On the Use of Provalets in a Predictive Maintenance Use Case

Adrian Paschke

Freie Universitaet Berlin, Germany
paschke@inf.fu-berlin.de

Abstract. In this paper we report on a predictive maintenance use cases using Provalet rule agents for implementing expressive rule-based streaming analytics and decision logic on top of online machine learning prediction models, which are dynamically applied to the streaming data coming from on-board asset monitoring sensors. Provalets are component-based mobile agents for rule-based inference analytics, which can be dynamically deployed as microservices into container environments via simple REST calls.

1 Introduction

More and more streams of data are being generated in real-time from a myriad of data sources, such as market data, mobile devices, Internet of Things (IoT) sensors, clickstream analysis as well as business internal transactional systems. Real-time insights (by reactive analytics) and predictions (by predictive analytics) must be derived from this data in motion to give a competitive edge to agile organizations that want to proactively act on these insights before they lose their value. Instead of collecting and analyzing all this data only in a centralized cloud, new distributed microservices architectures and mobile software agents for moving the streaming analytics and inference logic closer to the data sources at the edge (edge analytics) or fog (fog computing) are needed.

As a solution Provalets [11, 12] support an easy to use component-based microservice architecture for mobile rule agents. They act as a basis for an analytics-as-a-service business model, where data-driven inference and analytics operations are provided as component-based Provalet agents. Provalets encapsulates data intensive rule-based data processing and inference reasoning with decision and reaction logic [10] into rule-based Prova agents [13] [1], which are deployed as microservices into standardized containers such as Docker and trusted OSGi [8] container environments. Via REST-based interfaces Provalets can interact and can be composed to data analytics/inference pipelines. Instead of sending all data to a centralized external provider, the processing and inference agents can be moved closer to the data and their execution remains in the control of the data producer. Hence, privacy problems are avoided and additional

[1] http://prova.ws

permissions can be defined in the agents metadata and policies, so that the controlled Provalet container environment can enforce them and reject agents that request permissions that cannot be granted.

This paper will report on an industrial use case for Provalets in predictive maintenance. In the manufacturing industry high value is associated with the early discovery, warning, prediction, and prevention of anomalies by proactively triggering optimized maintenance actions. In condition-based maintenance sensors monitor the normal operating conditions of an asset and conduct maintenance based on the conditions assessed by the sensors. This requires efficient streaming analytics and continuous online machine learning of the prediction models in combination with event-based reaction rules / logic. The data processing and analytics should be done as close as possible to the IoT sensors (and gateways), in order to address typical problems such as security, privacy, scalability, reliability (e.g. offline connection problems), etc.

The further paper is structured as follows: In section 2 we begin by explaining the background of machine learning and streaming analytics / event pattern mining approaches in predictive maintenance. In section 3 we describe a predictive maintenance use case. We then summarize the core principles of Provalets in section 4 and describe the implementation in section 5. We summarize the application in section 6.

2 Background

In today's industrial processes real-time insights into the underlying data and event streams to trigger optimal reactions and make situation-aware decisions are crucial for the business competitiveness. While real-time reactions on detected complex events are important, the goal is to predict critical event occurrences before they actually happen and to trigger proactive actions. Figure 1 illustrates this rapidly decreasing knowledge value of events.

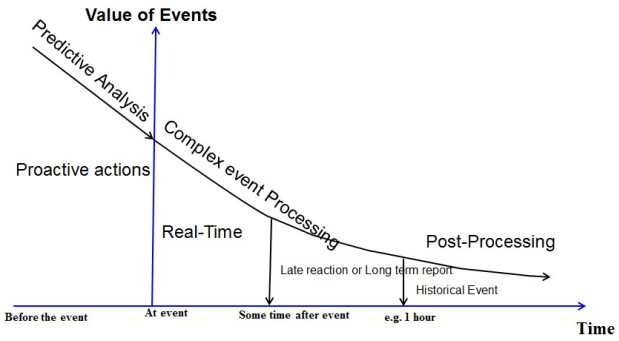

Fig. 1. Knowledge Value of Events in Streaming Analytics

For instance, in maintenance the value, e.g. in terms of life time and reliability of an asset/system, increases from pure *reactive maintenance* to *preventive maintenance* and *predictive maintenance* and further to *maintenance optimization*, as shown in figure 2.

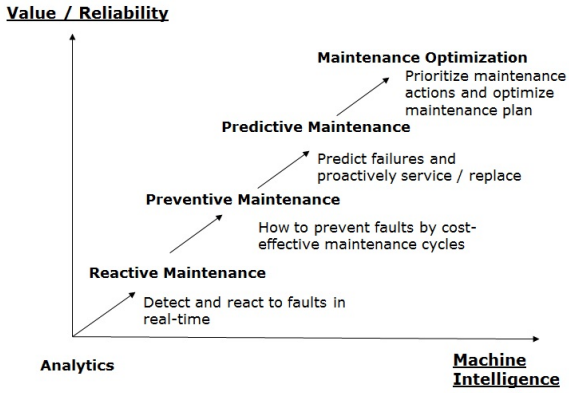

Fig. 2. Knowledge Value in Maintenance Analytics

While reactive maintenance tries to detect and react to failures in real-time, predictive maintenance foresees the breaking point of a system before it actually occurs and a chain of preventive and service reactions are triggered before expensive damages can occur. In more detail, the aims are to (1) reduce maintenance costs, e.g. by avoiding the entire system from getting harmed further by breakdowns (*corrective / reactive maintenance*), and by avoiding costs caused by scheduled, interval-based, preventive replacement of still functioning pieces (*preventive maintenance*), and (2) avoid breakdowns by triggering predictive maintenance activities as they are necessitated by their predicted life time or costs with respect to the actual conditions of the target system (*predictive maintenance*). *Condition-based maintenance* sensors continuously monitor the asset during normal operating conditions and a maintenance action is triggered based on their assessment. *Modular maintenance optimization* can further reduce costs and exploit synergy effects, e.g. by replacing components during light maintenance operations and thus bundling multiple predicted condition-based maintenance actions into one.

In contrast to reactions based on learned patterns from historic data, the challenge in condition-based maintenance is to predict unknown complex event patterns and abnormal episodes (*anomaly detection*) that did not occur in the system's past before. Different approaches can be distinguished such as:

– Supervised Anomaly Detection:
  The anomaly is an already known class and used to label the data records

for supervised learning of the anomaly classification model. This approach is applied for recurring anomalies, but it cannot be used for unknown anomalies.
- Static Unsupervised Anomaly Detection:
  The underlying solution idea is to learn the normal classes / event patterns from the monitored (sensor) data and detect anomalies as sudden occurrence of an unrecognized patterns.
- Dynamic Unsupervised Anomaly Detection:
  The system slowly deviates from past norms until an abnormal event is triggered, e.g. by exceeding the norms threshold. For instance, reliability measures, such as Mean Time Before Failure (MTBF), are used to scheduled preventive maintenance activities, where the goal ist to keep it running as long as possible, but avoid the system to break down completely, producing even more damage.
- Analysis of Time Series Data and Event Instance Sequences of the system:
  Allows to learn (machine learning) possible asset degradation models from time series data and detect anomalies with mined event patterns / episodes in order to base maintenance on continuously monitored conditions (*condition-based / predictive maintenance*). The conditions can be simple thresholds on a predicted decay trend, but can be also more complex event patterns and rules, taking into account the current conditional situation and event instance history.

In this use case we focus on condition-based predictive maintenance. The degradation models can be defined by physical laws determining the correct behavior of the asset or they can be learned from time series data by machine learning approaches. A common approach for the latter is to perform a regression analysis based on the data that comes from the sensors that are continuously monitoring the asset conditions. Figure 3 illustrates a typical regression analysis process.

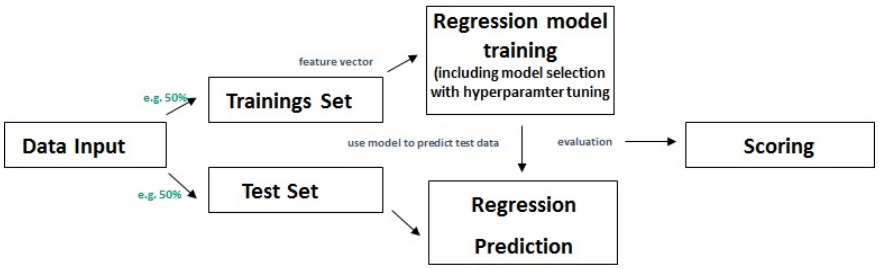

Fig. 3. Typical Process in Regression Analysis

Splitting the dataset into a training set and a test set, and selecting the right data has a direct impact on the quality of the final regression model. Hence,

instead of random data selection the semantics and distribution topology of the data can be used to select the right training data with respect to the observed data events. Different (increasing) sizes of training data need to be tested (in the example figure it is 50-50%) to achieve an accurate and statistical relevant training model.

Classical regression approaches [16] try to find the best approximating function which should be close to an unknown distribution function over an input space and output space sampled from a set of training data $D = (x_1, y_1), ..., (x_n, y_n)$. The quality in terms of average difference between the estimated and the actual outcome is measured by a loss function in the training phase.

Model selection then has the goal to select a final model that does not over- or underfit the given available data. Empirical risk minimization (minimization of the empirical error) typically leads to overfitting of the model. Costs functions are used to optimize the tradeoff between accuracy of the model and complexity of computing the approximation function. By properly tuning one or more hyperparameters in the tuning phase of the model selection the trade-off between the overfitting and underfitting tendency are regulated. However, finding the right values for the hyperparameters is problem-dependent and non-trivial.

Various ML approaches, from basic linear regression, random forest regression (RFR)[6] to support vector machines for regression[17], can be applied which not just support single outputs, but also multi-outputs. This includes approaches for transfer learning [9], multi-task learning [4] as well as solutions to decompose the multi-output problem into several single-output problems. In the Regularized Least-Squares (RLS) algorithm [1] the approximation function applies a non-linear mapping allowing to still coping with linear models by exploiting the representer theorem [15] and by reformulating the RLS problem using a kernel function[3], such as a positive definite Gaussian kernel (which is learning every possible function). The complexity is measured, e.g. with Euclidean norm of the set of weights describing the regressor. The loss function adopts, e.g. mean squared error loss. An alternative to RLS for searching the approximation function are Support Vector Models for Regression (SVMR) [17]. In contrast to RLS, where the solution is dense and the computational burden is high, the sparse solution of SVMR can be described by using only a limited subset of parameters. Accordingly it requires less training data and the computational burden is lower. Similar to RLS the SVMR training problem is reformulated using a kernel matrix formulation.
For tuning the hyperparameters of RLS and SVMR either rule of thumb heuristics or an exhaustive search for the optimal solutions is performed by solving SVMR /RLS multiple-times with different hyperparameters values. By estimating the generalization error of the regressor, e.g. using k-fold cross validation (KCV), the best hyperparameter values are found and the final model is trained with the values.

While these regression approaches are applied ex-post to the previously collected batch data and the model is computed only once, in Online SVMR (OSVMR) [7] the regression parameters are incrementally increased or decreased

each time a new sample is added. The model selection function is iteratively tuning the hyperparameters and recomputing the KCV validation. In order to balance the trade-off between expensive computational time and resources needed, and the accuracy of the selected model, Fumeo et al.[5] suggest a meta-heuristic optimization approach with an online KCV approach, which is exploiting the assumption that the set of hyperparameters will not vary too much from the previous computed best values, if the new updated data samples are not too large.

The (continuously) trained regression model acts as degradation model, which can be applied for regression prediction, i.e. predicting e.g. a trend in the decay of an asset. By defining a threshold it becomes possible to predict and decide when maintenance must be done, in order to avoid a breakdown. Figure 4 gives an example line plot displaying the predicted trend evolution of an asset's decay and defines a threshold used to trigger a proactive maintenance activity in order to avoid a breakdown.

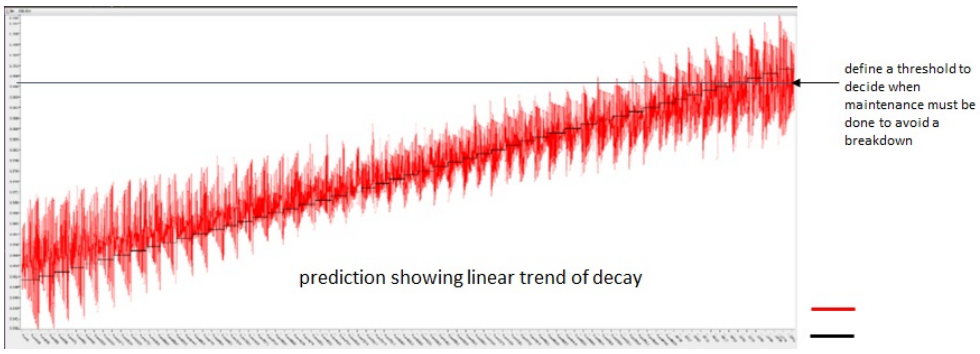

Fig. 4. Example Degradation Analysis with Maintenance Threshold

While this simple threshold-based maintenance trigger is an appropriate approach for degradation models with a clear linear trend applied to one particular component/asset, more sophisticated approaches are needed, if larger numbers of components and situation-awareness needs to be considered. Here one single occurrence of a threshold event might not be enough. A correlation between multiple events, possibly based on conditional situations, is required for predicting and triggering the predictive maintenance action. This leads to the need of complex event patterns and conditional rules.

Event Pattern Mining approaches learn patterns over time from event instance sequences (EIS). EIS are analyzed according to their occurrence and structure in order to mine, e.g. frequent patterns, association rules, time-series, episodes, and many more. Traditional Pattern Mining can retrieve various sorts of patterns as highlighted in Figure 5 and in [14]. These are split into basic, multidimensional and extended patterns and rules.

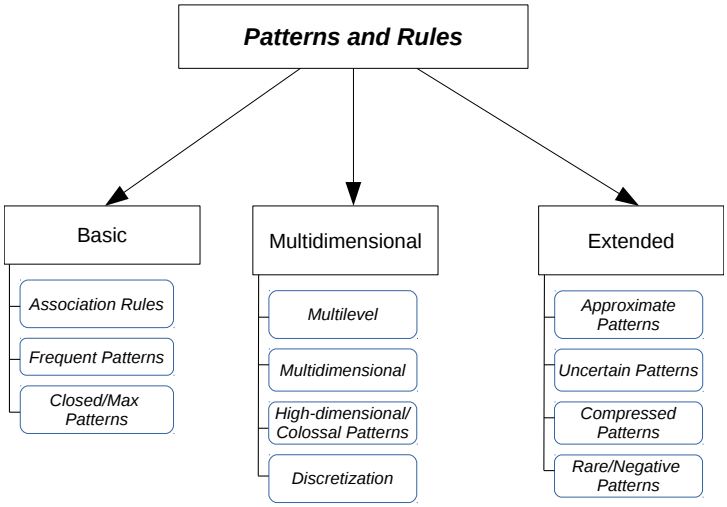

Fig. 5. Overview on Patterns and Rules

Figure 6 provides an overview on pattern mining research. General pattern mining methods are differentiated from applications and extensions. The Mining methods are split up into the basic methods, interesting patterns[2], distributed, parallel and incremental patterns, and extended patterns and pattern-based applications. The subcategories are explained in more detail in [14].

3 Predictive Maintenance Use Cases

The use case is based on a data source[3] from a numerical simulator of a naval vessel (Frigate) characterized by a Gas Turbine (GT) propulsion plant, consisting of components such as Propeller, Hull, GT, Gear Box and Controller. [2]. The data takes into account the performance decay over time of the GT components such as GT compressor and turbines. The propulsion system behaviour is described by the parameters:

- Ship speed (linear function of the lever position lp).
- Compressor degradation coefficient kMc.
- Turbine degradation coefficient kMt.

[2] The interestingness of a pattern is determined by means of the following criteria: *conciseness, coverage, reliability, peculiarity, diversity, novelty, surprisingness, utility* and *actionability*. These nine criteria can be further classified into *objective* and *subjective* patterns.

[3] http://archive.ics.uci.edu/ml/datasets/Condition+Based+Maintenance+of+Naval+Propulsion+Plants

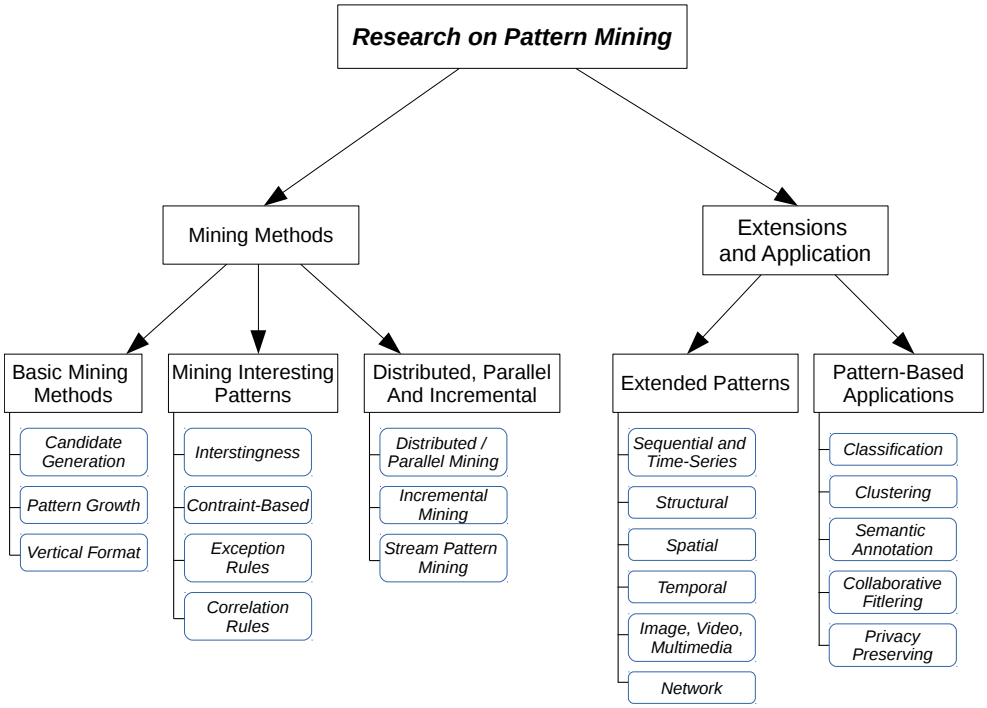

Fig. 6. Research Overview of Pattern Mining

so that each possible degradation state can be described by a triple (lp, kMt, kMc).

The range of decay of compressor and turbine has been sampled with an uniform grid of precision 0.001, discretized with a kMc coefficient within [1; 0.95] and a turbine coefficient within [1; 0.975]. Ship speed is sampled in the range of feasible speed from 3 knots to 27 knots with a granularity of representation equal to tree knots. A 16-feature vector (features: e.g. ship speed, high pressure turbine exit temperature, GT compressor inlet/outlet air temperature, ...) in the dataset is measuring ships state of the system subject to performance decay. The dataset is already clean, i.e. there are no missing data values (with two constant data values for the Compressor Inlet Air Pressure and GT Compressor Inlet Air Temperature).

Pearson correlation shows that there are correlations between some of the variables. By feature extraction 8 variables (HP Turbine exit pressure, Gas Turbine shaft torque, GT Compressor outlet air temperature, HP Turbine exit temperature, Turbine Injecton Control, Gas Generator rate of revolutions, Fuel flow, GT Compressor outlet air pressure). An auto-regressive (AR) model is trained on the data. By applying ML regression analysis (see section 2) different degradation models are trained. In particular, Online Support Vector Machines for Regression (OSVMR) [7] updates the trained regression function, whenever a new data sample is added to and irrelevant ones are discarded from the train-

ing set. This allows continuous OSVMR learning (with a Gaussian kernel and a non-linear optimization with Karush-Kuhn-Tucker conditions [16]) on the updates from the (simulated) monitoring sensors. Instead of doing a k-fold cross validation only once, the model selection (tuning the three hyperparameters of the kernel) is iteratively applied in each update step following the heuristic approach describe in [5] for optimizing the trade-off between the accuracy of the OL-SVR models and the computational time and resources needed in order to build them.

These learned models are used to predict anomaly-free measure values, i.e. "normal" behaviour. Some boundaries of normal functioning values are then defined around the average prediction error observed in training. As show in the line plot in figure 4 the prediction is showing a trend for the compressor decay. If the prediction error diverts from the boundaries, an alarm is triggered to alert that is time to inspect the machinery. The following reaction rules recognize the alarm situation and trigger a reaction:

```
PE(t)= abs(Prediction(t) x(t))

On each Frequency Band, compressor part
If PE(t) > Mean(PE(t)) + K*stddev(MA(t))
Then alm(t)= PE(t)
ELSE alm(t) = 0

MA(t)= Moving Average(alm(t), N=25, backward window)

IF MA(t) > Threshold Boundary
Do Alarm
```

4 Principles of Provalets

Provalets have much in common with apps in modern application-stores for mobile platforms. Details about underlying principles and the lifecycle of Provalets can be found in [11]. Here we briefly recapitulate the main principles.

Provalets are location-independent (mobile) rule-based software agents [13] which are deployed as microservices in component containers such as OSGi [8]. The implementation of Microservices is a software architecture style designing software applications as suites of independently deployable services providing (agent) intelligence in the endpoints, and decentralized control of languages and data. The Provalet microservices provide functional operations for rule-based linked data access [4], processing, inference reasoning and reactive messaging using Prova [5]. Prova (Prolog + Java) is both a declarative rule-based programming language and a Java-based rule engine. Provalets have a clear REST input and

[4] Prova has various built-ins for rule-based data access such as Java object access, file access, XML (DOM), SQL, RDF triples, XQuery, SPARQL

[5] http://www.prova.ws]

output interface, specifically an input URI and an output URI. They run in a controlled and secure container environment (OSGi or Docker). Provalets describe their functionality in terms of pre- and post-conditions on the sets of input and output data.

The container resource describes itself with metadata via a standardized API. A user or agent receives information about a container resource by sending an HTTP request to the container URI. For example, the container resource describes which permissions it can grant. To use a container resource to execute a Provalet the user sends an HTTP request adding three parameters to the container URI: the Provalet URI, the input URI and the output URI. This way it is straight forward to execute a Provalet from a standard Web client and lookup the results afterwards, by receiving the HTTP response of the output URI. Each Provalet has a unique URI that is resolvable via HTTP. Each Provalet is configured with one input URI that it is allowed to read from and one output URI that it is allowed to write to. Furthermore, the Provalet artifact address and the executing container resource need to be defined. The runtime environment should control which data type formats and which data sources are accessible by the Provalet including the control of permissions. The Provalet description also contains semantic metadata about the Provalet including runtime dependencies and policies such as permissions required on the runtime platform as well as the description of the functionality it provides in the form of statements about pre- and post-conditions over the sets of input and output data, definition of types, side-effects, legal norms and policies, etc. This supports automatic search of Provalets for their composition. The composition of Provalets is either executed by chaining the input and output connections via a pipes-and-filter streaming connections with a rule-based composition language or by a generic injection of mobile Prova code and consumption of their fully processed output. Typical workflow control constructs, such as sequential execution, parallel execution, conditional alternatives and repetitions, are supported in the rule-based composition language, as well as metareasoning and late binding capabilities.

Using conditional Provalet connectors Provalets can be chained and composed together, e.g. by splitting (on incoming input, multiple outgoing outputs) or joining (multiple incoming, one outgoing output) them. The data flow, e.g. in data pipelines, captures data dependencies between Provalet components. For the data passing between components we use Prova's event messaging rules [19], which not only can act as "data processors", but also can be used a basis for representing composite events, thereby implementing complex workflow patterns, especially state-based workflow patterns. In contrast to other related composition languages, we ground our rule-based composition language [21] on the logic-based semantics of Concurrent Transaction Logic, $\mathcal{CTR}$ [20], which gives a deductive database language, that integrates concurrency, communication and database updates in a semantic framework with a sound and complete model and proof theory.

Provalets describe permissions they require as metadata that is read by the runtime environment during deployment. By default Provalets are solely allowed

to see the data (streams) which are directly served by the configured input URI. Provalets may define additional required permission to access other data sources. For example to access additional static URIs or crawl URIs that are visible in the set of input data. The sources of data may be restricted by subnets, domains, protocols or even types of data a Provalet is allowed to see. Provalet may provide HTTP access credentials to the input and output resources upon request. Provalets must request permission to use additional computing resources on the machine they are executed. A Provalet may request harddisk space to store intermediate results. Other resources include memory, CPU time, account information, access to other web services. The latter can be used by a Provalet to enforce license models through trusted providers. It is the task of the runtime container of a Provalet to grant required permissions and allow access to requested resources.

5 Implementation

For the implementation of this use case we integrate Provalets into Knime data analytics workflows by implementing an user-defined Knime node for Provalets. Knime [6] provides a visual workflow language for modelling data analytics pipelines. We use this for the data preprocessing and AR - regression learning steps in the workflow, as shown in figure7.

For the abnormal event detection and reactions we define a Provalet that implements the two alarm rules. Since the input data from the Knime nodes are using the Predictive Modelling Markup Lanuage (PMML) an additional translator node is specified in the workflow which translates from PMML into Prova.

Provalets themselves are Maven OSGi artifacts. To generate a new Provalet in a Knime workflow a Maven archetype is used. An OntoMaven generated Provalet project provides all necessary dependencies and mechanism. The included *Provalet* class extends the *AbstractProvalet* from our ProvaletCore API and must be filled with the Provalet functionalities. The *ProvaletActivator* class extending the *AbstractProvaletActivator* serves as an OSGi entrance point to the Provalet. During the Provalet development the developer has to keep attention to only specify dependencies to APIs being OSGi capable. The artifact specification can be found in the Provalet description.

To execute the Provalet on the selected AR input resource (from the Knime AR model) the user needs to call the URI of a Knime workflow container resource (*containerURI*) via an HTTP GET request providing the URI of the Provalet (*ProvaletURI*), the input (*inputURI*) to the AR input resource and the output URI (*outputURI*), e.g. a REST call, as parameters:

```
<containerURI>?Provalet=<ProvaletURI>&input=<inputURI>&output=<outputURI>
```

In the OSGi framework (Apache Felix) the Provalet container bundle is started. It handles the Provalet call and answers the HTTP request with an

[6] https://www.knime.org/

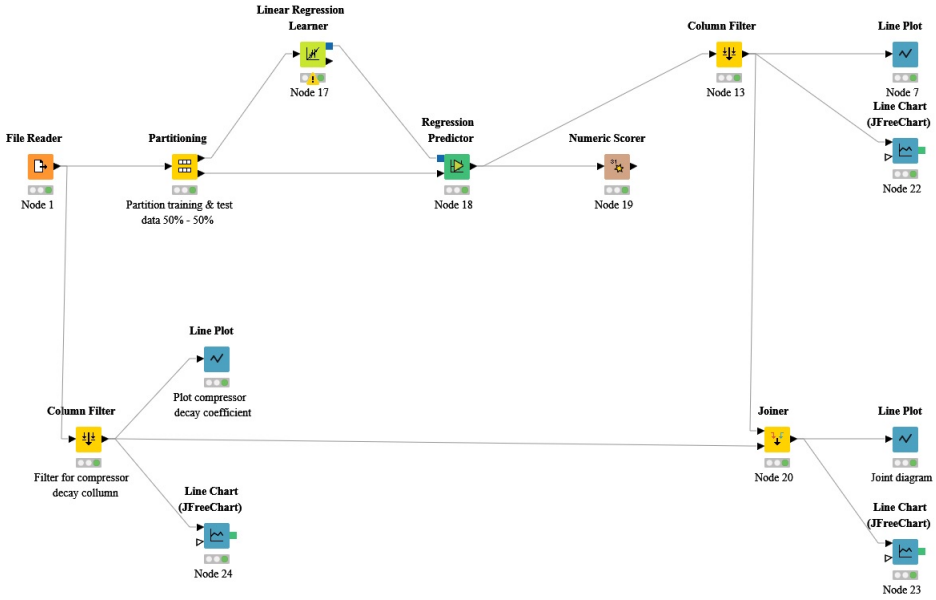

Fig. 7. Data Analytics Workflow for Predictive Maintenance

HTTP response message. First it resolves the Provalet characteristics by calling the *ProvaletURI* and reading the Provalet description which also includes the necessary artifact characteristics (groupId, artifactId, version and optionally the repository). The Provalet artifact and all its dependencies are resolved and downloaded to a local repository using the integrated Aether library [18].

The downloaded Provalet artifact are then deployed into the OSGi framework of the OSGI container resource and the Knime input representation of the *inputURI* is passed as an object to the working method of the Provalet. Finally the container resource starts the installed Provalet bundle. After execution of the working method the Provalet passed its resulting data back - the reaction trigger message, to the container. The container checks the contend and enforces restrictions on the Provalet execution and the output and writes the output representation to the `outputURI`.

Once the instantiated Provalet exists, verification of the Provalet constraints and rules can be performed using Prova's inference mechanisms. The OSGi bundle classloader is used to load the resources and instantiate a Provalet instance as OSGi component with the translated Provalet rules describing the alarm conditions and event pattern constraints.

The Provalet working modes of container resources are defined. Asynchronously working containers immediately respond with a HTTP response code indicating that the Provalet working method was successfully started. The user of an asynchronously started Provalet has in principal two possibilities to work with the results: (1) an agent polls the output URI after a defined time and (2) the

agent uses a subscription mechanism to be informed about updates in the output URI. In the synchronous working mode of a Provalet container the agent is redirected to the output URI once the results have been successfully written to the output URI. In this working mode the user can read the result immediately after receiving the HTTP response.

6 Summary

In this paper we reported on the use of Provalet microservices for predictive maintenance, with the goal of combining online machine learning and rule-based complex event processing / reaction logic. One advantage of Provalets is that the mobile rule agents can be deployed directly into containers (OSGi, Docker) running on the gateways or IoT monitoring sensors and hence can process and analyze the streaming data where it is produced.

7 Acknowledgments

This work has been partially supported by the "InnoProfile-Corporate Smart Content" project funded by the German Federal Ministry of Education and Research (BMBF) and the BMBF Innovation Initiative for the New German Länder - Entrepreneurial Regions.

References

1. A. Caponnetto and E. De Vito. Optimal rates for the regularized least-squares algorithm. *Foundations of Computational Mathematics*, 7(3):331–368, 2007.
2. A. Coraddu, L. Oneto, A. Ghio, S. Savio, D. Anguita, and M. Figari. Machine learning approaches for improving conditio-based maintenance of naval propulsion plants. *Journal of Engineering for the Maritime Environment*, –(–):–, 2014.
3. N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines: And Other Kernel-based Learning Methods*. Cambridge University Press, New York, NY, USA, 2000.
4. T. Evgeniou, C. A. Micchelli, and M. Pontil. Learning multiple tasks with kernel methods. *J. Mach. Learn. Res.*, 6:615–637, Dec. 2005.
5. E. Fumeo, L. Oneto, and D. Anguita. Condition based maintenance in railway transportation systems based on big data streaming analysis. *Procedia Computer Science*, 53:437–446, 2015.
6. A. Liaw and M. Wiener. Classification and regression by randomforest. *R News*, 2(3):18–22, 2002.
7. J. Ma, J. Theiler, and S. Perkins. Accurate on-line support vector regression. *Neural Comput.*, 15(11):2683–2703, Nov. 2003.
8. OSGI Alliance. OSGi Service Platform, Core Specification, Release 4, Version 4.2. Technical report, OSGI Alliance, Sept. 2009.
9. S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Trans. on Knowl. and Data Eng.*, 22(10):1345–1359, Oct. 2010.

10. A. Paschke. Rules and logic programming for the web. In *Reasoning Web. Semantic Technologies for the Web of Data - 7th International Summer School 2011, Galway, Ireland, August 23-27, 2011, Tutorial Lectures*, pages 326–381, 2011.
11. A. Paschke. Provalets - osgi-based prova agents for rule-based data access. In *On the Move to Meaningful Internet Systems: OTM 2015 Conferences - Confederated International Conferences: CoopIS, ODBASE, and C&TC 2015, Rhodes, Greece, October 26-30, 2015, Proceedings*, pages 519–526, 2015.
12. A. Paschke. Provalets - component-based mobile agents as microservices for rule-based data access, processing and analytics. *Journal Business & Information Systems Engineering*, 5, 2016.
13. A. Paschke and H. Boley. Rule Responder: Rule-Based Agents for the Semantic-Pragmatic Web. *International Journal on Artificial Intelligence Tools*, 20(6):1043–1081, 2011.
14. R. Schäfermeier, A.-A. Todor, A. La Fleur, A. Hasan, J. Einhaus, and A. Paschke. Corporate smart content evaluation. Technical Report TR-B-16-02, Freie Universität Berlin, 2016.
15. B. Schölkopf, R. Herbrich, and A. J. Smola. *A Generalized Representer Theorem*, pages 416–426. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.
16. B. Scholkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA, 2001.
17. A. J. Smola and B. Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 14(3):199–222, 2004.
18. Sonatype. Aether. http://aether.sonatype.org/, June 2011.
19. Z. Zhao and A. Paschke. Event-driven scientific workflow execution. In *Business Process Management Workshops - BPM 2012 International Workshops, Tallinn, Estonia, September 3, 2012. Revised Papers*, pages 390–401, 2012.
20. Z. Zhao and A. Paschke. A formal model for weakly-structured scientific workflows. In *Proceedings of the 6th International Workshop on Semantic Web Applications and Tools for Life Sciences, Edinburgh, UK, December 10, 2013.*, 2013.
21. Z. Zhao, A. Paschke, and R. Zhang. A rule-based agent-oriented approach for supporting weakly-structured scientific workflows. *J. Web Sem.*, 37:36–52, 2016.