pro sense

The Proactive Sensing Enterprise

ON-THE-FLY AND OFF-LINE SENSORY ANALYTICS

Deliverable nº: D2.3.2

EC-GA Number:	612329
Project full title:	The Proactive Sensing Enterprise

pro sense The Proactive Sensing Enterprise		Work Package:	**WP2**
		Type of document:	**Prototype**
		Date:	**08/07/2016**
Grant Agreement No 612329			
Partners:	**JSI**		
Responsible:	**JSI**		
Title:	**D2.3.2 On-the-fly and Off-line Sensory Analytics**	Version: **1.0**	**Page: 2 / 52**

Deliverable D2.3.2
On-the-fly and Off-line Sensory Analytics

DUE DELIVERY DATE: M32

ACTUAL DELIVERY DATE: JULY 2016 (M33)

Document History

Vers.	Issue Date	Content and changes	Author
0.1	2015-04-20	1st draft	Luka Stopar
0.2	2015-07-01	Revisions based on internal review by Nenad Stojanovic	Luka Stopar
0.3	2015-07-02	Revisions based on internal review by Ana Rita Campos	Luka Stopar
1.0	2015-07-08	Final formatting and layout	Luka Stopar

Document Authors

Partners	Contributors
JSI	Luka Stopar, Primož Škraba
MHWirth	Ehsan Peymani

Dissemination level: PU

Document Approvers

Partners	Approvers
Nissatech	Nenad Stojanović
UNINOVA	Ana Rita Campos

Executive Summary

This report documents the 2nd version prototype of the components developed in tasks T2.3 and T2.4 of the ProaSense project. Based on the revised architecture of deliverable D1.3, three components were developed. The first component is the Sensor Enricher, responsible for aligning raw data streams and enriching them with contextual/background knowledge and statistical aggregates. The Sensor Enricher produces a joint representation in the form of feature vectors, providing a common representation for the machine learning algorithms used by other components. The second component is Online Analytics. The Online Analytics component uses trigger in the form of JavaScript callbacks to produce use-case specific events. The final component is StreamStory. StreamStory is an interactive data stream modelling tool which uses a multi-scale representation to hierarchically represent data in a qualitative manner using states and transitions. It supports two modes of operation: an offline mode used to explore data and an online mode where its hierarchical models are applied to real-time data. When used in the online mode, a StreamStory model detects anomalies and can be configured to output predictions and detect activities.

We present the feedback from the first iteration cycle performed as part of task T7.5 and the revisions implemented in the second release of the prototype, along with a summary of changes compared to the 1st version prototype.

TABLE OF CONTENTS

Acronyms

Acronym	Explanation
OODA	Observe, Orient, Decide, Act
API	Application Programming Interface
WP	Work Package

1. Introduction

As European companies are facing increasing pressures due to globalization, uncertainties and increased regularities, they are forced to manage their production at the margins of performance, achieving better control through the whole production process. To cope with these challenges, dynamic business networks need to enhance their monitoring capabilities, within the network and across different levels. Real-time monitoring is required and has already been implemented in some enterprises in the form of extensive sensor systems. This is however only the first step to manage problems in such complex environments. The next step is enabling real-time monitoring to cope with the scale and dynamics of the business context. This kind of monitoring is the basis for a new level of (sensing) performance, since it is not only sensing these problems, but also focusing on a proactive approach in resolving problems before they occur.

ProaSense is a FP7-ICT project addressing the issue of the “digital enterprise”. Our aim is to design and develop methodologies and tools that support proactivity in the ecosystems of digital enterprises, making them able to anticipate problems and opportunities, and support their realization and resolution. As such, it has the potential to improve the competitiveness of European industry, enabling and shaping future developments in ICT, so that the demands of its society and economy are met.

To achieve this aim we follow a cognitive approach based on the Observe, Orient, Decide and Act (OODA) loop of situational awareness, recognized as one of the main models for the big data supply chain. In this model, decision making occurs in a recurring cycle of unfolding interaction with the environment, oriented via cues inherent in tradition, experience and analysis.

This deliverable presents the work done in the final years of tasks T2.3: Sensing Storage Analytics and T2.4: Sensing on-the-fly analysis of WP2. The main goals of these tasks are: (a) to develop off-line analytic methods for enrichment, in-depth analysis and interpretation of sensory data and (b) to develop on-the-fly (streaming) analytics methods for anomaly detection, prediction and interpretation of data streams.

Their core research contribution is provided by using a multi-scale methodology, representing data on several aggregation levels, providing ground for qualitative interpretation of the data and studying its integration into the streaming environment.

Based on the refined architecture of D1.3, we have developed three components in the Observe and Orient phases of the OODA loop: Sensor Enricher, Online Analytics and StreamStory, implementing the qualitative multi-scale methodology and offering data enrichment, prediction, anomaly and activity detection services to the components in other work packages.

1.1 OBJECTIVES OF THE DELIVERABLE

The deliverable presents the 2nd version prototype of the components developed as part of tasks T2.3: Sensing Storage Analytics and T2.4: Sensing on-the-fly analysis of the ProaSense project.

Based on the revised ProaSense architecture presented in deliverable D1.3 the prototype consists of three components, residing the real-time processing and analytical layers and implementing three sub-tasks of task T2.3: (a) data enrichment and alignment, (b) machine learning algorithms supporting supervised and unsupervised scenarios and (c) algorithms for explaining the hidden causal structure in the data and three sub-tasks of task T2.4: (a) unsupervised anomaly detection, (b) prediction and (c) interpretation of the stream structure, and offering prediction, anomaly and activity detection services to other components in the architecture.

The main objective of the deliverable is to describe these components from a technical point of view including their detailed architecture, functionality and implementation details.

1.2 STRUCTURE OF THE DELIVERABLE

The deliverable is structured as follows. Section 2 gives an overview of the components developed in tasks T2.3 and T2.4, including the feedback and revisions of the first evaluation cycle and a summary of differences compared to the first release. Section 3 provides a specification of the prototype developed in the second release, including the architecture of all the components, a detailed description of their functionalities and a description of the user interface. Section 4 presents the application of the components in the ProaSense project along with the technical validation. Section 5 presents the organization of the source code along with a list of all external libraries used to implement the components and their functionality. Finally, we present some conclusions in Section 6.

2. Overview of the WP2 Analytics Components

The goal of tasks T2.3 and T2.4 is to cover the building blocks for online and offline processing of sensory inputs. This includes three major components: (a) the Sensor Enricher component for enriching and aligning the raw data stream, (b) Online Analytics using use-case specific models to offer prediction and anomaly detection services and (c) StreamStory, modelling the data in a qualitative manner using a multi-scale methodology, offering state-based anomaly detection services and allowing the configuration of generic predictions and activity detection services.

2.1 EVALUATION FEEDBACK AND ISSUES ADDRESSED

This section summarizes the evaluation feedback gathered as part of task T7.5, presenting a summary of the identified issues and the revisions implemented in the second release. Based on the results of task T7.3, the first testing period focused on qualitative heuristic validation through scripting trials where evaluators were presented with several scenarios and, on each step, identified usability issues based on several heuristics, along with a textual description and severity level. Additionally, after finishing the scenarios, the evaluators were enquired about the ergonomics and ease of use of the platform, user interface and their stress level when using the component by employing questionnaires.

Analysis of the questionnaire results showed that users had trouble interacting with the platform. Based on the gathered responses, we categorize the issues into two categories:

- **Difficulties configuring the component:** much of the negative feedback gathered from the questionnaire results (SS4, SS6, SS7 and SS9 in deliverable D7.2) shows that users had difficulties interacting with various forms in the user interface. To alleviate this issue, several forms were redesigned and “help” icons added to each step, explaining the various options and consequences. Additionally, the forms have been extended with feedback in the form of notifications indicating the success of each step.
- **Difficulties interpreting the visualization:** another part of negative feedback addresses the difficulty in evaluators interpreting our qualitative state and transition based visualization (SS8 and SS11 in deliverable D7.2). To alleviate this issue, we implemented several new visualization services which assist users in identifying the meaning of states and transitions in our visualization. These services can be grouped into three categories:

- The first category includes services to suggest users the meaning of states. This group includes two services: the automatic state labelling service uses a data-driven approach to automatically label each state according to the attribute most specific for that state and the state narration service which generates a human-readable description of the state in the form of a few short sentences.
- The second category includes services to visualize the properties of each state and assist users in identifying the meaning of states. In addition to the attribute highlighting service already presented in deliverable D2.3.1, the second release adds two new services to the second category: the decision tree visualization visualizes the properties of the state using decision trees and the rule extraction service describes the properties of states using rules which describe the range of attributes inside the state.
- The third category includes additional and improved visualization services, which visualize some properties of states and transitions. This category includes timeline histograms visualizing when in time the state occurred and parallel coordinates.

Qualitative heuristic validation identified several bugs and missing features in our user interface. Based on the analysis of the user feedback we group these issues into the following categories:

- **Look and feel:** this category includes issues related to navigation, design, consistency, etc. and the general look and feel of the component. For example, many evaluators were unable to find the "Options" button on the main visualization panel. Indeed, in the first release the button only had an "Options" icon without any text. Other issues identified in this category include: could not find the model after dataset upload, no confirmation when saving and visualization not returning to previous state. For the second release all these issues were reviewed and fixed, help icons were added to form elements and visualization panels, describing what is shown and what the next step should be. Additionally, in the second release our user interface includes a login mechanism and dashboard where users can better create and manage their models.
- **Issues related to Internet Explorer**: several issues identified by heuristic validation were related to the use of Internet Explorer for evaluating the component. Indeed, we had failed to test our component with Internet Explorer before the first evaluation cycle. For the second release, we have fixed all these issues and performed extensive testing using several popular browsers including Chrome, Firefox, Internet Explorer and Opera.

- **Issues related to pre-configured models:** when evaluating the MHWirth use-case, we had technical difficulties due to the testing rig being taken to cold storage. We thus had to deploy our component on one of our own servers. Additionally, we were asked by the use-case partners to pre-construct the models used in the evaluation scenarios which rendered the evaluators unable to perform the first few steps in each of the scenarios – i.e. uploading the dataset and configuring the model.
- **Missing features:** one of the issues identified was the inability to modify an existing model once it had been created. Although direct modification is not possible due to the use of randomized algorithms, the second release stores the configuration used to create the model and allows users to modify and use it to create a new model, replacing the existing one.

2.2 DIFFERENCES COMPARED TO THE 1ST RELEASE

This section presents an overview of the differences of the 2nd release of the components developed as part of tasks T2.3 and T2.4 compared to the 1st release. These differences can be summarized in the following subsections.

Dashboard and user authentication: the second release of the components includes a dashboard with an authentication mechanism, where users can create and manage their StreamStory models.

State layout: to provide a better user experience, the algorithm used to layout the states in our hierarchical representation was extended with a repulsive scheme, ensuring that states on the same scales do not overlap.

Novel hierarchy construction algorithm: the second release provides an alternative hierarchy construction algorithm to agglomerative clustering presented in deliverable D2.3.1. The algorithm represents a top-down transition-based approach (in contrast to the bottom-up distance based approach used by agglomerative clustering) and constructs the hierarchy by recursively bi-partitioning the Markov chain by approximating the *min-cut* problem on the graph induced by the Markov chain. Additionally, the levels of the hierarchy are grouped and a single representative of each group selected in the final representation. This provides an additional perspective on the multi-scale structure of the data. The two approaches can be used to highlight different multi-scale aspects via a top-down versus a bottom-up approach.

Visualization services: to improve the intuitiveness of the component, several specifically designed visualization services were added or improved. These include automatic state labelling, state narration, timeline histograms, decision trees, contextualized histograms, etc. These are presented as part of the technical specification in Section 3.1.3.4.

Activity detection: StreamStory has been extended to support semi-automatic activity detection. In our framework, an activity is defined as a sequence of states, which is then matched to the sequence observed from the real-time data stream.

Support for new types of attributes: to support new use cases, StreamStory was extended to support categorical attributes, which we found were common to many datasets (e.g. contextual data). These attributes are internally represented as categorical feature vectors where each category is represented as a binary feature. The second release also allows users to configure "ignored" attributes. These attributes are not used in the model construction phase but are still visualized in the user interface. They can be used when searching for certain correlations between attributes or to identify certain states without influencing the state construction procedure.

Inclusion of time-based features into state construction: the second release adds an option to use features extracted from the timestamps of observations in the state construction phase. These are extracted as a categorical feature vectors and depend on the time unit used when configuring the model. For example, when using the monthly time unit, the feature vector will represent months in a year. Time-based features may help identify recurrent behaviour in the dataset by imposing a cyclic structure on the states.

New use-case scenarios: In the second release, two new use case scenarios were implemented. The first scenario predicts scrap based on the sensors installed at the production plant. Two models were developed in this scenario. The first model aggregates the sensor readings during a shift into feature vectors and predicts the final scrap rate at the end of the shift. In contrast, the second model models the probability of each individual part being scrapped when arriving at the end of the production line. The second scenario focuses on activity detection and detects slips cycles using StreamStory's activity detection service.

3. Technical Specification of 2nd Prototype

This section presents the technical specification of the 2nd prototype of the components developed as part of tasks T2.3 and T2.4. As part of these tasks, we have implemented three components highlighted in the overall ProaSense architecture in Figure 1.

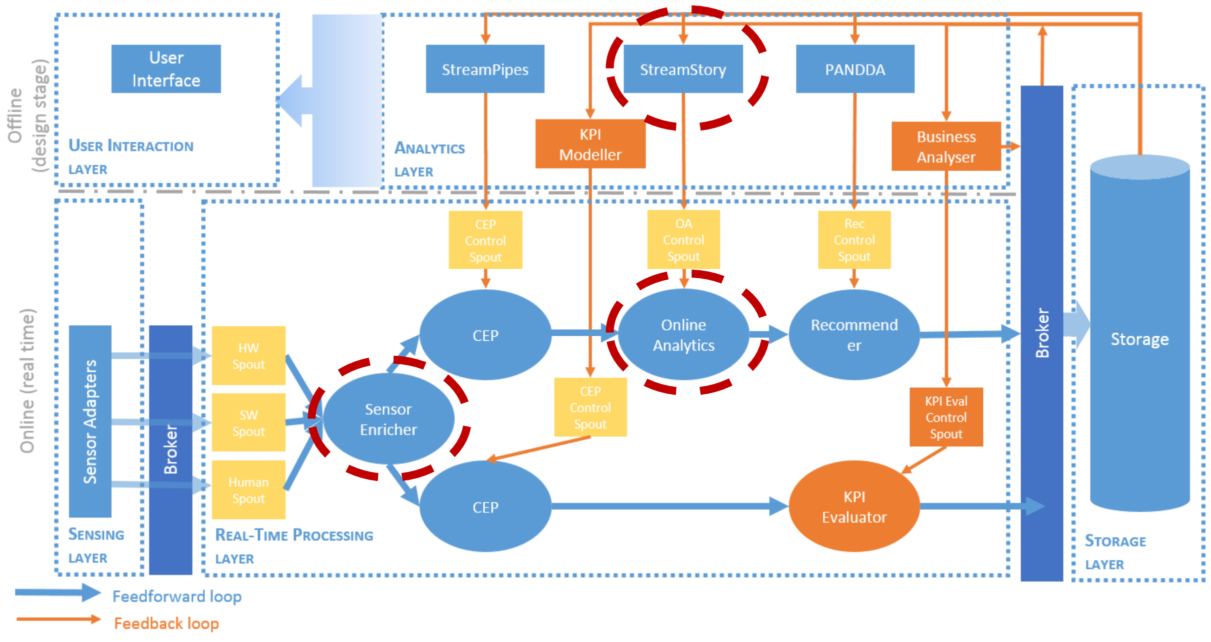


Figure 1: High level architecture of the ProaSense system, highlighting the components developed as part of tasks T2.3 and T2.4.

The first of these components is the Sensor Enricher, which is the first bolt in the ProaSense real-time processing layer. The Sensor Enricher is responsible for the enrichment and alignment of the raw real-time data streams, constructing feature vectors and providing a common representation for statistical/machine learning algorithms. It does so by merging the data stream and appending statistical aggregates along with domain specific knowledge.

The second component is the Online Analytics component, which is the third component of the upper branch of the ProaSense real-time processing layer. The Online Analytics component consumes the output of the Sensor Enricher and employs use-case specific models to produce predictions and detect outliers.

The third and final component is the StreamStory component, which resides in the analytics layer of the ProaSense architecture. StreamStory is the implementation of the multi-scale analysis approach, first presented in deliverable D2.3.1 and represents our core research contribution to the ProaSense project. StreamStory models the data streams using states and transitions in a hierarchical manner by constructing a hierarchical Markov chain model in an unsupervised fashion. It can be used in two modes: an offline mode where users can upload any dataset and interact with the hierarchical representation and an online mode where users can monitor their data in real-time and configure generic prediction, anomaly and activity detection services. As such, StreamStory implements two sub-tasks of task T2.3: (a) machine learning algorithms supervised and unsupervised scenarios and (b) algorithms for explaining the hidden causal structure in the data, and three sub-tasks of T2.4: (a) unsupervised anomaly detection, (b) prediction to extrapolate the development of data in the future and (c) interpretation of the stream structure to explain the hidden structure in the data.

The components developed in tasks T2.3 and T2.4 are implemented as modules in the QMiner (http://qminer.ijs.si) data analytics platform developed in at JSI. QMiner provides several functionalities for in-memory data storage, processing of large-scale, real-time, data streams and implements a comprehensive set of machine learning techniques. Its core functionality is implemented in C/C++ and exposed as a Node.js add-on, which makes it efficient and easily integrable with other systems. Some of its functionalities used in the ProaSense project include:

- **StreamAggregates**: these are a core functionality of QMiners' stream processing pipeline. They are algorithms which consume one or more data streams and produce some aggregate statistic as an output stream. StreamAggregates are mainly used by the Sensor Enricher for feature engineering and merging raw streams.
- **Feature Extractors**: allow for the transformations of raw data into feature vectors which can be consumed by machine learning algorithms at later stages. The primary feature extractors include: numeric, categorical, multinomial, vector space and others. Feature extractors are primarily used by StreamStory.
- **Machine learning algorithms**: QMiner implements a comprehensive set of techniques for supervised and unsupervised learning on data streams.

The components and their functionality will be presented in the following subsection.

3.1 SPECIFICATION OF SUB-COMPONENTS

3.1.1 SENSOR ENRICHER

The Sensor Enricher component, highlighted in Figure 2 is the first component in the ProaSense real-time processing layer. It receives raw data streams directly from the sensing layer and is responsible for their alignment and enrichment with background/domain knowledge and statistical aggregates.

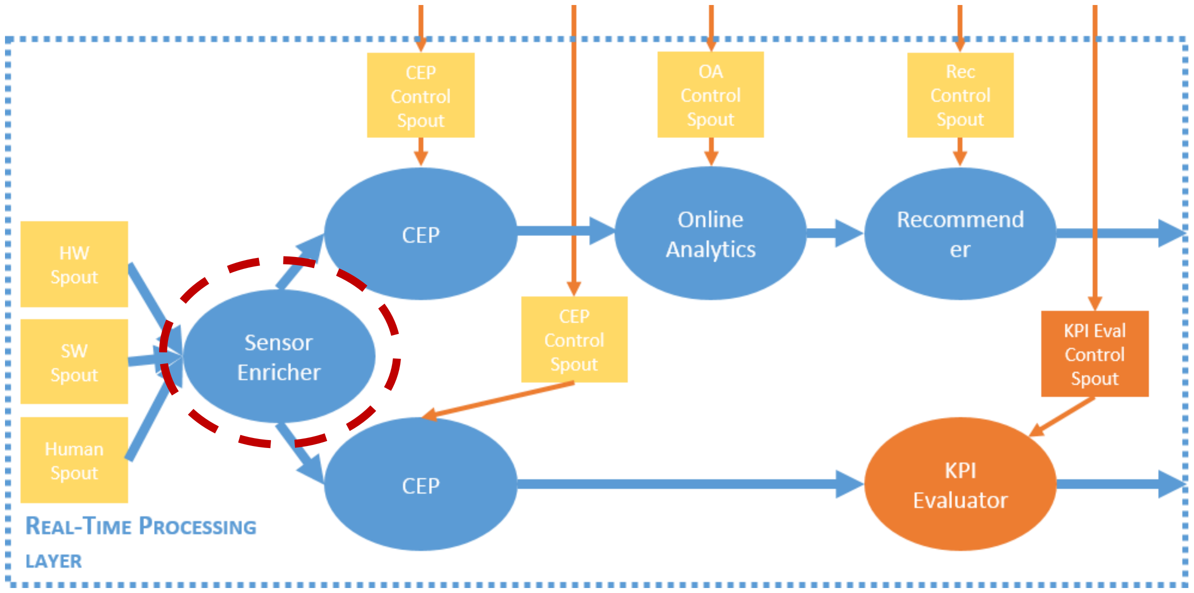


Figure 2: The Sensor Enricher component is the first component in the real-time processing layer of the ProaSense architecture. It lies between the sensing layer and CEP engine and is responsible for aligning the incoming data streams and enriching them domain knowledge and statistical aggregates producing feature vectors.

The output of the Sensor Enricher is sent to the CEP engine via the broker and is consumed by the components later in the real-time processing layer. Figure 3 shows the conceptual architecture of the Sensor Enricher.

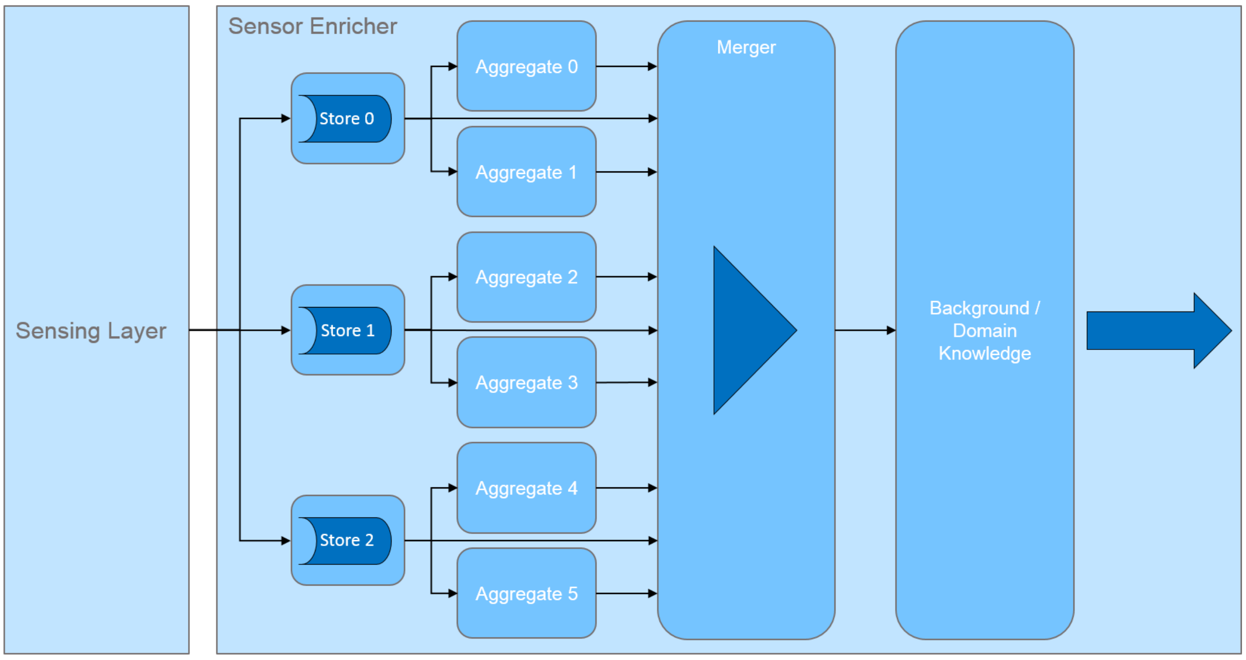


Figure 3: The conceptual architecture of the Sensor Enricher component.

In the first step of the architecture, multiple raw data streams are consumed and statistical aggregates appended. The supported aggregates are:

- **low pass filters:** such as moving average (MA) and exponential moving average (EMA),
- **moving variance and covariance:** the variance and covariance calculated on a sliding window,
- **filters**: filter observations based on design-time criteria,
- **histogram:** an online histogram which is maintained over a sliding window and produces a multi-dimensional output,
- **minimum, maximum and sum:** calculated on a sliding window and
- **linear regression:** univariate linear regression applied online and can also compute quantiles,

The second step merges the output of the first step into feature vectors in the form of a single multivariate data stream using the Merger sub-component introduced in deliverable D2.3.1, which also handles the interpolation of missing values.

In the final step, the Sensor Enricher uses trigger mechanisms to include domain knowledge into the feature vectors. Domain knowledge is included as a separate dimension in the feature vector and can be calculated by using feature extractors or by implementing a JavaScript interface.

3.1.2 ONLINE ANALYTICS AND STREAMSTORY

The Online Analytics component is the third component in the upper branch of the real-time processing layer and lies between the CEP engine and Online Decision-Making components (implemented in the scopes of WP3 and WP4). The component consumes the output of the Sensor Enricher and is responsible for predictions, anomaly and activity detection. After consuming and processing the incoming data stream, the online analytics component also acts as an adapter for the StreamStory component to the real-time processing layer. Its architecture is shown in Figure 4.

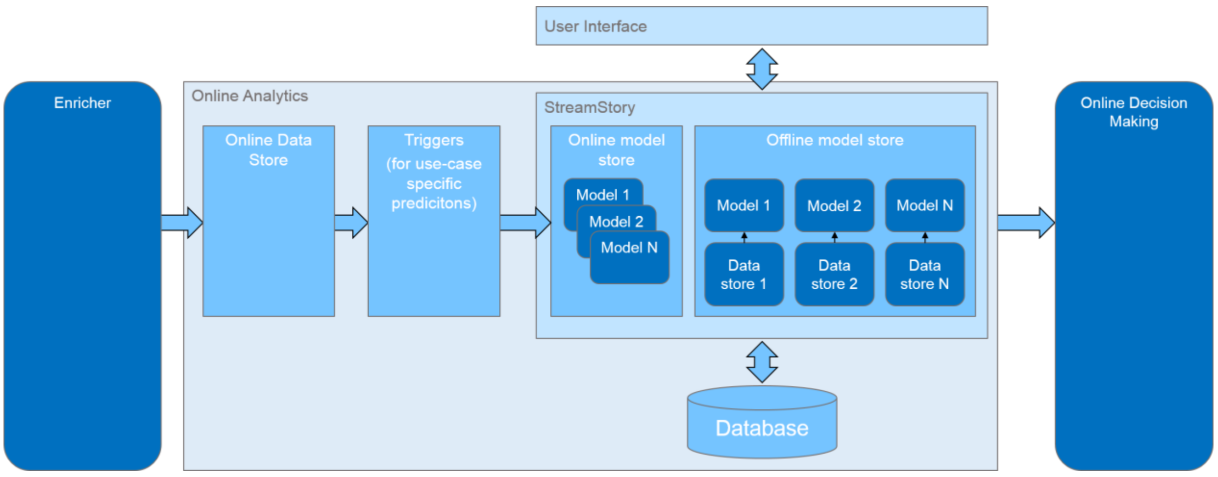


Figure 4: The high level conceptual architecture of the Online Analytics component.

The incoming data stream is first saved into an input store, storing a sliding window of the enriched data stream. Triggers are attached to the input store allowing for use-case specific predictions such as prediction of scrap parts for the Hella Saturnus use case or detecting anomalies in the friction coefficients for the MHWirth use case. These triggers are attached at design time and are implemented as JavaScript callbacks. They can make use of the sliding window of records stored in the input store and the analytics algorithms supported by QMiner. Finally, the data is resampled and routed to the StreamStory component, which offers generic and configurable prediction, anomaly and activity detection services and will be presented in the following sub-section.

3.1.3 STREAMSTORY

The StreamStory component is the main component developed in tasks T2.3 and T2.4 and is the implementation of the multi-scale methodology first proposed in deliverable D2.3.1. StreamStory models the data stream as a hierarchical Markovian process associating typical behaviour of the monitored system with conceptual states and transitions.

This representation is constructed using the following pipeline. First, it captures the structure of the data by employing unsupervised machine learning techniques to identify a systems' most typical states. Next, it captures the dynamics by modelling transitions between these states using a Markov chain framework. Finally, a hierarchical representation is constructed by aggregating both states and transitions. Appropriate levels of the hierarchy are then merged together and associated with unique scales. The final result is shown in a web-based user interface where users can interact with the hierarchical representation and configure it to output events in the form of predictions, anomalies and activities.

StreamStory's conceptual architecture is shown in Figure 5.

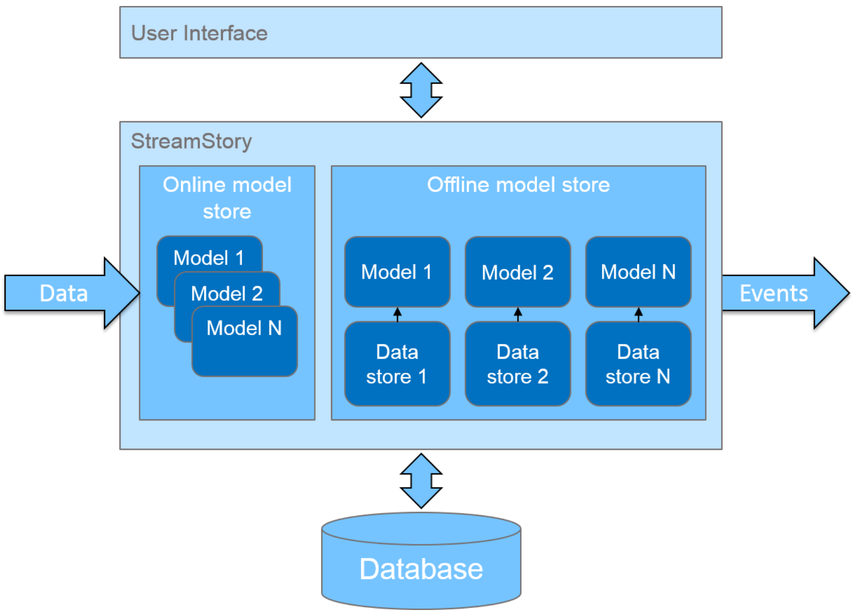


Figure 5: The conceptual architecture of the StreamStory component.

StreamStory operates in two models: offline and online. The offline mode can be used to explore any dataset and interact with its hierarchical representation before making any design-time changes to the schema of the real-time data. When used in online mode, the model is applied to the real-time data and can be configured to produce output events in the form of predictions, anomalies and activities. Online models, however, use a design time schema of the data.

To support the two modes, StreamStory manages its models in two separate stores. The first is the online model store, where all the models use the same data store, defining the schema of the data. Therefore, any attributes used to construct online models must be included in the schema. In contrast, each model in the offline model store uses its own unique data store instantiated before the model construction begins and can therefore include any attribute.

After authenticating, users can use the user interface shown on top of Figure 5 to manage their models through the dashboard, or interact with and configure their models through the model visualization page. We note that at the time of writing, StreamStory uses its own authentication mechanism, which will be replaced with a common ProaSense authentication mechanism as part of integration in the scope of WP6.

The database on the bottom of Figure 5 is used to store the configuration used to construct the models, store the models themselves and associate users with models through an access control list (ACL).

Overall, the component implements two sub-tasks of task T2.3: (a) machine learning algorithms supporting supervised and unsupervised scenarios and (b) algorithms for explaining the hidden causal structure in the data, and three sub-tasks of task T2.4: (a) unsupervised anomaly detection for unusual events and trends in the data, (b) prediction to extrapolate the development of data in the future and (c) interpretation of the stream structure to explain the hidden structure in the data. Additionally, StreamStory supports semi-automatic activity detection, where users can define activities as sequences of (sets of) states, which are then matched to the incoming data stream.

The architecture of a single StreamStory model is shown in Figure 6.

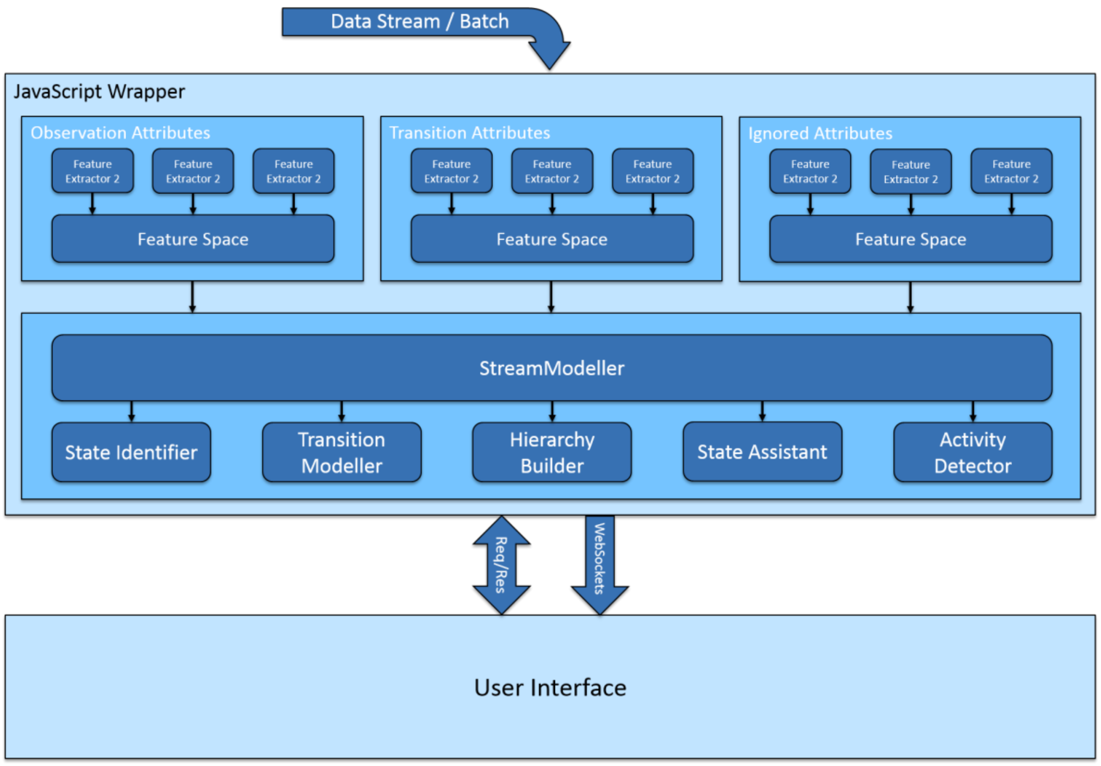


Figure 6: The architecture of a single StreamStory model. Each model consists of three feature spaces, each responsible for extracting and preprocessing a distinct set of attributes from the feature vectors, and the Stream Modeller which is the main component of the model.

A StreamStory model is implemented as a JavaScript wrapper around four components: three feature spaces and the Stream Modeller. The three feature spaces are responsible for selecting appropriate attributes from the data and the transformation of the raw data stream into (and out of) appropriate feature vectors, later used to build and maintain the hierarchical model.

Each feature space consists of several feature extractors, presented at the beginning of Section 3, responsible for the transformation of a single field. StreamStory supports two types of feature extractors:

- **Numerical:** consumes a numeric field and normalizes it and
- **Categorical:** consumes a field, which can assume a finite set of *n* values and outputs a vector of dimension *n* with a single entry set to 1, corresponding to the value of the field.

The output of the first feature space is used in the first step of the methodology to construct the states of a StreamStory model and assign the observations of the incoming data stream to a state in the model in real-time. The second feature space is used to model transition rates between the states. As mentioned later in Section 3.1.3.2, StreamStory supports modelling transitions based on the attributes used to construct the model. This feature can be used as a simulation tool to model the behaviour of the system under alternate configurations (for instance: "How would the dynamics change is we increase the cooling temperature?"). The second release of the StreamStory component adds a third feature space which outputs attributes ignored during model construction but still visualized in the user interface. This allows users to identify states or attributes correlated with some other attribute or class/label of interest. For example, if users wish to investigate attributes correlated with a higher scrap rate, then the following procedure can be followed:

1. First, they configure the scrap rate attribute as ignored
2. Next, construct a model using the attributes under investigation
3. Find states with high scrap rates at the appropriate scales and
4. Investigate the distributions of the investigated attributes in those states compared to other states with normal or low scrap rates.

The fourth component is the Stream Modeller, which is the core component of a StreamStory model. As mentioned in deliverable D2.3.1, this component is responsible for all the logic, including state and hierarchy construction, identification of the current state from the real-time data stream, prediction, anomaly detection, etc. and delegates these tasks to subcomponents. In addition to the four sub-components presented in D2.3.1, the second release adds a fifth component: Activity Detector, responsible for activity detection. The five subcomponents will be presented in the following subsections.

3.1.3.1 STATE IDENTIFIER

The first of the subcomponents is the State Identifier. When constructing a StreamStory model, the state identifier is responsible for the construction of the lowest-scale states and computing, and storing their statistics for later use in the user interface and anomaly detection. The states are identified by clustering the feature vectors extracted from the batch of data using one of the two available clustering algorithms presented in deliverable D2.3.1.

The second release allows the inclusion of time-based features in the lowest-scale state construction phase. This aims to capture time-based recurrent behaviour and is achieved by appending features constructed from the timestamps into the feature vectors used by the clustering algorithm. Once extracted, the time-based features are encoded as a categorical feature vector with categories depending on the time unit users configured in the configuration form. The following list describes the feature vectors for each time unit available.

- **Second**: the feature vector is encoded as a six-dimensional vector, each dimension representing 10 seconds in a minute. The value of the appropriate dimension is set to 1, while the other dimensions are set to 0. For instance, if the time of an observation is 7:43:55, the resulting feature vector becomes *(0,0,0,0,0,1)*.
- **Minute**: like with the previous time unit, the feature vector is encoded as a six dimensional vector, each dimension representing ten consecutive minutes in an hour.
- **Hour**: the feature vector is encoded with 24 dimensions, each dimension representing a single hour of day.
- **Day**: the feature vector is encoded with seven dimensions, each dimension representing the day of week.
- **Month**: the feature vector is encoded with twelve dimensions, each dimension representing the month of the year.

In the online mode, the State Identifier is responsible for identifying the current lowest-scale state from the incoming data stream by assigning the observations to the nearest centroid. For further details, we refer the reader to deliverable D2.3.1.

3.1.3.2 TRANSITION MODELLER

The second subcomponent, the Transition Modeller, models the dynamics in the data stream as transitions between states using the continuous time Markov chain framework. When constructing a model, the Transition Modeller associates each state constructed in the first step (e.g. lowest-scale state) with a unique state of the Markov chain and constructs a transition rate matrix used to represent the Markov chain. When interacting with the model, the Transition Modeller then aggregates the states of the Markov chain to compute the Markov chain corresponding to appropriate scales in the hierarchy.

Cox's proportional hazards model, presented in deliverable D2.3.1, used to construct the transition rate matrix representing the Markov chain, was found inaccurate, particularly for longer lasting states with high variation in the attributes, and was replaced by a matrix of nominal logistic regression models explained next.

Recall from deliverable D2.3.1 that all the data needed to describe a continuous time Markov chain can be represented using a transition rate matrix Q where each non-diagonal element represents the rate of jumping from state *i* to state *j* (in number of jumps per time unit) while the diagonal elements represent the rate of leaving state *i*. Since the jump process form state *i* to state *j* of a continuous time Markov chain can be characterized as a Poisson process, we can model its transition rate q_{ij} as a function of pre-selected attributes $q_{ij} = q_{ij}(x)$. We do this by estimating the transition rates as:

$$q_{ij}(x_k) = \frac{\Delta t}{\tilde{p}_{ij}(x_k)}$$

Where Δt is the resampling interval configured in the Online Analytics component and $\tilde{p}_{ij}(x_k)$ is the modelled probability of the process jumping from state *i* to state *j* in a single interval. Now suppose that the process is currently in state *i* and define the random variable:

$$J_i = j \Leftrightarrow X_{t+\Delta t} = j$$

J_i has a multinomial distribution with parameters $(p_{i1}(x_k), p_{i2}(x_k), \dots, p_{in}(x_k))$ which can be modelled using a nominal logistic regression model to estimate $\tilde{p}_{ij}(x_k)$.

To model the dynamics hierarchically, a separate Markov chain must be computed for each scale in the hierarchy. As presented in deliverable D2.3.1, the transition modeller uses the following formula to calculate the transition rate matrix at a specified scale of the hierarchy:

$$\Psi_l(Q) = \left(P'_{\Psi_l}\Pi \mathrm{P}_{\Psi_1}\right)^{-1} P'_{\Psi_l}\Pi \mathrm{Q} \mathrm{P}_{\Psi_1}$$

where $\Pi = diag(\pi)$, π is the stationary distribution of the initial Markov chain and P_{Ψ_1} is a *(n x m)* matrix describing which states are aggregated on that specific scale.

One of the requirements of task T2.4 is to predict undesired events (states) and provide a probability density function (pdf) of their expected occurrence time. As described in deliverable D2.3.1, StreamStory offers a generic prediction service where users can identify states corresponding to undesired events through the user interface, which the component then continuously monitors and outputs probabilities of arrival along with a pdf of the expected arrival time. To estimate the probability of arriving to state *j* from state *i* and the pdf, StreamStory solves the following differential equation:

$$pdf(0) = \delta_{ij}$$

$$pdf(t) = p_{ij}(t) - \int_0^t pdf(s) p_{jj}(t-s) dt$$

This pdf is then sent to the Online Decision Making component in the form of a histogram.

The second release exploits the Markov assumption to optimize the calculation of the pdf. Using the semi-group property of the Markov chain, we can estimate the transition probabilities at the time of the first bin as $P(t_1) = (I + \epsilon Q)^{t_1/\epsilon}$ which can be computed efficiently in $O\left(\log(\frac{t_1}{\epsilon})\right)$ time using the binary algorithm. We then only need to calculate the pdf at the other bins as:

$$pdf(t_n) = p_{ij}(t_n) - \sum_{k=0}^{n-1} pdf(t_{k-1}) * p_{jj}(t_{n-k})$$

where $p_{ij}(t_n) = (P(t_1)^n)_{ij}$ which can be updated in each iteration of the procedure and $p_{jj}(t_{n-k})$ are the probabilities of returning to state *j* which are stored in a list. Using this optimization, we were able to achieve a significant performance boost allowing for prediction of many more undesired states in real-time.

3.1.3.3 HIERARCHY BUILDER

Once the lowest scale states and Markov chain are constructed, it is the responsibility of the hierarchy builder to arrange them into a hierarchy. The hierarchy builders' input is a set of state centroids and the transition rate matrix used to describe the lowest scale Markov chain. From these inputs, it constructs a hierarchy of states and associates each level of the hierarchy with a specific scale used in the final model.

The hierarchy builder supports two methods to construct the hierarchical representation. The first approach uses agglomerative clustering by first constructing a pairwise distance matrix, with each element representing the Euclidean distance between the corresponding state centroids, and iteratively merging elements with the minimal distance. For further details, we refer the reader to deliverable D2.3.1.

The second method for constructing the hierarchy represents a top-down transition-based approach. It starts by assigning all the states into a single "super state" representing the coarsest possible scale. It then splits this state in two by approximating the *min-cut* problem in the graph representing the Markov chain. The method then works iteratively, splitting the "best" state until we have the initial Markov chain. The splitting procedure is based on spectral *bi-partitioning* of Markov chains using the sign structure of the eigenvector corresponding to the second largest eigenvalue of the *symmetrized* transition rate matrix Q_s. Indeed, the transition rate matrix of a continuous time Markov chain Q is precisely the negative Laplacian of the corresponding directed graph. Q in general is not symmetric and therefore may have complex eigenvalues. Thus, Q is first symmetrized using the following formula:

$$Q_s = \frac{1}{2}(\Pi Q + Q'\Pi)$$

where $\Pi = diag(\pi)$ is a diagonal matrix with the stationary distribution on the diagonal. This symmetrisation step ensures that all the eigenvalues (and eigenvectors) are real-valued and furthermore, matrix $\Pi^{-1}Q$ preserves the ergodic properties of Q.

The bi-partition function is then obtained by solving the following generalized eigenvector problem:

$$Q_s v = \lambda_2 \Pi v$$

And assigning states based on the sign structure of the generalized eigenvector *v*.

The iterative algorithm begins by assigning all the states of the original Markov chain to a single "super state" or partition. On the *m*-th iteration, it then assumes that *m* splits have already been made and selects which of the *m* partitions to split. Let $Q^{(i)}$ denote a transition rate matrix of the Markov chain induced on the states in the *i*-th partition. When bi-partitioning $Q^{(i)}$, we obtain *m+1* partitions inducing a Markov chain $Q_{m+1}^{(i)}$ on *m+1* states. The goal of the procedure is then to select the partition to split on each iteration. It does so by minimizing the relative entropy rate between the original Markov chain Q and $Q_{m+1}^{(i)}$:

$$i = \underset{i \in \mathbb{N}_m}{\operatorname{argmin}} D(Q || Q_{m+1}^{(i)})$$

Intuitively, the relative entropy rate provides a measure of "distance" between two Markov chains and is defined as:

$$D\left(Q || Q_{m+1}^{(i)}\right) = \sum_{\substack{i,j \in I \\ i \neq j}} \pi_i q_{ij} log \frac{q_{ij}}{\tilde{q}_{ij}} + \sum_{i \in I} \pi_i (q_{ii} - \tilde{q}_{ii})$$

However, this formula assumes that the two chains are defined on the same state space. Since this is not the case, we cannot use the above equation directly. We therefore define ϕ as the partition function mapping states of Q onto states of $Q_{m+1}^{(i)}$ and $\psi = \phi^{-1} \circ \phi$. Therefore, $\psi(i) \subseteq I$ represents the set of states aggregated into a single state. Using this notation, we can define the relative entropy rate as:

$$D\left(Q || Q_{m+1}^{(i)}\right) = \sum_{\substack{i,j \in I \\ \phi(i) \neq \phi(j)}} \pi_i q_{ij} log \frac{q_{i\psi(j)}}{\tilde{q}_{\phi(i)\phi(j)}} + \sum_{i \in I} \pi_i (q_{i\psi(i)} - \tilde{q}_{\phi(i)\phi(i)})$$

The final step when constructing the hierarchy is to remove all the least informative levels of the hierarchy and associate the remaining with scales used in the final representation. We achieve this by grouping similar levels of the hierarchy and selecting one representative in each group in the final representation. The grouping is performed by extracting feature vectors from Markov chains on all levels, cluster them using the k-means algorithm and selecting the medoid of each cluster in the final representation. The feature vectors extracted are the singular values of the transition rate matrices and represent the spectral properties of the associated Markov chains.

To render the hierarchical model in our visualization the first release used multi-dimensional scaling to embed each state onto a plane. However, this caused many of the states to overlap making the interpretation of the model difficult. The second release addresses this by a repulsive scheme to ensure the states on the same scales do not overlap.

3.1.3.4 STATE ASSISTANT

The state assistant is responsible for assisting users in identifying the meaning of states in the qualitative visualization. It does so through various visualization services, which can suggest attributes most typical for a state, visualize quantitative properties of states and visualize the states and transitions themselves. In addition to the histograms and attribute highlighting service, which highlights attributes most typical for the selected state by classifying feature vectors assigned to the selected state against feature vectors of other state, presented in deliverable D2.3.1, the second release provides other services which will be presented next.

Probability distributions and Attribute highlighting: The most basic exploratory tool to help a user is to plot the distribution of data inside each state in the form of a histogram. When clicking on a state, the histograms of all the attributes are shown on the right-side panel like in Figure 7. The second release of the component adds context to each distribution as a global distribution of that attribute in the background.

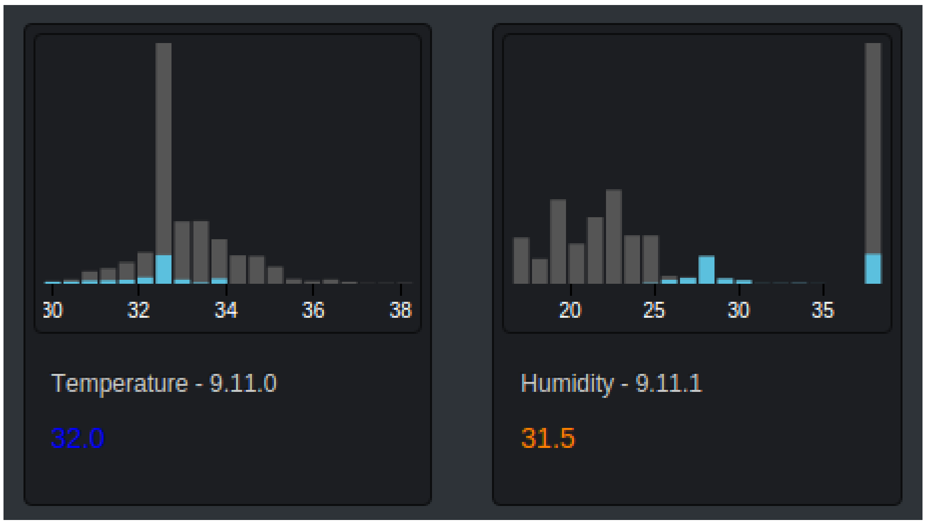


Figure 7: Histograms and attribute highlighting of a state with high humidity and low temperature observed in the Hella dataset.

To help users understand the values of attributes within a state, the attributes are coloured according to their typical values. For example, a bright orange colour indicates that the attribute typically has a higher value in this state compared to other states, while a blue value indicates this value is low compared to other states. For details about attribute highlighting, we refer the reader to deliverable D2.3.1.

Timeline histograms: Timeline histograms show users when in time the selected states appeared in the dataset used to construct the model. This can be helpful when searching for states corresponding to specific time intervals when problematic behaviour was observed or to identify the meaning of the state based on the users' time-based knowledge about the process. Additionally, timeline histograms show when certain states occur at specific time granularities including: yearly showing in which months the state occurs, monthly, weekly and daily. For example, observing that a state only occurs on Mondays or in the afternoon can give users insight into what the state represents. An example of timeline histograms is shown in Figure 14.

Automatic state labelling: Abstract states are often uninformative. While transitions between states give insight into the dynamics of the system, they fail to provide a comprehensive summary of the state. Indeed, the results of the first evaluation cycle showed that first time, non-technical users had difficulty interpreting unlabelled states. To help alleviate this, the second release of the StreamStory component provides an automatic, data-driven state labelling service. The service labels states using *<attribute, value>* pairs based on the inner-state distribution of attributes when compared to the cross-state global distribution. The attribute used in the label is selected as one of the input signals, while the displayed value is a descriptive (qualitative) level with range: lowest, low, high and highest.

Both the attribute and the level used in the label are computed by comparing the inner-state attribute distributions to the global cross-state distributions. The attribute selected by the naming process is the one with the lowest *p*-value when comparing the inner-state distribution to the global distribution. In the case of numerical attributes, the *p*-value is estimated by using the inner-state 40-th and 60-th percentile and comparing it to the global distribution. The level is then determined based on the *p*-value. When the *p*-value is below *0.12* the attribute is labelled as *lowest* or *highest*, while if the *p*-value is below *0.25* it is labelled as *low* or *high*. To determine the *p*-value of categorical attributes, the bin with the most mass is used as a candidate for the label. Bootstrapping is then employed on the inner-state distribution of the histogram and compared to the global histogram to estimate the *p*-value. An example of the labelling procedure is shown in Figure 8.

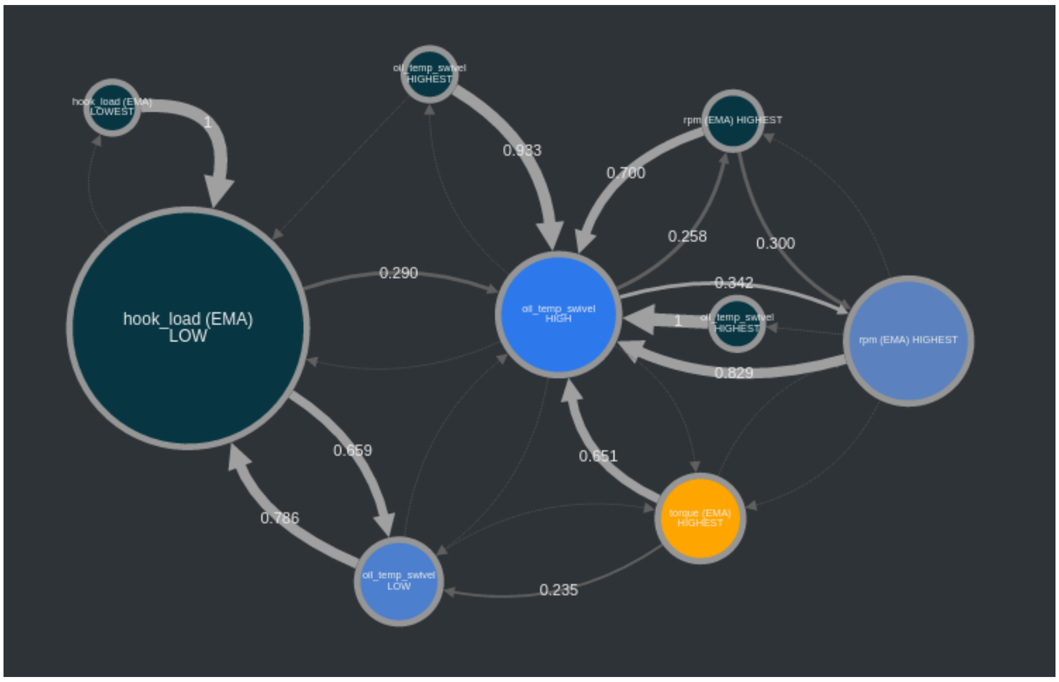


Figure 8: Example output of the automatic state labelling service on the MHWirth drilling dataset. The figure shows several states with the ones on the left representing non-drilling periods with low hook load and oil temperature, while on the right representing drilling periods with high RPM, torque and oil temperature.

State narration: The state narration service assists users by providing a qualitative description of a state in human readable form. The description comes on a form of a few short sentences describing the most typical attributes for the state and when in time this state typically occurs. The most typical attributes are extracted similarly as in the automatic state labelling service by comparing the inner state attribute distribution to the global distributions. The attributes are then ranked by the p-value and the top three shown.

The time-based description is generated by searching for peaks in the distribution of state occurrences on different time granularities. The description in the form of "The state typically occurs on Mondays between June and September" is generated if a peak with more than 70% support is found.

Decision Trees and Rule Extraction: An alternative description of states is generated through decision trees. Decision trees are classification models often used in domains such as medicine for their explanatory power and are provided here as another tool for explaining states. When a decision tree is induced, a splitting attribute and cut value are chosen recursively using design-time criteria. Users can then interpret the tree by traversing the path from the root to one of the leafs. We compute one tree for each state by classifying the observations of one state against observations of all other states, obtaining a qualitative description of the state.

We then extract rules from the decision tree in the form of $A_i > t_i \cup A_j \in (t_{j_1}, t_{j_2}]$ where A_i represent names of appropriate attributes and t_k represent thresholds. These rules are shown in the tooltip when hovering over a state.

3.1.3.5 ACTIVITY DETECTOR

The second release of the StreamStory component adds support for semi-automatic activity detection of predefined activities. Our framework defines an activity as a sequence of (sets of) states on appropriate scales, used as a template, which can be configured through the user interface (see Section 3.2.2.2).

When the model is applied on the real-time data stream, it hierarchically segments the data stream into string (or state) sequences by assigning observations to states on all scales. The segmented sequence, on appropriate scales, are then matched against the template sequences used to define the activities. If a match is found, an activity event is generated, describing the time interval when the activity occurred and the activities' identifier, a message is sent to other components of the ProaSense system through the broker. A symbolic example of such a matching is shown in Figure 9.

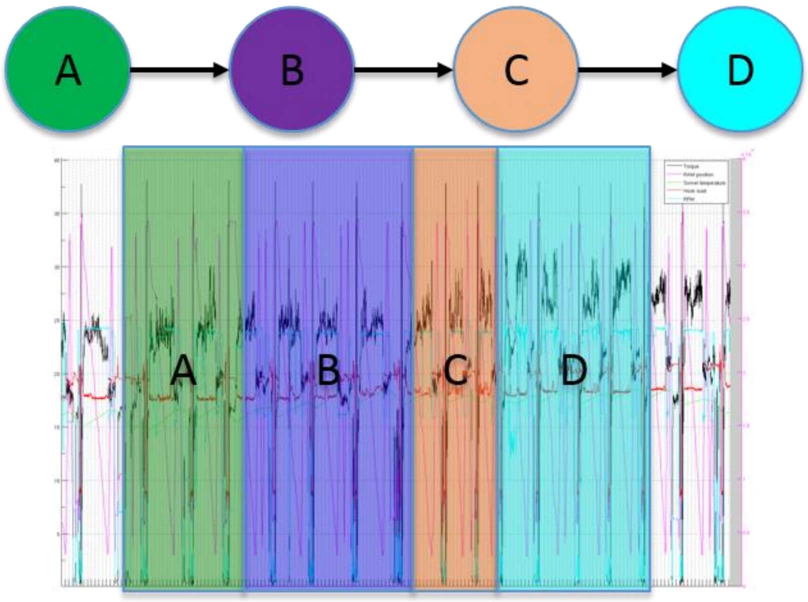


Figure 9: A symbolic example of an activity as a sequence of states (on top) matched against the MHWirth drilling dataset (on the bottom). In this example, we can see the data stream transitioning through the whole template sequence indicating a match and signalling the relevant activity.

At the time of writing, a match between observed and template sequences is reported when an exact match is found proving sufficient for the slips detection use-case presented in Section 4.2. Several extensions are possible however, including the use of partial ordering, various string distance metrics and/or dynamic time warping.

3.2 USER INTERFACE

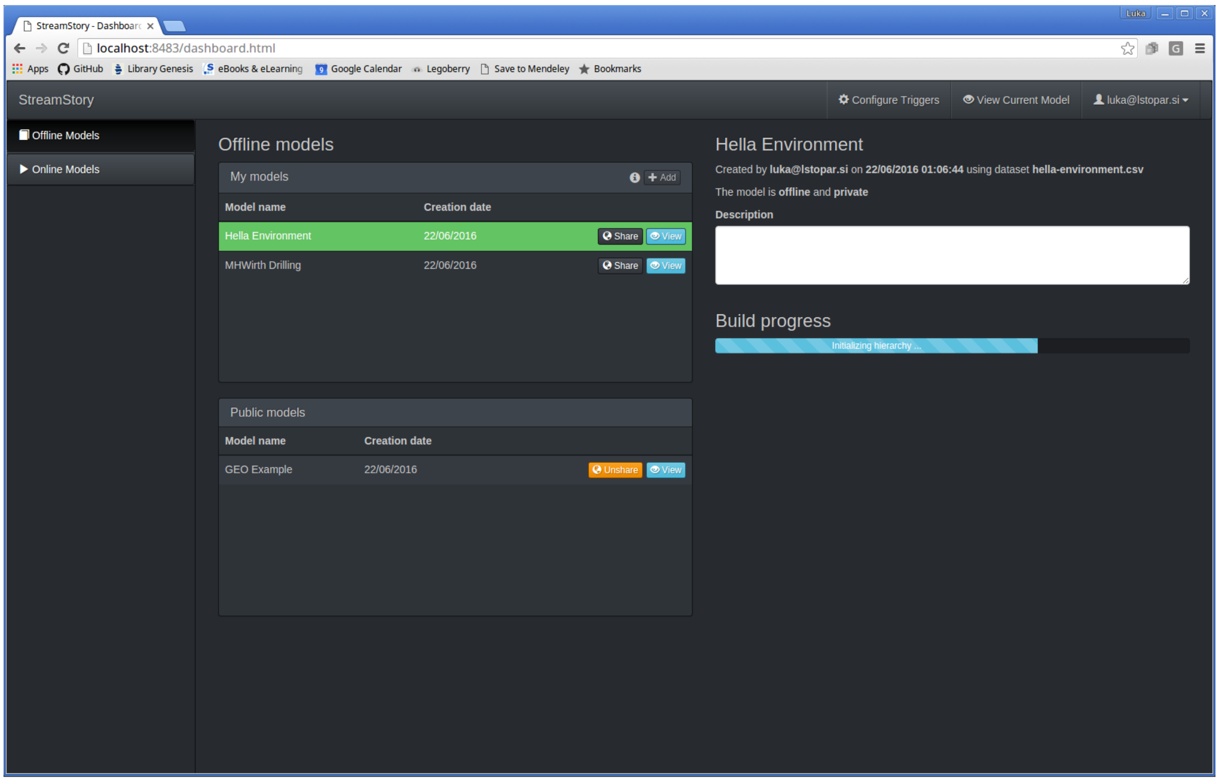
This section presents the user interface of the components developed in tasks T2.3 and T2.4. The user interface is based on the StreamStory component and designed to allow for management of and interaction with its hierarchical models.

After authenticating, users are presented with a dashboard, presented in Section 3.2.1, where they can create and manage their models, profile and configure specific triggers in the Online Analytics component. We note that at the time of writing, StreamStory uses its own authentication mechanism, which will be replaced with the common ProaSense login mechanism as part of WP6.

Users can interact with specific models on the main visualization page. This page is designed to allow users to explore interact with the hierarchical representation of the model as well as configure output events in the form of activities and predictions. The main visualization page is presented in Section 3.2.2.

3.2.1 DASHBOARD

Once authenticated, StreamStory users are presented with a dashboard shown in Figure 10.



Model name	Creation date	
Hella Environment	22/06/2016	Share View
MHWirth Drilling	22/06/2016	Share View

Model name	Creation date	
GEO Example	22/06/2016	Unshare View

Created by **luka@lstopar.si** on **22/06/2016 01:06:44** using dataset **hella-environment.csv**

The model is **offline** and **private**

Figure 10: A screenshot of StreamStory's dashboard. The dashboard contains two tabs where users can construct and manage offline and online models.

The dashboard consists of two main tabs: the “Offline Models” and “Online Models”, the profile options and trigger options, where users can configure specific triggers in the Online Analytics component.

The “Offline Models” tab consists of two tables, in the centre, and a model details panel on the right hand side of Figure 10. The upper table shows the user’s private offline models. To add a new model, they click the add button in the top-right corner of the table and are presented with a configuration form, where they can configure specific choices to create the model. When clicking an entry in the table, the details along with a description of the corresponding model are shown on the right panel. Each model has two buttons. If a user wishes to interact with a specific model, they can click the “View” button of the appropriate row in the table. By clicking the “Share” button, the model is moved into the bottom table and becomes available to all users. To remove a model, users can use the context menu by right-clicking the appropriate entry in the table.

Like the "Offline Models" tab, the "Online Models" tab consists of two tables in the centre and a details panel on the right. The upper table contains active online models currently being applied to the real-time data stream. Like in the "Offline Models" tab, they can add a model by clicking on the add button in the upper-right corner of the table. The "Share" button in each entry of the table is replaced with a "Deactivate" button. Once clicked, the corresponding model is moved to the bottom table and is deactivated (the model no longer processes the data stream and stops producing events). A model can be viewed and removed the same way as in the "Offline Models" tab.

3.2.2 MAIN VISUALIZATION PAGE

Once a model is selected or constructed on the dashboard page, users are redirected to the main visualization page shown in Figure 11.

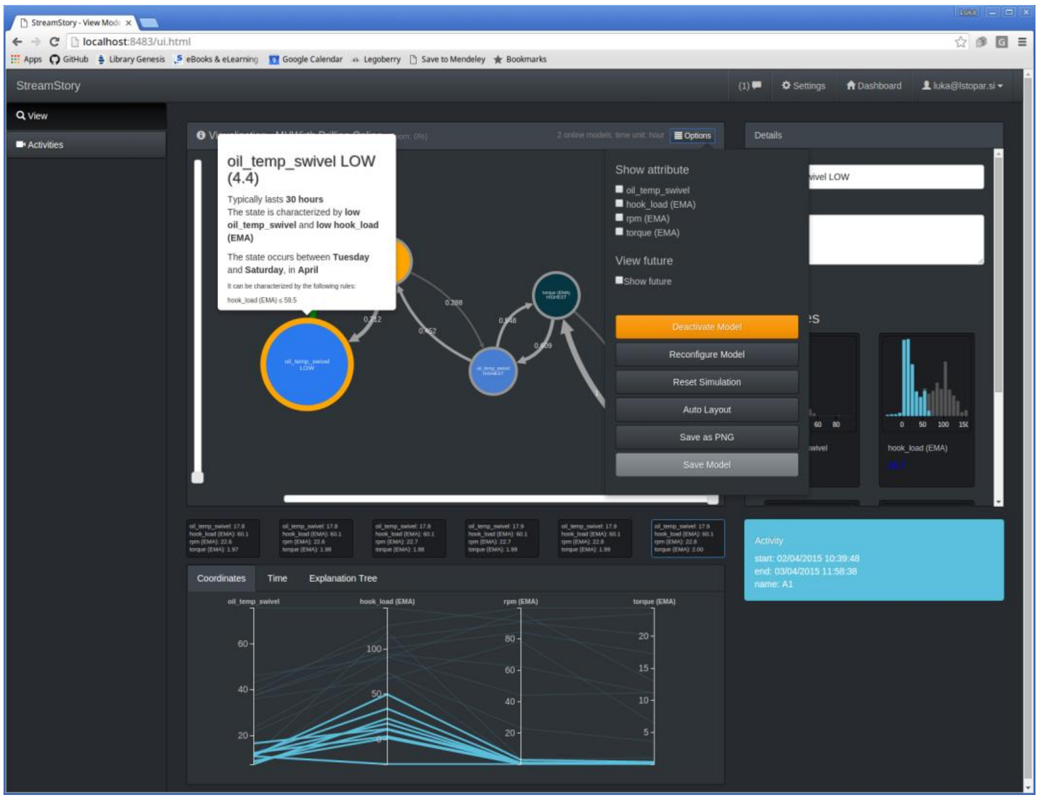


Figure 11: A screenshot of the main visualization page showing an online model constructed from the MHWirth dataset.

The main visualization page is designed to allow users to interact with our hierarchical model and configure it to produce events. It consists of two main tabs: "View tab" and "Activities" tab. When exploring offline models, parts of the page are hidden, including:

- The "Activities" tab,
- The "Undesired state" configuration,
- The messages icon, in the navigation bar, and panel in the bottom right corner,
- The central vertical strip representing the latest values processed by the component.

The two tabs will be presented in detail the following subsections.

3.2.2.1 VIEW TAB

The view tab is our main visualization. It displays the hierarchical model using a four panel user interface, an example of which is shown in Figure 11.

The visualization panel in the centre visualizes the model at the current scale as a graph. A state is represented with a circle whose size represents the time/probability spent in the state. This is computed from the stationary distribution of the Markov chain (at the currently shown scale). The labels on states initially show a qualitative description of the state in the form of <*attribute,level*> pairs computed by the automatic labelling service presented in Section 3.1.3.4. The label can be changed to reflect the user's belief about the state using the right panel presented later. The transitions are represented by arrows, where again, size represents the empirical transition probability.

When first opening the page, the model is shown in the coarsest scale with only a handful of states. Users can then traverse the hierarchical scales by either using the scroll button or the vertical scroll bar in the left of the panel. Alternatively, when right-clicking on a state, they can use the "Zoom Into" option and all other states are removed from the visualization, allowing them to explore only a subset of the hierarchy.

As seen in Figure 11, some of the states are highlighted. These are:

- **Current state:** displayed in orange, is the current state of the system, obtained by mapping the observations of the incoming data stream to states in the hierarchy. As new data arrives, the current state is updated in real-time.
- **Most likely future states:** the most likely future states are calculated from the current state using the jump matrix of the Markov chain on the appropriate scale. Their colours vary

according to the transition probability. For instance, bright blue corresponds to high transition probability while a dimmer blue colour corresponds to lower transition probability.

- **Selected state:** when clicking on a state, it becomes selected and additional visualizations appear in the bottom and right panels. The selected state is shown with a bold border.
- **Undesired state:** undesired states are continuously monitored, outputting predictions (e.g. alarms) about possible arrivals in the bottom-right panel. Undesired states are shown with a double border.

When hovering over a state, a tooltip appears showing information about the state and a description of the state. This information includes the states' label (automatically generated or user defined), its identifier consisting of the scale where it first appeared along with a consecutive number, and the average time spent in the state upon arrival. These are extracted by calculating the holding times of the underlying Markov chain. The description is the result of the state narration service and contains a short human-readable description of the state in the form of its typical attributes and typical times when the state was observed in the training set. The final piece of information in the tooltip contains quantitative rules describing the range of attributes in the state extracted from the decision trees presented in Section 3.1.3.4.

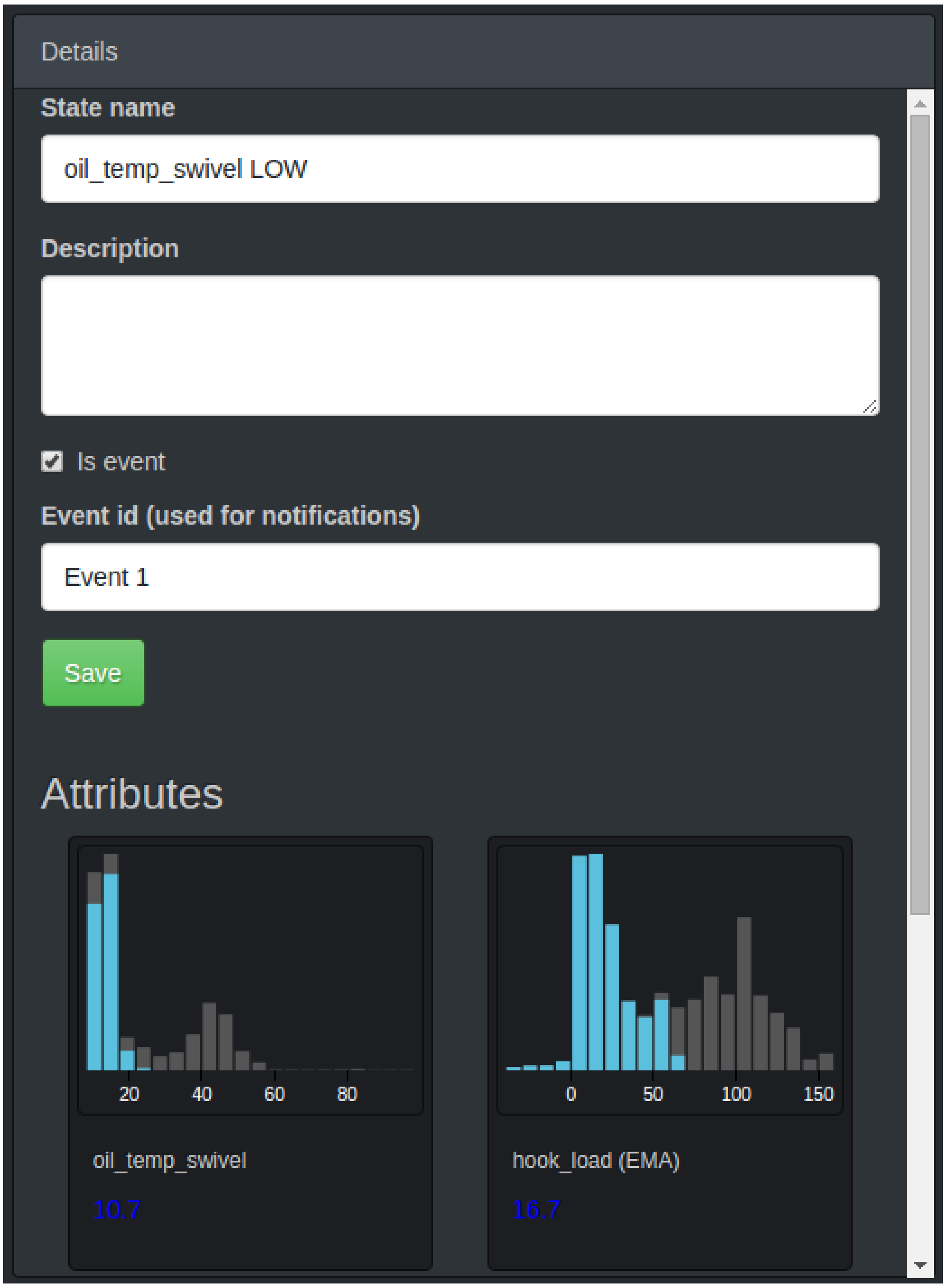
Users can use the "Options" button in the top-right corner of the visualization panel to:

- **view the distribution of specific attributes across states:** when clicking on an attribute each state is recoloured proportionally to the value of the attribute in that state. A bright orange colour indicates an attribute with a higher value in this state when comparing to other, while a bright blue colour indicates a low value.
- **observe state probabilities at future times:** when moving a slider, states get recoloured proportionally to the probability of moving to that state at a future/past time according to the slider value.
- **observe the model with modified transition configurations:** users can adjust the value of each transition attribute and observe the expected changes in the system's dynamics. When using this feature through the "Options" menu, the attribute is adjusted for all the states simultaneously. The attributes can also be adjusted for each individual lowest-scale state through the state details panel on the right of the user interface.
- **reconfigure the model:** when clicking the "Reconfigure Model" button, users are presented with the configuration form with preloaded configuration used to construct the model. They can then modify the configuration and use it to replace the existing model.

When clicking on a state, it becomes selected and additional visualizations/configurations are shown in the details panel on the right and the additional visualization panel on the bottom of the user interface. These are presented next.

The second panel is the details panel on the right of the user interface. The details panel allows users to modify the selected states' name and description and to configure it as an undesired state. When configuring a state as undesired, an additional event identifier has to be entered which is used in the output events.

Below the configuration form, the mean value and distributions of each attribute inside the state are shown as thumbnails. The distributions are shown in the form of histograms using the global distribution of the attribute as context in the background. The mean value is coloured based on the attribute highlighting service presented in deliverable D2.3.1. The details panel is shown in Figure 12.


Details

State name

oil_temp_swivel LOW

Description

Is event ✓

Event id (used for notifications)

Event 1

Figure 12: Screenshot of the state details panel.

The additional visualization panel on the bottom of the user interface shows three additional visualizations in the form of parallel coordinates, timeline histograms and state explanation tree. These are presented next.

Parallel coordinates view: an alternative tool to describe attributes inside a state uses parallel coordinates shown in Figure 13.

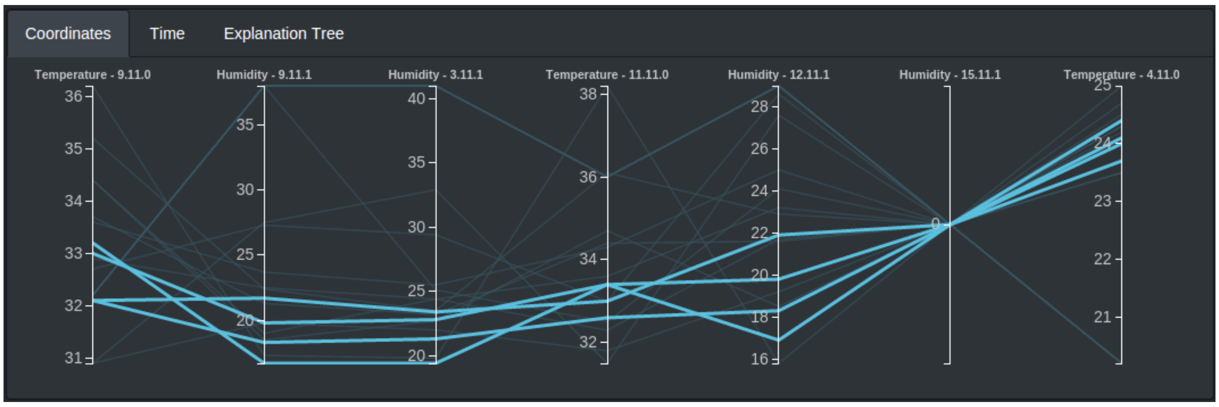


Figure 13: screenshot of the parallel coordinates view showing one of the states in the Hella environmental dataset.

Each line in the foreground shows one of the centroids, which are aggregated into the selected state by the Hierarchy Builder. As context, all the centroids are drawn in the background.

Time view: The time view, shown in Figure 14, shows when the selected state occurred in time.

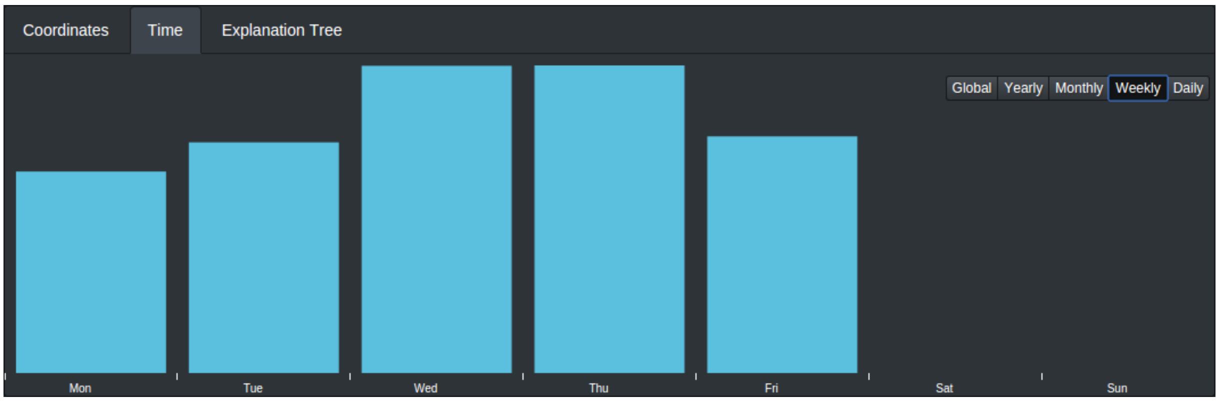


Figure 14: The time visualization widget showing a state that typically occurs on workdays.

The visualization is shown in the form of a histogram where each bin shows how many times the state was observed in the corresponding time interval. The time view consists of five sub-views: a global view shows when in time the state occurred during the entire time interval of the uploaded dataset and four views which show when the state occurred on specific time scales and can be used to identify recurrent behaviour in the dataset (for instance in Figure 14 we see a state which typically only occurs on workdays):

- **yearly:** shows how many times the state occurred in each month of the year,

- **monthly:** shows how many times the state occurred in each day of the month,
- **weekly:** shows how many times the state occurred in each day of the week and
- **daily:** shows how many times the state occurred in each hour of the day.

Explanation tree view: The explanation tree view (Figure 15) shows a description of a state by visualizing per-state decision trees described in Section 3.1.3.4.

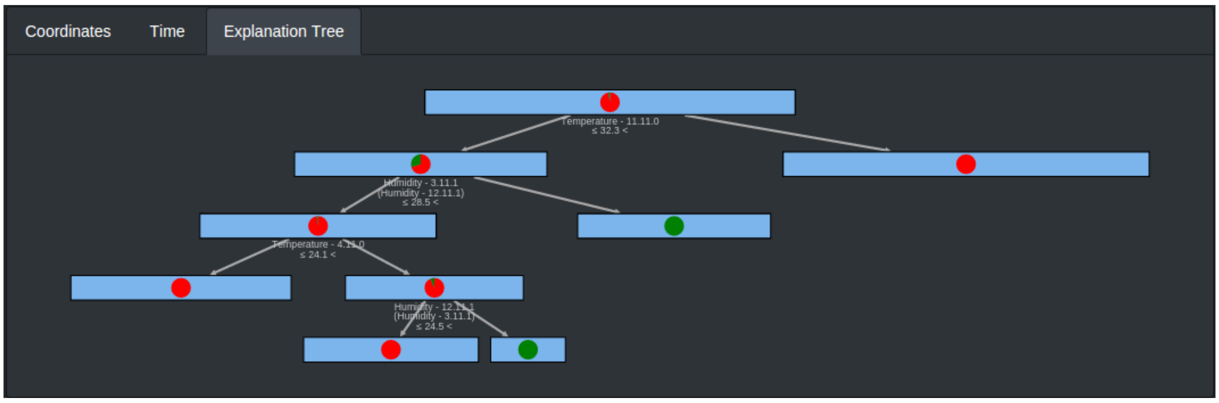
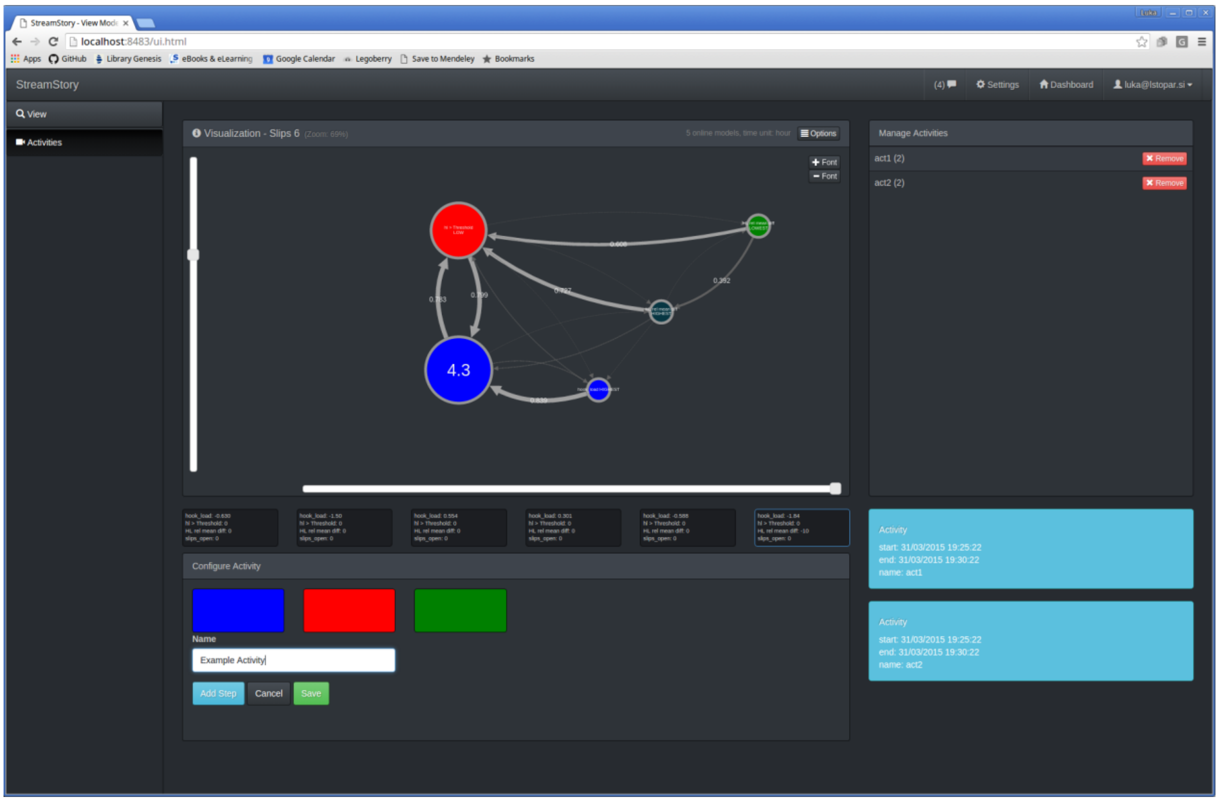


Figure 15: A decision tree used to visualize one of the states in the Hella environmental dataset.

The decision tree can be interpreted by traversing the path from the root node towards one of the leaves. Each node shows the distribution of observations from the selected state against the distribution of observations from all other states using a pie chart, where green colour represents the proportion of observations from the selected state. The width of each node is proportional to the number of observations in the node. Below each non-leaf node, there is the name of the attribute used to split the node as well as the splitting value.

3.2.2.2 ACTIVITIES TAB

The activities tab, shown in Figure 16, supports the definition and management of activity templates, which are then compared to the sequence of states observed in the monitored data stream by the Activity Detector presented in Section 3.1.3.5.



Activity
start: 31/03/2015 19:25:22
end: 31/03/2015 19:30:22
name: act1

Activity
start: 31/03/2015 19:25:22
end: 31/03/2015 19:30:22
name: act2

Figure 16: A screenshot of the activities tab. In this example, we defined a three-step activity as a sequence of three sets of states. The first step of the activity (blue) contains two states while the second and third steps contain single states.

The activities tab removes the highlighting from the states in the view tab and replaces the bottom and right panels.

To configure a new activity, users can use the “Configure Activity” panel on the bottom of the tab. Recall from Section 3.1.3.5, that our framework defines activities as sequences of (sets of) states. The steps of the sequence are represented by coloured thumbnails on top of the “Configure Activities” panel. Users can add states to the current step by right-clicking on a state and selecting “Add to Step” in the context menu. The state gets coloured using the colour of the last thumbnail. New steps are added by clicking the “Add Step” button in the bottom of the panel. Once the activity is configured, users must give it a name and click on the “Save” button.

The configured activities are shown in the “Manage Activities” panel on the right of the tab, where users can also remove any unneeded activities. Messages about the detected activities are shown in the messages panel in the bottom-right of the tab.

4. Applications and Technical Validation

This section presents the specific use cases and applications of the Online Analytics and StreamStory components presented in Section 3. Two of these scenarios are: gearbox and swivel monitoring and set point deviation, already presented in deliverable D2.3.1. After a discussion with the use-case partners, the set point deviation scenario was deemed unsatisfactory due to the high variance in the overshoots and settling times and was subsequently dropped. The gearbox and swivel monitoring scenario is implemented as a trigger in the Online Analytics component presented in Section 3.1.2.

Two additional scenarios developed in the second and third year of the project are presented in the following subsections.

4.1 PREDICTING SCRAP PARTS

The goal of this scenario is the prediction of scrap parts based on the readings from sensors installed at the Hella Saturnus use-case premises. The models presented in this section were developed offline and are executed by the Online Analytics component with use-case specific triggers. Although these models are static at the time of writing, they could be extended to the online machine learning setting using the stochastic gradient descent method.

We present two models for predicting scrap. The first model works by aggregating the readings of each sensor through the whole eight-hour shift, describing the sensors' distribution during the shift, into feature vectors and using a generalized linear model (GLM) to predict the scrap rate for the shift. The second model models the probability of each individual part being scrapped at the end of the production line. Since the response variable (scrap counts) in this case only arrives once per shift, at the end of the shift, while sensor readings arrive approximately once per minute, a special optimization was developed used to extract the linear coefficients used to model the probability of scrap.
The data used to construct the two models consists of:

- **Readings from the moulding machine**: This dataset contains readings from approximately 160 sensors installed inside the moulding machine. The sensors are all sampled at the same time when a part is being moulded, approximately once per minute.
- **Scrap reports**: The scrap reports consist of per-shift reports of scrap counts for each individual product produced during the shift. The report contains scrap counts for several types of scrap

including: dots, lines, under-moulded parts, rays, foreign objects, machine start, machine restart, etc. Some scrap types had to be dropped because they never appeared in the provided dataset.

- **Production plans**: the production plans were gathered in the form of PowerPoint presentations and contain the products being produced on each of the moulding machine in specific shifts. They were used to align the readings from the moulding machine with the scrap counts in the scrap reports.

The two models developed for this scenario are presented in the following subsections.

4.1.1 MODELLING SCRAP RATE

As mentioned above, the first of the two models predicts the scrap rate at the end of a shift by first describing the distribution of each sensor during the eight-hour shift, constructing feature vectors and using a generalized linear model in the form of:

$$\sigma(X\beta) \sim y$$

Where X represents a matrix of feature vectors extracted from each shift, y is a vector of scrap rates, β are the linear coefficients and σ is the logistic function.

The choice of this model is highly related to its computational performance. The convergence of this model was much faster which allowed us to examine which feature vectors could be used for best modelling and predicting the scrap of individual parts.

To construct the feature vectors the distribution of each sensor during the shift was constructed using the following features:

- **mean value**: the mean value of the sensor during the shift,
- **median value:** the median value of the sensor during the shift,
- **minimum and maximum:** the minimum and maximum value of the sensor during the shift,
- **absolute mean difference:** the absolute mean difference of sequential values calculated using the following formula: $\frac{1}{n}\sum_{i=2}^{n}|v_i - v_{i-1}|$,
- **variance:** the variance of the sensor readings during the shift. This feature was dropped after initial experiments as it only seemed to confuse the model and worsen the results and

- **percentiles:** a feature vector containing the 0.01th , 0.25th, 0.5th, 0.75th , 1st , 5th , 95th , 99th, 99.25th, 99.5th or 99.75th percentiles of the sensor reading during the shift.

Other features were also tested but ultimately dropped because they produced worse results.

The model was evaluated on approximately 120 shifts using leave-one-out cross validation. Since individual scrap types appear very rarely, the prediction was performed on groups of scrap (for example, predicting there will be dots, lines, rays or foreign objects). We tried to identify large groups with as many examples as possible. Our experiments showed that some types of scrap could be better predicted than others. Specifically, we found that "dots" only seemed to degrade the performance of the model and were omitted in most of the experiments.

Figure 17 shows the prediction results for various groups of scrap.

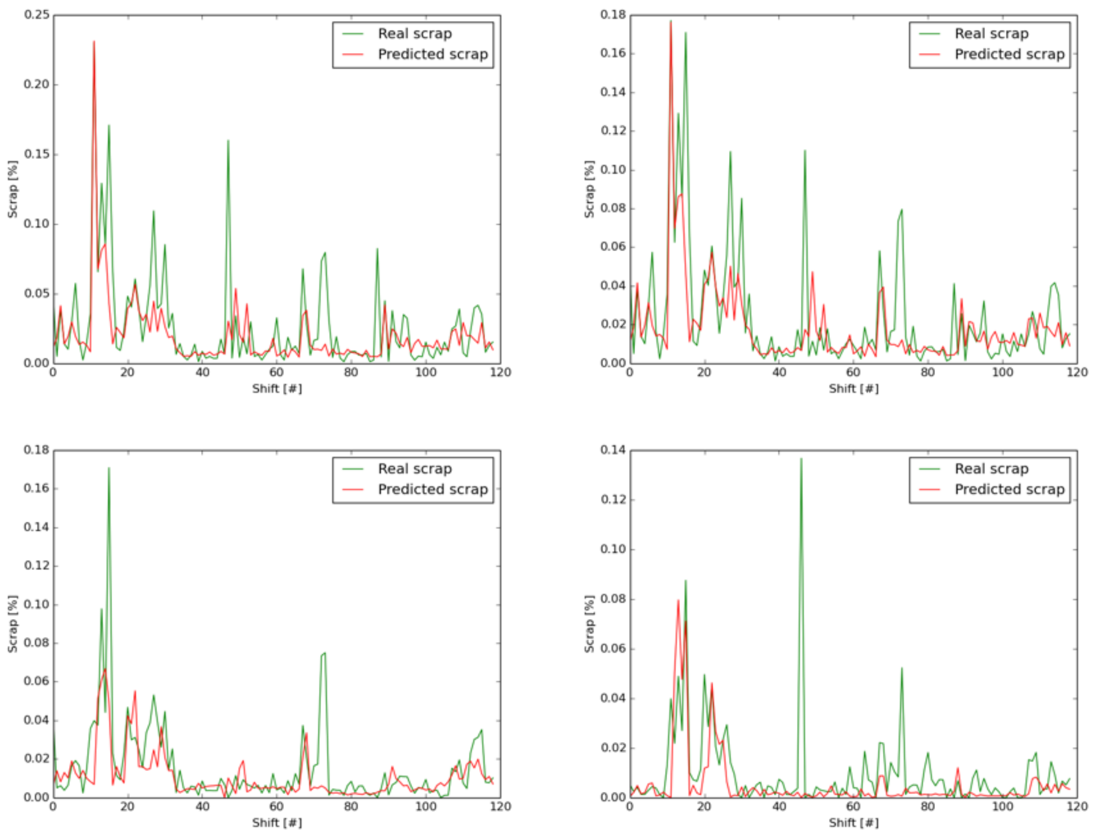


Figure 17: Results of the leave-one-out cross validation when prediction scrap rates for four different groups.

In Figure 17, we show the prediction results for four groups of scrap. The group shown in the top left corner, models all types of scrap except "dots" and produced a mean average error of 0.013 with the mean scrap rate per shift of 0.026. In addition to “dots,” the second group also excludes scrap associated with starting the machine (“zagon”). This group produces a mean average error of 0.00119 with the mean scrap rate of 0.024. The third group excludes “dots,” scrap associated with starting the machine and scrap associated with restarting the machine due to cleaning. This group produced an error of 0.0092 with a mean scrap rate of 0.0155. The final group includes “dots,” but excludes “rays,” scrap associated with foreign objects, machine restart due to cleaning and machine start. This group performed the worst with an error of 0.007 and an average scrap rate of 0.009.

4.1.2 MODELLING SCRAP PARTS

The second scenario models the probability of an individual part being scraped at the end of the production line, as soon as it leaves the moulding machine. Since the response variable was the end-of-shift scrap counts, standard machine learning algorithms could not be employed, as they require a response variable for each feature vector to model the probability of scrap. This probability was thus modelled by designing a specific optimization model and extracting linear coefficients.

The probability of a scrap part $p(x_j;\beta) = P(part\ scraped|X = x_j;\beta)$ was modelled as a logistic function of the input parameters and the linear coefficients using the following equation:

$$p(x_j;\beta) = \frac{1}{1+e^{-x_j\beta}}$$

The linear coefficients β were extracted by designing a loss function $\mathcal{L}$ and solving the following minimization problem:

$$\min_{\beta} \mathcal{L}(X;\beta)$$

In our experiments, several loss functions were developed. These are presented in Table 1.

Table 1: A table of loss funciton used to solve the minimization problem.

Loss function	Formula
Per-shift squared loss with L_2 regularization	$\sum_{i=1}^{n}\left(N_i - \sum_{j=1}^{m} p(x_j;\beta)\right)^2 + \lambda\|\beta\|_2$
Per-shift squared loss with L_1 regularization	$\sum_{i=1}^{n}\left(N_i - \sum_{j=1}^{m} p(x_j;\beta)\right)^2 + \lambda\|\beta\|_1$
Per-shift absolute loss with L_2 regularization	$\sum_{i=1}^{n}\left\|N_i - \sum_{j=1}^{m} p(x_j;\beta)\right\| + \lambda\|\beta\|_2$

Per-shift absolute loss with L_1 regularization	$\sum_{i=1}^{n}\left\|N_i - \sum_{j=1}^{m} p(x_j;\beta)\right\| + \lambda\|\beta\|_1$

To solve the minimization problem, we employed a combination of line search, momentum method and Newton's method. This proved essential, since each individual method either failed to converge to a good local minimum or had an impractically slow convergence rate. The optimization algorithm is presented below:

```
1. $\beta_0 \leftarrow select\ \beta_0\ randomly$
2. $\mathcal{L} \leftarrow loss\ function\ used\ in\ the\ optimization\ procedure$
3. $v \leftarrow 0$                          // initial momentum
4. $repeat$:
      a. $l \leftarrow \mathcal{L}(X;\beta_i)$            // calculate current loss
      b. $g \leftarrow \nabla\mathcal{L}(X;\beta_i)$           // calculate the gradient
      c. $H \leftarrow \frac{\partial^2}{\partial\beta_i\partial\beta_i}\mathcal{L}(X;\beta_i)$      // calculate the Hessian matrix
      d. $\alpha \leftarrow choose\ \alpha\ to\ minimize\ \mathcal{L}(X;\beta_i - \alpha g)$
         // select the best method
      e. $\Delta\beta = \underset{\{-\alpha g, -H^{-1}g, v\}}{\operatorname{argmin}} \{\mathcal{L}(X;\beta_i - \alpha g), \mathcal{L}(X;\beta_i - H^{-1}g), \mathcal{L}(X;\beta_i + v)\}$
         // update
      f. $\beta_{i+1} \leftarrow \beta_i + \Delta\beta_i$
      g. $v \leftarrow \gamma v - \sigma g$
5. $until\ convergence$
```

To construct feature vectors x_j, we used the results of Section 4.1.1. Specifically, we used the features identified as beneficial in Section 4.1.1 and converted them to be suitable for the real-time scenario. Thus for each sensor, the following features were constructed:

- **actual value**: the value of the observation,
- **moving average**: the mean value in a window of 15 sequential observations,
- **moving median:** the median value in a window of 15 sequential observations,
- **moving minimum:** the minimum value in a window of 15 sequential observations,
- **moving absolute mean difference:** the mean difference in a window of 15 sequential observations calculated using the following formula: $\frac{1}{15}\sum_{i=2}^{15}|v_i - v_{i-1}|$,
- **moving variance:** the variance in a window of 15 sequential observations,

- **percentile:** a categorical feature indicating whether the value is in the 0.01th, 0.25th, 0.5th, 0.75th , 1st , 5th , 95th , 99th, 99.25th, 99.5th or 99.75th percentile.

The model was evaluated using leave-one-out cross validation. In one round, the model was trained on *n-1* shifts and evaluated on the shift that was left out of the training phase. The metric used in the evaluation compares the scrap parts counted at the end of the shift to the sum of probabilities of scrap for each individual part produced in the shift and is calculated as a mean absolute difference:

$$\frac{1}{n}\sum_{i=1}^{n}\left|N_i - \sum_{j=1}^{m_i} p(x_j;\beta_i)\right|$$

Where n represents the number of shifts, N_i the actual counted scrap parts at the end of the shift, m_i the number of parts produced in shift *i* and $p(x_j;\beta_i)$ the probability of the *j*-th part being scraped calculated from coefficients β_i extracted in the *i*-th step of the validation.

Our experiments showed that the per-shift squared loss with L_2 regularization was superior to the other loss functions was superior to the other loss functions in Table 1. Indeed, the other loss functions either failed to converge to good minima, failed to converge at all, or produced worse results.

As in Section 4.1.1, we evaluated the model on several groups of scrap. Results of the two most successful groups are shown in Figure 18.

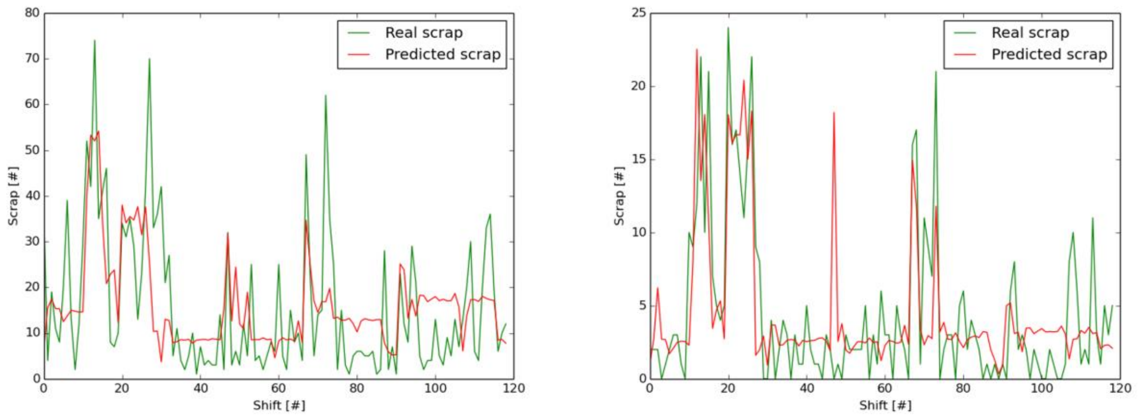


Figure 18: The results of the leave-one-out cross validation use to evaluate prediction of scrap parts.

The first group, shown on the left of the figure predicts all scrap types except "dots" and has an error of 9.61 with a mean of 15.95 parts scrapped per shift. The second group, shown on the right of the figure also excludes “single rays,” “rays” and scrap related to the machine being started and restarted due to cleaning and has an error of 2.71 with a mean of 4.47 parts scrapped per shift.

In addition to the error statistics for the scrap prediction, in Figure 18 we can see the qualitative performance of prediction. Intuitively we would like to predict when peaks occur even if we do not estimate the magnitude of the peak correctly. Most of the peaks are captured correctly, however, we do have some false positives (around shift 45 on the right graph) as well as missed peaks (around shift 110 on the right graph). These predictions only improve with additional data.

4.2 SLIPS DETECTION

MHWirth Use Case 3 is the activity and operation detection, where machine learning techniques and Markov chains are used to detect activities such as slips state and roughneck state, and to recognize operations such as drilling, tripping in and tripping out. This information will form the basis for the calculation of business oriented KPIs, e.g. the time spent in for every segment of tripping in or tripping out, or the total time for the entire tripping operations.

The first activity selected for this use case is the slips state detection. Slips are closed to hold the drill string stationary, mostly, for make-up and break-out operations. Slips are open to allow the drill string move up and down through the drill floor for example for drilling, back-reaming or tripping operations. Therefore, estimating the slips state would be a critical decision maker for stationary and non-stationary operations.

By knowing the frequency of the slips state change and the hook load, one can infer the start time, end time, and duration of make-up and break-down activities, and the rate at which the drill string is being extended or shortened. This is of particular interest for trip-in and trip-out operations. This usually happens when the drill bit needs to be replaced and the whole drill string must be first pulled out of the well and broken apart; then it is built and placed back with a new drill bit.

In the modern rigs, slips are hydraulically designed and controlled by the rig automation system through the commands that operators send. In traditional rigs, slips are manually placed in the rotary table. Slips state detection is of great value for both modern and traditional rigs. It facilitates detection of slips using rig data for traditional rigs. It also provides redundancy, reliability and robustness to operation detection in modern rigs in case of invalidity of recorded signals, e.g. due to faulty sensors or lack of feedback to the control system.

This section presents the application of StreamStory's activity detection service to the slips operation detection problem. We refer to a slips cycle as the operation where the slips start opened, stay there for a while and then transition to state closed. An example of a slips cycle is shown in Figure 19.

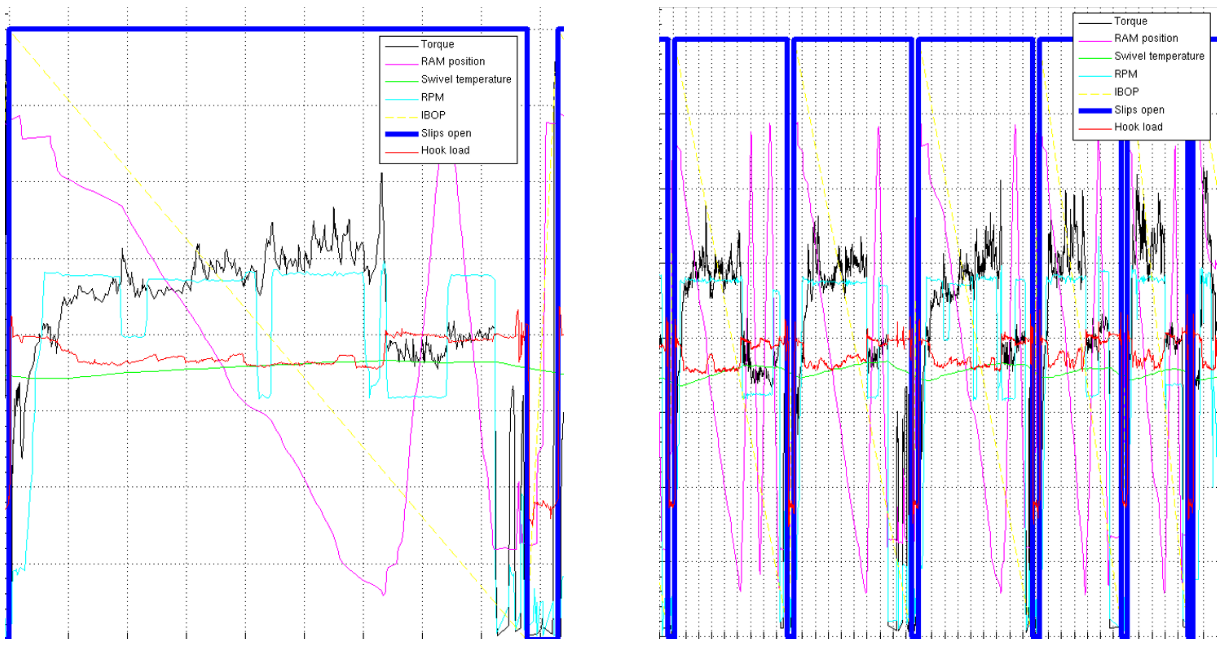


Figure 19: An example of a slips cycle (left) shown as a transition from state open to state closed of the slips signal represented by the blue line and an example of multiple slips cycles on the right.

We model the slips cycle using StreamStory's activity detection service using a model with the following attributes:

- **Hook load**: the raw value of the hook load,
- **Hook load > 12:** an artificial binary feature outputting 1 when the hook load is greater than a threshold of 12. The threshold was chosen experimentally to maximize the accuracy of the model.
- **Relative mean difference of the hook load:** the difference between the actual value of the hook load and its average value over a period of three hours.

To assist identifying states with open and closed slips, we added the "Slips Open" signal as an ignored attribute. The signal thus only appears in the visualization and does not influence the behaviour of the model. Figure 20 shows a screenshot of the model used in the experiment.

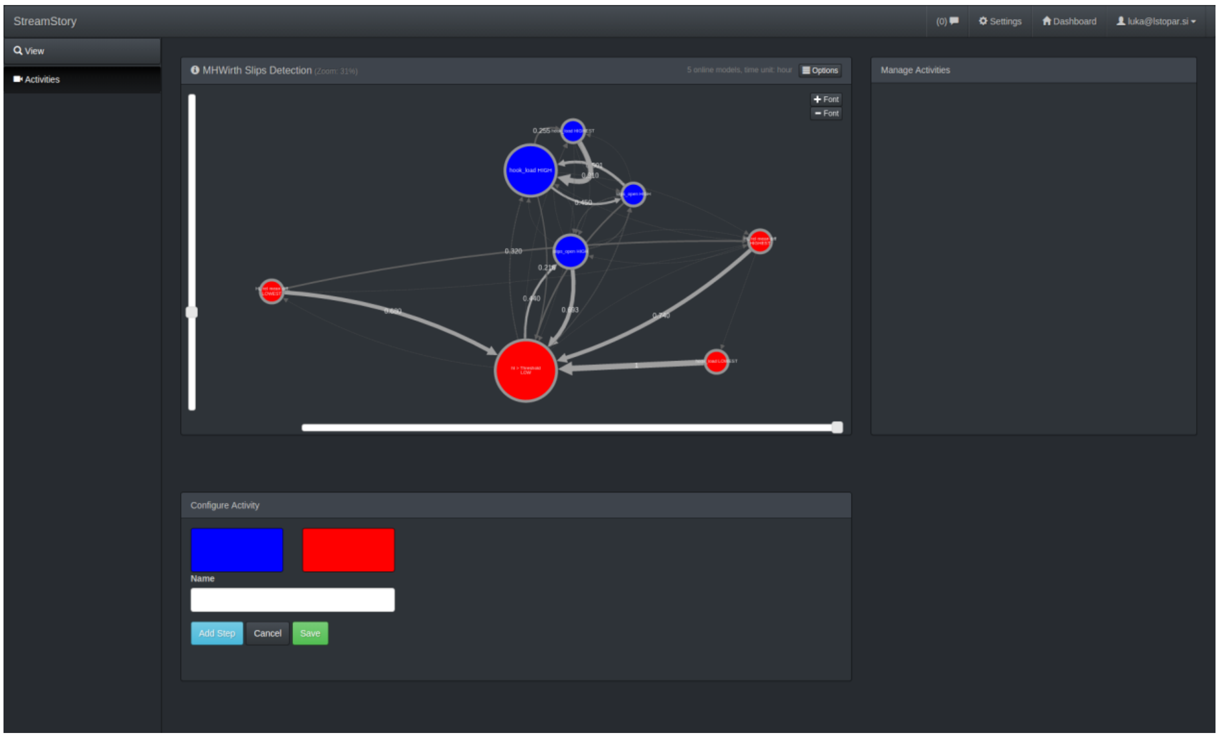


Figure 20: The model used to detect slips cycles in the slips detection scenario.

The activity sequence consists of two steps. States in the first step are coloured blue and represent states where slips are open. States in the second set are coloured red and represent states where slips are closed. Some states correspond to states where the slips are sometimes open and sometimes closed. These states were assigned experimentally.

To evaluate the model, we split the dataset into two sets: a training set and a test set. The two datasets were selected as continuous time intervals combining a total of one month of data. The data used in the test set is shown in Figure 21.

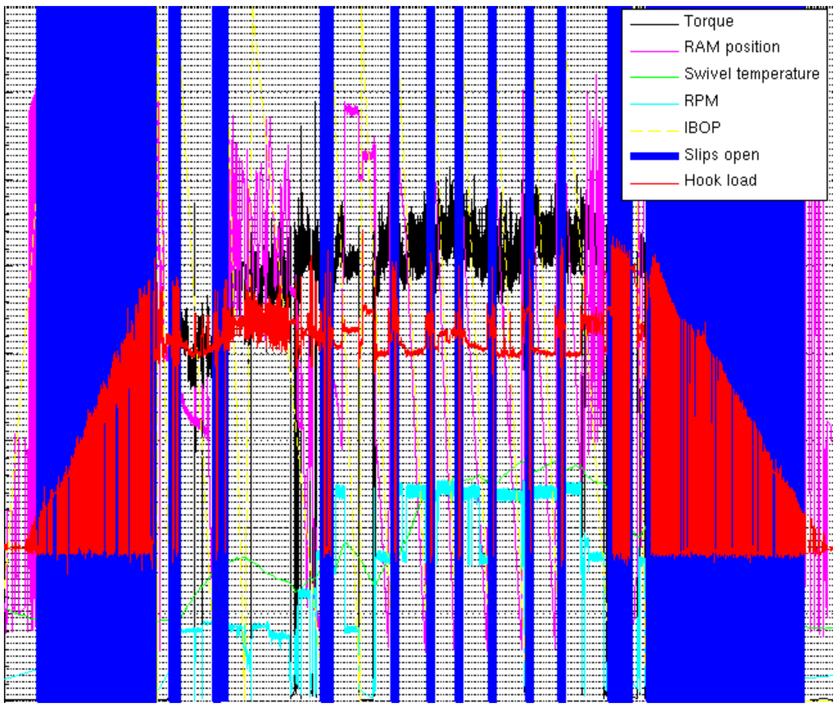


Figure 21: Screenshot of the interval used to evaluate the slips cycle detection model.

We evaluated the model by observing the behaviour of the slips signal during each activity event. The following cases were considered:

- **Single transition from open to closed**: if the signal made a single transition from open to closed, we counted a true positive (TP).
- **Single transition from closed to open**: in contrast, when the signal transitioned from closed to open, a false positive (FP) was counted.
- **No transition**: if the signal did not change states, a false positive (FP) was counted.
- **Multiple transitions**: if two transitions were observed (for example open -> closed -> open or closed -> open -> closed) we counted a false positive (FP) while if multiple slips cycles were observed in the signal, for instance n, then we counted one true positive (TP) and $(n-1)$ false negatives (FN).

In all cases, a 20 second tolerance was allowed. For instance, transitions closed (for less than 20 seconds) $\rightarrow$ open $\rightarrow$ closed were interpreted as open $\rightarrow$ closed.

Figure 22 shows the activities identified during the evaluation plotted alongside the slips signal during a trip in sequence.

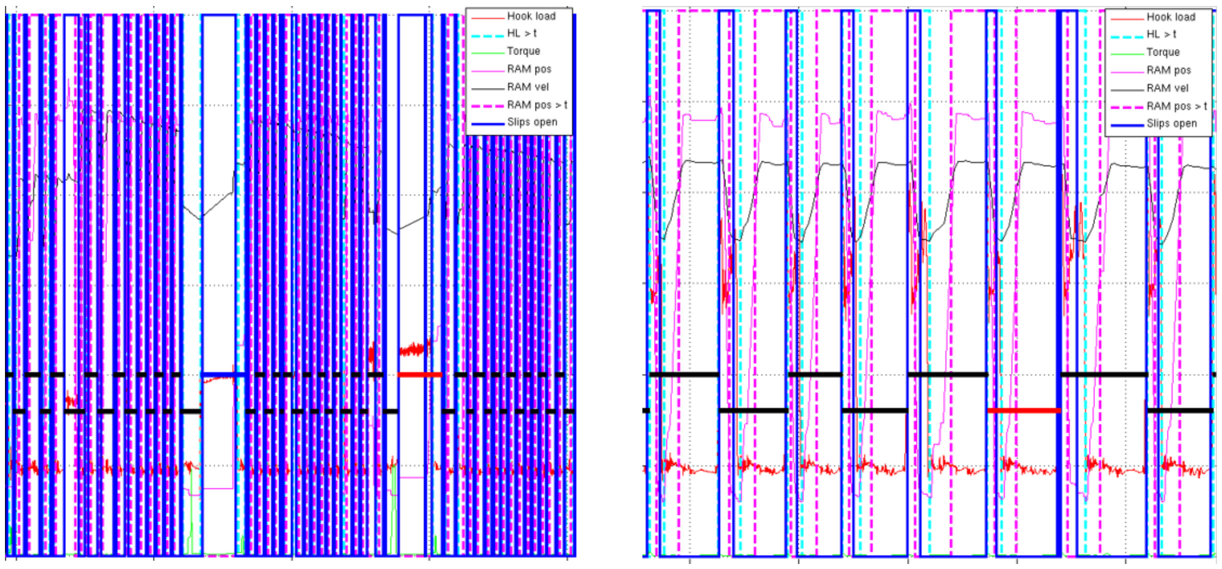


Figure 22: The detected activities plotted alongside the slips signal during a trip in sequence. To avoid overlap, the activities are presented in two rows as horizontal lines. Black lines represent the correctly identified activities (true positive), while the blue and red lines represent false positives and false negatives respectively.

The precision and recall of the model were then calculated using the following formula:

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

The model scored a precision of 98.9% and recall of 86.6% with a total of 269 detected activities.

5. 2nd Prototype Implementation

The components developed in tasks T2.3 and T2.4 were developed in two separate projects. The core functionality was implemented in C++ as part of the QMiner platform. QMiner consists of several layers including the data layer, analytics layer and JavaScript layer. Its functionality is implemented in C/C++ in the bottom layers and exposed as a JavaScript API to the Node.js platform, which acts as glue, gluing the functionalities into server-side applications.

5.1 APIS AND LIBRARIES

The server side dependencies used to develop the components in tasks T2.3 and T2.4 are the following:

- **LAPACKE**: standard C language APIs for LAPACK,
- **Async**: provides utilities for working with asynchronous JavaScript,
- **Express**: web server framework for Node.js,
- **EJS**: server-side web page rendering library,
- **Express-session**: session middleware for Express,
- **Body-parser**: body parsing middleware for Express,
- **Cookie-parser:** cookie parsing middleware for Express,
- **Multer:** Express middleware for handling form data,
- **Ws:** WebSocket implementation for Node.js,
- **Crypto**: JavaScript implementation of standard cryptographic algorithms,
- **Dateformat**: a JavaScript library for formatting dates,
- **Mathjs**: an extensive math library for JavaScript and Node.js,
- **Randomstring**: a Node.js module for generating random strings,
- **Node-mkdirp:** a Node.js library to create directories,
- **Nodemailer:** library for sending emails from Node.js,
- **Kafka-node:** Apache Kafka client for Node.js,
- **Node-mysql**: a Node.js client implementing the MySQL protocol,
- **Node-bunyan**: JSON logging library for Node.js applications,
- **Bunyan-format**: library for formatting Bunyan records,

The other set of libraries is used in the user interface, these include:

- **Bootstrap**: HTML, CSS and JavaScript framework for developing responsive projects,

- **Cytoscape.js**: graph theory library for analysis and visualization,
- **JQuery**: JavaScript library for HTML document traversal, manipulation, event handling and animation
- **D3.js**: JavaSctipt library for manipulating documents based on data,
- **qTip2**: tooltip plugin for JQuery,
- **Cytoscape.js-qtip**: Cytoscape.js extension that wraps qTip,
- **Cytocsape.js-ctxmenu**: context menu for Cytoscape.js,
- **D3.parcoords.js**: D3 extension to visualize parallel coordinates,
- **D3.tip**: tooltip extension for D3,
- **Dagre**: directed graph renderer for JavaScript,

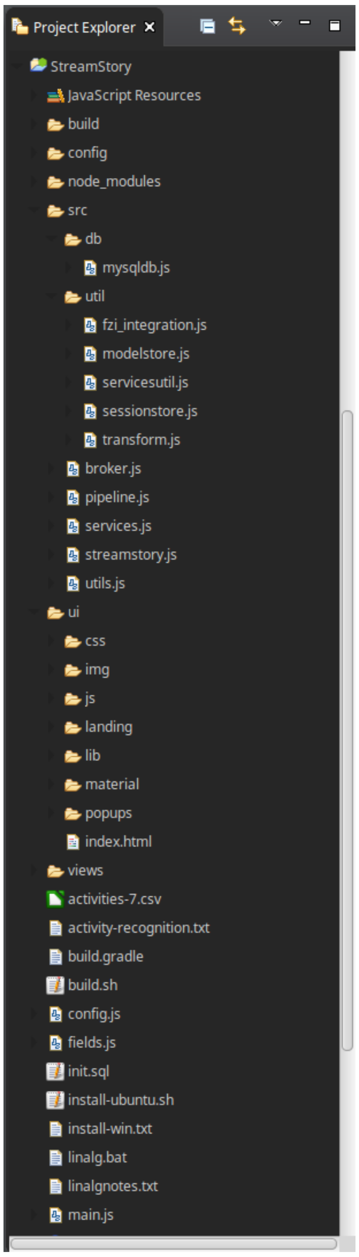
5.2 SOURCE CODE

Our source code is organized into two projects. The core functionality is included as part of the QMiner open-source project and is available in a public GIT repository[1]. Users should follow the online instructions[2] to compile QMiner with the LAPACKE library for the components to work properly. The actual components are implemented as a separate Node.js project and are available on GitHub[3].

The QMiner project is jointly developed between JSI and Quintelligence SME and is quite complex. Our contribution is spread among several files. Still much of this logic is located in folder *src/third_party/streamstory/* with the linear algebra API in files *src/glib/base/linalg* header and cpp files and some machine learning methods in *src/glib/mine/classification* and *src/glib/mine/clustering* header and cpp files.

[1] https://github.com/lstopar/qminer
[2] https://github.com/qminer/qminer/wiki/Building-OpenBLAS
[3] https://github.com/JozefStefanInstitute/StreamStory.git

The components were developed as a separate Node.js project. Figure 23 shows the layout of the project as seen through the Eclipse IDE. The execution starts in file main.js, where the configuration is read and QMiner is initialized. File pipeline.js initializes the stream processing pipeline of the Sensor Enricher and Online Analytics components and file streamstory.js initialized the StreamStory component. Once the components are running the RESTful API is initialized in file services.js. The module in file mysqldb.js is an interface to access the MySQL database and the communication with the broker is handled in broker.js. Folder *util* contains several utility modules that are used in other modules. The client side code is located in folder *ui* and the actual HTML files are located in folder *views*.



```
Project Explorer
StreamStory
  JavaScript Resources
  build
  config
  node_modules
  src
    db
      mysqldb.js
    util
      fzi_integration.js
      modelstore.js
      servicesutil.js
      sessionstore.js
      transform.js
    broker.js
    pipeline.js
    services.js
    streamstory.js
    utils.js
  ui
    css
    img
    js
    landing
    lib
    material
    popups
    index.html
  views
  activities-7.csv
  activity-recognition.txt
  build.gradle
  build.sh
  config.js
  fields.js
  init.sql
  install-ubuntu.sh
  install-win.txt
  linalg.bat
  linalgnotes.txt
  main.js
```

Figure 23: Layout of the Node.js project containing the component developed in tasks T2.3 and T2.4.

6. Conclusion

This deliverable presented the work done in tasks T2.3 and T2.4, which deal with offline and on-the-fly analysis of sensory inputs, in the final years in the ProaSense project. Based on the revised architecture presented in deliverable D1.3, we developed three components: Sensor Enricher, Online Analytics and StreamStory. Their detailed architecture, functionalities, applications, implementation and organization of source code were presented including a summary of changes compared to the first release presented in deliverable D2.3.1. The issues identified in the first evaluation cycle were summarized and the revisions included in the second release presented.