Definition of Architecture for Extreme-Scale Analytics
Work Package 4 Task 4.1 Deliverable D4.1

Authors

Ralf Klinkenberg, David Arnu, Edwin Yaqub, Fabian Temme, Mate Torok
RapidMiner GmbH

Antonios Deligiannakis, Alkis Simitsis, Nikos Giatrakos
Athena Research & Innovation Center

Elias Alevizos, Nikos Katzouris
National Center for Scientific Research "Demokritos"

Distribution list:

Group:	Others:
WP Leader: RapidMiner Task Leader: RapidMiner	Internal Reviewer Partner: CMRE INFORE Management Team INFORE Project Officer

Document history:

Revision	Date	Section	Page	Modification
0.1	27/11/2019	1-3	1-15	Creation
0.2	28/11/2019	1-3	1-15	Updated discussion on Big Data platforms and Machine Learning Libraries
0.3	02/12/2019	4	15-19	Architecture Scheme, Connection, Graphical Editor, Manager Components incorporation
0.4.	03/12/2019	4	20-23	Optimizer and Synopses Data Engine incorporation
0.5	06/12/2019	4	23-27	Complex Event Forecasting and Machine Learning Component incorporation
0.6	09/12/2019	5	27-34	Creation
0.7	10/12/2019	6, All	37	Creation, Self-review
0.8	16/12/2019	-	-	Submitted for internal review
0.9	23/12/2019	All	All	Internal review comments incorporated
1.0	27/12/2019	All	All	Final modifications by the Coordinator

Approvals:

First Author: Ralf Klinkenberg (RM) Date: 16/12/2019

Internal Reviewer: Elena Camossi (CMRE) Date: 20/12/2019

Coordinator: Antonios Deligiannakis Date: 29/12/2019

Table of contents:

1 Executive Summary

This deliverable defines the building blocks of the INFORE Architecture, their functionality and their interconnections in the scope **of a holistic, pluggable, extensible INFORE framework** that will evolve to an omnibus solution for extreme-scale streaming analytics.

Aligned with the objectives stated in the project proposal, INFORE aims at:

(i) supporting the non-programmer data analyst in rapid setup of streaming workflows tailored for her application scenario needs by providing graphical workflow design facilities,

(ii) automating the tuning of the underlying Big Data platform infrastructure that materializes the visually designed workflow as well as the provisioned physical resources in a way that optimizes specific performance measures,

(iii) providing real-time, interactive machine learning and data mining tools that can be leveraged by the designed workflows,

(iv) enhanced interactivity via data summarization and approximate query processing techniques,

(v) distributed complex event processing and forecasting techniques to not only detect business events of interest as soon as they occur, but also forecast their occurrence well in advance.

To achieve these goals, the definition of the INFORE Architecture includes the following loosely coupled, modular components: (i) Graphical Editor Component, (ii) Connection Component, (iii) Manager Component, (iv) Optimizer Component, (v) Synopsis Data Engine Component, (vi) Interactive Online Machine Learning Component, (vii) Complex Event Forecasting Component.

The Graphical Editor Component is an extension of the RapidMiner Studio developed in the scope of the project. An elaborate Streaming Nest operator is being developed within the Studio. The Streaming Nest operator is essentially an umbrella encompassing a family of streaming, logical (i.e., abstract, not tied to a particular implementation on a Big Data platform) operators developed in the scope of the project. This family of operators includes both data management operators such as filtering, join, projection, map, reduce, aggregations etc operators as well as logical operators for online machine learning, complex event forecasting and data approximation. Having designed the desired workflow using drag and drop functionality of the Graphical Editor Component, the user proceeds with visually creating connection objects for input, output streams and streaming backends (available Big Data platforms and respective clusters hosting them) in the Graphical Editor Component. These visually defined connection details are internally handled by the Connection Component. Upon submitting the workflow, the Manager Component takes over to convey the submitted workflow to the Optimizer Component. The Optimizer Component returns to the Manager Component a modified, optimized workflow where it has attached execution plan information related to (a) the Big Data platform on which each operator of the workflow will be executed, (b) the cluster resources that will be provisioned, (c) the cluster in which each operator will be deployed in case of multiple geo-dispersed clusters, (d) replacements of exact workflow operators with approximate ones provided by the Synopses Data Engine Component should the user have defined that some predefined inaccuracy guarantees can be tolerated by the application for reducing workflow execution time. The Manager Component may visualize the modified workflow and ask for user approval or execute the actual plan provided by the optimizer. To do so, all logical operators in the execution plan provided by the Optimizer are instantiated by their physical implementations. A dispatcher module as part of the Manager Component submits separate jobs for each Big Data platform and respective cluster, while output streams are provided to the desired applications. In that scope, the physical implementation of approximate query processing operators is included in the Synopses Data Engine Component. The physical implementation of machine learning operators resides in the Interactive Online Machine Learning Component and similarly for the Complex Event Forecasting Component.

This deliverable is in direct relation to deliverables of WP1, WP2, WP3 (up to date, use case requirements have been expressed in Deliverables D1.2, D2.1, D3.1) which aid in realizing the INFORE framework to specific application scenarios. WP5 specifies the internal details of the Optimizer Component starting in Deliverable D5.1 to be submitted in Month 16 of the project. WP6 develops the Synopses Data Engine Component (Deliverable D6.1 on Month 12 together with the current one, which is later enhanced in D6.3), the Interactive Online Machine Learning Component and the Complex Event Forecasting Component described in Deliverables described in Deliverables D6.2 (Month 16), D6.4, D6.5. The first, complete prototype of the INFORE Architecture is presented in the follow up Deliverable D4.2 on Month 16.

2 Introduction

INFORE's project objectives target the processing of large-scale data that is increasingly being produced as streams by a variety of industrial and scientific applications. To handle these massive data streams, transforming them, training and applying analytics functions on them, a flexible, extensible and customizable approach is needed that allows end users to easily model and manage streaming analysis workflows. Considering the evolution of streaming technologies, the subtle differences in their capabilities, as well as the complexity of streaming use cases, INFORE envisions a cross-platform framework approach to create (i.e., design) streaming processes, optimize these at different levels (i.e., functional blocks and process level), execute (i.e., deploy on the most suitable streaming Big Data platform and/or computer cluster) and analyse results in an interactive manner.

In order to effectively realize these design goals, INFORE considers the varying levels of data mining skills among its end-users and target personas. Therefore, the proposed approach takes into account ease of adoption and long-term maintainability as general guidelines. In short, the INFORE framework delivers:

a) A robust visual design *approach*, which *integrates* and *abstracts* various streaming stacks in a *flexible*, *pluggable* and *extensible* manner. This *eases* adoption by reducing pipeline creation to simple drag and drop composition of functionality blocks, while letting the user utilize the functionality of the underlying stacks to the maximum extent.
b) *An optimal configuration management* (consumption and setup) of these processes. The INFORE notion of optimality is quite *holistic* in that it aims at maximizing performance objectives by considering functional alternatives across different Big Data platforms and HPC cluster(s). Optimality also considers non-functional aspects such as results from similar historic executions, scalability requirements, and the technical limitations of available platforms.

The architecture addresses requirements that are derived from various industrial and scientific use cases in the project. These use cases help to materialize the scientific and technological objectives of the project. We describe these objectives in the following, refining the descriptions given in the project proposal, based on the deeper insights gained from the work conducted to date.

I. Real-time, interactive machine learning and data mining tools

A variety of stakeholders needs to consume large-scale streaming data. These include analysts, domain experts and data engineers. These and many other roles highly vary in their data mining skills. A vital objective of the INFORE architecture is to build *interactive tooling* to ease design and management of general as well as machine learning based streaming processes. Further, this tooling should allow *interactive queries* to be performed, leveraging the use of data synopses, when possible. This is critical to allow the user to interactively test different parameters and to gain insights on the data. This is typical in the INFORE use cases. For example, in the INFORE Financial use case, evaluating whether there is systemic risk often involves discovering correlations of different strength over different time periods, which the user can dynamically determine, without the need to continuously process thousands of financial data streams in their entirety. These features extend the traditional approach of other tools, which as of today provide very limited interactivity and tooling e.g., for machine learning on streaming data.

II. Distributed complex event processing and forecasting of future occurrences

A certain set of streaming applications require a sophisticated capability to process high-frequency real-time data for the identification of complex events with very low latencies. Classifying these events and forecasting the likelihood of their occurrence in future time window(s) requires the acquisition and preparation of context data (streams), handling noise, training and applying models at very fast rates since event patterns may indicate sensitive or critical incidents. The INFORE architecture meets this objective by targeting state-of-the-art streaming techniques from the field of Complex Event Processing and Forecasting, blending the use of rule-based deductive learning with more general predictive modelling.

III. A flexible, pluggable and extendable architecture

The INFORE Architecture recognizes the diverse feature set of different software stacks and the rapid evolution of the technology spectrum. Considering this fact, a fundamental objective for the INFORE architecture is to approach maximum flexibility, pluggability, and extensibility without compromising the ease of adoption. This multi-prong objective is achieved by following a component-based (visually usable) design, which provides conceptual pillars (explained in Section 4.3 in more detail) to achieve these design principles.

Flexibility is afforded by means of cross-stack feature integration and real-time (re-)configurability of INFORE process parameters, by means of an intelligent Optimizer Component. A meta-model is being developed that transforms the operator-based visual representation of streaming workflows into a machine-readable representation of process, that is consumed by the Optimizer to perform optimizations on the functional and non-functional aspects of the process. This representation can be further analysed and optimized by considering the existing implementations over different available Big Data platforms and/or HPC infrastructure, the current resource utilization levels, the availability of resources over time, the key performance requirements of the process, and business as well as technical constraints of the backend infrastructures.

In addition to the Optimizer, INFORE's modular design can integrate external components that may be developed by project partners and exposed for consumption as REST[1] API instances. These could be very helpful if certain use case workflows require to make queries to additional components. For example, queries to the Synopses Data Engine, the Complex Event Forecasting Component, or systems based on the Akka streaming platform that are interfaced by a REST API, can be simply added to the workflow.

Pluggability and extensibility are achieved by integrating selected technology stacks, libraries and external systems by loosely coupling them in graphical components called operators. These not just provide a layer of abstraction and hence a common approach to use distinct features of heterogenous underlying technology stacks, but also a convenient method to compose complex analytics functions as reusable pipelines (or processes).

IV. Construction of a framework for supporting non-programmer data analysts to specify processing workflows and data analytics tasks

The INFORE architecture occupies a central focus in the broader INFORE framework. The INFORE framework consists of:

A. A ***visual design approach*** to create streaming analysis workflows using a graphical (virtually code-free), drag and drop mechanism. As hinted to already, this is achieved by connecting graphical objects called operators, i.e. functional modules that allow data retrieval, data pre-processing, data transformations, and analytics functions. This approach provides a comprehensive and thoroughly documented guideline to structure any complex data analysis process by means of:
   a) Building blocks, which are a set of commonly used operators. These promote reusability.
   b) Sub-processes, which help organize a data analysis process into modules or sub-processes. This promotes maintainability, especially when multiple stakeholders jointly design complex processes, and re-usability.
   c) Parameter configurations, which allows to pass or reset values at the process level and at the operator level, respectively, during process design time and process execution time. This promotes customization.

B. A ***concrete implementation*** of all the necessary tooling (including operators, but also related components such as for connection management, deployment, resource monitoring, querying of ongoing execution / log outputs, synopses data engine, optimizer for intelligent (re)configurations, etc.).

[1] REST stands for Representational state transfer (webservices)

In this regard, INFORE is aiming for a full-fledged framework solution that can evolve as a de-facto standard to perform extreme-scale analytics for a variety of streaming use cases. This ideation stems from the scope of INFORE, which allows it to arguably deal with the complete life cycle including design and realization of streaming analysis workflows to deployment, management and evaluation of results.

V. Data summarization and approximate query processing techniques

INFORE deals with a diverse set of use cases from financial, life sciences, and maritime domains. Despite their unique specialities, these use cases have a common requirement in that they all need to deal with massive data streams in real-time. One of the major requirements (and hence, an important objective) is to provide implementation of approximate query processing algorithms that can generate representative summaries on top of streamed data - also termed as Synopses. The Data Synopses Engine is a crucial component in the INFORE architecture, that delivers this capability. INFORE is researching a range of techniques for synopses generation that may range from one-pass synopses over single-source data to more elaborate techniques that monitor cross-source correlations. A well-characterised synopsis may serve as a live view of an ongoing experiment at an intermediate or advanced stage. This plays a decisive role in certain use cases. Hence, this objective is considered a novel contribution in the overall set of INFORE capabilities.

VI. Rigorous testing and evaluation of controlled experiments and reviews by domain experts

INFORE partners plan to subject their use case implementations to a rigorous unit-level, integration-level, and end-to-end level testing, which may require multiple partners (both domain experts and technical experts) to join hands. The evaluation of various workflows under the Life Sciences, Maritime, and Financial use cases are being considered with the objective to understand the limits or boundaries of the INFORE architecture as well as the added value of the INFORE framework, especially in comparison to prior or contemporary art. For this, it is planned to compile a list of functional and non-functional aspects, metrics, or Key Performance Indicators (KPIs) to be able to fairly assess project contributions in terms of use cases or at least certain workflows that are part of the use cases.

In addition, at this stage, various benchmarking methodologies are also being considered. Given a lack of a singular widely adopted standard approach, we are also exploring ideas to formulate a custom approach for testing, evaluation, and benchmarking.

The following section on the state-of-the-art presents a short overview of some of the popular data streaming frameworks and machine learning libraries for streaming data.

3 State of the Art

This section gives an evaluation of already available frameworks and libraries for analysing streaming data and for machine learning on streaming data. Since the goal of the INFORE project is to have a flexible architecture that does not rely on a single technology stack, this evaluation is a very important first step to do. The idea is that for a given workflow visually designed in the RapidMiner Studio software, properly extended with INFORE functionalities, parts of the workflow engaging several operators may be executed in different Big Data platforms, possibly in a distributed way and in parallel. This is the case, for instance, when different implementations of operators exist in different such platforms or when a particular implementation exhibits higher performance in INFORE optimization-related benchmarks.

This section also covers references to related research projects to check for prior work and disseminating features of INFORE.

3.1 Streaming Frameworks

In recent years, many streaming frameworks were developed and have become popular. Especially with the support of big, open source communities and the backing of companies, these frameworks are still rapidly growing. Therefore, a complete overview of all tools and framework is out of the scope of this document. While some projects fill a niche role for very specific needs and demands, others follow a general approach. In this section, we will give a brief overview of some of the more popular and active frameworks.

3.1.1 Storm

The conceptual view of data processing in Apache Storm[2] is represented by a Storm Topology. A Storm Topology is a Directed Acyclic Graph (DAG) that includes Spouts and Bolts. Spouts are data stream sources; each Bolt, in turn, is where the actual processing takes place. Bolts can do anything from filtering, aggregations, joins, interacting with databases, and more, before emitting tuples to other Bolts or to applications.

For the physical execution viewpoint, there are three kinds of nodes on a Storm cluster. The Master node runs a daemon called Nimbus, responsible for assigning tasks to machines and for monitoring for failures. A ZooKeeper coordinates various processes and stores all the states associated with them. Finally, each Worker node runs a Supervisor daemon, which listens for work assigned to its machine and manages Worker processes.

Upon defining the topology, the developer can explicitly set the number of Worker processes (Java Virtual Machines - JVMs). A Worker process belongs to a specific topology and may include one or more Executors. Each Executor is devoted to a Spout or Bolt of this topology.

For each Spout or Bolt, the developer can explicitly declare the number of its Executors (threads). A Spout's/Bolt's definition also allows for setting the number of tasks for the Spout/Bolt. The tasks of a particular Spout/Bolt are running instances of the exact same Spout/Bolt that produce/process different data partitions. Moreover, Storm supports several grouping strategies that specify how data will be partitioned and exchanged among the tasks of Bolts. Custom groupings are possible, while built-in ones are also available.

3.1.2 Spark Streaming

The Spark[3] framework offers a very flexible execution environment for scalable computation. Initially designed as an improvement of the MapReduce paradigm for distributed computation (e.g., in a Hadoop cluster), Spark capsules the data in a Resilient Distributed Dataset (RDD), on which a program can be executed. This allows to apply iterative algorithms efficiently on a cluster. Spark Streaming extends the functionality of Spark to work on mini batches of incoming data. While this offers a lot of possible applications for streaming algorithms, the distribution and management of the distributed batches increases the latency of this framework, compared to other dedicated streaming solutions.

[2] https://storm.apache.org/
[3] https://spark.apache.org/

The Spark Streaming API[4] works on one or more discretized streams, each termed a DStream. DStreams can be created either from input data streams stemming from sources such as Kafka, or by transforming other DStreams. Data streams arrive at a Receiver process. To create a DStream, discretization takes place based on two types of time intervals. The "block interval" organizes incoming data streams into blocks of few tens of milliseconds, with each block being a data partition. Concatenating blocks creates micro-batches. These are then forwarded to the core of Spark. There, a micro-batch is treated as an immutable collection of data tuples organized into partitions (blocks), called an RDD as mentioned above. As time passes, new RDDs are instantiated and may undergo several transformations (map, reduceByKey, join, etc), window, or output operations. These operations may either be 'narrow', like map, which operate on a single partition and essentially pipeline the data of that partition to a resulting single partition, or 'wide' operations like reduceByKey which require to map the data across the partitions in new RDDs. A series of such transformations or window operations form a DAG where nodes are RDDs at various timestamps and arrows correspond to the desired operations on them. Such a DAG expresses the conceptual view describing the flow of data processing.

A Spark Cluster includes a Driver process at a Master (Driver) node and one or more Worker nodes. The Driver node is where the Spark application (i.e., the SparkContext) is created. A Worker node includes one or more Executors (JVMs) running tasks assigned by the Driver.

Each RDD undergoes a number of processing stages, translating the DAG of the conceptual view. A stage is formed as a set of narrow transformations that can be pipelined and executed by a single Worker independently. Having divided the execution graph into stages, within each stage a data partition is assigned to a single task. Then tasks are assigned by the Driver to Workers.

Parallelism can be tuned in several ways. For instance, one can pre-partition (i.e., before reaching a Receiver process) Kafka messages and create a DStream for each partition. Within Spark, the repartition transformation can create more or fewer partitions of a DStream. Redistributing streams using wide operations changes the partitioning of the streams as well. For example, the keyBy transformation repartitions data by hashing tuples based on key field(s).

3.1.3 Flink

In contrast to Spark, Flink[5] does not rely on processing micro-batches but has a continuous processing engine. This means the framework can handle each streaming tuple individually. Furthermore, Flink offers a rich API for manipulating streams and has libraries for machine learning and complex event processing (CEP) (FlinkCEP).

With this API, it is very easy to create new analytical processes, although it requires some experience on writing code. The deployment is also relatively simple but encompasses some steps that are not trivial for a non-code inclined user (compiling the process as a *.jar file and transferring it to a Flink cluster). But simplifying these steps is one of the goals of the INFORE project. The flexibility and scalability of Flink are strong arguments to use it as reference model for the first versions of some components of the project architecture. The basic building blocks of a Flink program are Data Sources (streams), operators, and data sinks bound together in a directed graph, which is not necessarily acyclic.

From a developer's viewpoint, Flink operators or data sinks may resemble Spark transformations, window operations, or output operations, respectively. However, the implementations of these operators differ significantly. Flink is a true streaming engine treating batch processing as special case of streaming with bounded data and not vice versa as it is the case with Spark. In Flink, each data source or operator (*map*, *keyBy*, *filter*, etc.) is implemented as a long running operator similar to Spouts and Bolts in Storm. Flink also gives low-level control on the exact stream partitioning after a transformation, like Storm groupings.

A Flink cluster is composed of (at least one) Master and several Worker nodes. The Master node runs a JobManager for distributed execution and coordination purposes, while each Worker node incorporates a TaskManager which undertakes the physical execution of tasks. Each Worker which is a Java Virtual Machine (JVM) process, has multiple task slots (at least one). Each operator or instance of an operator of the Flink program is assigned to a slot and tasks of the same slot have access to isolated memory shared only among tasks of the same slot. The new concept here involves task chaining. That is, Flink allows to place two operators (or instances of operators) together into one task

[4] https://spark.apache.org/streaming/
[5] https://flink.apache.org/

i.e., thread, for performance reasons. At the level of an operator/data source/data sink, parallelism is configured by calling a setParallelism method. Redistributing streams using wide operations (as in Spark) changes the partitioning of the streams as well. For instance, keyBy repartitions by hashing key field(s), broadcast replicates the operator outcome, and rebalance performs round robin repartitioning.

3.1.4 Akka

Akka[6] is a toolkit to build reactive systems for streaming. It features a high-performance set-up with a low memory footprint and a decentralized and clustered architecture. The guiding design principle is the Reactive Manifesto[7] published in 2014. The Streaming API of Akka offers a diverse collection of operators for manipulating data, which covers the typical stream analytics requirements, whereas a dedicated library for machine learning is missing.

Conceptually, Akka employs an actor-based model. The actor model is a model of computation with the following properties. Each actor is an independent process with its own encapsulated state and behaviour that communicates with other actors via messages. Each actor has an address, an incoming message box, a state, and a behaviour. Messages are sent to an address and are kept in the inbox of the designated actor until it is free to process the message. Message actions include:

- Send a finite number of messages to other actors.
- Create new actors.
- Change an actor's own behaviour.

An actor can communicate with any other actor it knows of. Actors can form a hierarchy, termed as an Actor System, where each actor is related to others with a parent, child, or sibling relation. A parent actor is the one that has created child-actors.

Typically, an Akka application is composed of one or more Actor Systems. Parallelism can be implemented either by having a parent actor assigning different pieces of work (operators) in a workflow to different child-actors or by having a parent actor creating multiple instances of the same child-actor, assigning different data chunks to be processed by each. The communication among actors can be established by a router actor which forwards messages in a way similar to Storm's grouping strategies, while custom routing schemes can be implemented as well.

Closer to the physical view, an actor system is executed in a single JVM. The idea is to have a number of threads roughly equivalent to the available CPU cores and typically the number of actors is greater than the available cores (so that a thread is unlikely to remain idle and pseudo-parallelism can internally be exploited). A Master – worker relationship is established among actors in an actor system. Each parent is responsible for the supervision of its child actors. In case the child dies, it is the parent's responsibility to react upon this event. The supervisor is free to choose from the following four strategies:

- Resume the subordinate, keeping its accumulated internal state.
- Restart the subordinate, clearing out its accumulated internal state.
- Stop the subordinate permanently.
- Escalate the failure, thereby failing itself.

In addition, to the parental supervision any actor may subscribe to be notified of the death of another actor.

3.1.5 Kafka Streams

The typical usage for Kafka is as a message broker between different data producers and consumers. It is designed to be high-throughput system, that easily scales to high loads. With Kafka Streams, there is a dedicated stream processing library available, that allows to integrate low level operator integration (e.g., including arbitrary Java or Python code). Kafka is a very popular choice for orchestrating message queues and inter process communication between different

[6] https://akka.io/
[7] https://www.reactivemanifesto.org

systems. One of the appealing things here is the simple architecture and the quick way to set-up a running system. Kafka is maintained as a top-level project of the Apache foundation, so it is also available as open source and has many and very active contributors.

Because of the wide-spread usage and the easy-to-use approach, Kafka is one of the main platforms considered for the INFORE project. The message broker service provided by Kafka will be an essential part of the underlying distribution system and a good candidate as an endpoint of the analytics platform.

Besides the listed frameworks, there are many other tools and programming languages that offer streaming capabilities of some sorts. But to our best knowledge, these are often very specialized for specific tasks, not very widespread or offered by commercial vendors (thus, conflicting the open source scope of generic INFORE components).

3.2 Machine Learning (ML) on Streaming Data

The main goal of the INFORE project is not only to have a scalable architecture to handle large streaming data, but also to make it pluggable and capable of providing an easy to use, graphically-assisted analytics solution. There are numerous analytics or machine learning libraries and tools available. Some of these solutions also provide support for or are specialized in handling streaming data. However, their implementations are restricted only on a specific Big Data platform.

One of the challenges of the INFORE project is that we aim at supporting many streaming machine learning platforms and libraries to leverage their individual strengths and to incorporate any existing or future facilities in the scope of an extensible, pluggable architecture. Therefore, in this section we discuss what is available in popular machine learning algorithmic suites, all of which are available as open-source. It is important to emphasize that the contributions of the toolkits discussed below are orthogonal to INFORE, since as stated above, they are built on a single platform each and thus INFORE can accommodate their libraries (wherever they include implementations of streaming algorithms) and (i) perform cross-platform optimization in workflows containing machine learning operators, with implementations available in various platforms, (ii) account for available HPC infrastructure, and (iii) rapidly explore and fine tune machine learning models under different parameterizations, exploiting the power of synopses provided by the respective INFORE architectural component.

This section will give a brief overview of existing machine learning stacks for streaming data and how they may fit into the INFORE requirements.

3.2.1 FlinkML

FlinkML, written in Scala, provides a set of scalable machine learning algorithms over Flink's distributed framework. FlinkML's library is built using the DataSet API of Flink, thus providing offline algorithms designed for batch, instead of stream processing. FlinkML supports the creation of machine learning pipelines i.e., the ability to chain together different transformers and predictors. Moreover, it includes a parameter server implementation[8] which is a paradigm for accomplishing distributed (parallel) machine learning. Additionally, FlinkML is designed for model-parallel machine learning but does not include provisions for fault tolerance and allows only asynchronous training. FlinkML is not included among the libraries of the latest version 1.9 of Flink, but it has been extended towards online, scalable machine learning by the Proteus-SOLMA library discussed below. Finally, it provides a connector to the SAMOA library, also detailed shortly.

3.2.2 Proteus-SOLMA

In a nutshell, SOLMA is intended to be a streaming version of FlinkML sitting on top of the DataStream, instead of the DataSet, Flink API. It provides scalable online machine learning algorithms and real-time interactive visual analytics for extremely large data sets and data streams. It is written in Scala and has been integrated into an enhanced version of Apache Flink developed within the scope of the Proteus H2020 project discussed below. SOLMA includes a limited set of online algorithms for classification (such as Passive Aggressive Classifier), regression (such as Competitive Online Iterated Ridge Regression) and few from other machine learning categories such as online anomaly detection using incremental Principal Component Analysis (PCA).

[8] https://github.com/FlinkML/flink-parameter-server

3.2.3 Spark MLLib

The MLLib library is an essential part of the Spark ecosystem and integrates well with Spark Streaming. Despite the fact that, with respect to offline machine learning algorithms, MLLib is, perhaps, the richest such library of its kind, there is a limited set of specialized streaming versions of common algorithms included in it. Online machine learning algorithms of MLlib are restricted to Linear Regression, Logistic Regression and K-Means. It is also possible to use a section of the streaming data as historic training data for a generic machine learning model from MLLib and then apply it on the incoming stream of new data.

3.2.4 Apache SAMOA

Apache SAMOA is a streaming machine learning library that aims to hide the complexity of the underlying streaming system. It allows the development and usage of distributed streaming machine learning algorithms independent of the underlying engine and as such can be deployed on many different platforms. For the INFORE project this is somewhat appealing concept that covers many of the requirements. Unfortunately, the project is still in a relatively early stage (Apache incubator) and the number of supported streaming platforms is limited.

3.3 Related Projects

FERARI[9]

The goal of the FERARI project was to build a framework that allows for efficient and timely processing of Big streaming Data. This framework includes a distributed complex event processing (CEP) engine, a query optimizer and a distributed online learning framework. The framework is materialized on Apache Storm using IBM Proton on Storm as its underlying CEP engine. The focus is on optimizing the execution of CEP operators over a geo-distributed network of sites. That is, each site (e.g. data center) in the network is supposed to run an Apache Storm topology and, given a workflow which is a graph of CEP operators, decides on which site each operator should get evaluated, i.e., it maps the CEP operator graph to the network graph, so that communication is minimized under network latency constraints.

This is only a special case of the optimization parameters in INFORE. Apart from choosing the site over the network where each query operator should get evaluated, the INFORE Optimizer (a) performs cross-platform optimization. That is, it chooses the Big Data platform over which each operator of the workflow should be executed, in case multiple implementations of an operator exist in different such platforms, (b) it performs resource provisioning for each operator over HPC infrastructures including heterogeneous clusters composed of CPUs and GPUs (c) it performs synopses-based optimization by replacing parts of the workflow with equivalent ones composed of approximate, instead of exact, operators to speed up the processing and reduce memory utilization under accuracy constraints, (d) it may devise the parallelization degree.

RHEEM[10]

RHEEM is a system designed to support cross-platform data processing. That is, it enables users to run data analytics over multiple data processing platforms. For this, it provides an abstraction on top of existing platforms in order to run data analytic tasks on top of any set of platforms. This approach aims at freeing data engineers and software developers from the burden of getting familiar with different data processing systems, their APIs, strengths and weakness; the intricacies of coordinating and integrating different processing platforms; and the inflexibility when tying to a fix set of processing platforms. Rheem has built-in support for the following processing platforms: - Java 8 Streams - Apache Spark - GraphChi - Postgres - SQLite.

RHEEM, similarly to INFORE, targets cross-platform optimization in the execution of workflows and provides a graphical user interface for designing such workflows. However, (a) its focus on streaming data is limited to JavaStreams, (b) it does not account for settings composed of dispersed sites, (c) it does not examine HPC infrastructures and (d) it lacks support for synopses-based optimization.

[9] http://www.ferari-project.eu/
[10] http://da.qcri.org/rheem/

VINEYARD[11]

VINEYARD built an integrated platform for energy-efficient data centers based on novel programmable hardware accelerators (Dataflow engines and FPGA-based servers). It developed a high-level programming framework and Big Data infrastructure for allowing end-users to seamlessly utilize these accelerators in heterogeneous computing systems by employing typical data center programming frameworks (in particular, Spark). VINEYARD accounts for HPC infrastructures and heterogeneous data centers (clusters) as INFORE does. Nonetheless, it leaves to the developer the responsibility of choosing whether Spark MLlib or the MLlib_accel API, provided by VINEYARD, will be used for the execution of a particular machine learning operator. Moreover, VINEYARD is focused on batch, instead of stream processing, it does not account for cross-(Big Data)platform optimizations and it does not provide a graphical user interface to assist workflow design.

PROTEUS[12]

PROTEUS aimed at evolving massive online machine learning strategies for predictive analytics and real-time interactive visualization methods – in terms of scalability, usability and effectiveness dealing with extremely large data sets and data streams – into ready to use solutions, and to integrate them into enhanced version of Apache Flink. The focus of the project was to extend Flink with support for hybrid, batch and stream, processing capabilities and to develop online machine learning algorithms to be executed over the distributed/parallel processing architecture of Flink. Hence, Proteus's focus was neither on cross-platform optimization nor on graphically assisted workflow design tools. Moreover, Proteus developed a synopses library, however, INFORE's synopses data engine is provided as a constantly running service, which (a) simultaneously maintains multiple synopses of different types and (b) synopses can be loaded on-the-fly from internal or external libraries. On the contrary, Proteus needs a separate job for each data summarization algorithm. The impact of the latter approach is that for monitoring thousands of streams simultaneously, it needs to submit and execute thousands of Flink jobs in parallel.

CELAR[13]

The CELAR platform incorporates intelligent decision-making algorithms to support multi-dimensional and multi-grained elasticity control over the cloud and its services. CELAR evaluates elasticity at multiple levels of the cloud stack (Paas, IaaS) and considers the impact of enforced actions in relation to cost, quality and allocated resources for elastic cloud services. To provide this functionality in a vendor-neutral manner, real-time monitoring and elasticity behaviour analysis of heterogeneous types of information collected from different and multiple data sources, is required. This important role, in the CELAR software stack, is the job of the Cloud Information and Performance Monitor Layer. The Cloud Information and Performance Monitor Layer runs alongside all the layers of the cloud infrastructure, in order to provide the intelligent decision-making mechanisms of the CELAR System with real-time, cost-enriched, monitoring metrics. The CELAR Elasticity Provisioning Platform is the central component of the entire platform. Its main goal is to provide the methods and tools to integrate and orchestrate all the submodules of the CELAR platform (decision module, monitoring system, application description tool, etc) into one functional elasticity middleware that can expose its functionality to the external applications with a unified and user-agnostic manner.

Elastic resource allocation is only one aspect of the work carried out in INFORE to maintain horizontal scalability in the long run. In addition, INFORE offers federated scalability, by scaling out the computation to multi-cloud platforms, vertical scalability by offering synopses as stated above. Furthermore, INFORE optimizes execution of cross-(Big Data) platform workflow and examines heterogeneous cloud environments incorporating hardware accelerators.

[11] http://www.vineyard-h2020.eu/
[12] https://github.com/proteus-h2020/
[13] http://www.celarcloudproject.eu/

Table 1 summarizes the features of INFORE in comparison with related projects.

Project Name	*Stream Processing*	*Cross-platform*	*HPC*	*Adaptive*	*Synopses-based Optimization*	*GUI- assisted workflow design*	*Geo-distributed Sites*
FERARI	✓	✗	✗	✓	✗	✗	✓
RHEEM	✗	✓	✗	✓	✗	✓	✗
VINEYARD	✗	✗	✓	✗	✗	✗	✗
PROTEUS	✓	✗	✗	✗	✗	✗	✗
CELAR	✗	✗	✗	✓	✗	✗	✗
INFORE	✓	✓	✓	✓	✓	✓	✓

Table 1: Comparison of INFORE with related projects.

4 INFORE Architecture

This section describes the structure and concepts defined for the architecture and how the different components interact with each other.

4.1 User Requirement Analysis

While the initial concept and requirements of the architecture were already envisioned during the project proposal phase, some refinements are always required during the starting phase of the project and while the system prototype takes shape. To ensure that all necessary requirements of the use case partners are met, an initial phase of requirement engineering was initiated in parallel with the requirements expressed in Deliverables D1.1, D2.1 and D3.1.

Here, the concept of a target persona and user stories were applied. These concepts are typically used by software development projects and designers. A persona represents the archetypical users of a tool and helps us as developers to better identify them. It consists of a sketch about the daily work of that person, her educational background, often used tools and some common personality traits to flesh out the characterization. The user story is another element that helps to refine the needs of the user. It is a simple sentence, that states the role of the user, her desired action and the goal that she wants to achieve. Each user story represents one task that is crucial for the work routine of a persona.

We asked the three use case partners from WP1, WP2 and WP3 to create sample personas of the latter users of the developed tools and the typical tasks they want to perform.

From all partners, we have collected in total seven different target personas that represent typical users of the final system. Additionally, six user stories where formulated that help to define the tasks these users would later like to perform. While these requirements focus more on the end users and their experience with the final tool, they already helped to formulate requirements, especially on the user interface and the graphical editor. It became clear that the requirements are not perfectly aligned for all partners. For some user types the multi-platform approach is crucial, as they need a system that eases the integration of different data streams and analysis platforms. For others (e.g., in the financial use case), the high level of interactivity and easy representation of data streams is more important. These findings again stressed out the importance of a flexible architecture that can handle the heterogenous requirements for streaming applications.

4.2 Architecture Overview

The INFORE Architecture is designed to tackle the project objectives described in Section 3.1. Conceptual pillars are used to create a component-based modular design. Figure 1 gives an overview of the conceptual design of INFORE Architecture. The integration concept is described in the next subsection, while the different components are explained in detail in the following subsections.

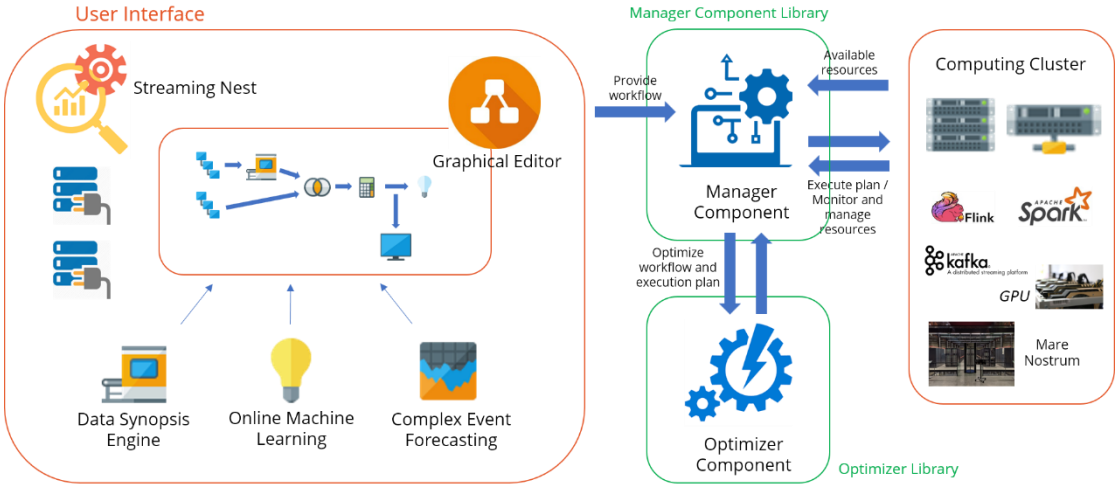

Figure 1: Overview of the conceptual design of the INFORE Architecture.

4.3 Integration

The INFORE Architecture will achieve design time and execution time reduction goals through a holistic integration approach based on *Conceptual Pillars*. These pillars are concretely realized as components which are the following:

1. Connection Component
2. Graphical Editor Component
3. Manager Component
4. Optimizer Component
5. Synopsis Data Engine Component
6. Complex Event Forecasting Component
7. Interactive Online Machine Learning Component

The central component is the Manager Component providing the interaction point between most of the other components. The Manager Component handles the actual execution and monitoring of the designed streaming analysis process.

All components will be developed in a modular way, with clear interaction-interfaces between them. This enables an easy exchange between the modules and provides the necessary pluggability of the INFORE Architecture.

4.3.1 Connection Component

The Connection Component enables access and preliminary management of the selected streaming backends (stream sources and sinks, Big Data platforms and respective computer clusters) that are utilized by the INFORE use cases and broader application scenarios. The Graphical Editor Component will be a streaming extension of the RapidMiner Studio (which is currently focused on offline, batch processing) to provide specific functions (as operators) that are developed in the selected streaming technologies.

Some aspects of streaming technologies being considered at this stage are explained in Section 4.2. Here, we briefly mention the mechanics of the Connection Component. The Connection Component functionality is realized as Connection objects in the Graphical Editor Component. The user of the INFORE Architecture only needs to provide the essential information for connecting to the backends without any coding needed. Then, the Connection objects can be utilized in the streaming analysis process, which itself is designed using the Graphical Editor Component.

For the scope of the INFORE project, we identified three functional concepts that include input streams, output streams and stream processing backends, where a streaming analysis process may be executed. A typical streaming analysis process will consume one or more input streams, perform transformations and may result in output stream(s). Input streams provide the input data, for which the streaming analysis process is designed. The streaming analysis process is application specific. Output data streams (which are not a necessity) mark the endpoint of a streaming analysis process. The nature of the application scenario can impose restrictions on their type, but the approach provides the required flexibility. Output streams can serve as Input streams for downstream analysis processes. Streaming processing backends are the computer clusters typically hosting some Big Data platform(s), which provide the physical infrastructure resources needed to execute the streaming analysis process. These are limited by the hardware and technology stacks the user has access to.

Keeping the above description as a pretext, the connectivity requirements become very important. The Connection Objects are thus configured for each supported compute backend e.g., to connect with Flink or Kafka backends. The Connection Object(s) are then used together with the Manager Component to establish connection for input stream, output stream, or to place a streaming analysis process on a compute backend and/or to configure environmental settings. The Connection Component can be extended with additional Connection objects to reflect the execution requirements of the streaming analysis process. For instance, if a streaming backend technology provides a set of desired connectors, which are installed in the Architecture implementation, then, these may be reused by the streaming analysis processes. As an example, a Flink-based streaming analysis process may output its results into a Kafka topic. In such a case, INFORE's Flink Connection object can encapsulate the Flink connectors for required sources (for input streams) and sinks (for output streams) to provide access in real time. A user can graphically define and internally create multiple Connection objects based on the available backends.

The concepts of input streams, output streams and stream processing backends capture the general constituents of a typical streaming analysis process. With some exceptions, all streaming technologies described in Section 4.2. can provide the actual functionality for these constituents, which are addressed in Section 4. For flexibility, the INFORE Architecture aims at combining different streaming technologies in one streaming analysis process. If more than one stream processing backends are provided to the INFORE Architecture, the architecture by virtue of the Optimizer Component can automatically decide on the optimal placement of different parts of the analysis process to different backends (Big Data platforms and/or cluster/HPC infrastructures).

4.3.2 Graphical Editor Component

The Graphical Editor Component enables users of the INFORE Architecture to easily design a streaming analysis workflow without any coding needed. This is achieved by encapsulating streaming analysis functionality into so called operators. The operators can be placed inside the graphical user interface via drag and drop actions and can be connected by drawing arrows to define the data flow in the streaming analysis workflow.

To ensure flexibility, the streaming analysis operators of the Graphical Editor Component, called “Logical Operators”, are designed as an abstraction layer, without defining the actual streaming technology used. Thus, the user can focus on the analytic setup of the process, without handling the technology specific aspects.

The Manager Component can automatically perform the desired functionality by utilizing the physical implementation(s) of the Logical Operators for the specific streaming processing backend used. If physical implementations for a particular Logical Operator in more than one streaming processing backends, the Optimizer Component can devise the selection of the physical implementation to optimize the processing of the workflow. This is of course only possible for functionality (operators) for which multiple implementations over different Big Data platforms exist. See Section 3.1 for an overview of the common streaming technologies.

The functionality of the Data Synopsis Engine Component, the Complex Event Forecast Component and the Interactive Online Machine Learning Component are represented as Logical Operators as well and can be placed (dragged and dropped) in the streaming analysis workflow and get graphically parameterized. The physical implementation of these operators on top of specific Big Data platforms (run on respective clusters) are developed within the scope of WP6.

These “INFORE Component” operators contain all information to execute the specific algorithms and methods, which are provided by external (external from the Graphical Editor Component’s perspective) libraries. They provide loose coupling between the Graphical Editor Component and the specific INFORE Component. The libraries can be developed independent of the Graphical Editor Component. An adaption is only needed if the interface changes. In addition, further components can be added in the future by just adding operators to the Graphical Editor Components. These aspects ensure the extensibility and pluggability of the INFORE Architecture. The workflow is to be fed to the Manager Component, which also handles the deployment and execution aspects on the streaming cluster.

Figure 2 shows a demonstration of a streaming analysis workflow, designed with the Graphical Editor Component.

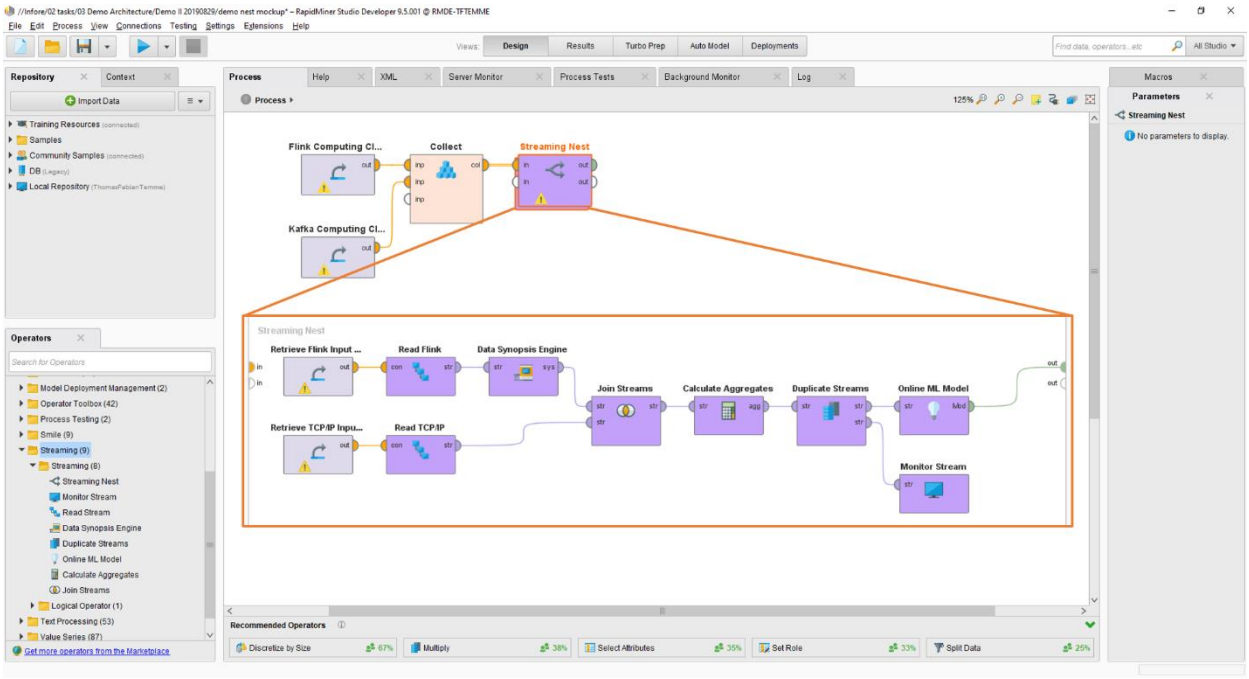

Figure 2: Demonstration of a streaming analysis workflow, designed with the Graphical Editor Component.

4.3.3 Manager Component

The Manager Component plays a central role in the INFORE architecture. The Manager Component is realized as a library, which allows extensibility and flexibility to integrate with potentially multiple systems or front ends. This arguably sustains the work done in INFORE beyond the scope of the project. On one hand, the Manager Component wraps the *data retrieval*, *preparation*, *transformation* and *modelling* methods that are interfaced with the Graphical Editor Component in a loosely coupled fashion. On the other hand, it utilizes the Connection Component to handle connectivity with the streaming backend(s), provides a tool-agnostic representation of the streaming analysis workflow (as required by the Optimizer Component to optimize the workflow), and handles the deployment aspects. The Manager Component is also intended to retrieve real time information from stream processing backends on the available capacity, usage of resources, workload status, and stream processing meta data.

There are three functional modalities which lie in the domain of the Manager Component. These are briefly explained below.

Workflow and Plan related functions:

The Manager Component receives the workflow from the Graphical Editor Component. Besides the actual streaming analysis process, the workflow also contains the connection objects for the input and output streams and the computing backends, on which the workflow shall be deployed and executed. The Manager Component converts the workflow, the resource and execution information into a JSON representation. This representation acts as a tool-agnostic workflow representation, which the Optimizer Component can optimize in a domain and technology independent fashion.

After performing the optimization of the workflow, the Optimizer Component passes it back to the Manager Component. The next steps involve the deployment of the optimized workflow or parts of this workflow on the desired backend(s) (Big Data platforms and available clusters). The basic idea is that the Manager Component can deploy an optimized workflow, or even an unoptimized (default) workflow. The latter may be the case if optimization is not possible, for instance, due to lack of operator physical implementations in more than one Big Data platforms.

Deployment related functions:

The Manager Component also implements the functionality to create a deployable artefact from the optimized workflow and dispatch it. Recall that the workflow or parts of the workflow may need to be deployed at different backends. The deployable artefacts are simply referred here as a plan to differentiate the streaming analysis workflow from its deployable unit(s) e.g. a Flink job or a Kafka program. The deployment functions provide the fine link, where the generic design of INFORE streaming analysis processes meet with the underlying execution technology.

Infrastructure related functions:

The Manager Component also provides basic monitoring and control functions. These help to understand and manage the infrastructure context. Monitoring is needed on two levels. First, the execution status of the workflow may need to be queried at some intervals, and secondly, the usage of resources and current execution workload may need to be fetched by the Optimizer to assess whether the desired KPIs are being met.

This process-level and system-level feedback assists the Optimizer to update its variables related to the execution time decisions it makes. This may result in an adaption of the deployment plan. The control functions wrap the basic functionality to scale the resources up or down. The idea is to wrap the API or commands provided by the streaming technology stacks, to easily fetch or execute basic monitoring or control functions but not over-emphasizing on standardization since many of these features are frequently updated and work with specific versions of third-party utilities. Hence, these dependencies need to be installed on the compute backends as well.

The Manager Component is also able to update the visual representation of the workflow in the Graphical Editor Component according to the optimized workflow, retrieved by the Optimizer Component.

Figure 3 illustrates the functionality of the Manager Component.

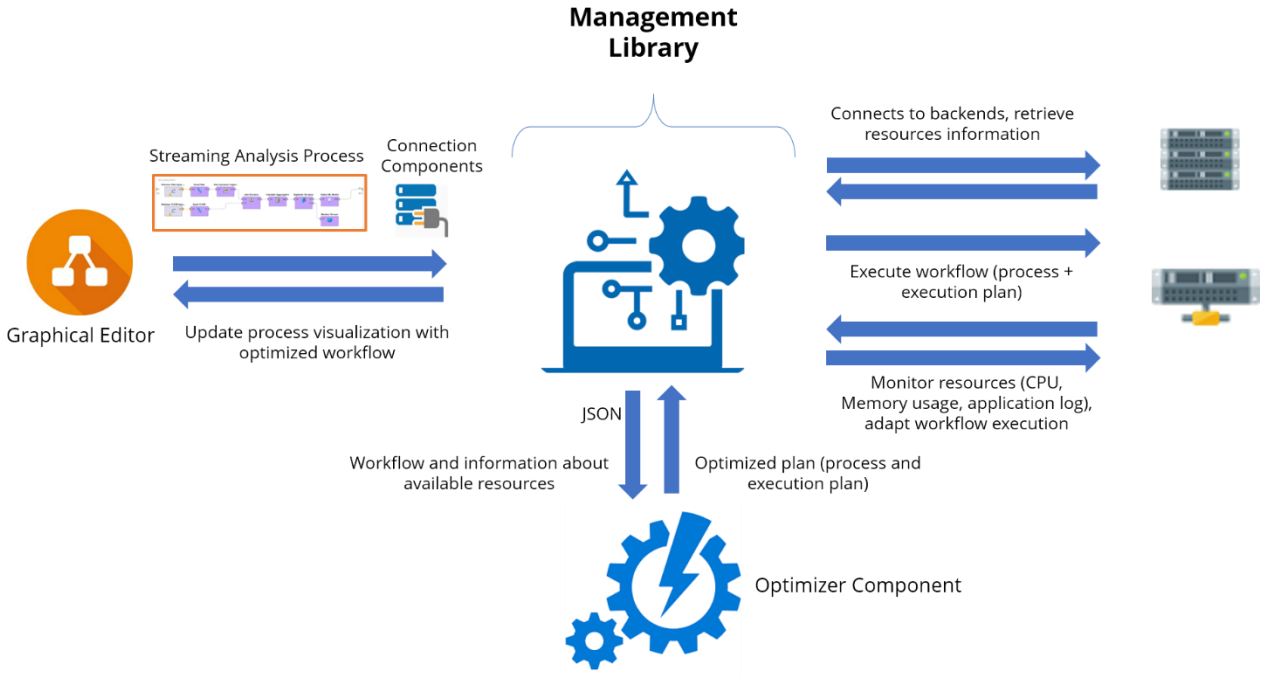

Figure 3: Overview of the functionality which is provided by the Manager Component. The diagram clearly shows the central role which the Manager Component takes in the INFORE Architecture.

4.3.4 Optimizer Component

The INFORE architecture aims at processing workflows involving large-scale streaming data. These are complex workflows, typically spanning multiple execution and storage engines. Identifying an efficient workflow execution could involve decisions made on more than one execution engine. For example, consider a stream involving two engines, Kafka and Spark. It is possible to optimize the stream on each engine using best practices for Kafka and Spark, respectively. Furthermore, engines employing an intrinsic optimizer (e.g., Spark Catalyst) can provide an efficient execution plan for the part of the workflow running on the said engine. However, no engine has a complete picture of the workflow.

INFORE Optimizer helps with providing a holistic approach covering the entire workflow. Note, that INFORE Optimizer does not aim at replacing the engine specific optimizers. Rather, it works complementary by identifying optimization opportunities outside an engine and enabling further intra-engine optimizations by actions like function shipping (i.e., move a computation closer to the data) and data shipping (i.e., move the data closer to the computation). A basic optimization for the example Kafka-Spark workflow would be to push a filter from Spark down to Kafka to reduce the amount of data shipped over to Spark.

Workflow optimization in INFORE is based on a multitude of optimization objectives and configuration or system parameters such as resource availability, resource efficiency, workload parameters at runtime, efficiency of streaming technologies, availability of operator implementation on multiple engine, availability of execution engines, and so on. Typical goals considered by the optimizer include increase resource utilization, reduce latency, improve quality, satisfy business and technical constraints. The multi-platform approach of the optimizer allows to efficiently handle and select the best suited resource available. For example, a join operation on multiple streams is a common implementation on most frameworks and so the optimizer can decide which platform to use, for example based on network locality or on available compute resources.

The Optimizer receives a workflow from the Manager Component. Then it optimizes the workflow (if applicable) and then sends the optimized workflow back to the Manager which handles workflow execution. This operation can be done either offline or online, during the running execution of a workflow, which allows the INFORE Architecture to adaptively react to changing condition of data streams and available resources.

The workflow metadata describe the designed streaming analysis process, the available resource information, the involved engines from the connected streaming backends, and user design choices. This information is encoded in a JSON format. An informed design choice we made is to decouple the workflow encoding of the Manager Component from that of the Optimizer. This is based on two reasons. First, in doing so, both the Manager and the Optimizer Components can be replaced by other tools if needed. This allows for increased pluggability and easy enhancement of the INFORE Architecture, and of potential future reuse of the code in follow-up applications.

The second reason was to enable an engine-agnostic workflow representation inside the Optimizer. Hence, a workflow designed for specific engines (e.g., Spark) is engine specific. For example, consider a workflow getting data directly from Kafka and containing a filter operator and a join operator implemented in SparkSQL. When the workflow is propagated into the Optimizer, it is converted to an engine-agnostic form that contains a logical filter operator and a logical join operator. This enables the Optimizer to look for optimization opportunities in other available engines (e.g., Kafka). A possible scenario for the example Kafka-Spark workflow could be as follows: (a) first, the workflow is transformed to an engine agnostic form and thus, the Spark filter and join are converted to an engine-agnostic filter and join, respectively; (b) then, the workflow is processed by the Optimizer that may identify an opportunity to push the filter back to Kafka; (c) next, the Optimizer converts the workflow to an engine specific workflow having two parts, one part with a filter to be applied to Kafka and one part with a join to be executed in Spark.

Besides workflow parsing, the Optimizer includes a component for statistics collection that keeps a history of statistic observations over past workflow execution, at the workflow level, at the operator level, and at the engine level. Once a workflow is sent to the Optimizer, the Optimizer enumerates the space of possible and promising execution plans for the workflow and estimates plan costs using a dynamic cost model that predicts workflow execution runtime based

on the historical statistics collected. The navigation of the execution plan space can be done exhaustively or in a greedy fashion using heuristics for improved performance.

The optimization methods are described in more detail in the Deliverables D5.1, D5.2 and D5.3.

4.3.5 Synopsis Data Engine Component

The Synopsis Data Engine (SDE) Component provides implementation of approximate query processing algorithms that can generate representative summaries on top of streamed data. The component includes a library implementing the query processing algorithms and operators, containing the configurations and parameters necessary to perform such a query to the SDE.

The usefulness of the SDE within INFORE streaming workflows is as follows. First, it can provide various kinds of scalability, including:

- Horizontal Scalability: although Big Data platforms such as Storm, Flink or Spark are destined by design to scale out the computation by parallelizing the processing load to a number of available processing units, the SDE can further boost horizontal scalability by working on carefully-crafted summaries of data and finally provide estimations of an operator's result, with accuracy guarantees. In that the processing load is shed due to the use of synopses and the computational complexity of the problem at hand is reduced.
- Vertical Scalability: this type of scalability concerns scaling the computation with the number of processed streams. Synopses provided by the SDE along with locality aware hashing techniques can reduce the computational load when, for instance, operations requiring pairwise comparisons among streams are engaged in each workflow.
- Federated Scalability: in settings composed of several geographically dispersed computing clusters or clouds, utilizing synopses in each of these and communicating synopses instead of full local streams reduces the communication cost of global (over the union of streams arriving at the various clusters) operator evaluation.

Second, the SDE and the library of data summarization techniques it includes can serve as tools provided to the INFORE Optimizer Component so that the latter can perform synopsis-based optimization on INFORE workflows, i.e., if the application has declared that it can tolerate approximate results to a submitted workflow, the Optimizer can replace exact operators engaged in the workflow, with equivalent, approximate ones so as to speed up the processing under certain accuracy constraints.

Figure 4 below illustrates the internal architecture of the SDE Component. The details of the architecture and the SDE library are included in Deliverable D6.1 submitted on M12 of the project as well. We will here elaborate on the SDE API, used by other INFORE Components as well as upstream (providing input) or downstream (receiving input) operators of a workflow engaging data synopses. The proof-of-concept implementation of the SDE Component is developed in Flink and Kafka, while the SDE library is implemented in Java. The SDE is to be provided as a continuously running service to different, currently executed/submitted workflows. Therefore, it can simultaneously maintain multiple synopses used in a variety of workflows.

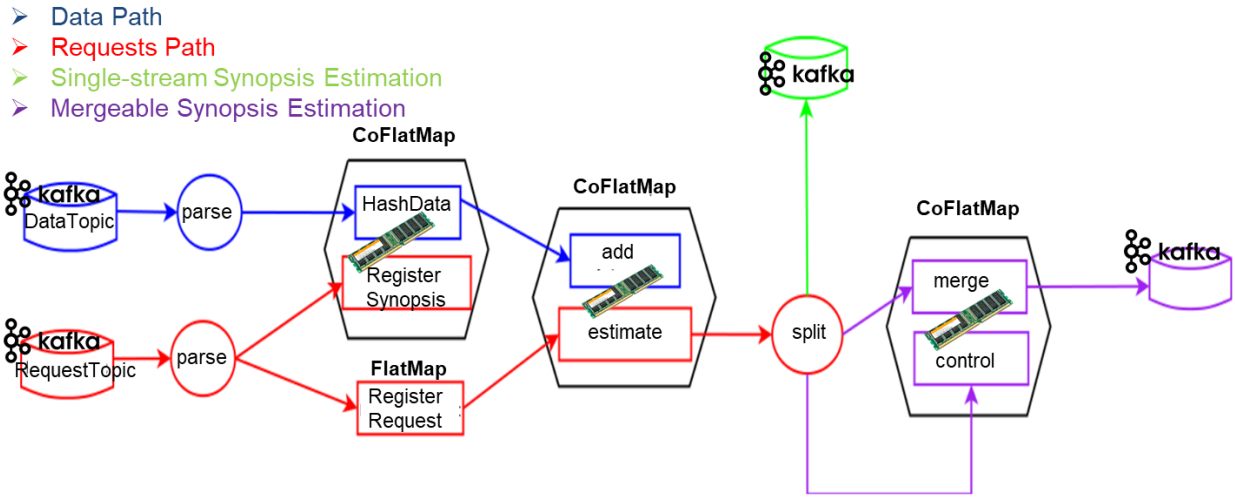

Figure 4: Illustration of the internal architecture of the SDE Component.

As shown in Figure 4, the whole inbound and outbound communication with the SDE is achieved via Kafka. All data tuples updating one or more currently maintained synopses arrive at a single DataTopic in Kafka. Moreover, all requests towards the SDE, arrive to it in a single RequestTopic in Kafka. In both cases, data tuples arriving at each topic are parsed internally by the SDE to extract information about the synopses they involve. The producers of the DataTopic and the RequestTopic are upstream operators of currently running workflows, while the consumers are the respective parsers. Finally, the results of queried synopses are streamed through one or more OutputTopics in Kafka. The producers of the output topic(s) are internal Flink operators delivering approximate query answers at the final stage of the SDE processing pipeline, while the consumers are downstream operators of running workflows.

The SDE API which is essentially implemented by the parsers consuming respective Kafka messages in Figure 4, provides the following facilities:

- Build/Stop Synopsis Request: a synopsis can be created or ceased on-the-fly, as the SDE is up and running. In that, the execution of other running workflows that utilize synopsis operators, is not hindered. A synopsis may be (a) a single-stream synopsis, i.e., a synopsis (e.g. sample) maintained on the trades of a single stock, or (b) a data source synopsis, i.e., a synopsis maintained on all trades irrespectively of the stock. Moreover, the API allows submitting a single request for maintaining a synopsis of the same kind, for each out of multiple streams coming from a certain source.
- Load Synopsis Request: the SDE library incorporates a number of synopsis operators, commonly used in practical scenarios. The Load Synopsis facility supports pluggability of the code of additional synopses at runtime, their dynamic loading and maintenance at runtime.
- Ad-hoc Query Request: the SDE accepts one-shot, ad-hoc queries on a certain synopsis and provides respective estimations (approximate answers) to downstream operators or application interfaces, based on its current status.
- Continuous Queries: continuous queries can be defined together with the request for building a synopsis and they provide an estimation of the approximated quantities, such as counts, frequency moments or correlations, when the synopsis is updated due to reception of a new tuple.
- SDE Status Report: the API allows querying the SDE about its status, returning information about the currently maintained synopses and their parameters. The purpose of this facility is two-fold. First, it is useful during the definition of new workflows, since it allows the application to discover whether it can utilize already maintained data synopsis. Second, such information is useful to the Optimizer Component which, given a workflow and an accuracy budget attempts to speed up the processing and harness memory utilization by replacing exact operators (e.g. for cardinality estimation) with equivalent, approximate ones. Here we note that this facility does not involve following some of the processing paths shown in Figure 4, but instead augmenting the QueryableState part of Flink's DataStream API, so that it provides synopsis-specific information at runtime.

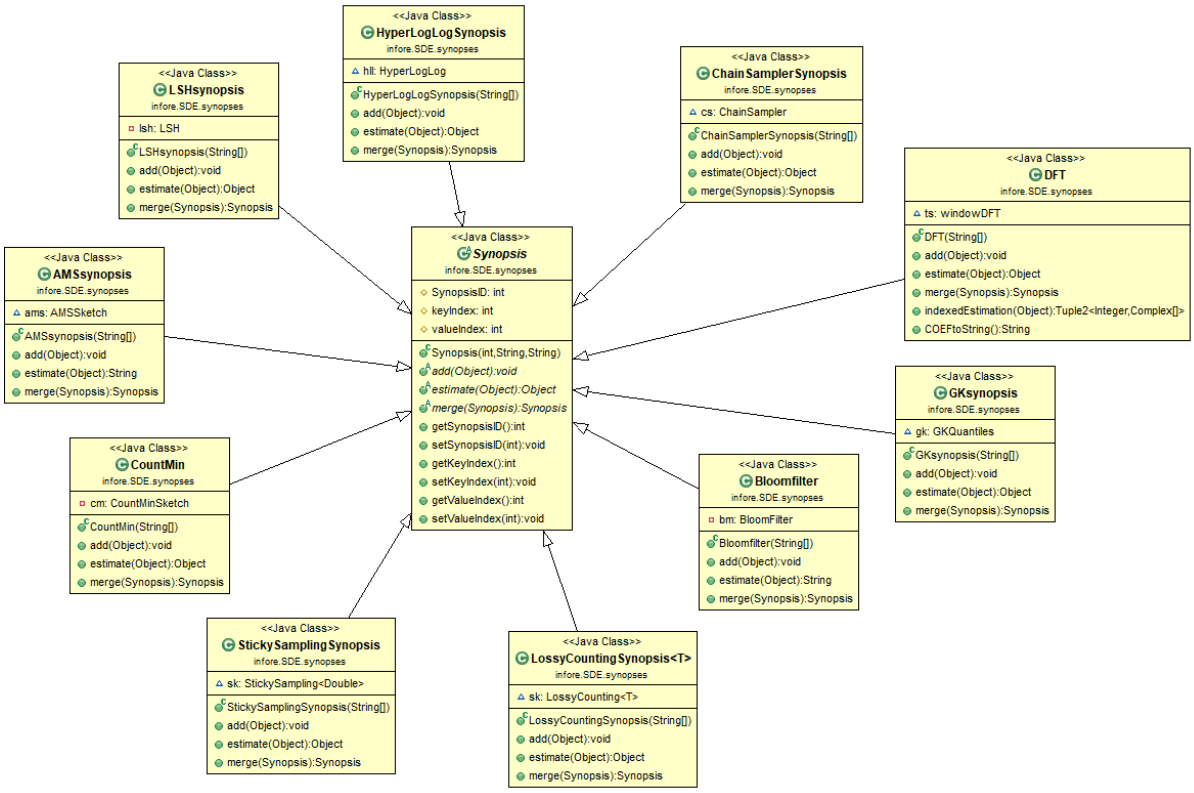

Figure 5: Overview of the SDE library.

As shown in Figure 5, the SDE library leverages subtype polymorphism to ensure the facile pluggability of new synopses definitions. In a nutshell (please see Deliverable D6.1 for further details), a generic Synopsis class is extended by classes implementing algorithms of specific data summarization techniques, and their processing methods override those of the parent class.

4.3.6 Complex Event Forecasting Component

The Complex Event Forecasting Component provides algorithms for complex event processing and complex event forecasting. Like the Synopsis Data Engine Component, the Complex Event Forecasting Component consist of a library providing the algorithms and methods for complex event processing and forecasting and operators containing the configurations and information necessary to execute these algorithms and methods.

The physical implementation of operators enables the users of the INFORE Architecture to perform complex event processing and forecasting on their streaming data. These physical implementations materialize the respective Logical Operators the users include in a designed workflow by simply drag and dropping them using the Graphical Editor Component.

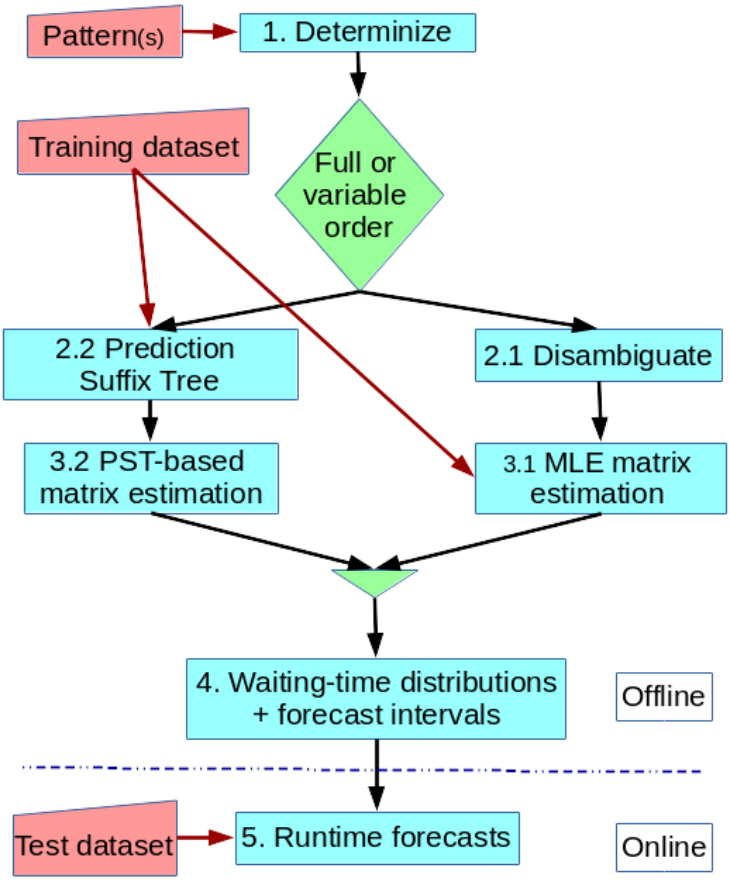

Figure 6: Internal architecture of the Complex Event Processing and Forecasting (CEP/F) Component.

The internal architecture of the Complex Event Processing and Forecasting (CEP/F) Component is shown in Figure 6. The various modules of the CEP/F Component will be described in more detail in Deliverable D6.3. Here we provide an overview at a higher level of abstraction.

Before the CEP/F Component can start consuming streaming data (a "test dataset") and produce forecasts in an online manner, it must first go through several offline steps. These offline steps require two input types: a) First, a set of patterns/queries with which a user needs to monitor streams of input events. These patterns are expressed in the form of symbolic regular expressions, i.e., regular expressions whose terminal symbols are not simple characters but Boolean expressions. For simplicity, we assume that a single pattern is provided. If multiple patterns are provided, the same steps must be repeated for each pattern. b) Second, a training dataset, representative of the streaming dataset to be encountered in runtime. This training dataset is used in order to build a probabilistic model for the pattern. Based on this model, the actual forecasts will be built.

The first step (1. Determinize) is to use the symbolic regular expression of the pattern in order to construct an automaton that will act as a computational model for it. If we are interested only in CEP (and not forecasting), then the rest of the offline steps may be skipped. The automaton can be used to detect complex events by directly consuming a stream of input events. From an architectural point of view, it is important to note that the automaton itself acts as a prototype/template for continuously spawning multiple automaton runs that actually consume the input events. For example, in the maritime use case, a pattern may need to be applied on a per vessel basis. In this case, for each new vessel, a new automaton run is created that will be responsible for this vessel.

If the goal is to perform forecasting, then there are two paths that can be followed in order to build the required probabilistic model (path 2.1-3.1 and path 2.2-3.2). In both cases, the goal is to derive a description of a pattern's automaton in the form of a k-order Markov chain. The right-hand path in Figure 6 (2.1-3.1) corresponds to a straightforward method for deriving such a Markov chain, whereas the left-hand path (2.2-3.2) to an optimized method that allows k to reach higher values. This Markov chain essentially describes how the automaton can move among its various states. For each such state, we can use the Markov chain to predict how the automaton will behave (which states it will visit) and when it is expected to reach a final state and thus detect a complex event (step 4). We currently assume stationarity. As a result, for the automaton to function as a forecaster in an online fashion, all we need to do is to enrich each of its states with a forecast (the Markov chain itself may be dropped after the forecasts have been estimated). The engine then works in a manner like that for CEP, with the difference that each automaton run, besides a reference to its prototype, also has a reference to a lookup table of forecasts.

The CEF module will communicate with the rest of the architecture via Kafka. For the CEF to function properly, two upstream and one downstream Kafka topics are required. As described above, the CEF will function in a two-phase manner, according to requests coming from the upstream topics: a) an "offline", training phase that will construct the necessary probabilistic models for a set of patterns, and b) an online testing phase, where the previously constructed models are used to actually generate forecasts. The three Kafka topics are the following:

- A ConfigTopic. Through this topic, the CEF module receives requests for training and testing. A training request must be accompanied by a set of patterns (along with their orders and partition attributes) and a set of declarations. A testing request must be accompanied by values for the parameters of confidence threshold, maximum spread and horizon. For an explanation of these parameters and concepts, please consult Deliverable D6.3. This topic is also used for stopping a running CEF process, as well as for querying the CEF module for its status and its various parameters.
- A DataTopic. Through this topic the CEF module receives both the training and testing datasets, in the form of streams of events (tuples). As soon as a training/testing request arrives at the ConfigTopic, the CEF module starts consuming data from the DataTopic to either train its models or produce forecasts.
- A ForecastTopic. This topic is essentially used only in the testing phase. This is where all Complex Events and Forecasts detected and generated by the CEF module are written.

The Complex Event Forecasting methods are described in more detail in the Deliverables D6.2, D6.4 and D6.5.

4.3.7 Interactive Online Machine Learning Component

The Interactive Online Machine Learning Component provides tools for interactive online machine learning. Like the Synopsis Data Engine Component and the Complex Event Forecasting Component, the Interactive Online Machine Learning Component consists of a library providing the algorithms and methods for its purpose and operators containing the configurations and information necessary to execute these algorithms and methods.

The operators enable the users of the INFORE Architecture to perform interactive online machine learning on their streaming data, by simply drag and dropping them in the streaming analysis process, which is designed by using the Graphical Editor Component.

The Online Machine Learning Component learns expressive and interpretable complex event patterns from streaming input of time-stamped information. It functions in an online fashion, i.e., it continuously makes predictions (detects complex events of interest) on incoming data, using the labelled fragments of the data as feedback, from which it updates its current model (event pattern set), thus improving both its predictive performance and the quality of the learnt event patterns over time.

The learnt patterns have the form of weighted logical rules, and the learning algorithm is capable of both inducing their structure (the actual rules) and optimizing their weights. Together, the rules and their weights define a probabilistic predictive model that is resilient to noise and uncertainty. The learnt patterns may subsequently be used for complex event recognition & forecasting, while, thanks to their interpretability, they may also be used by human experts for acquiring novel insights about the application domain via simple inspection. In addition to learning event pattern sets from scratch, the Online Machine Learning Component is also capable of revising existing pattern sets (e.g. an initial, potentially crude set of patterns provided by domain experts) from new data that stream-in.

The Online Machine Learning Component will communicate with the rest of the architecture components via Kafka. For this, two upstream and one downstream Kafka topics are required. These Kafka topics are as follows:

- A ConfigTopic (upstream): Through this topic, the Online Machine Learning Component receives a configuration of its learning algorithm's hyperparameters: a learning rate (Double), a regularization rate (Double), a Hoeffding test statistical confidence threshold (Double), a pruning threshold (Double) and a loss function name (String). A detailed description of these hyperparameters will be included in Deliverable D6.2. Also, via the ConfigTopic, the learning algorithm receives a set of declarative syntactic specifications for synthesizing rules from the encountered data, an application-specific form of domain knowledge, which will also be detailed in Deliverable D6.2. An initial event pattern set, potentially provided by domain experts, and constantly under revision from that point on, may also be provided through the ConfigTopic.
- A DataTopic (upstream): Through this topic the Online Machine Learning Component receives its input data in the form of a stream of event tuples. The Learner continuously listens to this topic and consumes data that arrive there, to first make predictions with its current event pattern and then use any potential labels in the data to update the event pattern set.
- A LearningResultsTopic (downstream): The Online Machine Learning Component outputs its results to this topic. There are three types of information that are output here: (i) the actual predictions of the learner, i.e., the complex events it detects from its input stream; (ii) online learning statistics, such as the learner's evolving online error rate and prequential F1-score on the input data and mean CPU processing time per input data point over time; (iii) the learner's evolving model (event pattern set) over time.

The Interactive Online Machine Learning methods are described in more detail in the Deliverables D6.2, D6.4 and D6.5.

5 Technical Constituents of Streaming Analysis Workflows

The next sections cover the actual implementation details of the architecture components and the definition of the interfaces between them.

5.1 Realization of the INFORE Architecture

The conceptual design of the INFORE Architecture has been described in Section 4. In this section, we describe the concrete realization of its components.

The project partner RapidMiner provides RapidMiner Studio, an open-source software solution for the Connection Component and the Graphical Editor Component. The existing user interface is enhanced to implement the connections to the other components of the architecture. RapidMiner Studio offers already a graphical representation of an analysis process, by providing operators which can be placed and connected with Drag & Drop into a process design GUI. This allows to hide the complexity interacting with different technology stacks from the user, by providing a common usage concept. At its current state, the Studio does not provide support for the functionality envisioned by INFORE and, thus, will be significantly extended in the scope of the project.

In particular, INFORE adds a so-called **Streaming Nest** operator to RapidMiner Studio. The Streaming Nest operator is a subprocess operator, which means that a family of operators can be placed inside of it. Streaming operators are placed inside the Streaming Nest operator and are connected during workflow design time to define the data flow of the streaming analysis process. Hence the subprocess of the Streaming Nest operator is the implementation of the Graphical Editor Component. Synopses Data Engine, Complex Event Forecasting and Interactive Online Machine Learning Logical Operators are added by INFORE, which are implementing the operator part of the corresponding components (see Sections 4.3.2, 4.3.5, 4.3.6 and 4.3.7). These "INFORE Component" operators contain all information to execute the specific algorithm and methods, which are provided by corresponding libraries. Figure 2 shows RapidMiner Studio, illustrating the drag and drop approach of the process design. The Streaming Nest operator and its subprocess is also demonstrated, as well as some example streaming and "INFORE Component" operator. Again, the front end allows to present an interactive workflow design to the users, while the underlying complexity is hidden from them. In the case of multiple available platforms, either the user can simply manually choose which platform to use, or call the Optimizer Component for an optimized workplan (see Section 4.3.4).

RapidMiner Studio also provides the concept of Connection objects. Thereby all information to connect to a specific system (for example a database) are packaged in an object inside RapidMiner Studio. This object can be stored in the RapidMiner Repository (the data and process storage system of RapidMiner Studio) and can be utilized in an analysis process by dragging and dropping it in the process design GUI. RapidMiner Studio also offers the possibility to easily create and configure these Connection objects. User management handling and the secure injection of critical information (e.g. passwords) are also provided by RapidMiner Studio. For more information about the Connection Management concept[14] of RapidMiner Studio. INFORE adds Connection object classes for all supported streaming backends (see Section 5.2), thereby implementing the described Connection Component (see Section 4.3.1). This is essential to support the cross-platform optimization of data stream analysis in INFORE.

Moreover, NFORE adds a Java library, implementing the functionality needed for the Manager Component (see Section 4.3.3). This Manager Component library provides capabilities to connect and receive resource information from a streaming computing cluster, to connect and consume data streams, to execute and deploy streaming analysis process and to produce data streams as an output. These capabilities are explained in more details in the following sections.

As an interface to the other INFORE Components, especially for the interface between Manager Component and Optimizer Component, a JSON representation of the streaming analysis process is added by the INFORE project. A ProcessToJSON converter class is implemented in the Streaming Nest operator. The inputs of this ProcessToJSON converter are the designed streaming analysis workflow and the resource information of the streaming processing backend. The first is provided by the Streaming Nest operator, the second is retrieved by utilizing the Connection object to connect to the streaming processing backend(s) and retrieving the corresponding information. The

[14] https://docs.rapidminer.com/latest/studio/connect/#connection-objects/

ProcessToJSON converter combines these two information sets and converts them to a JSON representation, which is provided to the Optimizer Component as an engine-agnostic workflow.

Engine-agnostic workflows are encoded into an AgnosticWorkflow class that consists of three core variables: Operators, OperatorConnections, Resources. These represent the operators of the workflow, the connections among them, and the resources allocated to the workflow, respectively. An example JSON representation of a simplistic workflow is depicted in Figure 7. This example shows an operator named “Logical Decision Tree” that gets its input from port “output 1” and propagates its result to the port “training set”. In this JSON description, the operator is a logical operator (isLogicalOperator=true). The operator description also specifies implementation details such as the class that implements this operator.

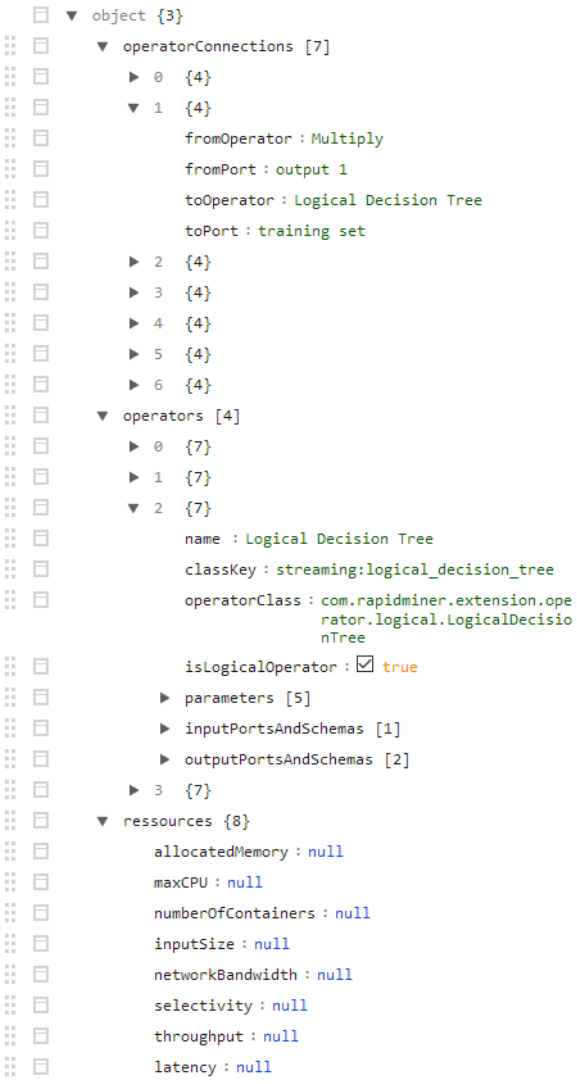

Figure 7: Example JSON representation of a streaming analysis workflow.

Table 2 gives an overview of the different fields and their meaning in the engine-agnostic workflow representation.

Key	Description	Key	Description
Operator Connections		isInputPort	Indicator if this is an input port
fromOperator	Name of the source operator	isConnected	Indicator if port is connected
fromPort	Name of the source port	schema	If object is a data set, schema of this data set
toOperator	Name of the sink operator	**Schema**	
toPort	Name of the sink port	fromMetaData	Indicator if schema is retrieved from meta data
Operator		size	Size of the data set at the port
name	(unique) name of the operator	attributes	List of Attributes
classKey	RM specific key for the operator class	**Attribute**	
operatorClass	Name of the java operator class	name	Name of Attribute
isLogicalOperator	Indicator if the operator is a logical one	type	Value type
parameters	List of Parameters (see below)	specialRole	Special role
inputPortsAndSchemas	List of input ports and their schema (see below)	**Resources**	
outputPortsAndSchemas	List of output ports and their schema (see below)	allocatedMemory	Allocated memory for the process
Parameter		maxCPU	Maximum number of CPU for the process
key	Key of the parameter	numberOfContainers	Number of containers used
value	Current parameter value (as String)	inputSize	Size of the input data
defaultValue	Default value (as String)	networkBandwidth	Available network bandwidth
range	String representation of the range	selectivity	Indicator of the selectivity
typeClass	Name of the Java parameter class	throughput	Current throughput
PortAndSchema		latency	Current latency of the network
name	Name of the in- or outputport		
objectClass	Name of java class of the object delivered or received at this port		

Table 2: Overview of the different fields and their meaning in the engine-agnostic workflow representation.

One of the facilities of the Manager Component is to convert or package one or more sub-sets of operations of the streaming analysis workflow (or the whole workflow) into a deployable artefact, which can then be deployed on one or more compute backends. The implementation of this feature is provided as part of the Manager Component's deployment functions (see Section 4.3.3). From the graphical modelling point of view, this can be achieved in a couple of ways. In the first scheme, the Optimizer may decide (during optimization of workflow) to place a certain group of operators as single or multiple deployable artefact(s) on a certain backend, and actuate on this decision, by invoking the relevant functions of the Manager Component.

In the other scheme, the human designer who creates the graphical workflow may indicate a set of operations as a single deployable artefact by grouping them. For example, this may be the case when one compute backend (say a Flink cluster) is available (or preferred). In this case, the logical operators can be configured to use Flink-based concrete implementations for data preparation, transformation, modelling and other operations. With this knowledge, the human designer may configure the NEST operator to: i) not rely on Optimizer for placement decisions or ii) indicate to Optimizer to respect the indicated grouping, or even iii) let the Optimizer overrule these groupings

(placement indicators) if it deems fit. This concept will be further refined as the prototyping work progresses, but the idea is to aim at maximum flexibility for the end users.

In the following section, we briefly describe the technical constituents of an INFORE streaming analysis workflow, and how INFORE processes abstract over the various streaming Big Data platforms.

5.2 Producers, Consumers and Compute Clusters

As explained in Section 4.3, the INFORE streaming analysis workflow is composed of input stream(s), the streaming analysis workflow and potentially output stream(s). The input operators of the analysis workflow are sometimes referred to as downstream or input receiving operators. The output operators are sometimes referred to as upstream or input providing operators. Entities which generate the stream and consume it are referred to as Producers and Consumers, respectively. Computer clusters refer to the stream processing backend, which are often used to deploy the stream processing workflow so complex computation is performed in a scalable manner on dedicated resources.

5.2.1 Establishing a reliable, robust, generic and flexible interface between disparate Components

The added value of INFORE is that its streaming analysis workflows follow an abstraction approach that allows them to span multiple streaming Big Data platforms for producers, consumers and compute clusters for execution. INFORE delivers this flexibility by performing the heavy lifting needed to integrate with different backends as part of its components may run on different Big Data platforms. Some of the biggest technical challenges faced here are listed below in terms of feature requirements:

- **Interfacing heterogenous or disparate components**: Providing a high level of abstraction is a huge challenge because a mechanism needs to be devised that would allow INFORE Components to communicate in a *reliable* and *robust* manner with possibilities to incorporate future components, which are themselves expected to be disparate and available on heterogenous backends. An acceptable solution should allow to provide a *generic* interface, which is easy to adopt across the different tiers of the platform.
- **Interfacing stream sources and sinks**: Another hard requirement is to let a variety of producer and consumer streams to be integrated. The schema of the data tuples is not always pre-defined or known but can be analysed at design time. Hence, it is highly desirable to use a *generic communication middleware* with support for simple and complex data structures. A feasible solution would not just make the sources and sinks (implemented in any language) available over a standard interface, but also the flexibility to deal with different data (tuple) types, a criteria to slice and dice different windowing of streams and ideally also support some degree of persistence and querying.
- **Dispatching and deploying streaming analysis workflows:** Dispatching (deploying) streaming analysis workflows to available compute cluster or a compute cluster of choice is a task which often requires to communicate with a specific Big Data platform using its custom APIs or client utilities, but here again INFORE provides a generic layer within the Manager Component (in principle, similar to Apache jClouds[15] library that allows to create applications that are cross-cloud portable).

Keeping these requirements in sight, it becomes vital to adopt a state-of-the-art communication middleware for INFORE. This middleware is expected to deliver above requirements and would serve at different tiers of the INFORE architecture to communicate with internal components (Synopses Data Engine, Complex Event Forecasting), various sources and sinks as well as assist in fetching results from ongoing experiments on the HPC (e.g. in the Life Sciences use case). Overall, this messaging and communication middleware must allow INFORE system to function seamlessly and detach the sender and receiver and receiver sides through a messaging queue, which serves to pass data as topics or digests. Thus, the choice of this middleware required a comparative analysis of the leading related technologies including RabbitMQ, ActiveMQ, Flume and Kafka to name the top candidates. A short overview of this comparative analysis is presented next.

[15] https://jclouds.apache.org/

5.2.1.1 Comparison of leading communication middleware technologies

ActiveMQ

Apache ActiveMQ[16] is a Java-based multi-protocol messaging server, which allows to integrate multi-platform applications using the Advanced Message Queue Protocol (AMQP). In addition, it also supports other open wire protocols like OpenWire, Stomp and MQTT. It is more suited for transmitting data in binary rather than text-based format. Its server side consists of Master broker nodes and a set of slave nodes, which can be paired together so that data can be provided efficiently to consumer processes, address fault-tolerance and move data between nodes. ActiveMQ can be setup as an embedded application with a small footprint, which then serves as a messaging endpoint for inter-application communication.

Although ActiveMQ allows for applications written in different languages to pass messages asynchronously, in its current implementation, its feasibility in high-throughput and transmission of large data streams is not the best. There is no easy way to batch messages together and it is assumed that it uses a batch size of 1. Experiments conducted on streams of logging data [1] revealed that the persistence mechanism in ActiveMQ consumed 70% more disk space than Kafka to store the same set of 10 million messages. The lag in performance at the persistence tier is attributed to the usage of JMS (Java Messaging System) specification which requires a thorough message header. The server processes seem to get hogged into maintaining the persistence indexes, that require B-Tree instance(s) to maintain metadata and state of each message. On the functional side, ActiveMQ is not intended for performing stream processing operations like windowing, aggregations or groupings on messages. Hence, its adoption would not satisfy most requirements of INFORE architecture and use cases.

RabbitMQ

RabbitMQ[17] is a lightweight and widely deployed message brokering system implemented in Erlang. Like ActiveMQ, it also supports several messaging protocols including AMQP, Stomp and MQTT. The server side of RabbitMQ supports consists of a cluster of nodes, which provide high availability and data replication at the persistence tier, for messages exchanged between producers and consumers. Connections from clients, the channels and queues used to pass data, are distributed across the nodes, to offer an efficient and scalable handling of data. In this aspect, RabbitMQ make up a formidable candidate technology for INFORE's communication middleware.

However, despite some interesting features, RabbitMQ is not a technology that is targeted for high throughput data stream processing as required by various INFORE use cases, intra-component interaction and interfacing between operators of INFORE's streaming analysis workflows and INFORE components. A performance evaluation of RabbitMQ regarding the production and consumption of high-throughput messages yielded results similar to ActiveMQ and inferior to Kafka. In [1] , Kafka is also compared with RabbitMQ. Kafka could produce up to 50,000 messages per second for a batch size of 1, which was two times higher than RabbitMQ. Configuring larger batch sizes in RabbitMQ is also not obvious. When message consumption is evaluated, Kafka performed four times faster than RabbitMQ (and ActiveMQ) by consuming upto 22,000 messages per second. This overhead is associated with the fact that RabbitMQ maintains the delivery state of each message, while Kafka does not. Finally, RabbitMQ requires the setup of Erlang execution environment on the machines. As Erlang is not so widely available or platform independent (unlike Java), the provisioning of Erlang runtime poses an additional requirement. Overall, RabbitMQ is not the best fit to be adopted as INFORE's messaging and communication middleware.

Flume

Apache Flume[18] is a project that provides a simple and flexible architecture based on streaming data flows. A Flume system is made up of Flume agents, which are Java processes. A Flume agent acts as a middle-man between the producer and consumer of data. The agent internally has a source component, a sink component and a channel in which data received by the agent is placed via the source component. The source component receives data from an external producer. The sink component reads the message from the channel and passes it out to an external consumer. The internal channel is backed up by a data store. A Flume agent can be configured in a multiplexing manner, i.e. it can maintain multiple channels which pass on the data to one or more external consumers (such as an HDFS cluster

[16] https://activemq.apache.org/
[17] https://www.rabbitmq.com/
[18] https://flume.apache.org/index.html

or a Kafka topic). Flume can be used for interaction between different applications or components that need to exchange or collect data. Flume agents can be horizontally scaled. Flume guarantees reliable delivery of a message, which is called as an event in Flume. Although Flume provides an interesting data flow management middleware, it is rather intended for scenarios, which need to push high throughput data from many sources into a collector sink, via a data-lake system that may perform transformations on passed data.

From the INFORE perspective, Flume has some major shortcomings. Flume does not offer a persistence tier as part of its framework. This implies that data in channels is not replicated across the Flume agents. Flume only provides weak ordering on transmitted messages, and duplicate messages are often sent due to its emphasis on reliable durable delivery, but which need to be cleaned at the consumer side, especially given their ability to induce noise. Flume's capabilities are further undermined due to its more complex management overhead, which can affect its throughput handling, scalability and reliability aspects if the data stores backing the channel(s) are not appropriately configured for all agents [2] . Due to its rather limited and very data-lake specific features, Flume is not a candidate for adoption in INFORE.

We conclude our comparative analysis with Kafka, which we present in the next dedicated section, to highlight how it is a clear winner to be adopted as our *communication middleware of choice* for INFORE.

5.2.1.2 Kafka - Communication middleware of choice

Kafka is a highly scalable and fault tolerant distributed data stream platform. It provides persistent storage on the server side, which is based on a cluster of so-called broker instances. The brokers provide data storage and replication by means of partitions in the broker. This leads to high availability of data and acts as a scalable tier that can deal with large number of producers and consumers - without causing contention at the network or storage level access. Due to these properties, Kafka plays a central role in the INFORE Architecture as its communication middleware of choice. It is employed due to its flexible API- based features, a highly performant backend that can deliver high throughput, large-scale data streams with reliability, strong ordering guarantees, replication-capable and scalable server-side. These features are able to meet data exchange and interfacing requirements of INFORE use cases and components. Hence, in the following, we describe how Kafka usage in INFORE Architecture spans across the board - from integrating different components and interfacing INFORE's streaming analysis workflow (operators) with different producers and consumers (sources and sinks).

As noted, initial prototyping efforts could leverage Kafka for bridging disparate components, which are distributed over different locations. In particular, in Sections 4.3.5, 4.3.6 and 4.3.7 we showcased that the Synopses Data Engine, the Complex Event Processing and Forecasting, as well as the Interactive Online Machine Learning Components of INFORE are interfaced with upstream and downstream operators via Kafka topics. Thus, treating a "topic" as a medium for asynchronous communication, applications written in any language and having complex dependencies can be loosely coupled using a clear interface and minimal effort. Kafka topics are at the heart of its publish/subscribe mechanism, that can send and receive continuous streams of data. Kafka producers write data or data streams to topic(s), which may be persistently stored on the brokers, and the consumers retrieve them via topic-based access.

Consumer access is also highly scalable and can be parallelized leveraging the same principle of clustered brokers and replicated partitions at the server side. A typical Kafka program (represented as a processor topology – see description of Stream API below) can use multi-threading to achieve concurrent execution, which improves performance. In this way, Kafka provides a technology that can pass data streams efficiently from producers to consumers and allows to write stream processing workflows that can be tuned for high performance and scalability.

Kafka provides a set of APIs, which are worth mentioning briefly:

- Connect API: This API allows to write connectors for bringing data from a source into the Kafka system and to bring data out from Kafka into a sink system.
- Producer API: This API can be used by Kafka applications to create topics and send data streams to topics.
- Consumer API: This API allows to receive data streams from Kafka topics into a Kafka application.
- Stream API: This API allows to process data streams from input topic(s), which transforms them to output stream(s), which can be written to other topic(s). A computation step (transformation) is referred as a processor node. The configuration of these nodes is called a processor topology.

- Admin Client API: This API allows to manage topics, brokers, access control lists and other objects on the Kafka server side.

To integrate Kafka in the INFORE streaming analysis workflows, the Manager Component functions and the Graphical Editor Component (operators) would be implemented to make use of Kafka APIs. Together with the Connection Component object, the operators for performing transformations enable the following capabilities:

- Connecting with the Kafka backend, creating topics, sending and receiving data streams to and from topics.
- Performing typical transformations on incoming data streams. This includes windowing, joining, aggregation and grouping, filtering, computing some statistics, mapping the tuple(s), training or updating models, applying models, etc.

These examples illustrate how Kafka plays an important technical role at different levels of the INFORE architecture by simplifying the complexity of integrating legacy, heterogenous or niche systems, helps standardize several interfaces in INFORE and provides a central communication technology for connecting the different components in the INFORE Architecture.

In the following section, we provide a short description of how INFORE deals with the requirement of dispatching (deploying or placing) streaming analysis workflows on compute clusters or Big Data processing platforms. The list of supported platforms is not fixed. The idea is to extend the support to more platforms in a gradual fashion.

5.2.2 Incorporating Big Data Platforms for Dispatching Streaming Analysis Workflows

As a reference on how Big Data platforms are incorporated in INFORE after the Optimizer Component selects a specific platform for dispatching or deploying a part of a streaming workflow, we take Flink for a more elaborate discussion. Besides streaming Big Data platforms, INFORE would also incorporate support for HPC infrastructures as the development progresses.

Flink provides a state-of-the-art implementation of streaming analysis functionality. It is also used as the implementing technology for components of the INFORE Architecture such as the Synopses Data Engine. Hence it is a very good demonstration of integrating a streaming analysis technology into the INFORE Architecture. In the following we describe how the Flink is integrated into the INFORE Architecture.

Flink consists of several distinct components, that interact with each other to provide a functional and scalable streaming environment. The server side of Flink comprises mainly of Job Manager, Resource Manager and Task Manager components.

The *Job Manager* is responsible for the execution of a single application. The manager transforms the application data and workflow description into an executable unit and requests the required resources. The *Task Managers* provide the slots for the actual execution. The slots inside each task manager run as threads in the same JVM instance, while several task managers can work on the same application, thus sharing data between separated JVMs.

In addition, the *Resource Manager* distributes requests of the *Job Managers* to the *Task Managers*. It can work with external resource managers such as YARN, or as a standalone deployment. Depending on the amount of parallelism and resources available, a single job can be deployed on multiple *Task Managers* and Flink can rescale running jobs.

Flink offers two different styles for deployment: framework and library. The framework deployment bundles the application into a JAR file and hands it over to a running service, like a resource manager or directly to a Flink Job Manager. This way an application can be directly planned for execution (in case of the submission to a Job Manager) or scheduled to hand over to a Job Manager (e.g., via YARN or a Flink dispatcher).

The library style uses two independent Docker images: one for bundling the application (including Job- and Resource Manager) and one for running the Task Managers. The framework deployment is a way of submitting an application via a client to an already running service. For large scale infrastructure deployments, this approach is probably easier to maintain and set-up. While the library style is a bit more flexible and more suited for a microservice oriented approach.

In regard of the INFORE architecture both styles can be easily integrated and do not differ too much. The framework style is the one that can, in combination with an existing resource manager, be deployed in existing cluster settings where additional task managers can spin up on demand and the processes can easily be scaled out. The deployment as a library is useful for rapid prototyping and testing out applications in a local setting.

Flink provides two core APIs, namely the DataStream API and the DataSet API. The DataStream API allows to manage bounded or unbounded streams of data and the DataSet API allows to manage bounded data sets. Flink also offers a Table API, which is a SQL-like expression language for relational stream and batch processing that can be easily embedded in Flink's DataStream and DataSet APIs.

For integrating Flink capabilities in INFORE, the functions of the Manager Component and the operators of the Graphical Editor Component will be implemented to wrap various features of the Flink APIs. Under the hood, this would allow to create a Flink program that can be deployed on the Flink cluster. A Flink program can perform various transformations on the data stream. These transformation operators are combined into a sophisticated topology called a streaming dataflow.

The execution of this dataflow is inherently parallel and supports distribution. Flink performs various execution-time optimizations on the streaming dataflows. During execution, a stream is divided into one or more stream partitions, and each operator is logically divided into one or more subtasks. The operator subtasks are independent of one another and can be executed in different threads. For distributed execution, Flink chains operator subtasks together into tasks and each task is executed by one thread. Chaining operators together into tasks is a useful optimization as it reduces the overhead of thread-to-thread handover, buffering and increases overall throughput while decreasing latency.

Due to these capabilities, Flink plays an important role in the INFORE architecture. It is currently used for implementing components or incorporating them as a streaming application. These components include the Synopsis Data Engine (SDE), a version of the parameter server for machine learning operators (another one is being implemented in Akka) and a proof-of-concept for the Complex Event Processing/Forecasting Component. However, we emphasize that INFORE Architecture is not limited to Flink as a streaming Big Data platform, in contrast, INFORE's approach is broad-scoped and purpose-built for abstraction i.e., it specifically aims at incorporating multiple platforms.

5.3 Creation and Deployment of a streaming analysis workflow using the INFORE Architecture

In the previous chapters, we described the component-based structure of the INFORE Architecture. The provided functionality and the interaction between the components are described in detail. In addition, challenges and requirements from integrating and building a *cross-platform* framework for different streaming technologies are discussed.

We now present a step-by-step process on how a streaming analysis workflow is created, optimized and deployed using the INFORE Architecture. The different steps and interactions and how the components come into play are detailed:

1. **Design of the streaming analysis workflow:**
   The user of the INFORE Architecture designs a streaming analysis workflow. Therefore, she uses the **Graphical Editor Component** (see Section 4.3.2) to define the logic of the streaming analysis, without the need to take care of the technology-specific details.
   a. The user can select Logical streaming Operators, which provide an abstraction level of generic streaming analysis functionality over the streaming platforms providing this functionality. The **Optimizer Component** (see Section 4.3.4) later chooses the streaming backend used, optimal for the current workflow.
   b. Functionality provided by the **Synopsis Data Engine Component** (see Section 4.3.5), **Complex Event Forecasting Component** (see Section 4.3.6) and the **Interactive Online Machine Learning**

Component (see Section 4.3.7) can be leveraged through the corresponding operators in the graphical editor.

c. The user creates Connection objects for input and output streams and streaming backends (for a detailed description of the **Connection Component** and the functional concepts of input and output streams and streaming backends, see Section 4.3.1) through the graphical editor. The user only needs to provide essential information to connect to the streams. The Connection objects are used in the design of the streaming workflow in the same drag-and-drop manner as the streaming operators.

2. **Handover of streaming analysis workflow to Manager Component:**
When the design process of the workflow is finished (e.g., when the user presses a submit button in the extended RapidMiner Studio), the workflow is handed over to the **Manager Component** (see Section 4.3.3). The Manager Component converts the workflow into its JSON representation (see Section 5.1). It also uses the Connection objects to connect to the provided input and output streams and streaming backends and retrieves information about volume and schema of input data, available resources of the streaming engines and output streams. This information is added to the JSON representation as well. The JSON is handed over to the **Optimizer Component** (see Section 4.3.4).
3. **Optimization of the streaming analysis workflow by the Optimizer Component:**
The **Optimizer Component** converts the JSON representation to the tool-agnostic workflow representation. An optimization of the workflow is performed. Depending on the user-specified parameters, the optimization can include the selection of the concrete implementations of Logical Operators (providing cross-streaming-platform optimization of the workflow), the execution order and bundling of operators to specific streaming executing jobs, the insertion of synopses and function and data shipping (moving execution to the data or vice versa). More details about the optimization will be included in Deliverables D5.1, D5.2 and D5.3.
4. **Providing optimized workflow back to Manager Component:**
The Optimizer Component provides the Manager Component with the optimized workflow. This optimized workflow is visualized in the Graphical Editor Component to inform the user about the changes of the Optimizer Component.
5. **Deployment of the workflow by the Manager Component:**
The Manager Component prepares the execution of the optimized workflow by creating execution jobs defined by the bundled operators in the workflow. The streaming execution jobs are deployed on the streaming backends by the Manager Component.
6. **Monitoring and management of the deployed workflows:**
The Manager Component can be used to monitor deployed workflows. If specified by the workflow, statistics are collected and provided to the Optimizer Component, and used for displaying an overview of the running workflows to the user through the Graphical Editor Component. Running workflows can be aborted, paused, edited and resumed through the graphical user interface. If conditions change (e.g. changing input data volume, changing available resources, etc.), steps 3. - 5. can be repeated to optimize running workflows to the changed conditions.
7. **Retrieving and consuming results:**
Depending on the specification of the designed workflow, results of the streaming analysis workflow can be provided by different means. Output streams can be created, which can be further consumed by different streaming consumers (with or without using the INFORE Architecture). Snapshots of streamed results can be stored for further batch processing or inspection. Webservices or monitoring dashboards can deliver the results to the target users of the designed streaming analysis workflow.

This step-by-step process is also illustrated in Figure 8.

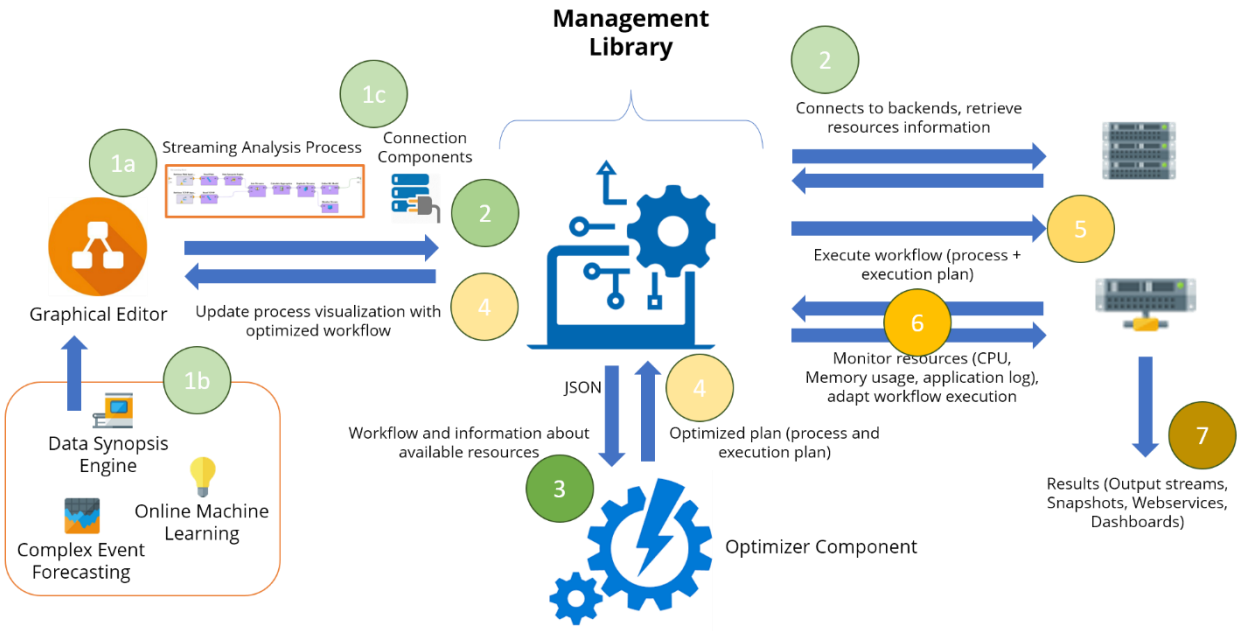

Figure 8: Overview of the step-by-step process for creating and deploying a streaming analysis workflow in the INFORE Architecture.

6 Conclusion

The project architecture presented in this report is defined as a result of requirements analysis from use cases, discussions among project partners, review of related projects and the state-of-the-art in streaming technologies. In this first project year, the focus was on realizing an integrated, extensible and flexible design for creating, optimizing and executing streaming analysis workflows. The initial prototyping of the INFORE architecture is in progress and the workflows from project use cases are planned for implementation. The layout of various workflows is being worked out and would further help in refining the architecture, e.g., in terms of the exact functionality and methods needed and which end-to-end execution is used. Based on the joint work achieved so far, the next steps seem to be feasible, on-track and expected to accelerate the incorporation of workflows across the use cases into the initial prototype.

In the following months, the initial system prototype and software components will be further developed (and documented in Deliverable D4.2). The objective remains to ease the specification of complex data processing workflows for non-expert programmers, while leveraging heterogeneous technology stacks to the maximum possible extent. In the next year, we also aim at collecting feedback for testing and further enhancement of the INFORE architecture from different stakeholders (use case and technical partners in the project), who would be involved in implementing the workflows. Additionally, we plan to explicitly address the dissemination with respect to the release of various software stacks (components) developed in the project, along with their usage documentation, to further enlarge the impact of INFORE architecture.

7 References

[1] J. Kreps, N. Narkhede, J. Rao. “Kafka: a distributed messaging system for log processing”. The 6th International Workshop on Networking Meets Databases, Athens, Greece, 2011.

[2] Pankaj Misra, Tomcy John. “Data Lake for Enterprises”. Packt Publishing. 2017. ISBN: 9781787281349