www.qualiMaster.eu

QualiMaster

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

Grant Agreement No. 619525

Deliverable D4.2

Work-package	WP4: Quality-aware Configuration and Adaptation of Stream Processing Pipelines
Deliverable	D4.2: Quality-aware Processing Pipeline Adaptation V1
Deliverable Leader	SUH
Quality Assessor	Ekaterini Ioannou
Estimation of PM spent	29
Dissemination level	PU
Delivery date in Annex I	31.07.2015
Actual delivery date	31.07.2015
Revisions	19
Status	Final
Keywords:	Quality-aware configuration, quality-aware adaptation, enactment pattern analysis, quality taxonomy, adaptation components, adaptive crawling, event detection, tool support

Disclaimer

List of Authors

Partner Acronym	Authors
MAX	-
LUH	Patrick Siehndel, Claudia Niederée
SUH	Holger Eichelberger, Cui Qin, Roman Sizonenko
SPRING	Stefan Burkhard
TSI	Gregory Chrysos

Table of Contents

Executive Summary

In the QualiMaster project, quality-aware configuration aims at framing the configuration and adaptation space before execution of the infrastructure and the pipelines running on the infrastructure. Based on the quality-aware configuration, the quality-aware adaptation monitors and analyzes the execution and decides about changes at runtime to achieve given processing performance and result quality requirements in changing environments. To realize the vision of configurable and adaptive Big Data analysis pipelines, tailored approaches to configuration and adaptation as well as realizing components and tools are needed.

In this deliverable, we report on both, recent improvements and extensions to the concepts underlying the configuration and adaptation approach in QualiMaster as discussed in previous deliverables as well as the related tool and infrastructure components realized so far. On the conceptual side, this includes in particular a discussion of the adaptation scenarios foreseen in QualiMaster, but also the extension of the quality taxonomy, the improvement of the configuration model and the runtime instantiation language as well as results from the ongoing analysis of the enactment patterns. On the component side, we report on the actual state of the constraint reasoning used in both, configuration and adaptation, the QualiMaster infrastructure configuration tool, the runtime instantiation language, the adaptive crawling and the event detection indicating potential adaptation needs from social Web data.

1. Introduction

Quality-aware configuration and adaptation of data processing pipelines is at the heart of the QualiMaster project. Configuration aims at defining the potential customization options for the infrastructure in terms of the Configuration Meta Model as well as an actual Configuration for a certain infrastructure installation. This happens before running the infrastructure by executing the QualiMaster infrastructure instantiation process, which is driven by the respective Configuration and produces a specifically configured variant of the infrastructure including the pipelines to be executed. The configured resource pool, the available data processing algorithms, the topological structure of the configured pipelines as well as quality constraints and Service Level Agreements SLAs expressed in terms of runtime observations frame the potential adaptations of the processing at runtime. During the execution of the pipelines, the runtime observations prescribed in the Configuration Meta Model are bound by monitored values, the quality constraints and Service Level Agreements are evaluated and lead to runtime changes of the data processing through the Adaptation Layer of the QualiMaster infrastructure.

Realizing the vision of configurable and adaptive data processing pipelines requires several ingredients, ranging from data streams to be analyzed over (families of) data analysis algorithms, a Configuration Meta Model for describing the configuration of a certain analysis setting, an adaptation approach integrated with the QualiMaster infrastructure as well as realizing components and supporting tools. Basically, the challenges, requirements, concepts, approaches and some initial tooling for performing quality-aware configuration and adaptation in QualiMaster have been already presented in D4.1. In this deliverable, we discuss the states of the first version of the software components realizing the quality-aware configuration and adaptation of processing pipelines. During the course of the project, the discussions with the other partners on the components to be provided by this work package, their interaction with the results of the other work packages as well as the realization of the individual components helped us in refining and extending our initial conceptualization. As this background is important for understanding the work done so far, we will not only describe developed components and their actual realization state in this deliverable, but discuss in particular the refinement and extensions over the concepts of D4.1. As a scope for the concepts and the components, we also discuss scenarios, which capture the intended adaptation that the QualiMaster consortium aims at realizing and analyzing in the remainder of the project.

The deliverable is structured as follows: In Section 2, we summarize and prioritize scenarios describing the intended adaptation for the QualiMaster infrastructure. In Section 3, we refine the concepts discussed in D4.1 and present improvements of the Quality Taxonomy, the Configuration Meta Model, and the Adaptation Language. We also discuss the actual state of the ongoing analysis of the enactment patterns introduced in D4.1. Based on the extended and refined concepts, we discuss in Section 4 the status of the individual components realized so far in this work package, their validation as well as their envisioned integration into the QualiMaster infrastructure. There, we discuss the state of the reasoning support, the QualiMaster Infrastructure Configuration tool, the adaptive crawling and the Social Web event prediction and the adaptation language. In Section 5, we conclude this deliverable and give an outlook on future work in WP4.

Relation to other deliverables:

- This WP takes the adaptation-related requirements discussed in D1.1/D1.2 as input, refines the requirements (as done in D4.1), and realizes them as described in this deliverable.
- D2.2 utilizes the algorithm-specific aspects of the quality taxonomy introduced in D4.1 and extended in this deliverable. As some data analysis components from WP2 are used to support the adaptation, there is also a link to the algorithms described in D2.1 and D2.2.
- Akin to D4.1 and D3.1, D4.2 and D3.2 are linked through the configuration of the hardware-based processing, the integration of the hardware-based processing into the pipeline execution, the hardware-related enactment patterns and the consideration of hardware in the adaptation scenarios.
- D4.2 relies on, refines, extends and implements the concepts discussed in D4.1. In turn, D4.1 discussed specific requirements drawn from D1.1/D1.2, which are in particular valid for D4.2.
- D4.2 relies on D5.2 as components developed in WP4 and described in D4.2 are integrated into the QualiMaster infrastructure in WP5. D4.2 will indicate links to work in WP5, such as the evolution of the QualiMaster infrastructure instantiation process or early integration of WP4 components into the QualiMaster infrastructure. The actual state of the integration will be documented in D5.3 (due in the following months).
- D4.2 is also related to D6.1, in particular in terms of the application protocol that the stakeholder applications use to subscribe to and communicate with the QualiMaster application. As part of the communication, also user triggers can be sent to the QualiMaster infrastructure to indicate user requests regarding the actual processing, which can lead to adaptations of the running pipelines.

2. Adaptation Scenarios

In this section, we discuss the relevant adaptation scenarios for the QualiMaster infrastructure. We elicited these scenarios from all partners of the QualiMaster consortium, in particular the industrial partners, in terms of bilateral or group discussions and initially collected them in a shared online scenario document. The adaptation scenarios refine the actual adaptation requirements to be covered by the QualiMaster adaptation approach (D4.1) and its realization. However, the set of potential adaptation scenarios encompasses five scenarios, each with several variants, so that we ultimately end up with more than 40 individual scenarios. Although we aim at realizing as many (diverse) scenarios as possible, it is necessary to prioritize them in order to focus on the most relevant ones first, also in synchronization with the actual realization state of the data processing algorithms (WP2), the realization of hardware-based execution WP3, the adaptation components discussed in this deliverable, the infrastructure (WP5), the pipelines and the stakeholder applications (WP6),

In this section, we first discuss in Section 2.1 a template for describing adaptation scenarios. In Section 2.2, we apply the template for describing the elicited adaptation scenarios. Finally, in Section 2.3 we prioritize the scenarios for realization and evaluation.

2.1 Adaptation Scenario Template

We utilize a template to provide a systematic description of the adaptation scenarios we elicited. As collecting adaptation scenarios is a specific form of collecting requirements, we rely pragmatically on an extended kind of use case template [7], inspired by the one that we already applied in D1.1 and D1.2.

An Adaptation Scenario is described using the following structure:

- **Identification:** Basically, each scenario receives a unique identification and, thus, the following sections of this deliverable, as well as other deliverables, can make unique references to them.
- **Setting:** An adaptation scenario describes the setting in which the desired or expected adaptation happens.
- **Trigger:** A scenario is caused by a certain (internal or external) trigger, which we will categorize in terms of the adaptation triggers we already introduced in D4.1. We repeat them here for the convenience of the reader.
    - 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 a configured update frequency. This trigger causes either an adaptation or, in case that no enactment is needed, it may cause an update of the common adaptation 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 start-up 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, which shall be considered of a lower priority than the other triggers.

- **Adaptation paths:** A scenario describes one or multiple potential adaptation paths (inspired by the main scenario and exceptions in use case templates). An adaptation path is given in terms of
  - *Cause* of this path, which may also refer to a previously executed adaptation path.
  - *Possible / expected effects* on one or multiple quality parameters. We indicate an increase of the respective parameter with (+), a decrease with (-) and a parameter on the same level as before with (~).
  - *Possible follow-ups*, i.e., adaptations leading to further adaptation paths in the same or other adaptation scenarios.
- **Variants:** For similar scenarios, we describe a representation specific scenario and indicate potential variants rather than elaborating all potential scenario variants. We also present only one "direction" of a scenario, e.g., the increase of the volume of an input data stream, but we consider the counter direction as an (implicit) variant, here the decrease of the volume after some time. The counter scenario considers similar actions as for the given scenario but supporting the counter direction of the adaptation.

Table 1 illustrates an empty adaptation scenario template. We apply this template and discuss the potential adaptation scenarios in the next section.

Identifier			
Name			
Setting			
Trigger			
Adaptation paths	**Cause**	**Possible effect**	**Possible follow-ups**
Variants			

Table 1: Template for describing adaptation scenarios.

2.2 QualiMaster Adaptation Scenarios

In this section, we discuss the potential adaptation scenarios that we collected from the partners of the QualiMaster project. We categorized the individual scenarios as follows:

- **Changing data streams:** If a stream quality parameter, such as volume or velocity changes, an adaptation of the data processing may be needed.
- **Detection of Social Web events:** Social Web event detection indicates actual or upcoming (near-future) events, which may change the data or the quality properties of the data streams. Actual events may lead to a changed focus of the processing while

near-future events can allow changing that processing before the actual adaptation trigger occurs at lower efforts or impacts.

- **User triggered adaptations:** The user requests a change of the processing through the stakeholder applications, e.g., the number of market players considered in the calculation, the analysis focus, or the precision of the computation.
- **Processing errors:** If a physical processing resource in the execution layer fails, e.g., a Storm node dies, certain recovery actions must be carried out to sustain the processing.
- **Requested resource reallocation:** An infrastructure administrator explicitly requests a change in the resource allocation, e.g., a pipeline is started or stopped and potentially other pipelines must be adjusted to this new situation.

We structure this section according to the categorization given above and detail the individual scenarios in terms of the template presented in Section 2.1. Furthermore, we give some concluding remarks on propagation of adaptations in Section 2.2.6.

2.2.1 Changing Data Streams

A typical situation in data stream processing is that the characteristics of the data stream change over time due to external reasons, e.g., the volume increases or the volatility decreases. This requires actions in particular to avoid overload conditions of the infrastructure or negative effects on other pipelines.

<table>
<tr><td>Identifier</td><td colspan="3">A-1</td></tr>
<tr><td>Name</td><td colspan="3">Changing Data Streams</td></tr>
<tr><td>Setting</td><td colspan="3">An oil platform sinks and the oil market becomes hectic. This leads to an increased volume of that specific sector (possibly also of related sectors). The impact on the respective QualiMaster pipeline depends on whether the affected markets are actually relevant to the analysis. If the market segments are relevant, the volume in the pipeline will also increase (this scenario will cover that specific setting). At the same time quality measures might increase, since there is an issue with this market sector and the user wants to observe it more closely.</td></tr>
<tr><td>Trigger</td><td colspan="3">External factor (here, data volume increase)</td></tr>
<tr><td rowspan="3">Adaptation paths</td><td>Cause</td><td>Possible effect</td><td>Possible follow-ups</td></tr>
<tr><td>Monitoring detects sustained increasing volume</td><td>• Data volume (+)
• CPU load (+)
• Computation time (+)</td><td>• Change algorithms to reduce load
• Allocate more computational resources
• Input load shedding</td></tr>
<tr><td>Change algorithms to sampling (may have large impact on data quality)</td><td>• Data quality (-)</td><td>• Change further algorithms to reduce load
• Change subsequent algorithms or algorithm parameters to improve resource quality without affecting output data quality (if possible at all)
• Allocate more computational resources
• Input load shedding</td></tr>
</table>

	Change algorithms or algorithm parameters	• CPU load (-) • Computation time (-) • Data quality (-)	• Allocate more computational resources • Input load shedding
	Allocate more computational resources, e.g., reassign resources	• CPU load (-) • Computation time (-)	• Input load shedding
	Input load shedding (throw away data) to prevent overload	• Data quality (-) • CPU load (-) • Computation time (-)	
Variants	Other inherent Big Data stream characteristics, in particular volatility (frequency of items) may change. Also the distribution of the data itself may change and, thus, cause imbalances in the pipeline, which can ultimately lead to similar adaptation paths.		

2.2.2 Detection of Social Web Events

The future event detection and prediction algorithms are used to find events and related dates that may happen in the future. By analyzing the stream of messages as well as news sources the mentions certain dates together with related entities and keywords can lead to a prediction of an event in the future. This prediction can be used to inform the user, notify the Adaptation Layer about an upcoming event or automatically request a change of the pipeline to adopt certain settings to the upcoming event. In contrast to the other adaptation changes, the (mid- and long-term) detection of future events can lead to the scheduling of adaptations in the future (proactive) as well as the notification of a certain adaptation time window, which may trigger a counter adaptation after the expected event happened.

Identifier	**A-2**		
Name	Detection of Future Events		
Setting	The event prediction algorithms collect several messages indicating that an important election of CEOs will take place in the near future. The related sector and company is of interest for the analysis so the pipeline needs to be adopted to focus on this event and possible side events. We can expect an increase in the number of related messages when the event occurs, additionally the stream may contain new terms related to the involved persons.		
Trigger	External trigger (increase of time mentions related to a certain topic / stock)		
Adaptation paths	**Cause**	**Possible effect**	**Possible follow-ups**
	Event prediction indicates important event related to company / sector of interest (mid- or long-term forecast). Increase of messages related to the specific company	The event may lead to • Data volume (+) • CPU load (+) • Computation time (+) but without prediction the information about a certain event and its side effect may be lost in data analysis and, thus affect the coverage quality parameter.	• Prepare for change of Algorithm (A-1) • Adopt stream to focus on event related content • Select algorithm for the specific kind of event • Reduce the aggregation level, e.g., from hourly to a minute-based aggregation leading to a parameter

			change (A-1)
	Prediction algorithms detect new relevant keywords / entities (short-term forecast). Adaptation of stream processing to event related messages can lead to a change of message types, (e.g. more similar messages), several messages related to the same entities.	Basically, such short-term predictions may affect the infrastructure • Data volume (+) • CPU load (+) • Computation time (+) but may also indicate a reason for focusing the analysis and, then, improve different data quality parameters.	• If possible, algorithms can adapt to the new type of messages, e.g., Named Entity Recognition (NER) puts focus on persons instead of locations. • Change the input stream filtering to adapt to new keywords / entities to capture also ongoing discussions and emerging topics.
Variants	Different types of events may lead to different adaptation paths and may require different adaptations. Some events can cause a slow increase in the number of related events, while other events just produce a short peak. Priming events may lead to several adaptations also related to the collection of other relevant content, while less influential events may be dealt with without an influence to the collection of other data. The type of the topic will also influence how the preparation for the future event looks like. This adaptation scenario can utilize different forms of focusing the data analysis based on detected events such as reducing an aggregation level (e.g., from hours to minutes) or increasing the neighbour level to consider also further related entities such as market players.		

2.2.3 User Triggered Adaptations

In this scenario, the user explicitly requests a change in processing by modifying some settings in the stakeholder application. To some degree, it must not necessarily be obvious to the application user that such a modification may cause a change of the processing. For example, even patterns in the interaction of the user with the stakeholder application may implicitly lead to such a request. However, as other more urgent adaptations may lead to goal or enactment conflicts, we consider user triggers of lower priority during the adaptation and even ignore or reject user triggers. For example, the Adaptation Layer may currently perform a change of the processing to reduce the actually resource utilization in order to speed up another pipeline due to an internal trigger issued by the Monitoring Layer, while a user trigger requests computation at higher precision requiring more resources. In this case, the user trigger would be rejected.

<table>
<tr><td>Identifier</td><td colspan="3">A-3</td></tr>
<tr><td>Name</td><td colspan="3">User triggered adaptation</td></tr>
<tr><td>Setting</td><td colspan="3">An institutional user wants to increase the number of market players in his/her analysis. This might, for example, be the effect of some correlation he/she has detected. In order to achieve the goal, the user wants to relax the input filter of a QualiMaster pipeline.</td></tr>
<tr><td>Trigger</td><td colspan="3">Application User</td></tr>
<tr><td rowspan="5">Adaptation paths</td><td>Cause</td><td>Possible effect</td><td>Possible follow-ups</td></tr>
<tr><td>Change source parameter</td><td>• Data volume (+)
• CPU load (+)
• Computation time (+)</td><td>• Adjustment of Twitter Selection required
• More CPU load on subsequent data processing elements</td></tr>
<tr><td>Change Twitter stream selection parameter</td><td>• Data volume (+)
• CPU load (+)
• Computation time (+)
• Latency (+)</td><td>• More CPU load on subsequent data processing elements
• Change of algorithms / parameters to reduce load (if possible without impact on data quality - if such algorithms exist)</td></tr>
<tr><td>Change algorithm / parameter</td><td>• CPU load (-)
• Data quality (-)</td><td>• Subsequent changes of algorithms / parameters</td></tr>
<tr><td>Add set of keywords to streaming algorithms</td><td>• Data volume (+)
• CPU load (+)
• Computation time (+)
• Latency (+)</td><td>• Structure of stream may change if the messages related to the new keywords are the majority in the new stream.</td></tr>
<tr><td>Variants</td><td colspan="3">• Modify the subscriptions to data sources. This leads to similar adaptation paths, but may impact other pipelines sharing the same source.
• Change of the computation approach, e.g. to suggest a change of the correlation computation (advanced user level).
• Change of the processing quality, e.g., accuracy. In particular, this includes increases but also relaxation of the processing quality, which may also lead to a “cheaper” computation.
• Modify how broad or narrow the stream is, i.e., to change the input stream filtering for certain keywords.
• Focus the analysis on certain keywords / entities / hashtags / symbols related to the sector or domain of interest or change the aggregation level. This user trigger is an immediate form of the (scheduled) triggers discussed in A-2.</td></tr>
</table>

2.2.4 Processing Errors

Sometimes, errors in the Execution Layer can disturb the intended processing. Basically, a fault such as a programming error in an algorithm, the pipeline, the infrastructure or the Execution Systems can cause a processing resource to fail, in some cases just to be temporarily unavailable. This may even be the case for the hardware-based processing, in particular if the connection with the software side is interrupted or not properly closed so that a hardware processing unit may actually be free for processing but still appears to be allocated. A mitigation of this case can be of lower impact if the respective algorithm already runs in parallel. However, also the enactment of an adaptation may fail so that a reconfiguration of the QualiMaster infrastructure excluding the failing configuration shall be calculated.

Although Algorithm Providers as well as the Consortium aim at avoiding programming errors by intensive tests and validations, they may appear anyway. Therefore, the partners decided to build a default value mechanism into the generated pipelines, so that if an algorithm cannot go on processing, at least other parts of the pipeline can continue analyzing data. In particular, in Apache Storm an individual Worker may die due to a programming error in the algorithm or the Execution System (here Storm, see D5.1 for details). Typically, the administrative setup allows the operating system to observe the execution state of related processes (such as the Supervisor processes in Storm) and perform a restart of the process if they fails. However, in adaptive pipeline execution the actual algorithm must be reassigned as a specific form of (recovery) adaptation. Subsequently, resources and processing tasks may need to be migrated within the cluster in order to mitigate this situation.

Identifier	**A-4**		
Name	Processing Error		
Setting	Data processing in the Execution Layer fails due to a bug in an algorithm.		
Trigger	Internal		
Adaptation paths	**Cause**	**Possible effect**	**Possible follow-ups**
	Algorithm detects a processing problem	• CPU load (-) • Computation time (-) • Latency (-) • Data quality (-)	• Continue with default values • Processing node dies • Restart processing • Change resource allocation (A-5) • Change algorithm (reassign task)
	Continue with default values	• CPU load (-) • Computation time (-) • Latency (-) • Data quality (-)	• Restart (of selected) processing elements
	Processing node dies	• Resource availability (-)	• Change algorithm (reassign task) • Change resource allocation (A-5)
	Restart processing	• Resource availability (-)	• Change algorithm (reassign task)

	Change resource allocation (A-5)	• CPU load (+) • Computation time (+) • Latency (+) • Data quality (+)	• Change algorithm • Restart processing
	Change algorithm (reassign task)	• CPU load (+) • Computation time (+) • Latency (+) • Data quality (+)	• Change resource allocation
Variants	Processing fails due to errors in an Execution System or due to problems enacting an adaptation.		

2.2.5 Requested Resource Reallocation

Administrative tasks on the QualiMaster infrastructure can request a change in the resource reallocation. Actually, this may imply a (re)configuration of the resource pool.

- If the resource pool changes, suddenly more or less resources such as servers or hardware processing units can become available.
- If the resource pool does not change, the resource consumers can change. For example, if the Infrastructure Administrator requests to start or stop a pipeline, the resources of other running pipelines may need to be considered for reallocation in order to optimize the entire compute cluster for the new situation.

The new resources can be of the same or a different quality than actually allocated resources so that quality parameters can change accordingly. Actually, administrative changes can be announced or requested by a distinct trigger created by the administrator, e.g., which pipeline to start / stop or which resources to change and how. In comparison to user triggers (A-3), administrative triggers shall be considered of a higher priority by the QualiMaster infrastructure.

Identifier	**A-5**		
Name	Requested Resource Reallocation		
Setting	The administrator requests the start-up of a new pipeline.		
Trigger	Infrastructure Administrator		
Adaptation paths	**Cause**	**Possible effect**	**Possible follow-ups**
	Request to start a new pipeline.	• Data volume (+) • CPU load (+) • Computation time (+)	• The pipeline fits into the resource allocation of the cluster so no adaptation is needed. • Running pipelines must be adapted by changing the resource allocation or algorithms / parameters.
	Change resource allocation	• CPU load (~) • Data quality (~)	• Subsequent changes of algorithms / parameters

	Change algorithm / parameter	• CPU load (-) • Data quality (-)	• Subsequent changes of algorithms / parameters
Variants	Alternatives are: stop pipeline, add new resources, disable existing resources, and in the extreme case also stop less important pipelines.		

2.2.6 Propagation of adaptation effects

In a multi pipeline setting (DoW task T4.4) an adaptation of one pipeline can lead to subsequent adaptations in other running pipelines (cross-pipeline adaptation). For example, as already mentioned in Section 2.2.5, starting a pipeline may lead to a resource shift among the pipelines or even a reallocation of the hardware-based processing. Furthermore, resource conflicts can lead to a relaxation of the current resource allocation within the modelled service level agreements in order to satisfy resources required by other pipelines. Similar cases may also occur as subsequent actions in a single pipeline, so that one adaptation causes further reactive adaptations or scheduled / planned adaptations along the pipeline (called wavefront adaptation in D4.1).

2.3 Prioritization of the scenarios

In Section 2.3, we presented 5 scenarios and more than 40 scenario variations. Please note that in addition to the mentioned variants for each variant also the implicit counter direction applies. Although we aim at realizing, testing and evaluating as many relevant scenarios as possible, we prioritized the discussed scenarios based on discussion with the partners. This led to the following prioritization sequence.

1. Changing data streams (A-1): For the typical adaptation scenario in data stream processing, we focus first on volume changes of the input data streams, and as secondary scenario changes to the variety.
2. Requested resource reallocation (A-5): Here we focus on starting and stopping pipelines in the single as well as in the cross-pipeline case, in particular to utilize hardware-based processing adequately. Actually, a simplified form of start-up adaptation is already required if the initial algorithms of a pipeline shall be assigned dynamically. Currently, we do not plan to realize further variants, as they are combinations of A-1 and A-3 at higher adaptation priority than A-3, but explicitly triggered by QualiMaster tools such as the QualiMaster infrastructure configuration tool (QM-IConf[1]).
3. We also aim at one specific variant of future event detection and adaptive filtering (A-2) as well as selected user triggers (A-3), which requires the integration of the (extended) stakeholder application protocol discussed in D6.1. We assigned here a lower priority level due to the ongoing research, implementation and integration work of the related components
4. Processing error (A-4): We focus on sustaining the processing when compute resources die unexpectedly as well as switching to default values to support the development of pipelines. For the beginning, we concentrate on working pipelines without exceptional cases and take counter measures for processing errors into account during later stages of the project.

If not already done or in progress (as for A-1 and A-5), we plan to realize for each scenario the respective enactment mechanisms, the respective monitoring support (in collaboration with WP2, WP3 and WP5), the adaptive analysis and planning as well as the integration into the top-

[1] In the meantime we decided to change the acronym of the QualiMaster Infrastructure Configuration tool from QM-IC used in D1.2 and D4.1 to QM-IConf.

level adaptation control-script (see D4.1 for more details). For each realized scenario (according to the prioritization we aim at 7 individual scenarios), we will validate the functionality and perform quantitative experiments to characterize the impact and the benefits of the adaptation (as also suggested by the advisory board).

3. Concepts and Design

Since the completion of D4.1, we updated, improved and realized several of the concepts presented in D4.1, considering also the feedback given by the consortium, the reviewers and the advisors. Further, we developed additional concepts. In this section, we present an update of the concepts from D4.1 as well as new concepts and follow the logical sequence used in D4.1, i.e., from the quality taxonomy over quality-aware configuration to adaptation. In particular, we discuss updates for the following:

- QualiMaster Quality Taxonomy (Section 3.1)
- QualiMaster Configuration Meta Model (Section 3.2)
- Adaptation Enactment Patterns (Section 3.3)
- Adaptation Specification Language (Section 3.4)

3.1 Quality Taxonomy

The QualiMaster Quality Taxonomy collects and categorizes the relevant quality dimensions for the project. As discussed in D4.1, we constructed the taxonomy from an explicit survey in the consortium. Basically, the quality taxonomy follows the terminology and structure of ISO/IEC 25010 [20][2] with a specific extension towards Big Data quality parameters in the Scalability dimension. As also indicated in D4.1, the QualiMaster Quality Taxonomy is intended to be refined and extended if needed.

We decided to extend the taxonomy from D4.1 in two ways. One extension to the taxonomy originates in the suggestions from the first year review, in particular to also take novelty, diversification or user interaction into account. A second extension is triggered by the practical realization of adaptation scenarios, in particular to adjust and to optimize the parallelization of pipelines in order to respond to changing stream characteristics (A-1) as well as requested resource changes (A-2, A-5) in terms of pipeline start-up and shutdown.

Figure 1 illustrates the updated Quality Taxonomy. Actually, we extended two dimensions, namely resource utilization and functional suitability as we explain below.

In the **resource utilization** sub-dimension we added parameters supporting the analysis of the parallelism of a pipeline. These are:

- *Capacity:* The actual utilization of a pipeline node in (0; 1). Higher values indicate an overloaded pipeline node. The capacity can be computed by $capacity = \frac{\#executed \cdot latency}{1000 \cdot time}$ and corresponds to the capacity of a Storm Bolt[3]. Similarly, the average capacity of an entire pipeline can be calculated.
- *Parallelization:* For each pipeline node we consider the actually used number of executing units, e.g., for software-based processing the number of JVMs realizing Storm workers.

In the **algorithm suitability** sub-dimension, we consider some additional quality parameters, also to take up the review recommendations. These are:

- *Novelty:* The novelty dimension plays an important role for Information Retrieval (IR) systems [28] and is also of high relevance for several algorithms within the QualiMaster Infrastructure. Basically all algorithms or methods using stored Information can

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

[3] Technically, Storm Spouts do not define the measures to calculate the capacity. Similarly, the Storm measurements form Bolts acting as sinks in a pipeline lead to a capacity of 0. Thus, we obtain realize the capacity based on own measurements.

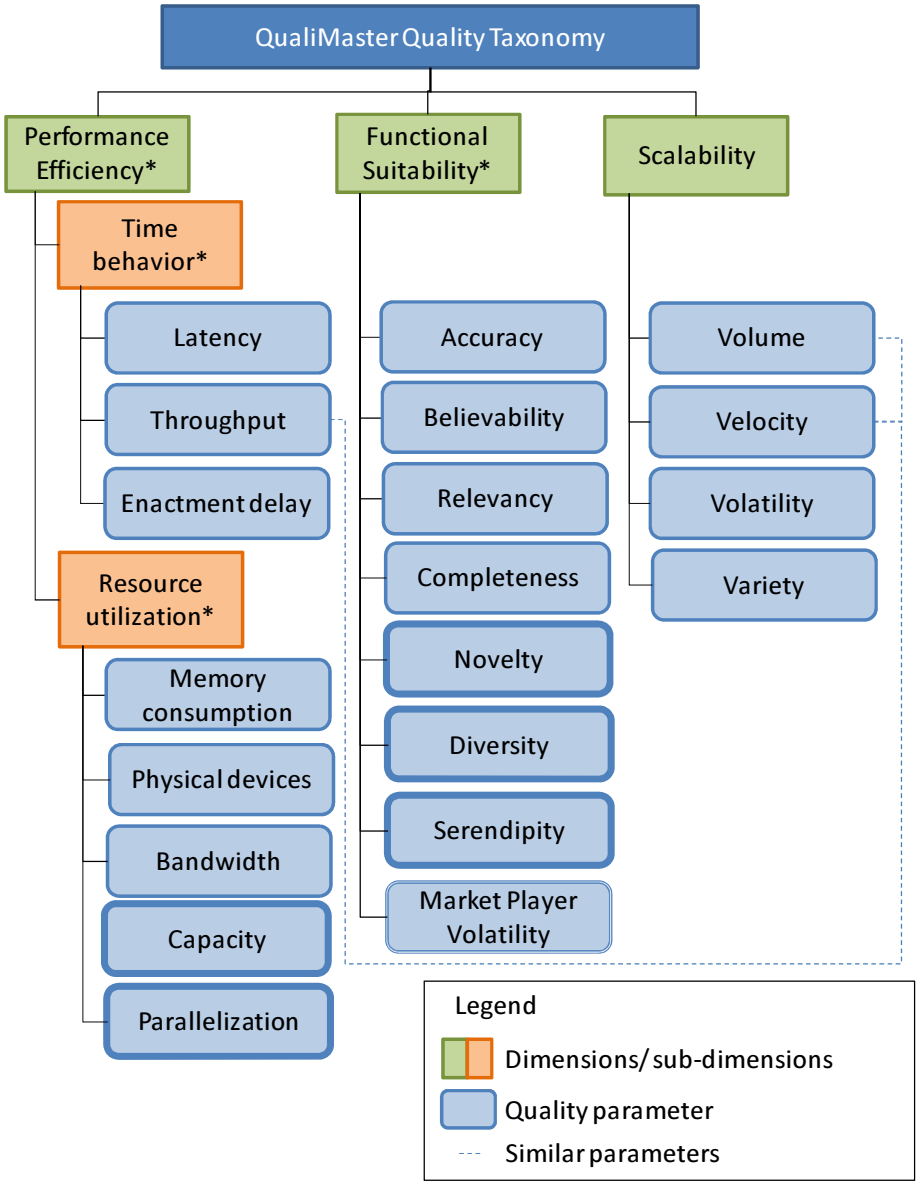

Figure 1: Updated Quality Taxonomy (changes marked by a thicker border).

downgrade the relevance of old data in various ways. So novelty can be measured by analyzing to which extend old data is used, and how old this data is. For instance, when generating graphs on the relation between stocks the temporal change might be relative important. Also the size of sliding windows or the period in which recalculations are performed influences the novelty of the results. Additionally the change of elements can be interpreted as novelty [9] relating this metric to anomaly detection. One scenario where novelty might be considered is the visualization of relations or graphs, were new patterns might be of special interest. Tuning the algorithms or methods towards novelty can interfere with measures like relevance or completeness, experimental evaluations will show which setup is optimal in which scenario.

- *Diversity:* Diversification aims at showing different facets of a result set [10]. For instance in the area of IR the diversification of the search results makes sure that if the result set contains elements belonging to different categories, for all of these categories some elements are shown [1]. Similar measures can be used in the area of book

recommendations [31]. In QualiMaster we can measure the diversity for several algorithms or methods, e.g., stocks are grouped into different industry sectors, when monitoring a specific domain, a user may also want to see how other elements out of this domain are influenced, thus a diversification of the monitored elements may lead to a higher user satisfaction.

- *Serendipity:* Serendipity is defined in the literature as a measure indicating how surprising results returned for a user query by an information retrieval system are. In many cases users are interested not only in relevant and diverse, but also in surprising content [22]. In our work we are planning to incorporate this measure for evaluating our algorithms in terms of surprising output. A deeper analysis of the relations within our graphs (similar to those described in literature [4], see also D2.1) might lead to interesting results which can be highlighted for the user.

One further potential influence to the adaptation and the processing quality could be the interaction of the user with the stakeholder applications, as one can observe, e.g., in terms of the clicks a user performs. On the one side, the stakeholder applications are outside the scope of the infrastructure as they act as clients, i.e., they subscribe to one or multiple pipeline sinks and display the analyzed data or perform user-specific post-processing of the received data. Thus, we do not consider such user values as direct quality parameters of the QualiMaster infrastructure. On the other side, the stakeholder applications are able to communicate user requests in terms of user triggers (adaptation scenario A-3) to the QualiMaster infrastructure via the application protocol described in D6.1. Thus, the user intentions that may be derived from the user clicks could be observed by the stakeholder applications and, if significant and relevant, turned into user triggers for the adaptation.

3.2 Configuration Meta Model

In deliverable D4.1, we introduced the concepts and design of the QualiMaster Configuration Meta Model and detailed specific configurations of the QualiMaster infrastructure. In particular, we discussed the QualiMaster Configuration Meta Model in terms of individual modules (subsections in D4.1), specifying the introduced configurable elements as well as their interrelations in terms of typed references and constraints. The Configuration Meta Model allows the domain user to configure the resource pool supporting software- and hardware-based execution (D4.1, Section 7.2.3), Data Management entities, in particular data sources and sinks (D4.1, Section 7.2.4) and individual Data Processing Algorithms (D4.1, Section 7.2.5) as a foundation for composing data processing pipelines. At its heart, the Configuration Meta Model supports the topological configuration of Data Processing Pipelines (D4.1, Section 7.2.7) consisting of data sources and sinks, data management elements for storing intermediary data, Algorithm Families grouping algorithms of the same functionality (D4.1, Section 7.2.6), and interconnecting data flows. To enable adaptation, the Configuration Meta model supports the definition of Observables for monitoring run-time properties through quality parameters (D4.1, Section 7.2.2), quality and SLA constraints as well as high-level Adaptation Settings allowing the domain user to control certain aspects of the run-time adaptation (D4.1, Section 7.2.8). For defining the Configuration Meta Model as well as individual configurations, we use the Integrated Variability Modelling Language (IVML) [11, 13, 14, 19][4].

As indicated in D4.1, the Configuration Meta Model can be extended to reflect additional configuration knowledge, in particular, to meet new or changed requirements drawn by the research work in the QualiMaster project or requested by Infrastructure Users. In this section, we discuss the most important changes to the Configuration Meta Model over the initial version described in D4.1. We detail the changes by first describing the previous state given in D4.1,

[4] Historically, the “I” in IVML indicated the FP7 project INDENICA. In the mean time we decided to change the name to indicate its wider application scope.

discussing then the identified issues indicating the need for a change, and, finally, introducing the changes that we made. Actually, we focus in this section on the modifications and assume that the reader is already familiar with the concepts of the Configuration Meta Model and the underlying modeling language IVML. For further details, please refer to D4.1.

The main modifications to the Configuration Meta Model are:

- Support for **arbitrary field types** in the input / output items defining the data flow at individual pipeline elements, such as sources, sinks or processing elements. This allows the Pipeline Designer / Algorithm Provider to work with domain specific types rather than a limited set of predefined types.
- User-defined **item names** support the Algorithm Provider in semantically linking the configuration with the derived elements in the implementation, such as the item or algorithm interfaces.
- **Textual description** of algorithms providing additional descriptive information for supporting reuse by a search functionality (as mentioned in D5.2).
- Technical **access to the hardware-based execution**, e.g., network ports to monitor the available / used FPGAs or to change the hardware-based algorithm being executed.
- **Artefact information** enabling the automated integration and execution of pipelines and the infrastructure. Here, we allow to configure the location of the Pipeline Elements Repository and to define artefact specifiers for deploying individual pipelines as well as the entire Configuration (Meta)[5] Model as an artefact.
- **Development support** in terms of a debug mode, which allows optimizing pipelines for production, in particular to disable logging, which can impact the pipeline performance.

Below, we will now discuss each modification in the sequence given above. Finally, we conclude with potential future changes in Section 3.2.7. Please note that we changed the Infrastructure Instantiation Process discussed in D5.2 accordingly to reflect the new capabilities of the Configuration Model in the derived / generated artefacts.

3.2.1 Arbitrary Field Types

As described in D4.1, we use the IVML compound type `Item` (defined in the IVML module `Basics`) to specify the items of a data stream handled by data sources, data sinks, algorithms and algorithm families. Thereby, we use the type `FieldType` to characterize the type of individual fields in the `Item`. In the previous design of the Configuration Meta Model, `FieldType` has been defined as an enumeration denoting only basic types such as `Integer`, `String`, or `Boolean`. Recently, the partners acting as Algorithm Providers identified an increased need to pass domain-specific objects through a pipeline, such as an object representing a Twitter feed. Initially, this led to the inclusion of the type `Object` into `FieldType` enumeration, inspired by the respective class in Java. Although `Object` allowed us to defined pipelines on arbitrary domain-specific types, it also imposed limitations. On the one side, due to the generality of `Object` (it complies with every type), the static pipeline analysis defined through configuration constraints was not able to detect whether a pipeline is (fully) composed correctly. On the other side, the `Object` type required the Algorithm Provider to cast individual tuple fields back to the expected type, leaving important aspects of pipeline (type) consistency in the hands of the Algorithm Provider.
Thus, in discussion with WP2, we decided to extend the notion of the `FieldType` in the Configuration Meta Model, allowing the configuration of arbitrary types provided by algorithm implementations. The new design of the related parts is illustrated in Figure 2. In contrast to the

[5] Akin to D4.1, we refer with Configuration (Meta) Model to both, the definition of the types and constraints in the Configuration Meta Model as well as a particular Configuration.

initial version of the Configuration Meta Model, where we defined `FieldType` as an enumeration, we reconstructed it now as a compound containing configurable elements for capturing enough information of the specific type to generate, integrate and build actual as well as future versions of QualiMaster pipelines. Consequently, all uses of `FieldType` became typed references to the (singleton, shared) type definitions given in the configuration. To maintain compatibility with the initial version of the Configuration Meta Model, we modeled the already known (enumeration-based) types in terms of compound variables, defined them as built-in types, and changed the existing configuration accordingly. Moreover, defining new types or (carefully) changing existing types belongs now to the responsibilities of the Infrastructure Administrator (potentially on behalf of or in collaboration with the Pipeline Designer).
As indicated in Figure 2, a `FieldType` can be detailed by five slots, which we discuss now. Basically, a `FieldType` receives a unique name so that it can be identified and selected by a domain user. The slot `class` must be configured with the fully qualified class name of the implementing type, actually a Java class. If the implementing type is part of the default Java libraries, no specific `artifact` containing the implementation needs to be configured. If the type is domain- or application specific, `artifact` contains the Maven artefact specification of the providing artefact. This is required to derive the correct Maven build specifications for the interfaces and the pipelines. Due to the distributed nature of data processing in Apache Storm, data types that shall be passed along with the data stream must be serializable, i.e., Storm must be able to turn an instance into a form that can be sent through and received from network connections. Basically, one can rely on the easy-to-apply default Java serialization (the specific class and its fields must be serializable, i.e., at least transitively implement the Java interface `Serializable`, or excluded from serialization). To achieve better performance (typically lower latency and higher throughput), the use of the serialization mechanism built into Storm, namely kryo[6], is recommended. To enable kryo serialization, the data classes must provide type-specific serialization code, either within specific methods of the class (akin to manual serialization in the

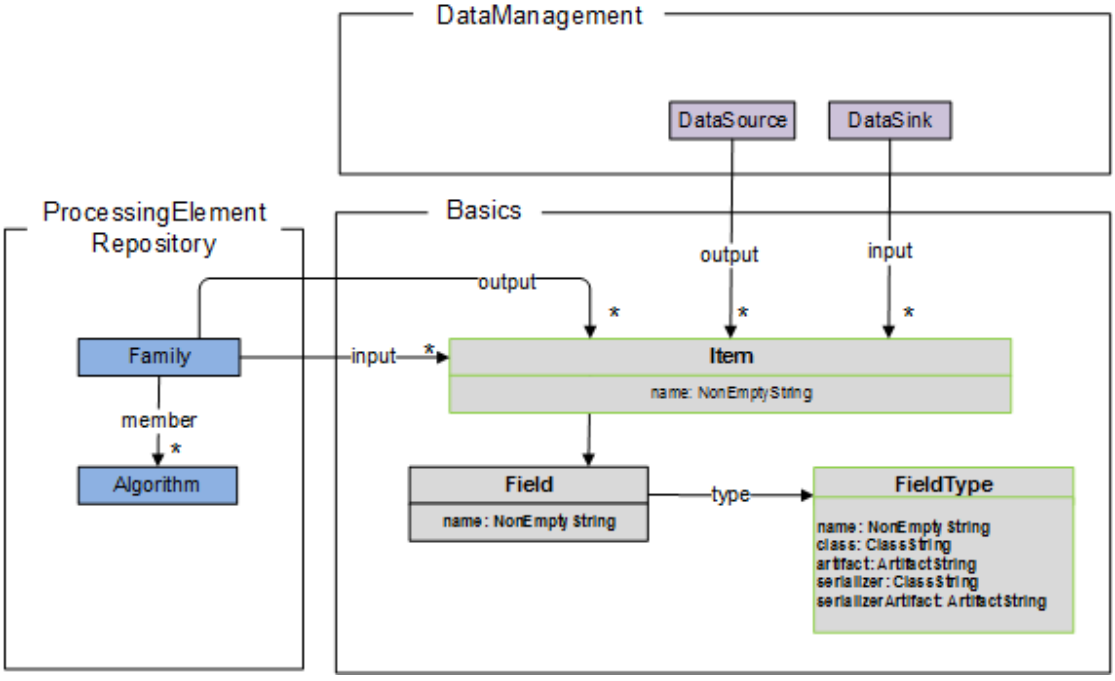

Figure 2: Updated Basics module of the Configuration Meta Model showing the configurable `FieldType` and its relations.

[6] http://code.google.com/p/**kryo**/

Java mechanism) or in terms of a (registered) type-specific serializer. As we aim at using the more efficient kryo-based approach[7] in later stages of the project, we decided to allow the Algorithm Provider to follow either style. Therefore, the `FieldType` defines an (optional) `serializer` slot indicating the fully qualified class name of the responsible kryo serializer as well as an (optional) Maven artefact in `serializerArtifact` providing the serializer implementation (typically the same as `artifact`).
For a better understanding of the `FieldType` configuration, we take a simple string list type representing the basic types of the Java library as well as a domain-specific object type representing a Twitter feed as examples to explain the required information of each configurable element in both distinct cases. Figure 3 illustrates the definition of the `FieldType` in IVML as a fragment of the Basics IVML module of the Configuration Meta Model as well as its constraint requiring a unique name.

```
compound FieldType {
   NonEmptyString name;
   ClassString class;
   OptionalArtifactString artifact; // optional
   OptionalClassString serializer; // optional
   OptionalArtifactString serializerArtifact; // optional
}
setOf(refTo(FieldType)) types = {};
Constraint typeNamesUnique = types->collect(t|t.name).size()
    == types.size();
```

Figure 3: IVML fragment defining the FieldType.

As shown in Figure 4, a descriptive `name STRINGLISTTYPE`[8] is configured as type identification of the string list type. The `class` needs to be configured by the qualified class name referring to the fully implemented type, as shown in the example, `java.util.List<String>` refers to the implementation class name of `List` in the Java library as well as its element type `String`. As these types are already defined by types of the Java library, no `artifact` must be given. Further, we rely on the default serialization of Java for the moment.

```
FieldType StringListType = {
     name = "STRINGLISTTYPE",
     class = "java.util.List<String>",
};
```

Figure 4: IVML fragment specifying the configuration of an arbitrary type, here a list of Strings.

As a second example, we configure now a domain-specific type frequently used for the analysis Actually, in the Semantic Web analysis algorithms in QualiMaster, Twitter feeds are represented as a list of a Java type defined by an algorithm. As depicted in Figure 5, we call this type `LISTIFEXPERT`. In addition to the qualified class name of the `List` defined in the Java Library, the class configuration slot also defines the qualified implementation name of its element type, here `eu.qualimaster.types.IFExpert`. Furthermore, we need to configure an artefact so that the specific type `IFExpert` can be resolved, actually `eu.qualimaster:SpecificTypes:0.0.1-SNAPSHOT` as given for the slot `artifact`. As above,

[7] Intended for later optimization phases of the project, which requires, in turn, changes to the pipeline instantiation enabling kryo support for the generated classes.
[8] Actually, the priority pipeline already used this type to define the control stream for the correlation computation. Thus, we redefined it using the same name to maintain compatibility and to avoid interface changes.

```
FieldType ListIFExpertType = {
    name = "LISTIFEXPERT",
    class = "java.util.List<eu.qualimaster.types.IFExpert>",
    artifact = "eu.qualimaster:SpecificTypes:0.0.1-SNAPSHOT"
};
```

Figure 5: IVML fragment specifying the configuration of an arbitrary type, here a list of Twitter `IFExpert` type.

we rely on the default serialization of Java for the moment and do neither specify the serializer nor its artefact.

3.2.2 Item Names

In addition to the introduction of the `FieldType` in the `Basics` module, we extended the compound `Item` representing an input / output type to a pipeline element by a user-defined `name`. Actually, adding a name does not seem to be a big deal, but it greatly affects the understandability of the generated artefacts. Initially, we assumed that multiple items on the input or output side of a pipeline element is a rather exotic case and, thus, in the multiple item case, just numbered the types along their sequence of definition. However, already in the QualiMaster priority pipeline, the sources emit two different streams, actually the data stream and an internal control stream for the correlation matrix computation. Thus, WP2 soon issued a change request that a meaningful identification of input / output data interfaces would be very appreciated. Therefore, we introduced the configurable `name` of an item to support a human understandable identification and to ease the communication between Pipeline Designer and Algorithm Provider. The related change in `Item` is also illustrated in Figure 2.
This modification required changes to the pipeline consistency constraints. As indicated in D4.1, the input items of a processing element must match the output items of the preceding data source or processing element. We defined an additional constraint restricting that the input name of a processing element must be equal to the output name of its preceding data source and processing element akin to the previous type constraint.

3.2.3 Descriptive Text

As mentioned in D5.2, the QualiMaster Configuration Meta Model shall capture descriptive information for algorithms providing a search capability on the description of algorithms. The partners of the consortium currently playing the role of Algorithm Providers suggested this change in order to support reuse, i.e., to search for already configured existing algorithms rather than accidentally re-defining and re-developing algorithms. Thereby, a description in addition to the (unique) name is beneficial.
As such a description can also be helpful to reuse other configuration parts in the QualiMaster Configuration Meta Model, we defined a `description` for algorithms, but also for data sources, data sinks, families as well as pipelines. The related search capability will be integrated into the QualiMaster Infrastructure Configuration tool (QM-IConf) in the future.

3.2.4 Access to Hardware-based Execution

A particular capability of the QualiMaster infrastructure is to integrate software-based and hardware-based data stream processing. As discussed in D4.1, the configuration type `MPCCNode` models a Maxeler MPC-C series compute node consisting of a number of Central Processing Units (CPUs) and Data Flow Engines (DFEs) for the hardware-based execution. So far, the `MPCCNode` allowed configuring the (network) host address of a MPC-C node. However, to enable adaptation of the hardware-based execution, the QualiMaster infrastructure needs access to the configured compute nodes, on the one side to determine the available compute resources at runtime and on the other side to enact changes in the processing, in particular to load new algorithms on demand (enactment pattern ES-2 in D4.1).

Therefore, we equipped the `MPCCNode` with a `monitoringPort` so that the Monitoring Layer of the QualiMaster infrastructure can obtain resource information from the respective `MPCCNode`. We also added a `commandPort` enabling the Coordination Layer to communicate adaptive enactments such as algorithm changes to the respective `MPCCNode`. As we will explain in the next section, the Configuration (Meta) Model itself becomes an artefact during infrastructure instantiation and is accessible to the QualiMaster infrastructure, so that information like the communication ports or the host address can be utilized.

3.2.5 Automated Integration Support

In the previous version of the infrastructure part of the QualiMaster Configuration Meta model, information about the Processing Elements Repository was not made explicit. This imposes a problem if the installation of the QualiMaster infrastructure shall happen in a setting where location of the repository differs, i.e., a change of the infrastructure derivation process is needed. Furthermore, the QualiMaster Infrastructure Configuration tool (QM-IConf) shall be enabled to explicitly deploy versions of the pipelines that are considered to be stable for running on the cluster (called acknowledgement in D1.1/D1.2).
Thus, we support now the configuration of the Pipeline Elements `repositoryURL` in the infrastructure settings, which is taken over into the Maven build specifications generated during the infrastructure derivation process.
For enabling the explicit deployment of stable pipelines through QM-IConf, we equipped the individual pipelines (`Pipeline` type in the Configuration Meta Model) with a Maven artefact specification (`artifact`), separated the instantiation of the individual pipelines and their build processes in the infrastructure derivation, prepared the explicit deployment by installing Sonatype Nexus[9] as a user frontend of the QualiMaster Maven repository and prepared a specific repository connector for QM-IConf, which is able to perform the deployment. It is worth to note that the QualiMaster Configuration Meta Model now contains a constraint that limits artefact specifications to the form `groupId:artifactId:version` akin to Maven.
Within the QualiMaster infrastructure, the `artifact` specification of a pipeline is used by the Coordination Layer to download a pipeline before starting it. In order to obtain the artefact specification, but also to consider the contained adaptation model in the Adaptation Layer, we included an artefact specification for the entire Configuration (Meta) Model (`modelArtifact`) and package the actual Configuration along with the Instantiation and Adaptation Model during the infrastructure derivation. This artefact is then loaded from the Pipeline Elements Repository by the Coordination Layer and provided to the upper layers of the QualiMaster Infrastructure so that the QualiMaster infrastructure itself is configured through the Configuration Model.

3.2.6 Development Support

During the development of a pipeline, it is often helpful to have additional information on the actual processing, partly consisting of information provided by Storm, partly of additional information from the algorithms or the generated pipeline artefacts. However, such additional information, e.g., through logging, can impact the runtime performance of a pipeline (as we also found for logging information created by the IVML Reasoner, see Section 4.1 for more details). Thus, we decided to introduce a debugging mode for individual pipelines in the configuration. In particular, this debugging mode takes control over the logging of individual pipelines. As we generate the integrative pipeline artefacts based on the Infrastructure Configuration, we can just exclude debug code from code generation if the debug mode is disabled so that it cannot impact the runtime performance of a pipeline.

[9] http://www.sonatype.org/nexus/

3.2.7 Future Work

Until the end of the upcoming integration phase and the submission of D5.3, we plan to change the Configuration (Meta) Model only for solving urgent problems. For the future, we consider the following further extensions.

- Enable re-usable pipelines and support the generation of sub-topologies. As explained in D5.2, sub-topologies implement distributed algorithms that are fully developed by Algorithm Providers. However, the past experience with this development approach showed that realizing an error-free sub-topology can be an obstacle, in particular to developers, who are not fluent with Apache Storm. Furthermore, as the structure of a manually developed sub-topology is not part of the Configuration, the Adaptation Layer may run into trouble in reconfiguring (the resource usage of) such a topology. However, realizing this extension requires intensive collaboration with the QualiMaster partners, as it particularly shall support them in developing new pipelines.
- Provide sufficient configuration options to derive also the Storm and infrastructure configuration files from the Infrastructure Configuration to ensure consistency.

3.3 Enactment patterns

Enactment of adaptive decision can happen in different parts of the QualiMaster platform and the running pipelines as well as in different forms. In D4.1, we introduced 6 primary and 12 secondary enactment patterns, whereby the secondary patterns are intended to support and to be combined with the primary patterns to achieve gap-free enactment, i.e., enactment with minimal impact on the processed data streams. However, performing an enactment can be limited by the underlying Execution System. In D4.1, we mentioned as future work an empirical analysis of the enactment patterns to investigate whether the patterns can be realized at all in the environment of the QualiMaster infrastructure as well as to determine their impact on the quality parameters and the data processing. In this section, we discuss the initial results of an ongoing analysis of the adaptation patterns, involving both, a technical discussion whether already analyzed patterns can be realized, but also an empirical analysis of the effect of a certain pattern.

So far, we focused our analysis in particular on patterns that are supported by actually implemented enactment mechanisms such as commands provided by the Coordination Layer or which appear most beneficial for to project. In more details, we analyzed (using the identifiers defined in D4.1):

- Change functional parameter of algorithm (EP-1)
- Select algorithm from processing family (EP-2)
- Switch between software- and hardware-based processing (EP-4)
- Parallelize processing element (EP-5) combined with migration of processing (EP-6).

We discuss the results for the analyzed patterns in individual sections below. First, in Section 3.3.1, we start with the experimental setup. Then, in Section 3.3.3 we focus on EP-1, in Section 3.3.4 on EP-2, in Section 3.3.5 on EP-4 and, finally, in Section 3.3.6 on EP-5 / EP-6.

After completing D4.1, in particular our hardware partners identified issues in applying an algorithm switch (EP-4) between software-based and hardware-based correlation computation. After switching the correlation computation, the respective algorithm needs to rebuild the algorithm state, which can be rather time consuming, e.g., at least one value per market player involved in the correlation matrix is needed. The partners believe, that this can be solved by a form of state transfer between the involved algorithms (ES-11), running both algorithms in parallel (EP-5) until the end of the current analysis window is reached (ES-12). Thus, as a basis for further work, we discuss the initial design of state transfer strategies in Section 3.3.7. In Section 0, we conclude our work on enactment patterns and discuss future work.

3.3.1 Experimental Setup

Analyzing a complex pipeline such as the QualiMaster priority pipeline for the effect of individual enactment patterns is problematic, as it is difficult to control the experimental setting, i.e., the dependent experimental variables. Thus, we decided to design specific test pipelines, which are much easier to control and to observe.
A test pipeline consists of a test data source, at least one test family and a test data sink. The test data source produces random integer values at a constant frequency. If not described otherwise, the test data source in our experiments produces 1000 data items per second. As working with real algorithms would not support us in controlling the experimental setting, we rely on test algorithms that basically just emit the received integer data item. All pipeline nodes produce time-stamp based logs for the analysis of the experiment as we detail below.
To emulate a certain (realistic) behaviour, we equipped our test algorithms with two specific experiment-driven parameters, namely delay and the aggregation factor as we explain now:

- The *delay* defines how long (in milliseconds) a received tuple shall be postponed until it is (re-)emitted, i.e., how long the test algorithm pretends to process a single data item. Setting the delay to 0 implies that the received data item is emitted immediately.
- The aggregation factor (akin to the aggregation size in [3]) characterizes how many input items are being aggregated by the algorithm until an output item is emitted, i.e., how many output items shall be skipped (shedded) for a given input item. If the factor is 1, all input items are emitted. Currently, we do not consider the case that input items are multiplexed.

Figure 6 illustrates the most basic abstracted form of a test pipeline we are using. The test pipeline is composed of three basic elements, i.e., a data source, a processing family and a destination sink. In particular, the family is equipped with two simple algorithms which only pass through the received data based on the given delay and aggregation factor. Dependent on the respective experiment, we consider also more complex test pipelines. However, we aim at keeping the pipelines as simple as possible in order to have a better understanding of the execution environment and the pipeline. Please note that for example switching between sub-topologies is subsumed by this abstraction, as due to technical reasons the layout of the instantiated pipeline differs regarding switching between simple Java algorithms and sub-topologies (as explained in more detail in D5.2).
As our experiments are based on a simple test pipeline described above, we configured a local Storm cluster at SUH as experimental execution environment, in particular to perform pre-experiments without disturbing the ongoing work on the cluster provided to the consortium by TSI. In the future, we consider re-validating our results on the TSI cluster, in particular for more complicated pipelines, for using hardware-based processing, or if we need more resources. Akin to the TSI cluster, we installed on the SUH cluster:

- Linux (Ubuntu 12.04.4 LTS)
- Java (version 1.7.0_71)
- Apache Storm (version 0.9.3)

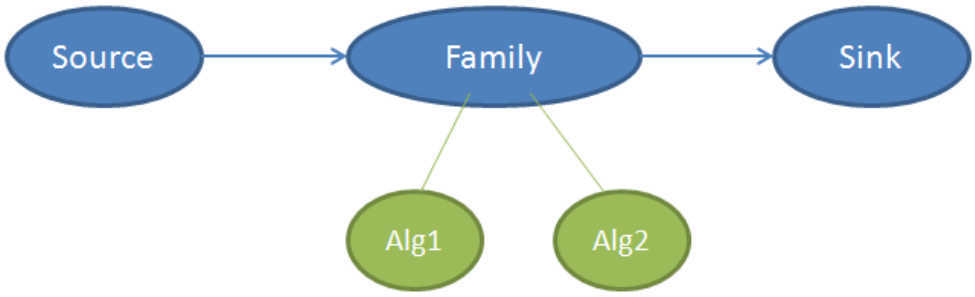

Figure 6: Overview of the abstract test pipeline.

- Apache ZooKeeper (version 3.4.6)
- The most recent version of the QualiMaster infrastructure.

The experimental Storm cluster at SUH consists of one Storm Nimbus node acting as the master node assigning tasks to Supervisors and five supervisors controlling the actual Worker processes (cf. D5.1 for more details on Storm). Furthermore, we installed a small Zookeeper sub-cluster, coordinating the Nimbus and Supervisors within a Storm cluster. To establish a stable and reliable cluster, we configured the Zookeeper cluster for three Zookeeper nodes as an ensemble, in particular following online sources[10] recommending an odd number of Zookeeper machines. All machines in the cluster are connected via Gigabit-Ethernet, which allows a transfer rate of around 100MByte/s among the machines. In addition, we synchronized the system clocks of the cluster machines using the Network Time Protocol (NTP) in order to enable comparisons of the timestamp-based logs. Due to time synchronization, the actual time difference among the machines is less than 3 ms.

3.3.2 Experiment Analysis

Basically, we are interested in the actual throughput / latency of the items (as defined by the Quality Taxonomy) and also whether all items are processed (including the processing sequence). Therefore, we log the individual elements at sources, sinks and processing elements for later analysis. Individual items are logged as the form of comma-separated values (CSV) file along with a timestamp indicating the arrival time (in milliseconds system time) and the sequence number of the logged data item.
We analyze the experimental logs in Microsoft Excel, by first separating the timestamp and the sequence number into two different columns, then deriving the timestamp in seconds and finally obtaining a subtotal summarizing the number of data items in each second as aggregation unit. We selected throughput in seconds as is a frequently used unit. In particular this complies with the maximum deviation of the synchronized time in the cluster, which is several orders of magnitude less than our aggregation unit. Therefore, the time series charts we derive from this data indicates the number of items over timestamp in seconds rather than milliseconds. Due to the aggregation, the produced logs do not exceed the maximum memory of Microsoft Excel, which we use for analyzing the logs and producing time series charts. However, in case of larger logs we will consider R[11] or similar scalable systems.
For better control of the experiment, we prepare for each individual enactment pattern (at least) one experimental script. Although we could link these scripts directly against Storm, we decided to work with the QualiMaster infrastructure and to utilize the experiments also as an additional

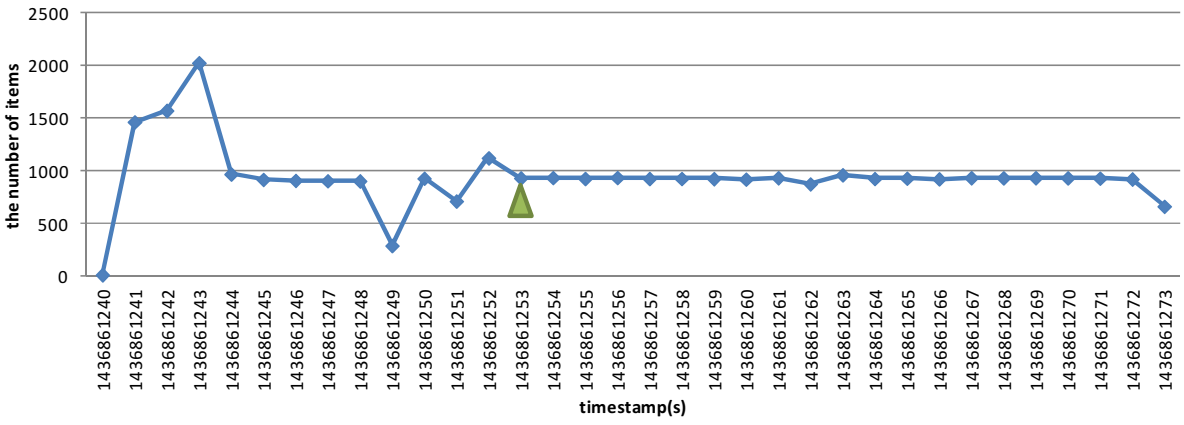

Figure 7: Throughput while running the test pipeline without interruption.

[10] http://http://zookeeper.apache.org/
[11] http://www.r-project.org/

validation of the QualiMaster infrastructure. Such an experimental script is a Java program, which launches the respective commands provided by the Coordination Layer of the QualiMaster infrastructure (cf. D5.2) such as starting or stopping the test pipeline as well as switching algorithms etc. During an experiment, we run the respective script five times to detect deviations and analyze the individual runs.
As part of pre-experiments, we figured out that a pipeline needs some time after its start-up to stabilize the processing, i.e., the throughput around the expected value of 1000 items per second. Figure 7 illustrates the throughput in terms of the number of items per second of the data stream processing along with a running pipeline without any interruption or execution of enactments. In the diagram, the pipeline takes around 14 seconds to stabilize the throughput (indicated by the green triangle). We recorded some instability in throughput of a maximum length of 50 seconds. According to the experience we made so far, these instabilities seem to appear randomly and are potentially caused by the Storm framework. In the following sections, in order to focus on the impact of individual enactment patterns, we will exclude the start-up phase of the pipeline from the displayed charts and focus on the throughput during time period around the enactment. Moreover, we will focus on the throughput measured at the end (Sink node) of the test pipeline.
Below, we discuss the results of all script executions. Thereby, we will present representative graphs for the data we obtained from the experimental logs.

3.3.3 Change functional parameter (EP-1)

Changing a functional parameter of an algorithm can be requested to change the processing of an algorithm, e.g., to adjust an analysis window or to select a specific algorithm variant implemented in the same algorithm. In an extreme form, the code of a software-based algorithm can be re-instantiated to turn a certain (frequently used) parameter into a constant to optimize the processing for performance (ES-4). However, without analysis it remains unclear whether changing a functional parameter for an algorithm yields negative impacts on the data processing. In case of a real algorithm, the effect of changing a parameter on the quality parameters we are interested in depends on the implementation of the algorithm. Therefore, this experiment just focuses on the basic mechanism of requesting the change of the parameter until the algorithm is notified and leave experiments with real QualiMaster algorithms for future analysis. Furthermore, we focus on changing parameter values at runtime leaving the option to re-instantiate code (ES-4) for later experiments.
The actual mechanism behind the change of a functional parameter is provided by the central memory of a Storm pipeline, the so called Zookeeper. Actually, the Zookeeper is a kind of shared directory structure of Zookeeper nodes (ZNode), where each ZNode can carry payload data. In particular, interested parties can be informed upon the change of a ZNode (observer design pattern [17]), which is also utilized by Storm, e.g., to distribute Worker assignments to the Supervisors. The Zookeeper can be accessed through the Curator framework[12], which, in turn, is used by the QualiMaster Coordination Layer to change respective ZNodes upon a coordination command, here, changing a functional parameter for a certain processing element in a given pipeline.

[12] http://curator.apache.org/curator-framework/

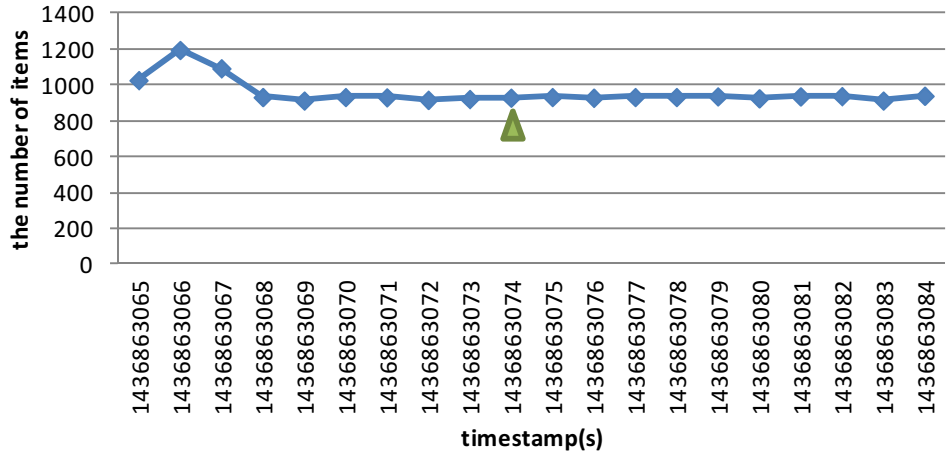

Figure 8: Throughput while analyzing the effect of changing a functional parameter.

First, we focus on a parameter that does not have an actual effect on the algorithm. As an effect, the throughput of the pipeline should not change. Figure 8 illustrates the effect of changing such a functional parameter. In the diagram, the flag parameter arrives at timestamp 1436863074 s (green triangle). As indicated by Figure 8, this enactment causes almost no fluctuation on the throughput.

However, a real algorithm will probably react somehow when a parameter is changed. We illustrate this as an example of changing the delay of the actual processing node as shown in Figure 9. Here, the complete effect of changing a functional parameter takes around 2 s, i.e., from enacting the change (indicated by the green triangle) it needs 2 s to propagate the effect through the test pipeline to the sink node. Actually, we determined the time between sending the respective command and the first effect at timestamp 1436428581 s to 20ms, while we recorded the end of the effect 2 second later at 1436428582 s.

From the results analyzed above we conclude that changing a functional parameter is per se not problematic, i.e., just sending and processing the respective command does not have negative effects. However, the actual change to the data processing, here in terms of throughput, depends on the affected algorithm and its position within the pipeline, so that slowing down or speeding up an algorithm can yield subsequent ripple effects in the pipeline and may require that following adaptations take this into account.

3.3.4 Select algorithm from processing family (EP-2)

Selecting an algorithm from its processing family, i.e., among alternative algorithms with different quality tradeoffs, is at the conceptional heart of the QualiMaster project and, thus, EP-2 is one of the key enactments. Akin to changing a parameter of an algorithm at runtime, EP-2 is implemented in the QualiMaster infrastructure through a change in the Zookeeper directory,

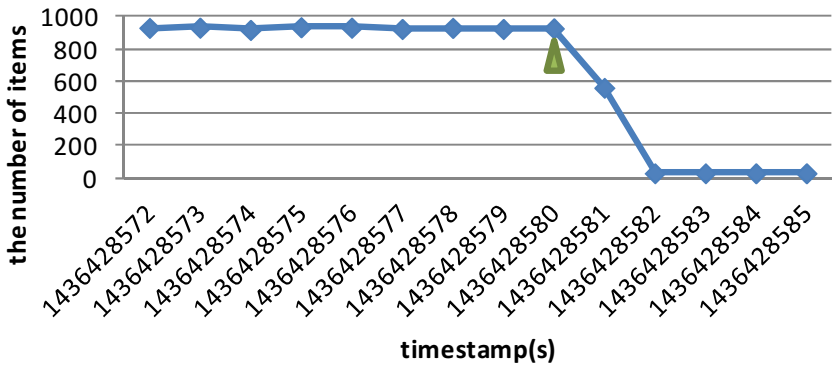

Figure 9: Throughput while sending a delay parameter to the executed algorithm.

leading to notification of the respective topology element, actually a Bolt.
As already indicated above for, due to technical reasons, we need to differentiate the following two base cases:

- Switch among simple algorithms
- Switch among sub-topologies

For both base cases, we analyze the respective effects with and without delays, as we aim at evidence whether switching with a filled (input) buffer[13] may lead to a backlog of items. Such a backlog may then anyway be processed by the algorithm that was active before the switch as the underlying Storm framework does not know about switching algorithms. Originally, we observed such a situation in an early version of the Priority Pipeline, but at that time we could only speculate about the respective root cause. Similarly, ripple effects of an algorithm switch in later parts of a pipeline are important to understand, as we will consider this in our ongoing analysis and in the design of the state transfer in Section 3.3.7. Below, we discuss now the two base cases, each with and without intentional delay caused by a processing element.

3.3.4.1 Switch among simple algorithms without delay

In this section, we analyze the effects of switching a simple Java algorithm without delay. Figure 10 illustrates the throughput in terms of the number of items per second during the experiment. We perform a switch from Alg1 to Alg2 at timestamp 1432199588 s and record further 13 seconds to capture for potential effects on the throughput.
Comparing the throughput before and after switching the algorithm shows that the amount of the processed items remains nearly constant. Thus, we conclude that performing an algorithm switch of a simple Java algorithm which processes the input items immediately does not cause negative effects.

3.3.4.2 Switch among simple algorithms with delay

In contrast to the previous experiment, we discuss now the effects of switching from an algorithm with delay to an algorithm without delay. Therefore, we wait until the pipeline reaches a stable state, enact then the delay on Alg1 to validate the delay effect first, wait some time and switch then to Alg2. As we tested with different delay parameters, the reduced number of items is proportional to the given delay, e.g., a 30 ms delay leads to 30 times less items of the expected throughput of 1000 items per second, i.e., around 33 items per second. In this experiment, we use a representative delay of 30 ms, but we made similar observations with

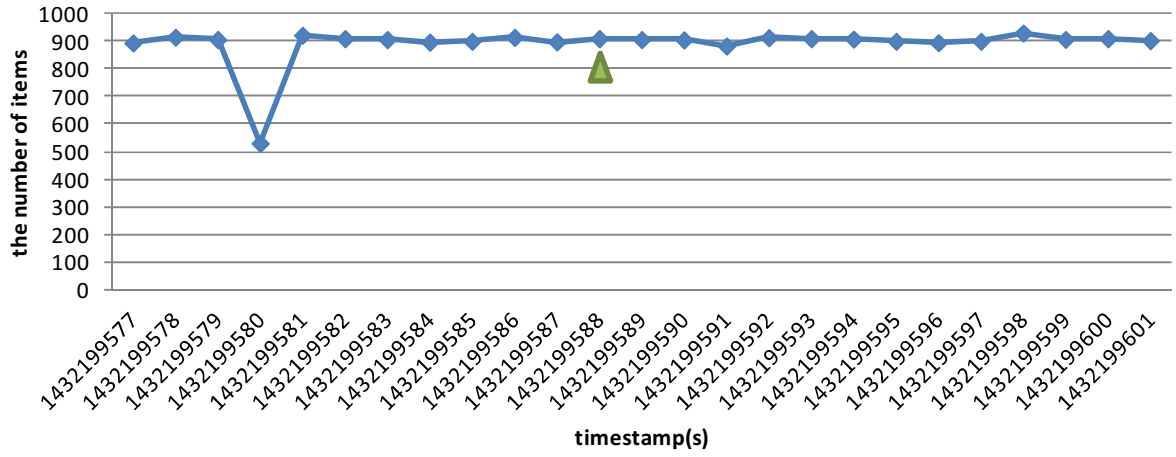

Figure 10: Throughput while switching among simple algorithms (no delay).

[13] Technically, Storm aims at immediately transferring processed items to the next Worker, which buffers them in its input queue and dispatches them as soon as possible to the input queue of the responsible Executor. The Executor fetches them item-by-item and calls the processing method of the implementing Bolt.

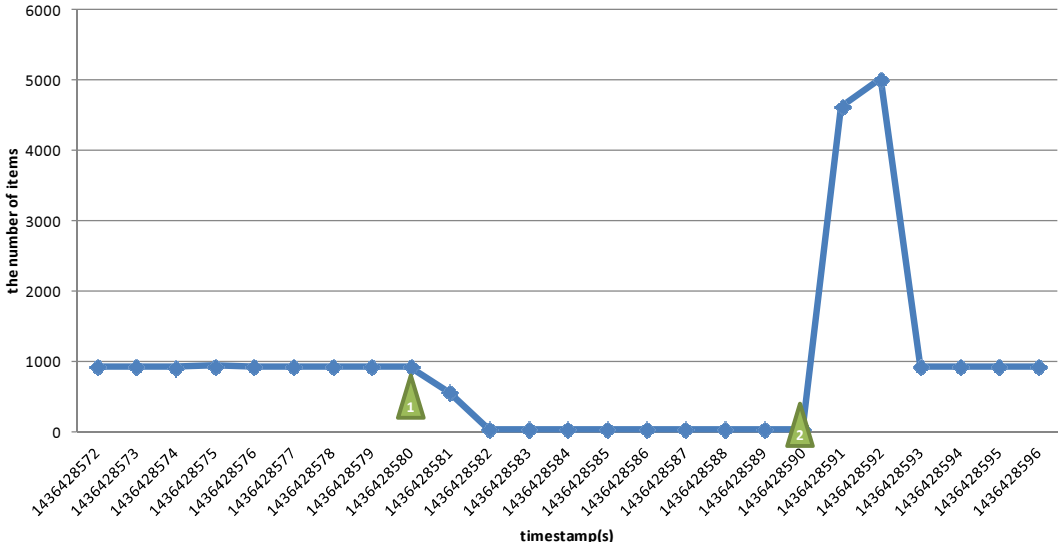

Figure 11: Throughput while switching among simple algorithms (delay).

other values of the delay parameter.

As shown in Figure 11, we wait until the pipeline reaches a stable processing 1000 items per second. At 1436428580 s (the first triangle highlight) we enact the delay parameter, which leads to a ripple effect of around 2 s as mentioned in Section 3.3.3. The delay decreases the throughput 33 items per second. After 10 s running the algorithm with 30 ms delay, we switch from the slow algorithm Alg1 to Alg2 running without delay. This happens specifically at timestamp 1436428590 s (the second triangle highlight) and causes a spike in the throughput of around 5000 items per second for duration of 3 seconds. As we are not operating the pipeline at the limit of the available resources (we also performed some initial experiments at 15000 items per second), the pipeline can quickly recover and process the actual backlog. After this time, the pipeline regained the stable throughput of around 1000 items per second.

In order to verify these results, we repeated the same experiment with different delay parameters, i.e., the respective delay of 50 ms and 100 ms. The results show that the slow algorithm leads to a backlog of items in (more precisely before) its executing Bolt and the items are queued in the internal buffer provided by Storm but processed directly after switching in terms of an event peak. This is due to the queue mechanism of Storm, which prevents the data loss and guarantees the data being completely processed. Actually, this mechanism could also help us migrating the queued items from the originally active algorithm to the changed one and to avoid that the original algorithm keeps on processing. Therefore, we plan to analyze the effect of explicitly rejecting (rather than accepting) items in the disabled sub-topology.

3.3.4.3 Switch among sub-topologies without delay

The switching experiments discussed so far take only simple Java algorithms into account. As the QualiMaster priority pipeline involves sub-topologies, i.e., explicitly distributed Storm algorithms, we are also interested in the effects caused by switching from / to a sub-topology and whether the effects differ from simple Java algorithms. Akin to Section 3.3.4.1, we discuss in this section experiments with sub-topologies running without delay and focus on delays in the next section.

In this experiment, the actual switching to another sub-topology-based algorithm happens at timestamp 1436459531 s. As depicted in Figure 12, the throughput before and after the switching point remains nearly constant, namely around 1000 items per second. Akin to the case of switching to a simple Java algorithm, we conclude that the switching among sub-topology-based algorithms without delay does not harm the data processing of the pipeline.

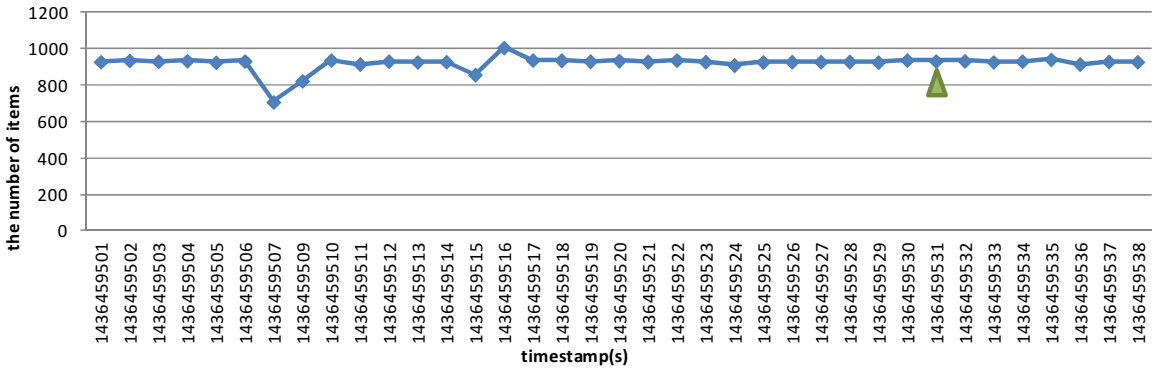

Figure 12: Throughput while switching among sub-topologies (no delay).

3.3.4.4 Switch among sub-topologies with delay

In this section, we discuss switching a sub-topology-based algorithm running with a certain delay. Akin to the experiment in Section 3.3.4.2, we first wait until the stable state, enact then the delay of 30ms on Alg1, wait for some time and switch then to Alg2.

Figure 13 shows the throughput in terms of the number of items for switching from a delayed sub-topology algorithm to a non-delayed sub-topology algorithm. The enactment of the delay on Alg1 happens at timestamp 1436459957 s (the first triangle highlight). As indicated by Figure 13, the throughput is reduced to 33 items per second akin to Section 3.3.4.2. At timestamp 1436459967 s (the second triangle highlight), we switch the processing family to the non-delayed sub-topology algorithm Alg2. Instead of coping with the queued items immediately as appeared in the case of switching a simple algorithm with delay, in this experiment, the processing of the backlog items is deferred by 3 seconds (the third triangle highlight), which actually seems to be caused by Storm framework. To explain this effect, we analyzed the timestamps of the arrived items. By comparing the timestamps of the items, we realized that after switching the delayed sub-topology of Alg1 is still processing its backlog of items. We can also see from the diagram at timestamp 1436459979 s (the fourth triangle highlight) the throughput starts to reduce and returns back to the expected throughput (1000 items/s) after 5 seconds processing the queued items in the previous delayed sub-topology. The backlog of items is due to the queuing mechanism of Storm, which guarantees that the data fully processed and prevents accidental data loss. In particular, in this case, the sub-topology-based algorithm consists of the Spouts/Bolts so that the items are queued on the algorithm level instead on the family level as this happened in the experiment described in Section 3.3.4.2.

For the moment, we conclude that a gap-free switching between sub-topology based algorithms of significantly different latency needs further consideration. As one option, we will analyze also here the effect of explicitly rejecting items in the disabled sub-topology. Further, we aim at analyzing whether disabling the processing guarantees of Storm could help. Ultimately, we may take an explicit buffer transfer (ES-1) into account, although this will require a modification of Storm.

3.3.5 Switch between hardware- and software-based processing (EP-4/ES-2)

As indicated in D3.1, the reconfigurable hardware is integrated along with the QualiMaster infrastructure aiming at a performance gain of the data processing, in particular, for the optimized hardware-based data processing algorithms. Basically, a flexible interface between Storm and Maxeler has been implemented as a communication framework between the QualiMaster pipeline and the reconfigurable hardware. In order to verify the impact of the hardware integration on the enactment of adaptation decisions, in this section we aim at figuring out whether and how the switching to hardware-based processing affects the throughput of a running pipeline.

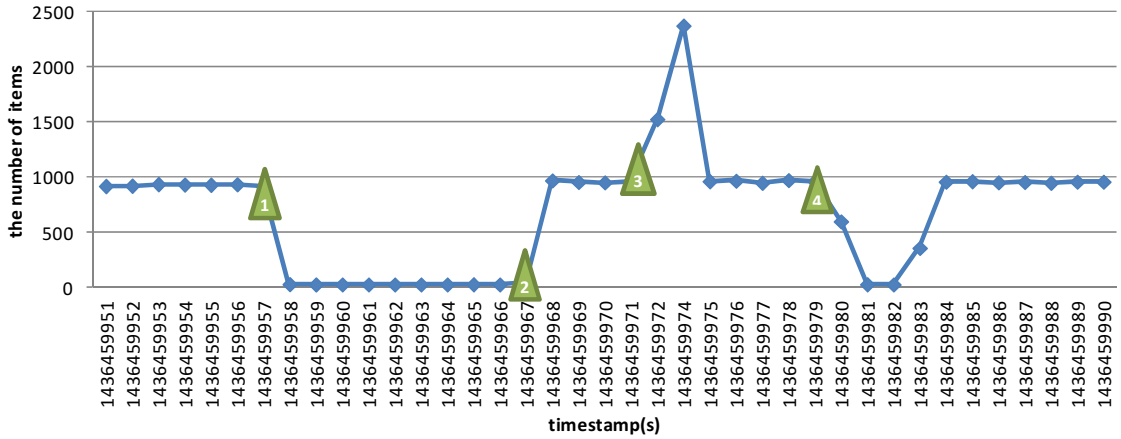

Figure 13: Throughput while switching among sub-topology-based algorithms (delay).

In this experiment, the family element of the test pipeline is equipped with a simple software Java algorithm (adopted from the previous experiments) and a hardware-based algorithm implemented in the same style. Technically, the hardware algorithm batches 1000 data items, and then passes them to the Maxeler hardware located at TSI and, finally, returns back the processed data items. Akin to the software test algorithms the hardware-algorithm just passes through the data according to the given parameters of delay and aggregation. It is noteworthy that the hardware-based algorithm is actually integrated by a Storm sub-topology including a Bolt sending data items to the hardware and a Spout receiving the data from the hardware (cf. D3.2 for the design of the integration). Furthermore, it is important to indicate that we kept the experimental setting as described in Section 3.3.1, i.e., the software pipeline is running at SUH while the hardware is running at TSI, although this may affect latency and throughput due to the geographical distribution. In future, we plan further experiments that run only on the TSI side to avoid such effects.

Figure 14 illustrates the throughput while switching from the software-based algorithm to the hardware-based algorithm. The pipeline is first running using the software-based algorithm. At timestamp 1437644175 s (indicated by the green triangle), we switch the running algorithm to the hardware-based algorithm. Figure 14 indicates that the throughput during the switch to the hardware algorithm is reduced to, in the diagram, to around 400 items per second. Currently we believe that the reduction of the throughput during the switch is due to the on-demand initialization of the communication with the remote hardware at TSI and the communication delay mentioned above. Further it is interesting that after the switch, in contrast to the plain Storm pipelines discussed in the previous sections, the throughput reaches the exact rate of data items emitted by the source, namely 1000 items per second. Already for our experimental setting this indicates that integrating hardware-based co-processors can be beneficial in

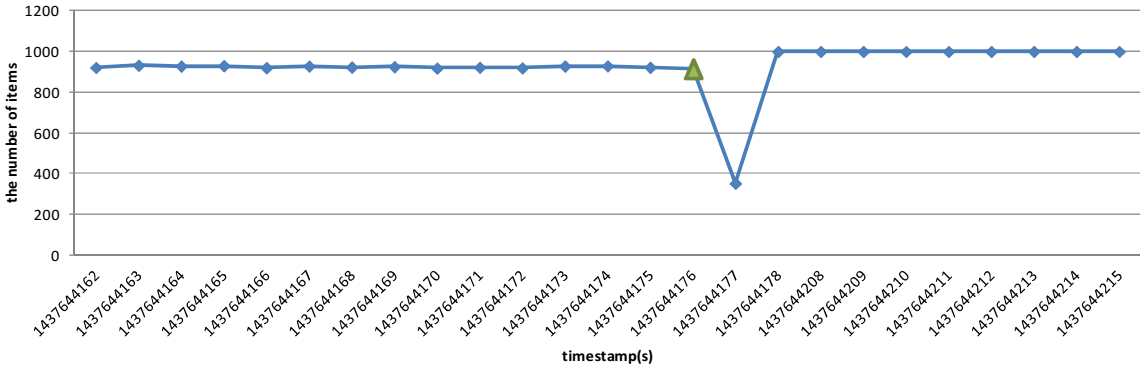

Figure 14: Throughput while switching between software- and hardware-based processing algorithms.

situations where the hardware can be utilized adequately (see D3.2 for a related discussion).
Please note that in this experiment, we do not consider the delay parameter as we did in the previous experiments. Our aim of this experiment is to have a first impression of the effect of the integration of software- and hardware-based algorithms. Thus, we scheduled further experiments with the delay parameter and to conduct experiments on the TSI cluster to avoid network communication latencies.

3.3.6 Parallelize processing elements (EP-5/EP-6)

According to D4.1, EP-5 aims at parallelizing a processing element, i.e., to utilize additional resources in parallel to speed up data processing or to compensate computationally intensive processing elements. Therefore, Storm offers the rebalance operation, which allows defining at runtime a new number of workers for the pipeline / topology as well as the individual number of executors to be utilized per processing element, i.e., per Spout or Bolt. The tasks represent clones of Bolts that can be used to process grouped data or to define residual capacity for scaling up a cluster. The number of tasks is fixed by the topology definition and constant at runtime. In any case, the main Storm topology invariant holds, namely that

$$\#Executors \leq \#Tasks$$

This requires that each logical task is processed by at least one physical executor and limits the scalability of a Storm cluster, i.e., it is advisable to start a topology with a certain number of provisional tasks. Dependent on the actual task / executor assignment produced by the pluggable Storm scheduler (strategy design pattern [17]), a Worker JVM can host multiple executors and an executor can handle multiple tasks. One extreme case is to run all executors in one single JVM and tasks "sequentially"[14] in one executor, which occurs in particular when testing topologies on a local machine, the so called "local cluster". In the other extreme case, each executor runs in an own JVM and each task in an own executor.
According to the online documentation of Storm it seems that the rebalance operation aims at optimizing the entire compute cluster and, thus, may rebalance even multiple dependent topologies at once. The latter can be avoided using a specific scheduler, such as the Storm isolation scheduler, which can prevent the reassignment of certain reserved resources. However, during the rebalance operation, Storm stops, rearranges and starts the affected pipelines, i.e., it causes the related data processing to pause for a certain time.
In this experiment, we aim at analyzing the characteristics of the rebalance operation and to discuss alternatives, i.e., we use the rebalance operation to change the parallelism of a pipeline. Therefore, we implemented a respective Coordination Layer command, which finally calls the Storm rebalance operation for the indicated pipeline. The start-up setting of this experiment is one worker and two executors. Figure 15 depicts the results of rebalancing the test pipeline, here only by changing the number of workers for the entire pipeline from 1 to 2 at timestamp 1432202870 s (green triangle). Directly after sending the rebalance command, Storm stops the actual pipeline, shuts down the individual workers and restarts new workers (JVMs). In summary, this stops data processing for 8 seconds and causes a further instability of 10 seconds. Actually, Storm performs the same basic operations if the number of executors of a certain processing element shall be changed. According to our experience, the processing downtime increases with the complexity of the pipeline, i.e., depends on the affected number of JVMs. As mentioned above, depending on the active Scheduler, Storm may even rebalance the entire cluster involving also further pipelines. Thus, we conclude that with the recent Storm versions a gap-free change of the parallelism of a pipeline is not possible.

[14] The actual sequence is determined in Storm by the predecessor pipeline element depending on the defined grouping style, e.g., according to data fields, in a random manner, etc. Thus, thinking about a "sequential" execution of the tasks is a simplification that might help to understand Storm at a glance.

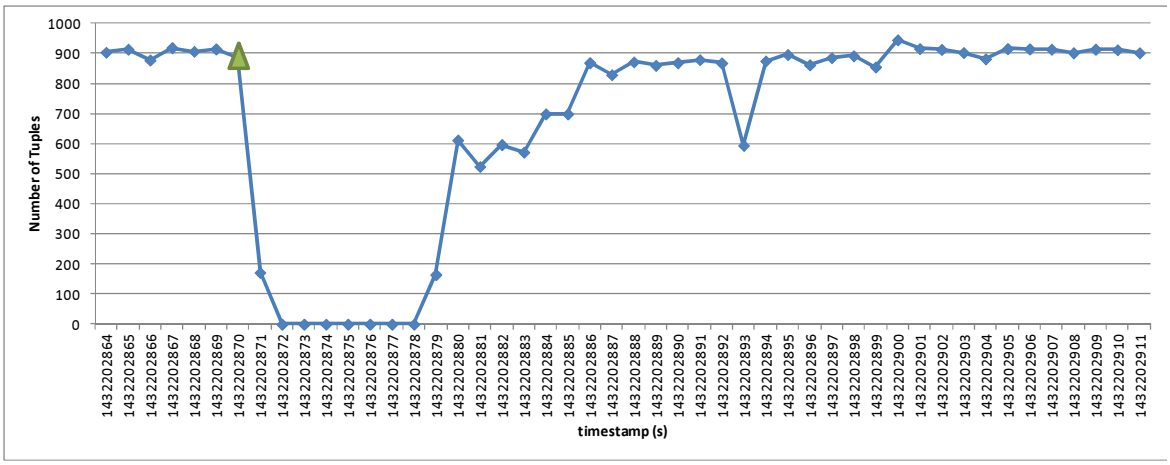

Figure 15: Throughput while rebalancing the number of workers for the test pipeline.

To achieve a gap-free enactment also in the case of changing the parallelism, we started to investigate whether a modified version of Storm could allow changes of the parallelization at runtime. Therefore, we analyzed the data structures and the internal processing of Storm. As explained above, Storm stores its worker assignments in a central data structure on the Zookeeper(s). This assignment defines the tasks that shall be executed on a certain worker / executor. Basically, modifying this assignment can cause individual workers to stop and new ones to start. However, certain conditions in the official version of Storm aim at avoiding such "illegal" assignments. Thus, we followed the path of a changed assignment through the Nimbus, Supervisor, Executor and Task code, modified the conditions and interpretation of the assignments accordingly and, finally, adjusted the default shutdown functions of executors, the item / tuple routing and enabled a queue transfer among old and new executors to avoid item loss. In contrast to the experimental setup, we rely here on Storm version 0.9.5, which was the most recent stable release when we prepared this deliverable. We selected that version, because it contains improvements regarding fault tolerance and failover of workers, which we expect to be beneficial for our modifications. Actually, these improvements in Storm will also help realizing the processing error adaptation scenario (AS-5).

As a preliminary result on a local cluster running on a usual Intel Core i7 laptop with 16 GBytes main memory, we can change the parallelism on the level of executors at runtime in a test pipeline in less than 150 ms. During this reconfiguration time we run the old and the new executors in overlapping fashion so that a potential downtime on the affected path is minimized and the current processing of items can be completed. Actually, enacting the change can cause a delay of several seconds, as the changed assignment is passed through the Storm components mentioned above by actively polling the Zookeepers with a certain configurable frequency. So far, as the other experiments described in this Section occupied our cluster, we were not able to validate the change of parallelism on the level of workers (realizing EP-6), i.e., on a distributed cluster, but we are confident that we can achieve good results there. We scheduled experiments dedicated to our modifications as soon as the validation of the new functionality is successfully completed

3.3.7 Strategies for State Transfer (ES-11)

Additional challenges are added to the situation of adaptation-driven switching in the case, where the processing is not stateless. If, for example, a processing element aggregates the number of tweets about one stock in the last five minutes, it is not possible to switch between to processing elements without taking this state into account. Optimizing the time needed to perform the adaptation and minimizing the impact on the data streams (gap-free enactment) requires a careful design of the algorithm switch (EP-2) for stateful components, which may involve a generic form of state-transfer (ES-11). In this section, we discuss the initial design of

different strategies combining enactment patterns to achieve gap-free switching of stateful processing elements in a data processing pipeline. We start with an overview on related work, discuss then the dimensions to be considered for a design as well as three strategies to enable the algorithm switch for stateful processing elements.

Maintaining the state of a reconfigurable software system is a research topic already for a longer time. For example, in [29] Wemerlinger and Fiadero use Category Theory to describe the system state and potential adaptations. In that work, the authors discuss the transfer of the state between old and new components to realize component replacement and introduce specific situations when reconfigurations of individual components can happen, such as the quiescent state, where a component is not interacting with other components so that it can be disconnected and removed safely from the system. For distributed CORBA-based systems, Almeida et al. allow in [2] replacement, migration, addition, and removal of components, which explicitly provide access to their state as well as specific lifecycle operations, such as passivating a component. Transferring the state of a component can particularly be seen as a dynamic update. For service-oriented systems, [25] aim at a dynamic update connector, which combines request buffering (ES-1), request redirection (ES-8) and state transfer (ES-11) to achieve a form of gap-free update, namely continuous availability of services, i.e., no interruption and only minor decline of the quality during the update. Moreover, Zhang and Cheng use in [30] state diagrams and temporal logic to model state transfer in adaptive software systems. Also in distributed stream processing systems, some research work addresses the problem of state transfer. For example, in [23], Rundensteiner et al. perform state relocation of data stream operators that run out of memory, i.e., the authors move parts of the state to the persistent memory and load it back if possible. Castro Fernandez et al. suggest in [15] a set of state management primitives for data stream processing systems, namely checkpoint, backup, restore and partition, to achieve fault tolerance and to enable cloud scaling. In particular, the authors distinguish between the processing state consisting of key-values and the most recent timestamps handled by a processing element, the buffer state and the routing state. Actually, the management primitives in that work focus on scalability rather than algorithm switching and need further refinement. Further, Nasir et al. discuss in [21] a practical load-balancing approach considering the state of operators in key-grouping data stream operations. In summary, maintaining the state of components in a reconfigurable system is still a challenging research question as noted by Gheis et al [18], in particular, if state operations shall be generic and certain properties shall be ensured, e.g., gap-free enactment and short adaptation time.

In this section we distinguish the original processing element and the target processing element, where the *original processing element* is the one that does the processing before the switch and the *target processing element* is the one that is supposed to take over the processing after the switch. Please note that following the pipeline terminology introduced in D4.1, the term processing element may be a single algorithm, but in particular also a distributed algorithm realized by a sub-topology.

Obviously the state managed by a processing element (and taken into account for its computation) might vary widely in complexity as well as in the time period covered by the state. This will also require different switching strategies (protocols) for managing the state transfer between the original and the target processing element. As indicated in D4.1, such reconfiguration protocols (inspired by [BoyerGruberPous13]) can be defined in the enactment phase of our adaptation control language rt-VIL. In this first discussion of state transfer, we will focus on some frequent cases, without claiming to fully cover all cases that might occur. The state transfer discussion will be further extended in subsequent deliverables, in particular the upcoming D5.3, also taking up the experiences gathered with the application of the switching strategies.

In this deliverable we consider three main switching strategy, which fundamentally distinguish in the way the state is transferred:

1) *Warming-up strategy*, which builds up the required state information in the target processing element before switching,
2) *Direct Transfer Strategy*, which actually transfers state information between the two processing elements and
3) *External Storage Strategy*, which relies as implied by the name on external storage of the state.

Of those three, the direct transfer strategy requires the most complex switching protocol and is discussed in different variants. Those switching strategies are discussed below together with their advantages and disadvantages.

3.3.7.1 Warming up strategy

This switching strategy accumulates the cases, where the target processing element needs some warming up, before getting into a state, where it can produce valid results. The rough idea here is that the original and the target processing element run in parallel until the target processing element has accumulated sufficient state information to deliver valid results. Running processing elements in parallel, discussed in terms of active operator replication in [15], can lead to a duplication of the resource usage, but in contrast to [15] or similar works, we do not aim at keeping this allocation for a longer time (in the sense of passive replication in [15]), but just for a temporary period enabling a consistent and timely switch among stateful algorithms.

For original and target processing elements to run in parallel incoming items have to be presented to both processing elements. This also implies increased resource consumption for the transition time.

A frequent case for the warming up is the aggregation over a (not too large[15]) time window of length t_w (sliding or fixed window). In this case, switching can be performed, when the target processing element has seen items covering time span t_w.

In more detail, the switching strategy will do the following (see also Figure 16): At time t_0, upon switching request the incoming stream will be duplicated and presented to both the original and the target processing element as illustrated in Figure 16 a). Both elements will process the stream in parallel for the time window t_w. During this time window, only the results from the

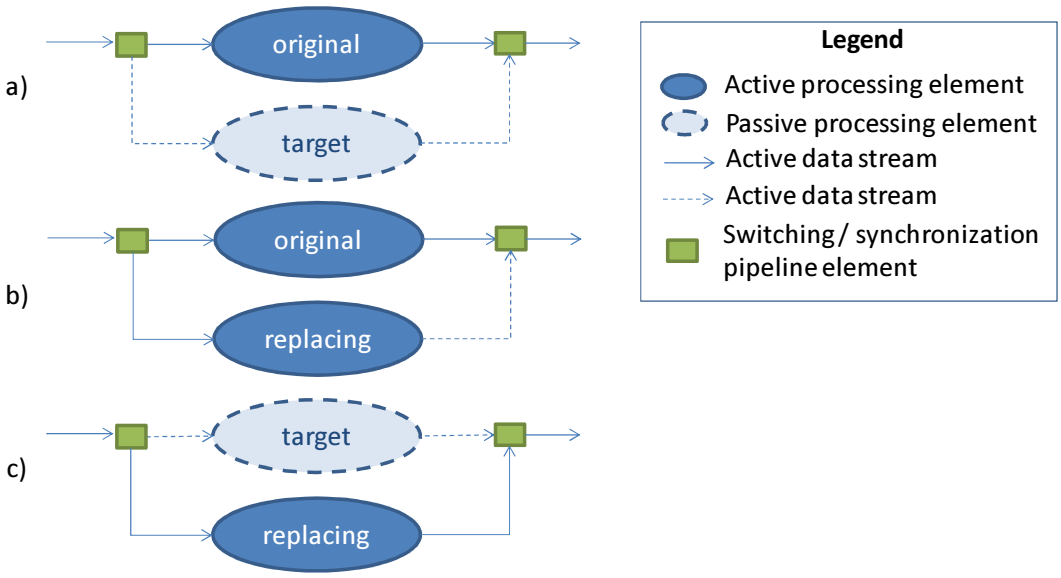

Figure 16: Warming up strategy.

[15] Note, that for larger time windows such as days and beyond the strategy of running both processing elements in paralell would be too expensive.

original processing element should be used, since the target processing element is still warming up accumulating its first valid state as shown in Figure 16 b). At time point, t_0+ t_w the target processing element can start delivering results. At this point in time the original processing element can be discarded and the results from the target processing element can be used in the processing pipeline as indicated in Figure 16 c). Thus the overall switching time in this case is t_w.

Note, that the windows used by the original processing element and the one used by the target processing element are not completely in synch. This may lead to a small result discrepancy at the time of switching between the result streams. However, the advantage of this switching method is that is makes no further assumptions about the processing elements (and the algorithms used by them).

Another situation that can be covered by the Warming up strategy is the need for a set up time, i.e. the situation, where the target processing element needs some time t_{setup} before being able to start processing elements from the stream. In our setting this might be the time that is needed to load an algorithm into the reconfigurable hardware. Obviously, this time has to be taken into account both for stateless and stateful processing. The overall switching time, in this case adds up to t_{setup}+ t_w.

3.3.7.2 Direct State Transfer Strategy

In cases, where the state is too complex to be rebuild in a warming up phase (see above), state information can be directly transferred between the original and the target processing element before the target processing element starts processing. Obviously, such a state transfer is coupled with a delay in processing (the time that is needed for the state transfer), which depends on the size and complexity of the state information and the connectivity between those

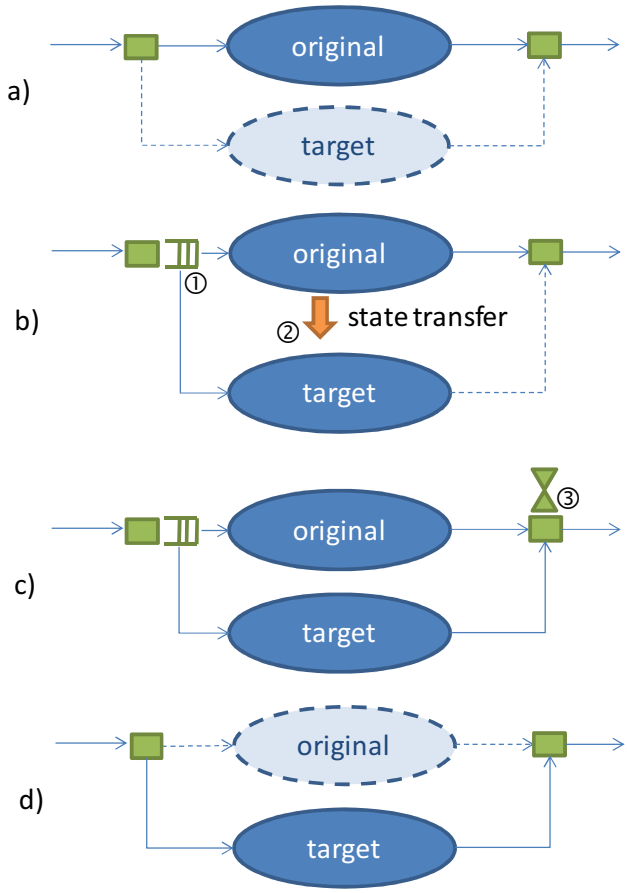

Figure 17: Direct state transfer strategy.

processing elements.
The basic idea of the Direct State Transfer is illustrated in Figure 17. Starting with an original and a target processing element, the input stream items are buffered and duplicated as shown in Figure 17 b) ① and the state of the original processing element is transferred to the target processing element as depicted in Figure 17 b) ②. Depending on the actual strategy, the output stream must be synchronized as we discuss below (Figure 17 c)) in order find an appropriate point in time to switch among the processing elements (Figure 17 d)). For all Direct State Transfer Strategies it is required that the participating processing elements agree on a format, in which state information is transferred and on a protocol for this transfer.
In case, the state transfer time is rather small, a variant of Direct State Transfer, which we call **Simple State Transfer Strategy** can be used. With this strategy, at time t_0, when the switching request arrives, the original processing element is stopped and its current state is transferred to the target processing element. Subsequently the input stream is routed to the target processing element, which then starts processing the stream at time $t_0 + t_{tt}$, where is the time required for the state transfer. Please note, that this strategy creates a gap in the result stream of length t_{tt} (plus the latency of the processing element).
For cases, where the state transfer time is too large to tolerate the result gap implied by it[16], more complex switching protocols are required in the context of the state transfer. For avoiding the gap on the result stream, it is possible to continue running the original processing element during the state transfer, which produces results during this time. This implies that some type of synchronization is afterwards required between the results produced by the original processing element and the target processing element (Figure 17 c)). Note, that the items already processed by the original processing element cannot simply be dropped for the target processing element, since they are required for updating the state. We summarize this subclass of Direct State Transfer Strategies under the term **Post-synchronized State Transfer**.
A precondition for the envisioned synchronization is that the target processing element (at least temporarily) processes the items in the stream faster than the original processing element. Only in this case it is possible to create an output stream, which neither contains neither a gap nor a duplication of results. For synchronization we envision three possible approaches:

- **Input synchronization** will switch between original and target processing element, once the target processing element has overhauled the original processing element in terms of the input items processed. The assumption above ensures that this will happen. Technically this requires additional mechanisms for the synchronization.
- **Semantic result synchronization** will define synchronization points based on the knowledge how the results are created and selects the actual switching point based on those switching points.
- **Generic result synchronization** compares the result streams produced and performs the switch, when the result streams are sufficiently correlated or similar enough for a given period of time.

3.3.7.3 External storage strategy

For very large state information and for state information covering very long time frames, it might make sense to resort to external storage, which is shared between the original and the target processing element. This approach clearly requires more interaction between the methods used in the original and the ones used in the target processing element: they have to agree, in which format accumulated status information is stored and how it can be accessed. The data management layer foreseen in the QualiMaster architecture is closely related to this idea.
For deciding about the use of external storage, it is important to balance several parameters, such as the frequency and time delay, for accessing the external information and the size of the

[16] This might also depend on the application settings, in which the respective pipeline is operating.

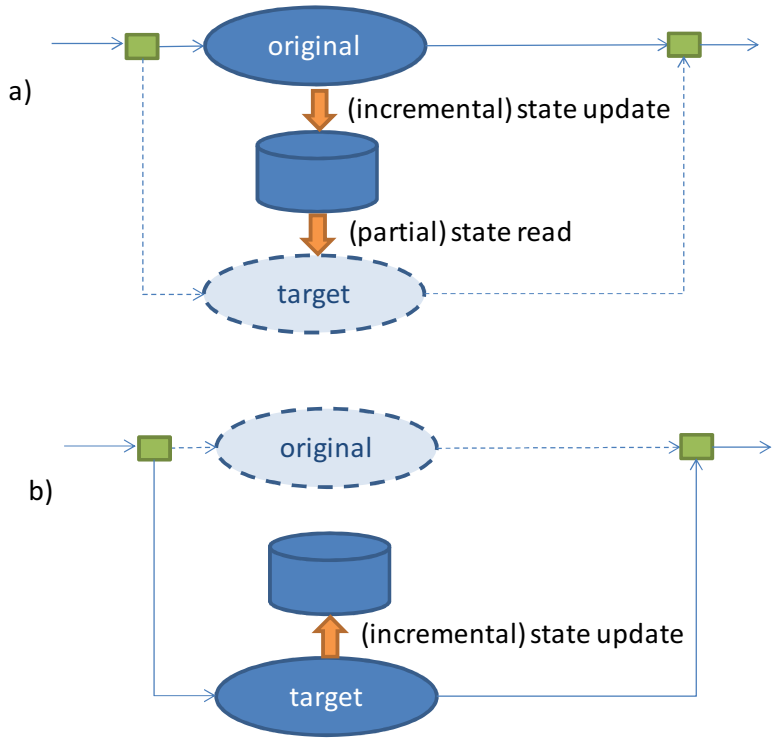

Figure 18: External storage strategy.

accumulated status during stream processing and the implied overhead for a direct status exchange.

During normal processing, the original processing element persists (parts of) its state (regularly) to an external store as shown in Figure 18 a). If all of the accumulated status information is in the external store, switching can be done directly as indicated in Figure 18 b) utilizing enactment pattern EP-2 (see also Section 3.3.4 for more details). In this case switching time is equal to the time the target processing element needs for loading enough status information from the external store (a refinement of [23]) such that it can start its processing. This can be considered as setup time t_{setup} as in the case of the warming up strategy. Since writing into the storage requires some overhead, there might also be status information, which is not (yet) in the external storage. This might be information queued (or prepared) for being written out or more short term information, which is supposed to be aggregated (e.g., to hourly statistics) before being written to external storage. In the case of queued information, the original processing element can be passivated leaving it still time to write out its status, before the target processing element takes fully over.

For short term status information, which is not in the external store one of the strategies described above can be chosen.

3.3.8 Conclusions and Future Work

Enacting adaptive changes on a real-time data stream processing system aiming at gap-free enactment is a challenge. As a basis to make rational decisions about the design of enactment strategies and related adaptation protocols, we presented the results of an initial analysis of the enactment patterns introduced in D4.1. Basically, the results show that enactment in Storm typically comes with a basic command delay of 20-30 ms. While the results for the simple algorithms and parameters are encouraging, the experiments also show that even for a “simple” algorithm switch among real-world sub-topologies, further patterns must be considered such as the effect of buffering. The analysis there also provided evidence for assumptions made for observations on real-world pipelines. Actually, the gap-free change of the parallelization in a Storm cluster is a technical challenge. Therefore, we started to perform experimental

modifications in Storm, which we also consider as a last resort for the algorithm switch among sub-topologies. Moreover, we designed and discussed alternatives for state transfer among algorithms in order to speed up the enactment or even to achieve gap-free enactment.
In the future, we will continue this analysis and work on solutions for the detected issues. This will also act as a foundation for realizing the state transfer strategies described in this section.

3.4 Adaptation specification

In deliverable D4.1, we discussed the design of the adaptation cycle in QualiMaster, the Runtime Variability Instantiation Language (rt-VIL) to be used as a control language for specifying the adaptive behaviour of the QualiMaster infrastructure, and, the intended integration. As discussed in D4.1, rt-VIL combines concepts from adaptation languages, in particular Stitch [6], namely strategies and tactics, with concepts from the Variability Instantiation Language (VIL) [11, 12], which we use for (pre-runtime) instantiation of software product lines, such as the QualiMaster pipelines (as detailed for the infrastructure derivation process in D5.2). In the mean time, rt-VIL has been realized and (initially) integrated with the QualiMaster infrastructure as we will detail in Section 4.5. In this section, we discuss additional concepts that are beneficial as part of rt-VIL. We identified these concepts while describing the first adaptation specifications for the QualiMaster priority pipeline using rt-VIL.
The new concepts are:

- Explicit **monitoring mapping phase** supporting the configuration of runtime quality parameters (Section 3.4.1).
- **Transactional change history** enabling rollback and simplifying the development of adaptation specifications (Section 3.4.2).
- **Calls as parameters** increasing the reuse of existing model parts (Section 3.4.3).
- **Type aliases** simplifying the use of complex adaptation- and instantiation specific types (Section 3.4.4).

In the remainder of this section, we briefly summarize the high level concepts of rt-VIL presented in D4.1 and detail then new concepts.
According to the concepts presented in D4.1, the variability modeling language IVML [11, 13, 14, 19] is used to describe the Configuration Meta Model with consistency constraints and, in terms of a specific configuration the relevant structural aspects for the adaptation such as the pipelines and quality constraints. The Configuration Meta Model already defines variables that represent the system state at runtime, i.e., it allows expressing quality constraints before runtime of the platform and takes these constraints into account as soon as the system state is known through monitoring. The adaptation is then defined using rt-VIL, i.e., it takes the system state characterized by monitored information into account and modifies only those parts of the configuration to indicate the new desired system state that shall be enacted. Before enactment, the constraints are validated in order to ensure that the enactment will lead to a valid configuration. As the modifications and the related enactments can be considered as instantiation at runtime, it is natural to us to use the existing Variability Instantiation Language (VIL) as a basis for rt-VIL.

3.4.1 Mapping Monitored Information into rt-VIL

Mapping information from the QualiMaster Monitoring Layer into the execution environment of rt-VIL (in the Adaptation Layer) needs some specific considerations. Basically, it appears to be an obvious approach that the monitored information is transferred using some QualiMaster specific program code. However, programming the traversal of a (typed) topological configuration such as QualiMaster pipeline involves a certain effort, as the related interfaces of EASy-Producer are rather generic, i.e., handling different specific types becomes quickly rather complex, and using the interfaces requires explicit exception handling. Furthermore, evolving the (pipeline) configuration model along with the code is a rather tedious and error-prone task.

As alternatives, rt-VIL now offers two ways of realizing the mapping of monitored information adequately, namely

- As a *language capability*, i.e., the mapping is directly specified in rt-VIL. Basically, VIL as the underlying language of rt-VIL is designed to support traversals of (typed) topological configurations, in particular relying on the dynamic dispatch capabilities of VIL rules. This allows specifying conveniently type-specific instantiation, e.g., specific instructions for processing elements, sources and sinks. Furthermore, specifying the link between monitoring and adaptation in rt-VIL allows domain users to adjust and extend the mapping along with modifications and extensions of the underlying Configuration Meta model. One specific extension example is the monitoring of additional quality parameters (REQ-C-11 in D4.1), which requires the definition of additional runtime variabilities in the Configuration Model as well as handling them in the adaptation specification. Here, the QualiMaster Infrastructure Configuration Tool (QM-IConf) can assist domain users, as the required changes can be performed in a consistent manner, shielding the user from technical details (REQ-C-14 in D4.1).
  Thus, we introduce an optional explicit monitoring mapping phase into rt-VIL. Akin to other optional capabilities in rt-VIL, the mapping phase is defined by overriding the predefined rule

  ```
  bindValues(Configuration config, mapOf(String, Real) bindings)
  ```

  Mapping the monitoring values can now happen within this rule, e.g., also changing the scale of individual values if needed for specifying the adaptation. Furthermore, defining the mapping can be stated in a rather domain-specific way by using specific types provided by the system under adaptation. In Figure 19, we depict an rt-VIL example for mapping the pipeline quality parameters / observables to runtime variables. This example uses a domain-specific helper type named `FrozenSystemState` to access the monitored values.

```
bindValues(Pipeline p, FrozenSystemState state) = {
   String n = p.name;
   p.latency = state.getPipelineObservation(n,
       TimeBehavior.LATENCY);
   p.throughputItem = state.getPipelineObservation(n,
       TimeBehavior.THROUGHPUT_ITEMS);
   p.throughputVolume = state.getPipelineObservation(n,
       TimeBehavior.THROUGHPUT_VOLUME);
   p.accuracyConfidence = state.getPipelineObservation(n,
       FunctionalSuitability.ACCURACY_CONFIDENCE);
   p.accuracyErrorRate = state.getPipelineObservation(n,
       FunctionalSuitability.ACCURACY_ERROR_RATE);
   p.capacity = state.getPipelineObservation(n,
       ResourceUsage.CAPACITY);
   p.executors = state.getPipelineObservation(n,
       ResourceUsage.EXECUTORS);

   map(Source s : p.sources) {
       bindValues(s, p, state); // dynamic dispatch
   };
}
```

Figure 19: Binding monitoring values in rt-VIL for the QualiMaster infrastructure.

- Through the *language infrastructure*. While the explicit mapping of monitoring values in rt-VIL is extensible and can help understanding the actual adaptation for a technical

expert, it may also be an obstacle to performance, as also unneeded monitored values are transferred. Thus, as a programmed alternative, one can use the rt-VIL language infrastructure to perform an on-demand mapping. Actually, in VIL configured values can be obtained through specific types that are created dynamically based on the Configuration Model, e.g., through types such as `ProcessingElement`, `Source` or `Sink`. Accessing configured values happens through specific operations of these types. If the monitored information is associated to qualified names uniquely identifying the individual elements of a pipeline (in particular in the multi-pipeline case), these accessor operations can be modified in a way that they return the monitored value. To construct these qualified names, either the pipeline elements must refer to their containing pipeline (this is currently not the case, see D4.1) or (domain-specific) structural information about the pipeline model parts (the pipeline definition is in the same IVML namespace) must be taken into account.

Currently, we consider the explicit mapping based on the language capabilities as more appropriate, but we may switch to the programmed language infrastructure based approach if we any experience performance problems.

3.4.2 Transactional Change History

In D4.1, we indicated that changes to the runtime configuration leading to inconsistencies may be repaired or reverted. As repair operations can be rather domain specific, we equipped rt-VIL with a generic default behaviour in case that the reasoning detects consistency issues. Basically, we track every change of the runtime configuration performed by rt-VIL. To be able to revert changes caused by individual strategies or tactics and to consider an alternative strategy or tactic on a lower ranking position, we designed a transactional history for the changes to the configuration. Upon execution, strategies and tactics open a transaction, which can either be reverted if validation of the respective strategy or tactic fails, or committed. Currently, we consider this as the default strategy to handle detected configuration inconsistencies, so that the domain users do not need provide a certain specification. In addition, a rt-VIL specification can override the default behaviour if needed.

For enactment, the runtime configuration can be projected to the changed parts of the configuration, i.e., only variables changed due to the execution of strategies or tactics (not by the mapping of monitoring values discussed in Section 3.4.2) become visible in order to simplify the specification of the enactment. However, in the case of topological configurations such as the QualiMaster pipelines, the projection to changed variables is not sufficient. Let us consider that the adaptation would require the switch of a certain algorithm in an algorithm family (enactment pattern EP-2). Then the projection would return exactly the changed runtime variable holding the actual algorithm of a family, but the context of the variable, i.e., the family and the containing pipeline would be omitted. As the context can be important during enactment, e.g., to traverse the changed pipelines and to schedule wavefront adaptations, rt-VIL now provides access to the context in terms of configured IVML references, i.e., the containing topologies and the specification of enactments can focus on the changes while taking the context of the change into account.

3.4.3 Calls as Parameters

As part of our work on the adaptation scenarios, here AS-1, we developed an initial algorithm to analyze the parallelism in a Pipeline / Storm topology. Actually, this algorithm must traverse the topological structure of a given pipeline. As also the mapping of the monitored information, the enactment of the adaptive decisions as well as the algorithm assignment at the start-up of a pipeline (adaptation scenario AS-2) require a traversal, we recognized that the respective VIL rules (for dynamic dispatch) are repeated in multiple places of the rt-VIL specification. As this is a clear obstacle to consistency, reuse and evolution, we decided to extend VIL/rt-VIL/VTL with the concept of functions that can be passed as parameters, namely call parameters. This is

inspired by programming languages such as C (function pointers) or functional languages such as Lisp (lambda functions), which even have been adopted in recent versions of the Java programming language.
Basically, a function parameter allows passing a function *A* (in VIL/rt-VIL a rule, in VTL a sub-template definition) into a function *B* and to call *A* from within *B*. In case of topology traversals, this enables to separate the walkthrough operations from the calculation / collection of the result, which happens in another function given as a parameter to the traversal function. Figure 20 illustrates the basic form of this language concept in terms of a generic traversal of a QualiMaster pipeline defined by the `traverse` rule, which receive a parameter of type `callOf`, a (function) call parameter is determined by the rule `func`, which collects in this example just the nodes in the sequence of traversal. The application of this generic traversal is indicated by the comment at the bottom of Figure 20, i.e., by executing `traverse` on a specific pipeline in combination with `func`.

```
traverse(Pipeline pip, sequenceOf(PipelineNode) result,
    callOf(PipelineNode, sequenceOf(PipelineNode)) call) = {
    map (Source src : pip.sources) {
        traverse(src, result, call);
    }
}

traverse(PipelineNode node, sequenceOf(PipelineNode) result,
     callOf(PipelineNode node, sequenceOf(PipelineNode)) call) = {
     call(node, result);
}

traverse(ProcessingElement elt, sequenceOf(PipelineNode) result,
    callOf(PipelineNode node, sequenceOf(PipelineNode)) call) = {
    call(elt, result);
    map (Flow f : elt.output) {
        if (!done.contains(f.destination)) {
            done.add(f.destination);
            traverse(f.destination, result, call);
        }
    }
}

// further cases through dynamic dispatch

func(PipelineNode node, sequenceOf(PipelineNode) result) = {
    result.add(node);
}

// application: traverse(p, func);
```

Figure 20: Generic traversal of a topology with call parameters.

In contrast to existing languages, which typically bind function parameters statically, VIL/rt-VIL performs dynamic dispatch even on functions passed through a call parameter, i.e., the rule `func` in Figure 20 can be overridden by more specific types for node in order to perform more specific actions. As VIL/rt-VIL also considers refining scripts, later extensions, e.g., to the Configuration Meta Model, can easily be considered.

However, the traversal illustrated in Figure 20 is still limited to the stated parameters of the call, i.e., collecting a different result is not possible in this example. This level of flexibility can be achieved in VIL/rt-VIL through the combination of a type that can hold any instance (called `Any` in VIL, similar to Object in Java) and dynamic dispatch on (function) call parameters. Revisiting the example in Figure 20, the call type can also be

```
callOf(PipelineNode, Any)
```

so that (after changing the `traverse` rule accordingly) the result instance and the function passed as arguments finally determine the operation for `call` to be executed during the execution. As a consequence, the different forms of traversals in existing VIL/rt-VIL scripts can be unified and (even late domain-specific) extensions to the Configuration Meta Model can be considered.

3.4.4 Type aliases

Actually, the examples shown in Section 3.4.2 suffer from repeated complex parameter types, which are already slightly simplified using `Any` instead of a specific type. This is similarly to the specification of the QualiMaster infrastructure derivation process, where complex map and collection types are used. As a conclusion of this observation, we decided to extend VIL/rt-VIL/VTL with the concept of a type (alias) definition akin to IVML, which, in turn, was inspired by the respective concept in the C programming language. Revisiting Figure 20 again, we can now declare for example

```
typedef TCallType callOf(PipelineNode, Any);
```

defining the new type `TCallType` as an alias of `callOf(PipelineNode, Any)` and use the alias type accordingly in the traverse rules

```
traverse(ProcessingElement elt, Any result, TCallType call) = {
    //...
}
```

3.4.5 Future work

The current version of rt-VIL provides a solid basis for the specification of the adaptation in QualiMaster. As part our actual work on the adaptation scenarios, we specified strategies and tactics as well as entire algorithms in rt-VIL. Future work will be focusing on the integration of optimization algorithms to determine the most beneficial plans, on integrating the prediction of quality parameters as well as further domain- and application-specific functionality. As part of this work, we aim at understanding which parts are better done within rt-VIL itself, within the rt-VIL library as types or operations, within the rt-VIL runtime environment, the (IVML) reasoning support or within the QualiMaster infrastructure. Furthermore, we aim at understanding how to differentiate reusable generic functionality, domain-specific (stream-processing related) functionality and application-specific functionality. Some components which deserve such an integration will be presented in the next section. Furthermore, we aim at evaluating the approach as well as the scenarios and their performance of rt-VIL as well as the overall benefit of the adaptation in QualiMaster.

Finally, we started to develop initial concepts for the cross-pipeline and reflective adaptation. While basics of both tasks from the work plan have already been considered in the basic design of rt-VIL, e.g., to develop a resource distribution algorithm for one and multiple pipelines in rt-VIL, to prototype the start-up scenario for multiple pipelines or to foresee that an rt-VIL model can be changed (at runtime) by the reflective adaptation, more conceptual work will be done in the next months and aligned with the existing concepts and tools.

As a further contribution, WP4 started to perform a mapping study for acquiring performance and data quality knowledge from the algorithm, enabling the tradeoff-making for algorithm families and the prediction of changes of the performance / data quality of entire pipelines.

Based on this mapping study, the measurement and collection of adaptation knowledge will be performed in a close collaboration of WP2 and WP4 in the next months and integrated into the domain-specific part of rt-VIL.

4. Component Realization

So far, we presented the evolution of the concepts foreseen by WP4 to achieve quality-aware configuration and adaptation. For turning these concepts into reality, WP4 realized related (infrastructure) components and tools and prepared them for the integration with the QualiMaster infrastructure. In this section, we provide an overview on the individual components, their realization and (where applicable the current) validation state. Following the sequence of topics used so far, we discuss in this section the following:

- IVML reasoner used as a common component in quality-aware configuration and adaptation (Section 4.1)
- QualiMaster Infrastructure Configuration Tool QM-IConf (Section 4.2)
- Adaptive Crawling enabling dynamic pipeline sources (Section 4.3)
- Social Web Event Prediction supporting the proactive pipeline adaptation (Section 4.4)
- Adaptation Specification and Instantiation language rt-VIL (Section 4.5)

4.1 IVML Reasoner

As briefly introduced in D4.1, IVML is a highly expressive language that provides modeling concepts and elements for creating complex Configuration Meta Models and related Configurations. As IVML supports Non-Boolean types, collections, user-defined types such as compounds, quantors and iterators over ground instances as well as user-defined functions supporting dynamic dispatch, this implies a non-trivial problem of checking whether a specific configuration is valid or to derive unspecified values from already given configuration values (value propagation). To solve these tasks, we designed and implemented the IVML reasoning component (aka SSE reasoner) that is integrated in the EASy-Producer Tool and, thus, in the QualiMaster Infrastructure configuration tool (QM-IConf) and in the QualiMaster infrastructure.
In this section, we discuss first the historical background of the IVML reasoner in Section 4.1.1, then the reasoning process underlying the actual version used in QualiMaster in Section 4.1.2, the actual state of the realization in Section 4.1.3 and recent results from performance experiments in Section 4.1.4. We conclude in Section 4.1.5.

4.1.1 Historical Background

In its history, the IVML reasoner evolved from an integrated customized third-party tool to a unique built from scratch component. Due to complexity and expressiveness of IVML most existing solvers or model checking tools cannot be applied for model validation and value propagation as we showed in an extensive analysis in [8]. As a result and by taking into account experience drawn from several industrial projects, we conclude there that a rule (evaluation) engine is a solid basis for the IVML reasoning support. However, to achieve encompassing reasoning, in particular to also support rather rare configuration situations, we expect that a combination with a conventional solver could be beneficial, e.g., working on a reduced instance of configuration provided by the rule engine or by integrating solving capabilities into a rule engine.
Following this hypothesis, our initial implementation was based on the Jess rule engine[17] due to its established efficiency and wide applicability in academia. However, the actual license of Jess, which prevents the inclusion into an existing (open source) system, caused us to switch to Drools Expert[18], an open source rule engine implementation, unfortunately accompanied by a loss in performance.
In the QualiMaster project, two further issues aggravated the downsides of using Drools as a basis for reasoning:

[17] http://www.jessrules.com
[18] http://www.drools.org/

1) The modeling concepts we apply for enabling the configuration of complex topological configurations of pipelines such as typed references, container iterators and dynamically dispatched user-defined functions (as detailed in D4.1) are not adequately supported by Drools and would require significant mapping effort to emulate them using Drools concepts. Furthermore, these IVML concepts require deep control over the rule execution process, which is not accessible to us as it is a black box.
2) Although we improved our initial Drools implementation based on regular performance experiments, the time needed for translating and loading the IVML model into the Drools knowledge base and to perform reasoning would not allow runtime reasoning as we envision it as part of the adaptation cycle introduced in D4.1.

Finally, we decided as part of our work on configuration and adaptation components in QualiMaster to equip EASy-Producer with the capability to evaluate individual (constraint) expressions and, in turn, to realize on this basis a reasoner that does not depend on any third-party tools. The IVML reasoning capability becomes a fundamental part of QM-IConf and the adaptation cycle in the QualiMaster infrastructure.

4.1.2 Reasoning Process

The reasoner is designed to conduct validation of IVML model as well as value propagation, i.e., the derivation of actually unknown values from given values through (propagation) constraints. As input, the IVML reasoner receives an IVML Configuration Meta Model and a specific Configuration, which structurally complies with the given Configuration Meta Model. As explained in D4.1, an IVML model is stated in terms of a project, the modularization unit in IVML. Such a project defines its own scope of imports of other IVML projects, type definitions, typed variables and constraints. Three basic rules apply for IVML models:

- A variable (including slots of a compound type) may have a default value, more precisely a default expression, which can refer to other default values. Default values must be evaluated before any other scope constraint is evaluated.
- Except for the default value explained above, the actual value of a variable can only be changed once within the scope of the same. In other words, a default value can be overridden and an already assigned value can only be overwritten by the same value in the same project.
- Imported projects can contain values that are needed to evaluate the constraints in the importing project and, thus, must be evaluated before the constraints of the importing project. Except of project imports[19], which are evaluated in the given sequence and before other constraints, there is no predefined sequence for the constraints in an IVML model. This allows the domain engineer to define constraints without much knowledge about the actual evaluation and reasoning process. Variables in imported projects (if visible) can be overwritten once in the importing project.

Basically, the constraints in an IVML model are either assignment constraints that derive values from expressions and assign them to a variable or validation constraints that check the validity of a configuration.

The reasoning process in the IVML reasoner consists of three main phases:

1. **Pre-processing** – gathers information on the constraints and builds the constraint base to reason on.
2. **Constraint evaluation** – evaluates the specified constraints including re-evaluation of dependent constraints upon change of individual variable values until convergence.
3. **Post-processing** – creates the reasoning result, in particular detailed information about failed (sub-)expressions and their involved variables.

[19] Actually, IVML also provides the concept of nested eval scopes, which indicate a similar precedence as import statements. As we do not use evals in QualiMaster, we will not consider this concept any further.

We will now provide details on the individual reasoner phases.

During **pre-processing**, all constraints in the current scope are collected and added to a constraints base. After collection, the constraints base consists of:

- *Default value constraints* – value assignment expressions for default values of variable declarations. For consistent handling of the reasoning process, these assignments are represented internally as constraints.
- *Type constraints* – consistency constraints defined along with type definitions, e.g., in the QualiMaster Configuration Meta Model the `NonEmptyString` type.
- *Project scope constraints* – constraint directly defined in the project scope.
- *Compound constraints* – constraints defined for variables of compound type. These constraints become only effective if a variable of the respective compound type is defined. In particular, compound constraints include the assignment expressions for the default values of compound slots.
- *Constraint variables* – constraints assigned to a constraint variable. As illustrated in D4.1, IVML has the capability of overwritable constraints represented as variables of type `Constraint`. Actually, the content of such a variable is treated as an usual constraint and evaluated along with the other constraints (unless changed through an assignment constraint). In QualiMaster, we use this specific form of constraint for two purposes. 1) As EASy-Producer can assign customizable messages to individual variables, we utilize the constraint variable to obtain a user-supporting message in case that the respective constraint fails. 2) We utilize collections of constraints to allow the user to configure SLA constraints for certain model elements such as sources, sinks or processing elements.
- *Annotation constraints* - include the default values, assignments and validation of IVML annotations[20], which allow an orthogonal classification of variables. In QualiMaster, we use annotations for indicating runtime variables, i.e., variables that become effective at runtime of the infrastructure through monitoring and adaptation.
- *Imported constraints* – type, project, compound constraints or constraint variables defined by imported projects, i.e., no default value constraints.

As next step we create a variable-constraint mapping between the variables used in a constraint and the constraint itself. Actually, this mapping is inspired by the well-known RETE-algorithm [16] and leads to a simplified way of ensuring constraint re-evaluation upon value changes of dependent variables.
After pre-processing is done and all relevant information on the actual constraint base is collected, the **constraint evaluation** phase starts. Within this phase, constraints in the constraint base are evaluated and, if required, re-evaluations due to changes of dependent variables are performed. As indicated above, the constraint evaluation phase effectively starts at the leaves of the import hierarchy and performed bottom-up. Constraint evaluation stops if the reasoning converges, i.e., no more constraints are scheduled for evaluation. In more details:

- If the value of any variable in a constraint is not defined, the constraint is not evaluated at this point.
- If the evaluation of a constraint fails it is registered as failed constraint. If a value assignment fails, e.g., as a value already has been assigned in the same scope, the variable is registered as a failed assignment.
- If a constraint evaluated successfully it is registered as valid constraint (i.e., removed from failed constraint register). If a value of the variable is changed during the evaluation, the dependent constraints stored in the constraint base are scheduled for re-evaluation.

[20] As the original concept name “attribute” overlaps a frequently used concept in feature modeling, we recently renamed this concept to “annotation”.

When all constraints in the constraints base of the current scope are evaluated, they are filtered to determine those constraints that should be passed on as imported constraints. Default constraints, annotation handling constraints and all simple assignments (assignments that hold only constant values) are filtered out.
In the end all relevant variables of this particular scope are frozen. Freezing might affect all variables, only specified variables or variables that meet a defined condition (managed by annotation validation constraint in the freeze block)
After all project scopes are evaluated, the **post-processing phase** starts. In this phase all failed assignments (variables) and constraints are converted into a reasoning result message and provided as a reasoning output. If none of the constraints failed, the message indicates that there is no conflict and the configuration can be considered as consistent.
Actually, the full process of **normal reasoning** is needed only if intentionally all constraint shall be evaluated, e.g., to validate a new configuration without actually modifying it. If we can trust in the actual value of certain IVML variables, we can speed up this process through **incremental reasoning for runtime**. This is the case in runtime reasoning as described by the QualiMaster adaptation cycle in D4.1. There, the configured values used for platform instantiation are considered as immutable, and, thus they are frozen before instantiation. Thus, the Monitoring Layer and the Adaptation Layer just need to fill the non-frozen runtime variables with actual values and the reasoning can leave out the default values and the plain value assignments (used to describe a configuration in IVML) for frozen variables.

4.1.3 Realization State

In this section, we provide a brief overview on the actual support of the constraint evaluation and the reasoning for IVML modeling concepts. Most of the IVML concepts were already briefly introduced in D4.1. A more detailed explanation is given in [19]. The concepts and the realization state ("+" for realized, "-" for currently not supported) is illustrated in Table 3 and Table 2. This indicates that the IVML reasoner and its underlying constraint evaluation are rather advanced and will be completed soon to provide full IVML capabilities to QualiMaster and for future extensions of the Configuration Meta Model.
Akin to EASy-Producer and its languages such as IVML, VIL and rt-VIL, the IVML reasoner is subject to automated regression testing on IVML models during continuous integration. Currently, we validate the IVML reasoner with 76 test cases, 10 of them are defined or based on different versions of the QualiMaster Configuration (Meta) Model.

4.1.4 Reasoning Performance

For reasoning on configurations, performance, in particular execution time is one of the crucial indicators. This is in particular true for reasoning at runtime. In this section, we describe now two forms of empirical performance results: 1) on typical experiments for Software Product Line configurations, and 2) distinct experiments on the QualiMaster configuration.

4.1.4.1 Configuration Experiments

In Software Product Line engineering, the complexity of a configuration reasoning problem is typically expressed in terms of the variable-constraint ratio, i.e., as described in [24] variables and constraint ratio is typically 10:1 in large-scale real world variability models. We described an initial experiment to test reasoning performance on IVML models in [8]. There, we performed reasoning on several artificially generated models with different variable-constraint count:

- 10 variables and 1 constraint
- 100 variables and 10 constraints
- 1000 variables and 100 constraints

IVML concept	Status
project	+
boolean	+
integer	+
real	+
string	+
enumerations	+
container	+
type derivation and restriction	+
compounds	+
null values	+
decision variables	+
constraints	+
constraints as variables	+ (currently be re-evaluated during reasoning)
configurations	+

Table 3: Realization status of IVML core concepts.

IVML concept	Status
annotations	+
extended compounds	+
referenced elements	+
project versioning	+
project composition	+
project interfaces	+
partial configuration	+
freezing configurations	+
partial evaluation (eval blocks)	-

Table 2: Realization status of advanced IVML concepts.

The variables are of type Boolean, Real and Integer. Three levels indicate the complexity of generated constraints based on the number of operations performed in the constraint.

- Level 1: Simple constraints of 2 variables/constants one Boolean operator.
  Example:
  `a and b` (for Booleans), `a > b` (for Integers and Reals).
- Level 2: Concatenation of 3 variables/constants by 2 Boolean operators.
  Example:
  `(a = b) xor (c <= 2).`
- Level 3: Concatenation of 5 variables/constants by 4 Boolean and arithmetical operators.
  Example:
  `(a <= 15) xor (b - c >= d)`

The measures were collected by executing each model ten times for every reasoner (Jess, Drools, Drools v2 and the IVML reasoner), average time calculates. Table 4 and Figure 21 present the results. The test were conducted on Intel Core i7-3520M CPU 3,90 GHz, RAM 6,00 GB, Windows 7 64-bit.

Test results represented in Table 4 represent performance issues of historical IVML reasoning support development described in Section 4.1.1. We see that Jess outperformed both versions of our Drools implementations. Only with the introduction of the custom designed IVML reasoner we were able to achieve better or comparable results observed for Jess.

Nr	Number of elements		Jess	Drools	Drools v2	IVML reasoner
	Variables	Constraints	Reasoning time (milliseconds)			
Complexity Level 1						
1	10	1	147	1778	47	1
2	100	10	88	2132	234	24
3	1000	100	375	8333	1841	305
Complexity Level 3						
4	10	1	34	1895	142	1
5	100	10	92	2677	218	11
6	1000	100	853	14460	2106	331

Table 4: Reasoning performance of IVML reasoner and the historic reasoners.

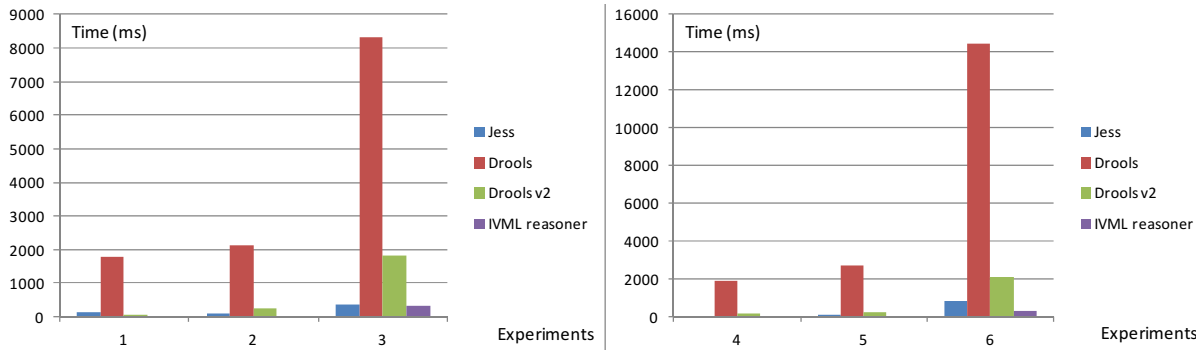

Figure 21: Comparison of the IVML reasoner and the historical reasoners.

For a more detailed evaluation and comparison we performed an additional series of experiments on artificially generated models with only Boolean variables, Level 3 constraints and variable-constraint ratio of 1:3 and 1:1.
The experiments were performed on the machine described above. Due to the different support of IVML concepts, we focused this time on the two most recent approaches, Drools v2 and the IVML reasoner. We executed the reasoner on each model 10 times. We also measured not only the total reasoning time, but specific time for all three reasoning phases introduced above (as well as comparable phases for Drools v2). Table 5 summarizes the results.

Number of elements		Reasoners							
		IVML reasoner				Drools v2			
Variables	Constraints	Total (ms)	Preprocessing (ms)	Evaluation (ms)	Postprocessing (ms)	Total (ms)	Preprocessing (ms)	Evaluation (ms)	Postprocessing (ms)
variables : constraints = 1 : 3									
100	300	40	30	8	2	3151	3135	16	0
300	900	251	229	17	5	10390	10312	78	0
500	1500	614	587	21	6	15609	15484	125	0
1000	3000	2632	2581	44	7	35536	35224	312	0
1500	4500	5878	5803	65	10	62711	62243	437	31
variables : constraints = 1 : 1									
100	100	25	18	6	1	1451	1435	16	0
300	300	111	98	11	2	4212	4166	46	0
500	500	257	241	12	4	6136	6105	31	0
1000	1000	1014	991	16	7	12527	12465	62	0
1500	1500	2176	2149	22	5	21201	21060	141	0

Table 5: Detailed performance test.

As shown in Table 5 and Figure 22, the actual version of the IVML reasoner outperforms Drools v2 significantly in total reasoning time. The main reason is a more efficient and controlled pre-processing phase of the IVML reasoner. Drools Expert creates a KnowledgeBase[21], a repository of all the application's knowledge definitions. It contains rules, processes, functions, and type models and aims at speeding up the reasoning process for the application area Drools is designed for. This is done internally as a black box process and takes up to 80% of the pre-processing time.

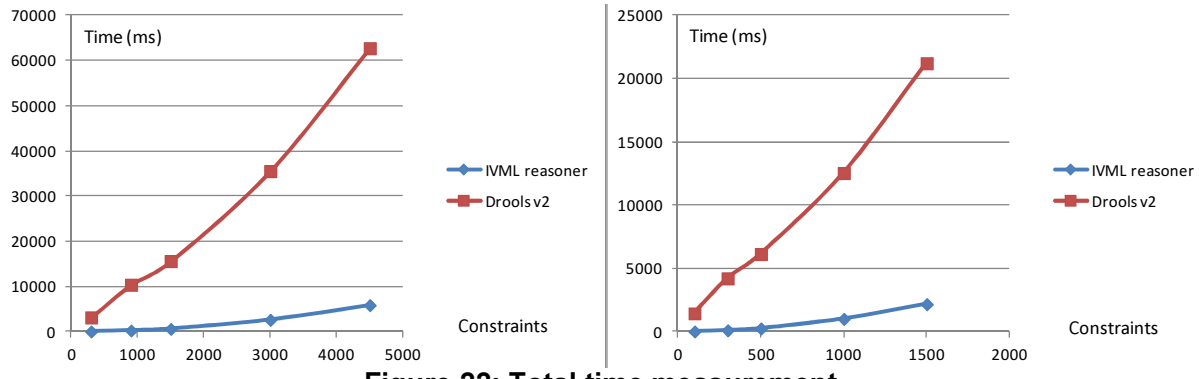

Figure 22: Total time measurement.

[21] http://docs.jboss.org/jbpm/v5.1/javadocs/org/drools/KnowledgeBase.html

The constraint evaluation is also more efficient in IVML reasoner (Figure 23).

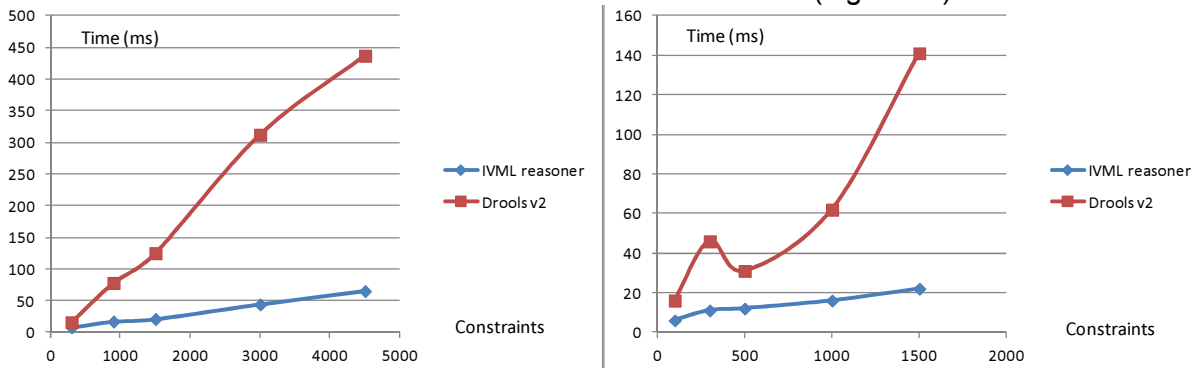

Figure 23: Constraint evaluation time measurement.

The only phase in which Drools v2 outperforms the IVML reasoner is the post-processing phase. This is due to a more complex processing of failed elements in the IVML reasoner, when the reasoning result contains not only the failed element itself, but a textual comment and a detailed hint where the problem is located.

4.1.4.2 Experiments on the QualiMaster Configuration

So far, we presented results for artificially created models. In the QualiMaster project in particular the (runtime) reasoning time on the QualiMaster Configuration (Meta) Model is of particular interest. This differs from the previous models in the sense that it consists of less variables, may scale to arbitrary size due to the openness of the topological pipeline configuration and contains non-trivial and complex constraints (as we explained in D4.1 for the pipeline constraints). In this experiment, our goal is to analyze the performance of the IVML reasoner for both normal and incremental reasoning types. The normal reasoning typically happens on user side in the QualiMaster infrastructure configuration tool (QM-IConf), while incremental reasoning is part of the Monitoring and the Adaptation Layer. We performed the experiments in the same way as previous discussed above, i.e., we use the same computer and for 10 executions of the experiment we report the average reasoning time.

The IVML reasoner performs the reasoning operations on the QualiMaster Configuration (Meta) Model in very good reasoning time of less than 200 ms. The results are shown in Table 6 and Figure 24. As indicated above, incremental reasoning works on a partial set of relevant constraints and, thus, can avoid the evaluation of the more than 2000 (instantiated) constraint of the Configuration (Meta) Model. Actually, incremental reasoning is 37% faster than the normal one. That allows the reasoner to be used both in building a configuration in interactive manner, for headless infrastructure instantiation and for runtime validation as part of the QualiMaster adaptation cycle.

	Normal	Incremental
Number of variables	176	176
Number of (instantiated) constraints	2531	141
Reasoning time (ms)	191	121

Table 6: Normal and incremental reasoning comparison.

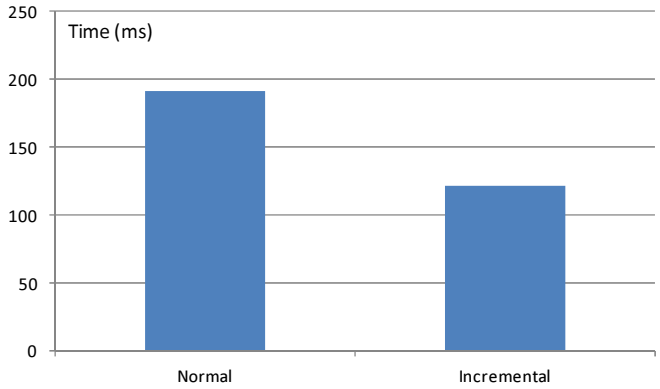

Figure 24: Normal and incremental reasoning comparison.

As part of our experiments, we also figured out that internal logging in the reasoner during development can slow it down significantly. Basically, for logging we compose complex strings indicating the constraint under reasoning, the actual variable settings etc. For disabling the reasoning, we used the constant-on-compile technique [26], i.e., forced the Java compiler to remove all logging statements from the code based on a constant. Our experiments show that the IVML reasoner without logging is by almost 40% faster that with logging (see Figure 25). Based on this all logging is disabled in all releases. To ensure this, we plan to apply EASy-Producer to instantiate a debugging enabled reasoner for releases and nightly builds. Actually, these results inspired us to add a debug / development mode for pipelines to the QualiMaster Configuration (Meta) Model as described in Section 3.2.6.

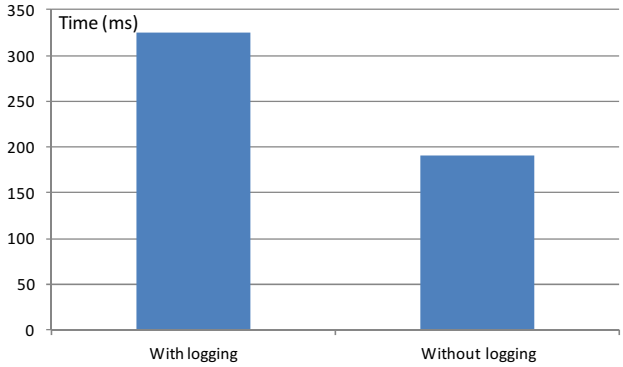

Figure 25: Effect of deactivated / removed logging calls on reasoning.

4.1.5 Future Work

Although the reasoning on Configurations in general and on the QualiMaster model in particular is quite fast and provides the required IVML reasoning capabilities for the QualiMaster project, we plan to improve the IVML reasoner in several dimensions:

- Currently, the reasoner re-evaluates too many constraints as e.g., indicated in Table 6. Here, we aim at optimizing the dependency structure and the constraint base handling, e.g., by recent rule evaluation concepts for selecting constraints for re-evaluation.
- Include optimizations for runtime reasoning, e.g., to exclude even more complex constraints defined only on frozen configuration values.
- Perform specific reasoning operations to support the adaptation, e.g., the derivation and validation of adaptation plans in rt-VIL.

The IVML reasoner is part of EASy-Producer and was made available as part of the recent (Open Source) release in July 2015.

4.2 QualiMaster Infrastructure Configuration Tool

The QualiMaster Infrastructure Configuration Tool (QM-IConf) supports the domain user in configuring the QualiMaster Infrastructure. In particular, it enables users to define pipelines in a graphical way using a drag & drop editor and it allows users to check the validity of the configuration using the IVML reasoner (Section 4.1).

While simple modifications of the Configuration Meta Model can easily be handled by QM-IConf itself, as it interprets the structure of the Configuration Meta Model during tool runtime, more complex changes such as introducing arbitrary types as discussed in Section 3.2 require program changes of the QualiMaster specific editors in QM-IConf. Thus, we evolved QM-IConf along with the Configuration Meta Model so that input / output items can be named, (domain-specific) data types can be added or modified and data types can be used in input / output items.

Moreover, as indicated in D4.1, we extended QM-IConf with a constraint editor, allowing the user to define structural as well as SLA constraints on the model elements, the individual pipelines and their pipeline elements. Again, this editor integrates functionality provided by EASy-Producer, here partial / embedded xText editing capabilities on IVML constraint expressions, but also requires specific integration into the QM-IConf editors. Figure 26 illustrates the constraint editor in the context of the priority pipeline. Basically, the constraint editor (lower part of the dialog in Figure 26) inherits also the highlighting and content-assist capabilities from EASy-Producer and, thus, is able to support the user in specifying full OCL-like IVML constraints (although we expect that for SLA constraints simple relational expressions of variables and constants or derived values will be sufficient). In addition to the plain editor, the constraint editor dialog also displays the available variables in the actual context of the Configuration Meta Model.

As prescribed by the QualiMaster architecture presented in D5.1/D5.2, the QualiMaster Infrastructure Configuration Tool QM-IConf acts as a frontend for rt-VIL at design / development time. The underlying functionality for developing and maintaining rt-VIL scripts is provided by the Configuration Core as described above by reusing the rt-VIL language implementation. Akin to

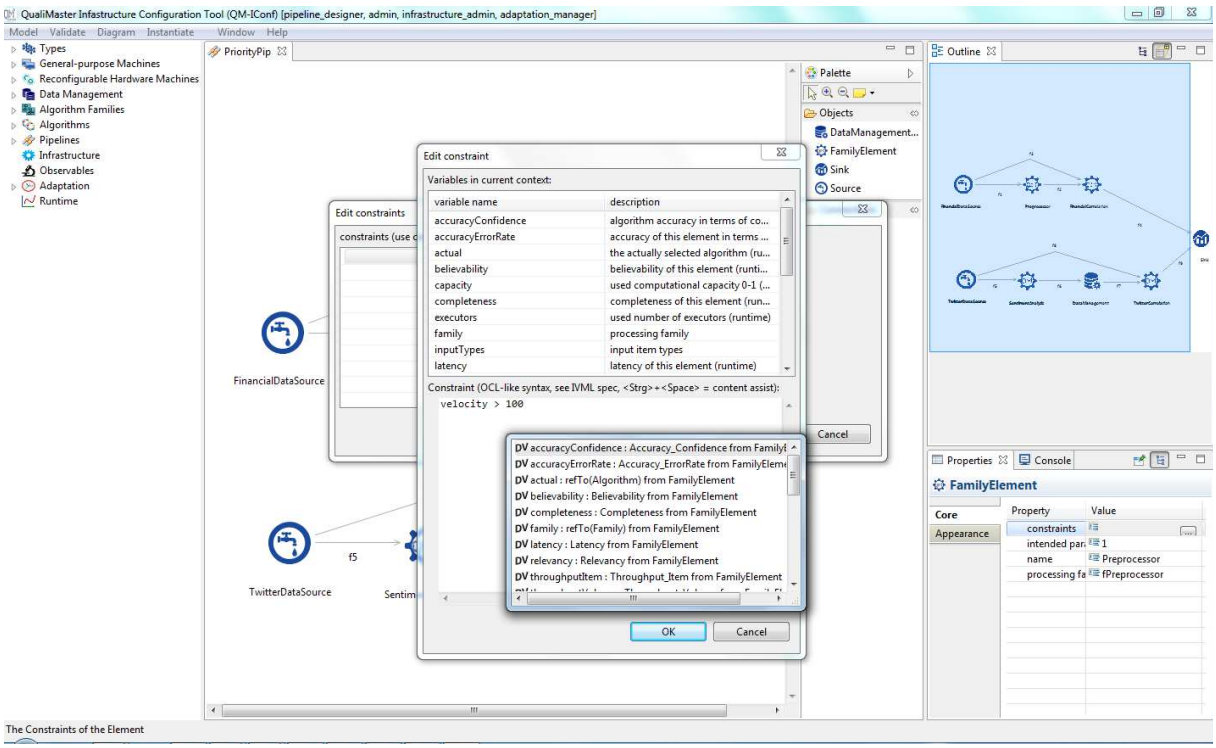

Figure 26: Pipeline constraint editor with content assist and constraint selection dialog.

the arbitrary types for pipeline elements, the graphical configuration tree of QM-IConf has been extended to provide access to the adaptation specification and its simulation environment as shown in Figure 27. As already indicated in D4.1, we consider rt-VIL as the expert level of defining or maintaining the adaptive behaviour in QualiMaster. Further levels for users with less experience as requested by REQ-C-14 in D4.1 will be realized in the remainder of the project and be made available in the same category of the configuration tree, hiding the rt-VIL editor, which will then only open on a specific request.

Due to the internal changes in QM-IConf in order to comply with the Configuration Meta Model, the QualiMaster partners needed to replace their local installation several times, in particular including their local workspaces and the specific (uncommitted) changes they made to the Configuration. To avoid such replacements of the entire tool and the workspace, we decided to include QM-IConf into the continuous build environment for the QualiMaster components, to collect upon successful builds all required bundles and to create a continuous (nightly) Eclipse update site for QM-IConf (as we already do for EASy-Producer). Based on this update site, we enabled the self-update capabilities of QM-IConf, i.e., the underlying Eclipse RCP platform. Although this complicates the release process of QM-IConf, as the bundles must be explicitly installed into an empty RCP environment, it allows our partners to stay up-to-date with co-evolution of the Configuration Meta Model and QM-IConf.

Furthermore, the QualiMaster consortium decided to provide a demonstration release of QM-IConf to the interested public. As external parties do not have access to the Subversion repository storing the QM-IConf model (actually, it is located in the QualiMaster development repository), we added a demonstration mode to the QM-IConf tool, so that we can ship an appropriate version of the Configuration (Meta) model with the tool. In detail, the demonstration mode does not require a login, but also disables the model update / commit, the RCP update and enables a reset functionality for the model, so that the user can recover even from extreme accidental changes. Along with the demonstration video[22] on QM-IConf, this version of QM-

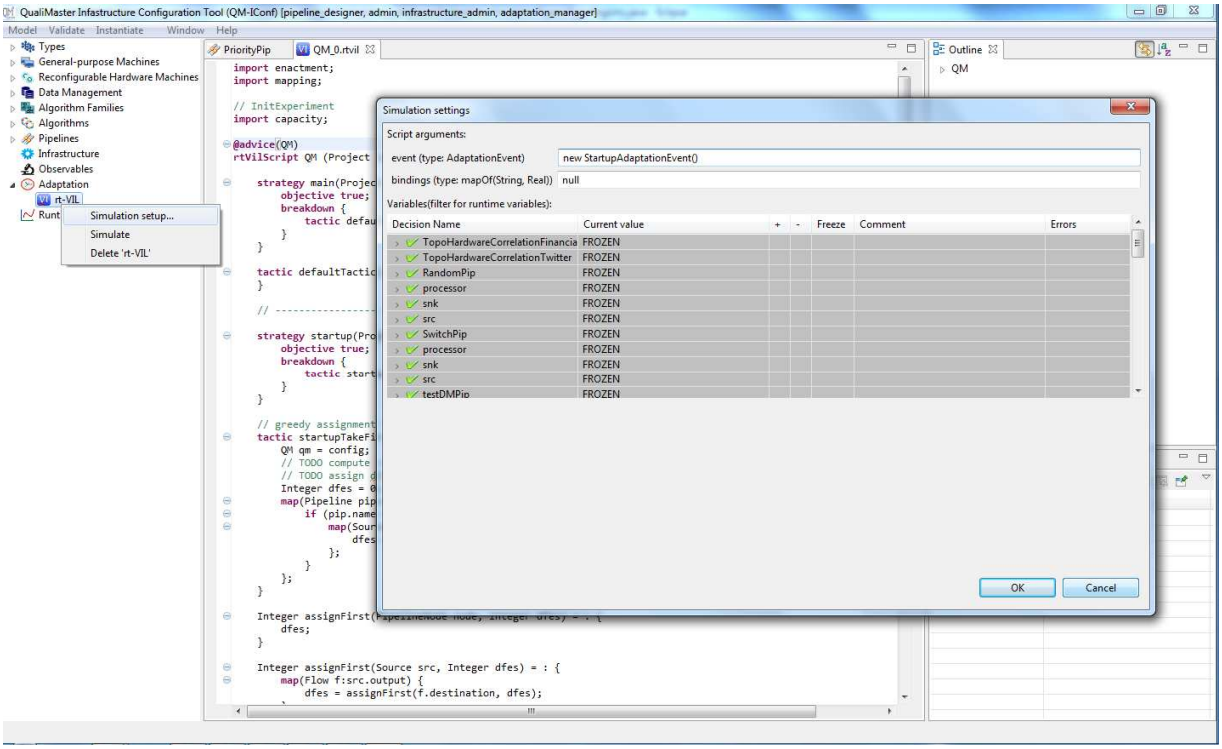

Figure 27: rt-VIL editor with simulation settings dialog and rt-VIL context menu (overlay for illustration).

[22] http://qualimaster.eu/?page_id=247, https://www.youtube.com/watch?v=iwwCvJzD31k

IConf has been made available through the QualiMaster Web site[23] as well as a specific Web site on topological configuration at SUH[24]. Furthermore, in July 2015, a new (Open Source) release of the Configuration Core EASy-Producer has been published and announced.
Although QM-IConf was significantly extended, further work is needed, to fulfil the requirements from D1.1/D1.2 and to improve its user support. The following improvements were either suggested by the reviewers or, in the meantime, by the partners, in particular during the technical work day at the Chania consortium meeting in May 2015:

- *Support for simplifying the configuration of individual algorithms*, in particular their input- and output items as well as the runtime parameters is in development. Initially, an upload functionality into the pipeline elements repository was foreseen, but due to the decision of the consortium to rely on Maven, directly selecting a Maven artefact and interpreting its contents seems to be more appropriate. Therefore, we started to equip the algorithm artefacts with a specific manifest, which either points to the distinct (Java) algorithm so that it's actually supported interface can be derived by analyzing the code or which contains the full interface specification (in case of a hardware-based algorithm). Here, the technical side, i.e., reading the manifests, retrieving the dependent artefacts and the code analysis is realized, while the integration into QM-IConf is under development.
- *Support for specifying Maven artefacts.* Currently, the user needs to input Maven artefact specifications as a string consisting of the group identification, the name and the version of the artefact. However, this is rather inconvenient and error prone, as the user needs either to know the repository structure by heart or to utilize a web browser and to copy the respective specification parts into QM-IConf. Thus, we plan to follow the suggestions of the partners to realize a selection component, which allows the user to select from the repository tree as far as possible rather than typing a full Maven artefact specification.
- We plan to modify the *reasoner integration* so that it is automatically executed when saving the model rather than on explicit request. Also here, the partners made first positive experience if a modelled configuration, which looks consistent at a glance, does not correctly translate into code, e.g., as during the infrastructure derivation process Maven complains about inconsistencies. Running the reasoner when saving the model can prevent this (as also suggested by the partners).
- First steps have been undertaken to highlight validation errors in the graphical user interface of QM-IConf rather than showing just a list of errors. While marking the individual editor components (such as text fields) or even graphical elements in the pipeline editor such as pipeline elements and data flows is working, we are currently extending the post-processing step of the reasoner so that more detailed information about failed constraints and variables is provided. This integration with QM-IConf is ongoing work.

In addition, dependent on the changes we decide for the Configuration Meta Model as discussed in Section 3.2.6, further capabilities will be integrated into QM-IConf.

4.3 Adaptive Crawling

In this section we present the current state of the adaptive crawling component with a focus on first evaluation results and a comparison of different strategies for the adaptation of data streams. An initial description of the component was given in D2.1 in Section 5.1.

[23] http://qualimaster.eu/?page_id=256
[24] http://www.sse.uni-hildesheim.de/topological-configuration

4.3.1 Adaptation Strategies

A first set of strategies on how to adapt a stream of collected messages from Twitter to get a stream with high recall and precision was described in Deliverable 2.1. The two main strategies described there are the

- Adaptation of the stream based on co-occurring keywords (or hashtags / cashtags) which are monitored within sliding windows of different sizes.
- Expansion of the used keywords for the filtering based on background knowledge from DBpedia or Wikipedia.

One of the main problem of these two strategies is the fact that adding a very generic keyword like “apple” can result in a stream which is no longer focused on the main areas of interest.
One way to deal with this problem is to evaluate new keywords first before they are added to the real stream used in the QualiMaster infrastructure. This evaluation can be done by using just this keyword as source for a new stream and evaluating the new stream based on its relevance for the given domain.
Another way to evaluate new keywords is to add these keywords to the stream and evaluate how the stream changes over time due to the inclusion of the new keyword. In case the overall focus of the stream changes to much the keyword can be removed and marked as irrelevant to prevent the use of the keyword in the future.

4.3.2 Evaluation Measures

In D2.1 we analyzed how different strategies perform in terms of relevance of the collected tweets. We showed that the collection co-occurring terms is a very promising approach for expanding the number of used keywords while keeping the relevance at a relative high level.
For this section we will not only focus on the evaluation of the relevance of the collected tweets, but also include the latency for adapting the stream to an upcoming event. These metrics are connected to each other; a faster reaction to new keywords in the stream comes at the cost of possible wrong adaptations, which would reduce the relevance. Since the relevance of the stream is always dependent on what a user might consider as useful it is important to evaluate how broad or narrow a stream gets, dependent on the chosen parameters.
For measuring relevance we manually evaluated, whether the suggested keywords are useful for expanding the stream. In comparison to the automatic evaluation this gives very direct insight into the influence of different parameters on the resulting adaptation.
For measuring the latency for the adoption of the stream we decided to use a set of well known events, which are related to a defined set of keywords. In a scenario like this we can perform various experiments and analyze how different parameters influence the time it takes for the adoption.

4.3.3 Used Datasets

For evaluating different parameters and different quality metrics we mainly used the following two different datasets:

- Dataset-1 is a collection of stock related tweets gathered between October 2014 and June 2015. It consists of about 5,000,000 million Tweets. A detailed description of this dataset can be found in D2.2.
- Dataset-2 was collected during the group phase of the soccer world championship in 2014 and contains about 17 Million Tweets. The tweets were collected using a set of keywords related to Soccer and Brazil.

The different datasets allow us to set up various experiments and repeat them with different parameters. One alternative to this controlled experimental environment would be to use several adaptive crawlers at the same time with varying parameters and compare the results afterwards.

4.3.4 Evaluation Results

For the evaluation we focus on the quality of the resulting stream in terms of Relevance and Latency of the adaptive streaming.
For measuring the relevance of the stream we focus on the possible candidates for adaptation. These candidates can be hashtags or cashtags as well as normal keywords. The first selection of these candidates is based on a sliding window. The sliding window moves over a defined number of tweets, or over a defined period of time. All keywords and hashtags belonging to the tweets inside the sliding window are stored in a Hashmap containing the keywords and the number of occurrences inside the sliding window. If the number of occurrences exceeds a certain number, the corresponding Hashtag or keyword is considered to be very relevant for the stream and added as a filter.
The two main parameters we can change in this evaluation scenario is the size of the window and the threshold for the occurrences within the window. We analyzed different sizes and different thresholds in terms of their usefulness for an adaptive streaming approach.

4.3.4.1 Evaluation of Relevance

The first evaluation was performed on Dataset-1. We run our adaptive crawling algorithms over the data and evaluated how many of the candidates where related to the stock market. We choose to vary the window size between 1000 and 100,000 and the threshold between 10% and 80%.
Table 7 shows the number of possible candidates for the different setups. It is obvious that larger windows decrease the number of possible candidates, as well as larger thresholds. It is more unlikely for a window of a size of 10000 tweets to be filled with 10% of tweets containing a certain keyword than it is for a window of size 10000. The main advantage of larger windows is the reduced vulnerability against spam. Our observations of Twitter showed that spammers tend to create short bursts of irrelevant messages in a short time. These messages are capable of filling up a small sliding window and leading the stream towards irrelevant content. Due to the short time periods in which these spammers are active, larger windows are not that easily flooded with spam messages.
Besides the number of possible candidates we also analyzed how many of these candidates are relevant for the given domain. This test was performed manually by looking a 100 random selected candidates and validating their relevance. The results of this analysis are shown in Table 8. We can see that the results show some patterns, for small windows of size 1000 – 5000 the best ratio is gained with a threshold around 20%. With larger windows, this value decreases.

		Threshold Precentage									
		5	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**
Window Size	**1000**	494	268	165	64	38	34	23	22	22	2
	2000	319	191	73	29	24	7	2	1	1	1
	5000	184	108	33	9	2	1	0	0	0	0
	10000	141	59	14	2	1	0	0	0	0	0
	15000	81	39	8	1	1	0	0	0	0	0
	20000	91	22	5	1	1	0	0	0	0	0

Table 7: Number of Candidates for Stream Expansion

Window Size	Threshold Precentage 5	10	20	30	40	50	60	70	80	90
1000	11%	11%	17%	15,60%	8,57%	8,82%	4,34%	4,54%	4,54%	0,00%
2000	11%	17%	20,55%	10,34%	8,33%	0,00%	0,00%	0,00%	0,00%	0,00%
5000	21%	25%	27,27%	11,11%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
10000	20%	27,11%	21,40%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
15000	24,69%	31%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
20000	29%	27%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%

Table 8: Percentage of relevant candidates for Stream Expansion.

For verifying the evaluation results we performed the same set of experiments on Dataset-2, the results of these experiments are comparable to the ones with Dataset-1 and show that many possible selections of windows size and threshold may be considered to be used, depending on the requirements of the user.

4.3.4.2 Measuring Adaptation Latency

For the evaluation of the time, it takes for the stream to be adapted to a new keyword, we performed a series of experiments on Dataset-2. The advantage of this Soccer World Cup Dataset is the existence of a number of well-defined events, namely the different Games during the tournament, all these games were commented using special hashtags indication the two teams of a match. For instance, a hashtag like #BraGer indicates that the content of the Tweet is related to the game between Brazil and Germany. For a series of 10 Games from different phases of the tournament, we analyzed if the adaptation can find the game and how long before the match the adaptive stream would use the hashtag as a new filter depending on window size and threshold.

Window Size	Threshold Precentage 0.1	0.5	1	5	10	20	30	40	50	60	70	80	90
2000	14	12	12	10	9	8	7	7	7	7	7	7	7
5000	13	12	11	8	8	7	7	7	6	6	6	5	5
10000	12	11	10	8	7	7	6	5	4	4	4	4	4
20000	12	10	9	7	7	5	4	4	4	4	3	2	1
40000	11	9	8	7	5	4	4	2	2	2	2	2	1

Table 9: Found matches (out of 16).

The results of this series of experiments are shown in Table 9 and Table 10. We can see that smaller windows with smaller thresholds can find more of the events, this is expected as described in the previous section. The average amount of minutes was in most of cases around 30 minutes which would be enough time for adding the required hash tags to the filters of the stream.

4.3.5 Conclusion and Future Work

In this set of evaluations we performed a series of experiments in order to choose good parameters for a running adaptive streaming component. While the final choice of the parameters always depends on the requirements of the user, this set of evaluations allows us to

	Threshold Percentage													
	0.1	0.5	1	5	10	20	30	40	50	60	70	80	90	Average
2000	44,24	24,23	28,50	46,82	35,20	18,15	18,61	19,19	19,80	20,76	17,18	17,84	18,34	**25,30**
5000	29,72	36,41	29,72	17,93	33,45	19,80	17,47	18,84	19,69	20,99	23,25	19,61	19,78	**23,59**
10000	24,23	29,72	46,82	33,45	19,80	18,84	20,99	19,61	24,99	25,12	25,20	25,41	25,66	**26,14**
15000	28,50	46,82	35,20	19,80	18,84	19,61	25,12	25,41	26,08	23,05	26,46	37,03	37,22	**28,40**
20000	28,28	35,20	18,15	18,84	19,61	25,41	23,05	37,03	37,35	37,76	41,23	48,27	9,55	**29,21**
Average	**30,99**	**34,48**	**31,68**	**27,37**	**25,38**	**20,36**	**21,05**	**24,02**	**25,58**	**25,54**	**26,66**	**29,63**	**22,11**	

Table 10: Average Adaptation Time before Event (Minutes).

select predefined parameter sets, matching the users requirements. At the current state a completely autonomous adaptation component is still in danger of adding less relevant keywords to the stream, therefore we plan as future work a live on the fly evaluation of new keywords. Additionally we will investigate how the output of other components can be integrated into the adaptive streaming component. Of special interest is here the integration of results from the graph streaming component for evaluating how new parts are connected, the user profiling component for allowing different weights for different tweets based on the user profile and the input from the event prediction and event detection components.

4.4 Event Prediction In Social Media

On-line adaptation of pipeline parameters to an event in the data sets typically results in a delay of certain duration where some event relevant messages can be lost. Such an adaptation can be optimized or carried out before the event, in case latter could be predicted in advance (adaptation scenario A-2). Forecasting of events is typically a difficult and erroneous task. However occurrence of some events (e.g., such as scheduled or re-occurring ones) can be predicted with a certain accuracy. Also, often chains of events can be created based on historical data (flood follows tsunami follows earthquake) and, of course, insider information leaks, which often result in rumors in social media can be considered as event predictions.

In social media people often try to forecast or predict future events (examples shown in Figure 28). The source for predictions can be rumors, thoughts, or some background information. Another source of predictions can be scheduled events such as those appearing in Yahoo calendar.

In the deliverable D2.1 in the Section 6.3.1 we presented use case 8 with calendar based event extraction. Extracted events can be directly presented to financial experts and further used for adaptation of the pipeline. Another source for prediction is content base event prediction, described in Section 4.4.1.

Figure 28: Sources for event forecasts in social media (examples).

Although we doubt that accurate automatic event prediction will be possible in the near future, tools, able to forecast events related to financial areas, could provide an interesting and useful information source for financial experts.
In this Section we will describe a pipeline component for estimating future events for certain market players using social data streams as described in Use Case 11 (according to the numbering in D2.1).

4.4.1 Content Based Event Prediction

In a recent previous work [27] we have shown that the frequency of past event mentions in news corpora can be linked to importance of this event. Also, the longer the time period after the event where it is mentioned in the news, the more influential was the event.
Often people discuss in social media about upcoming events or rumors. Example tweets are shown in Figure 28. A natural language parser can be exploited to parse out the date mentioned in the tweet about a certain stock and save it in a database. In case the extracted date is situated in the future, we can consider this tweet as prediction for some event related to the mentioned market player.

Use Case 11	Extracting future events from social data streams
Description	Detect future date mentions in social media streams as future events
Input	• Social media streams • (optionally) a user query
Steps	1. Receive social media streams 2. Detect future date mentions (such as ”next week” or ”in September”)using NLP parser 3. Acquire context information from the data item 4. Extract related market player name from the data item 5. Save mentioned time event along with the context information aggregated from all tweets mentioning this date and market player.
Output	A set of contextual features (e.g. keywords) describing an event, which can be used to adapt the monitoring of news feeds or social media streams in order to focus on the emerging event.

4.4.2 Implementation Details

Accurate date extraction including both implicit (e.g., next week) and explicit (e.g., Mar 24) temporal expressions is vital to our approach. We use the SUTime (Stanford Time Parser) state-of-the art toolkit [5] for this task.
Algorithm 1 describes the main steps. First the market player name is extracted from the tweet, following by a set of date mentioned. Each date is compared to some reference date (in our case with current time). In case the extracted date is situated in the future, it will be added to the calendar as event estimation, relevant for the extracted market player.

4.4.3 Summary

The most important task for the algorithm evaluation is to acquire a suitable ground truth (see Section Event Detection in Deliverable D2.2). Similar to the Event Detection component, our plans are to create a testbed with real events relevant to a set of market players and a set of tweets gathered from that periods in close collaboration with WP5. Our quality evaluation will be in terms of accuracy and visualized through precision recall curves. We will test both algorithms of the family, namely the calendar based and content based approaches. Finally we plan to perform to analyze closer the underlying tweets and gain insights into the structure of

Algorithm 1: Collecting future date mentions from a social stream

Input: M: Social message
$rtime$: reference time (now)
Output: Calendar with predicted dates $C = (d)$

```
1 begin
2     C ← ∅ // Initialization
3     MP ← parseMarketPlayer(M) // get market players' name from the tweet
4     for (date) ∈ NLP.parseDates(M) do
5         if date > rtime then
6             C.add(date, MP, M)
7     return C
```

predictable events and introduce additional measures, such as diversity of event facets we are able to extract.
Additionally, we will focus on the influence of the predicted events on the quality measures of the QualiMaster Quality Taxonomy. This requires not only the collection of test data with real events, but also a detailed description about how certain events influenced the quality parameters. We assume that based upon a data collection like this, a set features can be found, which allows a prediction not only for the time of an event, but also for its possible influence on the pipeline. Then, the event prediction component can provide the Adaptation Layer with distinct triggers on predicted events, their temporal duration and their forecasted impact on certain quality parameters.

4.5 Adaptation Specification Language (rt-VIL)

Based on the initial and refined concepts of rt-VIL presented in D4.1 and discussed in Section 3.4, we realized rt-VIL as a component for both, the Configuration Core (EASy-Producer) and the QualiMaster infrastructure. In this section, we discuss the realization state of rt-VIL. We start with the Configuration Core, i.e., the realization of rt-VIL in EASy-Producer. Then we briefly explain the integration of the Configuration Core with QM-IConf and the QualiMaster infrastructure (to be detailed in the upcoming deliverable D5.3). Finally, we discuss in the state of the generic and extended type system / language library of rt-VIL, which is responsible for supporting domain-specific modeling of the adaptation behavior for the QualiMaster infrastructure and initial measures of the execution performance of rt-VIL.
For the **Configuration Core**, we utilized the existing language infrastructure of VIL as a solid foundation, which already implements the structure and the evaluation of expressions and VIL rules as well as the VIL type system. On top of this infrastructure, we realized the rt-VIL grammar, the related (extended) xText editor, the executable rt-VIL model as well as an (initial) rt-VIL simulation environment allowing the user to execute a rt-VIL specification with given script parameters and (runtime) configuration values. For short, rt-VIL has been implemented as described in D4.1 and Section 3.4 of this deliverable. As part of this, the IVML reasoner (Section 4.1) has been integrated with the rt-VIL language environment so that the validity of the changed configuration can be ensured. As described in D4.1, rt-VIL consults the reasoner as default activity after at the end of a strategy / tactic, which may lead to the detection of invalidities and trigger the transaction based rollback of the configuration described in Section 3.4.2). For testing rt-VIL, we implemented more than 30 language specific test unit cases complementing the more than 250 scenario and unit test cases for VIL and VTL. Akin to the other domain-specific languages of EASy-Producer, these test cases are used for automated regression tests while building EASy-Producer on the continuous integration server of the Software Systems Engineering Group of SUH provided to the QualiMaster partners.

For the configuration of the QualiMaster infrastructure, the user interface level of rt-VIL (language infrastructure including the editor and the simulation environment) have been integrated into the QualiMaster infrastructure configuration tool (**QM-IConf**) as already mentioned in Section 4.2. Furthermore, an initial integration of the non-interactive parts of rt-VIL with the **QualiMaster infrastructure** has been performed, so that the Coordination Layer can load the respective models, the Monitoring Layer can utilize the rt-VIL monitoring mapping (Section 3.4.1) and the reasoner (Section 4.1) to detect SLA violations and, finally, the Adaptation Layer can execute rt-VIL specifications and determine as well as enact required changes to the QualiMaster infrastructure at runtime. We will detail the integration of rt-VIL into the QualiMaster infrastructure in the upcoming deliverable D5.3.

As indicated above, rt-VIL is used in both, the configuration side and the runtime side of the QualiMaster infrastructure. In the Configuration Core (EASy-Producer), rt-VIL acts as a generic language adaptation control language, without particular knowledge about the System Under Adaptation, in our case the QualiMaster infrastructure. In contrast, in QM-IConf the user shall be able to utilize domain-specific concepts such as descriptors for the resources that can lead to adaptations (reflecting the quality taxonomy from D4.1 and Section 2), the adaptation events indicating particular adaptation needs, and the coordination commands for requesting specific enactments from the Coordination Layer. As already indicated in D4.1, these domain-specific concepts become part of a (domain-specific) version of rt-VIL in terms of individual types extending the rt-VIL type system. However, these types must also be backed by the real functionality[25], such as accessing detailed information of the events or executing a certain coordination command in the Coordination Layer. Therefore, we realized a **type library component**, that can be integrated with the QualiMaster infrastructure, QM-IConf and optionally with EASy-Producer. The integration with EASy-Producer is needed to allow experienced users to consistently work on the detailed levels of the Configuration Meta Model, the platform instantiation process as well as on the adaptation specification in the same tool. From the perspective of EASy-Producer, the type library is a QualiMaster-specific extension of EASy-Producer. For QM-IConf and EASy-Producer the type library component plays the role of an OSGi/Eclipse bundle that can be installed into Eclipse RCP (for QM-IConf) or Eclipse (for EASy-Producer). It contains the respective parts of the QualiMaster infrastructure so that the type library can be executed during rt-VIL simulation runs. For the QualiMaster infrastructure, the type library component is deployed as a Maven artefact, which directly binds against the infrastructure functionality. In summary, the type library consists of 10 types representing coordination events, 8 types representing adaption events, 5 types representing observables and, thus, the QualiMaster quality taxonomy and an additional type simplifying the mapping of monitored information. As these types must comply with the actual implementation, we derive the information about the types directly from the actual Maven artefacts of the QualiMaster infrastructure implementation using Java reflection analysis and certain Java annotations to guide the reflection analysis. To ensure the consistency, this can be performed as part of the continuous integration. As mentioned in Section 3.3, in the future we will extend the rt-VIL type library and, for domain-specific functionality also the type library component with optimization functions to support and determine adaptation plans.

[25] In case of the rt-VIL simulation environment, some functionality such as the actual execution of a coordination command shall only be available as a stub as it actually shall not be executed during a simulation, e.g., the actual modification of a running instance of the QualiMaster infrastructure.

5 Summary and Outlook

Quality-aware configuration and adaptation are core topics of the QualiMaster project. They enable flexibility before runtime (configuration) and at runtime (adaptation). In this deliverable, we provided an update on the concepts and the realization state of the components foreseen to realize both, configuration and adaptation.

Regarding configuration, we discussed extensions of the Configuration Meta Model driven by recent experience with the approach, which allow us to model and configure the data processing in QualiMaster in a more flexible way. Subsequently, we evolved and improved the platform instantiation process as well as the user tooling, namely the QualiMaster Infrastructure configuration tool (QM-IConf). One central component of QM-IConf is the IVML reasoner, which was developed, improved and evaluated in the QualiMaster project and showed good performance in practical as well as in experimental settings.

Regarding adaptation, we performed fundamental experiments for the enactment patterns introduced in D4.1, discussed strategies to perform efficient algorithm switching using state transfer, implemented adaptation components such as adaptive crawling and event prediction from Social Web data and realized and validated the initial version of the runtime instantiation and adaptation language rt-VIL. Furthermore, we started realizing the adaptation scenarios. Currently, we concentrated on Changing Data Streams (A-1), in particular regarding parallelization and Requested Resource Allocation (A-2), regarding adaptive pipeline start-up, but the described components also build a foundation for Future Event Prediction (A-2). The scenarios are supported by work on automatically determining and optimizing the parallelization of a pipeline, for which we realized an encouraging prototypic modification of Storm for runtime changes of the parallelization, designed an initial algorithm for resource distribution and realized it in rt-VIL. Although we started with a single pipeline case here, we already planned an extension for the multiple-pipeline case on the entire cluster.

The next tasks of our future work are to continue the analysis of the enactment patterns, in particular regarding combinations of patterns to make the switch of algorithms more effective and to achieve gap-free enactment (using buffering and state transfer). This work may indicate the need to further modifications to the underlying Stream processing system, for which we aim at submitting a patch to the Storm community. A further important topic in the next months is to determine the adaptation knowledge for switching among algorithms and for setting parameters. Therefore, we started a mapping study on automatically obtaining performance and data quality measures. Based on existing work, we will then analyze in close collaboration with WP2 and WP3 the algorithm families to acquire empirical algorithm models. We also will use this as a foundation for the (dynamic) pipeline analysis and turn the knowledge into rt-VIL adaptation specifications, validate and analyze them. Further, we will combine the experience we obtain from this work with the upcoming concepts for cross-pipeline adaptation and reflective pipeline adaptation, where the respective tasks defined in the DoW just started last month.

6 References

[1] R. Agrawal, S. Gollapudi, A. Halverson, and S. Ieong. Diversifying search results. In *International Conference on Web Search and Web Data Mining (WSDM '09)*, pages 5–14, 2009.

[2] J. P. A. Almeida, M. Wegdam, M. van Sinderen, and L. Nieuwenhuis. Transparent dynamic reconfiguration for corba. In *International Symposium on Distributed Objects and Applications (DOA '01)*, pages 197–207, 2001.

[3] H. Berthold, S. Schmidt, W. Lehner, and C.-J. Hamann. Integrated resource management for data stream systems. In *Proceedings of the 2005 ACM Symposium on Applied Computing*, SAC '05, pages 555–562, New York, NY, USA, 2005. ACM.

[4] I. Bordino, Y. Mejova, and M. Lalmas. Penguins in sweaters, or serendipitous entity search on user-generated content. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 109–118. ACM, 2013.

[5] D. Chen and C. D. Manning. A fast and accurate dependency parser using neural networks. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 740–750, 2014.

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

[7] A. Cockburn. *Writing Effective Use Cases*. Addison Wesley, 2000.

[8] INDENICA Consortium. Variability Engineering Tool (interim). Technical Report Deliverable D2.4.1, 2012. http://www.indenica.eu.

[9] D. Dasgupta and S. Forrest. Novelty detection in time series data using ideas from immunology. In *Proceedings of the international conference on intelligent systems*, pages 82–87, 1996.

[10] M. Drosou and E. Pitoura. Search result diversification. *SIGMOD Record*, 39(1):41–47, 2010.

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

[12] H. Eichelberger and K. Schmid. Indenica variability implementation language: Language specification, version 0.93. Technical report, 2014. http://projects.sse.uni-hildesheim.de/easy/docs/vil_spec.pdf.

[13] H. Eichelberger and K. Schmid. Mapping the Design-Space of Textual Variability Modeling Languages – A Refined Analysis. *International Journal on Software Tools for Technology Transfer*, pages 1–26, 2014. published online: http://link.springer.com/article/10.1007/s10009-014-0362-x.

[14] H. Eichelberger and K. Schmid. IVML – A DSL for Configuration in Variability-Rich Software Ecosystems. In *Software Product Line Conference (SPLC '15) Vol. 2*, 2015. to appear.

[15] R. Castro Fernandez, M. Migliavacca, E. Kalyvianaki, and P. Pietzuch. Integrating scale out and fault tolerance in stream processing using operator state management. In *International Conference on Management of Data (SIGMOD '13)*, pages 725–736, 2013.

[16] C. L. Forgy. Expert systems. chapter RETE: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem, pages 324–341. IEEE Computer Society Press, Los Alamitos, CA, USA, 1990.

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

[18] K. Geihs, C. Evers, R. Reichle, M. Wagner, and M. U. Khan. Development support for qos-aware service-adaptation in ubiquitous computing applications. In *Symposium on Applied Computing (SAC '11)*, pages 197–202, 2011.

[19] C. Kröher H. Eichelberger, S. El-Sharkawy and K. Schmid. Indenica variability modeling language: Language specification, version 1.22. Technical report, 2014. http://projects.sse.uni-hildesheim.de/easy/docs/ivml_spec.pdf.

[20] 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.
[21] M. A. U. Nasir, G. De Francisci Morales, D. Garcá-Soriano, N. Kourtellis, and Serafini M. The power of both choices: Practical load balancing for distributed stream processing engines. In *International Conference on Data Engineering (ICDE '15)*, pages 137–148, 2015.
[22] H. L. O'Brien. Exploring user engagement in online news interactions. *Proceedings of the American Society for Information Science and Technology*, 48(1):1–10, 2011.
[23] E. A. Rundensteiner, L. Ding, Y. Zhu, T. Sutherland, and B. Pielech. Cape: A constraint-aware adaptive stream processing engine. In N. A. Chaudhry, K. Shaw, and M. Abdelguerfi, editors, *Stream Data Management*, volume 30 of *Advances in Database Systems*, pages 83–111. 2005.
[24] S. She, R. Lotufo, T. Berger, A. Wasowski, and K. Czarnecki. Variability Model of the Linux Kernel. In *4th International Workshop on Variability Modeling of Software-intensive Systems (VaMoS 2010)*, 2010.
[25] J. Shen, X. Sun, G. Huang, W. Jiao, Y. Sun, and H. Mei. Towards a unified formal model for supporting mechanisms of dynamic component update. *SIGSOFT Softw. Eng. Notes*, 30(5):80–89, September 2005.
[26] M. Svahnberg, J. van Gurp, and J. Bosch. A Taxonomy of Variability Realization Techniques. *Software – Practice and Experience*, 35(8):705–754, 2005.
[27] G. Tran, E. Herder, and K. Markert. Joint graphical models for date selection in timeline summarization. *ACL*, 2015.
[28] S. Vargas. Novelty and diversity enhancement and evaluation in recommender systems and information retrieval. In *ACM SIGIR Conference on Research and Development in Information Retrieval, (SIGIR '14)*, page 1281, 2014.
[29] M. Wermelinger and J. L. Fiadeiro. Algebraic Software Architecture Reconfiguration. *SIGSOFT Softw. Eng. Notes*, 24(6):393–409, 1999.
[30] J. Zhang and B. H. C. Cheng. Model-based development of dynamically adaptive software. In *Proceedings of the 28th International Conference on Software Engineering (ICSE '06)*, pages 371–380, 2006.
[31] C.-N. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen. Improving recommendation lists through topic diversification. In *International conference on World Wide Web (WWW '05),*, pages 22–32, 2005.