Experiences with the Model-based Generation of Big Data Pipelines

Holger Eichelberger[1] Cui Qin[1], Klaus Schmid[1]

Abstract: Developing Big Data applications implies a lot of schematic or complex structural tasks, which can easily lead to implementation errors and incorrect analysis results. In this paper, we present a model-based approach that supports the automatic generation of code to handle these repetitive tasks, enabling data engineers to focus on the functional aspects without being distracted by technical issues. In order to identify a solution, we analyzed different Big Data stream-processing frameworks, extracted a common graph-based model for Big Data streaming applications and developed a tool to graphically design and generate such applications in a model-based fashion (in this work for Apache Storm). Here, we discuss the concepts of the approach, the tooling and, in particular, experiences with the approach based on feedback of our partners.

Keywords: Big Data, stream-processing, model-based development, code generation, Apache Storm.

1 Introduction

Developing Big Data applications is complex. Besides the inherent complexity of the various data processing and analysis algorithms comes the need to deal with the technical complexity of distributed high-performance software architectures. While modern Big Data frameworks like Apache Storm[3], Spark[4], Flink[5], or Hadoop[6], reduce this complexity, they also introduce their own level of complexity. Ideally, data scientists would be able to focus only on the functional complexity while ignoring technical complexity, e.g., how to program the links between different data processors or how to transport complex data types among them. We faced this problem directly in our collaborative projects with researchers from this area. Thus, we decided to pursue the possibility of hiding this complexity and easing the development in the area of Big Data stream-processing. This is a particularly challenging area that gets significant attention due to the ability to provide data processing capabilities at (nearly) real-time. The solution for which we opted — and which we will describe here — is to create a model-based approach that allows data engineers to model a solution on an abstract level that fits their understanding of the data processing pipeline and to relate the various steps in the pipeline to adequate implementations.

[1] University of Hildesheim, Software Systems Engineering, Marienburger Platz 1, 31141 Hildesheim, {eichelberger, qin, schmid}@sse.uni-hildesheim.de

[3] http://storm.apache.org

[4] http://spark.apache.org

[5] https://flink.apache.org/

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

In our research, which was conducted in the FP7-project QualiMaster[7], we actually addressed an even more complex problem as we aimed at the creation of self-adaptive data processing pipelines. Thus, in this project there exist various algorithmic variants , e.g., different algorithms for calculating financial correlations in real-time [Ny16] to be executed on a heterogeneous resource pool consisting of server machines and hardware co-processors. At runtime the layout of the data processing pipeline is constantly optimized to ensure maximum performance. However, here we will focus only on the model-based development of Big Data pipelines (independent of any adaptation).

In the following section, we will discuss the foundations and the meta-model of our model-based approach. We generate the necessary code for the technical parts of the data processing architecture from the models. This will be briefly described in Section 3. Section 4 will then discuss the experiences we made with the approach and, in particular, how this approach contributed to the cooperation with the various data scientists. Finally, in Section 5 we will summarize and provide an outlook on future work.

2 Model-based Design of Big Data Applications

In this section, we detail our approach to model-based development of Big Data streaming applications. We start with an overview on concepts of stream-processing frameworks. Then, we detail the modeling concepts and, finally, the design tool that we realized to ease the creation of the models.

As a foundation for creating an application meta model for the QualiMaster project, we analyzed the data processing concepts used in more than 12 open source and research stream-processing frameworks that we considered as candidates for our infrastructure [Qu14]. During this analysis, we identified as relevant concepts for the meta model two types of stream processors and one common concept. We found approaches:

- Supporting only a **fixed set of pre-defined data stream analysis operators**. Examples are Borealis [Ab05] or PIPES [KL09].
- Focusing on **user-defined algorithms**, such as Storm, Heron [Ku15] or IBM System S [Ge08]. In particular, the algorithms themselves can be distributed over a cluster.

Some recent open source approaches such as Spark and Flink tend to combine both types. As a common concept, all stream processors we examined rely on a data flow graph (sometimes also called network of operators or query plan). A data flow graph consists of operator nodes (data sources, data processors, sometimes also data sinks) connected by data flow edges. In some approaches, like Storm, the data flow graph can be cyclic, i.e., data processors can feed back their results to their predecessors. Technically, the data flow graph is often realized in terms of a program (Spark, Storm, Heron, Flink), through a DSL, e.g., in [Ge08], or as a model [Ab05]. However, none of the approaches supports heterogeneous processing involving hardware co-processors out of the box.

[7] http://qualimaster.eu

Several model-based approaches to stream-processing applications have been developed, which are typically also based on the concept of data flow graphs. For example, Borealis [Ab05] ships with a a graphical editor and interprets the designed query model at runtime. jStorm[8] realizes a Domain-Specific Language (DSL) for Storm, which is interpreted at startup time of the application. Apache Nifi [9] enables the graphical composition of static Big Data streaming applications. Moreover, Guerriero et al. [Gu16] model and generate Big Data applications for different data processing frameworks. However, our approach differs in its specific capabilities for flexibly exchanging alternative data processing algorithms and utilizing a heterogeneous resource pool including hardware co-processors.

The requirements for designing an **application meta model** for the QualiMaster project include the support for user-defined algorithms, flexible (runtime) algorithm exchange as well as a heterogeneous resource pool. The resource pool may include commodity servers and specialized hardware co-processors such as Field Programmable Gate Arrays (FPGAs) [Ny16]. At the heart of our model is a **processing pipeline**, i.e., a data flow graph consisting of data sources, processors and data sinks. For testing, experiments and runtime adaptation, we allow the flexible exchange of the algorithms before and at runtime. Our modeling concept for enabling such exchanges is the **algorithm family**, which represents a group of algorithms performing a similar data analysis task at potentially different runtime characteristics. Thus, in our model, data processors specify the implementing family, which, in turn, defines its constituting data processing algorithms. Algorithms can be simple Java classes, co-processor algorithms, which were designed and compiled for a certain FPGA architecture or distributed algorithms represented as sub-pipelines. Similarly, data sources and data sinks specify their implementation, i.e., components that obtain data items from a data provider or pass data items to an end-user application, respectively. Moreover, the meta model allows capturing available and required processing resources. Available resources can be commodity servers and hardware co-processors. Required resources are declared by the pipeline nodes or by an algorithm if it runs only on a certain co-processor. More details on the underlying meta-modeling approach are discussed in [Ei16].

Not all possible models can be successfully executed by a stream-processing framework. To avoid illegal model instances as well as data processing problems at runtime, we equipped our meta model with a number of constraints. The constraints ensure that:

- A pipeline represents a **valid data flow graph**, i.e., there are no incoming flows to data sources and no outgoing flows from data sinks. Depending on the underlying processing frameworks, cycles in data flows can be forbidden by a further constraint.
- A **valid data flow** from node n_1 to n_2 ensures that n_2 can consume the type of data produced by n_1. Requiring this form of type safety avoids serious runtime errors. While some processing frameworks support such restrictions, e.g., Flink, others like Storm focus on processing untyped objects only.

[8] https://github.com/alibaba/jstorm
[9] https://nifi.apache.org

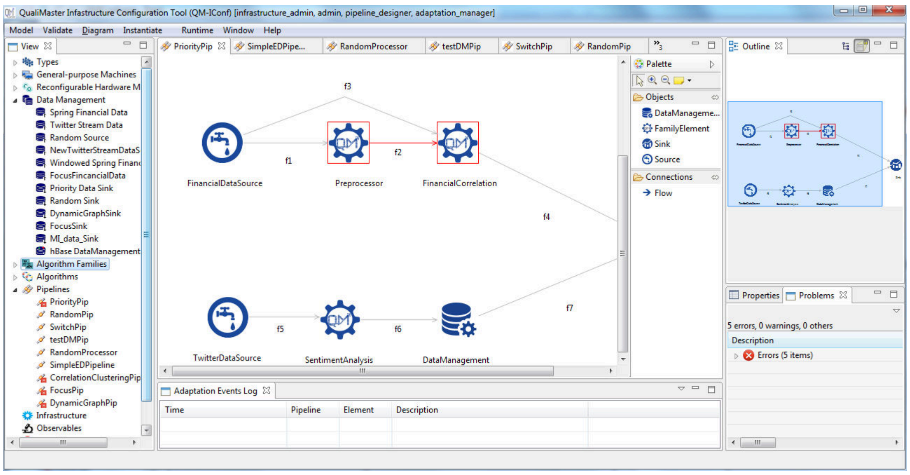

Fig. 1: Interactive pipeline design and generation tool QM-IConf indicating validation errors.

- **Interface compatibility** of exchangeable algorithms hold, i.e., algorithm families and contained members must be interface compatible regarding the consumed and produced types of data.

To ease the creation of application models, we developed a design tool, the QualiMaster Infrastructure Configuration (QM-IConf) tool. QM-IConf is an Eclipse RCP application implemented on top of an own model management infrastructure [Ei16]. Figure 1 depicts a screenshot of QM-IConf. On the left, the tool displays all model concepts ranging from the data types for pipelines over resources, algorithms, algorithm families to pipelines in the suggested sequence of defining them. For pipelines, we integrated a graphical drag-and-drop editor, which we generate using the Eugenia/Epsilon framework[10]. Figure 1 depicts a fragment of a financial risk analysis pipeline. QM-IConf allows validating whether a model fulfills all constraints using the forward-chaining reasoner implemented by our model management infrastructure. For illustration, validating a model with 9 pipelines each with 5 nodes on average takes around 150 ms on a Dell Latitude 6430 laptop. Figure 1 shows a validation error for two pipeline nodes marked by red rectangles. More detailed information on the error is available in the error view in the lower right corner.

3 Model-based Generation of Big Data Applications

In this section, we discuss the code generation for pipelines including the technical parts of the data processing architecture based on the meta-model introduced in Section 2.

Figure 2 illustrates the architecture overview of the model-based generation of data processing pipelines. We consider the execution environments including the stream-processing

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

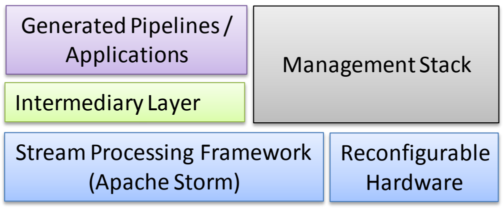
Generated Pipelines / Applications
Intermediary Layer
Management Stack
Stream Processing Framework (Apache Storm)
Reconfigurable Hardware

Fig. 2: Architecture overview.

Framework (Apache Storm) and reconfigurable hardware as a heterogeneous resource pool. Based on the model-based design discussed in Section 2, we generate pipelines as executable code for the underlying stream-processing framework. To structure the generated implementation as well as to simplify the generation and the generated code, we rely on an Intermediary Layer, i.e., a library with extended Storm concepts, such as a processing node which reacts on external signals. On top of the generated pipelines, we provide a Management Stack allowing us to control and monitor runtime execution through QM-IConf (see Figure 1). More specifically, we mainly perform the following generation steps:

- **Interfaces**. For processing nodes in a pipeline, we generate data source, family and data sink interfaces based on their input and output types to enforce type-safe implementation of pluggable components, in particular data processing algorithms. This allows the data engineers to design pipelines first and to implement algorithms later.
- **Data serialization support**. In a heterogeneous processing environment, data including complex nested types must be transferred among various types of processing frameworks: Storm supports default Java serialization and the faster kryo[11] framework, while co-processors are realized in different programming languages. As manually implementing serialization mechanisms is a tedious and error-prone task, we generate the required implementation of type-specific serializers based on the modeled data types, in our case kryo and protobuf[12] for the co-processors.
- **Integration of hardware co-processors**. Hardware co-processors must be fitted into the communication patterns of the processing framework. This requires network connections as well as sender/receiver functionality in terms of pipeline nodes. While the user just models the algorithm, the generation produces transparently the required communication nodes based on the aforementioned serializers.
- **Pipelines**. We traverse the modeled data flow graph of a pipeline starting at its data sources following the data flows to all subsequent pipeline nodes. For each node type, we apply a specific code generation template creating an implementation based on the the respective family interface, which can handle runtime parameter changes and algorithm exchanges. For a hardware co-processor algorithm, we integrate the code generated in the previous step. For the generation of complex algorithms given

[11] https://github.com/EsotericSoftware/kryo
[12] https://developers.google.com/protocol-buffers/

as reusable sub-pipelines, we perform a recursive pipeline instantiation. Finally, we create and execute a Maven build specification, which compiles the code, integrates all required dependencies and derives deployable pipeline artifacts.

The data engineers can execute the generation process through QM-IConf. For example, for the example model described in Section 2 the generation including compilation and download of the most recent versions of algorithm components takes around 4 minutes.

4 Experiences and Discussion

We report now on the experiences we made in applying our approach and the feedback we received from our project partners. 7 data engineers from 3 groups participated, all familiar with Storm, Hadoop, and Java. They modeled 5 different data analysis pipelines. We structure the discussion in chronological fashion, starting with the project initial phase, the introduction of the model-based approach, its application and its evolution over time.

At the **beginning of the project**, the data engineers were rather skeptical about a model-based approach as this was unfamiliar to most of them. As the model management framework already existed before the project, we were able to provide an initial version of the meta model and QM-IConf already after some months. To support our partners applying the approach, we gave some hands-on workshops **introducing the approach**, QM-IConf and how to implement the component interfaces. Here, the data engineers were puzzled about the enforced type safety as this is not required by Storm as well as the (generated) component interfaces as exchanging algorithms was not in their focus. After their first modeled pipeline was successfully generated and executed, the data engineers started to use the approach and to focus more and more on the implementation of the data analysis algorithms. In this phase we received a lot of feedback on desired features, but also on problems in QM-IConf and the generation, which helped us stabilizing the approach.

In **later phases**, the data engineers acknowledged the abstraction level provided by the models and the structuring effect of the interfaces on their work and the usefulness of typed data items. Moreover, as some data engineers never looked into the generated pipeline code and inspected how the algorithm interfaces are used, they were surprised about the ”magic” of transparent data serialization or runtime algorithm changes. This is in particular true for complex pipeline structures, e.g., co-processor integration or efficient algorithm switching [QE16], which require additional pipeline nodes and specific communication patterns. For a specific financial algorithm [Ny16], we compared two variants aiming at the same computation, a generated sub-pipeline and a manual software implementation. We found that both variants produce the same results and operate on the same cluster / distribution settings at the same performance. Thus, the model-based generation successfully shields data engineers from complex or error-prone development tasks and allows working on a higher level of abstraction, e.g., families or sub-pipelines. However, we also noticed negative effects during **integration phases**: Modeling and code generation became the center of the integration work and integrating changes in models by different partners at the same time and fixing issues in the generated code significantly increased our workload.

The realization work in the project follows agile ideas, i.e., changing and restructuring code to introduce new or advanced capabilities is not unusual. In addition to the algorithms, also the meta model, the pipeline models and the code generation undergo an **evolution**. According to our experience, model-based generation helps to quickly and consistently deploy changes and enhancements in a consistent manner across a set of pipelines. This releases the data engineers from maintaining pipeline code over time. Of course, they still have to maintain and optimize the functionality of their algorithms. However, also bugs can easily be deployed into multiple pipelines as an accident. Here, an important support mechanism is **continuous integration**. We apply continuous integration to both, the code base for the algorithm components as well as the validity of the model and the code generation using a headless version of our tool. This ensures the deployment of up-to-date pipelines and acts as an early indicator of integration problems, such as accidental changes in the models. However, intensive unit tests for some components increase the overall build time, so for identifying problems it is faster to change and modify (even the generated) code locally, to perform intensive testing and to integrate the changes back into their algorithms, the model using QM-IConf or the generation templates.

We conclude that model-based development has several positive effects on the development of Big Data streaming applications. Technical benefits include avoiding issues of manually implementing schematic code, easing and speeding up the realization of complex pipeline structures, e.g., algorithm switching or consistently changing and evolving code over time. Our approach allows data engineers to concentrate on their core competences, i.e., developing new data analysis algorithms and composing them to Big Data applications. In our case, the more the data engineers worked with the approach, the more their confidence in stability and correctness increased. However, this also requires an increasing level of quality assurance to avoid that (avoidable) issues affect the achieved trust.

5 Conclusions and Future Work

Developing Big Data applications involves complex tasks, in particular as actual processing frameworks focus on the programming of the applications. Model-based development can release the data engineers from programming tasks and allow them to focus on their specific expertise. In this paper, we presented a model-based approach to the development of Big Data streaming applications. The approach enables flexible exchange of data analysis algorithms at runtime and allows utilizing a heterogeneous resource pool. We discussed an analysis of stream-processing frameworks, our meta model for Big Data streaming applications, tool support for designing streaming applications and the generation process to obtain implementations for Apache Storm. We discussed our practical experiences with the approach and conclude that model-based design and code generation of Big Data stream applications pays off, in particular to realize schematic code or complex capabilities, which are not supported by actual stream-processing frameworks. Moreover, data engineers in our project appreciate the approach and its effects on their work. However, this does not come for free. Effort is needed for creating the models, realizing model transformations and code generation as well as maintaining and evolving the approach over time.

In the future, we aim at the generation of more optimized self-adaptive code, e.g., to increase the efficiency of switching among alternative algorithms. We also consider extending our approach to support alternative stream processors as target platform.

Acknowledgment

We thank Sascha El-Sharkawy, Roman Sizonenko, Aike Sass, Niko Nowatzki and Patrik Pastuschek for their work on QM-IConf. This work was partially supported by the QualiMaster (grant 619525) funded by the European Commission in the 7th framework programme. Any opinions expressed herein are solely by the authors and not of the EU.

References

[Ab05] Abadi, D. J.; Ahmad, Y.; Balazinska, M.; Cetintemel, U.; Cherniack, M.; Hwang, J.-H.; Lindner, W.; Maskey, A.; Rasin, A.; Ryvkina, E.; Tatbul, N.; Xing, Y.; Zdonik, S. B.: The Design of the Borealis Stream Processing Engine. In: Conference on Innovative Data Systems Research (CIDR '05). Pp. 277–289, 2005.

[Ei16] Eichelberger, H.; Qin, C.; Sizonenko, R.; Schmid, K.: Using IVML to Model the Topology of Big Data Processing Pipelines. In: International Software Product Lines Conference (SPLC '16). Pp. 204–208, 2016.

[Ge08] Gedik, B.; Andrade, H.; Wu, K.-L.; Yu, P. S.; Doo, M.: SPADE: The System S Declarative Stream Processing Engine. In: Intl. Conf. on Management of Data (SIGMOD '08). Pp. 1123–1134, 2008.

[Gu16] Guerriero, M.; Tajfar, S.; Tamburri, D. A.; Di Nitto, E.: Towards a Model-driven Design Tool for Big Data Architectures. In: Intl. Workshop on BIG Data Software Engineering (BIGDSE '16). Pp. 37–43, 2016.

[KL09] Klein, A.; Lehner, W.: Representing Data Quality in Sensor Data Streaming Environments. J. Data and Information Quality 1/2, 10:1–10:28, Sept. 2009.

[Ku15] Kulkarni, S.; Bhagat, N.; Fu, M.; Kedigehalli, V.; Kellogg, C.; Mittal, S.; Patel, J. M.; Ramasamy, K.; Taneja, S.: Twitter Heron: Stream Processing at Scale. In: Intl. Conf. on Management of Data (SIGMOD '15). Pp. 239–250, 2015.

[Ny16] Nydriotis, A.; Malakonakis, P.; Pavlakis, N.; Chrysos, G.; Ioannou, E.; Sotiriades, E.; Garofalakis, M.; Dollas, A.: Leveraging Reconfigurable Computing in Distributed Real-time Computation Systems. In: Workshop on Big Data Processing - Reloaded. http://ceur-ws.org/Vol-1558/, 2016.

[QE16] Qin, C.; Eichelberger, H.: Impact-minimizing Runtime Switching of Distributed Stream Processing Algorithms. In: EDBT/ICDS Workshop on Big Data Processing - Reloaded. http://ceur-ws.org/Vol-1558/, 2016.

[Qu14] QualiMaster consortium: Quality-aware Processing Pipeline Modeling, Deliverable D4.1, http://qualimaster.eu, 2014.