in [123] based on IBM System S and SPADE supports user-defined stream processing operators, but is limited to a certain programming DSL (preventing the integration of reconfigurable hardware). In QualiMaster, the combination of quality properties and user-specified algorithms contributes to Challenge C1 on unifying configuration and adaptation as quality parameters and their taxonomy must be defined, attached to the relevant configurable elements, and characterized by constraints during configuration time in order to become active at runtime. In addition, deciding on the set of relevant quality properties is also affected by Challenge C2, as one must consider that hardware-based execution typically implies constant resource consumption (determined as part of the compilation and layout process for reconfigurable hardware). Moreover, additional

	Aurora [2]	TelegraphCq [26]	Borealis [1]	Rtstream [145]	Qstream [122]	MavStream [25]	PIPES [93]	CluStream [87]	[59]	[89]
Reliability		x								
Time-behavior	p	p	p	t	r,l,v	p,	t,v	t,b	t,v	x,v
Resource utilization					x	x		x		
Accuracy						x	x		x	
Believability							x			
Completeness	x		x	x			x		x	x
Consistency									x	
Volatility						x				
Item characteristics								x		

Table 1: Frequently addressed quality dimensions (x = supported, for time behaviour, t = processing time, k = latency, v = volume, p = throughput, b = bandwidth)

specific quality parameters must be considered such as the switching time as an aspect of time behavior (as uploading a new algorithm, i.e., reprogramming the reconfigurable hardware, may take a few seconds and even an activated algorithm may need some settling time to produce valid results), additional network usage for integrating the reprogrammable cluster or the availability and optimal use of (free) DFE boards at runtime.

4.1.3 Data Quality Analysis

In QualiMaster, data quality analysis focuses on two distinct fields, namely the analysis of quality characteristics for individual algorithms, e.g., to provide quality input for adaptation, as well as the analysis of aggregated characteristics for entire pipelines. We discuss both topics in this section, while we focus in particular on the analysis of aggregated characteristics.

Analyzing and profiling individual data analysis algorithms for certain quality properties is a common practice in data engineering, social web analysis and software performance engineering, as for example described in [89, 46, 73]. In particular, for analyzing technical quality characteristics, several tools are available such as ArcheOpterix [4, 142], LibReDE [131] or PerfExplorer [74]. We aim at evaluating such tools for supporting the development of (distributed) data analysis algorithms in WP2 and for supporting the runtime decision making process during adaptation.

The analysis of aggregated quality parameters composed from algorithms as a data analysis pipeline is still a challenge. We focus on the data stream analysis case in the remainder of this section. Basically, we distinguish existing approaches into formal, formalized as well as approximate approaches. We will discuss examples for these approaches below.

In formal approaches, the quality parameters of the individual data analysis algorithms are modeled manually and then composed, either in terms of symbolic calculation or through aggregations. For example, an overall queuing network analysis of the MavStream system based on fixed operators is discussed in [25]. In this case, the result is, after extensive mathematical work, a set of formulae characterizing individual quality properties (based on distribution assumptions, simplifications, etc.). A more pragmatic (formalized) way is based on the estimation of quality parameters per data processing operator / algorithm and to derive aggregated quality parameters using aggregation formulae as done in [92, 93]. Instead of using aggregation functions, the authors in [89] model the stream processing system based on Petri nets (MEDAL approach) and derive quality parameters from the system model. However, all these formalized approaches rely on the assumption that basic quality parameters can be described or at least approximated in terms of (complete) mathematical functions. Basically, such formalizations are required for all data involved processing operators / algorithms and all quality parameters, again relying on assumptions or approximations. Other approaches are even more pragmatic, rely on measuring the quality characteristics (prior to runtime or at runtime) such as [73] and perform an approximate (quantized) aggregation for the entire data stream analysis system. This is done, for example, for throughput by [137, 138, 144, 152] and for error propagation by [82]. In order to predict the quality parameters of data stream analysis systems, some approaches such as [82] rely on simulation. Although approximate approaches can be applied to arbitrary data analysis algorithms (they become more precise the more information is available about the algorithms [12]), they may require (offline) profiles of the algorithms at hand during the startup phase and may struggle in case of deployment to a different hardware settings. On solution to the hardware dependency problem is parametric information as described for parametric performance specification in [95]. Further, approximate approaches may require updates of their profiles at runtime (as e.g., done in [73]) in order to provide the runtime adaptation with actual information on the quality characteristics.

As QualiMaster aims at the composition of arbitrary (user-defined) analysis algorithms, formal or formalized approaches are not applicable without putting a burden on the user or the algorithm engineer. Thus, the work in QualiMaster will rely on offline and online sampling of quality properties as well as approximate aggregation of quality properties for data analysis pipelines (Challenge C6).

4.2 Configuration

Software Product Line Engineering (SPLE) [31, 96, 111] is a successful industry-relevant approach to mass customization, configuration, tailoring, and reuse of software. Several examples are known in which SPLE helped to significantly reduce the time-to-market as well as costs while increasing the quality of the product. For example, as recently reported, the AEGIS product line by Lockheed Martin and the U.S. Department of Defense helped saving 80 million dollars over 3 years [65]. Please note that focus here exclusively on Software Product Line Engineering as agreed in the DoW. However, it is interesting to note other approaches to software configuration exist such as software factories [64], knowledge-based configuration [134], or (plain) model-based engineering [21].

In contrast to single system development, a Software Product Line (SPL) develops a family of similar, but differently configured systems. One of the fundamental concepts to manage such families is **variability modeling**, i.e., explicitly defining the differences between the individual systems (called configurable elements or variabilities) as well as the interdependencies among them. Coping with the complexity of such configurations, e.g., several thousands of decisions with interdependencies are reported in literature [15], is one of the key topics in SPLE. Deriving a specific member from a family includes the specification of its configuration, the validation of the configuration, and, finally, turning the configuration into artifacts, e.g., modifying, deleting or generating configuration files or source code as well as compiling, packaging and possibly deploying the derived product.

Over the years, several approaches to the modeling of variabilities and configurations of a product line have been developed. Currently, the most frequently used family of approaches in industrial practice is **feature modeling** [15], which was initially introduced in [88]. Basically, the configurable element in a feature model is called a feature, frequently characterized as a user distinguishable functionality of a system [30]. A feature model consists of a feature tree, i.e., if a parent feature is selected, its sub-tree is enabled for more detailed configuration (similar de-selection). Features may be mandatory, optional or alternative and can be part of multiple selections. Typically, inclusion and exclusion constraints can be stated to further restrict the configuration space. Further approaches include decision modeling [38, 121, 129], orthogonal variability modeling [111] and several others. Today, many variations of these basic approaches exist, for example feature models with cardinalities on multiple selections as a form of implicit constraint [91] as well as approaches supporting rather complex forms of configurable elements or constraints up to first-order logic, as, e.g., identified for textual variability modeling languages in [45, 48]. Recently, a consortium submitted the Common Variability Language (CVL) [66] to the Object Management Group (OMG) as a proposal for a standard. For some of these approaches the semantic equivalence of their modeling concepts has been shown, e.g., for free feature diagrams in [124] and for basic decision modeling and feature modeling in [49].

Due to the tree characteristics of feature models, many tool realizations focus on a graphical representation of variability models, either as a tree structure such as a file system explorer or a graph. Decision models are frequently rendered as tables, while orthogonal variability models are also typically displayed as a graph, frequently linked to UML [107] diagrams. In contrast to "traditional" variability modeling languages, recently several **textual variability modeling** languages have been invented such as CVM/VSL [117], TVL [29], Clafer [7] or IVML [34, 139]. A recent systematic analysis of textual variability modeling languages is described in [45, 48]. Many textual variability modeling languages, in particular the most recent ones, provide concepts that enable complex configurations, in particular Non-Boolean variability, complex constraint languages, evolution, ecosystem support or even some form of references among configurable elements (with quite diverse semantics).

Topologies (in the sense of connected structures such as graphs according to [16]) can be used to represent the physical and logical structure of alarm systems [16], embedded systems [16], warehouse layouts [32] or the topology of a data processing pipeline in QualiMaster. Although the need for topologies is not new, only few approaches target the modeling of such interconnected structures and many of the variability modeling approaches mentioned above (Clafer [7] and CVL [66] were considered in [16]) do not provide (comprehensive) support for the **modeling of topologies** [16, 44]. Recently, the integrated variability modeling of topologies has been identified as a research challenge in SPLE [16]. One approach to realize topological configurations is to keep the topology in another type of model and to combine variability modeling such as feature modeling with external configuration formats in a **heterogeneous variability modeling** approach, as, e.g., proposed in [50]. However, the modeling of such configurations is not integrated, difficult to manage as different types of models and tools must be combined and the abilities to specify constraints across the integrated models are typically limited. Berger et al. [16] rely on the UML [107] for variability modeling of topologies as none of the considered variability modeling approaches provided sufficient support. However, the authors aim at limiting valid connections in topologies through specific types using multiple inheritances among different containment hierarchies. As known from object-oriented modeling, such an approach leads to an inflation of the inheritance hierarchy. Another approach to topologies in configuration is rather simple and relies on typed references among configurable elements. References among configurable elements are known for a longer time, e.g., in terms of feature references [39], but typically these references are not typed, i.e., they can connect arbitrary configurable elements, and, if used in feature trees, do not support arbitrary (graph) topologies. References among configurable elements can (depending on their actual semantics [45, 48]) enable **integrated modeling of topologies** as also explained in [44]. Depending on the actual constraint language, further restrictions of the structure of a topology can be stated in terms of constraints. In contrast to many other approaches, IVML [34, 139] provides these capabilities as support for topology configurations was one of the industrial requirements in the FP7 project INDENICA [34]. In QualiMaster, this integrated topological variability modeling enables the configuration of the infrastructure along with its data processing pipelines in an integrated manner.

As described above, a variability model describes the configuration opportunities and a configuration defines the actual values for a certain setting. If the configuration complies with all constraints of the variability model, it can be used for **instantiation**, i.e., for turning the values into artifacts such as source code. Please note in staged approaches, such as [39], a configuration must not be complete in order to drive a (partial) instantiation. However, there is currently little conceptual support for automating the instantiation of a product line. Actually, many research approaches rely on so-called instantiators, i.e., manually programmed plug-ins that perform the instantiation of one or multiple artifact types. Industrial tools, such as pure::variants [17], provide artifact-specific instantiation languages (as well as integrations to generic model transformation languages). Recently, further artifact specific languages have been proposed such as TDL for use cases [150]. In contrast, one generic approach to variability instantiation is the Variability Instantiation Language (VIL) [36, 42, 140], which was also developed in the INDENICA project. In particular, VIL is integrated with IVML and enables the instantiation of topologies as this is required in QualiMaster.

In summary, there is a broad range of variability and configuration modeling languages, approaches and (research) tools for software product lines available. To our very best knowledge there is only one available integrated approach that supports the actual configuration requirements of QualiMaster, provides flexibility for future extensions and supports the realization of the configuration challenges discussed in Section 3. Thus, we will use IVML as variability modeling language (supports challenge C1 through so-called attributes, Challenge C2 by combining different support for hardware and software configuration and Challenge C3 through references and complex constraints) and VIL as variability instantiation language (supports Challenge C2 for generating the software / hardware integration, Challenge C3 through generation capabilities for references and Challenge C4 through flexible language-based definition of the instantiation). Both languages, IVML and VIL are realized and integrated in the Open Source toolset EASy-Producer [42]. However, EASy-Producer is designed as a tool for software (product line) engineers rather than for domain users such as data stream engineers or Infrastructure Users (D1.2). Thus, as initially described in D5.1, we develop in the QualiMaster project a domain-specific configuration frontend that supports the needs of domain experts in configuring the QualiMaster infrastructure and its data analysis pipelines, the latter in graphical and interactive way.

4.3 Adaptation

In this section, we provide an overview on related work in terms of resource-aware adaptation and characterization of software resources, specification of adaptive behavior, proactive / evolutionary adaptation as well as adaptation in data stream processing.

To enable resource-aware adaptation in a software system, a characterization of the actual resource consumption of the system or, preferably, its individual parts (such as components) is needed. In software design, these qualities (also called non-functional properties) are typically modeled as additional information attached to the parts, e.g., in terms of (actual) consumption and related constraints as done in WSLA [100], CQML [118], SLang [128], utility functions [10, 130], average measurements for certain operation (load) levels [148], interval resource usage functions [99], probability distributions [81, 95, 142] or a framework of actual measures, sampled resource usage functions, statistical characterizations and resource usage contracts in OMG MARTE [109]. While most of these approaches consider resource consumption as aggregated values, other approaches on simulating quality properties of software architectures consider abstractions from the execution environment. Having such abstractions at hands, resource profiles become deployable on other infrastructures as, e.g., the CPU cycle time becomes a parameter of the profile. For example, PALLADIO [12] relies on resource demand specifications, which are turned into resource uses based on the actual (specification of the) execution environment and parametric (component) constraints. While some approaches consider the replacement of resources such as [72], others try to select the best runtime configuration out of a given set of configurations such as [6] or to solve resource problems by runtime selection of partially pre-instantiated artifacts, as, e.g., done in [119]. Although we also aim at instantiating artifacts for optimization or to take alternative pipelines through instantiation at runtime into consideration, we primarily aim at changing system properties at runtime, thus, unifying typical techniques from adaptive software systems with (dynamic) software product line approaches. Further, some work considers not only technical software quality, but combines this with measures of the functional quality, such as data quality [58, 86] as already discussed in Section 4.1.2.

Most of the approaches to self-adaptive systems rely on rules or utility functions for the specification of the adaptation behavior [113]. On the one side, policies or rules, sometimes also called event-condition-action rules, are popular approaches to define adaptation behavior (e.g. in [41, 78, 151]) and may even be connected with other adaptation components such as monitoring as, e.g., recently done in MONINA [35, 78]. In order to be understandable and maintainable, some approaches represent rules in a hierarchical form and even introduce dynamic rule selection, e.g., taxonomies in [112], the strategy-tactics-action-based language Stitch [28] (designed for system administrators) or its successor S/T/A [73]. In contrast, utility functions can target individual quality properties or represent a combination of multiple properties, e.g., in terms of the compound utility discussed in [63] and enable a more continuous form of adaptation. On the other end, there are formal approaches to the specification of adaptation such as FORMS [147] or ActivForms [77]. While the authors of [77] focus on the verification of the adaptation components and realize the interpretation of an adaptation language, they also point out the issues of a formal approach, in particular the required expert knowledge on formal methods.

Allowing the developer or Infrastructure User to specify the adaptive behavior may lead to a significant specification effort and enable a rather open adaptation space. To avoid inefficient adaptation mechanisms, the adaptation space is frequently restricted to a meaningful subspace. For example, in component-based systems (the QualiMaster infrastructure with its individual algorithms can be considered as a component-based system) replacing a component may require a certain safe state [146] that can only be reached by certain sequences of passivating, modifying connectors, removing and re-enabling component instances. While some approaches aim at specifying these architectural rules, such as properties and constraints [53], reconfiguration protocols [20] or rules for hierarchical reconfiguration of nested components [69], others aim at deriving such sequences automatically, e.g., as evolutionary adaptation paths in [114, 153]. However, as a single component instance may be used in various different contexts as noted in [13], architectural rules may still not limit the adaptation space in an effective way. Here, some approaches rely on auxiliary models to characterize the adaptation space, such as profile guided composition models (based on context attributes taken from a variability-decision space of a DSPL) [3], quality attributes in variability models [84] or the co-specification of context [119] or hardware [120] variability.

Several classifications of adaptation approaches are described in literature, such as [5, 24, 27], but they are rather diverse regarding the proposed classification dimensions. Among these dimensions, two are of particular relevance to QualiMaster, namely whether an approach is anticipating (i.e., it is able to consider upcoming changes and plan for them, also called proactive adaptation) and whether the adaptive system supports (self-)evolution over its lifetime. Typically, anticipating adaptation relies on predicting the future context, i.e., how quality properties will behave in the short-term future as, e.g., done for memory consumption in [126, 127], workloads in [70, 97] or QoS properties in [73]. In contrast, some approaches aim at anticipating adaptivity through self-evolution of the adaptation model. For example, the Keep Alive Model (KAMI) approach is based on discrete time Markov chains and achieves self-evolution through a Bayesian estimator in [62] or parameter updates in [51]. Further, KAMI may consider multiple alternative execution flows of the adaptive system [61] and work on several kinds of models such as continuous time Markov chains [52] or Bayesian decision models [14], the latter combining probability and utility for decision making. Other kinds of evolution aim at characterizing the drift of quality properties over time in terms of linguistic values [141]. One specific topic in using self-evolving adaptation models is their initialization. Examples are initialization from other models (e.g., goal models in case of the Bayesian decision models in [14]), offline training [3, 97], online learning [3, 51, 52, 62] or online testing [81].

Adaptation in data stream processing typically focuses on (fixed) stream processing operators discussed in Section 4.1.3. On prominent mean to adapt stream processing is load shedding (sometimes also called data admission), i.e., dropping out data items to avoid infrastructure overload [1, 25, 136, 145]. Further means are switching to approximate calculation such as sampling or interpolation [93, 122], changing the analysis window size [93], the available compute resources per stream operator [87], in that work in terms of parameter adaptation, to re-route items [1, 122] or to change among fixed query plans at runtime [122]. Thereby, work such as [1] focuses on flexible tradeoffs. More recent work also focuses on learning about adaptations in (big) data analysis infrastructures or even composing analysis flows. For example, in [18] the authors use machine learning and planning to optimize the selection of the most appropriate data analysis flow among a fixed set of flows composed from typical data stream operators. For reflective adaptation[4], we will take this and similar approaches into account as a basis for the related QualiMaster concepts. In contrast, [115] aims at automatically composing such flows specified in terms of flow patterns such as input/output tagging, inherited substitutable algorithms, algorithm parameters and structural variations such as optional components. Also other work such as [61] discussed above aims at adapting processing flows.

Although the work of [115] has some conceptual overlaps with QualiMaster, we aim at algorithms given in a general-purpose programming language or, even in terms of hardware-based execution, a combination of reactive, proactive and reflective adaptation, a wider range of tradeoffs, as well as flexibility through integrating configuration with adaptation (Challenge C1, Challenge C5). Moreover, QualiMaster targets the analysis of real-time streams, i.e., determining the next feasible adaptation must be fast, but enacting the adaptation must not taint the real-time properties of the stream or provide techniques to cope even with complex adaptations (Challenge C7).

[4] Please consider that the related task T4.4 is scheduled to start in month 18 of the project.

5 Requirements

In this Section, we detail the specific requirements for the configuration and adaptation of the QualiMaster infrastructure, in particular its instantiation for the financial domain to perform systemic and real-time risk analysis. Therefore, we analyzed the requirements from the QualiMaster deliverables D1.2 and D5.1 as well as the DoW for the QualiMaster project. In this section, we summarize our findings in terms of requirements that are relevant to the modeling of configuration and adaptation and relate the individual requirements to the respective source document if applicable. In particular, we do not discuss requirements that describe the administrative operation of an infrastructure, e.g., starting or stopping the infrastructure or the pipelines, as such operations are in the responsibility of WP5.

Following the overall structure of the deliverable, we structure the discussion in this section into requirements for modeling the configuration (at configuration time) in Section 5.1 and the adaptation (at startup and runtime) in Section 5.2. However, this distinction cannot easily be made for all requirements, as some requirements actually support the adaptation, but detail configuration requirements. We will list such requirements along with the configuration requirements in Section 5.1. In the remainder of this deliverable, we will use the unique requirements identifiers introduced in this section to indicate their realization in terms of concepts, configurable elements, and model structures, etc.

5.1 Requirements for the QualiMaster Configuration Model

In this section, we discuss the requirements that primarily refer to the QualiMaster configuration (meta) model. Please note that the configuration of visualization algorithms has been removed from the infrastructure configuration use cases in D1.2.

REQ-C-1 Application Users do not need to know details about the configuration of the QualiMaster platform, in particular not how the platform can be or is configured as stated in D1.2 (Section 2.1). In particular, Application Users must not modify pipelines as stated in D1.2 (Section 5).

REQ-C-2 The Platform Administrator must be able to administrate the physical computing resources for software-based execution as stated in D1.2 (UC-PA10 and UC-PA11). Example properties of the physical computing platforms are their resources, the network identification, etc.

REQ-C-3 The Platform Administrator must be able to administrate reconfigurable hardware units (such as MAXELER Data Flow Engines) as stated in D1.2 (UC-PA11 and UC-PA12). Based on personal discussions with experts from MAXELER, the available number of Central Processing Units for communication and control processes, the available number of DFEs for processing hardware-based algorithms and a network address / port is sufficient to characterize the MPC-C series cluster (see also D5.1, Section 6.2), which is used in the QualiMaster project.

REQ-C-4 The QualiMaster infrastructure must support multiple and various real-life data sources (at least financial domains and web data as stated in D1.2 (Section 6.1.1), possible with each source having a different type of data as stated in D1.2 (REQ-DS1). The Platform Administrator must be able to configure the data sources as stated in D1.2 (UC-PA7).

REQ-C-5 Akin to the data sources, the Platform Administrator must be able to configure the data sinks as stated in D1.2 (UC-PA7), i.e., the physical points where the communication with the application happens. The QualiMaster consortium decided to also record the output types at data sinks in order to define the data supported handover to the applications.

REQ-C-6 The Platform Administrator must be able to administrate the storage of data as stated in D1.2 (Section 2.2) and D5.1 (Section 5.3.1). This includes the recording of historical data at the configured data sources or sinks as stated in D1.2 (REQ-DS3) and detailed in D5.1.

REQ-C-7 The Platform Administrator must be able to administrate software- as well as hardware-based algorithms providing metadata such as the containing algorithm family, the name of the algorithm, the inputs and outputs, functional parameters as well as the actual implementation in terms of an algorithm package as stated in D1.2 (UC-PA3, UC-PA4, UC-PA5, UC-PA6). Based on typed data sources, in particular for the financial domain, the QualiMaster consortium agreed to describe the inputs and outputs in terms of stream data items with fields and types. Thereby, "unstructured" streams mentioned in UC-PA3 and UC-PA7 shall be represented in terms of a generic, serializable type leaving the option that further experience allows to make the notion of "unstructured" streams more specific.

REQ-C-8 The Platform Administrator must be able to administer the pool of algorithm families. In D1.2 (UC-PA3, UC-PA4, UC-PA5, UC-PA6), algorithm families are treated implicitly as part of administrating the algorithms (REQ-C-6). Consequently, families know their contained algorithms and exhibit the common "interface", i.e., the accepted input / output as well as the functional parameters.

REQ-C-9 The Pipeline Designer must be able to define the structure of data processing pipelines in terms of pipeline elements. Pipeline elements include data sources, data sinks, data processing elements (referring to algorithm families) and the data flow among the pipeline elements as stated in D1.2 (UC-PD1 and UC-PD2). Further, specific pipeline elements for storing intermediary processing data in the data management layer must be provided as stated in D5.1 (Section 4.3.1). According to D1.2 (REQ-DS2), the QualiMaster infrastructure must support filtering of data according to criteria, i.e., within a data processing pipeline either as a data processing element referring to a filter family or as a generic parameterizable data stream operation. The set of generic data stream operations to be supported is currently in discussion in the consortium. The integration of historical data into a data processing pipeline as stated in D1.2 (REQ-DS3) may happen through specific data sources or directly through the implementation of individual data processing algorithms. If possible, the configuration of pipelines shall be supported through a graphical editor, as the QualiMaster consortium believes that this simplifies the configuration tasks and supports the future adoption of the QualiMaster solutions.

REQ-C-10 The Pipeline Designer must be able to execute a static analysis of the feasibility of the designed pipelines as stated in D1.2 (UC-PD1 and UC-PD2), e.g., by type checking as mentioned in D1.2 (Section 7.1.1), i.e., type conformance checking of the data analysis pipelines based on the input and output field types of the data sources and algorithm families, respectively.

REQ-C-11 The Adaptation Manager must be able to define and specify means for measuring quality parameters. This includes quality characteristics of the different data processing elements, methods for measuring different data processing elements and methods for estimating the end-to-end quality of pipelines as stated in D1.2 (Section 2.2, as well as UC-AM1 and UC-AM2).

REQ-C-12 The Adaptation Manager must be able to define prediction mechanisms for quality parameters for proactive adaptation as stated in D1.2 (Section 2.2 as well as UC-AM4).

REQ-C-13 The Adaptation Manager must be able to define a set of "adaptation rules" on the pipeline level for reactive and proactive adaptation as stated in D1.2 (Section 2.2 as well as UC-AM3 and UC-AM4).

REQ-C-14 The Adaptation Manager must be able to specify the high-level settings of the adaptation mechanism and even change them at runtime as stated in D1.2 (UC-AM6 in D1.2). This extends REQ-C-13, which can be considered as a configuration of the adaptation behavior on a higher level of detail also requiring more knowledge about adaptation in QualiMaster.

REQ-C-15 The QualiMaster infrastructure must provide support for the execution and optimization multiple pipelines, i.e., cross-pipeline execution and optimization as stated in the DoW.

5.2 Requirements for the QualiMaster Adaptation Model

In this section, we detail the requirements that are specific for the QualiMaster Adaptation Model. In contrast to the requirements discussed in Section 5.1, which mostly involved human actions, we discuss here requirements that aim at the autonomous adaptation behavior of the QualiMaster infrastructure.

REQ-A-1 An Application User must be able to pass triggers to the platform through the application GUI as stated in D1.2 (Section 2.1). This is a specific way of causing adaptation from the viewpoint of an Application User as stated in D1.2 (Section 5). However, the Application User may not be aware that he/she is triggering an adaptation as for example a “knob” may provide options to cause such triggers. However, the QualiMaster infrastructure may refuse such triggers as they are considered of lower priority than internal adaptation causes, e.g., caused by monitoring SLAs.

REQ-A-2 The QualiMaster infrastructure must support dynamic means to exploit mechanisms to maximize computational performance as stated in D1.2 (REQ-Q4).

REQ-A-3 The QualiMaster infrastructure must provide means to autonomously measure and optimize its resource usage as stated in D1.2 (REQ-Q5).

REQ-A-4 The QualiMaster infrastructure must support timeliness in the processing of real-time data streams in order to produce up-to-date (real-time) analysis results as stated in D1.2 (REQ-Q1).

REQ-A-5 The QualiMaster infrastructure must support means to customize the coverage of the data sources to produce a comprehensive market analysis as stated in D1.2 (REQ-Q2). This may happen automatically, but also be realized through application triggers (cf. REQ-A-3).

REQ-A-6 The QualiMaster infrastructure must support means to specify the accuracy of the performed calculation as stated in D1.2 (REQ-Q3). Changing the accuracy may happen automatically, but also based on application triggers (cf. REQ-A3).

REQ-A-7 The QualiMaster infrastructure must provide means to characterize and react on the content quality of the data processing (cf. REQ-Q6 in D1.2).

REQ-A-8 The QualiMaster infrastructure must measure and cope with scalability aspects in Big Data architectures (cf. REQ-Q7 in D1.2)

REQ-A-9 The Platform Administrator must be able to monitor the pipeline operations as stated in D1.2 (UC-PA15).

REQ-A-10 The Adaptation Manager must be able to monitor and analyze the execution of adaptation rules. This shall be supported by the reflective adaptation indicating when configuration or “adaptation rules” shall be adjusted to further optimize the operation of the pipelines as stated in D1.2 (Sections 2.2 as well as UC-AM5).

REQ-A-11 The QualiMaster pipeline must support switching between hardware- and software-based stream processing at runtime as stated in D5.1 (Section 2.2). Further, switching algorithms in a family at runtime must be supported as the basic adaptation concept in QualiMaster as envisioned in the DoW.

REQ-A-12 The QualiMaster infrastructure must support (dynamic) pipeline analysis to determine the potential impact of an adaptation on a whole pipeline, in particular for quality parameters requested by REQ-A-2 - REQ-A-8.

We will use the unique requirements identifiers in the remaining sections in order to indicate the realization of the individual requirements along with the concepts of the configuration and adaptation model.

6 QualiMaster Quality Dimensions and Quality Analysis

In this section, we discuss the quality dimensions that the QualiMaster consortium identified for the work in the QualiMaster project. Akin to Section 4.1, the focus of this section is not on development, process or product quality. The focus of this section is on the runtime quality, i.e., parameter that can only be measured at runtime in order to determine whether the infrastructure is working under good conditions, to figure out whether deviations occurred and, in case of deviations, what adaptations at runtime shall be enacted. The quality parameters are of particular importance for WP4 to design and realize the adaptation. These quality parameters are also important for the other work packages:

- WP2 uses the quality dimensions to characterize individual algorithms. In this regard, but also to avoid overlaps among the deliverables, we present the current state on all QualiMaster quality dimensions here, but detail the algorithm specific parts for WP2 in D2.1.
- WP3 relies on the quality dimensions to contribute and consider specific parameters for hardware-based algorithm execution.
- WP5 uses the quality dimensions to design and realize the QualiMaster infrastructure enabling monitoring, processing and adaptation enactment according to the quality parameters.
- WP6 utilizes the quality dimensions to communicate the analysis results to the Application User, and support (a limited set of) mechanisms to issue actual requests to the infrastructure ("user triggers").

Thus, determining the quality parameters and their taxonomy, the quality dimensions, is a foundational step in QualiMaster, which also contributes to a common quality terminology.

In fact, determining the quality parameters is an incremental process. On the one side, the quality parameters drive and foster the quality-related work in the work packages as indicated above. On the other side, research on algorithms, pipelines and adaptation may indicate that defined quality parameters do not lead to expected results, cannot be measured as intended, need to be refined, discarded extended. Moreover, insights from the evaluation as well as increased collaboration with stakeholders and practitioners due to results from the research and development work in the project may lead to new quality parameters. Thus, the QualiMaster consortium created an initial set of quality parameters based on a consortium internal quality survey. Furthermore, the consortium considers the initial set of quality parameters discussed in this section as subject to revision during the course of the project and to be reported in subsequent deliverables. For example, we consider reliability as an aspect of operating the infrastructure and defer the consideration of related quality parameters to future work. As a consequence, work and approaches on algorithms, configuration, and adaptation must be flexible so that quality parameters can be added, redefined, adjusted or removed if needed.

In Section 6.1, we report on the quality survey we conducted to determine the initial set of quality parameters for the QualiMaster project. In Section 6.2, we summarize the results of the quality survey in terms of a quality taxonomy for the project. Finally, in Section 6.3 we outline our plans for quality analysis, in particular on pipeline level.

6.1 Quality Survey

As discussed in Section 4.1, the terminology used for describing quality parameters is quite diverse. A similar diversity regarding runtime quality is inherently present in the QualiMaster consortium as it consists of researchers and practitioners from various fields, including financial software development, data stream processing, hardware-based processing and acceleration, social web analysis, and software engineering. After initial discussions on quality parameters, in particular within WP4, we decided to conduct an (informal) quality survey in the QualiMaster consortium. The aim of the survey was to collect the knowledge and opinion of all partners, and to obtain their personal feedback on the importance of individual quality parameters for the work in the QualiMaster project. In contrast to a personal discussion, the rationale of conducting a survey was to give each participant the opportunity and the time to think carefully about proposed parameters (drawn from the DoW, D1.2 and the WP4 discussions), missing parameters as well as the relevance for the own field and the overall project.

The core partners of WP4 structured the survey into a brief introduction and seven groups of quality parameters according to the individual parts of a pipeline (data source, analysis algorithms, data sink, data analysis pipeline) as well as infrastructure healthiness (as a basis for successful execution), user triggers (REQ-A-1) and infrastructure level adaptation settings (REQ-C-14). Within each group of quality parameters, we briefly explained the role of the group in the infrastructure as well as the individual quality parameters identified (in the terminology used by the consortium so far). Then, we asked for the relevance of the individual parameters in terms of a 5-level Likert scale[5] ranging from very relevant to very irrelevant. We also provided options to enter an optional comment for each quality parameter and to suggest further parameters. Before asking for participation, we presented the quality survey to the staff members of SUH as an informal pre-study to resolve issues before conducting the survey. The time frame for processing the survey document was two weeks. The survey was conducted in autumn of 2014.

After receiving feedback from all partners, we summarized the answers using an Excel sheet, turned the scales into integers ranging from 4 (very relevant) to 0 (very irrelevant) and rated the quality parameters / triggers / settings in each group according to the average relevancy. As an initial indicator, we selected the top 50% rated parameters for inclusion into the initial quality taxonomy. Please note that the top 50% rated parameters may lead to more than half of the parameters as the average relevancy of several parameters may be rated equally. Further, we paid special attention to those parameters indicated by the DoW, D1.2 and those suggested by the participants. However, due to the limited size of the base population (and six received responses), we do not perform a statistical evaluation of the results here.

Now, we briefly mention the selected quality parameters for the individual groups:

- **Data source quality:** Item frequency, data volume and data integrity were ranked on the top positions. The comments on item frequency and data volume indicated that a strong variation is expected for both parameters at runtime, that the actual relevance may differ according to the investment horizon of the user and that both parameters may indicate a potential overload situation already at the source side of a pipeline. Also, data completeness was indicated as a relevant parameter. In addition, burstiness / volatility of the data were mentioned as a potential quality parameter. A further technical requirement was indicated in terms of the data update frequency characterizing the need for reloading the data schema such as daily changing market players.
- **Algorithm quality:** Data confidence, item latency, correctness and expected switch time (between algorithms, in particular software and hardware) were ranked on the top positions. However, in order to estimate and reduce the computational cost, we consider also memory consumption as a quality parameter in QualiMaster (due to REQ-A-3). Based on the algorithms considered in WP2, future research will indicate whether the startup time required to obtain stable results (after a dynamic switch of algorithms) needs to be taken into account in the future.
- **Overall pipeline quality:** Execution latency (all participants ranked this as very relevant), aggregated correctness, aggregated confidence, and throughput were ranked on the top positions. Further, the overall CPU time consumed by a pipeline was suggested, but, as software-based and hardware-based execution work according to different principles, this needs deeper investigation if actual work indicates a relevancy of this parameter.
- **Data sink quality:** Item frequency, data volume and data confidence were rated as the most relevant parameters, in particular to characterize the expected output quality on the Application User side. Please consider that further latency or disruption may happen during the internet transmission of results from the QualiMaster infrastructure to the Application on user side. Volatility of considered market players was suggested as a domain-specific

[5] http://en.wikipedia.org/wiki/Likert_scale

parameter during the discussions before the quality survey, but currently (indicated by the comments) not ranked as very relevant.

- **Infrastructure quality / healthiness:** The number of (available) physical devices, i.e., resources, the actually used devices, and the actual connection bandwidth were ranked as important infrastructure quality parameters. Further, the comments indicate that the relative speedup of individual algorithms, in particular software-based vs. hardware-based algorithms as well as statistical measures of the parameters shall be collected in order to characterize algorithms as well as the overall infrastructure.
- **User triggers:** The number of stocks to be processed (impacting the data source volume), the market depth and the target confidence were ranked as the most important (mostly domain-specific) triggers. A rather close rating indicates the importance of the number of markets as well as filter terms for the social network streams. Further, the selection of market player pairs was suggested as a new trigger, as a stakeholder does not necessarily need all combinations of market players rather than the most important ones.
- **Infrastructure settings** for the adaptation, i.e., the high-level “knobs” an Adaptation Manager may influence (REQ-C-14): Highly ranked settings are the relative importance of the quality dimensions as well as the tradeoffs to be considered for cross-pipeline adaptation.

For creating the (initial) quality taxonomy, we classify the quality parameters mentioned above according to the discussion in Section 4.1. In contrast, user triggers and infrastructure settings represent additional requirements (as reflected in D1.2). In particular, the infrastructure settings are considered in the Configuration Meta Model and the user triggers in the (realization of the) adaptation model.

6.2 Quality Taxonomy

Based on the results of the quality survey described in Section 6.1 and the related work on runtime quality in software systems in general (Section 4.1.1) and, in particular, data stream analysis (Section 4.1.2), we introduce in this section the (initial) quality taxonomy for QualiMaster. As discussed above, this taxonomy may be subject to change based on future research results as well as feedback from the evaluations and the stakeholders.

Figure 2 depicts the actual structure in terms of dimensions / sub-dimensions as well as the related quality parameters. As introduced in Section 4.1.1, we based the high-level structure on the ISO/IEC 25010 standard, added dimensions where needed and mapped the terms from the quality study to terms used in literature as discussed in Section 4.1.2. The high-level structure of our taxonomy consists of

- Performance efficiency in terms of time-behavior and resource utilization aligned with ISO/IEC 25010 aiming at aspects within the QualiMaster infrastructure.
- Functional suitability, in particular based on data/content-related quality parameters, also aligned with ISO/IEC 25010.
- Scalability as an addition, in particular with a focus on (external) Big Data quality aspects.

Below, we present and discuss the dimensions and parameters in the context of the QualiMaster project according to the structure of Figure 2. We indicate the intended measurements for the individual parameters in italics.

6.2.1 Performance Efficiency

QualiMaster focuses on processing of online data streams for real-time applications such as the risk analysis of financial markets. As discussed in Section 4.1.1, in the ISO/IEC 25010 standard [80] this dimension consists of *time behavior* and *resource utilization*.

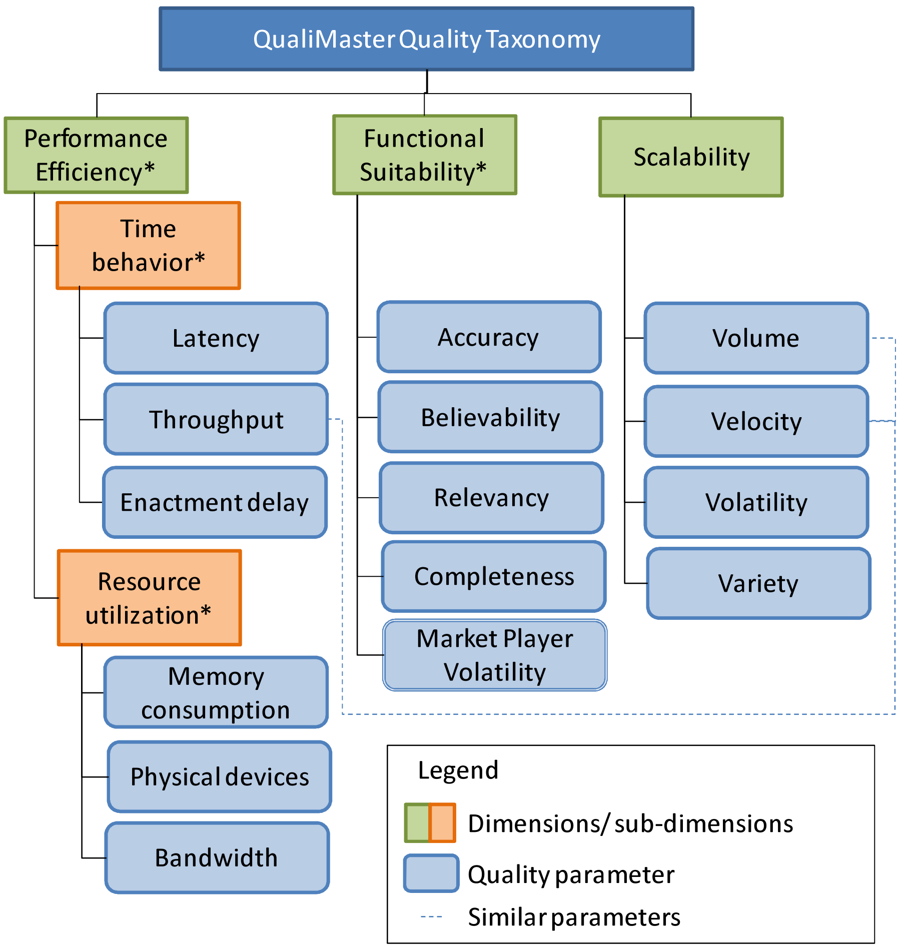

Figure 2: The QualiMaster quality taxonomy (* indicates dimensions based on [80])

6.2.1.1 Time Behavior

Time behavior, in particular in quality parameters expressing timeliness is an important quality aspect that has to be handled by QualiMaster (REQ-A-4). The focus is to be able to provide an analysis of the input streams at a very low time granularity, in the sub-seconds time range as usual in the financial application domain. Consider, for example, a financial risk analysis application that measures the correlation between a set of market players of a certain portfolio based on live data streams of the trading of their stocks in stock markets. The goal is to provide a time series showing the measurement of the correlation between these market players that is updated at a time granularity that is, ideally, as low as the time granularity in which the input data stream is arriving.

Based on the quality survey, we identified four quality parameters that address time behavior:

- **Latency:** The *difference in time granularity between the input stream and the output stream* is called the latency of data processing, i.e., the delay between the arrival of new data and the reflection of this data on the output data stream. Basically, latency is caused by the data processing, in particular the processing algorithms and network transmissions in a distributed cluster. Latency can vary from an algorithm to another (also within the same family) and its measurement depends on the type of the algorithm. Please note that aggregating algorithms may change the semantics of the aggregated latency for multiple algorithms as we discuss in Section 8.3. On this level, latency corresponds to response time [106] (milliseconds resolution or, in the extreme case, execution time in nanoseconds resolution for CPU / processor time). Please note that due to the design process, the latency of hardware-based algorithms is determined during compilation time and constant. On one hand, for stream processing algorithms, the latency is measured as the delay between receiving a stream input item and producing its corresponding output item. On the other hand, for batch processing algorithms, the latency is measured by the overall computation time of the batch processing job. The design and development of processing algorithms in WP2 will take this quality parameter into account in order to minimize the processing latency by design. Please note that from a pipeline point of view, latency encompasses the entire data processing including effects of adaptation (see enactment delay below), data transmission among distributed processing units as well as management overhead of the processing framework.

- **Throughput:** Is measured as *amount of items that is processed per unit of time*, i.e., items per second [1, 2, 25], or, in relation to the volume, as *bytes processed per unit of time*. Depending on the measurement, throughput is related to data volume or to latency, i.e., increasing volume at same processing rate decreases the throughput, while increased latency at constant volume decreases the throughput. Please note that we classified throughput under time behavior, as throughput was explicitly mentioned in this dimension in [80]. However, we also consider throughput as an important parameter to express the scalability of individual parts as well as the QualiMaster infrastructure in the Big Data context. Corresponding to latency, the aim of QualiMaster is to maximize the throughput of a given processing pipeline under varying conditions.
- **Enactment delay:** The enactment delay is the *time caused by the enactment of adaptive decisions* due to the adaptation of the QualiMaster infrastructure, e.g., by changing the actual algorithm of a family, to (prepare and) realize a switch to or from hardware (thus, called switch time in the quality survey) or to adjust the overall data processing. For a catalog of enactment patterns, please refer to Section 8.2. Actually, enactment delay contributes to the (overall pipeline) latency and may be considered as a specific form of latency. As the enactment delay is one of the measurement to express the quality of the adaptation (the reaction time of determining adaptive decisions is another measurement, but considered of lower importance on real-time processing as discussed for Challenge C7), we introduce the enactment delay as specific parameter. The enactment delay may have impact on the latency / timeliness of a running pipeline only when an adaptation actually takes place. The aim of research and development in WP4 is to avoid (in the best case) or minimize this form of latency (Challenge C7).

6.2.1.2 Resource Utilization

The resource utilization characterizes consumption of computational resources (such as memory and physical devices used) by the QualiMaster infrastructure, and, in particular, the processing pipelines and determines the computational cost of the processing. The goal is to reduce the consumptions of resources where possible, which may lead to an overall reduction of energy consumption (except for temporary resources that may be needed for adaptation, cf. Section 8.2). Resources can be characterized in terms of *physical* and *logical resources* [46], i.e., whether they correspond to hardware or to units of the operating system, as well as *intra-device* or *inter-device usage*. In particular, QualiMaster aims at the optimization of intra-device and inter-device of physical resources. Please note that due to the design process, the internal resource utilization of hardware-based algorithms fixed during compilation time, while the amount of CPUs and Data Flow Engines (DFEs) used for computation can vary dynamically at runtime. There is typically a trade-off between the processing speedup that is gained by using re-configurable hardware and their computational cost. This is in particular true, as the amount of available CPUs and DFEs for hardware-based execution is limited due to their price compared to general-processing resources.

Based on the quality survey, we consider the following resource utilization parameters:

- **Memory consumption** represents the *actual use of memory for a certain unit* of the QualiMaster infrastructure on various levels of aggregation, e.g., on infrastructure level (inter-device utilization), on pipeline level (inter-device utilization), algorithm level (intra-device utilization) or on device / machine level (intra-device utilization). Please note, that memory consumption is measured in memory allocation and unallocation, while memory consumption is a derived measure of the difference between allocation and unallocation [46]. Knowledge about the memory consumption (and their overuse) may indicate a reason for performance degradation. QualiMaster aims at balanced memory consumption on a minimized number of devices (computational cost), in order to cope with peak usages, provide resources for additional value-added computation and to allow multiple-pipelines to be executed optimally (to be addressed in the third period of the project).

- **Physical devices** form the computational basis for the Execution Systems in the QualiMaster infrastructure. Knowledge about the *number of actually used machines* for software-based execution as well as the CPUs and DFEs for hardware-based execution on both, infrastructure and pipeline level is important to understand the actual (inter-device) resource utilization. Conversely, knowledge about the *number of actually available machines*, i.e., CPUs and DFEs availability, is an indicator of the healthiness of the infrastructure and characterizes the opportunities of migrating algorithms between software and hardware-based processing, to execute further pipelines or value-added computation.
- **Bandwidth** refers to the *actually used amount of bytes transferred via network* among the devices used for software- as well as hardware-based execution (inter-device utilization). High utilization of network resources may cause performance and availability degradation within the heterogeneous compute cluster.

Currently, additional resource-related quality parameters such as energy consumption were not explicitly addressed in the quality survey. However, through balanced memory consumption we aim to achieve a good baseline in terms of energy consumption and consider energy consumption as a potential future extension of the quality taxonomy, in particular having the (quantitative) analysis of the adaptation enactment patterns in Section 8.2 at hands.

6.2.2 Functional Suitability

This dimension aims at the right or specified results for the required degree of precision [80]. As QualiMaster focuses on data (stream) processing, functional suitability consists of data-related parameters (REQ-A-6, REQ-A-7), in particular (to a certain degree) **domain-specific quality parameters**. In QualiMaster, functional suitability specifically addresses the quality of the outcome at the individual data processing algorithms (potentially considered as subsequent input) as well as the overall outcome of a pipeline at its sinks.

Due to the results of the quality survey, the functional suitability dimension of the QualiMaster quality taxonomy consists of:

- **Accuracy** according to [143], i.e., the integrity of the data and measures the extent to which data is correct, reliable and certified free of error. On the one side, this includes the accuracy / integrity of the source data. On the other side, we use this property to measure the accuracy of the data processing and analysis output. Both sides address REQ-A-6. The measurement of the data accuracy can vary from an algorithm to another, in particular across different research fields, depending on the concrete definition of the accuracy of the specific algorithms. In particular, we aim at two measurements, namely the
  - ***Confidence*** *or precision value of a prediction or a classification algorithm*
  - ***Error rate*** *of an approximation algorithm representing the correctness of the processing.*

  In particular, integrating the different notions of confidence among data stream processing and social web analysis, e.g., on pipeline level is a challenge (Challenge C6) as we discuss further in Section 6.3.
- **Believability** is the extent to which data is accepted or regarded as true, real and credible in [143]. In general, the importance and the potential impact of the content of social media streams varies a lot depending on the source. We measure the data creditability based on the *estimation of the creditability of the source or the authors of the content*. This can be achieved, for example, by manually or automatically evaluating the popularity of a specific source (e.g., blog) or by measuring the influence of users in social networks, and taking this measurement as a basis for filtering the user-generated content in these networks.
- **Relevancy** characterizes whether data is helpful for the task at hand [143]. This is in particular important for the (integration of) the social media analysis algorithms. One of the

challenges in processing and analyzing social media streams and web sources in general is finding and extracting relevant information with respect to the user query out of the large volume of unstructured or semi-structured and noisy data that comes from a variety of heterogeneous sources. By relevancy we refer to the *estimation of the relevance of the processed content* (e.g., a blog or a news item) to the user query (e.g., to a specific stock or market player). For example, measuring the similarity degree between the user query and the entities mentioned in an online post gives an estimation of the relevance of that post to the query, which in turn can be used to judge the reliability of the analysis that is done based on the content of this post.

- **Completeness:** As described earlier in Section 4.1.2, completeness refers to the extent to which data is sufficient in breadth, depth and scope for the task at hand [143]. In QualiMaster we use this property to measure the completeness of the input streams (REQ-A-5). This can be computed by *measuring the coverage of the sources* (e.g., in terms of stock markets, market players etc.), the sampling degree that is applied to the streams, and/or the recall of the applied algorithms (e.g., in classification algorithms). The goal is to maximize the completeness of the data as much as possible. However, this implies a trade-off with other quality dimensions such as timeliness and cost efficiency.
- **Market Player Volatility:** This quality parameter captures the domain-specific concept of variability in the financial domain. As explained in D1.2, we aim at measuring the volatility of markets, financial instruments and individual market players. As we expect a strong correlation between the systemic risk of a market player and its future volatility, we aim at using this quality parameter to judge the quality of the overall risk analysis independently of the calculations in the respective pipeline, and, if adequate, to adjust the processing dynamically to focus on those market players with an indication for high systemic risk.

6.2.3 Scalability

By scalability we refer to the capability of the QualiMaster infrastructure to process data at varying data volume, velocity, variety and volatility (burstiness). This mostly corresponds to the three dimensions of Big Data[6], namely volume, velocity and variety (REQ-A-7). The particular aim of this dimension is to characterize whether and how the QualiMaster infrastructure as well as its parts copes with the Big Data challenges.

According to the quality survey, we consider in this dimension the following:

- **Volume** of the data stream, i.e., the aggregated amount of data items (or bytes) that are given to the system for processing [60, 40]. However, as amount itself is just an aggregated number, we will consider here *volume per time unit* instead, i.e., the volume aspect of **throughput**. According to the quality survey, this is important for both, data sources and data sinks. This can also be measured for the volume handled by batch processing. Please note that we focus on the classical Big Data V's. In [40], also veracity (consistency, reliability and trustworthiness) as well as value (the added-value of data) were introduced. We handle veracity in terms of functional suitability and value by the algorithms / pipelines.
- **Velocity**, i.e., the speed of the data in terms of *data items per time unit* (called item frequency in the quality survey, e.g., number of stock ticks per second or tweets per second). Mostly, velocity is referred as a general aspect of frequency [60]. Thus, we consider velocity as the element aspect of **throughput** (Section 6.2.1). Velocity can also be considered as the externally observed throughput at the data sources and an externally observable parameter at the user side through the data sinks. This parameter was considered as most important quality parameter for both, data sources and data sinks.
- **Volatility** in the sense of technical burstiness, i.e., how data deviates over time. This was suggested as a measure in the quality survey. In particular, we aim at *deviation of volume and velocity at data sources*. Please note that the related parameter discussed in Section 4.1 mostly focused on the age of the data.

[6] http://en.wikipedia.org/wiki/Big_data

- **Variety** deals with the complexity of big data and information and semantic models behind these data, including the notion of structured, unstructured, semi-structured and mixed data (sources) [40]. For evaluation purposes, we aim at the *number / types of streams* (or data sources) that can be handled in parallel, such as the number of financial and social media streams. Variety can also be measured in terms of some domain-specific criteria such as, the number of financial markets or stocks that are analyzed simultaneously, or the depth of the market data streams.

In particular, the parameters of the scalability section can be used to characterize external visible quality of the data processing in terms of SLAs, e.g., minimum velocity at data sources (to indicate that negotiated quality cannot be achieved due to external problems) or velocity expected at data sinks by the users. In addition, we plan to consider also scale out measures for evaluation, i.e., whether and how the QualiMaster infrastructure can scale out to existing machines in times of changing environmental conditions. Further, statistical calculations on these parameters can indicate the minimum, average or maximum capability of the QualiMaster infrastructure on a reference installation for a certain application case, e.g., at the QualiMaster installation at the Technical University of Crete.

6.2.4 Summary

In this section, we briefly summarize the quality dimensions introduced above and their expected trade-offs in terms of Table 2. Please note that we consider availability (as also discussed as a dimension [80] in Section 4.1) as a capability of an infrastructure provider, who offers the computation capabilities to customers rather than a quality of the platform itself.

ID	Quality dimension	Aim	Quality parameters	Tradeoffs
Q1.1	Time Behavior	Real-time processing of streams and efficient processing of historical data	• Latency • Throughput • Enactment delay	Q1.2, Q3
Q1.2	Resource utilization	Minimize consumption of computational resources (and energy)	• Memory consumption • Physical devices • Bandwidth	Q1.1, Q2, Q3
Q2	Functional Suitability	Optimize the (domain-specific) quality of the processing results	• Accuracy • Believability • Relevance • Completeness • Market Player Volatility	Q1.1, Q1.2, Q3
Q3	Scalability	Maximize the Big Data capability of the infrastructure	• Volume • Velocity • Volatility • Variety	Q1.2, Q2

Table 2: Summary of quality dimensions and expected tradeoffs.

6.3 Pipeline Quality Analysis

While most quality parameters discussed in Section 6.2 can be measured in isolation, e.g., for a certain algorithm or the entire infrastructure, some parameters must be aggregated in order to reflect the quality of a pipeline and to adapt the platform in this respect (Challenge C6). Furthermore, having methods for estimating aggregated quality of a pipeline at hands, pipeline analysis can be performed, either as part of the pipeline design (REQ-C-9), the adaptation (REQ-A-12) or for reflective adaptation. In this section, we summarize the quality parameters on pipeline level (taken from the quality survey and represented in terms of the quality taxonomy) and indicate our plans for performing quality analysis in the remainder of the project. Please note that the quality parameters on algorithm level are taken up and discussed in D2.1.

Based on the quality survey (Section 6.1) and the QualiMaster quality taxonomy (Section 6.2), we aim at the following quality parameters on pipeline level:

- **Data source:** Velocity, volume, volatility (burstiness), accuracy (integrity), and completeness.
- **Data processing pipeline:** Latency, throughput, and accuracy (in terms of correctness as well as confidence).
- **Data sink:** Velocity, volume, accuracy (confidence) and volatility of market players.

On the data processing pipeline level, the quality parameters must be aggregated from the measurements of the currently active algorithms, the data sources or data sinks, respectively. We plan to perform the aggregation for the individual quality parameters on data processing pipeline level as follows:

- **Latency:** Measurement of the execution time consumed per item, considering that individual algorithms such as correlation computation may aggregate items, i.e., they combine multiple input items into one output item. For determining the overall latency, we plan to consider an aggregation factor per algorithm as suggested in [25]. Further, we aim at considering algorithm profiles and the aggregation factors to predict the expected latency based on the source velocity.
- **Throughput:** Efficient monitoring, analyzing and predicting the throughput also on fixed data stream operations has be reported to be challenging [25, 137, 138, 144, 152]. Thus, we plan to rely on the approximate approach for estimating the throughput by Tatbul [137, 138] for estimating the throughput of a pipeline. The core idea there is twofold: a) Throughput is propagated backwards through the pipeline, starting at the sinks in order to figure out whether the requirements there (SLAs) can be met at the sources with their actual quality properties. b) Approximate computing in terms of quantized quality functions represented as numeric tables. Initially, the tables represent the quality requirements at the sinks. During the analysis, the tables are propagated backwards through the pipeline and updated at each analysis step. While an analysis of an existing pipeline can apply this approach, supporting the adaptation must take alternative algorithms into account without calculating all alternatives. For this purpose, we aim at combining the backward propagation approach with ideas from [61] for adapting over networks with alternative paths.
- **Accuracy:** For predicting the confidence, we plan to extend the approach by Tatbul mentioned above. For correctness in terms of errors, we plan to base our work on the forward propagation of errors through processing networks by Jaques-Silva et al. [82]. However, here the challenge is to find a combination between algorithms with proven quality guarantees and such with soft quality statements such as social web analysis algorithms in close collaboration with WP2.

Work on data pipeline analysis is scheduled for future work in the QualiMaster project due to the focus on realizing the priority pipeline in the first period and as than working versions of the algorithms, priority pipeline and the infrastructure are available and quantitative experiments also on initial realizations of the pipeline quality analysis can be performed.

7 QualiMaster Configuration Model

In this section, we discuss our approach to the quality-aware configuration of the QualiMaster infrastructure. As discussed in Section 4.2 as well as in D5.1, we use the EASy-Producer toolset [42] a basis for our work and, in particular, rely on its configuration modeling language IVML, the INDENICA Variability Modeling Language, and VIL, the Variability Instantiation Language, developed in previous research projects such as FP7 INDENICA[7]. In Section 7.1, we give a brief introduction into the concepts of IVML. In Section 7.2, we discuss the Configuration Meta Model for the QualiMaster infrastructure and the realization of the related requirements from Section 5.1. Thereby, we visualize the IVML concepts rather than presenting the complete IVML models. Please note that the whole Configuration Meta Model is given for illustration in Appendix A in terms of IVML. The Configuration Meta Model is then transformed into implementing software artifacts using the QualiMaster platform instantiation process. We describe this process in more detail in Deliverable D5.2 along with the current state of the architecture QualiMaster Infrastructure. Finally, in Section 7.3 we give a brief overview on the actual state of the tool support, i.e., EASy-Producer in general and the QualiMaster infrastructure configuration tool that is realized on top of EASy-Producer.

7.1 Configuration Modeling Overview

In this section, we provide a brief overview on the concepts of IVML, the language to define the Configuration Meta Model as well as specific configurations of the QualiMaster infrastructure. IVML is a textual approach to model a Configuration Meta Model based on decision modeling concepts (cf. Section 4.2 for other approaches).

In IVML, a *project* is the top-level language element that identifies the configuration space of a certain (software) project. In terms of a product line, such a project can represent reusable artifacts as well as (partially) instantiated artifacts of a product. Within a project, the relevant configurable elements are defined. In contrast to many other approaches [45, 48], configurable elements in IVML are expressed as in terms of **typed decision variables** (variables for short) inducing a (strongly) typed language. A decision variable represents a decision the user has to take while defining the configuration of a product, e.g., whether and how much reconfigurable hardware is present in a certain QualiMaster installation setting. The IVML type system consists of basic types (Boolean for alternative decisions, Integer, Real, String for Non-Boolean decisions), enumerations (for alternatives) compound types and container types (for multiple selections) over all the previously mentioned types. A compound type groups multiple configurable elements into a single named user-defined unit (similar to structs or records in programming languages). This allows combining semantically related decisions from which each element has to be configured individually. The project language element also serves as modularization unit, i.e., an IVML model can be composed from various projects through imports, which makes the declared types and decision variables available to the importing project. In QualiMaster, model composition helps us to organize the Configuration (Meta) Model akin to the architecture of the infrastructure, i.e., to increase the understandability of the model, but also to separate the model parts along the groups of Infrastructure Users defined in D1.2 and to support (physical) access protection.

In addition to the definition of own compounds, IVML allows the **derivation of new types** based on existing types in two ways, namely refinement and type derivation. The *refinement* of a compound makes all of its decision variables available to refined compounds, which may add further decision variables (similar to sub-classing in object-orientation). Compound refinement enables extensibility, an important capability for open world scenarios such as service-based systems or software ecosystems. In QualiMaster, this capability enables us, e.g., to define a basic type for reconfigurable hardware as well as the specific types used in the project, such as the MAXELER MPC-C series. If other types of reconfigurable hardware shall be used later in the project or after the end of the project, the Configuration Meta Model (and similarly the infrastructure derivation) can be extended accordingly. The *derivation of types* allows the introduction of new types based on the restriction of existing types (such as positive Integers as a restriction of Integers). A specific

[7] http://indenica.eu

form of type derivation is type aliasing, i.e., the use of an existing type via a different name without limiting expression, e.g., to distinguish quality parameters through their type.

One particular capability of IVML is to support **references** among configurable elements, similar to typed pointers in a programming language. References allow sharing a configurable element and, in particular, to model connected structures (we call them topologies). This enables us to model QualiMaster data processing pipelines. Thereby, we extensively use refined compounds, on the one side to limit the valid connections among pipeline elements and on the other side to allow future extensions.

Basically, a decision variable accepts any value of its type, including **null**, a value indicating that a variable is not configured. **Constraints** further restrict the possible values of a decision variable, thus narrowing the (valid) configuration space. For example, reconfigurable MAXELER hardware without Data Flow Engine (DFE) boards are not allowed in QualiMaster (we give more complex examples for algorithm and pipeline constraints in Section 7.2). IVML constraints are inspired by the OMG Object Constraint Language (OCL) [108], which supports first-order logic constraints (on ground instances) in combination with relational and arithmetic expressions. Therefore, the notation of IVML constraints as well as the semantics of constraints is rather similar to OCL. However, we did not take over all concepts from OCL, particularly not those specifying the application of constraints in the context of UML concepts. In contrast to OCL, IVML supports side effects by value assignments and value propagations to express effects on a product line configuration. Further, IVML provides more convenient type selection mechanisms, in particular dynamic dispatch of user-defined constraint functions, so that type-specific constraints can be modeled in an extensible manner. In addition, IVML allows defining constraints as decision variables. Initially, this was intended to disable individual constraints on purpose. In QualiMaster, we use this capability to naturally express user defined (collections of) constraints such as data sink SLAs and to distinguish between constraints defined by the QualiMaster consortium and user defined constraints. However, for specifying SLAs we expect that a limitation of the IVML constraint language to relational and arithmetic constraints will be required so that these constraints can efficiently be processed by the QualiMaster adaptation (see Section 8).

In IVML, projects as well as individual decision variables can be **attributed by further information**. In QualiMaster, we use this additional information to model the so called binding time, i.e., the point in the software lifecycle when the decision for a specific value must be made. This enables us to distinguish between configurable elements that must be specified for the instantiation of the QualiMaster platform, and configurable elements for which the value is known only at runtime, i.e., where the constraint evaluation and the effect of constraints is postponed into runtime under the control of the adaptation.

While decision variables are defined in the Configuration Meta Model, the actual value of the variable to be instantiated, e.g., the number of available DFE boards, is defined by a **configuration**. A configuration must comply with Configuration Meta Model in terms of structure (decision variables and types). Further, a **valid configuration** fulfills all constraints and can be used for the instantiation of the QualiMaster platform, i.e., invalid configurations shall not be instantiated and, thus, prevent erroneous execution before deployment. In contrast to many other related languages, in IVML a configuration can be expressed using the same concepts as for defining the Configuration Meta Model, in particular using value assignments. Further, IVML supports the freezing of decision variables in order to specify the configuration of values that shall be considered during instantiation. This is required to enable partial and staged configurations, e.g., to pre-configure the generic QualiMaster infrastructure generically for the financial domain. Defining a configuration is supported by several means, ranging from concepts to tool support. On the conceptual side, default values defined in the Configuration Meta Model allow a basic pre-configuration. On the tool side, the definition of the Configuration Meta Model as well as the configuration is supported by the EASy-Producer toolset [42]. Therefore, EASy-Producer provides specific editors, model validation through constraint evaluation and constraint propagation, i.e., the (conditional) derivation of actual values of decision variables from other decision variables. The latter two capabilities are realized by a specific reasoning mechanism for IVML models.

Please note that IVML offers even more modeling capabilities, which are currently not required in QualiMaster, such as conditional / versioned imports or variability interfaces to hide decision variables intended for multi-party product line settings such as software ecosystems.

In terms of syntax and concepts, IVML is inspired by related variability modeling languages such as CVL [66], CVM/VSL [117], TVL [29] or Clafer [7], but also by programming languages such as Java or C. As stated above, regarding constraints, IVML was mainly inspired by OCL [108]. Relying to some degree on existing language concepts was a conscious decision to provide practitioners with familiar concepts and to reduce the learning curve. Previous experience with industrial partners showed that IVML can quickly be learned and applied by computer experts after some basic introduction in product line engineering in general and IVML concepts in particular.

7.2 QualiMaster Configuration Meta Model

In this section, we describe the Configuration Meta Model for the QualiMaster infrastructure. The Meta Model is defined by the QualiMaster consortium and supports the configuration and instantiation of the QualiMaster platform for a certain physical and application setting. Usually, this model will not be changed by the user groups defined in D1.2, but it can be adapted by experienced users as part of an adoption of the QualiMaster approach.

In this section, we discuss the model mainly in terms of its design and concepts visualized in a UML-like graphical notation[8], in particular as a favor for the reader so that no deeper IVML knowledge is needed. Further, we discuss IVML fragments for the individual parts of the model to explain how the Configuration Meta Model is specified and the rationales behind the individual concepts. For the complete Configuration Meta Model and an example of a QualiMaster Configuration in IVML syntax, please refer to Appendix A.

In Section 7.2.1, we provide an overview on parts of the Configuration Meta Model and then discuss the details of the individual parts in individual subsections. Please note that the Configuration Meta Model reflects the current state of discussion and will probably be extended and refined in the future course of the project to reflect additional configuration knowledge. In particular, we aim at detailed knowledge on the Execution Systems to support the Infrastructure Users in setting up and bootstrapping the QualiMaster infrastructure in their specific setting.

7.2.1 Configuration Meta Model Overview

The Configuration Meta Model particularly aims at the *configuration of the real-time data stream processing* in the QualiMaster infrastructure, i.e., it specifies the

- Observables and quality parameters that can be monitored (REQ-C-11, REQ-C-12),
- Hardware pool for software- and hardware-based execution (REQ-C-2, REQ-C-3),
- Configuration of the data management layer (REQ-C-4, REQ-C-5, REQ-C-6),
- Data processing algorithms for real-time data stream processing (REQ-C-7),
- Algorithm families consisting of algorithms (REQ-C-8),
- Data processing pipelines (REQ-C-9, REQ-C-10), which are modeled as a data flow topology,
- High-level adaptation settings for the Adaptation Manager (REQ-C-14), and
- Pipelines selected for execution (REQ-C-15) as a global infrastructure setting.

[8] There are several similarities but also differences between UML and IVML. For example, the basic data types are rather similar, IVML compounds can be understood as UML classes without methods and multiple inheritance, IVML references can be seen as the only kind of association, the constraint language is similar, etc. Due to the different modeling scopes of IVML and UML, there are also differences such as that IVML allows global variables, there is no package merge in IVML, IVML supports conditional imports, IVML supports a more specialized form orthogonal information (attributes) and the constraint languages differ in particular capabilities and semantics, e.g., configuration constraints are not always free of side effects.

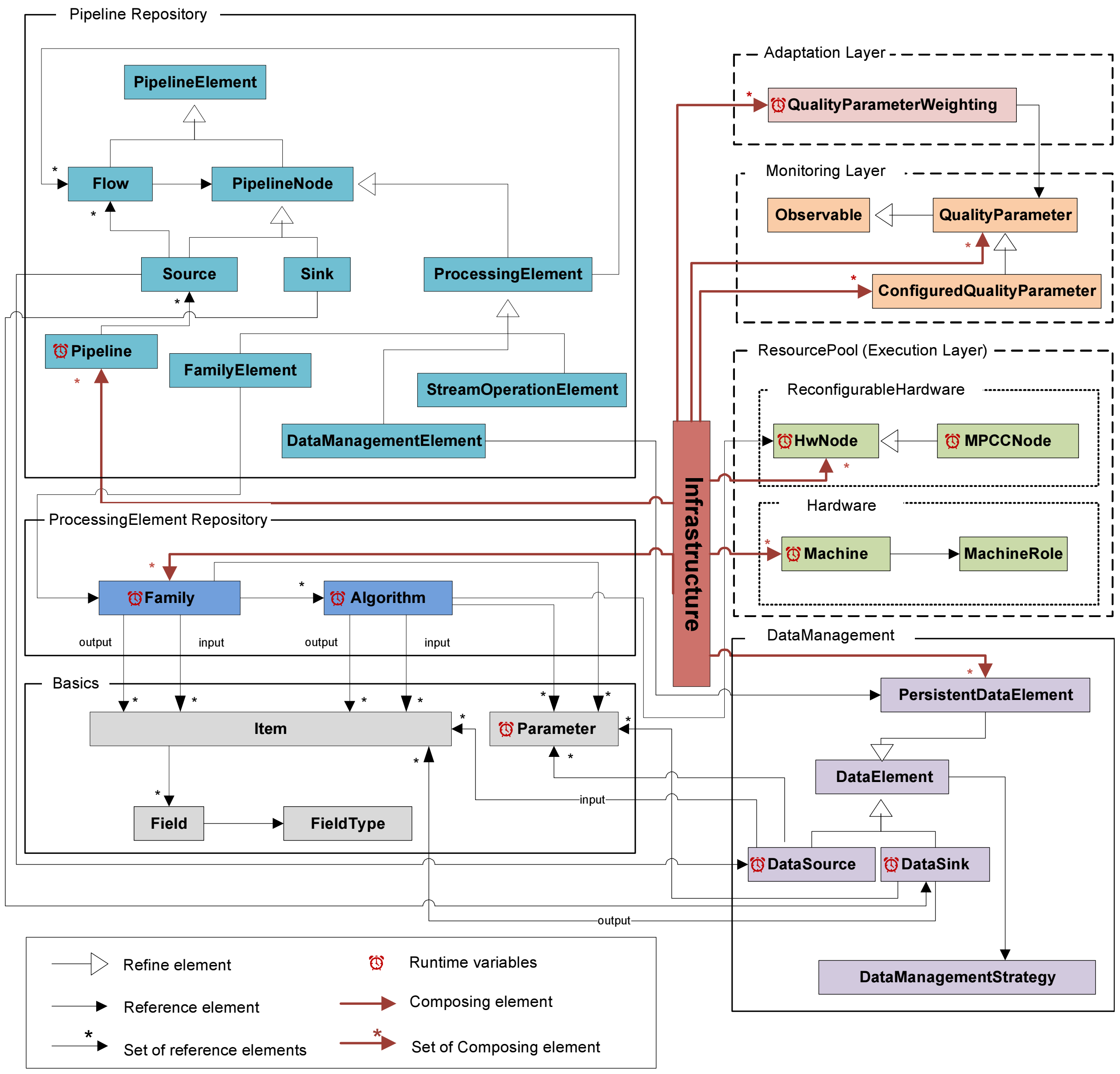

Figure 3: Overview of the QualiMaster Configuration Meta-Model.

The **design goal** of the Configuration Meta Model is to capture the information needed to statically check the feasibility of a platform before instantiation and to actually instantiate the platform for a specific application setting. In particular, this includes technical configuration knowledge that is gathered during the QualiMaster project. Further, wherever possible and adequate, we aim at an Execution System independent model, so that configuration information and the instantiation process can be transferred across (different versions) of Execution Systems.

The structure of the Configuration Meta Model is organized according to the QualiMaster infrastructure design as described in deliverable D5.1. In addition, the Configuration Meta Model consists of a `Basics` package, which declares common types such as the derived types `NonEmptyString` or `PositiveInteger` as well as compound types such as `Parameter` or `Item` for specifying data sources, data sinks, algorithms and algorithm families. Figure 3 provides an overview on the Configuration Meta Model and its configurable elements (without their individual details) mostly in terms of IVML compound types, refinement of compound types and typed references among them. In particular, the references among the compound types of the configurable elements in the `Pipeline Repository` part form the topology that allows modeling the structure of data processing pipelines in terms of QualiMaster concepts (REQ-C-9), in particular algorithm families and access to the data management layer.

Below, we will detail the parts of the Configuration Meta Model in individual subsections in the sequence given above. Thereby, each subsection will first give a short motivation, discuss the configurable elements and, finally, explain the constraints that apply for the respective configurable elements. Please note that the realization of the Configuration Meta Model is separated according to the structure shown in Figure 3 and finally linked together to form a common model using the model composition mechanism of IVML. The configuration is organized similarly and separated into individual physical folders (also the Configuration Meta Model is located an own folder) to support access protection according to the Infrastructure User roles described in D1.2.

7.2.2 Observables

The observables part of the QualiMaster Configuration Meta model details the information that can be observed in the QualiMaster infrastructure at runtime. In particular, observables are quality parameters according to the quality taxonomy discussed in Section 6.2. As the monitoring of further technical information may be required to support adaptation, operation and maintenance, we use the more general term observables for this model part. The remaining model parts rely on the definitions given in the observables part to declare runtime variables for the observables and quality parameters. Figure 4 illustrates the observables model part.

Basically, all observables can be expressed in terms of basic IVML types. However, we use a specific derived type for each observable / quality parameter to support the identification of the related runtime variables during (runtime) reasoning and adaptation (this is not shown in Figure 4). Further, we define a compound hierarchy acting as descriptors for observables, namely `Observable`, `QualityParameter` and `ConfiguredQualityParameter`. `Observable` aims at a base type for future extensions by technical observables. `QualityParameter` represents built-in quality parameters according to the quality taxonomy in Section 6.2. `ConfiguredQualityParameter` denotes user-configured quality parameters according to REQ-C-11 and REQ-C-12. Both types of descriptors for quality parameters are stored in individual sets (in the figures we depict this as a relation between the `Infrastructure` and the configured instances), while the built-in quality parameters are directly frozen in this model part. Derived quality parameters can be stated in terms of default values, either as value expression or through a specific user-defined function. Ultimately, user-defined quality parameters lead to a refinement of the predefined types of the Configuration Meta Model described below and require specific tool support as well as a code extension (cf. Section 7.2.5) of the QualiMaster infrastructure for monitoring (REQ-C-11) and prediction (REQ-C-12).

Basically, the constraints defined for the used derived data types apply, such as that some strings must not be empty. In addition, the configured quality parameters must have unique `type` names.

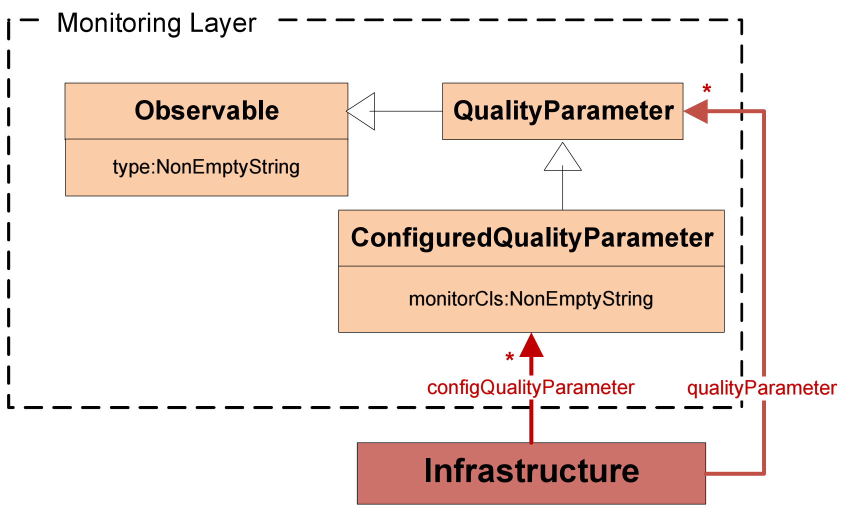

Figure 4: The observables in the Configuration Meta Model including quality parameters.

7.2.3 Resource Pool (Execution Layer)

The configuration of the QualiMaster infrastructure specifies the resource pool underlying the Execution Layer (see D5.1/5.2), i.e., the physical machines that can be used for executing pipelines and data analysis algorithms. The hardware level covers both, software-based and hardware-based execution as stated in REQ-C-2 and REQ-C-3. First, we discuss the configuration of software-based and then the configuration of hardware-based execution.

The aim of the configuration of the **software-based execution** is the specification of the compute nodes and servers that are actually available for executing QualiMaster pipelines. In the Configuration Meta Model, this is reflected in the configurable element `Machine` (see Figure 5), which describes a general-purpose machine. `Machine` is characterized in terms of its (network) name, the available resources (memory, processors, and processor frequency), the communication ports used within the cluster, and the machine role of the machine in the cluster installation. A set of default ports is used by Apache Storm in absence of a specific configuration. Currently, the following machine roles are foreseen: Manager and Worker. For distributed real-time processing on Apache Storm, the manager role corresponds to the Nimbus nodes and the worker role to the Storm Supervisor nodes. Please note that a sufficient number of communication ports must be configured so that the amount of workers required by the pipelines to be executed can be fulfilled, as otherwise Storm rejects the execution of pipelines. Thus, having the communication ports available at configuration time is important for REQ-C-10. All configured `Machine` instances are stored in a collection representing the part of the resource pool for software-based execution. Further, this model part defines the infrastructure healthiness quality parameters `bandwidth` (per machine), available machines and used machines according to Section 6.2.

Due to technical reasons (partly imposed by Apache Storm) the following constraints apply:

- At least one Worker must be assigned so that pipelines can be executed.
- At least one Manager must be assigned so that the cluster can be managed centrally.
- Manager machines must not have configured communication ports as this is handled dynamically by Nexus / Storm.
- Worker machines have either configured communication ports or receive the default ports of Apache Storm.
- Machines must have unique names, i.e., machines must be uniquely identifiable.

For illustration, Figure 6 depicts a fragment of the related IVML specification of the Configuration Meta Model (runtime variables are not shown) in terms of a language unit (`project`) declaring the `MachineRole` enumeration type and the `Machine` compound type as well as defining the constraints described above. The depicted project utilizes some types defined in the imported project `Basics` such as `NonEmptyString`, `PositiveInteger`, `MemorySize`, `Frequency` or `PortInteger`. In particular, the specialized types avoid repeated declaration of their limiting

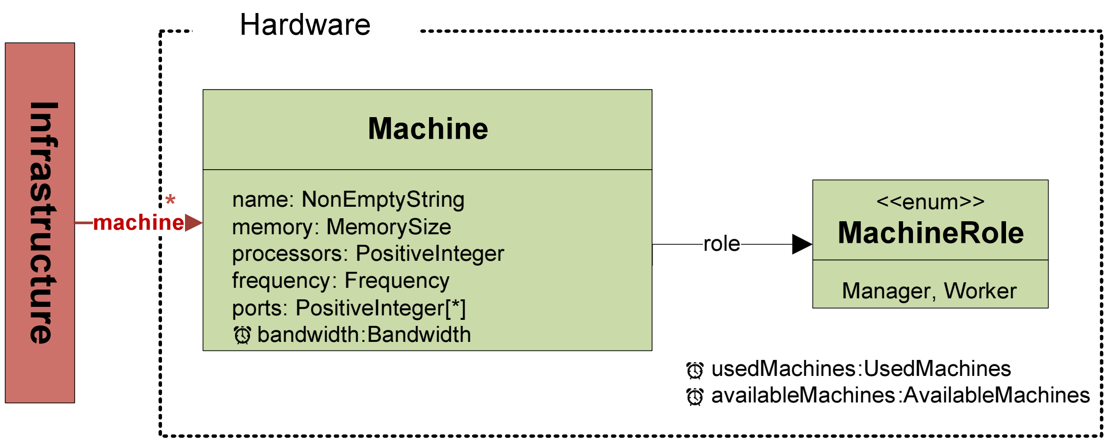

Figure 5: Part of the Configuration Meta Model for software-based execution.

constraints and can be used by the tool support to offer specialized editors, e.g., allowing ports to be specified in user-friendly ranges rather than individual numbers. The IVML fragment also attributes additional information to all model elements, here, the binding time of the individual decision variables. While the default variable binding happens during instantiation / compilation (the traditional SPLE binding time), some variables are marked for runtime binding, i.e., as runtime decision variables, such as `bandwidth` in `Machine` or `usedMachines` in the global scope. The machines for software-based execution that are available in the heterogeneous resource pool are configured in the `machines` collection.

Please note that constraints in Figure 6 are given in two different scopes, namely within the compound type and directly in the project. A constraint stated in a compound type *C* restricts the configuration of *C*, i.e., the constraint is applied to all instances of *C* by implicit all-quantification. In Figure 6, the two constraints in `Machine` target the (conditional) configuration of ports depending on the configured worker role. The second constraint may lead to a value propagation if the value of `ports` is not configured or to an equality check else, and potentially to a validation error if the equality condition does not hold. In contrast, constraints in the project scope can be applied to global decision variables only and may need explicit quantification.

In Figure 6, the two global constraints at the end of the fragment ensure that at least one machine instance of each role is configured (implying that at least two machines are configured). The constraint relating available ports to configured pipelines will be discussed along with the infrastructure settings in Section 7.2.9.

In summary, the configuration of the general purpose machines realizes REQ-C-2.

```
project Hardware {
  import Basics;
  import Observables;

  attribute BindingTime bindingTime = BindingTime.compile to Hardware;

  enum MachineRole {Manager, Worker};

  compound Machine {
    NonEmptyString name;
    MemorySize memory;
    PositiveInteger processors;
    Frequency frequency;
    setOf(PortInteger) ports;
    MachineRole role;
    // machine-specific runtime variables
    assign(bindingTime = BindingTime.runtime) to {
      Bandwidth bandwidth;
    }

    // managers do not have configured ports
    role == MachineRole.Manager implies ports.isEmpty();
    // workers without configured ports receive the default Storm ports
    role == MachineRole.Worker and ports.isEmpty() implies ports == {6700, 6701, 6702, 6703};
  }

  setOf(Machine) machines;

  // At least one manager must be assigned
  machines->exist(machine|machine.role == MachineRole.Manager);
  // At least one worker must be assigned
  machines->exist(machine|machine.role == MachineRole.Worker);
.. // Machine names must be unique
  machines->collect(m|m.name).size() == machines.size();

  // global runtime variables
  assign(bindingTime = BindingTime.runtime) to {
    UsedMachines usedMachines;
    AvailableMachines availableMachines;
  }
}
```

Figure 6: IVML fragment specifying the for configuration opportunities for software-based execution.

While the QualiMaster infrastructure requires the presence of general-purpose cluster machines, **hardware-based execution** is optional, in particular to enable the use of the QualiMaster infrastructure also in settings where no reconfigurable hardware is (yet) present or reconfigurable hardware is considered as too expensive. In case reconfigurable hardware is present, the configuration defines the environment for the dynamic migration of analysis tasks to and from reconfigurable hardware at runtime. In general, reconfigurable hardware may range from the different hardware series provided by Maxeler over the series provided by other vendors of reconfigurable hardware to Graphics Processing Units (GPU). A specific kind of reconfigurable hardware may require different information for configuration, instantiation or adaptation at runtime. Thus, in the Configuration Meta Model, reconfigurable hardware is modeled in an extensible way, e.g., also for later load testing or adoption as described in the DoW, rather than in the generic way we used for general-purpose hardware. While the `HwNode` defines the quality parameter `bandwidth`, the `MPPCNode` defines the quality parameters for used / available CPUs and DFEs according to Section 6.2. The configurable elements for configuring hardware-based execution are depicted in Figure 7.

The root configurable element hardware-based execution is `HwNode`. `HwNode` defines only the (descriptive) name of the hardware board or cluster of boards, respectively. In QualiMaster, we currently focus on the Maxeler MPC-C series for hardware-based execution. Therefore, according to the MPC-C series node architecture (see REQ-C-3 or D5.1, Section 6.2) the specific configurable element `MPPCCNode` can be configured with the available number of Central Processing Units (CPUs) and Data Flow Engines (DFEs). The host address allows connecting to the underlying hardware in order to access the functionality of the MaxelerOS, which manages the CPUs and DFEs. Please refer to D3.1 for further details. All configured `HwNode` instances are stored in a collection accessible to the `Infrastructure`.

The constraints for the Hardware-based execution are implied by the types used for defining `MPPCCNode`, i.e., the number of CPUs and DFEs must be positive (rather than possibly negative or 0 as for the general type Integer). Further, `HwNode`s must be uniquely identifiable via their name.

In summary, the configuration capabilities provided by the Configuration Meta Model for hardware-based execution realize REQ-C-3.

7.2.4 Data Management

The Data Management Layer of the QualiMaster infrastructure is responsible for the management of the physical data sources, data sinks and for automatically storing input and output data according to certain storage strategies (see D5.1). In particular, this enables multiple pipelines to use the same logical source from the data management layer, i.e., storing raw input as historical data cannot be triggered by multiple pipelines. Further, the Data Management Layer also supports persisting temporary streaming data generated by processing elements (as specified in the

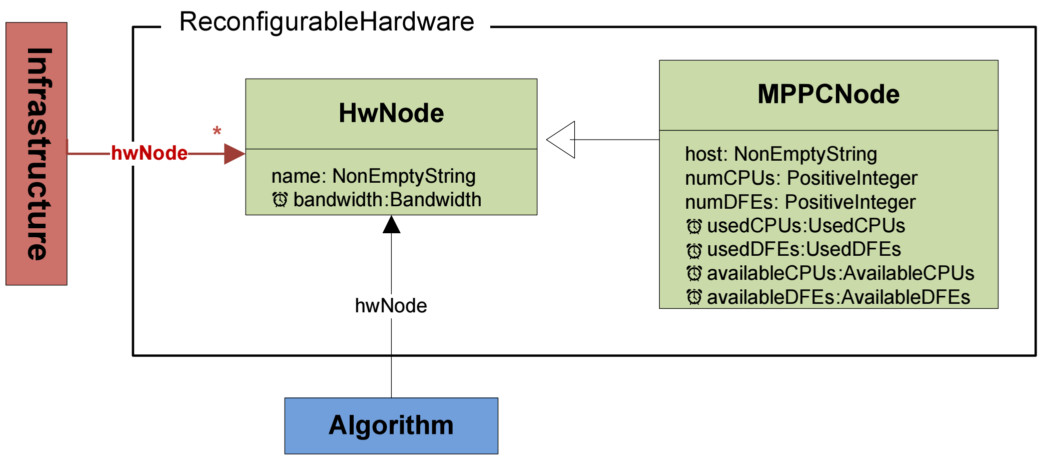

Figure 7: Part of the Configuration Meta Model for hardware-based execution.

processing pipelines) as well as historical data for batch processing.

As shown in Figure 8, the common configuration options are defined in `DataElement` and include the (descriptive) `name`, the `storageLocation`, the data management `strategy` as well as additional strategy-related data such as the `timeline` (e.g., keep data for at maximum timeline of 40 days) or the `cutofCapacity` (e.g., to keep at maximum 1TBytes of historical data). Finally, a `DataElement` enables the specification of user defined `constraints`, in particular SLA constraints for the refining compounds that rely on the runtime variables defined there.

Since data streams are typically unbounded in size and time, storing all results of the continuous stream processing is not feasible. Therefore, some strategies for regular storage cleanup are required. The problem we are tackling here is quite similar to the caching problem in operating and storage systems. Thus, we adopt some of the widely used caching strategies to decide on which items should be kept and which items have to be replaced by newly arriving items. Given a limitation on the storage capacity of the infrastructure, say *k* Bytes (`cutoffCapacity`), the data management layer simply keeps adding new items to the storage space until the maximum

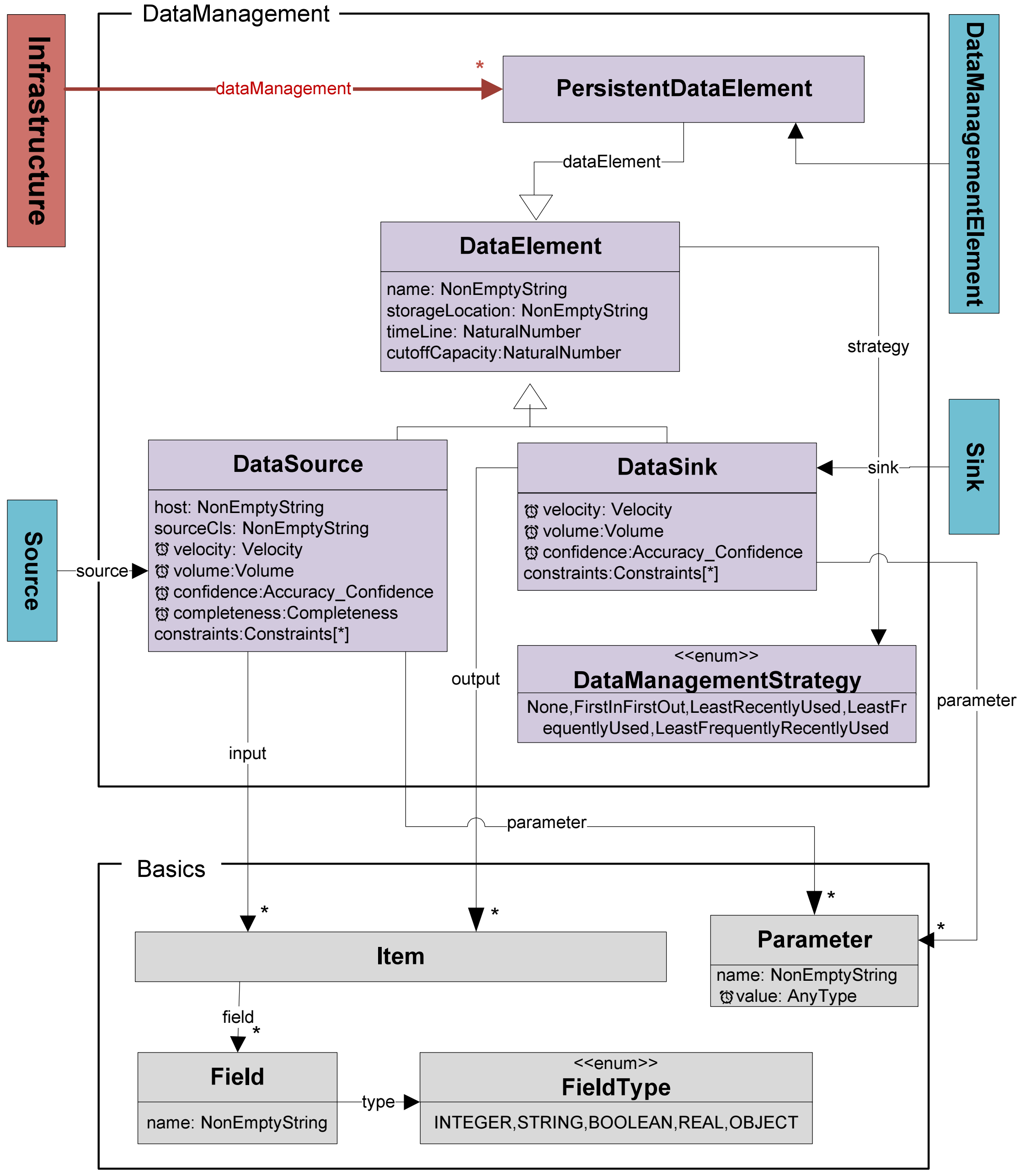

Figure 8: Part of the Configuration Meta Model representing the configuration of the Data Management layer.

capacity *k* is reached.

We consider the following strategies upon arrival of news data items to be stored:

- First-in First-out (FIFO): This a simple strategy that removes the oldest items from the storage space in order to insert the new item.
- Least Frequently Used (LFU) [154]: This strategy keeps track of user access to the content of the storage. It also logs the frequency of accessing each item. When the `cutoffCapacity` is reached, new items replace the least frequently accessed items. The idea is that those least frequently accessed items are estimated to be least relevant for the users and hence the cost of replacing them should be low.
- Least Recently Used (LRU) [85]: This strategy is similar to the LFU except that the system here logs the time at which each item has been most recently accessed. This allows ordering the items based on their access history. The least recently accessed items are considered to be less relevant for the users and therefore are candidates for replacement.
- Combination of LFU and LRU: It is also quite common that a combination of the LFU and LRU strategies is used like the work of [102] in order to have a better estimation of importance of the items and reduce the risk to replace an item that might still be of use in the future.

The strategies described above are represented by the enumeration `DataManagementStrategy`.

A `DataSource` is a refined `DataElement` and denotes the actual physical source as well as the types of the input items. An input item may consist of multiple named and typed fields representing the typed sequence of individual data in a data stream. The related `Item` type is inspired by the Apache Storm notion of data streams and defined in `Basics`. Further, `Item` is composed of a list of `Field`s carrying (a descriptive) name and type information. A `DataSource` may be configured at runtime through a set of functional `Parameter`s, e.g., to control the abonnement of markets or market players as well as to define filter terms directly on the source. The `sourceCls` refers to the actual implementation class of the data source, e.g., to retrieve financial raw data from the SPRING source using a specific protocol or to access data from Twitter (cf. Section 7.2.5). A `DataSource` declares runtime variables that represent runtime quality properties according to the quality taxonomy in Section 6.2, such as the `volume` or the `volatility`.

Akin, a `DataSink` is a refined `DataElement`, which passes the output items to the QualiMaster applications. A sink may have functional parameters such as credentials. Further, a `PersistentDataElement` is also a refined `DataElement`, representing intermediary data to be accessed from the pipelines. Both elements will be refined, e.g., by added decision variables, depending on the access needs of the QualiMaster applications and the realization of the Data Management Layer. Akin to `DataSource`, a `DataSink` defines the respective runtime variables for quality properties.

Most of the constraints for the Data Management Layer are implied in the types used for defining the variables. We model the data management strategy as an enumeration, some constraints are required to link `timeline` or `cutoffCapacity` with the configured `strategy`. Later extensions of the Configuration Meta Model may represent strategies as compound types, thus simplifying the constraints through refined data types. Further, data sources and data sinks must be uniquely identifiable via their names for code generation during platform instantiation.

In summary, the types defined for the configuration of the Data Management Layer realize REQ-C-4, REQ-C-5, and REQ-C-6. In particular, the functional parameters help realizing REQ-A-1 at data source level.

7.2.5 Data Processing Algorithms

This part specifies the configuration of the data processing algorithms. A data processing algorithm realizes a specific analysis task, can be adjusted via functional parameters (e.g., by runtime adaptation) and can be implemented in terms of a

- Plain Java algorithm,
- Pre-laid out Storm topology (considered as a black box algorithm),
- Single hardware-based algorithm or
- Library of hardware-based algorithms that may cover an entire sub-pipeline, potentially with optional algorithms at the beginning or end of the sub-pipeline or even alternative hardware-based algorithms.

Please refer to D5.2 for a more detailed description of the four algorithm forms.

The data processing algorithm part of the Configuration Meta Model illustrated in Figure 9 covers these four implementation forms. However, configuring these different alternatives, in particular the different choices in alternative 4) may be an error-prone and tedious activity. Thus, in QualiMaster,

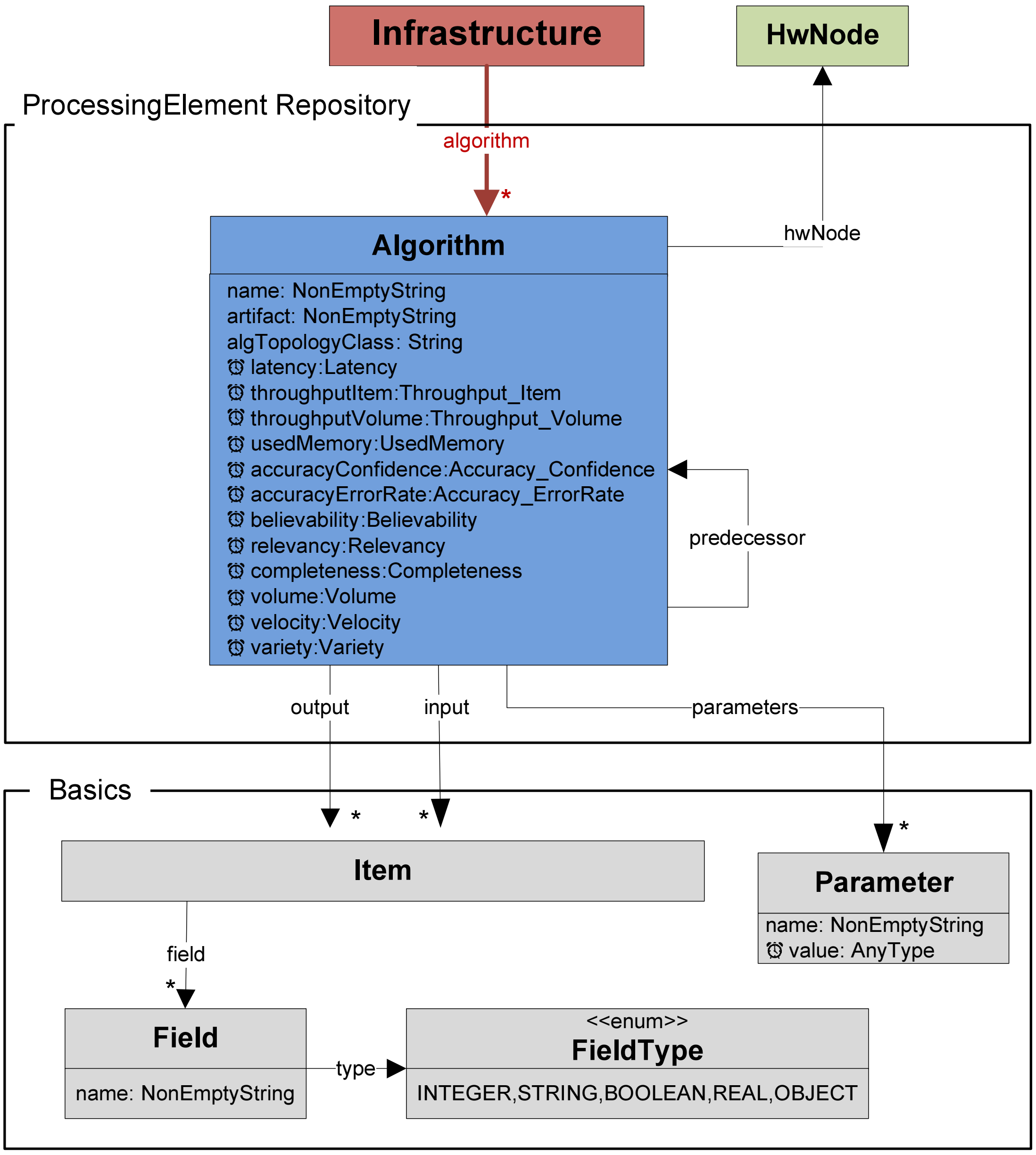

Figure 9: Part of the Configuration Meta Model representing the configuration of the data processing algorithms.

the implementation of algorithms is physically represented as an **extension package** (alternatively, it may contain code modules for data sources or monitoring), which carries knowledge about the contained algorithm, e.g., the respective implementation alternative, the implemented input / output types as well as the (known) quality characteristics. Loading this package into the Algorithm Repository via the QualiMaster infrastructure configuration tool will perform the required configuration activities and simplify the configuration. Although it might be possible to capture all this information in the Configuration Meta Model, we decided to store the implementation specific information as well as the quality properties in a database in the Data Management Layer as this will unify and simplify the tradeoff making at runtime. This is due to the design goal of the Configuration Meta Model to capture the information that is needed to perform static feasibility checks and to perform the infrastructure derivation. Thus, the Configuration Meta Model can abstract over the different implementation forms in terms of the `Algorithm` type. The notion of a data processing algorithm in this configuration part abstracts over the different implementation forms. Further, the `Infrastructure` knows all configured `Algorithm` instances.

The input / output of an algorithm are formalized by the `Item` type as already introduced in Section 7.2.4. In addition, the operation (and the quality) of an algorithm may be influenced by functional parameters that can be changed at runtime possibly influencing the quality characteristics of the algorithm (as described for data sources in Section 7.2.4), such as the window length to be processed or filter criteria. An algorithm may also specify the direct `predecessor` it depends on. Currently, this is mainly intended to capture the relevant dependencies in libraries of hardware algorithms described above and to perform static feasibility checks on the configuration, in particular the pipelines.

Currently, the constraints for the configuration of the Data Processing Algorithms are implied in the types used for defining the variables. Please note that this is in particular true for unique field names as well as unique parameter names. Further, algorithms must be uniquely identifiable via their names for code generation during platform instantiation and adaptation.

The `Algorithm` type and its related configurable elements described in this Section realize REQ-C-7. In particular, the algorithm parameters help realizing REQ-A-1 for certain algorithms.

7.2.6 Algorithm Families

Algorithm families group data processing algorithms with similar functionality, but different quality characteristics and tradeoffs. Basically, the part of the Configuration Meta Model for algorithm families follows the modeling for Algorithms as shown in Figure 10. An algorithm family defines the input and output types of its members and the (common) set of functional parameters that all family members shall support. Thus, an algorithm family abstracts over the contained algorithms and does not take implementation level information into account.

As depicted in Figure 10, a `Family` is configured by its name, input / output item types, functional parameters (cf. Section 7.2.4 and 7.2.5) as well as the member algorithms of the family. Further, an algorithm `Family` defines the same runtime variables for quality parameters as an Algorithm and, in particular, a runtime variable containing the `actualMember` to be executed. The runtime variables for quality parameters are bound by constraints to the `actualMember`. The `actualMember` enables to define and enact algorithm changes at runtime (cf. Section 8) as well as the specification of (pipeline) constraints. As usual, the `Infrastructure` knows all configured `Family` instances.

Due to the semantic relations described above, we defined several constraints defined on the Algorithm Families (in addition to the binding of quality properties mentioned above). In particular, these constraints must be validated when the membership relations are changed, especially when new members are added.

- An algorithm family must contain at least one algorithm.
- The input item fields and types of all member algorithms must match the input types of the algorithm family. Thus, the input sent to the family can be processed by all members of the family.

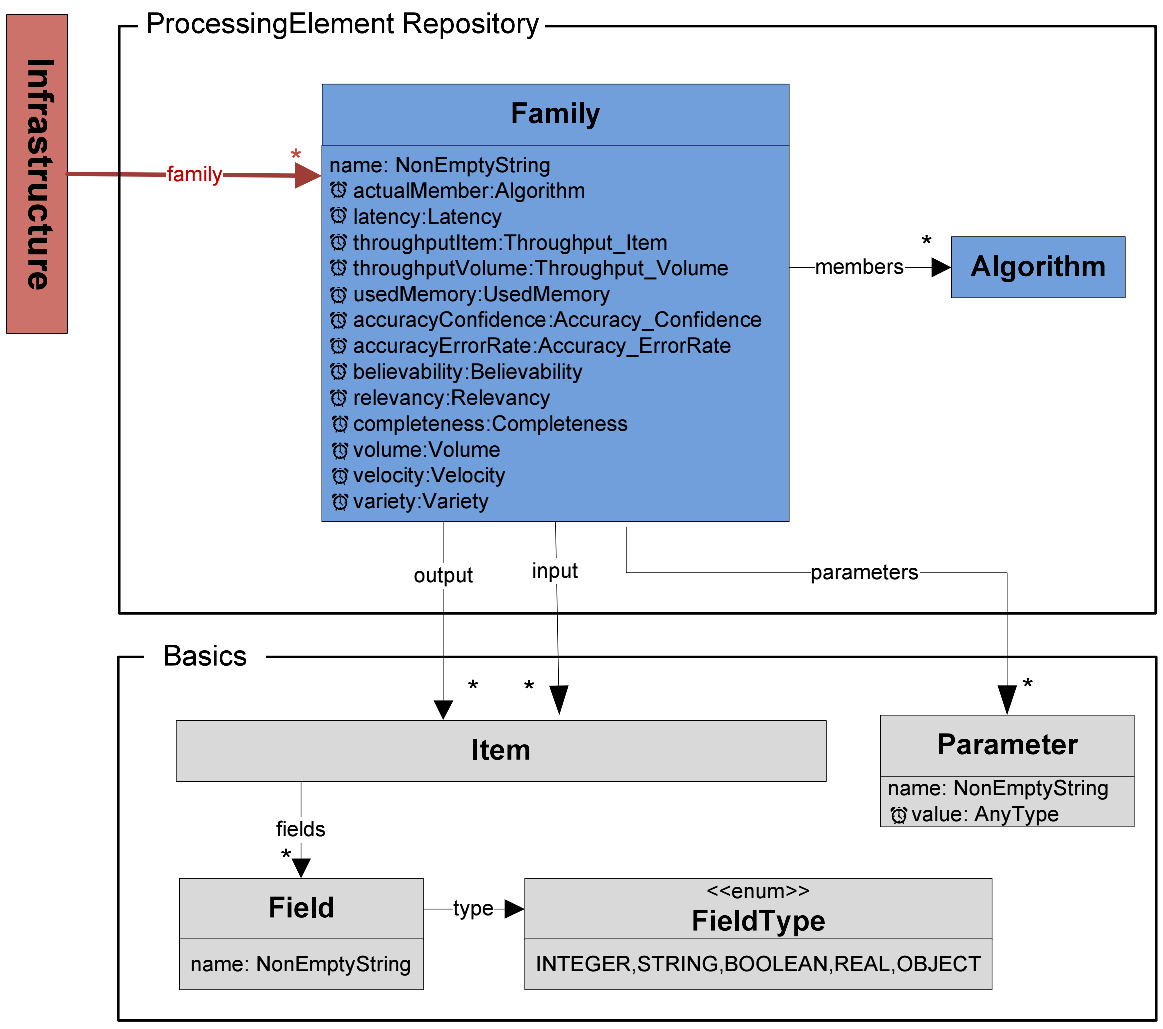

Figure 10: Part of the Configuration Meta Model representing the configuration of the algorithm families.

- The output item fields and types of all member algorithms must match the output types of the algorithm family. Thus, the output produced by any member of the family corresponds to the output of the family.
- The functional parameters of a family contain all functional parameters of all member algorithms. Thus, it is open to an individual algorithm whether it actually implements a certain parameter, e.g., changing a window size may not be possible for all kinds of algorithms.
- Families must be uniquely identifiable via their names, in particular for code generation during instantiation and adaptation.

In summary, the configuration options for processing families fulfill REQ-C-8.

7.2.7 Data Processing Pipelines

The concept of adaptive data processing pipelines is one of the core ideas of the QualiMaster project. According to the pipeline execution model described in Section 2.2, the data stream is provided by a Data Source to the pipeline for processing. The data stream flows through a number of data processing elements, each representing either

- An Algorithm Family, i.e., a set of alternative algorithms to be selected dynamically at runtime.
- A Data Management element in order to store intermediary information produced while processing a data stream in the Data Management Layer.

- A Generic Data Stream Processing Operation such as filter, select or project as also used by approaches discussed in Section 4.1.2. Actually, the set of generic stream processing operations to be provided by the QualiMaster infrastructure is in discussion.

Finally, the data stream reaches a specified Data Sink and can be visualized or processed by applications. Thus, the concepts of a QualiMaster data processing pipeline correspond to an extended data flow graph as frequently used for specifying the data processing in the data stream community (see e.g., [25, 82]). There, a data flow is stated in terms of sources, sinks and data stream processing operations, i.e., QualiMaster extends the traditional notion of a data flow graph by Algorithm Families and Data Management operations. Please note that workflow languages such as the Business Process Execution Language[9] are typically control-flow oriented and provide procedural concepts such as (while) loops that do not apply to data stream processing. In contrast, control-flow oriented languages may be used to specify or visualize the internal behavior of algorithms except for (black box) sub-topologies.

Figure 11 visualizes the configurable elements for data processing pipelines. We call all configurable elements that can be used in a pipeline a `PipelineElement`. In general, a `PipelineElement` can be configured by user-define pipeline constraints. A `PipelineNode` is a refined `PipelineElement` running on a certain number of Workers (`parallelism`) that represents a `Source` (references a `DataSource` from Section 7.2.4), a Sink (references a `DataSink` from Section 7.2.4) or an actual `ProcessingElement`. A `ProcessingElement` can either be a `FamilyElement` representing a processing `Family` (see Section 7.2.6), a data

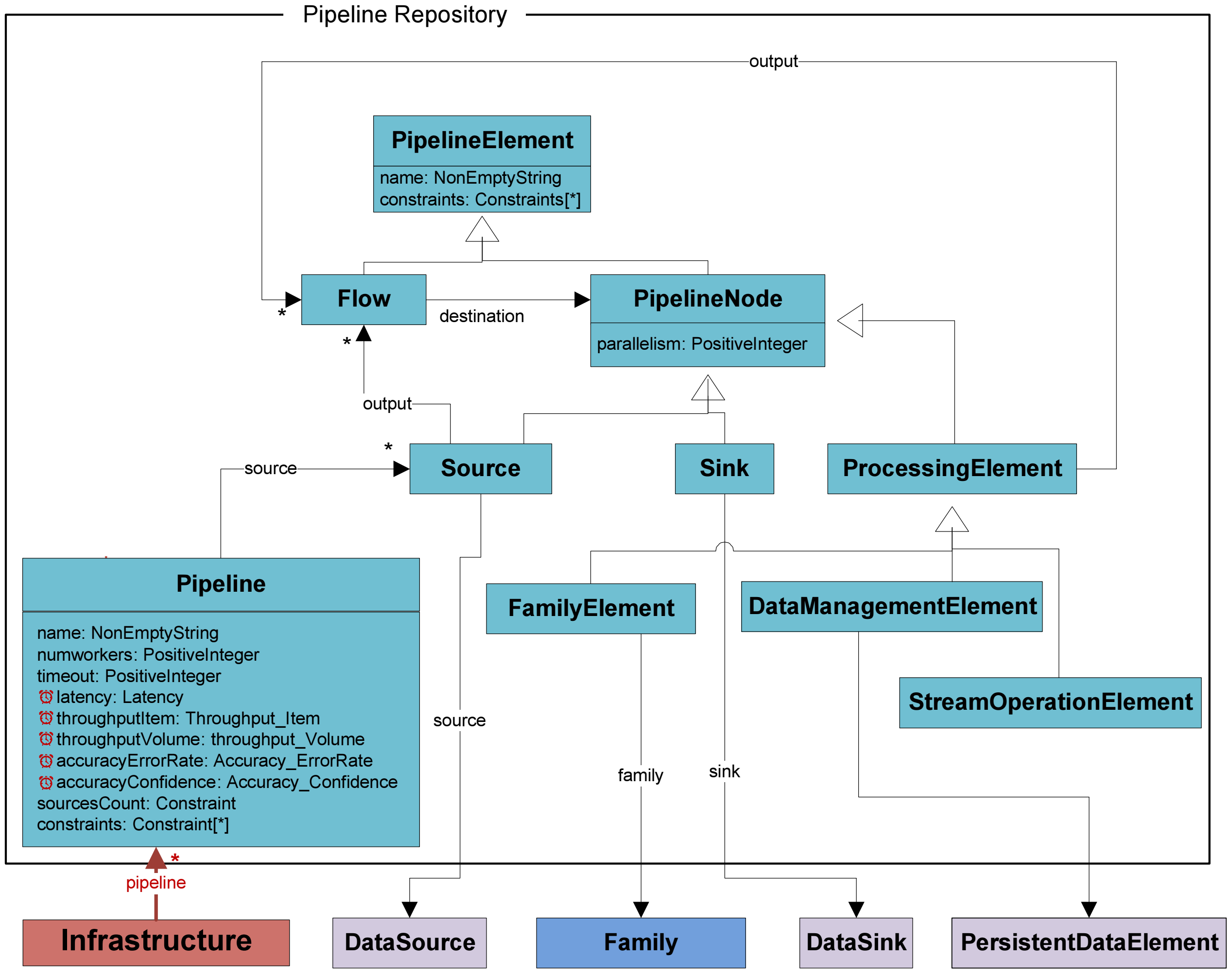

Figure 11: Part of the Configuration Meta Model representing the configuration of data processing pipeline topologies.

[9] http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

`DataManagementElement` causing storage actions to a certain `PersistentDataElement` in the Data Management Layer (see Section 7.2.4) or a `StreamOperationElement` (currently not further specified, but intended to represent generic stream operations such as filter, project, join, etc.). A `Flow` represents the connection between two adjacent `PipelineNode`s in a pipeline. Basically, the semantics of a flow is that all items of the declared types from the start point are transported to the end point of the flow. If the end point declares less item types in the same sequence as the start point then we consider this as an implicit projection. If the end point declares more types, multiple input flows can be joined implicitly. Finally, a `Pipeline` is defined via its data sources, which link via flows to processing elements, etc. Further, a `Pipeline` can be configured in terms of its required number of worker nodes, its message passing timeout and by user-defined pipeline constraints. Further, the `Infrastructure` knows all configured `Pipeline` instances. According to the quality taxonomy discussed in Section 6.2, the `Pipeline` compound defines runtime decision variables in order to provide access to the results of the pipeline quality analysis (REQ-A-12, cf. Section 6.3).

Please note that in the Configuration Meta Model, the connection between pipeline nodes through flows represents a graph, i.e., the topology of a data processing pipeline through typed configuration references. Please note also that Configuration Meta Model for the data processing pipelines is tailored for the pipelines that are supported by the QualiMaster approach and the realizing infrastructure, i.e., we aim at avoiding illegal pipelines (such as not starting with a source) already in the configuration. As more pipeline capabilities may become available or needed during the project, we will evolve the Configuration Meta Model accordingly.

Based on the previous model parts and their constraints, the following constraints apply to pipelines (and realize REQ-C-10):

- Processing pipelines must be structurally valid, i.e., processing elements must be connected correctly. A pipeline must have at least one source. Data sources are the only pipeline nodes that have no input flow while data sinks are the only pipeline nodes that have no output flow.
- Processing pipelines must be type conformant. The input items of a processing element must match the output items of the preceding data source or processing element. More precisely, there must be an overlap between the input items and the output items of the preceding elements in order to allow implicit stream joins and projections. Data Management elements forward their input to their output and are, thus, transparent. The input item types must match the data source must match the output item types of the preceding processing element.
- Pipelines must be uniquely identifiable via their names, in particular for code generation during pipeline instantiation and adaptation.

The realization of the pipeline part of the Configuration Meta Model shown in Figure 11 in terms of IVML is illustrated in Figure 12. Basically, the compounds and their refinement hierarchy shown in Figure 12 correspond to the pipeline concepts discussed above (`StreamOperationElement` is omitted). As explained above, the first pipeline constraint is mainly expressed due to the types of pipeline elements and the typed references. In addition, the constraint in `Flow` avoids that data flow goes back to a source and the constraint in `Pipeline` requires that a valid pipeline has at least one source. The specification of the type conformance constraint is “distributed” across the model elements and relies on (internal) decision variables defined in `PipelineNode` representing the input and output types. This is due to the fact that a `DataManagementElement` is considered as being transparent to the data flow (input item types are the same as output item types).

```
// project scope, imports and attributes omitted

abstract compound PipelineElement {
  NonEmptyString name;
  setOf(Constraint) constraints = {}; // user constraints
}

compound Flow refines PipelineElement {
  refTo(PipelineNode) destination;
  destination.typeOf() <> Source;
}

abstract compound PipelineNode refines PipelineElement{
  Items inputTypes;
  Items outputTypes;
}

compound Source refines PipelineNode {
  setOf(refTo(Flow)) output;
  refTo(DataSource) source;
  inputTypes = source.input; // take over types from source, pass through and check type
  outputTypes = inputTypes;
  typeCheck(self, output);
}

// Sink omitted as similar to Source, generic stream operation type omitted

compound ProcessingElement refines PipelineNode {
  setOf(refTo(Flow)) output;
  typeCheck(self, output); // check types for all processing elements
}

compound FamilyElement refines ProcessingElement {
  refTo(Family) family;
  inputTypes = family.input; // take over types from family
  outputTypes = family.output;
}

compound DataManagementElement refines ProcessingElement {
  refTo(PersistentDataElement) dataManagement;
  inputTypes = outputTypes; // transparent to processing
}

compound Pipeline {
  NonEmptyString name;
  setOf(refTo(Source)) sources;
  PositiveInteger numworkers; // timeout omitted
  setOf(Constraint) constraints = {};
  sources.size() > 0;
  // runtime variables omitted
}

sequenceOf(refTo(Pipeline)) pipelines; // name uniqueness constraint omitted

// number of workers is less then reserved ports for machines
pipelines->apply(refTo(Pipeline) pipeline; Integer totalNumWorkers = 0 |
  totalNumWorkers = totalNumWorkers + pipeline.numworkers) <= machines->apply(Machine machine;
  sequenceOf(Integer) usedPorts = {} | usedPorts.union(machine.ports.asSequence())).size();

def Boolean typeCheck(PipelineNode src, setOf(refTo(Flow)) output) =
  output->forAll(f|typeCheck(f.destination, src));

  //explicit propagation
  def Boolean typeCheck(PipelineNode src, PipelineNode dst) =
    if isDefined(dst.inputTypes) and isDefined(src.outputTypes)
      then src.outputTypes.overlaps(dst.inputTypes)
      else dst.inputTypes == src.outputTypes endif;
```

Figure 12: IVML fragment specifying the Configuration Meta Model for pipeline topologies.

Thereby, the input and output item types of data sources, data sinks and families act as anchor points and the constraint becomes effective due to side effects, namely value assignment and value propagation. Thereby, known item types are assigned incrementally, while remaining types are either set correctly through the type check or, if already assigned and input / output types do not match, cause a validation failure. In more detail, for a `Source` the input corresponds to the input of the physical data source just passed through to the output of the source. Akin, for a `Sink` the output is determined by the physical data sink and passed through from its input. Input and output of `FamilyElements` depend on the specification of the underlying family. Further, in the future, generic data stream operations (`StreamOperationElement` is omitted in Figure 12) may change the data items, i.e., the output depends on the configuration. Type checking is actually performed by the function `typeCheck`, which considers all flows of a given pipeline node and either leads to a value propagation or, if values are known, requires that the input types of the destinations are a sub-sequence of the preceding output types. `typeCheck` is applied as a constraint in `Source` as well as generically in `processingElement`. Please note that this "distributed" constraint could also be specified through a set of specific functions and dynamic dispatch, but the specification would be more complicated. Also recursive functions may be used, but this would require explicit cycle checking (not needed in the actual model as always only two pipeline nodes are compared) and may impact the efficiency of constraint validation.

In summary, the data processing pipelines part of the Configuration Meta Model utilizes the capabilities required by Challenge C3 and realizes requirement REQ-C-9. In this section, we modeled QualiMaster data processing pipelines as a specialized IVML topology. The QualiMaster platform instantiation process (cf. D5.2) turns this topology into an implementation for Apache Storm. In Section 7.3, we discuss how configuration can be done using the QualiMaster infrastructure configuration tool, a domain-specific configuration frontend supporting the user in configuring such specialized topologies in terms of data processing pipelines.

7.2.8 Adaptivity

The adaptivity part of the Configuration Meta Model targets the high-level "knobs" and settings an Adaptation Manager can change in order to influence the adaptive behavior without deeper knowledge on the adaptation behavior specification. Please note that this model part is not the Adaptation Model, which specifies the adaptive behavior. The Adaptation Model will be introduced in Section 8.

According to the quality survey described in Section 6.1, the configuration of the quality tradeoffs and the cross pipeline tradeoffs shall be available to the Adaptation Manager. Basically, this model part defines weights to be assigned to the individual quality parameters as illustrated in Figure 13. Thus, the compound `QualityParameterWeighting` combines a reference to a quality descriptor defined in the observables part in Section 7.2.2 with a weighting. The decision variable `pipelineImportance` captures then all weights. We use the same compound (for now) to

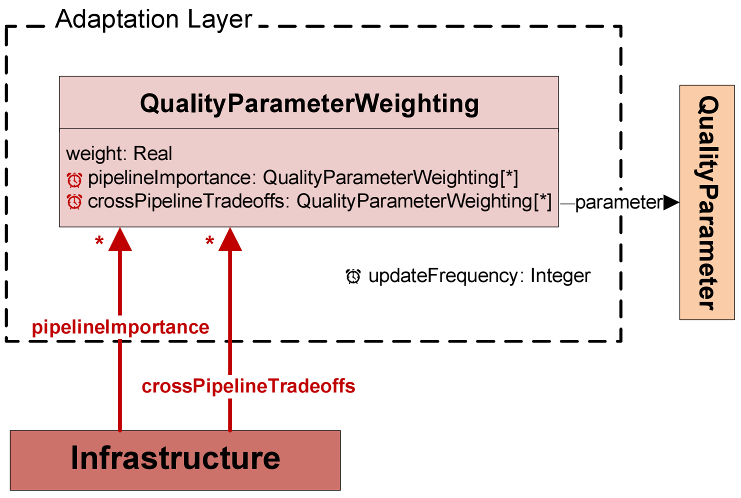

Figure 13: Overview on the adaptation part of the Configuration Meta Model.

describe the importance of the cross-pipeline tradeoffs, stored in `crossPipelineTradeoffs`. Furthermore, we allow changing the `updateFrequency` of the regular adaptation schedule (either 0 to disable the regular schedule or more than 500 milliseconds). We declare these settings as runtime variables, as they may potentially be changed at runtime, and provide default values.

This model part realizes requirement REQ-C-14.

7.2.9 Infrastructure

The infrastructure part of the QualiMaster Configuration Meta Model links all the individual model parts discussed above in order to describe the configuration of the overall QualiMaster infrastructure / platform for a certain setting.

Currently, configuring the QualiMaster infrastructure allows selecting the pipelines to be instantiated and executed (REQ-C-15), i.e., the infrastructure part mainly consists of a collection decision variable, which specifies the selected pipelines. Thus, we do not illustrate this model part in terms of a figure. Future versions of this part may also contain details about the installed Execution Systems, the repositories or the technical communication information needed to communicate with the individual layers of the QualiMaster infrastructure, in particular the Coordination Layer.

The infrastructure part defines one constraint, which requires that the number of workers required by all selected pipelines must be less or equal than the number of reserved ports for all machines in the cluster. An erroneous configuration may prevent Storm from starting pipelines (incrementally). This contributes to the realization of REQ-C-10. Due to future configuration knowledge, this constraint may be refined in subsequent versions of the Configuration Meta Model.

7.3 Tool Support

Tool support is required to enable the user to define a configuration, validate it, execute the platform instantiation and, in if required for adopting the QualiMaster infrastructure, to adjust the Configuration Meta Model or the specification of the platform instantiation (Challenge C4, cf. D5.2 for more details). We utilize two specific tools for this purpose, namely the *QualiMaster infrastructure configuration tool* and *EASy-Producer*, a prototypical SPLE tool developed by SUH. While the QualiMaster infrastructure configuration tool supports the domain users in configuring their platform, EASy-Producer is used for defining and changing the Configuration Meta Model and the derivation process. Please note that EASy-Producer does currently not provide domain-specific capabilities, while the QualiMaster infrastructure configuration tool is a domain-specific frontend supporting the Infrastructure Users in their tasks. In this section, we give an overview on the current state of both tools and, in particular, the improvements over existing work that already have been done during the QualiMaster project. We start with the QualiMaster infrastructure configuration tool and discuss the improvements for EASy-Producer afterwards.

The **QualiMaster infrastructure configuration tool** (QM-IC for short as in D1.2) aims at simplifying the configuration of the QualiMaster infrastructure for the infrastructure users, namely, Infrastructure Administrator, Pipeline Designer and Adaptation Manager. As described in D5.1, QM-IC tool is based on the Configuration Core, which consists of EASy-Producer. QM-IC tool is being developed in the QualiMaster project. Where possible, it reuses functionality of EASy-Producer, e.g., to load IVML or VIL models, to reason on IVML models or to execute VIL models for instantiation. However, EASy-Producer does currently not provide domain-specific capabilities required to support the QualiMaster Infrastructure Users in their tasks, in particular not for configuring data processing pipelines. Therefore, QM-IC tool acts as a domain-specific configuration frontend, which also shields Infrastructure Users from the generic SPLE capabilities of EASy-Producer. Please note that in another application setting, a domain-specific configuration tool could allow the user to model rooms with alarm zones, fire detectors and sirens required in the alarm system case [16], or (3D) racks and stapler cranes for configuring a warehouse layout [32].

On the user interface, QM-IC tool provides access to configuration tasks according to the parts of the Configuration Meta Model introduced in Section 7.2. These tasks are indicated on the left side of Figure 14, which shows a screenshot of configuration tool. Each of the tasks provides a specific

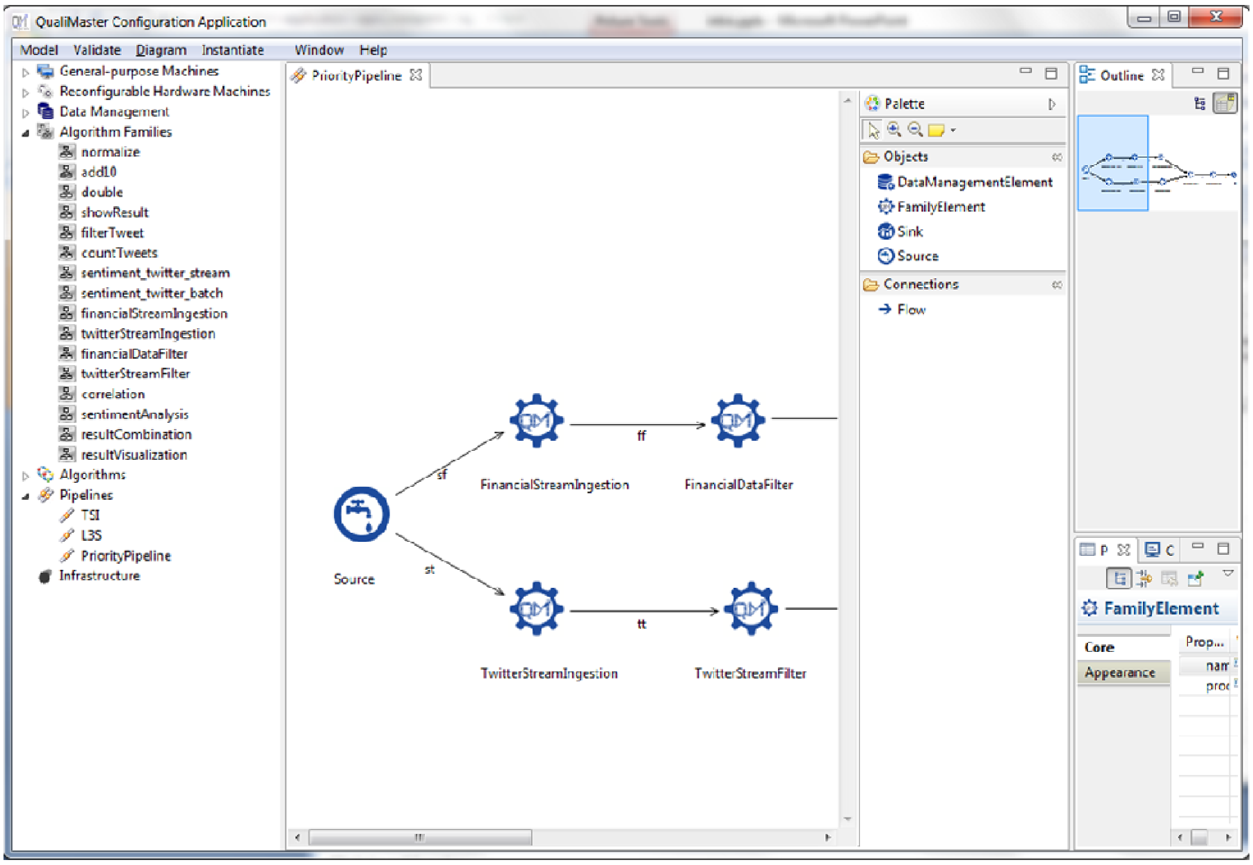

Figure 14: QualiMaster Infrastructure Configuration Tool QM-IC tool (left: model parts and contained configurable elements, right: domain-specific pipeline editor).

editor to support the configuration activities. The editors for most parts of the Configuration Meta Model are dynamically derived from the Configuration Meta Model rather than being implemented manually. Thus, the editors are automatically synchronized against the Configuration Meta Model.

In particular, for configuring QualiMaster data processing pipelines, the configuration tool provides a graphical drag & drop editor, which exactly supports the QualiMaster pipeline modeling concepts and the selection of already configured elements such sources, sinks or processing families (REQ-C-9). The graphical editor is shown on the right side of Figure 14. The pipeline editor represents a graphical data flow specification as usual in the data stream processing community (see e.g., [25, 82]) as discussed in Section 7.2.7, here using icons inspired by Storm and the specific concepts of QualiMaster. As an alternative, we considered a textual domain specific language as, e.g., used in [123], but the QualiMaster consortium decided for the graphical editor in the tradition of stream data processing as more appropriate representation for the infrastructure users and for future adoption. Actually, the graphical pipeline editor is generated using Eclipse Eugenia[10] / Epsilon[11] based on an editor specification derived from the IVML pipeline model discussed in Section 7.2.7, i.e., a synchronization with the Configuration Meta Model currently needs a regeneration and compilation of the editor components. The property selections for characterizing the individual pipeline elements (depicted in the lower right side of Figure 14) are directly linked against the underlying IVML configuration and reuse UI components provided by EASy-Producer.

IVML and the domain-specific configuration frontend of QM-IC tool realize Challenge C3. The QualiMaster infrastructure configuration tool is a basis for further research on automated derivation of domain-specific configuration applications, either within QualiMaster (e.g., on ensuring consistency in Task T4.2) or in further work outside the project.

[10] http://eclipse.org/epsilon/doc/eugenia/
[11] https://www.eclipse.org/epsilon/

Following the refinements of the use cases in D1.2, QM-IC tool integrates the configuration functionality for all Infrastructure Users, in particular to provide a comprehensive overview on the configuration of a QualiMaster infrastructure. Thus, the views of the three infrastructure user groups are available in one tool, but also access protected based on user roles. As a consequence, a Platform Designer who is not a Platform Administrator may view the settings for the reconfigurable hardware but he/she is not permitted to change these settings. These permissions are obtained from the user authentication against the Pipeline Repository, which is required to access the Configuration (Meta) Model or to synchronize it back. Thus, the configuration tool also realizes REQ-C-1, i.e., (application) users do not have access to the configuration as the required permissions may not be granted to them. Finally, QM-IC tool allows executing the QualiMaster platform instantiation process (cf. D5.2).

Although the current version of the Configuration tool supports most of the configuration and instantiation activities, several specific capabilities still need to be developed, such as the configuration of platform quality parameters (UC-PA1, UC-PA2), the adaptation rules and settings (UC-AM1-UC-AM4), the adaptation monitoring view (UC-AM5), the deployment and startup operations (UC-PA8, UC-PA9) as well as a user-friendly editor for SLA constraints.

EASy-Producer [42] is an Eclipse-based research tool for the management and development of product lines, which provides particular support for hierarchical product lines, multi-product lines and Software Ecosystems. EASy-Producer has mainly be developed in FP7 INDENICA[12] and maintained and improved in the national funded project ScaleLog. In QualiMaster, EASy-Producer has been maintained and improved to tackle the configuration challenges identified in Section 3. So far, we worked on the following topics in QualiMaster:

- **Topology modeling and instantiation:** Actually, IVML was designed with capabilities for configuration references due to an industrial requirement in the INDENICA project [34]. However, so far this capability was not used intensively. Thus, expressing complex topology constraints was rather difficult, in particular in topologies over refinement hierarchies as we use them for data processing pipelines (cf. Section 7.2.7), where the evaluation of constraints depends on the actual type of object being evaluated. Object oriented systems handle this through polymorphism, script-like languages, such as VIL through dynamic dispatch. Inspired by type selection capabilities and the extensibility of VIL, we extended IVML constraints with dynamic dispatch, i.e., user-defined functions can now be called dynamically based on the actual parameter type. This also supports the formulation of some of the challenging constraint problems discussed in [16].
- **Constraint evaluation and propagation:** We revised the constraint evaluation mechanism of EASy-Producer. Our initial approach based on a rule-based truth maintenance system, such as Jess[13] or Drools[14] for constraint evaluation, model validation and value propagation [36]. However, that approach was not able to fulfill the specific constraint evaluation capabilities required for QualiMaster. Additionally, its resource consumption was too high for runtime reasoning, a functionality that we need for bridging the gap between SPLE and adaptive systems (Challenge C1). The revised mechanism is being implemented as part of the QualiMaster project. Early evaluations of the revised reasoning mechanism show its feasibility in terms of concept completeness for IVML and resource consumption, in particular for the expected QualiMaster configuration models.
- **Configuration access during the instantiation process:** While specifying the first versions of the QualiMaster platform instantiation process we uncovered the need for a better integration between configuration (IVML) and instantiation (VIL), in particular to simplify the access to the actual configuration values and to improve the readability of the instantiation specifications. Initially, VIL was equipped with a generic form of integration based on a common type for decision variables and functions providing the access to the actual configuration value given in terms of the name of the configurable element. In

[12] http://www.indenica.eu
[13] http://herzberg.ca.sandia.gov
[14] http://www.drools.org/

QualiMaster, we extended this integration by turning the static but extensible VIL type system into a dynamic type system reflecting the IVML types in VIL. Since currently the specific types of an IVML model are available, also the content assist mechanisms of the EASy-Producer editors can support the user adequately and dynamic dispatch in VIL can even be used for IVML types. This capability helped us in specifying the QualiMaster platform instantiation process (cf. D5.2).

- **Runtime instantiation support:** We extended the VIL set of operations with specific operations for runtime instantiation to support adaptation in QualiMaster, in particular with access to IVML runtime variables (they are not relevant in a traditional product line instantiation processes) as well as access to the model validation and automated value propagation (runtime reasoning). As part of the realization of the Adaptation Model, which is discussed in Section 8, these further specific capabilities will be added.

In addition, many more detailed improvements have been carried out in the first year of the project, as, e.g., documented in the recent versions of the IVML [139] and VIL [140] language specifications. Both, the QualiMaster infrastructure configuration tool as well as the more generic EASy-Producer tool profit from all these improvements.

The configuration activities in QualiMaster are supported by two specific tools. On the one side, the QualiMaster infrastructure configuration tool, which is being developed for QualiMaster, supports domain-specific configuration activities and QualiMaster specific concepts such as reconfigurable hardware, processing families and adaptive data processing pipelines. On the other side, the Open Source EASy-Producer toolset is the support for evolving and adopting the Configuration Meta Model and the platform derivation process of QualiMaster. Further, EASy-Producer is used as a basis for realizing the QualiMaster infrastructure configuration tool. As discussed in this section, further development for both tools is needed to support the full breadth of the QualiMaster use cases.

8 QualiMaster Adaptation Model

In this section, we discuss the approach that QualiMaster takes to realize the specific adaptation challenges of distributed real-time data stream processing (in particular Challenges C1, C5, C6 and C7). Further, in this section, we discuss how we will realize a flexible specification of the adaptation behavior, and, thus, the "adaptation rules" requested by REQ-C-13.

We start with an overview on the adaptation approach in Section 8.1. In Section 8.2, we present patterns for enacting adaptation decisions on the QualiMaster platform, in particular for adapting the processing of real-time data streams. Then, in Section 8.3, we discuss the flexible specification of the (reactive and proactive) adaptation behavior integrated with the Configuration Meta Model. Finally, in Section 8.4, we will discuss the actual state of the related tool support.

Please note that, in contrast to Section 7, this section presents the conceptual foundation of our work as the specification language, the enactment patterns and the adaptation components will be researched, realized and evaluated in the remainder of the QualiMaster project. Further, as already mentioned in Section 2, we do not detail the conception for reflective and cross-pipeline adaptation in this deliverable, as the related tasks are scheduled to start in month 18. However, during the preparation of the configuration and the adaptation behavior model, we already took both topics into account in order to support later research and integration.

8.1 Approach Overview

In this Section, we provide an overview on the QualiMaster adaptation approach and, thus, detail the initial overview given in Section 3. We structure this section along the MAPE-K cycle (monitor-analyze-plan-execute over a common knowledge base) [22, 76, 90] and discuss how the individual parts of the cycle will be realized in QualiMaster. An overview is depicted in Figure 15.

As indicated in Section 3, an important part of the knowledge base is the *runtime configuration*. It consists of the configuration used for the instantiation of the QualiMaster infrastructure and the runtime variables indicating the actual state of the system. Examples for runtime variables are selected algorithms, parameter settings and observables such as quality parameters. In particular, configuration values used during the instantiation of the QualiMaster infrastructure will be available, but, unless marked by specific IVML attributes, considered as constant during runtime. In contrast, runtime variables change during runtime (in particular quality parameters as observed by monitoring) or are being changed by the adaptation such as the currently selected algorithms.

A prerequisite to adaptation is that the system being adapted is **monitored**. Basically, the quality parameters are defined in the Configuration Meta Model (see Section 7.2.2). All quality parameters that are either used in the (SLA) constraints of the Configuration Meta Model or in the specification

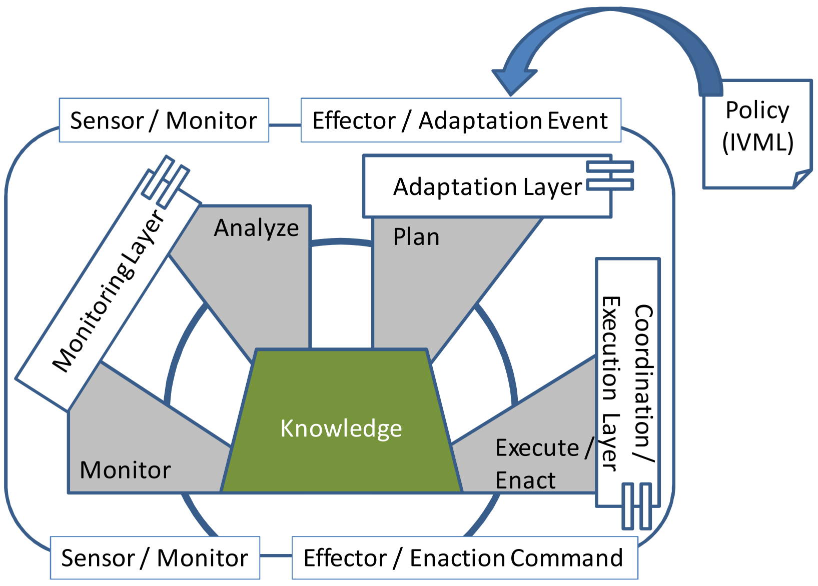

Figure 15: MAPE-K adaptation cycle in QualiMaster (based on [76])

of the adaptive behavior (see Section 8.2) are considered for monitoring (through the platform instantiation). While the values of the quality parameters are physically monitored in the Execution Layer (through instrumented code, generated code or existing interfaces as described in D5.1), the Monitoring Layer of the QualiMaster infrastructure receives, aggregates and, if required, persists the values in terms of quality profiles in the Data Management Layer.

Next, the actual values of the quality parameters are **analyzed**. In the architectural blueprint of adaptive systems [76] as well as in many approaches in literature, this is done by an own component. However, in QualiMaster, the Monitoring Layer mainly acts as an aggregator and overall control mechanism for monitoring, while monitoring actually happens in the Execution Layer as indicated above. Thus, we realize the analysis tasks in the Monitoring Layer, in particular the regular analysis of SLA constraints defined in the Configuration and the pipeline quality analysis (REQ-A-12). This is due to the fact that we aim at minimizing the communication among the layers of the QualiMaster infrastructure so that the infrastructure components communicate only if actually needed. Therefore, upon startup of a pipeline, these constraints are loaded into the Monitoring Layer and inform the Adaptation Manager accordingly, e.g., in case of SLA constraint violations.

Determining a **plan** for the adaptation based on the actual situation is the next step in the MAPE-K cycle. This is handled by the Adaptation Layer of the QualiMaster infrastructure, which becomes active on certain *adaptation triggers*. Currently, we foresee the following adaptation triggers:

- Internal triggers, i.e., triggers caused by the QualiMaster platform itself, such as
    - SLA constraint violation detected by the Monitoring Layer.
    - Regular adaptation schedule, i.e., an internal trigger that is issued regularly based on the configured value of `updateFrequency` (cf. Section 0). This trigger causes either an adaptation or, in case that no enactment is needed, it may cause an update of the common knowledge.
    - Changes to the Configuration Model without restarting the pipelines, such as SLA changes, modifications of the resource pool, etc.
    - Errors in the Execution Layer, e.g., caused by data processing or by dynamic changes at runtime. If specified in the adaptation behavior, this may also lead to some form of recovery to the most recent successful runtime configuration.
    - Administrative events, such as startup or shutdown of a pipeline or the whole infrastructure.
- External triggers caused by data or the user, such as
    - External events, e.g., due to regular domain-specific calendars such as the announcements of the central banks in the financial domain detected from Web data.
    - Unforeseen emerging events detected from Web data.
    - User triggers issued by the QualiMaster applications. Please note that these triggers shall typically be considered of a lower priority than the other triggers.

Upon an adaptation trigger, the Adaptation Layer executes the actual Adaptation Behavior Specification, i.e., it freezes the actual quality properties temporarily for this particular execution and (dynamically) selects the adaptation strategy to be executed. Ultimately, the *adaptation strategy* determines the actions causing runtime changes in the QualiMaster platform. Thereby, the strategy causes changes in the runtime configuration, which require a validation based on the (runtime) configuration constraints, and, finally, leads to changes in the QualiMaster platform. Adaptation strategies (called “adaptation rules” in D1.2) are defined by the Adaptation Manager at configuration time and executed at runtime. However, further adaptation strategies may be added at runtime, e.g., as a form of enactment or due to the reflective adaptation.

Finally, the adaptation decision leads to the **enactment** (**execution**) of the runtime configuration. Therefore, the changed runtime decision variables are turned into commands for the Coordination

Layer based on the Adaptation Behavior Specification. In case of multiple commands to be executed in one step, the Coordination Layer executes them in a transactional manner, taking care of the overall consistency of the Execution Systems (as described in D5.1). The Coordination Layer records such a transaction via the Data Management Layer in the adaptation log. This allows determining the impact of the adaptation decision as, e.g., and to dynamically select adaptation strategies as, e.g., done in [73].

All adaptation steps rely on **common adaptation knowledge**. Basically, this knowledge consists of the runtime configuration including the runtime and the pre-runtime settings, which were used for instantiating the running platform and pipelines. Due to the topological configuration of the pipelines, important customizable parts of the QualiMaster architecture are already reflected in the configuration. As indicated above, the runtime variables represent the actual state of the QualiMaster infrastructure relevant for adaptation at runtime. The common (runtime) knowledge is further complemented by an architectural model, i.e., the components that can be affected at runtime (such as events representing the triggers and the coordination commands) as well as the information stored in the Data Management Layer. For this purpose, the Data Management Layer contains tradeoff information among the individual algorithms (either collected at runtime or provided through the extension package manifests), impact information on the enactment of adaptation decisions, an adaptation log, etc.

8.2 Enactment Patterns

Enactment of adaptation decisions can happen in different parts of the QualiMaster platform as well as in different forms. In this section, we discuss enactment patterns capturing the potential enactments. The patterns have been identified by a systematic analysis of the QualiMaster use cases (D1.2), the QualiMaster infrastructure (D5.1) and the data stream processing literature (cf. Section 4.1) following the overall goal to adapt the data processing by **avoiding / minimizing the impact on the real-time properties** of the processed data streams (Challenge C7).

The patterns we describe in this section are intended as a research plan for the first months of the second period of the project, i.e., they need to be analyzed, evaluated and, in case of successful evaluation, implemented through the adaptation specification. We plan to perform intensive experiments with the identified patterns to analyze their runtime behavior, their impact on the quality as well as their practical feasibility. This may lead to the exclusion of individual patterns, the refinement of existing patterns, but also the inclusion of new patterns. Although we discuss in this section patterns for the entire infrastructure, we focus in particular on the adaptation of the real-time data stream processing, i.e., the data analysis pipelines.

In Section 8.2.1, we introduce the structure we use for describing enactment patterns. Then we use the pattern structure to describe two main types of patterns, namely

- **Primary enactment patterns** (Section 8.2.2) intended to change the data processing as requested by the adaptation strategies.
- **Secondary enactment patterns** (Section 8.2.3) that can be combined with the primary pattern, i.e., optionally support the primary enactment patterns. Secondary patterns are intended to be executed explicitly by an adaptation strategy or applied implicitly by the Execution Layer.

In Section 8.2.3, we illustrate the applications of some patterns in terms of an example.

8.2.1 Pattern Structure

In this section, we describe the structure we use for presenting and discussing the enactment patterns we identified for the QualiMaster infrastructure. As indicated above, future work may change the pattern set and, if required, also the pattern structure.

Relying on a pattern structure is a well-established approach for reporting on software development knowledge. Originally, this was applied to describe design patterns [57]. In recent time, patterns have also been used to describe software architectures [23], programming approaches in terms of idioms [23], or variability instantiation patterns [33, 43]. Thus, the top-level elements of the structure are rather standard. They include:

- **Identification:** The identification of the pattern.
    - **Name:** Each pattern receives a meaningful name.
    - **Identifier:** Along with the name, each pattern receives a unique identifier in order to enable references, also for future deliverables.
- **Purpose:** A short description of the situation to be solved by the pattern.
- **Context:** The context restricts the situations in which the pattern is applicable.
    - **Execution System:** This details the QualiMaster Execution Systems (cf. D5.1) the pattern applies to, i.e., software-based or hardware-based execution (possibly qualified by "stream processing" or "batch").
    - **Synergies / Overlaps:** We record whether (we expect that) this pattern has synergies or overlaps with other patterns, e.g., whether a pattern is (technically) subsumed by another pattern.
- **Approach:** The approach the pattern takes to adapt the QualiMaster infrastructure at runtime.
    - **Key idea:** This describes the key idea how the enactment happens as well as similarities identified in literature.
    - **Advantages / Disadvantages:** A brief discussion of expected benefits or already identified issues, possibly along with alternative realization options.

8.2.2 Primary Enactment Patterns

In this section, we describe those patterns that are intended to change the data processing, in particular the real-time data stream processing in the QualiMaster infrastructure. The patterns are (without aiming at a specific sequence):

- Change functional parameter of algorithm (EP-1)
- Select algorithm from processing family (EP-2)
- Perform load shedding (EP-3)
- Switch between software- and hardware-based processing (EP-4)
- Parallelize processing element (EP-5)
- Migrate the execution of processing tasks (EP-6)

We now discuss the individual patterns:

EP-1 Change functional parameter of algorithm.

Purpose: Affect and optimize the behavior and the qualities of an algorithm without switching to another algorithm in the same algorithmic family.

Context:

- **Execution System:** Software-based or hardware-based execution.
- **Synergies / Overlaps:** This pattern can be used to determine an approximate start value to reduce the startup time and to optimize the settling behavior of an algorithm, possibly affecting the internal state of the algorithm (ES-11). Further, this pattern can be realized by turning variable parameters into constants in order to optimize the runtime performance, e.g., though instantiation of algorithms at runtime (ES-4). The application of this pattern may be combined with switching between algorithms in the same family (EP-4), in particular between hardware and software, but also with a temporary parallelization in order to reduce the settling time (EP-5, ES-5). Changing functional parameters is used in some existing approaches, such as [87, 93, 122].

Approach:

- **Key idea:** The available functional parameters are specified as runtime variables in the configuration (see Section 7.2.5 and 7.2.6). These runtime variables can be changed by the adaptation, checked by the runtime reasoning, and, finally, sent as a functional parameter modification command by the Adaptation layer to the Coordination Layer. The Coordination Layer forwards the command to the responsible Storm Bolt, which calls the respective method of the algorithm family interface. This leads to a change of the parameter implementation, either directly in a single Java realization, via network in a hardware-based implementation or through the implemented means of a topology based algorithm.
- **Advantages / Disadvantages:** Typically, changing an external parameter in an algorithm can be done very efficiently. The algorithm receiving the new parameter may change the behavior, its quality parameters (in particular time-behavior and functional suitability, cf. Section 6.2), but also cause a certain instability of the results for a certain (settling) time.

EP-2 Select algorithm from processing family.

Purpose: Select the optimal algorithm according to the current situation.

Context:

- **Execution System:** Software-based or hardware-based execution (stream processing).
- **Synergies / Overlaps:** May require switching to hardware (EP-4), a change of the topology in case of topology-based algorithms (ES-8), a transfer of the internal state of the algorithm if possible (ES-11) or running algorithms in parallel (ES-5) in order to optimize the settling behavior. Switching among topologies can be realized by stream re-routing (ES-8) and may require explicit synchronization (ES-12). Akin to EP-1, also temporary parallelization can be applied to reduce the settling time (EP-5, ES-5).

  Changing a whole query plan (consisting of generic data stream operators) at runtime in order to optimize the quality is for example done in [122], the adaptation of general processing flows in [61] and the automated composition of processing flows based on flow patterns in [115].

Approach:

- **Key idea:** Algorithm families are groups of algorithms with similar functionality, but different quality tradeoffs. Switching the actual algorithm used for processing can improve, but also decrease quality properties, potentially leading to quality impacts in a whole data analysis pipeline. The available algorithms are defined in the configuration. The algorithm family refers to the actual algorithm as a runtime decision variable. Akin to EP-1, changing the actual processing algorithm based on the executed adaptation strategies leads to a change of the runtime configuration, a runtime validation and, ultimately, to the respective algorithm switch command sent by the Adaptation Layer to the Coordination Layer. The coordination layer forwards the command to the right Storm Bolt using a Storm signal. The respective Bolt receives the signal and changes the active algorithm accordingly, possibly executing further secondary patterns based on further command send to the Coordination Layer.
- **Advantages / Disadvantages:** The new algorithm may affect the quality parameters of the entire pipeline (in particular time-behavior and functional suitability, cf. Section 6.2), but also cause a certain instability of the results for a certain (settling) time.

EP-3 Perform load shedding.

Purpose: Reduce the data stream volume / load, in particular in cases of extreme load due to an increasing volume of the data streams.

Context:

- **Execution System:** Software-based or hardware-based execution (stream processing).
- **Synergies / Overlaps:** Load shedding can be enabled or controlled by a functional parameter (EP-1), a bypass of a load shedding algorithm (family) designed into the pipeline (ES-2), a bypass of an internal load shedding implementation in a Bolt (ES-2) or by changing the processing pipeline (ES-8).

Approach:

- **Key idea:** Exclude items from a data stream before they consume (additional) resources, as performed for example in [1, 25, 136, 145] and discussed in D2.1. As analyzed in literature, several positions for load shedding in data processing can be considered with different impact on quality parameters as well as on the processing result. Static load shedding is defined once based on a static analysis, while dynamic load shedding can be changed on demand, e.g., through a functional parameter. Dropping items is the simplest approach to load shedding, e.g., to discard items in a random fashion. Filtering is a semantic, but also more resource-consuming way of reducing the overall load due to a high data stream volume.
- **Advantages / Disadvantages:** Discarding items from a data stream may affect the accuracy of the analysis. However, it is an approach to avoid overload in extreme situations, in particular if applied at (or shortly after) the pipeline data source.

EP-4 Switch between software- and hardware-based processing.

Purpose: Improve the performance by exploiting the parallel stream processing capabilities of reconfigurable hardware.

Context:

- **Execution System:** Hardware-based execution (collaborating with software-based execution).
- **Synergies / Overlaps:** This pattern can be considered as a special case of EP-2, i.e., synergies and overlaps apply as discussed above. As hardware-based processing involves different types of hardware, stream re-routing (ES-8) is required upon switching. Further, switching to a hardware-based algorithm can require reprogramming of the reconfigurable hardware, i.e., uploading the correct algorithm to the DFEs (ES-2), potentially unloading an existing algorithm. Further, buffering (ES-1) or explicit synchronization (ES-12) may be required.

Approach:

- **Key idea:** Switching among software-based and hardware-based processing can be realized akin to EP-2, i.e., upon the decision of an adaptation strategy, an algorithm switch command is sent by the Adaptation Layer to the Coordination Layer. This may lead to a change of the actual algorithm uploaded to the reconfigurable hardware. Finally, the command leads to a Storm signal message received by the respective Bolt. As prepared by the platform instantiation (cf. D5.2), the receiving Bolt can switch among software-based and hardware-based processing, by enabling the respective family member, in the hardware case the generated hardware connector.
- **Advantages / Disadvantages:** Switching of algorithms may cause quality changes as well as temporarily instable results through an algorithm-specific settling time. This enactment pattern requires stream re-routing. In particular, switching between software-based and hardware-based processing may increase the switching time (cf. Section 6.2.1.1) due potential reprogramming of the reconfigurable hardware.

EP-5 Parallelize processing element.

Purpose: Utilize additional resources for parallel processing.

Context:

- **Execution System:** Software-based or hardware-based execution.
- **Synergies / Overlaps:** This pattern can be realized as a change of the physical processing topology (ES-8), e.g., as a special (implicit) case of EP-2 as actually done for hardware processing. Migration of processing tasks to other resources of the cluster may be required (EP-6).

Approach:

- **Key idea:** Use parallel stream-based processing to achieve better results by allocating more compute resources, potentially also on different physical machines. Please note that parallelizing a processing element is different from threading or spawning a further process (ES-5), as this happens on pipeline / topology level, i.e., data stream re-routing and dynamic data distribution (e.g., using the shuffling strategy in Storm) is required.
- **Advantages / Disadvantages:** Applying this pattern requires additional compute resources and may increase the complexity of pipeline processing. Further, its realization depends on the actual capabilities of changing the physical processing topology or redistributing resources / migrating execution at runtime.

EP-6 Migrate the execution of analysis tasks.

Purpose: Perform the computation on additional or alternative resources.

Context:

- **Execution System:** Software-based or hardware-based execution.
- **Synergies / Overlaps:** This pattern can be understood as the generic super-pattern of changing algorithms across resources (EP-2), switching to hardware (EP-4) or parallelizing processing elements (EP-5). In addition, this pattern enables the use of alternative execution systems or further resources for software-based processing. We kept the individual patterns in order to analyze and discuss them separately. Migrating the execution may require the insertion or removal of data buffers (ES-1)

Approach:

- **Key idea:** Physically shift the processing within the cluster or even to external resources, i.e., resources outside the QualiMaster computer cluster. An Example for an external resource is a (private, public, hybrid) cloud if data can be processed there, e.g. due to legal or privacy issues. This can be done by rebalancing the distributed processing in Storm, but also by considering alternative Execution Systems in the Execution Layer. One example for alternative Execution Systems are Hadoop[15] and Spark[16],. Typically, Spark requires more memory resources (resource utilization), but offers a better processing performance (time-behavior), while Hadoop is more resource efficient, but also slower in processing.
- **Advantages / Disadvantages:** Migration of tasks may require that respective resources are freed or even powered up. Depending on the available resources, alternative execution systems may (at least temporarily) run in parallel. This pattern may significantly increase the switching time (time-behavior) and lead to temporary instability of the analysis results. Switching to external systems may increase the latency, but also provide nearly unlimited resources, thus, enable processing even in extreme load situations while, potentially, sacrificing time-behavior quality (at least temporarily). Migrating processing tasks in Storm is supported by rebalancing a running topology, which actually causes an interruption of data processing. In contrast, dynamically migrating tasks including alternative execution systems can lead to an overall balanced processing, but requires adequate switching capabilities.

[15] http://hadoop.apache.org/
[16] https://spark.apache.org/

Currently, the platform instantiation process (cf. D5.2) realizes the following enactment patterns for a generated Storm topology: Change functional parameter (EP-1), Switch algorithm (EP-2), and switch to / from hardware (EP-4). Further enactment patterns will be considered after a thorough analysis as indicated in Section 8.2.

8.2.3 Secondary Enactment Patterns

In this section, we describe additional (sub-)patterns, which can optionally be used to support or optimize the primary enactment patterns. In particular, multiple secondary patterns may be combined with a primary pattern. The patterns are (without aiming at a specific sequence):

- Data buffering (ES-1)
- Reprogram reconfigurable hardware (ES-2)
- Bypass processing element (ES-3)
- Instantiate algorithm (ES-4)
- Execute tasks in parallel (ES-5)
- Adapt monitoring (ES-6)
- Bind / unbind algorithm (ES-7)
- Re-route streams (ES-8)
- Change Storm topology (ES-9)
- Use less precise algorithm for settling results (ES-10)
- Perform state transfer (ES-11)
- Synchronize enactment and data processing (ES-12)

We discuss now the individual patterns:

ES-1 Data buffering.

Purpose: Bridge gaps cause by dynamic enactments to avoid accidental loss of data or local overload.

Context:

- **Execution System:** Software-based or hardware-based execution (stream processing).
- **Synergies / Overlaps:** Data buffers can, e.g., be used while algorithm switching (EP-4), switching to hardware (EP-4) or execution migration (EP-6).

Approach:

- **Key idea:** Insert a data buffer, e.g., on the input side of a Bolt to temporarily store data items, defer processing, synchronize processing and enactment or avoid data loss. On the one side, switching and settling behavior of individual algorithms determine whether a data buffer is needed, e.g., through threshold values for the respective quality parameters. Based on actual quality parameters, the Adaptation Layer can decide about the need for a buffer, send a respective command to the Execution Layer and cause the dynamic insertion / removal of a data buffer in the respective Bolt. The buffer size may be controlled dynamically, e.g., based on actual quality parameters, either implicitly in the Bolt, or explicitly through the Coordination Layer. Please note that configuring a data buffer at runtime does not happen through functional parameters, but implicitly through mechanisms of the QualiMaster infrastructure.
- **Advantages / Disadvantages:** A data buffer may temporarily affect the real-time properties of the data stream (latency), but enable synchronization or avoid data loss.

ES-2 Reprogram reconfigurable hardware.

Purpose: Provide reconfigurable hardware with the algorithms to be executed.

Context:

- **Execution System:** Hardware-based execution.
- **Synergies / Overlaps:** Reprogramming hardware-based algorithm may require the insertion or removal of data buffers (ES-1), bypassing software-based algorithms for switching (ES-2) or, if reconfigurable hardware must be freed, also switching back to software-based processing (EP-4).

Approach:

- **Key idea:** For executing a hardware-based algorithm, the underlying reconfigurable hardware must be reprogrammed, i.e., the compiled and laid-out algorithm must be uploaded to the DFE, in Maxeler environments through the Maxeler OS. This requires free DFE resources, i.e., either a DFE is available and can be used or a decision must be made whether the actual use of a DFE is not efficient in the current situation. The latter requires a switch back to software-based processing (EP-4).
- **Advantages / Disadvantages:** Reprogramming reconfigurable hardware with a new algorithm typically may take up to seconds in which the related DFE is not available.

ES-3 Bypass processing element.

Purpose: Optimize quality parameters by simplifying the data processing or enable explicit alternatives in data processing.

Context:

- **Execution System:** Software-based or hardware-based execution (stream processing).
- **Synergies / Overlaps:** Bypassing a processing element can be seen as a special case of changing pipeline topology (ES-8).

Approach:

- **Key idea:** Basically, bypassing a processing element realizes an optional algorithm, either explicitly or implicitly. An explicit optional algorithm is stated in the in the pipeline specification by a pipeline constraint. Implicit bypassing happens through the enactment of adaptive decisions, i.e., built-in without explicit specification by the user. On the one side, this may lead to a change of the pipeline topology, i.e., to exclude the processing element from processing and to avoid the execution of adaptation-related code. On the other side, this can also be realized within a Bolt, i.e., to accept input items and to directly bypass them.
- **Advantages / Disadvantages:** Bypassing algorithms can avoid unnecessary processing and may optimize the time-behavior such as latency. Depending on the actual implementation of the bypass (pipeline level, Bolt level), this may also require global changes to the processing (if not already prepared by the platform instantiation or through ES-8).

ES-4 Instantiate algorithm.

Purpose: Optimize the time-behavior of software-based algorithms through the creation of variants in which functional parameters are replaced by constants.

Context:

- **Execution System:** Software-based execution.
- **Synergies / Overlaps:** A static form of setting functional parameters (EP-1), which may be realized through binding / unbinding the implementation (ES-7). May require

data buffering (ES-1) during implicit switch (EP-2) of an algorithm to its dynamically created variant.

Approach:

- **Key idea:** Functional parameters can be used to enable or disable parts of the algorithm. Instead of checking parameter-dependent alternatives or calculating parameter-dependent values for every data item, constant parameters can improve the time-behavior of the execution, as unused alternatives can be removed by the compiler [125] or similar mechanisms. Replacing variable parameters by constants is a traditional variability implementation technique in software product lines [105, 135]. In contrast, replacing functional modules at runtime is an approach to realize variability in service-based systems [43, 33]. This pattern combines both approaches. Therefore, the algorithm code is processed, parameters are replaced by constants, the code is optimized, and loaded at runtime into the QualiMaster platform as a (runtime) member into the respective algorithm family. This causes a dynamic change of the configuration model.
- **Advantages / Disadvantages:** Depending on the actual functional parameter, replacing a parameter by a constant may influence the quality parameters akin to setting a usual functional parameter (EP-1). Instantiating an algorithm at runtime is a non-trivial technical task. However, for hardware-based algorithms this pattern does not lead to significant runtime improvements according to the experience of Maxeler with hardware algorithms relying on several hundreds of parameters.

ES-5 Execute tasks in parallel.

Purpose: Execute tasks in parallel on the same machine in order to improve overall quality parameters and optimize the switching time and settling behavior.

Context:

- **Execution System:** Software-based or hardware-based execution.
- **Synergies / Overlaps:** (EP-5) may rely on this pattern for realization. Variant algorithms for parallelization may be created by instantiation at runtime (ES-4).

Approach:

- **Key idea:** Run two algorithms, either the same, members of the same family or variants due to (ES-4), in parallel within a processing element. On the one side, running algorithms in parallel, e.g., by using multiple physical CPUs, and internally distributing the input items can optimize the processing quality. On the other side, running algorithms in parallel on the same data can help to optimize the switching time and the settling behavior, e.g., switching to the new algorithm after it became stable while working on the same data. In addition, more complex adaptation actions can be executed in parallel in order to avoid impacts on the data processing.
- **Advantages / Disadvantages:** On the one side, this pattern can be used to exploit further internal resources, such as CPUs on the same physical machine as well as to optimize the switching / settling behavior. On the other side, running algorithms in parallel consumes also more resources and may need explicit synchronization of processing and data items.

ES-6 Adapt monitoring.

Purpose: Reduce monitoring overhead by dynamically enabling or disabling monitoring or by changing the monitoring frequency.

Context:

- **Execution System:** Software-based execution.
- **Synergies / Overlaps:** Changing instrumented code can be seen as a specific form of instantiating an algorithm with respect to monitoring probes (ES-4).

Approach:

- **Key idea:** Typically, monitoring happens in addition to the normal operations and, thus, causes overhead. Avoiding monitoring overhead while obtaining the relevant information is an important goal in software measurement (as discussed, e.g., in [46]). In stable environments, i.e., quality parameters are considered as sufficiently confident and stable, this overhead can be reduced by disabling monitoring or changing the monitoring frequency if applicable. This can be achieved by disabling the entire monitoring or, on a different level of granularity, disabling monitoring probes for individual observables so that further processing of raw monitoring information does not happen. In case of frequently used (expensive) monitoring probes (such as for memory allocation in Java), a dynamic re-instrumentation of the code can help saving more resources.
- **Advantages / Disadvantages:** May temporarily increase the resource usage, affect quality parameters if re-instrumentation is required and affect quality parameters in case that monitoring is not performed if actually needed.

ES-7 Bind / unbind algorithm.

Purpose: Dynamically load needed and unload unused algorithms and their dependencies to save resources.

Context:

- **Execution System:** Software-based execution.
- **Synergies / Overlaps:** Depending on the actual implementation, this pattern may support the switching of algorithms (EP-4) as well as the dynamic instantiation of algorithms (ES-4).

Approach:

- **Key idea:** Realize algorithms in terms of modules that make their dependencies explicit and use a module system that allows dynamic loading / unloading of components such as OSGi[17] to optimize resource utilization.
- **Advantages / Disadvantages:** As only running algorithms and their dependencies are kept in memory, less resources, in particular memory are consumed. However, explicit loading and unloading of algorithms may increase the switching time, lead to temporarily increased resource.

ES-8 Re-route streams.

Purpose: Switch among algorithms by changing the respective input / output streams.

Context:

- **Execution System:** Software-based or hardware-based execution.
- **Synergies / Overlaps:** Supports the realization of switching algorithms (EP-4), switching among software- and hardware-based execution (EP-4) as well as migration of analysis tasks (EP-6). Data buffers (ES-1) may be used in combination.

  Re-routing of data items is for example utilized in [1, 122].

Approach:

- **Key idea:** Redirect a data stream among alternative implementations or Execution Systems, e.g., from software-based processing in Storm to hardware-based processing on Maxeler DFEs.
- **Advantages / Disadvantages:** Re-routing a data stream may cause a gap in data processing so that data is accidentally lost. While Storm provides mechanisms for

[17] http://www.osgi.org/Main/HomePage

properly switching / filtering streams, switching within a Bolt, e.g., to reconfigurable hardware or a sub-topology by a network connection may require buffering.

ES-9 Change Storm topology.

Purpose: Adjust the pipeline implementation to cope with a changed environment.

Context:

- **Execution System:** Software-based execution (stream processing).
- **Synergies / Overlaps:** Bypass processing elements (ES-2) or parallelize processing elements (EP-5) can be seen as a special case of changing a Storm topology. Data buffers (ES-1) as well as stream re-routing (ES-8) can support the realization.

Approach:

- **Key idea:** Changing the Storm topology of a data processing pipeline enables additional dimensions of flexibility, in particular to adjust the processing and to control resource utilization. A replacement topology can be obtained by creating a respective pipeline configuration and executing the QualiMaster platform instantiation process for that specific pipeline.
- **Advantages / Disadvantages:** Changing a topology is a technological challenge as it changes a laid-out and allocated distributed processing topology. Currently, Storm does not provide support for this enactment pattern. However, there are several alternatives to enable dynamic changes of a Storm topology:
    - Change the topology by stopping the running one, i.e., disconnect the streams, buffer the data, stop the running topology, start the replacement topology and reconnect the streams. However, this causes an interruption of the data processing, and, thus taints the real-time properties of the data streams.
    - Start the replacement topology in parallel and connect the input streams also to the replacement topology. Keep both topologies running until the results of the replacement topology are stable, switch then the data streams from the data sinks to the applications, disconnect the input of the old topology and shut down the old topology. This approach requires that the resources for running the replacement topology are available, i.e., value-added processing may need to be temporarily disabled. Applying this approach will cause a significant peak resource usage, but does not cause (significant) impact on the data streams.
    - Spilt up the (logical) data processing pipeline into parts, which are executed as individual sub-topologies (not to be confused by sub-topologies realizing alternative algorithms). Then, an individual sub-topology can be replaced akin to a replacement topology, but with less impact on the overall infrastructure, i.e., running the replacement sub-topology it in parallel until the results are stable and then re-routing the streams accordingly. Although this approach may reduce flexibility and strategies for splitting a topology are needed, the peak resource usage shall be less impacting.
    - Lay out an augmented topology, i.e., a topology with potential alternatives, in the extreme case a kind of maximum topology. Although this allows switching among (anticipated) alternatives, it leads to increased overall resource utilization, in particular as several alternative paths and the occupied resources may actually not be used at all.

ES-10 Use less precise algorithm for settling results.

Purpose: Use an alternative, less precise but also less resource consumptive algorithm to reduce switching time and improve settling behavior.

Context:

- **Execution System:** Software-based or hardware-based execution (stream processing).
- **Synergies / Overlaps:** Parallelize processing element (EP-5) as well as changing the pipeline topology (ES-8) can support this pattern. Switching algorithms (EP-2, EP-4) are more complex as the less precise algorithm must be considered for switching / setup. State transfer (ES-11) can be an alternative to this pattern.

Approach:

- **Key idea:** Run a less precise algorithm such as summarization or sketching in parallel to the normal data processing and switch first to that algorithm until the actual alternative algorithm becomes stable and then to the alternative algorithm.
- **Advantages / Disadvantages:** The switching time may be reduced as the change can happen immediately and, depending on quality requirements, the “real” alternative algorithm may take over faster. However, running an additional algorithm in parallel also consumes more resources.

ES-11 State transfer.

Purpose: Reuse the internal state of the running algorithm in order to reduce switching time and improve settling behavior.

Context:

- **Execution System:** Software-based or hardware-based execution.
- **Synergies / Overlaps:** This pattern can support the switching of algorithms (EP-4), but probably not across software- and hardware-based execution. Further, this pattern is an alternative to ES-10, i.e., running a less-precise algorithm in parallel.

Approach:

- **Key idea:** State-based algorithms require some time to build up a state and to produce stable results. Upon switching algorithms, the internal state of an algorithm is lost. The replacing algorithm needs to build up its internal state before results become stable. If applicable, the internal state of an algorithm can be transferred to the alternative algorithm while switching algorithms.
- **Advantages / Disadvantages:** This pattern can reduce the startup time and improve the settling behavior, but requires algorithms with compatible internal states. The realization may need to follow some form of component reconfiguration protocol [20].

ES-12 Synchronize enactment and data processing.

Purpose: Synchronize data processing and enactment to avoid inconsistent processing state or invalid processing results.

Context:

- **Execution System:** Software-based or hardware-based execution.
- **Synergies / Overlaps:** This pattern must be taken into account for all primary enactment patterns (EP-1, EP-2, EP-3, EP-4, EP-5, and EP-6).

Approach:

- **Key idea:** Enactment signals and data stream processing may occur in parallel, i.e., immediate changes caused by the enactment may disturb data processing as, e.g., the actual algorithm or parameters are changed during processing. Synchronization problems may also occur if enactment must happen at two different points of data processing, e.g., for a Bolt switching between data streams and further Bolts realizing the data processing, i.e., two Storm signals must be synchronized. This can be handled by deferred execution of the enactment in safe states of data processing as a kind of (implicit) reconfiguration protocol [20].

- **Advantages / Disadvantages:** Keeping enactment and processing in a synchronized state helps ensuring the validity of the data processing results. Maintaining synchronization may imply more complicated data processing code due to explicit synchronization or exclude certain realization alternatives, e.g., to avoid multiple signals needed for enactment and defer the actual enactment until a current data processing task is finished.

Currently, the platform instantiation process (cf. D5.2) realizes the following enactment patterns for a generated Storm topology: Bypass processing element (ES-3) in case of switching to / from a sub-topology and re-route streams (ES-8) for, both sub-topologies and switching to / from hardware-based processing. Further, the QualiMaster infrastructure realizes the reprogramming of reconfigurable hardware (ES-2), in particular during startup of the instantiated QualiMaster platform. Further secondary enactment patterns will be considered after a thorough analysis as indicated in Section 8.2.

8.2.4 Application of Enactment Patterns

In this section, we describe an example illustrating how enactment patterns can be combined to realize adaptation. In general, one adaptation strategy may trigger (explicitly or implicitly) several enactment patterns. Regarding the implementation, we rely in our argumentation on the topology generated by the QualiMaster platform instantiation process (cf. D5.2).

Let us consider the case that a software algorithm shall be switched at runtime. Basically, this can be done by changing the actual instance of the family interface in a Bolt, i.e., by overriding the running algorithm with the one specified in the received adaptation signal. However, the signal may be received in parallel to the actual processing, i.e., directly changing the algorithm may impact the result stream or data items may be lost. A programmatic approach could be to avoid mutual reentrant execution of methods of a Bolt, i.e., to enforce that either data processing or signal handling is performed. In total, this may affect the latency of the produced data stream. An alternative is to schedule the new algorithm for replacement (instead of doing the replacement directly, an implicit form of ES-12), and to perform the replacement after the actual algorithm processed its (current) item. If the Bolt supports dynamic wiring / unwiring (ES-7), the replacement algorithm must first be loaded, then scheduled for replacement and after replacement the old algorithm may be unloaded. Here, loading of the algorithm and scheduling the replacement as well as unloading the algorithm shall be executed in parallel tasks (ES-5) in order to avoid impacts on the real-time streams. If the actual exchange of the algorithms cannot be done synchronously with data stream processing, buffering of input items (ES-1) is required in order to avoid data loss. In case that the replacement algorithm is state-based and building up the state affects the quality of the processing, state transfer (ES-11), a less precise algorithm for improving the settling behavior (ES-10) or parallelizing the replacement algorithm (ES-5) with deferred scheduling of the replacement can be applied until the results become stable.

8.3 Flexible Adaptation Behavior Specification

In this section, we detail our concepts for specifying the adaptive behavior in the QualiMaster infrastructure and how we approach the unification of configuration and adaptation (Challenge C1). An initial overview on our approach is also given in [47]. This section details the term “adaptation rule”, which we used generically in D1.2. In particular, we aim at a flexible approach (Challenge C5) that allows us to

- Structure and define the adaptation behavior in terms of quality parameters, tradeoffs, triggers / events and enactments (Challenge C6).
- Adjust the adaptation behavior during the project in order to reflect the most recent knowledge on the adaptation of real-time data processing pipelines and the actual needs of the financial use cases. This allows us to support further quality properties if actually needed.

- Change the adaptation behavior in order to meet the needs of different applications (in finance) or other domains.
- Specify reactive and proactive adaptation and even to support a change of the adaptation behavior as a result of reflective adaptation, i.e., the work on specifying reactive and proactive adaptation provides a framework for realizing reflective adaptation.

In Section 8.3.1, we outline the basic concepts of our approach and its inspiration sources in literature. In Section 8.3.2, we discuss our language-based specification of the adaptive behavior in detail. Please note that the grammar of the proposed language is displayed in Appendix B.

8.3.1 Concept Overview

As discussed in Section 4.3, several approaches described in literature support the specification of adaptive behavior, ranging from rule-based over utility-based to formal approaches. After a thorough study of the literature (as indicated in Section 4.3), we decided to base the QualiMaster adaptation approach on the Stitch language [28] and its recent extension S/T/A [73]. We selected Stitch due to its practical approach, in particular it is designed to enable administrators to define and maintain the adaptive behavior of complex software systems. Stitch was experimentally evaluated in administrative contexts and applied to support administrators in maintaining infrastructures. This practical approach is an advantage over formal approaches, which typically need solid background knowledge in formal methods [77]. Another reason for selecting Stitch is due to its structured approach, i.e., in contrast to (flat) rule-based approaches, it leads to a hierarchical structure supporting design as well as maintenance of an adaptation specification. Stitch consists of

- **Strategies**, which represent high-level adaptation objectives capturing the logical aspect of planning an adaptation. A strategy distinguishes (potentially alternative) ideas on how to accomplish an adaptation in a certain setting from the low-level implementation details such as the actual enactment. For a system, multiple strategies can be specified, but only applicable strategies are considered (based on the system state and the adaptation impact history) and the most appropriate strategy for the current setting is executed. Example strategies in QualiMaster include single-pipeline optimizations, cross-pipeline optimizations, specific handling of user triggers (in contrast to autonomous pipeline adaptations) or how to handle administrative changes to the resource pool.
- **Tactics**, describing the technical realization of the potential adaptations that are enabled by a strategy. When specifying the adaptation behavior, the Adaptation Manager describes how the system shall act in terms of tactics, i.e., which tactics can be employed to fulfill the strategies objective in the context of the current system state. Tactics may be guarded by conditions or tradeoffs, but also selected dynamically according to their expected impact or impact history [73]. Example tactics in QualiMaster include adapting multiple algorithms in a whole pipeline or to focus in certain situations on the change of the most critical data processing algorithm or algorithm parameter.
- **Actions**, the atomic steps that make up a tactic. Actions cause the enactments and, thus, the dynamic changes to adapt a system.

Based on these concepts, Stitch and S/T/A provide a solid basis for the QualiMaster adaptation approach. However, Stitch and S/T/A focus on architecture-based adaptation, i.e., they rely on an explicit architecture model of the system being adapted, do not take configuration information (Challenge C1) into account and do not provide specific means for adaptation in the context of a real-time data stream processing systems (Challenge C7). At a glance, architecture-based adaptation may not appear to be an ideal basis for QualiMaster, as we do not maintain an explicit architecture model. Thus, it is important to recall that we capture essential architectural information in the QualiMaster Configuration Meta Model, in particular in terms of the processing pipeline configuration as discussed in Section 7.2.7. Although this may be sufficient for adapting the QualiMaster platform, we do not focus only on the special case of topological configurations that capture parts of the (instantiated) architecture. We provide also concepts to enable (classical) architecture-based adaptation as well as a combination of architecture- and configuration-based adaptation. In QualiMaster, this enables us, e.g., to create commands for the Coordination Layer, i.e., instances of architecture components. Further, architecture-based capabilities enable adaptation for those parts of the QualiMaster platform that are currently not in the (data stream processing) focus of the QualiMaster Configuration Meta Model, such as batch processing.

Through strategies, tactics and actions, the Adaptation Manager can specify how the infrastructure shall be adapted for different quality parameter (REQ-A-2 - REQ-A-8). Access to the actual values of these parameters is provided through monitoring / analysis in terms of runtime decision variables (also on pipeline level REQ-A-12) and through the Data Management Layer in terms of quality profiles or impact histories. Please note that the QualiMaster consortium will provide adaptation specifications that can be adapted for domain-specific purposes during adoption of the QualiMaster approach.

In summary, the QualiMaster approach to the specification of adaptation behavior provides the following capabilities over Stitch and S/T/A:

- **Unification of adaptation and configuration:** Following challenge C1, we aim at unifying configuration and adaptation, i.e., at utilizing configuration information and constraints captured in the configuration for adaptation. Moreover, we aim at integrating configuration and architecture-based adaptation as described above. From the SPLE point of view, adaptation can be understood as a dynamic form of instantiation at runtime based on a consistent runtime configuration, i.e., a Dynamic Software Product Line (DSPL) [67, 71]. Subscribing to that view, we rely on and refine existing concepts from (traditional) product line instantiation for runtime instantiation and, in particular, adaptation. Thus, we use the Variability Instantiation Language VIL [36, 42, 140] as basis for the QualiMaster adaptation specification language. VIL already provides a rich and extensible set of types and related operations (corresponding to the actions in Stitch). In addition, VIL allows grouping operations to reusable functions as well as to import, reuse and extend existing functions (not supported by Stitch or S/T/A). Consequently, we call the QualiMaster adaptation specification language **Runtime Variability Instantiation Language** (**rt-VIL**). As rt-VIL is based on VIL, it enables traditional instantiation activities at runtime, e.g., turning source code variables into constants (enactment pattern ES-4) or compiling the modified code.
- **Runtime reasoning:** Also constraints defined in the Configuration Meta Model, in particular those involving runtime decision variables, must be considered during adaptation to ensure that the enactment happens on a valid runtime configuration (as already mentioned in Section 3). On the one side, runtime reasoning is used to identify violated SLA constraints and may trigger adaptation. On the other side, runtime reasoning is used to validate the runtime configuration during adaptation (focusing on the changed decision variables). Therefore, we rely on the EASy-Producer reasoning support, which is also applied in the QualiMaster Configuration Tool. Actually, automated correction of an invalid configuration is a recent research topic in SPLE [9, 149]. As a reasoning-related research challenge, we envision to work in later stages of the project on the correction of invalid runtime configuration models in order to improve the adaptation behavior. However, to ensure consistency in architecture-based adaptation, specific validation mechanisms or even a reconfiguration protocol [20] must be employed.
- **Specification of the enactment mapping:** In Stitch, actions directly manipulate the software system under adaptation. In contrast, rt-VIL manipulates the runtime configuration and validates the configuration by runtime reasoning. Finally, the changed runtime decision variables lead to the enactment of the adaptation decisions. This needs some kind of mapping between the Configuration Meta Model and the QualiMaster platform, more precisely the Coordination Layer. Therefore, rt-VIL provides an extension point that can be used to turn changed runtime decision variables into commands for the Coordination Layer. Thereby, different enactment patterns discussed in Section 8.2 can be realized. Please note that the enactment mapping is system dependent, i.e., it relies on components defined by the underlying system, here the QualiMaster infrastructure, i.e., this extension point allows us to define and use rt-VIL even independent of the QualiMaster infrastructure. For convenience, these components are mapped into the rt-VIL type system so that they can

be used for defining the adaptation and enactment behavior. In particular, this explicit mapping defines the intended sequence of the individual enactment actions. In other settings, this enactment mapping can be used to realize a reconfiguration protocol [20].

- **Support for nested structures and topologies:** Our initial paper prototyping of adaptation behavior specifications for QualiMaster using the Stitch [28, 73] concepts showed that describing the adaptation for nested structures, topologies such as pipelines and type hierarchies is difficult. Therefore, in contrast to Stitch, but aligned with VIL, we support sub-strategies as well as parameterizable strategies and tactics. Furthermore, to handle the various (refined) types defined in the QualiMaster Configuration Model (akin to typical architectures), we enable parameterizable strategies and tactics to be selected dynamically using dynamic dispatch akin to VIL rules. Dynamic dispatch is an extended form of late binding for all parameters in contrast to late binding for the first implicit parameter in many object-oriented languages such as Java or C++. We also support a programmatic enumeration of sub-strategies or tactics to cope with the openness of the Configuration Meta Model, i.e., configurations may be defined or changed after specifying the adaptation.
- **Support for event-triggered adaptation:** As outlined in the introduction of Section 8, adaptation in QualiMaster is triggered by events, such as regular schedules or SLA constraint violations. Akin to enactment commands, these event types are architecture components of the QualiMaster infrastructure, which are mapped into the rt-VIL type system and allow the selection of strategies and tactics depending on the actually causing event. As in Stitch [28] or S/T/A [73], an adaptation in rt-VIL is triggered by a single event and we assume (time-discrete) sequential processing of these events in order to avoid parallel activation of multiple strategies.
- **Adaptation of data stream processing systems:** In usual systems, adaptation can be enacted in terms of a simple sequence of actions, possibly following a reconfiguration protocol [20], e.g., to ensure that components are properly passivated before any change happens to them. In data stream processing, the system state depends on the processing of the individual data stream items. As a consequence, adapting multiple processing elements may cause an inconsistent system state. Let us consider that the adaptation specification determines that the first and the last processing element in a processing pipeline must be adapted. Enacting both modifications at the same point in time leads to a) a correct adaptation of the first processing element and b) to a potentially premature adaptation of the last processing element as the causing data item may still be in processing at the first processing element. Thus, we need to provide a mechanism to propagate the adaptation along the data processing pipeline (as an explicit form of enactment pattern ES-12). We call this form of adaptation a **wavefront adaptation**[18]. As the ability for wavefront adaptation depends on the underlying system, our approach supports wavefront adaptation through specific events sent to the Coordination Layer, which schedules an adaptation on / after arrival of the respective. Further, rt-VIL supports the realization of enactment patterns (cf. Section 8.2) for the adaptation of real-time data stream processing through the Coordination Layer. Please note that processing the adaptation specification must happen in a fast and efficient manner, but not necessarily in real-time. In contrast, the enactment strategies shall maintain the real-time properties of the adapted data stream. Both, wavefront adaptation and the realization of enactment patterns realize Challenge C7.
- **Transparent integration of hardware-based processing:** Switching among software-based and hardware-based processing as well as changing algorithm parameters happens transparently through the enactment comments of the Coordination Layer, which transforms generic enactment comments in execution system specific comments. This contributes to Challenge C2 and realizes REQ-A-11.

Please note that rt-VIL is intended to describe the overall adaptation behavior as VIL aims at the overall instantiation process of a product line. Therefore, we focus on the adaptive decision making

[18] According to our knowledge, this term is currently only used in physics, more precisely in optics.

and enable to execute complex functionality such as the calculation of a Pareto front [4, 68, 73, 86, 94, 104] through the language infrastructure, e.g., provided by the QualiMaster infrastructure through specific language extensions of rt-VIL. In Section 8.3.2, we discuss the core concepts of VIL, i.e., the basis of rt-VIL. In Section 8.3.3, we detail the language concepts of rt-VIL based on an extension of Stitch, S/T/A and VIL.

8.3.2 VIL Core Concepts

VIL serves as the base language for rt-VIL due to our aim of unifying of configuration and adaptation (Challenge C1). Thus, rt-VIL relies on the basic concepts and the language infrastructure of VIL. Please note that we actually use the reference implementation of VIL for the realization of the QualiMaster platform instantiation process as described in D5.2, i.e., the VIL concepts and infrastructure are implemented and available. The core concepts of VIL are:

- **Extensible artifact meta model:** VIL supports artifacts as a first class concept, i.e., everything that can be instantiated (modified, transformed, generated, or deleted) is regarded as an artifact. Artifacts include traditional file system elements such as files and folders as well as runtime components that enable architecture-based adaptation. The artifact meta model (or artifact model if we refer to the actual instances available during the execution of a VIL or a rt-VIL specification) describes the operations that can be performed on certain types of artifacts. Depending on the installed extensions, artifacts may provide access to their internal structure as well as specialized operations, as e.g., for XML files or Java source files.
- **Extensible type system:** VIL is a typed language based on an extensible and dynamic type system. The artifact meta model is an important part of the VIL type system illustrated in Figure 16. In addition, the type system also contains primitive types (`String`, `Real`, `Integer` or `Boolean`), container types (`Map`, `Set` or `Sequence`) as well as configuration-related types to access (in rt-VIL also to modify) the underlying IVML configuration or the Configuration Meta Model. The actual version of the VIL type system consists of of more than 30 types defining more than 180 operations. The VIL type system is extensible, in particular in terms of additional artifact types. Also types defined in an IVML model can be mapped into VIL in order to simply the specification of instantiation processes. rt-VIL

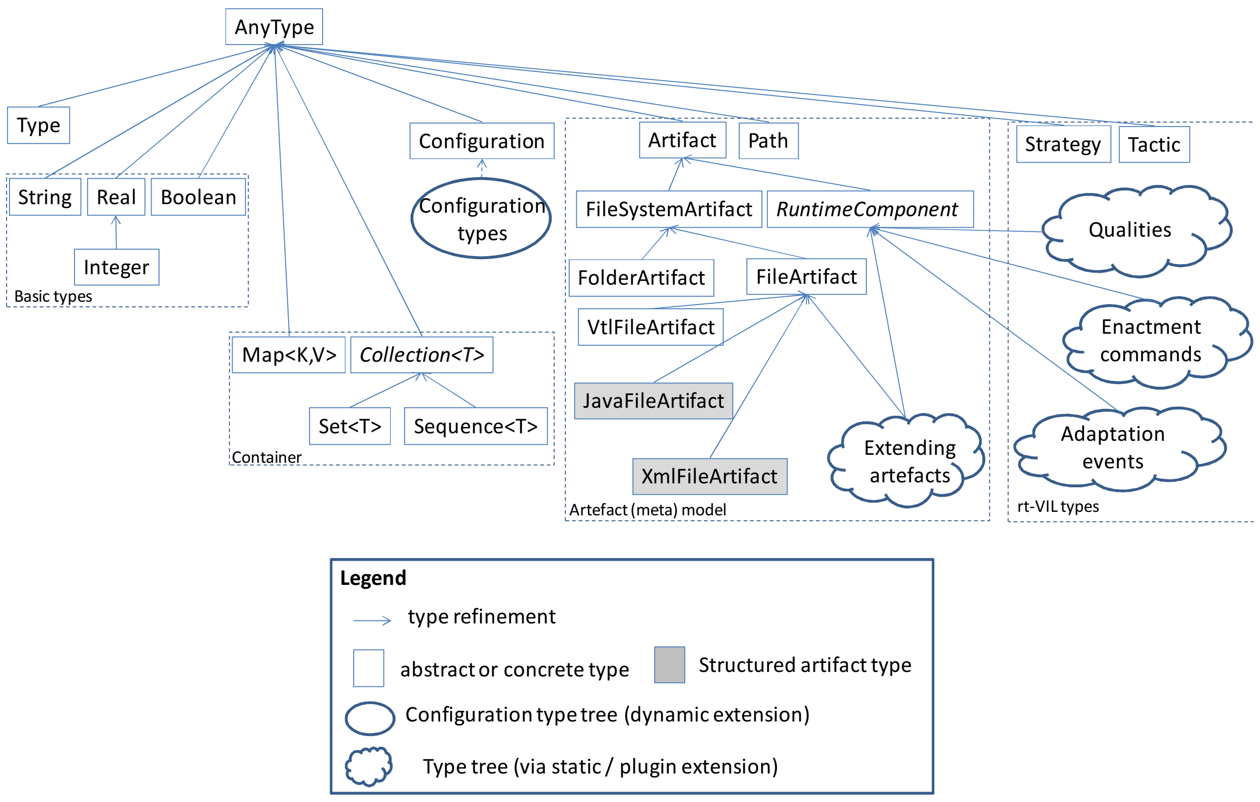

Figure 16:VIL / rt-VIL type system overview.

extends the basic VIL type system in terms of runtime components such as adaptation events, enactment commands and quality descriptors.

- **Instantiation process:** The goal of VIL is to describe the instantiation process, i.e., to define the relation between configuration and artifacts as well as the point in time when artifacts are modified, created or deleted. Therefore, VIL receives a source artifact model to containing the source artifacts, a target artifact model indicating the location where instantiated artifacts shall be stored as well as the actual configuration. Both artifact models may be the same, but in hierarchical or multi product line settings they typically differ. The instantiation process is described in terms of VIL rules and the types and operators provided by the VIL type system. VIL rules can be seen as (named and parameterizable) functions that can optionally have pre and post conditions (inspired by rules of the *make* build system [132]). As rules can be triggered through pre conditions, but also called explicitly, VIL allows specifying the instantiation process in a hybrid way using a rule-based, imperative or mixed style. rt-VIL reuses these concepts, but defines a different execution model and focuses on the modification of the (runtime) configuration in terms of strategies and tactics. Strategies and tactics utilize operations of the type system directly or through VIL rules. Further, if required, rt-VIL may execute the platform instantiation process or selected parts such as the instantiation of a pipeline.
- **Artifact instantiation:** Actually, VIL consists of two languages due to pragmatic considerations. While VIL focuses on the instantiation process for an entire product line, the Variability Template Language VTL aims at the instantiation of individual artifacts. VTL is inspired by the xtend language [54], but integrated with VIL, i.e., VTL template scripts are based on the same type system and artifact models as VIL. Typically, rt-VIL will utilize artifact instantiation via VTL only for runtime code instantiation (enactment pattern ES-4) or through the execution of (parts of) the platform instantiation process.

8.3.3 rt-VIL Language Elements

In this section, we discuss the language concepts of rt-VIL in terms of their syntax and (informal) semantics. Thereby we focus on rt-VIL and detail only those language elements of VIL that are actually needed to describe rt-VIL. For more details on VIL, please refer to the actual VIL language specification [140][19].

In the following sub-sections, we discuss individual language elements of rt-VIL, namely, variables in Section 8.3.3.1, expressions in Section 8.3.3.2, the rt-VIL module called script in Section 8.3.3.3, strategies in Section 8.3.3.4, tactics in Section 8.3.3.5 and enactment in Section 8.3.3.6.

We use the following styles throughout this section to illustrate the rt-VIL language elements:

- The syntax as well as the examples is illustrated in `Consolas` or, within text, in `Courier New`.
- **Keywords** are highlighted using bold font.
- *Elements and expressions* that shall be substituted by concrete values, identifiers, etc. are highlighted in italics.
- Identifiers are used to define names for modeling elements that allow the clear identification of these elements. We define identifiers following the conventions typically used in programming languages. Identifiers may consist of any combination of letters and numbers, while the first character must not be a number.
- Statements are separated using a semicolon ";" (most other language concepts may optionally be ended by a semicolon).

[19] The VIL language specification is a living document that we update in order to reflect improvements and extensions. As soon as the initial version of rt-VIL is implemented, the relevant parts of Section 8.3 of this deliverable will be included into the VIL language specification.

- Different types of brackets are used to indicate lists "()", sets "{}", blocks "{}", etc. This is similar to the Java programming language.
- We indicate comments using "//" and "/* ... */" (cf. Java).

We use the following structure to describe the language elements:

- **Syntax:** This part illustrates the syntax of a language element. We use the syntax to illustrate the valid definition of elements as well as their combination. Please note that we give a formal definition of the grammar in Appendix B.
- **Description of syntax:** Here, we provide the description of the syntax and the associated semantics. We describe each element, the semantics and the interaction with other elements. As we focus on rt-VIL rather than VIL, we refer for details to the VIL language specification [140].
- **Example:** The use of the abstract concepts is illustrated in a (simple) example.

8.3.3.1 Variables

Akin to programming languages, the actual state of the execution of a rt-VIL/VIL specification is given in terms of global and local variables, which may be explicit or implicit. Global variables are defined in the global scope, the top-level specification module in rt-VIL/VIL, and visible within all strategies, tactics and VIL rules. In contrast, local variables are defined within a strategy, tactic or VIL rule and, thus, belong to the respective local scope. Explicit variables are defined by the Adaptation Manager in the specification, while implicit variables are defined automatically by rt-VIL/VIL, such as a (read-only) collection of all sub-strategies and tactics used within a strategy. As rt-VIL/VIL is a typed language, also variable declarations are typed. The syntax for declaring a variable is similar to well-known programming languages such as Java or C. Variables can be declared and initialized later using the syntax `Type name;` Optionally, variables can be initialized as part of the declaration Type name = expression; using an expression that either matches the type of the variable or can automatically be converted to the type of the variable.

In addition, global rt-VIL variable declarations can be marked by a modifier as `persistent`, i.e., the actual value will be persisted at the end of the script execution and restored in the next execution regardless of given initialization expressions. This provides an additional kind of common adaptation knowledge in addition to the decision variables of the runtime configuration, the runtime components and the QoS impact database.

8.3.3.2 Expressions

rt-VIL allows specifying complex expressions in order to define conditions, to calculate values or to call VIL rules. Basically, expressions follow the same syntax as in VIL [140], which is inspired by OCL [108] and aligned with IVML.

Typically, expressions can be evaluated and the evaluation leads to a certain value. However, not all decision variables from IVML must be defined for instantiation, as, e.g., undefined compound variables may indicate the absence of an artifact or a functionality. Thus, VIL variables may be undefined and, consequently, VIL expressions evaluate to undefined in case that one of the involved variables is undefined or one of the used expressions evaluates to undefined. As in OCL [108], undefined expressions are handled gracefully in VIL, i.e., the statement using the expression is not further evaluated. This may (intentionally) lead to disabled strategies or tactics or, if instantiation is applied, partially instantiated artifacts.

8.3.3.3 Script

In rt-VIL, a script (`rtVilScript`) is the top-level containing element. This element is mandatory as it identifies the scope for the strategies, tactics and VIL rules to be defined. The definition of a script requires a name in order enable script imports or extensions. rt-VIL scripts can be imported through their name using an import statement. Import statements are given before the script scope. Scripts can be extended akin to classes in object orientation allowing strategies, tactics or rules to be overridden. Further, the definition of a script requires a parameter list specifying the expected information from the execution environment. At least, the source artifact model (carrying the actual state of the runtime components), the actual (runtime) configuration with bound runtime decision variables (from monitoring / analysis), and the target artifact model (containing the instantiated artifacts or components, such as enactment commands) must be provided as parameters.

In a script, all decision variables of the (runtime) configuration are available through their (qualified) name. As IVML configurations may be partial or even composed dynamically, the actual decision variables, their types and type definitions are not necessarily known at the point in time when the script is specified. Thus, the validity of IVML names can only be determined at execution time of the script when also the actual (runtime) configuration is known. This may complicate the development of VIL scripts as actually unknown identifiers will at least lead to a warning. To support the Adaptation Manager in specifying valid and readable scripts, rt-VIL provides the **`advice`** annotation specifying the actual IVML model. This annotation allows using IVML identifiers instead of variable names and, moreover, maps all IVML types into the rt-VIL type system and enables dynamic dispatch also over IVML types.

Syntax:

```
//imports
@advice(ivmlName)
rtVilScript name (parameterList) extends name₁ {
  //optional version specification
  //global variable declarations / definitions
  //strategy declarations
  //tactics declarations
  //rule declarations
}
```

Description of syntax[20]:

- First, all referenced scripts are imported. The basic syntax for an import statement is `import name;` For more details on the import syntax and the import conventions, please refer to [140].
- An optional advice annotation may declare the underlying Configuration Meta Model to map IVML identifiers and types into rt-VIL. The basic syntax is shown above. More details are given in [140].
- The keyword **`rtVilScript`** defines that the identifier *name* is defined as a new adaptation behavior script with contained strategies, tactics and VIL rules.
- The *`parameterList`* denotes the arguments to be passed to a rt-VIL script for execution. In rt-VIL, the source and target artifact model (given in terms of the type `Project`), the (runtime) configuration as well as the triggering event are passed in.
- A rt-VIL script may optionally extend an existing (imported) VIL script. This is expressed by **`extends`** *$name_1$*, whereby *$name_1$* denotes the name of the extending script. The strategies, tactics and rules of the extended script are available, in particular for overriding.
- Further, optional global variables may be declared or defined, respectively. A variable declaration consists of a type and a unique variable name, optionally followed by an initialization expression based on already defined variables or parameters as shown in Section 8.3.3.1. All types defined by the rt-VIL type system can be used.
- A rt-VIL script may contain strategies, tactics and VIL rules. While there is no predefined sequence of defining strategies and tactics, the actual definition sequence is taken into

[20] Akin to VIL, rt-VIL scripts can explicitly be versioned in order to support evolution. We do not detail versioning in this deliverable, as versioning will initially not be used in QualiMaster and may be beneficial in later stages of the project.

account if, however, the respective preconditions does not lead to a unique selection of strategies or tactics at execution time. VIL rules can be used to define functions or instantiation tasks. The sequence of VIL rules is not relevant for execution. Extending scripts may override strategies, tactics and rules by specifying the same signature as the original declaration in an extended script. Further, extending scripts may participate in dynamically dispatched execution by defining new typed alternatives. We discuss strategies in more detail in Section 8.3.3.4, tactics in Section 8.3.3.5 and VIL rules as part of Section 8.3.3.6.

Example:

```
@advice(QM)
rtVilScript QualiMasterFinancial (Project source, Configuration config, Project target,
    AdaptationEvent trigger) {
    // variables, strategies, tactics, etc.
}
```

The example above shows a very basic, empty rt-VIL script. The script is linked against the QualiMaster Configuration Meta Model (currently just called QM) and called `QualiMasterFinancial`. The script receives the source artifact model, the (runtime) configuration, the target artifact model (that shall be modified) and the actual adaptation event.

8.3.3.4 Strategy

A strategy represents a high-level adaptation objective capturing the logical aspect of planning an adaptation. It encapsulates the relevant sub-strategies and tactics for the realization of the stated adaptation objective. The impact of executing strategies can be recorded in the QoS database and used to realize proactive adaptation.

Below, we discuss the execution semantics of a rt-VIL script in order to highlight the interplay of strategies, tactics and VIL rules.

- First, the applicable top-level strategies are determined. Multiple strategies may define the same objective [28, 73] so that additional preconditions (including the triggering event type) can be used to differentiate among the applicable strategies. If still multiple strategies are applicable, they are processed in order of their specification. Strategies can handle distinct events, e.g., a startup event to initialize the runtime variables considering the overall resource utilization or to process a user trigger (REQ-A-1). Initialization of runtime variables such as the active algorithm or functional parameters may also be given in terms of default values in the configuration. Please note that imports support the modularization of rt-VIL scripts, i.e., a top-level script may import separate scripts handling for example different lifecycle phases such as startup, runtime and shutdown of a pipeline.
- A strategy can specify multiple sub-strategies and tactics to realize the adaptation towards the actual (failing) objective. The selection of the sub-strategies and tactics during script execution can be static or dynamic. In the static case, the first applicable sub-strategy or tactic in the given sequence is executed depending on their preconditions. In the dynamic case, a weighting function (inspired by [73]) determines the ranking of the sub-strategies and tactics, which, in turn, determines the sub-strategy or tactic to be executed. As in S/T/A [73], the weighting function may consider historical information such as the previous impact. In contrast to Stitch [28] and S/T/A [73], we allow sub-strategies to enable a hierarchical breakdown of the adaptation specification along nested structures of the underlying application (such as the infrastructure-pipeline-elements structure in QualiMaster) and, thus, dynamic execution also on the nested levels.
- Ultimately, a tactic is executed to determine the new settings of the runtime configuration. However, a tactic may fail, e.g., as its changes lead to an invalid runtime configuration detected by runtime reasoning. A tactic may revert to the last valid configuration, signal its failure or even cause the end of the script execution. In case of a failing tactic, the strategy continues with the next applicable sub-strategy or tactic in the sequence of ranking, respectively.

- A strategy succeeds if at least one of its sub-strategies or tactics succeeds. Finally, this determines whether the top-level strategy succeeds or fails.
- If the executed top-level strategy succeeded, the enactment of the runtime configuration starts, i.e., a predefined VIL-rule (overridable as a kind of strategy method [57]) is executed that specifies the mapping from changed decision variables into enactment commands, in QualiMaster for the Coordination Layer.

Syntax:

As discussed above, a strategy can have an objective, preconditions, or a weighting function and denote sub-strategies and tactics that are supposed to handle the situation indicated by the objective. The syntax of the strategy header follows the syntax of VIL rules, which, in turn, is inspired by a combination of OCL [108] and make [132].

The body of a strategy is separated into three parts, namely 1) the objective (along with supporting local variables), 2) the breakdown of the strategy into sub-strategies and 3) tasks (along with the optional weighing function) and post processing. Actually, the objective is optional, in particular to support startup and shutdown strategies, which may need to be executed regardless of objectives. Also the weighting function is optional in order to enable static (reactive) rule-based adaptation. Please note that both, objective and weighting function may rely on VIL rules or basic operations that can be provided by programmed extensions. This enables the use of complex functions, e.g., to realize a self-adjusting adaptation through the common knowledge such as done in [14, 62, 141]. Sub-strategies and tactics can be stated using different syntactical forms as indicated in the syntax below, e.g., including a logical guard expression [28], a descriptive record (supporting the weighting function) or a maximum execution time [28]. Finally, post processing can execute further VIL rules, e.g., to revert to a previous runtime configuration.

```
strategy name (ParameterList) = post : pre {
  // local variable declarations
  objective expression;
  breakdown {
    weighting (name : expression);
    // optional sub-strategies
    strategy name(ParameterList);
    strategy guardExpression name(ParameterList);
    strategy guardExpression name(ParameterList) with (name = value);
    strategy guardExpression name(ParameterList) @numExpression;

    // tactics
    tactic guardExpression name(ParameterList);
  }
  // post processing
}
```

Description of syntax:

- The keyword `strategy` indicates on this level the declaration of a strategy.
- A strategy is identified by its *name*, e.g., for referencing the strategy in other strategies.
- A strategy can declare parameters, which can be used within the strategy or for defining the pre- or post-condition. Parameters are given in terms of types and parameter names separated by commas if more than two parameters are listed. Top-level strategies refer to the parameters of the containing script, either specifying all parameters declared by the script (in case of a strategy that performs instantiation) or by just declaring the actual runtime configuration and the triggering event. Sub-strategies may declare specific decision variable types instead of the entire runtime configuration. Further, a strategy may declare the specific event type it reacts on as parameter (and as an implicit precondition).

- Akin to VIL rules, a strategy may have logical pre- and post-conditions. As multiple strategies may define the same objective, the precondition can be used to select among the applicable strategies [28, 73]. If given, the post-condition is checked after the execution of a strategy to determine (in addition to the objective) whether the application of the strategy was successful. Post-conditions in strategies are inspired by the effect specification of tactics in [28]. If neither pre- nor post-condition are given, also the separating colon can be omitted.
- Within the rule body marked by curly brackets, first local variables can be declared and initialized. In particular, information that is relevant to the calculation of the objective or the weighting function can be collected here, i.e., these variables are intentionally visible to the entire strategy block.
- The objective states a logical condition utilizing observed quality properties (via the runtime variables of the configuration) in expressions such as their target range or utility / cost functions akin to [73]. Please note that the objective is a part of the precondition, which is syntactically highlighted due to its importance for adaptation.
- Breakdown of the strategy into sub-strategies and tactics. The breakdown block is executed once to collect the sub-strategies and tactics. This is required to enable the collection of an unknown number of alternatives at design time due to the openness of the configuration in QualiMaster as discussed above. In a second step, the identified sub-strategies and tactics are ranked and executed.
    - The weighting function enables the dynamic selection of the contained sub-strategies and tactics for both, reactive adaptation (without updated common knowledge) and for proactive adaptation (with prediction or updated common knowledge such as QoS impacts). Akin to [73], the weighting function determines dynamically the ranking of the sub-strategies and tactics. Basically it is a function that maps the alternatives (sub-strategies or tactics) under consideration of the actual system state (taken from the runtime configuration or the runtime components) to a real value, e.g., using a utility-cost calculation. Thereby, it acts like an iterator over the alternatives and selects the best option with access to additional information given in the listing of the sub-strategies and tactics. In particular, rt-VIL operations provide access to the historical impact of a given alternative [73], the prediction of a certain quality property (REQ-C-12), or the aggregated quality of a (desired) pipeline. Further, the weighting function can take the adaptation configuration (see Section 0) into account to consider the importance of the different quality parameters. If specified, the weighting function must be given directly at the beginning of the breakdown block.
    - The remainder of the block declares the alternative sub-strategies and tactics. As described above, sub-strategies and tactics can be guarded by an expression [28], which influences the ranking. Sub-strategies and tactics are referenced by their signature and, in case that a sub-strategy or tactic is used multiple times, a descriptive record can be given to support the calculation of the weighting function. Finally, inspired by [28], a maximum execution time (expression) can be stated, i.e., a relative time bound within the sub-strategy or tactic is either completed or it fails.
- The post processing part allows to execute further VIL rules and operations, e.g., to revert to a previous successful runtime configuration.

Finally, a strategy calls the predefined rt-VIL rule `Boolean validateTactic(Configuration config)` the end of a (so far) successful tactic in order to validate the results of the executed operations. By default, this predefined rt-VIL rule performs runtime reasoning (on the changed variables) utilizing the EASy-producer runtime infrastructure. This method can be overridden to realize other forms of validation, e.g., for traditional architecture-based adaptation. Please note that successful top-level strategies call further predefined methods for updating the common knowledge and enactment as we describe in Section 8.3.3.6.

Example:

The first example below illustrates a simple reactive strategy, which considers exactly one quality parameter, namely the pipeline throughput. As input, the strategy receives the actual QualiMaster runtime configuration (expressed through the type `QM` made available through a respective `advice`) and an unspecified adaptation event. In case that a pipeline does not fulfill an overall pipeline throughput value (here, a fictive configuration value specified as a fixed condition in a VIL rule[21]), it aims at changing the pipeline with highest throughput deviation (`candidate`), with a priority on changing a single algorithm parameter. In the extreme case of no successfully executed tactic, the strategy just performs load shedding on the input of `candidate`. Please note that a top-level strategy is implicitly ended by a validation of the changed variables of the runtime configuration and, in case of success, an enactment of the changed variables. We provide details on the enactment in Section 8.3.3.6.

```
strategy infrastructure (QM config, AdaptationEvent trigger) = {
  setOf(Pipeline) issues = throughputFailingPipelines(config);
  Pipeline candidate = issues.sortAscending().first();

  objective null != candidate;

  breakdown {
    strategy changeSingleParameter(candidate, trigger);
    strategy changeSingleAlgorithm(candidate, trigger);
    // extreme fallback
    tactic shedLoad(candidate, trigger);
   }
}

setOf(Pipeline) througputFailingPipelines(QM config) = {
  config.pipelines()-> select(p|
    p.throughput() < config.minPipelineThroughput());
}
```

The second example illustrates a strategy with dynamic selection of a comprehensive loop-enumeration of the tactics (based on all alternatives defined in the Configuration). As stated by the signature, the strategy handles a regular adaptation on a `FamilyElement` of a pipeline. The objective of the strategy is that no quality parameter of the family element fails (the related definition is not shown in the example). The weighting function targets the optimization of a utility-cost tradeoff given in terms of two VIL functions (not detailed below). Thereby, additional information defined by the listed tactics is used. For illustration, this example simply enumerates all possible tactics for all available algorithm parameters and family members using map expressions, the VIL version of a loop. Please note that a sub-strategy is implicitly ended by a validation of the changed variables in the runtime configuration. We provide details on the enactment in Section 8.3.3.6.

```
strategy changeAlgorithm (FamilyElement elt, RegularAdaptationEvent trigger) = {
  setOf(Quality) failed = failedQualities(elt);
  objective failed.isEmpty();
  breakdown {
    weighting (e: utility(e.elt, e.quality) – cost(e.elt, e.quality));
    map(Quality q: failed) {
      map(Parameter p: elt.family().parameter()) {
        tactic changeParameter(elt, trigger, p) with (elt = p, quality = q);
      }
      map(Algorithm a: elt.family().members()) {
```

[21] VIL rules return the latest calculated value, i.e., there is no explicit return statement.

```
        tactic changeAlgorithm(elt, trigger, a) with (elt = a, quality = q);
      }
    }
  }
}
```

8.3.3.5 Tactics

A tactic defines the steps of adaptation to be carried out in order to achieve the objective defined by a calling strategy. As in Stitch [28], a tactic cannot call other tactics or strategies, just rt-VIL operations or rt-VIL rules. Akin to strategies, the syntax of the tactic header follows the syntax of VIL rules, which is inspired by a combination of OCL [108] and make [132].

A tactic can have a precondition [28] and an effect specification [28] (we represent this as the post-condition of a tactic in rt-VIL). A tactic is enabled if its precondition holds. A tactic is successful, if its operations are executed successfully and its (optional) post-condition holds. So far, tactics are rather similar to VIL rules. In contrast, tactics serve as a basis for recording the QoS impact through strategies. Further, the predefined rt-VIL rule `Boolean validateTactic(Configuration config)` is called at the end of a (so far) successful tactic in order to validate the results of the executed operations. By default, this predefined rt-VIL rule performs runtime reasoning (on the changed variables) utilizing the runtime version of EASy-producer. This method can be overridden for other forms of validation, e.g., for traditional architecture-based adaptation.

Syntax:

```
tactic name (ParameterList) post : pre {
  // local variable declarations
  // runtime configuration changes, rule calls
  // alternative or iterative execution
}
```

Description of syntax:

- The keyword `tactic` indicates on this level the declaration of a tactic.
- The *`name`* allows identifying the rule for explicit rule calls or for script extension.
- The *`parameterList`* specifies explicit parameters which may be used as arguments for precondition rule calls as well as within the rule body. Parameters are given in terms of types and parameter names separated by commas if more than two parameters are listed. Parameter must either be bound by the calling strategy.
- The optional post-condition *`post`* specifies the expected outcome of the tactic execution in terms of a logic expression akin to the effects specification in [28].
- The optional precondition `pre` specifies whether the tactic is considered for execution at all (in addition to the precondition in the strategy). If neither pre- nor post-condition are given, also the separating colon can be omitted.
- The body of the strategy is specified within the following curly brackets. Local variable declarations, rule calls, alternative and looped execution and, in particular, changes to the runtime variables of the configuration can be specified here.

Example:

This example continues the second example from Section 8.3.3.4 on changing algorithms and parameters of a QualiMaster pipeline. The tactic shown below performs all operations needed to change a tactic, i.e., it transfers the parameter values of the actual family to the new algorithm and finally changes the current algorithm of the actual family. Please note that a tactic is implicitly ended by a validation of the changed variables in the runtime configuration. We provide details on the enactment in Section 8.3.3.6.

```
tactic changeAlgorithm (FamilyElement elt, AdaptationEvent trigger,
  Algorithm newAlgorithm) : {
  Algorithm current = elt.family().current();
  // transfer parameter
  map(Parameter p: elt.family()) {
    newAlgorithm.algorithm.parameter()
      ->select(q|q.name == p.name).first().setValue(p.value());
  }
  // set algorithm
  elt.family().setCurrent(newAlgorithm);
}
```

8.3.3.6 Enactment

In Stitch [28] and S/T/A [73], enactment happens through the execution of actions, i.e., actions bind the system under adaptation to the execution of the adaptation behavior specification. In contrast, we perform a configuration-based adaptation, i.e., we must validate the constraints of the Configuration (Meta) Model upon changes of runtime decision variables as explained in Sections 3 and 8. Further, our approach aims to be independent from the underlying system and, thus, needs to specify the mapping from the runtime configuration to the system under adaptation. In QualiMaster, this leads to a translation of the actual values in the runtime configuration into commands of the Coordination Layer (cf. D5.1).

In rt-VIL, enactment happens if a top-level strategies succeeds. Then, rt-VIL executes predefined rt-VIL rules (similar to the strategy method pattern [57]), which can be overridden if required. For updating the common adaptation knowledge, e.g., with respect to the impact, rt-VIL calls

```
update(Strategy strategy)
update(Tactic tactic)
```

for each succeeded strategy and tactic. Finally, rt-VIL executes

```
enact(Project source, Configuration changed, Project target)
```

to map and execute the adaptation decisions. The passed in configuration contains the actually changed runtime decision variables that shall be enacted.

An illustrating example combining the enactment of algorithm changes with triggering a subsequent wavefront is shown below as a continuation of the example in Section 8.3.3.5. Basically, the `enact` method considers each changed pipeline and the contained pipeline elements through dynamic dispatch (not all VIL rules for the pipeline types are shown). The enactment of the `FamilyElement` creates the respective commands for the Coordination Layer in terms of a command sequence. It is important to note that the Coordination Layer receives (symbolic) names for pipeline elements, which must be translated to actual commands for the Execution Systems. Finally, the enactment rule schedules the subsequent pipeline elements for wavefront adaptation. Enacting the changes to the algorithms without scheduling the wavefront can be expressed by the first part of `enact(Family)` directly in the predefined enact function.

```
enact (Project source, QM changed, Project target) {
  map(Pipeline p: changed.pipelines()) {
    map(Source s: pipeline.sources()) {
      enact(p, s);
    }
  }
}
```

```
enact(Pipeline pipeline, PipelineElement elt) {
  // dynamic dispatch default - do nothing
}

enact(Pipeline pipeline, FamilyElement elt) {
  Family family = elt.family();
  CommandSequence cmd = new CommandSequence();
  if (null != family.current()) { // was changed?
    cmd.add(new AlgorithmChangeCommand(pipeline.name(),
      family.name(), family.current().name());
  }
  map (Parameter p: family.parameters()) {
    cmd.add(new ParameterChangeCommand(pipeline.name(),
      family.name(), p.name, p.value()));
  }
  map(Flow f: src) {
    PipelineElement e = f.destination();
    enact(e);
    cmd.add(new ScheduleWavefrontAdaptationCommand(
      pipeline.name(), e.name()));
  }
  cmd.execute();
}
```

8.4 Tool Support

In this section, we discuss the current state of the tool support for adaptation in QualiMaster based on the topics discussed in this section. Akin to the introduction of Section 8, we will follow in this section the MAPE-K cycle with an additional initial step on modeling the adaptation behavior specification.

- **Modeling:** Basically, modeling the adaptive behavior with rt-VIL relies on the Configuration (Meta) Model and the Configuration Core, i.e., EASy-Producer. As concluded in Section 7.3, the configuration support provides a solid basis for our work. Thus, we will use the VIL implementation as a basis and extend the grammar, the semantic analysis, the language infrastructure (for the type system) as well as the language execution (model) in order to realize rt-VIL on top of VIL. Initial steps in this direction have already been carried out, i.e., the extension of the grammar, the language infrastructure and semantic analysis and the language execution have been prepared, the rt-VIL grammar has been specified (see also Appendix B), and the related parser and Eclipse editor (including initial syntax-highlighting and content-assist) have been generated using xText[22]. As rt-VIL will extend EASy-Producer with domain-independent capabilities for runtime adaptation, also VIL will be extended by common functionality and elements of both languages, such as type checking operations for VIL expressions (actually present in IVML) or functions (as a special case of VIL rules).

  rt-VIL will also extend the QualiMaster configuration tool, in terms of a high-level specification support using a specialized editor as well as a detailed adaptation support as requested by REQ-C-13 and REQ-C-14.

- **Monitoring / Analysis:** As discussed in D5.1, the monitoring according to MAPE-K is realized by the QualiMaster Monitoring Layer, which relies on the information provided by the Execution Systems as well as the component-based resource consumption information provided by SPASS-meter [46]. As described in Section 8.1, the Monitoring Layer performs also the analysis of the SLA constraints defined in the Configuration and sends respective

[22] http://www.eclipse.org/Xtext/

triggering events to the Adaptation Layer. An initial version of the Monitoring Layer has been developed as described in D5.2.

- **Planning:** The adaptation plans (called “adaptation rules” in D1.2) are described in terms of rt-VIL strategies and tactics. As the generic implementation of rt-VIL is realized by EASy-Producer, the runtime version of EASy-Producer will serve as a basis for the implementation of the Adaptation Layer. A build-level tool for packaging an Eclipse-independent version of EASy-Producer in a flexible way (for the QualiMaster Adaptation Layer it will contain the QualiMaster-specific extension of rt-VIL, which is not required by other applications) has been developed and integrated into the EASy-Producer build process. The Adaptation Layer provides QualiMaster specific interfaces and functionality on top of rt-VIL as described in D5.2, hiding the generic capabilities of EASy-Producer. An initial version of the Adaptation Layer has been created as described in D5.2.
- **Enactment:** As described in this section, the enactment of the adaptation decisions happens in QualiMaster through the Coordination Layer, i.e., rt-VIL transforms the changes to the runtime configuration into commands of the Coordination Layer as shown in Section 8.3.3.6. An initial version of the Coordination Layer has been developed (see D5.2).
- **Knowledge:** Common adaptation knowledge will be handled by the Data Management Layer, i.e., the QoS impact history, (actualized) quality profiles, adaptation logs as well as the state of persistent variables will be stored there. For more details regarding the state of the Data Management Layer please refer to D5.2.

Monitoring of the pipeline operations (REQ-A-9) as well as the execution of the adaptation (REQ-A-10) are foreseen as a runtime extension of the QualiMaster configuration tool, which provides an integrated view on the infrastructure configuration. Here, problems identified by the monitoring or the actual state of the execution can be visualized, e.g., within the graphical pipeline editor.

In summary, the partners started the development of the tool support for the adaptation as well as its integration into the QualiMaster infrastructure. The realization of rt-VIL will profit from the broad and stable basis of VIL and, in turn, VIL and EASy-Producer will benefit from common developments for rt-VIL.

9 Conclusion and Future Work

Software Product Line Techniques, such as configuring and instantiating, aim at achieving more flexibility, shorter time to market, less production costs and better overall quality, in particular through consistently tailoring the software systems to the need of the customer. Adaptation of a software system at runtime aims at providing dynamic capabilities to cope with unforeseen changes in the environment, such as high load situations. Combining both approaches, namely configuration and tailoring before runtime with adaptation and re-configuration at runtime, allows a consistent characterization of the adaptation space before runtime (e.g., in terms of Service Level Agreements), to optimize a system for adaptation before runtime through instantiation and to adapt the system within these boundaries. This is in particular true, if also the instantiation of the system before and at runtime can be specified in a flexible way. In contrast to single-system development, such an encompassing configuration and adaptation approach supports the adoption for different application settings and domains.

In QualiMaster, we aim at such an encompassing configuration and adaptation approach for real-time data stream processing, which opportunistically utilizes software-based and hardware-based execution. In this deliverable, we discussed basic terminology in Section 2 as well as overall challenges for WP4 (Section 3) on quality-aware configuration and adaptation and identified the main topics for our work, namely:

- Defining and analyzing a specific set of quality parameters,
- Enabling a flexible quality-aware configuration and instantiation of the QualiMaster infrastructure, and
- Adapting the real-time data processing at runtime based on the configuration of the instantiated QualiMaster infrastructure and the actual quality parameters observed at runtime.

We discussed related work for these three topics in Section 4 as well as requirements for the work in this work package and the concepts in this deliverable drawn from previous deliverables and the DoW in Section 5. Based on these requirements, we discussed the concepts and the current state of the realization for the three main topics of this deliverable.

- We performed a quality survey, derived the relevant quality properties, discussed an extensible quality taxonomy based on recent standards and outlined our plans for pipeline level quality analysis in Section 6.
- Driven by the configuration requirements, we created a Configuration Meta Model, structured it along the architecture of the QualiMaster infrastructure and discussed the Configuration Model in detail according to this structure in Section 7. We modeled the Configuration Meta Model in terms of IVML, the INDENICA variability modeling language, which is, in particular, able to cope with runtime decision variables (for integrating adaptation and configuration) and topological configuration (for modeling and structurally analyzing data processing pipelines). As described in D5.2, the instantiation process of the QualiMaster infrastructure is able to generate the integrating implementation of the topological pipeline configuration for Apache Storm. Finally, we presented the current state of the realization, in particular regarding the QualiMaster infrastructure configuration tool, a domain-specific user-supporting frontend, and the Configuration Core (EASy-Producer).
- Based on the foundation provided by the Configuration Core, we derived the adaptation approach for the QualiMaster infrastructure. In particular, we discussed a set of enactment patterns to realize runtime adaptation in QualiMaster and our approach to the flexible specification of the adaptation behavior. Thereby, we aim at an extension of successful work in the field of adaptive software systems, which combines adaptation and configuration (as an extension of our instantiation approach) and targets adaptation of real-time data stream processing. We presented this approach in terms of a description of the syntax and semantics of rt-VIL, the run-time Variability Instantiation Language. Finally, we discussed the state of the realization based on the capabilities of the Configuration Core.

The approaches and concepts presented in this deliverable realize or at least target all requirements stated in Section 5. As also indicated in the respective sections, all configuration requirements are reflected in the Configuration Meta Model, i.e., the configuration options for the resource pool (software- and hardware-based execution), the data management, the pipeline design and the multi-pipeline execution (REQ-C-2 - REQ-C-10, and REQ-C-15). These requirements are already realized in the QualiMaster infrastructure configuration tool. This is also true for shielding the Application User from the configuration (REQ-C-1) through the credential-protected views of the infrastructure configuration tool. The configuration of quality parameters and adaptation (REQ-C-11 - REQ-C-14) is provided by the Configuration Meta Model and will be realized in the QualiMaster infrastructure configuration tool in the future. Regarding adaptation requirements, all requirements are covered by concepts, in particular user triggers (REQ-A-1) through specific events, quality dimensions (REQ-A-2 - REQ-A-8, REQ-A-12) through runtime variables as well as strategies and tactics. The integration of reconfigurable hardware in the adaptation (REQ-A-11) happens transparently through Coordination Layer commands. Operation level monitoring (REQ-A-9, REQ-A-10) is scheduled for future realization by tool support based on the QualiMaster platform.

The development and realization of the approaches and concepts cover the challenges presented in Section 3 for the work in this work package. In more detail, we target the

- Unification of adaptation and configuration (Challenge C1) by combining the advanced variability modeling concepts provided by INDENICA Variability Modeling Language (IVML) and instantiation concepts of the Variability Instantiation Language (VIL) with the concepts of rt-VIL presented in this deliverable.
- Unification of software- and hardware-based processing (Challenge C2) through the configuration of the resource pool, the transparent integration of hardware-based algorithms into algorithm families, the creation of hardware connectors while platform instantiation and the transparent handling of enactment comments created by a rt-VIL specification and executed by the Coordination Layer.
- Topological configuration (Challenge C3) by specific concepts of IVML, namely typed references, an explicit type system supporting sub-typing and topology-related constraint capabilities. While basic concepts for topological variability were already present, the work in QualiMaster contributed additional capabilities and allowed us to validate the approach in terms of a complex topological setting. The setting is constituted by the QualiMaster data processing pipelines, a data flow graph connected to configuration parts, such as the resource pool, the data management or the algorithm families.
- Flexible automated instantiation (Challenge C4) through the concepts and capabilities of VIL, a language for specifying the instantiation process and the artifact instantiation of Software Product Lines. The QualiMaster platform instantiation process validates these concepts and capabilities in terms of the instantiation of Storm data processing topologies from (more abstract) QualiMaster data processing pipelines. Although VIL existed before QualiMaster, the work on the platform instantiation helped extending and validating VIL for the (practical) instantiation of topologies.
- Flexible adaptation specification (Challenge C5) by the concepts of rt-VIL. rt-VIL will be realized on top of VIL, which provides a solid basis, and enables also traditional product line instantiation at runtime.
- Combination of data and technical quality (Challenge C6) by the quality taxonomy constructed based on the results of a quality survey. All quality parameters have been included into the Configuration Meta Model in order to serve as a basis for SLAs, pipeline constraints and adaptation.
- Adaptation for real-time data processing (Challenge C7) by specific enaction patterns discussed in Section 8.2. These patterns will be realized through generic and domain-specific adaptation rt-VIL strategies, tactics as well as through the concept of a wavefront adaptation.

In summary, the requirements and challenges for quality-aware configuration and adaptation in QualiMaster are covered by the approaches and concepts in this deliverable. In the future, we will work on the realization of the concepts, in particular on rt-VIL, the enactment patterns, the adaptation specification and the configuration of quality parameters and adaptation through the QualiMaster infrastructure configuration tool. We also aim at validating and evaluating the components that will be developed, starting with the analysis of the enactment patterns, ranging over test cases for the new components up to scenario evaluations on generated and real data for the quality-aware adaptation of real-time data stream processing.

10 References

[1] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Çetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, S. B. Zdonik. The Design of the Borealis Stream Processing Engine. *CIDR*, 277–289, 2005.

[2] D. J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, S. Zdonik. Aurora: A New Model and Architecture for Data Stream Management. *The VLDB Journal*, 12(2):120–139, August 2003.

[3] N. Abbas J. Andersson. Architectural reasoning for dynamic software product lines. *Workshop on Dynamic Software Product Lines (DSPL '13), SPLC Workshops*, 117–124, 2013.

[4] A. Aleti, S. Bjornander, L. Grunske, I. Meedeniya. ArcheOpterix: An Extendable Tool for Architecture Optimization of AADL Models. *ICSE Workshop on Model-Based Methodologies for Pervasive and Embedded Software (MOMPES '09)*, 61–71, Washington, DC, USA, 2009. IEEE Computer Society.

[5] J. Andersson, R. Lemos, S. Malek, D. Weyns. Modeling Dimensions of Self-Adaptive Software Systems. B. H. Cheng, R. Lemos, H. Giese, P. Inverardi, J. Magee, editors, *Software Engineering for Self-Adaptive Systems*, volume 5525 of *LNCS*, 27–47. Springer, 2009.

[6] D. Ayed. DiVA: Dynamic Variability in complex Adaptive systems. *International Workshop on Aspects, Dependencies and Interactions (held at ECOOP '08)*, 57–61, 2008.

[7] K. Bak, K. Czarnecki, A. Wasowski. Feature and Meta-models in Clafer: Mixed, Specialized, and Coupled. B. Malloy, S. Staab, M. van den Brand, editors, *International Conference on Software Language Engineering (SLE '10)*, 102–122. Springer, 2010.

[8] L. Baresi. Self-adaptive systems, services, and product lines. *International Software Product Line Conference (SPLC '14)*, 2–4, 2014.

[9] J. Barreiros A. Moreira. A Cover-based Approach for Configuration Repair. *International Software Product Line Conference (SPLC '14)*, 157–166, 2014.

[10] J. Bartholdt, M. Medak, R. Oberhauser. Integrating Quality Modeling with Feature Modeling in Software Product Lines. *International Conference on Software Engineering Advances (ICSEA '09)*, 365–370, 2009.

[11] C. Batini, C. Cappiello, C. Francalanci, A. Maurino. Methodologies for Data Quality Assessment and Improvement. *ACM Comput. Surv.*, 41(3):16:1–16:52, July 2009.

[12] S. Becker, H. Koziolek, R. Reussner. The Palladio component model for model-driven performance prediction. *Journal of Systems and Software*, 82:3–22, 2008.

[13] S. Becker, H. Koziolek, R. H. Reussner. Model-based performance prediction with the palladio component model. *International Workshop on Software and performance (WOSP '07)*, 54–65, 2007.

[14] N. Bencomo, A. Belaggoun, V. Issarny. Dynamic Decision Networks for Decision-making in Self-adaptive Systems: A Case Study. *International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '13)*, 113–122, 2013.

[15] T. Berger, R. Rublack, D. Nair, J. M. Atlee, M. Becker, K. Czarnecki, A. Wasowski. A Survey of Variability Modeling in Industrial Practice. *International Workshop on Variability Modelling of Software-intensive Systems (VaMoS '13)*, 7, 2013.

[16] T. Berger, S. Stanciulescu, O. Øgård, Ø. Haugen, B. Larsen, A. Wasowski. To connect or not to connect: experiences from modeling topological variability. *International Software Product Line Conference (SPLC '14)*, 330–339, 2014.

[17] D. Beuche O. Spinczyk. Variant management for embedded software product lines with pure::consul and AspectC++. *Conference on Object-oriented programming, systems, languages, and applications (OOPSLA '03) Companion*, 108–109, 2003.

[18] A. Beygelzimer, A. Riabov, D. Sow, D. S. Turaga, O. Udrea. Big Data Exploration Via Automated Orchestration of Analytic Workflows. *International Conference on Autonomic Computing*, 153–158, 2013.

[19] M. Bovee, R. P. Srivastava, B. Mak. A conceptual framework and belief-function approach to assessing overall information quality. *Int. J. Intell. Syst.*, 18(1):51–74, 2003.

[20] F. Boyer, O. Gruber, D. Pous. Robust Reconfigurations of Component Assemblies. *International Conference on Software Engineering (ICSE '13)*, 13–22, 2013.

[21] M. Brambilla, J. Cabot, M. Wimmer. *Model-Driven Software Engineering in Practice*. Morgan & Claypool Publishers, 1st edition, 2012.

[22] Y. Brun, G. Marzo Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Müller, M. Pezzè, M. Shaw. Engineering self-adaptive systems through feedback loops. B. H. Cheng, R. Lemos, H. Giese, P. Inverardi, J. Magee, editors, *Software Engineering for Self-Adaptive Systems*, volume 5525 of *LNCS*, 48–70. Springer, 2009.

[23] F. Buschmann, K. Henney, D. C. Schmidt. *Pattern-oriented Software Architecture: On Patterns and Pattern Languages*. John Wiley and Sons, 2007.

[24] C. Canal, J. Manuel Murillo, P. Poizat. Software Adaptation. *J. UCS*, 14(13):2107–2109, 2008.

[25] S. Chakravarthy Q. Jiang. *Stream Data Processing: A Quality of Service Perspective Modeling, Scheduling, Load Shedding, and Complex Event Processing*. Springer, 1st edition, 2009.

[26] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, F. Reiss, M. A. Shah. TelegraphCQ: Continuous Dataflow Processing. *International Conference on Management of Data (SIGMOD '03)*, 668–668, 2003.

[27] B. H. Cheng, R. Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, G. Marzo Serugendo, S. Dustdar, A. Finkelstein, C. Gacek, K. Geihs, V. Grassi, G. Karsai, H. M. Kienle, J. Kramer, M. Litoiu, S. Malek, R. Mirandola, H. A. Müller, S. Park, M. Shaw, M. Tichy, M. Tivoli, D. Weyns, J. Whittle. Software Engineering for Self-Adaptive Systems: A Research Roadmap. B. H. Cheng, R. Lemos, H. Giese, P. Inverardi, J. Magee, editors, *Software Engineering for Self-Adaptive Systems*, 1–26. Springer, 2009.

[28] S.-W. Cheng, D. Garlan, B. Schmerl. Stitch: A Language for Architecture-based Self-adaptation. *J. Syst. Softw.*, 85(12):2860–2875, 2012.

[29] A. Classen, Q. Boucher, P. Heymans. A text-based approach to feature modelling: Syntax and semantics of TVL. *Sci. Comput. Program.*, 76:1130–1143, 2011.

[30] A. Classen, P. Heymans, P.-Y. Schobbens. What's in a Feature: A Requirements Engineering Perspective. *International Conference on Fundamental Approaches to Software Engineering (FASE'08)*, 2008.

[31] P. Clements L. Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2002.

[32] INDENICA Consortium. Description of Feasible Case Studies. Technical Report Deliverable D5.1, 2011. http://www.indenica.eu.

[33] INDENICA Consortium. Variability Implementation Techniques for Platforms and Services (Interim). Technical Report Deliverable D2.2.1, 2011. http://www.indenica.eu.

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

[35] INDENICA Consortium. Tool Suite for Deployment, Monitoring and Controlling of Virtual Service Platforms. Technical Report Deliverable D4.2.2, 2013. http://www.indenica.eu.

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

[37] I. Crnkovic, M. Larsson, O. Preiss. Concerning Predictability in Dependable Component-Based Systems: Classification of Quality Attributes. R. de Lemos, C. Gacek, A. Romanovsky, editors, *Architecting Dependable Systems III*, volume 3549 of *LNCS*, 257–278. Springer, 2005.

[38] K. Czarnecki, P. Grünbacher, R. Rabiser, K. Schmid, A. Wasowski. Cool Features and Tough Decisions: A Comparison of Variability Modeling Approaches. *International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS '12)*, 173–182, 2012.

[39] K. Czarnecki, S. Helsen, U. Eisenecker. Staged Configuration Through Specialization and Multi-Level Configuration of Feature Models. *Software Process Improvement and Practice*, 10(2):143–169, 2005.

[40] Y. Demchenko, P. Grosso, C. de Laat, P. Membrey. Addressing big data issues in Scientific Data Infrastructure. *International Conference on Collaboration Technologies and Systems (CTS '13)*, 48–55, May 2013.

[41] J. Dubus P. Merle. Applying OMG D&C Specification and ECA Rules for Autonomous Distributed Component-based Systems. *models@run.time06*, 2006.

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

[43] H. Eichelberger, C. Kröher, K. Schmid. Variability in Service-Oriented Systems: An Analysis of Existing Approaches. *International Conference on Service-Oriented Computing (ICSOC'12)*, 516–524, 2012.

[44] H. Eichelberger, C. Kröher, K. Schmid. An Analysis of Variability Modeling Concepts: Expressiveness vs. Analyzability. J. M. Favaro M. Morisio, editors, *International Conference on Software Reuse (ICSR '13)*, 32–48, 2013.

[45] H. Eichelberger K. Schmid. A Systematic Analysis of Textual Variability Modeling Languages. *International Software Product Line Conference (SPLC'13)*, 12–21, 2013.

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

[47] H. Eichelberger K. Schmid. Resource-optimizing Adaptation for Big Data Applications. *International Software Product Line Conference (SPLC '14) - Volume 2*, 10–11, 2014.

[48] H. Eichelberger K. Schmid. Mapping the Design-Space of Textual Variability Modeling Languages – A Refined Analysis (in press). *International Journal on Software Tools for Technology Transfer*, 2014. published online: http://link.springer.com/article/10.1007/s10009-014-0362-x.

[49] S. El-Sharkawy, S. Dederichs, K. Schmid. From Feature Models to Decision Models and Back Again - An Analysis Based on Formal Transformations. *Software Product Line Conference (SPLC '12)*, 126–135, 2012.

[50] C. Elsner, P. Ulbrich, D. Lohmann, W. Schröder-Preikschat. Consistent product line configuration across file type and product line boundaries. *International Software Product Lines Conference (SPLC' 10)*, 181–195, 2010.

[51] I. Epifani, C. Ghezzi, R. Mirandola, G. Tamburrelli. Model Evolution by Run-time Parameter Adaptation. *International Conference on Software Engineering (ICSE '09)*, 111–121, 2009.

[52] A. Filieri, C. Ghezzi, G. Tamburrelli. A formal approach to adaptive software: continuous assurance of non-functional requirements. *Formal Aspects of Computing*, 24(2):163–186, 2012.

[53] J. Floch, S. Hallsteinsen, E. Stav, F. Eliassen, K. Lund, E. Gjørven. Using Architecture Models for Runtime Adaptability. *IEEE software*, 62–70, 2006.

[54] Eclipse Foundation. Xtend - Modernize Java, 2013. Online available at: http://www.eclipse.org/xtend.

[55] C. Franke, M. Hartung, M. Karnstedt, K.-U. Sattler. Quality-aware mining of data streams. F. Naumann, M. Gertz, S. E. Madnick, editors, *International Conference on Information Quality (ICIQ '05)*, 300–315, 2005.

[56] S. Frølung J. Koistinen. QML: A Language for Quality of Service Specification. Technical Report HPL98-10, HP Labs, HP Software Technologies Laboratory, February 1998.

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

[58] M. Garcia-Valls, P. Basanta-Val, I. Estévez-Ayres. Supporting Service Composition and Real-time Execution Throught Characterization of QoS Properties. *Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '11)*, 110–117, 2011.

[59] S. Geisler, S. Weber, C. Quix. An Ontology-based Data Quality Framework for Data Stream Applications. *International Conference on Information Quality (ICIQ '11)*, Adelaide, Australia, 2011.

[60] A. Ghazal, T. Rabl, M. Hu, F. Raab, M. Poess, A. Crolotte, H.-A. Jacobsen. BigBench: Towards an Industry Standard Benchmark for Big Data Analytics. *International Conference on Management of Data (SIGMOD '13)*, 1197–1208, 2013.

[61] C. Ghezzi, L. S. Pinto, P. Spoletini, G. Tamburrelli. Managing Non-functional Uncertainty via Model-driven Adaptivity. *International Conference on Software Engineering (ICSE '13)*, 33–42, 2013.

[62] C. Ghezzi G. Tamburrelli. Reasoning on Non-Functional Requirements for Integrated Services. *International Requirements Engineering Conference (RE '09)*, 69 –78, 2009.

[63] V. Grassi, M. Marzolla, R. Mirandola. QoS-aware Fully Decentralized Service Assembly. *International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '13)*, 53–62, 2013.

[64] J. Greenfield, K. Short, S. Cook, S. Kent. *Software Factories: Assembling Applications with Patterns, Models, Frameworks and Tools*. Wiley, 2004.

[65] S. P. Gregg, R. Scharadin, E. LeGore, P. Clements. Lessons from AEGIS: Organizational and Governance Aspects of a Major Product Line in a Multi-program Environment. *International Software Product Line Conference (SPLC '14)*, 264–273, 2014.

[66] CVL group. Common Variability Language (CVL). Technical Report OMG Revised Submission, ad/2012-08-05, 2012. available at: http://www.omgwiki.org/variability.

[67] S. Hallsteinsen, M. Hinchey, S. Park, K. Schmid. Dynamic Software Product Lines. *Computer*, 41(4):93–95, 2008.

[68] M. Harman, Y. Jia, J. Krinke, W. B. Langdon, J. Petke, Y. Zhang. Search based software engineering for software product line engineering: a survey and directions for future work. *International Software Product Line Conference (SPLC '14)*, 5–18, 2014.

[69] C. Heinzemann S. Becker. Executing Reconfigurations in Hierarchical Component Architectures. *International Symposium on Component-based Software Engineering (CBSE '13)*, 3–12, 2013.

[70] R. N. Herbst, N. Huber, S. Kounev, E. Amrehn. Self-adaptive Workload Classification and Forecasting for Proactive Resource Provisioning. *International Conference on Performance Engineering (ICPE '13)*, 187–198, 2013.

[71] M. Hinchey, S. Park, K. Schmid. Building dynamic software product lines. *Computer*, 45(10):22–26, Oct 2012.

[72] H. Höpfner C. Bunse. Resource substitution for the realization of mobile information systems. *International Conference on Software and Data Technologies (ICSOFT '07)*, 2007.

[73] N. Huber, A. van Hoorn, A. Koziolek, F. Brosig, S. Kounev. Modeling Run-time Adaptation at the System Architecture Level in Dynamic Service-oriented Environments. *Serv. Oriented Comput. Appl.*, 8(1):73–89, March 2014.

[74] K. A. Huck, A. D. Malony, S. Shende, A. Morris. Scalable, automated performance analysis with TAUH and PerfExplorer. *Parallel Computing (ParCo '07)*, 2007.

[75] R. N. Ibbett N. P. Topham. *The Architecture of High Performance Computers*. Springer, 1982.

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

[77] M. U. Iftikhar D. Weyns. ActivFORMS: Active Formal Models for Self-adaptation. *International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '14)*, 125–134, 2014.

[78] C. Inzinger, W. Hummer, B. Satzger, P. Leitner, S. Dustdar. Generic event-based monitoring and adaptation methodology for heterogeneous distributed systems. *Software Practice and Experience*, 44(7):805–822, 2014.

[79] ISO. Software engineering – product quality – part 1: Quality model, cd 9126, 2001. http://www.iso.org/iso/catalogue_detail.htm?csnumber=22749.

[80] ISO. Software engineering-Software product Quality Requirements and Evaluation (SQuaRE) Quality model, CD 25010.2, 2011. http://www.iso.org/iso/catalogue_detail.htm?csnumber=35733.

[81] D. Ivanovic, P. Kaowichakorn, M. Carro. Towards QoS Prediction Based on Composition Structure Analysis and Probabilistic Environment Models. *Principles of Engineering Service-Oriented Systems (PESOS '13)*, 11–20, May 2013.

[82] G. Jacques-Silva, Z. Kalbarczyk, B. Gedik, H. Andrade, W. Kun-Lung, R. K. Iyer. Modeling stream processing applications for dependability evaluation. *International Conference on Dependable Systems Networks (DSN '11)*, 430–441, June 2011.

[83] M. Jarke, M. A. Jeusfeld, C. Quix, P. Vassiliadis. Architecture and quality in data warehouses: An extended repository approach. *Information Systems*, 24(3):229 – 253, 1999.

[84] S. Jarzabek, B. Yang, S. Yoeun. Addressing quality attributes in domain analysis for product lines. *IEE Proceedings - Software*, 153(2):61–73, 2006.

[85] T. Johnson D. Shasha. 2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm. *International Conference on Very Large Data Bases (VLDB '94)*, 439–450, 1994.

[86] C. Junghans, M. Karnstedt, M. Gertz. Quality-Driven Resource-Adaptive Data Stream Mining. *SIGKDD Explorations*, 13(1):72–82, 2004.

[87] C. Junghans, M. Karnstedt, M. Gertz. Quality-driven resource-adaptive data stream mining. *SIGKDD Explor. Newsl.*, 13(1):72–82, August 2011.

[88] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, A. S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21 ESD-90-TR-222, Software Engineering Institute Carnegie Mellon University, 1990.

[89] K. Kapitanova, S. H. Son, W. Kang, W.-T. Kim. Modeling and Analyzing Real-Time Data Streams. *International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC '11)*, 91–98, March 2011.

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

[91] C. H. P. Kim K. Czarnecki. Synchronizing Cardinality-Based Feature Models and their Specializations. *European Conference on Model Driven Architecture (ECMDA'05)*, 2005.

[92] A. Klein, H. H. Do, G. Hackenbroich, M. Karnstedt, W. Lehner. Representing data quality for streaming and static data. *International Conference on Data Engineering (ICDE '07) - Workshops*, 3–10, 2007.

[93] A. Klein W. Lehner. Representing Data Quality in Sensor Data Streaming Environments. *J. Data and Information Quality*, 1(2):10:1–10:28, September 2009.

[94] A. Koziolek R. Reussner. Towards a Generic Quality Optimisation Framework for Component-based System Models. *International Symposium on Component Based Software Engineering (CBSE '11)*, 103–108, 2011.

[95] H. Koziolek, J. Happe, S. Becker. Parameter Dependent Performance Specifications of Software Components. C. Hofmeister, I. Crnkovic, R. Reussner, editors, *Quality of Software Architectures*, volume 4214 of *LNCS*, 163–179, 2006.

[96] F. van der Linden, K. Schmid, E. Rommes. *Software Product Lines in Action - The Best Industrial Practice in Product Line Engineering*. Springer, 2007.

[97] M. Litoiu, M. Mihaescu, B. Solomon, D. Ionescu. Scalable Adaptive Web Services. *International Workshop on Systems development in SOA environments (SDSOA'08)*, 47–52, 2008.

[98] L. Liu L. Chi. Evolutionary Data Quality. A theory-specific view. *International Conference on Information Quality (ICIQ '02)*, 292–304, 2002.

[99] P. Lopez-Garcia, L. Darmawan, F. Bueno. A Framework for Verification and Debugging of Resource Usage Properties. *International Conference on Logic Programming*, 104–113, 2010.

[100] H. Ludwig, A. Keller, A. Dan, R. P. King, R. Franck. Web Service Level Agreement (WSLA) Language Specification, v1.0, January 2003. http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf.

[101] C. Marchetti, B. Pernici, P. Plebani. A Quality Model for Multichannel Adaptive Information Systems. *International World Wide Web Conference (WWW '04)*, 48–54, 2004.

[102] N. Megiddo D. S. Modha. ARC: A Self-Tuning, Low Overhead Replacement Cache. *Conference on File and Storage Technologies (FAST '03)*, 115–130, 2003.

[103] P. N. Mendes, C. Bizer, Z. Miklos, J.-P. Calibmonte, A. Moraru, G. Flouris. Conceptual model and best practices for high-quality metadata publishing. Technical report, 2012. PlanetData, Network of Excellence, Deliverable D2.1.

[104] A. Murashkin, M. Antkiewicz, D. Rayside, K. Czarnecki. Visualization and Exploration of Optimal Variants in Product Line Engineering. *International Software Product Line Conference (SPLC '13)*, 111–115, 2013.

[105] D. Muthig T. Patzke. Generic Implementation of Product Line Components. *Net.ObjectDays (NODE'02)*, 313–329, October 2002.

[106] F. Naumann. *Quality-driven answering for integrated information systems*, volume 2261 of *LNCS*. Springer, 2002.

[107] Inc. (OMG) Object Management Group. Unified Modeling Language, Superstructure version 2.2. Specification v2.2 formal/2009-02-02, Object Management Group, February 2009. http://www.omg.org/docs/formal/2009-02-02.pdf.

[108] Object Management Group, Inc. (OMG). Object Constraint Language. Specification v2.00 formal/2006-05-01, Object Management Group, 2006. http://www.omg.org/docs/formal/06-05-01.pdf.

[109] Object Management Group, Inc. (OMG). A UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems, version 1.1. Technical Report formal/2011-06-02, Object Management Group, June 2011. http://www.omg.org/spec/MARTE/1.1/PDF/.

[110] Object Management Group, Inc. (OMG). Structured Metrics Meta-Model (SMM). Technical Report formal/2012-01-05, Object Management Group, January 2012. Online available at: http://www.omg.org/spec/SMM/1.0/.

[111] K. Pohl, G. Böckle, F. van der Linden. *Software Product Line Engineering: Foundations, Principles, and Techniques*. Springer, 2005.

[112] R. Popescu, A. Staikopoulos, A. Brogi, P. Liu, S. Clarke. A Formalized, Taxonomy-driven Approach to Cross-layer Application Adaptation. *ACM Trans. Auton. Adapt. Syst.*, 7(1):7:1–7:30, 2012.

[113] A. J. Ramirez, B. H. C. Cheng, P. McKinley, B. E. Beckmann. Automatically Generating Adaptive Logic to Balance Non-functional Tradeoffs During Reconfiguration. *International Conference on Autonomic Computing (ICAC '10)*, 225–234, 2010.

[114] A. J. Ramirez, D. B. Knoester, B. H. C. Cheng, P. K. McKinley. Applying Genetic Algorithms to Decision Making in Autonomic Computing Systems. *International Conference on Autonomic Computing (ICAC '09)*, 97–106, 2009.

[115] A. Ranganathan, A. Riabov, O. Udrea. Mashup-based Information Retrieval for Domain Experts. *Conference on Information and Knowledge Management (CIKM '09)*, 711–720, 2009.

[116] C. T. Redman. *Data Quality for the Information Age*. Artech House, Inc., Norwood, MA, USA, 1st, 1997.

[117] M.-O. Reiser. Core Concepts of the Compositional Variability Management Framework (CVM) – A Practitioner's Guide. Technical Report 2009-16, Technische Universität Berlin, 2009.

[118] S. Roettger S. Zschaler. CQML+: Enhancements to CQML. *International Workshop on Quality of Service in Component-Based Software Engineering*, 43–56, 2003.

[119] K. Saller, M. Lochau, I. Reimund. Context-aware DSPLs: model-based runtime adaptation for resource-constrained systems. *Workshop on Dynamic Software Product Lines (DSPL '13)*, 106–113, 2013.

[120] K. Saller, S. Oster, A. Schürr, J. Schroeter, M. Lochau. Reducing feature models to improve runtime adaptivity on resource limited devices. *Workshop on Dynamic Software Product Lines (DSPL '12)*, 135–142, 2012.

[121] K. Schmid I. John. A Customizable Approach To Full-Life Cycle Variability Management. *Science of Computer Programming*, 53(3):259–284, 2004.

[122] S. Schmidt. *Quality-of-Service in Data Stream Processing*. PhD thesis, TU Dresden, 2007.

[123] S. Schneider, H. Andrade, B. Gedik, A. Biem, K.-L. Wu. Elastic scaling of data parallel operators in stream processing. *International Symposium on Parallel Distributed Processing (IPDPS '09)*, 1–12, May 2009.

[124] P.-Y. Schobbens, P. Heymans, J.-C. Trigaux. Feature Diagrams: A Survey and a Formal Semantics. *Proceedings of the 14th IEEE Requirements Engineering Conference (RE'06)*, 139–148, 2006.

[125] J. Shirazi. *Java Performance Tuning*. O'Reilly, 2003.

[126] N. Siegmund, M. Rosenmüller, C. Kästner, P. G. Giarrusso, S. Apel, S. S. Kolesnikov. Scalable Prediction of Non-functional Properties in Software Product Lines. *International Software Product Line Conference (SPLC '11)*, 160–169, 2011.

[127] N. Siegmund, M. Rosenmüller, C. Kästner, P. G. Giarrusso, S. Apel, S. S: Kolesnikov. Scalable Prediction of Non-functional Properties in Software Product Lines: Footprint and Memory Consumption. *Inf. Softw. Technol.*, 55(3):491–507, 2013.

[128] J. Skene, F. Raimondi, W. Emmerich. Service-Level Agreements for Electronic Services. *IEEE Transactions on Software Engineering*, 36(2):288–304, 2010.

[129] Software Productivity Consortium Services Corporation. *Reuse Adoption Guidebook, Version 02.00.05*, November 1993.

[130] S. Soltani. *Towards Automated Feature Model Configuration with Optimizing the Non-Functional Requirements*. PhD thesis, Simon Fraser University, Burnaby, Canada, 2012.

[131] S. Spinner, G. Casale, X. Zhu, S. Kounev. LibReDE: A Library for Resource Demand Estimation. *International Conference on Performance Engineering (ICPE '14)*, 227–228, 2014.

[132] R. M. Stallmann, R. McGrath, P. D. Smith. GNU Make - A Program for Directing Recompilation - GNU make Version 3.82, 2010. http://www.gnu.org/software/make/manual/make.pdf.

[133] D. M. Strong, Y. W. Lee, R. Y. Wang. Data Quality in Context. *Commun. ACM*, 40(5):103–110, May 1997.

[134] M. Stumptner. An overview of knowledge-based configuration. *AI Commun.*, 10(2):111–125, April 1997.

[135] M. Svahnberg, J. van Gurp, J. Bosch. A Taxonomy of Variability Realization Techniques. *Software – Practice and Experience*, 35(8):705–754, 2005.

[136] N. Tatbul. QoS-Driven Load Shedding on Data Streams. *Workshops XMLDM XML-Based Data Management and Multimedia Engineering (XMLDM '02)*, 566–576, 2002.

[137] N. Tatbul, U. Çetintemel, S. Zdonik. Staying FIT: Efficient Load Shedding Techniques for Distributed Stream Processing. *International Conference on Very Large Data Bases (VLDB '07)*, VLDB '07, 159–170, 2007.

[138] N. Tatbul S. Zdonik. Dealing with overload in distributed stream processing systems. *International Conference on Data Engineering (ICDEW '06) - Workshops*, 24–30, 2006.

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

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

[141] R. Torres, N. Bencomo, H. Astudillo. Addressing the QoS Drift in Specification Models of Self-Adaptive Service-Based Systems. *International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE '13)*, 28–34, May 2013.

[142] C. Trubiani, I. Meedeniya, V. Cortellessa, A. Aleti, L. Grunske. Model-based Performance Analysis of Software Architectures Under Uncertainty. *International Conference on Quality of Software Architectures (QoSA '13)*, 69–78, 2013.

[143] R. Y. Wang D. M. Strong. Beyond Accuracy: What Data Quality Means to Data Consumers. *J. Manage. Inf. Syst.*, 12(4):5–33, 1996.

[144] Y. Wei, V. Prasad, S. H. Son. QoS Management of Real-Time Data Stream Queries in Distributed Environments. *International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC '07)*, 241–248, 2007.

[145] Y. Wei, S. H. Son, J. A. Stankovic. RTSTREAM: real-time query processing for data streams. *International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC '06)*, 141–150, April 2006.

[146] M. Wermelinger J. L. Fiadeiro. Algebraic Software Architecture Reconfiguration. *SIGSOFT Softw. Eng. Notes*, 24(6):393–409, 1999.

[147] D. Weyns, S. Malek, J. Andersson. FORMS: Unifying Reference Model for Formal Specification of Distributed Self-adaptive Systems. *ACM Trans. Auton. Adapt. Syst.*, 7(1):8:1–8:61, 2012.

[148] T. Wood, L. Cherkasova, K. Ozonat, P. Shenoy. Profiling and Modeling Resource Usage of Virtualized Applications. *International Conference on Middleware (Middleware '08)*, 366–387, 2008.

[149] Y. Xiong, A. Hubaux, S. She, K. Czarnecki. Generating Range Fixes for Software Configuration. *International Conference on Software Engineering (ICSE '12)*, 58–68, 2012.

[150] W. Yu, W. Zhang, H. Zhao, Z. Jin. TDL: A Traceability Description Language From Feature Model to Use Case for Automated Use Case Derivation. *International Software Product Line Conference (SPLC' 14)*, 187–196, 2014.

[151] E. Yuan, S. Malek, B. Schmerl, D. Garlan, J. Gennari. Architecture-Based Self-Protecting Software Systems. *International Conference on Quality of Software Architectures (QoSA '13)*, 33–42, 2013.

[152] W. Yuan, V. Prasad, S. H. Son, J. A. Stankovic. Prediction-based qos management for real-time data streams. *International Real-Time Systems Symposium (RTSS '06)*, 344–358, Dec 2006.

[153] J. Zhang, B. H.C. Cheng, Z. Yang, P. K. McKinley. Enabling safe dynamic component-based software adaptation. *Architecting Dependable Systems III*, volume 3549 of *LNCS*, 194—211. Springer, 2005.

[154] B. T. Zivkov A. J. Smith. Disk caching in large database and timeshared systems. *International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '97)*, 184–195, 1997.

Appendix A: QualiMaster Configuration Meta Model in IVML

In this section, we present the QualiMaster Configuration Meta Model in IVML, the INDENICA variability modeling language briefly introduced in Section 7.1. For details regarding the IVML syntax and semantics, please refer to the most recent IVML specification[23]. This section is structured according to presentation of the Configuration Meta Model in Section 7.2 including a section on a configuration example.

Due to technical reasons, all constraints in the QualiMaster Configuration Meta Model are defined in terms of constraint variables although constraint variables. To support readability, we omitted the constraint variables for the model constraints in this appendix.

A.1 Basic definitions

This section depicts the basic type definitions for the QualiMaster Meta Model.

```
project Basics {

  //basic types
  typedef NaturalNumber Integer with (NaturalNumber >= 0);
  typedef PositiveInteger Integer with (PositiveInteger > 0);
  typedef NonEmptyString String with (NonEmptyString.size() > 0);
  typedef PortInteger Integer with (PortInteger > 0 and PortInteger < 65536);
  typedef NonNegativeReal Real with (NonNegativeReal >= 0);
  typedef MemorySize PositiveInteger;
  typedef Frequency PositiveInteger;

  //attribute types
  enum BindingTime {compile, startup, runtime};
  attribute BindingTime bindingTime = BindingTime.compile to Basics;

  //item field and parameter types
  enum FieldType {INTEGER, STRING, BOOLEAN, REAL, STATUS, RSSFEED};

  typedef Items sequenceOf(Item);
  typedef Parameters setOf(Parameter)
    with (Parameters->collect(p|p.name).size() == Parameters.size());
  typedef Fields sequenceOf(Field)
    with (Fields->collect(f|f.name).hasDuplicates() == false);
  typedef FieldTypes sequenceOf(FieldType);

  compound Item {
    Fields fields;
    fields.size() > 0;
  }

  compound Field {
    NonEmptyString name;
    FieldType type;
  }

  // parameters are different than fields as they may have startup values

  compound Parameter {
    NonEmptyString name;
  }

  compound IntegerParameter refines Parameter {
    assign(bindingTime = BindingTime.runtime) to {
      Integer value;
```

23 http://projects.sse.uni-hildesheim.de/easy/docs/ivml_spec.pdf

```
    }
  }

  compound StringParameter refines Parameter {
    assign(bindingTime = BindingTime.runtime) to {
      String value;
    }
  }

  compound RealParameter refines Parameter {
    assign(bindingTime = BindingTime.runtime) to {
      Real value; // this may get a "default/startup" value from the configuration
    }
  }

  compound BooleanParameter refines Parameter {
    assign(bindingTime = BindingTime.runtime) to {
      Boolean value; // this may get a "default/startup" value from the configuration
    }
  }

}
```

A.3 Observables

This section depicts the definitions of the configurable elements and related constraints for the observables explained in Section 7.2.2.

```
project Observables {

  import Basics;

  attribute BindingTime bindingTime = BindingTime.compile to Observables;

  // time-behavior
  typedef Latency NaturalNumber;
  typedef Throughput_Item Frequency;
  typedef Throughput_Volume NaturalNumber;
  typedef EnactmentDelay NaturalNumber;

  // resource utilization
  typedef UsedMemory MemorySize;
  typedef UsedMachines NaturalNumber;
  typedef AvailableMachines NaturalNumber;
  typedef Bandwidth NonNegativeReal;

  typedef Accuracy_Confidence Real;
  typedef Accuracy_ErrorRate Real;
  typedef Believability Real;
  typedef Relevancy Real;
  typedef Completeness Real;
  // MPVolatility?

  typedef Volume NonNegativeReal;
  typedef Velocity Frequency;
  typedef Volatility Real;
  typedef Variety NaturalNumber;

  abstract compound Observable {
    NonEmptyString type; // not nice
  }
  compound QualityParameter refines Observable {
  }

  compound ConfiguredQualityParameter refines QualityParameter{
    NonEmptyString monitorCls = null;
  }

  setOf(ConfiguredQualityParameter) qualityParameters = {
    {type="Latency"},
    {type="Throughput_Item"},
    {type="Throughput_Volume"},
    {type="EnactmentDelay"},
    {type="UsedMemory"},
    {type="UsedMachines"},
    {type="AvailableMachines"},
    {type="Bandwidth"},
    {type="Accuracy_Confidence"},
    {type="Accuracy_ErrorRate"},
    {type="Believability"},
    {type="Relevancy"},
    {type="Completeness"},
    {type="Volume"},
    {type="Velocity"},
    {type="Volatility"},
    {type="Variety"}
  };
  qualityParameters->collect(p|p.type).size() == qualityParameters.size();

  setOf(ConfiguredQualityParameter) configuredParameters = {};
  configuredParameters->collect(p|p.type).size() == configuredParameters.size();

  freeze {
    qualityParameters;
  }
}
```

A.3 Resource Pool (Execution Layer)

This section depicts the definitions of the configurable elements and related constraints for the resource pool characterizing the Execution Layer explained in Section 7.2.3 including the configuration for both, software and hardware based execution.

```
project Hardware {

  import Basics;
  import Observables;

  attribute BindingTime bindingTime = BindingTime.compile to Hardware;

  enum MachineRole {Manager, Worker};

  compound Machine {
    NonEmptyString name;
    MemorySize memory;
    PositiveInteger processors;
    Frequency frequency;
    setOf(PortInteger) ports;
    MachineRole role;
    //runtime variables
    assign(bindingTime = BindingTime.runtime) to {
      Bandwidth bandwidth;
    }
    // managers do not have configured ports
    role == MachineRole.Manager implies ports.isEmpty();

    // workers without configured ports receive the default Storm ports
    role == MachineRole.Worker and ports.isEmpty() implies
      ports == {6700, 6701, 6702, 6703};
  }

  setOf(Machine) machines;

  // At least one manager must be assigned
  machines->exists(Machine machine | machine.role == MachineRole.Manager);

  // At least one worker must be assigned
  machines->exists(Machine machine | machine.role == MachineRole.Worker);

..// Machine names must be unique
  machines->collect(m|m.name).size() == machines.size();

  //global runtime variables
  assign(bindingTime = BindingTime.runtime) to {
    UsedMachines usedMachines;
    AvailableMachines availableMachines;
  }

}

project ReconfigurableHardware {

  import Basics;
  import Observables;

  attribute BindingTime bindingTime = BindingTime.compile to ReconfigurableHardware;

  compound HwNode {
    NonEmptyString name;
    //runtime variables
    assign(bindingTime = BindingTime.runtime) to {
      Bandwidth bandwidth;
    }
  }

  compound MPCCNode refines HwNode{
    NonEmptyString host;
    PositiveInteger numCPUs;
    PositiveInteger numDFEs;
    //runtime variables
    assign(bindingTime = BindingTime.runtime) to {
      UsedMachines usedCPUs;
      UsedMachines usedDFEs;
      AvailableMachines availableCPUs;
      AvailableMachines availableDFEs;
    }
  }

  sequenceOf(HwNode) clusters;
..// Cluster names must be unique
  clusters->collect(h|h.name).size() == clusters.size();

}
```

A.4 Data Management

This section depicts the definitions of the configurable elements and related constraints for the data management layer explained in Section 7.2.4.

```
project DataManagement {

  import Basics;
  import Observables;

  attribute BindingTime bindingTime = BindingTime.compile to DataManagement;

  compound DataElement {
    NonEmptyString name;
    String storageLocation;
    DataManagementStrategy strategy;
    NaturalNumber timeLine;
    NaturalNumber cutoffCapacity;

    strategy <> null;
    // relate strategy, timeline and cutoffCapacity
    strategy == DataManagementStrategy.LeastFrequentlyUsed or strategy ==
      DataManagementStrategy.LeastFrequentlyRecentlyUsed implies cutoffCapacity > 0;
    strategy == DataManagementStrategy.LeastRecentlyUsed or strategy ==
      DataManagementStrategy.LeastFrequentlyRecentlyUsed implies timeLine > 0;
    strategy == DataManagementStrategy.None or strategy ==
      DataManagementStrategy.FirstInFirstOut implies
        timeLine == 0 and cutoffCapacity == 0;
  }

  compound DataSource refines DataElement {
    NonEmptyString host;
    Items input;
    Parameters parameters;
    NonEmptyString sourceCls;
    setOf(Constraint) constraints;
    //runtime variables
    assign(bindingTime = BindingTime.runtime) to {
      Velocity velocity;
      Volume volume;
      Volatility volatility;
      Accuracy_Confidence confidence;
      Completeness completeness;
    }
  }

  sequenceOf(refTo(DataSource)) dataSources;
  //unique source names required
  not(dataSources->collect(s|s.name).hasDuplicates());

  compound DataSink refines DataElement {
    Items output;
    setOf(Constraint) constraints;
    //runtime variables
    assign(bindingTime = BindingTime.runtime) to {
      Velocity velocity;
      Volume volume;
      Accuracy_Confidence confidence;
    }
  }
  sequenceOf(refTo(DataSink)) dataSinks;
  //unique sink names required
  not(dataSinks->collect(s|s.name).hasDuplicates());

  compound PersistentDataElement refines DataElement{
  }

}
```

A.5 Data Processing Algorithms

This section depicts the definitions of the configurable elements and related constraints for the data processing algorithms explained in Section 7.2.5.

```
project Algorithms {

  import Basics;
  import Observables;
  import ReconfigurableHardware;

  attribute BindingTime bindingTime = BindingTime.compile to Algorithms;

  compound Algorithm {
    NonEmptyString name;
    NonEmptyString artifact;
    Items input;
    Items output;
    Parameters parameters;
    refTo(HwNode) hwNode;
    String algTopologyClass = null;
    refTo(Algorithm) successor = null;

    //runtime variables
    assign(bindingTime = BindingTime.runtime) to {
      Latency latency;
      Throughput_Item throughputItem;
      Throughput_Volume throughputVolume;
      UsedMemory usedMemory;
      Accuracy_Confidence accuracyConfidence;
      Accuracy_ErrorRate accuracyErrorRate;
      Believability believability;
      Relevancy relevancy;
      Completeness completeness;
      Volume volume;
      Velocity velocity;
      Variety variety;
    }
  }

  setOf(refTo(Algorithm)) algorithms;
  //unique algorithm names
  algorithms->collect(a|a.name).size() == algorithms.size();
}
```

A.6 Algorithm Families

This section depicts the definitions of the configurable elements and related constraints for the algorithm families explained in Section 7.2.6.

```
project Families {

  import Basics;
  import Algorithms;

  attribute BindingTime bindingTime = BindingTime.compile to Families;

  compound Family {
    NonEmptyString name;
    Items input;
    Items output;
    Parameters parameters;
    setOf(refTo(Algorithm)) members;
    //runtime variables
    assign(bindingTime = BindingTime.runtime) to {
      refTo(Algorithm) actual;
      Latency latency;
      Throughput_Item throughputItem;
      Throughput_Volume throughputVolume;
      UsedMemory usedMemory;
      Accuracy_Confidence accuracyConfidence;
      Accuracy_ErrorRate accuracyErrorRate;
      Believability believability;
      Relevancy relevancy;
      Completeness completeness;
      Volume volume;
      Velocity velocity;
      Variety variety;

      //bind family runtime variables to actual algorithm
      latency = actual.latency;
      throughputItem = actual.throughputItem;
      throughputVolume = actual.throughputVolume;
      usedMemory = actual.usedMemory;
      accuracyConfidence = actual.accuracyConfidence;
      accuracyErrorRate = actual.accuracyErrorRate;
      believability = actual.believability;
      relevancy = actual.relevancy;
      completeness = actual.completeness;
      volume = actual.volume;
      velocity = actual.velocity;
      variety = actual.variety;
    }

    //require correct input/output item types for all members
    members->forAll(refTo(Algorithm) algorithm | input == algorithm.input);
    members->forAll(refTo(Algorithm) algorithm | output == algorithm.output);
    //require superset of parameters
    members->forAll(refTo(Algorithm) algorithm | contains(parameters,
     algorithm.parameters));
    //at least one member per family required
    members.size() > 0;
  }

  setOf(refTo(Family)) families;
  families->collect(f|f.name).size() == families.size();
  def Boolean contains(Parameters parameters1, Parameters parameters2) =
    parameters2->forAll(p2 |
      parameters1->exists(p1 | p1.name == p2.name and p1.typeOf() == p2.typeOf()));

  def sequenceOf(FieldType) collectFieldTypes(Item item) =
    item.fields->collect(Field f | f.type);

}
```

A.7 Data Processing Pipelines

This section depicts the definitions of the configurable elements and related constraints for the data processing pipelines explained in Section 7.2.7.

```
project Pipelines {

  import Basics;
  import Families;
  import DataManagement;
  import Hardware;

  attribute BindingTime bindingTime = BindingTime.compile to Pipelines;
  attribute Boolean userVisible = true to Pipelines;

  // all elements in a pipeline
  abstract compound PipelineElement {
    NonEmptyString name;
    setOf(Constraint) constraints = {}; // user constraints
  }

  // a flow among pipeline nodes
  compound Flow refines PipelineElement {
    refTo(PipelineNode) destination;
    destination.typeOf() <> Source;
  }

  // all nodes in a pipeline
  abstract compound PipelineNode refines PipelineElement{
    assign (userVisible = false) to {
      Items inputTypes;
      Items outputTypes;
    }
  }

  compound Source refines PipelineNode {
    setOf(refTo(Flow)) output;
    refTo(DataSource) source;
    inputTypes = source.input;
    outputTypes = inputTypes;

    typeCheck(self, output);
  }

  compound Sink refines PipelineNode {
    refTo(DataSink) sink;
    outputTypes = sink.output;
    inputTypes = outputTypes;
  }

  // inner node that processes something
  compound ProcessingElement refines PipelineNode {
    setOf(refTo(Flow)) output;
    typeCheck(self, output);
  }

  compound FamilyElement refines ProcessingElement {
    refTo(Family) family;
    inputTypes = family.input;
    outputTypes = family.output;
  }

  compound DataManagementElement refines ProcessingElement {
    refTo(PersistentDataElement) dataManagement;
    inputTypes = outputTypes;
  }

  abstract compound StreamOperationElement refines ProcessingElement {
    // needs to define inputTypes <-> outputTypes in refined types
  }

  compound Pipeline {
    NonEmptyString name;
    setOf(refTo(Source)) sources;
    PositiveInteger numworkers;
    PositiveInteger timeout = 100;
    //runtime variables
    assign(bindingTime = BindingTime.runtime) to {
      Latency latency;
      Throughput_Item throughputItem;
      Throughput_Volume throughputVolume;
      Accuracy_Confidence accuracyConfidence;
      Accuracy_ErrorRate accuracyErrorRate;
    }
    setOf(Constraint) constraints = {}; // user constraints
    //at least one source required
    Constraint sourcesCount = sources.size() > 0;
  }

  sequenceOf(refTo(Pipeline)) pipelines;
  //unique pipeline names
  not(pipelines->collect(p|p.name).hasDuplicates());

  def Boolean typeCheck(PipelineNode src, setOf(refTo(Flow)) output) =
    output->forAll(f|typeCheck(f.destination, src));

  //explicit propagation
  def Boolean typeCheck(PipelineNode src, PipelineNode dst) =
    if isDefined(dst.inputTypes) and isDefined(src.outputTypes)
      then src.outputTypes.overlaps(dst.inputTypes)
      else dst.inputTypes == src.outputTypes endif;
}
```

A.8 Adaptivity

This section depicts the definitions of the configurable elements and related constraints for the high-level adaptation settings explained in Section 0.

```
project Adaptivity {

  import Basics;
  import Observables;

  attribute BindingTime bindingTime = BindingTime.compile to Adaptivity;
  compound QualityParameterWeighting {
    refTo(QualityParameter) parameter;
    Real weight;
  }

  assign(bindingTime = BindingTime.runtime) to {
    Integer updateFrequency = 1000; // in milliseconds
    updateFrequency == 0 or updateFrequency > 500;

    // convention: default all weights with 1
    setOf(QualityParameterWeighting) pipelineImportance = null;
    setOf(QualityParameterWeighting) crossPipelineTradeoffs = null;
  }
}
```

A.9 Infrastructure

This section depicts the definitions of the configurable elements and related constraints for an entire QualiMaster infrastructure as explained in Section 7.2.9.

```
project Infrastructure {

  import Basics;
  import Hardware;
  import ReconfigurableHardware;
  import Families;
  import Observables;
  import Adaptivity;
  import Pipelines;

  attribute BindingTime bindingTime = BindingTime.compile to Infrastructure;

  setOf(refTo(Pipeline)) activePipelines;

    // function calculating the workers of the active pipelines
  def Integer activePipelinesNumWorkers() =
    activePipelines->apply(refTo(Pipeline) pipeline; Integer totalNumWorkers = 0 |
      totalNumWorkers = totalNumWorkers + pipeline.numworkers);

    // function calculating the (different) configured worker ports
    def Integer workerMachinesPortsCount() =
      machines->select(m|m.role=MachineRole.Worker)->apply(Machine machine;
      setOf(Integer) usedPorts = {} | usedPorts.union(machine.ports)).size();

  // number of workers for the active pipelines is less then reserved ports
  activePipelinesNumWorkers() <= workerMachinesPortsCount();
}
```

A.10 Configuration

This section shows some fragments of a QualiMaster infrastructure configuration for illustration. A configuration is structured akin to the Configuration Meta Model, i.e., in several individual projects, which are located in different physical folders in order to support role-based access control. Please note that we do not show the entire configuration for the actual QualiMaster infrastructure as it consists of more than 900 lines of IVML specification.

```
// resource pool part for software-based execution
project HardwareCfg {
  import Hardware; // import respective Meta Model part

  machines = {
    Machine {
      name="snf-618463",
      memory=1000000,
      processors=2,
      frequency=10,
      role = MachineRole.Worker
    }
    // further machines omitted
    Machine{
      name="snf-618466",
      memory=1000000,
      processors=2,
      frequency=10,
      role = MachineRole.Manager
    }
  };

  freeze { // freeze configuration for instantiation
    machines;
  }
}

// resource pool part for hardware-based execution
project ReconfigurableHardwareCfg {
  import ReconfigurableHardware; // import respective Meta Model part

  clusters = {
    MPCCNode{
      name = "MPCCCluster",
      host = "127.0.0.99:6666",
      numCPUs = 12,
      numDFEs = 4
    }
  };

  freeze { // freeze configuration for instantiation
    clusters;
  }
}


// configuration of algorithms
project AlgorithmsCfg {
  import ReconfigurableHardwareCfg; // referenced configuration elements
  import Algorithms; // import respective Meta Model part

  Algorithm aNormalize = {
    name = "normalize",
    artifact = "integration.algs.Normalize",
    output={{fields={
      Field{name="streamID", type=FieldType.STRING},
      Field{name="timestamp", type=FieldType.STRING},
      Field{name="quote", type=FieldType.REAL},
      Field{name="volume", type=FieldType.INTEGER}
    }}},
    input={{fields={Field{name="springData", type=FieldType.STRING}}}}
  };

  // further algorithms omitted

  algorithms = {refBy(aNormalize)}; // further algorithms omitted
  freeze { // freeze whole project for instantiation
    AlgorithmsCfg;
  }
}


// configuration of families
project FamiliesCfg {
  import ReconfigurableHardware; // referenced configuration elements
  import AlgorithmsCfg; // referenced configuration elements
  import Families; // import respective Meta Model part

  Family fNormalize = {
    name = "normalize",
    members = {refBy(aNormalize),refBy(hNormalize), refBy(topoNormalize)},
    actual = refBy(aNormalize), // default for startup
    output={{fields={
      Field{name="streamID", type=FieldType.STRING},
      Field{name="timestamp", type=FieldType.STRING},
      Field{name="quote", type=FieldType.REAL},
      Field{name="volume", type=FieldType.INTEGER}
    }}},
      input={{fields={Field{name="springData", type=FieldType.STRING}}}},
    parameters={BooleanParameter{name="debug"}}
  };

// further families omitted

  families = {refBy(fNormalize)}; // further families omitted

  freeze { // freeze configuration for instantiation
    FamiliesCfg;
  }
}


// configuration of the TSI hello world pipeline (in own project)
project PipelinesCfgTSI {
  import FamiliesCfg;
  import DataManagementCfg; // referenced configuration elements
  import Pipelines; // import respective Meta Model part

  // forward declarations for references
  Source src_TSI;
  Sink snk_TSI;
  FamilyElement peNormalize;
  // further variables omitted

  // define flows
  Flow f1_TSI = {
    name = "src-normalize",
    destination = refBy(peNormalize),
  };

// further flows omitted

  Flow f5_TSI = {
    name = "show-snk",
    destination = refBy(snk_TSI),
  };

  // configure pipeline nodes
  src_TSI.name = "src";
  src_TSI.output = {refBy(f1_TSI)};
  src_TSI.source = dataSources[0];

  peNormalize.name="peNormalize";
  peNormalize.family = refBy(fNormalize);
  peNormalize.output = {refBy(f2_TSI)}; // flow not shown here

  // further pipeline nodes omitted

  snk_TSI.name = "snk";
  snk_TSI.sink = dataSinks[0];

  // configure pipeline
  Pipeline pipTSI = Pipeline {
    name = "pipTSI",
    sources = {refBy(src_TSI)},
    numworkers = 1
  };

  freeze { // freeze configuration for instantiation
    PipelinesCfgTSI;
  }

}


// configuration of the available pipelines (imports pipeline projects)
project PipelinesCfg {
  import PipelinesCfgTSI; // import individual pipelines
  // further pipelines omitted

  pipelines = {refBy(pipTSI)}; // further pipelines omitted

  freeze {
    pipelines; // freeze for instantiation
  }
}

// configuration of the overall infrastructure
project InfrastructureCfg {
  import Infrastructure; // import respective Meta Model part
  import PipelinesCfg; // referenced configuration elements

  activePipelines = {pipelines[0]};

  freeze {
    activePipelines; // freeze for instantiation
  }
}
// observables and adaptivity configuration omitted here
// top-level project according to EASy conventions (as used in rt-VIL examples)
project QM {
   // import all configurations
  import InfrastructureCfg;
  import ReconfigurableHardwareCfg;
  import PipelinesCfg;
  import ObservablesCfg;
  import AdaptivityCfg;
  import HardwareCfg;
  import AlgorithmsCfg;
  import FamiliesCfg;
}
```

Appendix B: rt-VIL Grammar

In this section, we illustrate the actual grammar of rt-VIL. The grammar is given in terms of a simplified xText[24] grammar (close to ANTLR[25] or EBNF). Simplified means, that we omitted technical details used in xText to properly generate the underlying EMF model as well as trailing ";" (replaced by empty lines in order to support readability). Please note that some statement-terminating semicolons are optional in order to support various user groups each having individual background in programming languages. Further, please note that we focus here on the elements of rt-VIL and do not repeat the entire VIL grammar (9 pages), which can be found in [140].

```
ImplementationUnit:
    LanguageUnit*

LanguageUnit:
      (Advice)*
      'rtVilScript' Identifier
      '(' ParameterList? ')'
    (ScriptParentDecl)?
      '{'
      VersionStmt?
      rtContents
      '}' ';'?

rtContents:
      (
       GlobalVariableDeclaration
      | RuleDeclaration
      | StrategyDeclaration
      | TacticDeclaration
      )*

GlobalVariableDeclaration:
      'persistent'?
    VariableDeclaration

StrategyDeclaration:
      'strategy' Identifier
      '(' (ParameterList)? ')'
      '=' RuleConditions?
      '{'
      VariableDeclaration*
      ('objective' Expression)
      ('breakdown' '{' BreakdownElement* '}')
      RuleElement*
      '}'
      ';'?
```

[24] http://www.eclipse.org/Xtext/
[25] http://www.antlr.org

```
BreakdownElement:
    VariableDeclaration
      | ExpressionStatement
      | BreakdownStatement

BreakdownStatement:
      ('strategy' | 'tactic')
    LogicalExpression?
    QualifiedPrefix
      '(' ArgumentList? ')'
      (
      'with' '('
      BreakdownWithPart (',' BreakdownWithPart)
      ')'
      )?
      ('@' Expression)?

BreakdownWithPart:
      Identifier '=' Expression

TacticDeclaration:
      'tactic' Identifier
      '(' (ParameterList)? ')'
      '='RuleConditions?
      RuleElementBlock
      ';'?
```