An Approach to Provide User Guidance in Special Purpose Machines and its Evaluation

Valentin Plenk*, Sascha Lang†, Florian Wogenstein‡
Institute of Information Systems at Hof University, Hof, Germany
Email: *valentin.plenk@iisys.de, †sascha.lang@iisys.de, ‡florian.wogenstein@iisys.de

***Abstract*—This paper proposes to make complex production machines more user-friendly. Improved machines help the operator in case of an error message or a process event by displaying recommendations, such as "at the last 10 occurrences of this event the operators performed the following keystrokes". The messages are generated from statistical data on former user-interaction and previous process-events. The data represents the knowledge of all the machine operators. The data is gathered by logging user-interaction and process-events during regular operation of the production machine. This approach allows to store the operators' expert knowledge in the production machine without human intervention.**

Keywords–machine-learning; human machine interfaces; special-purpose machines; production machines

I. INTRODUCTION

State of the art appliances (e.g., photocopiers) are equipped with user interfaces that help the operator fix problems (e.g., paper jam). The implementation of the software driving the interface contains structured knowledge about error scenarios and step by step instructions on how to deal with them.

While this approach leads to very well usable appliances, its proliferation is hampered by the engineering effort required for the definition of the error scenarios. This effort is only economically reasonable if it can be refinanced over a large number of appliances. In the context of production machines, particularily special purpose machines, where the usual lot size is in a range below 10 similar machines per annum [2], a different approach is needed.

To deal with that problem the authors propose to use machine-learning algorithms to generate situation-specific user guidance information from former user interactions and previous process events.

Similar applications of machine learning algorithms are commonly used to enhance user interfaces in smartphones or other IT-systems [3] [4]. In the production-machine sector, however, the application of machine learning algorithms is apparently limited to applications dealing with pattern detection in process data or distingushing different datasets [5] [6] [7] [8]. In these papers the authors use the output of the algorithms to detect errors or problems with production quality. This information is then presented to the production-machine operator requesting him to deal with the situation. While this approach undoubtedly helps to make processes more stable, it also demands ever more expertise of the operators.

Challiol et al. [9] describe an application striving to display content dependent on the users context. While addressing a completely different application domain this is similar to our approach in terms of matching context to content. The context matching algorithm proposed in [10] is quite abstract as is its performance evaluation presented in [11].

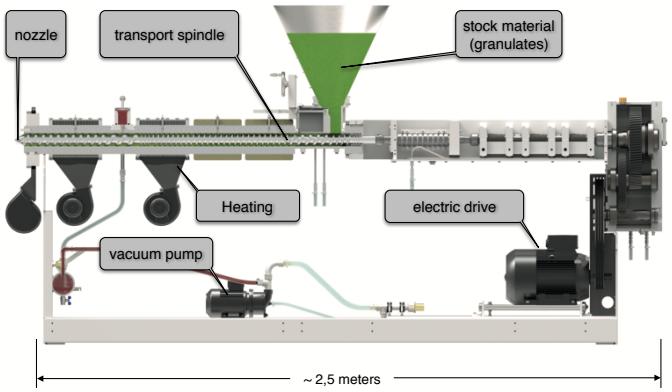

Figure 1. Working principle of an extruder
(courtesy: Hans Weber Maschinenfabrik, Kronach, Germany)

All the papers cited above assume the content to be available, while we propose to automatically generate the content of the recommendations. The algorithms presented in Section V build a knowledge base that can be edited, i.e., the recommendations can be presented to an experienced operator who can modify or delete them. This is a marked difference to most machine learning algorithms whose knowledge base cannot be edited.

In this paper we extend our previous work. Our initial approach presented in [1] transformed the raw data into an event sequence. In [12] we added a second group of algorithms working with the raw data and performed tests to compare the performance of both algorithm types. In this paper we present a more detailed evaluation of our algorithms. Section VII-C introduces a bigger dataset which we processed differently based on feedback from our test users.

Section II describes the application context and the system architecture. Section III explains the fundamental idea of the knowledge base. In Section IV we detail strategies for data preprocessing. The main component, i.e., the recommendation algorithms, is presented in Section V. Section VI briefly presents the GUI of the system. In Section VII we describe an mostly automatic way to score the quality of the recommendations generated by our algorithm. Section VIII summarizes the different scores achieved by the different algorithms and reviews the computing exigencies of the algorithms. Section IX briefly summarizes our findings so far and then gives an outlook on our work schedule for the future.

II. TEST ENVIRONMENT

The test scenario for the development of the system is a plastics extruder (see Figure 1). The basic purpose of this machine is to melt and transport plastic granulates by means of a threaded spindle towards a nozzle. The main process

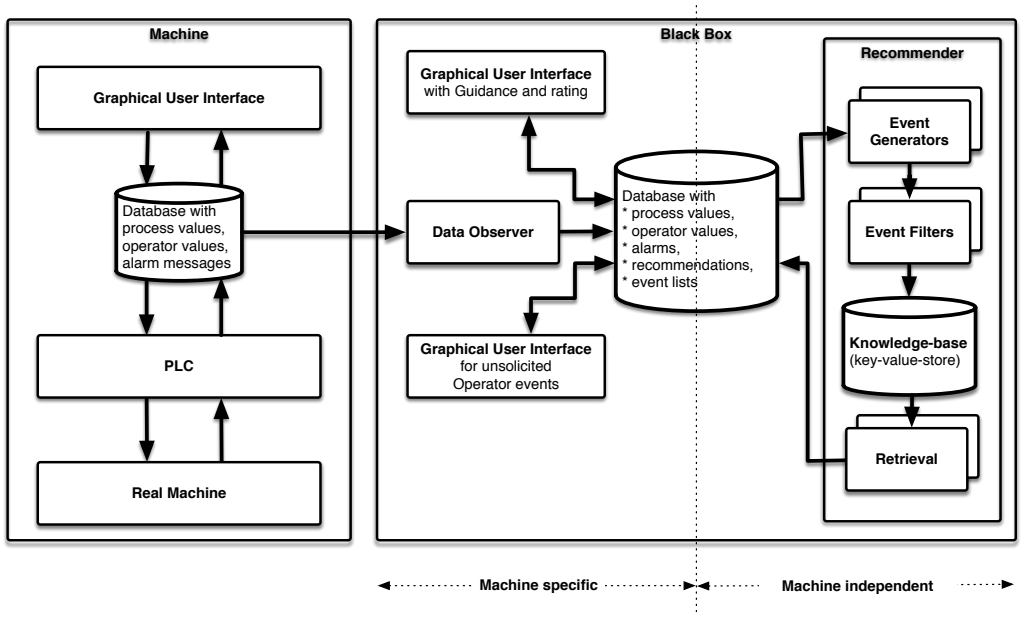

Figure 2. Structure of the system

parameters are the speed of the spindle, the temperature of the extruded material and the pressure of the material at the nozzle. We have equipped three production machines with our systems. All three extruders are manufactured by Hans Weber Maschinenfabrik GmbH. They provide the machine specific part of our system (see Figure 2). The machines are operated by our other project partners: Two of them by H.N. Zapf GmbH & Co. KG and one by Rehau AG & Co. KG. These two partners provide us with data and feedback from the users.

Figure 2 shows the structure of the system. The box on the left represents the existing machine consisting of the actual machine, a PLC-controller for the real-time control and a graphical user interface (GUI). The GUI communicates with the PLC by means of a database containing all PLC-variables and all alarms raised by the PLC. This is a fairly standard architecture.

Our addition is shown in the right box. The Black Box is basically a PC running several software components:

The DataObserver reads the data in the machine database, transforms it to our internal format and stores it in a second database. This component is machine specific and needs read-only access to the PLC variables.

The database is the interface between the machine specific parts and the machine independent part. It stores the PLC-variables and the PLC-alarms written by the DataObserver as well as the output of the Event Generators and the Recommender. The two main tables in the database are shown in Tables I and II. The PLC-variable table contains all the process and operator values of the machine's PLC. Each variable is stored in its own column. The DataObserver adds a new row every 10 seconds. The second table stores alarms which are raised by the machine.

The Recommender is machine-independent. It is the key component of the system. It builds a knowledge base from the logged PLC-variables and generates recommendations when it detects a new alarm message in the database.

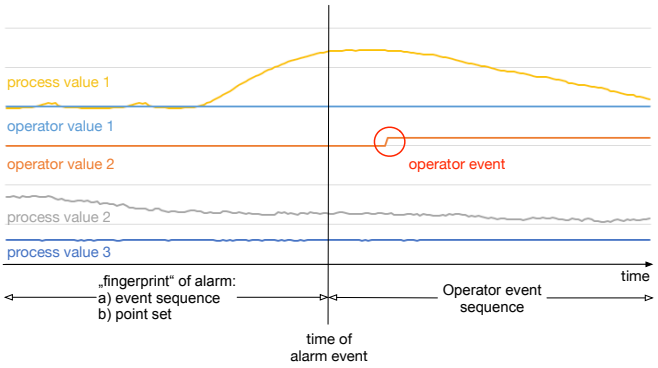

Figure 3. Principle of user guidance generation

The GUI is used to translate the recommendations into operator instructions that reflect the wording on standard GUI of the machine. It displays a recommendation and allows the operator to evaluate the quality of the recommendation.

III. Proposed Approach

The basic idea of our approach is to extract operator knowledge from the continuous stream of PLC-variables logged during the operation of the machine. We want our algorithms to be as generic as possible and to work without deeper understanding of the machine. However, we need information to recommend actions to the machine operator: Which columns in the PLC variable database (Table I) correspond to values affected by the process, e.g., actual temperature, and which are controlled by the operator, e.g., temperature set-point. We call the first set of variables process values and the second set operator values. The rows of the database hold consecutive values for these variables.

With this information we propose to generate user guidance as shown in Figure 3: The alarm messages in the alarm database are used to tag discrete parts of the endless stream of

data logged in the PLC-variable database. The alarm tag and the data before the occurrence of the alarm are regarded as a fingerprint that identifies the alarm. This fingerprint consists of process and operator variables. The data after the occurrence of the alarm also consists of process and operator values. The actions performed by the operator will change some of these operator values. By detecting these changes we build a sequence of operator events. This sequence represents the actions of the operator corresponding to a fingerprint.

With fingerprints and corresponding operator sequences we build a key value store representing the operators' expert knowledge. The fingerprint is the key and the operator sequence the value. These key-value pairs are generated by monitoring the continuous stream of data logged during the operation of the machine. When the machine raises an alarm the algorithms generate a key for that new problem. Now the key-value pairs in the knowledge base are searched for the best matching elements. The best matches are then provided as a recommendation to solve the problem. The fingerprint and the operator sequence performed by the operator is then stored in the knowledge base as a new key value pair.

In Section V we describe two generally different approaches for the processing of the fingerprints: One set of algorithms directly uses the process and operator variables. The other set converts the data in the fingerprint into an event sequence. Events are generated for the parts of a time series where the data changes. Several events for one or several columns constitute an event sequence.

IV. Generation of events

The curves for process value 2 and process value 3 presented in Figure 3 show little variance over time. By generating events for the points in time where a variable changes we can suppress bits of data with no or low variance.

We distinguish between process events and operator events. The process events are generated from process values, the operator events from operator values.

For process events we have developed two different algorithms. The selection of the appropriate algorithm for each process value needs to be done by a person with sufficient knowledge about the process and the machine.

We encode all events as string values containing the name of the PLC variable and either a numerical value or a class name. This encoding is not as efficient as a binary coding but it allows us to interpret the event lists during our tests.

A. Operator Events

Operator events represent human interaction with the machine. These events are identified by checking columns containing operator values.

We developed two algorithms to generate these operator event sequences.

The first type compares all consecutive rows within an alarm situation. For each column whose value changes it generates an event when the new value stays constant for at least three rows. We need this restriction because some operator inputs are made by a hand wheel. To detect the final value we wait until the operator has not made a change for 30 seconds. If we use this method we can get multiple steps for one operator value.

The second method lakes all operator values in the moment the alarm starts and compares them with the values they have at the end of the alarm. If one value has changed it will then be part of our operator sequence.

In consultation with our project partners we use the second type for dataset 4. As a further improvement we replace the absolute value with a relative value and come up with recommendations like "Change drive speed by +5 rpm" instead of telling "Set the drive to 15 rpm".

In both cases these events represent operator interactions. They are coded as string values, e.g., `Loop2_SP_170`, containing the name of the PLC variable and the (relative) value of the setpoint adjusted by the operator.

B. Process Events

For process variables we distinguish two types of variables: The first group of variables has an associated setpoint. The second group does not have such a setpoint, but we are able to create our own setpoint.

For both classifications we chose to use literals instead of numbers for our classes. Thus, we can easily distinguish between operator events and process events when reviewing the data.

TABLE I. Example of the database with PLC-Variables

PointID	AIn1_1	AOut1_1	AIn1_2	AIn2_1	AOut2_1	Loop1_1_PV	Loop1_1_SP	Loop1_1_OP	...
2016-09-27 07:42:37	87.488	87	65.169	95.985	95	22.5	120	0	...
2016-09-27 07:42:47	87.47	88	65.638	95.973	95	22.6	120	0	...
2016-09-27 07:42:57	87.03	87	65.461	96.027	96	22.6	120	0	...
2016-09-27 07:43:07	88.378	88	64.559	96.011	96	22.7	120	0	...
2016-09-27 07:43:17	85.695	86	64.93	96.008	96	22.8	120	0	...
2016-09-27 07:43:27	87.792	88	65.272	95.968	95	22.8	120	0	...
2016-09-27 07:43:37	87.615	88	65.589	96.004	95	22.7	120	0	...
2016-09-27 07:43:47	86.761	87	65.272	95.946	96	22.8	120	0	...
2016-09-27 07:43:57	87.446	87	65.191	95.997	96	22.8	120	0	...
2016-09-27 07:44:07	87.213	87	65.182	95.969	96	22.8	120	0	...
2016-09-27 07:44:17	86.472	86	64.172	95.973	96	22.8	120	0	...
2016-09-27 07:44:27	86.634	86	65.349	95.893	96	22.7	120	0	...
2016-09-27 07:44:37	87.291	87	64.776	96	96	22.6	120	0	...
2016-09-27 07:44:47	87.58	87	65.428	96.023	96	22.5	120	0	...
2016-09-27 07:44:57	86.966	87	65.522	95.981	96	22.6	120	0	...
...									

1) Process Values with Setpoint: We need a pair of process and operator variable. The latter contains the setpoint. We calculate the deviation from the setpoint of the variable

$$Var_{rel} = \left| \frac{Var_{Process}}{Var_{Setpoint}} \cdot 100\% \right| \quad (1)$$

Then, we sort Var_{rel} into one of the classes shown in Table III and generate an event every time the process value enters a new class. These events are also coded as string values, e.g., `Loop_1_2_PV_X`.

2) Process Values relative to Moving Average: For process variables without corresponding setpoint, we calculate the arithmetic mean over the last n values. With this as the setpoint we treat the variable as described in Section IV-B1.

For our fingerprint, we have arbitrarily chosen to use the five minutes before an alarm occurred. So, we have 30 datasets per alarm. To create a suitable moving average we have to take much more then 30 datasets. At the moment, we again arbitrarily chose to use 360 values, which equals one hour. Both the time which is used for the fingerprint and the amount of values used for the moving average need further investigation whether they are reasonable values or not.

V. Recommendation Generation

In case the machine needs operator assistance, i.e., it raised an alarm by adding a new row in the error message table, our guidance generation algorithm is triggered. The fingerprint of the current situation is used as a search key for the knowledge base.

Section V-A describes a set of algorithms that convert the data in the fingerprint into an event sequence and then searches for this sequence. In Section V-B, we introduce algorithms that directly use the N-dimensional point set of process and operator variables in the fingerprint as a key.

A. Recommendation Generation using Event Sequences

1) Map: In [1], we used a content addressable memory, i.e., a Java map, to store the knowledge base. We build the knowledge base by iterating through all past occurrences of alarms. For each fingerprint we calculate the event sequence and use it as key and the corresponding operator event sequence as the value. Each key-value pair is then stored in the map. For multiple entries, we store the frequency of the respective sequence in the past.

To generate a recommendation, we simply generate the event sequence corresponding to the fingerprint of the current alarm and query the map for this key. For map entries with several values for one key, we return several recommendations and their frequency in the past. In case this key is not in the map, we cannot generate a recommendation.

TABLE II. Example of the database with machine alarms

MainNr	SubNr	Start	End
212	0	2016-09-27 07:42:37	2016.09-27 07:45:47
213	0	2016-09-27 07:42:47	2016.09-27 07:48:57
214	0	2016-09-27 07:42:47	2016.09-27 10:50:36
215	0	2016-09-27 07:42:47	2016.09-27 07:46:47
216	0	2016-09-27 07:42:47	2016.09-27 07:46:57
221	0	2016-09-27 07:42:57	2016.09-27 07:46:07
224	0	2016-09-27 07:43:17	2016.09-27 07:49:17
126	0	2016-09-27 07:46:17	2016.09-29 00:48:02
...			

TABLE III. Classes for process variables

Var_{rel}	$\leq 1.77\%$	$\leq 3.16\%$	$\leq 5.62\%$	$\leq 10.0\%$	$\leq 17.78\%$
Class	A	B	C	D	E
Var_{rel}	$\leq 31.62\%$	$\leq 56.23\%$	$\leq 100.0\%$	$\leq 177.8\%$	$> 177.8\%$
Class	F	G	H	I	X

2) Map with Statistical Event Filtering: The map presented in Section V-A1 will only find a recommendation if the search key exactly matches one of the stored keys. If the event sequences contain spurious events, e.g., caused by noise, we cannot find a match. So, we created an algorithm to suppress the irrelevant events.

We use a statistical approach identifying unimportant events to remove them from the process event sequence. The filtered event list is then processed as described in Section V-A1.

The filtering is done in two steps. First, we iterate through the event sequences of all stored fingerprints for one alarm and count how often an event is contained in the set of event sequences. In the next step, we remove all events with a frequency below a defined threshold from the event sequences. So far our tests indicate that 0.5 is a reasonable choice.

3) String matching: The map algorithm presented in Section V-A1 requires the key in the query to be absolutely identical to one key in the map. Thus, it is very sensitive to changes in the key, i.e., the event-sequence. We alleviate this rigorous selection criterion by storing all keys and the corresponding values in a vector. We then loop through all the elements and compare the sequence of the query to the sequence stored in the vector. We return the value as a recommendation that is most similar to the key in the query.

We evaluate the similarity with two distance measurements for string values: the Levenshtein distance and the Jaccard distance, as described in [13]. For the comparison, we treat each event in the sequence as one letter.

a) Levenshtein distance: For two strings of length a and b the Levenshtein distance is defined as:

$$d_l = \begin{cases} \text{if } a = b, & 0 \\ \text{otherwise}, & min\left\{ d_{lv}\left(a, b_{1:|b|-1}\right) + 1 \right. , \\ & d_{lv}\left(a_{1:|a|-1}, b\right) + 1, \\ & \left. d_{lv}\left(a_{1:|a|-1}, b_{1:|b|-1}\right) + 1 \right\} \end{cases} \quad (2)$$

b) Jaccard distance: For the sets A and B each containing all letters of the strings a and b, the Jaccard distance is defined as:

$$d_j(A, B) = 1 - J(A, B) = 1 - \frac{|A \cap B|}{|A \cup B|} \quad (3)$$

with $0 \leq d_j(A, B) \leq 1$.

c) Some Examples: Table IV gives some examples by comparing four different event sequences. Smaller numbers in the distance columns indicate more similar sequences. The values in the Jaccard and the Levenshtein columns use different scales and are only comparable inside their respective column.

The first row shows the distances between two event sequences having one event switched. The Jaccard distance has the lowest possible value of 0, whereas the Levenshtein distance returns 2 by a maximum possible value of 3.

In the second example, only one event differs from another element. Here the Jaccard distance returns 0.5 whereas the Levenshtein distance returns 1.

The third row shows an example with one switched event and one different event. Jaccard judges them with a distance of 0.5 whereas Levenshtein returns 3. Although they are not completely different Levenshtein returns its maximum value.

The last three rows show the performance on completely different event sequences. Both distance measurements confirm this dissimilarity.

In conclusion, the Jaccard distance is more robust if some event sequences are switched. If both have a completely different order but the same elements, they will be treated as similar. Whereas the Levenshtein distance will treat them as totally dissimilar in the worst case.

B. Recommendation using point sets

The approach described in Section V-A3 allows for disturbance by not requiring a complete match between search key and stored key. We can take this idea one step further by regarding the data points in the fingerprint as a set of n-dimensional points. For one timestamp every process and operator value is one dimension.

As in Section V-A3, we store the point sets, i.e., the keys, and the corresponding operator events, i.e., the values, in a vector. We then loop through all the elements and compare the sequence of the query to the sequence stored in the vector. We return the value as a recommendation that is most similar to the key in the query. The procedures to find the most similar point set are described in the following sections.

1) Distance measurements for points: Bacher et al. [14] propose several ways to compare two points in space.

a) Minkowski-Metric: For two points a and b with the dimension i the Minkowski-Metric is defined as:

$$d_m(a,b) = \left[\sum_i |a_i - b_i|^r\right]^{\frac{1}{q}} \tag{4}$$

If we modify the two parameters q and r, we get different distance measurements. In particular, we use:

- $q = \infty$ and $r = \infty$: Chebychev- or Maximum-distance
- $q = 1$ and $r = 1$: Manhattan-distance
- $q = 2$ and $r = 2$: Euclidean-distance
- $q = 1$ and $r = 2$: Squared Euclidean-distance

TABLE IV. COMPARISON BETWEEN JACCARD AND LEVENSHTEIN DISTANCE

EventSequence 1 (query)	EventSequence 2 (key)	Jaccard	Levenshtein
T_1_A#T_3_C#AIn1_B	T_1_A#Ain1_B#T_3_C	0	2
T_1_A#T_3_C#AIn1_B	T_1_B#T_3_C#AIn1_B	0.5	1
T_1_A#Ain1_B#T_3_C	T_1_B#T_3_C#AIn1_B	0.5	3
T_1_A#T_3_C#AIn1_B	T_2_A#AIn2_C#T_4_D	1	3
T_1_A#Ain1_B#T_3_C	T_2_A#AIn2_C#T_4_D	1	3
T_1_B#T_3_C#AIn1_B	T_2_A#AIn2_C#T_4_D	1	3

b) Jaccard-Metric: The Jaccard similarity can be used for sets, as described in Section V-A3. It can also be used for points. Let a and b be two n-dimensional data points. With i being one of the dimensions the Jaccard distance for two points is defined as:

$$d_{jp}(a,b) = 1 - \frac{\sum_i a_i + \sum_i b_i - 2\sum_i \min(a_i, b_i)}{\sum_i a_i + \sum_i b_i - \sum_i \min(a_i, b_i)} \tag{5}$$

c) Canberrra-Metric: The Canberra metric for two points a and b with i being a dimension it is defined as:

$$d_c(a,b) = \sum_i \frac{|a_i - b_i|}{a_i + b_i} \tag{6}$$

Identical differences in the nominator result in a higher weighting for small values of the denominator.

2) Distance measurements for point sets: In this section, we discuss distances for point sets. The Hausdorff distance as well as the Linkage measurements make use of point distances presented in Section V-B1. Jaccard, Dice and Sokal-Sneath distances on the other hand do not use them.

a) Hausdorff-Distance: The Hausdorff-distance defines a similarity between two not empty sets A and B.

Let a be an element of set A and b an element of B. And d being a distance defined in Section V-B1, the distance between a and B is defined as:

$$D_H(a,B) = \min_{b \in B} d(a,b) \tag{7}$$

With this definition, the Hausdorff-distance between two sets A and B can be expressed as:

$$d_h(A,B) = \max\left\{\max_{a \in A} D(a,B), \max_{b \in B} D(b,A)\right\} \tag{8}$$

b) Single-, Complete- and Average-Linkage: These metrics determine the distance from one point of one set to all the other points of the second set.

The Single-Linkage method only uses the smallest distance between two points of both sets. The Complete-Linkage on the other hand uses the longest.

The Average-Linkage method calculates the arithmetic mean of all point-to-point distances. This mean is taken as the result.

Let A and B be two point-sets. Together with a and b being points in A and B and d being a distance defined in V-B1, the Single-Linkage can be written as:

$$D_{SL}(A,B) = \min_{a \in A, b \in B} d(a,b) \tag{9}$$

With the same notation, the Complete-Linkage can be written as:

$$D_{CL}(A,B) = \max_{a \in A, b \in B} d(a,b) \tag{10}$$

Let $|A|$ be the number of points contained in set A and $|B|$ the number of points in set B, then the Average-Linkage is defined as:

$$D_{AL}(A,B) = \frac{1}{|A|\,|B|} \sum_{a \in A, b \in B} d(a,b) \tag{11}$$

VI. Guidance Display

As soon as the Recommender has found operator sequences that fit the current fingerprint the GUI decodes the operator event sequence and displays the recommendations as shown in Figure 4.

On the left hand a list of currently raised alarm messages is shown. The list on the top right shows the recommendations found for the current fingerprint belonging to this alarm. Underneath this lists the selected recommendation is displayed with detailed actions that the operator has to take.

The recommendations can be scored by the operator. In future this will help our algorithms to create better results.

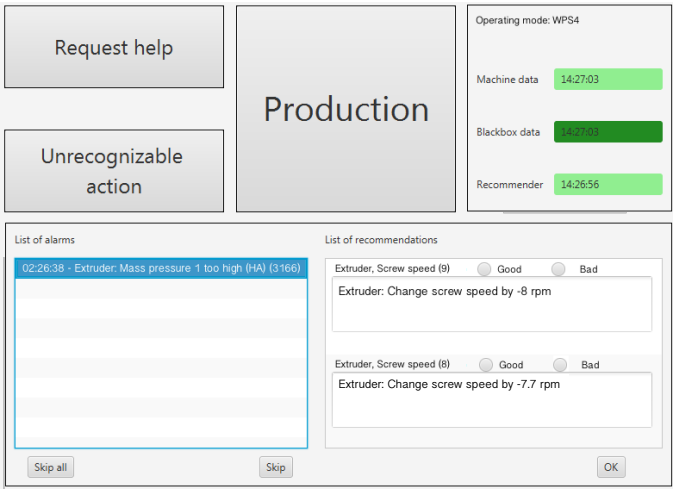

Figure 4. An example user guidance display

VII. Quality of Recommendations

In the previous section we described a system for generating user guidance. Since this system has been designed to interact with the operator during production it is not well suited for comparative testing of different algorithms and parameter sets lest we disrupt production or frustrate the operator. Instead in this section, we use a "batch-mode"to automatically evaluate our algorithms.

The batch-mode basically iterates through a long time series of PLC-variables and alarms. It triggers the recommender for each alarm encountered during the iterations and requests a recommendation. This recommendation is compared to the operator sequence stored in the time series. This sequence is taken as the expected recommendation or "truth" for this particular alarm. If the recommendation is equal to the operator sequence in the test-set the test is positive and C_{pos} is incremented. If a different sequence is returned or the algorithm returns nothing the test fails and C_{neg} is incremented. With these two parameters we can calculate the Quality Q_{alg} for the algorithm:

$$Q_{alg} = \frac{C_{pos}}{C_{neg} + C_{pos}} \cdot 100\% \qquad (12)$$

We need to train the recommender algorithms before feeding them alarms. We generate the training data by dividing the dataset into four subsets with an equal number of alarms. One of these sets is taken as the test-set. The others are combined into a training set. Thus, we can run four tests on each of our algorithms by using the test-sets as training data. The remaining sets of alarms and data are used to train the algorithms.

TABLE V. Dataset 1: The five most frequently raised alarms

Alarm number	Alarm count	With op-sequence	Different op-sequences
1101	171	57	12
1012	107	86	19
12	106	53	19
22	106	49	16
2	59	30	10

The expression in Equation (12) ranges between $0 \leq Q_{alg} \leq 100\%$. However, a closer look at the test-sets reveals cases where an operator sequence only occurs in the test set and not in the training set. Thus, the algorithm cannot learn this recommendation and is not able to detect it correctly. Some other keys have more than one associated value. In these cases the algorithm is not able to decide which operator sequence is the right one. Consequently the maximum possible score is less than 100%.

For our evaluation, we use the data gathered during 5 minutes before the alarm to generate the fingerprint. The operator sequence is generated for the duration of the alarm, i.e., for the period of time in which the alarm condition is true.

A. Dataset 1 – Simulation

As a first dataset, we use a slightly extended version of the data, we used in [1]. This dataset was obtained by operating a simulation model of the production machine. It contains $N_{Rows} = 1,150,063$ rows and $N_{Alarms} = 1,254$ alarms.

As in [1], we focus on alarm 1101. There are 171 instances of this alarm in the dataset. For 57 of these instances, we could generate the 12 different operator sequences shown in Table VI. The remaining 114 occurrences of the alarm do not contain operator sequences because the simulation was not always operated properly.

8 of these operator sequences corresponding to 10 occurrences of alarm 1101 are only contained in one of the four subsets. Thus these sets are either part of the training data or of the test data. Therefore, we reduce the maximum possible score from 57 to 47.

For the event based algorithms, we found 3 ambiguous event sequences. Table XI shows all event sequences. The ambiguous event sequences have more than one corresponding operator sequence. We consider ambiguous sequences as unlearnable. Our algorithm will always recommend the sequence

TABLE VI. Dataset 1: Operator sequences

Frequency	Operator Sequence	Learnable
36	Loop1_3_SP_170	yes
7	Loop1_2_SP_170#Loop1_3_SP_170	yes
4	Loop1_4_SP_170	yes
2	Loop1_3_SP_250	no
1	Loop1_3_SP_225	no
1	AOUT1_1_OutValue_10	no
1	AOUT1_1_OutValue_29	no
1	AOUT1_1_OutValue_50	no
1	AOUT1_2_OutValue_9	no
1	Loop1_7_SP_250	no
1	Loop1_2_SP_250#Loop1_3_SP_250	no
1	Loop1_3_SP_170#Loop1_4_SP_170	no

with the highest frequency. Therefore, we subtract the number of the other sequences from the maximum possible. In our case this reduces the maximum possible by 10 to 37.

For event sequence based algorithms we get a maximum possible score of

$$Q_{alg_{1ev_{max}}} = \frac{37}{57} \cdot 100\% \approx 65\% \quad (13)$$

For the point based algorithms, we get a maximum score of

$$Q_{alg_{1p_{max}}} = \frac{47}{57} \cdot 100\% \approx 82\% \quad (14)$$

B. Dataset 2 – Converted Data from real Machine

The second set of process- and operator values was taken from a production machine. This dataset is identical to the dataset we introduced in [12].

The database of that machine does not yet conform to our interface standard and thus did not log the alarms. We resorted to generating our own alarms based on process experts' definitions for lower and upper bounds of process values. The first occurrence of a value being outside the bounds was the starting time for an alarm. The stopping time was taken the moment the value was back inside the bounds.

The resulting database contains $N_{Rows} = 1,223,992$ rows describing the machine state and $N_{Alarms} = 5,667$ alarms. Table VII shows the five most frequent alarms in the dataset. With the machine logging the data every 10 seconds, we have a runtime of $\approx 3,400$ hours. This means, we have 1.7 alarms per hour. According to H.N. Zapf GmbH & Co. KG, the project partner operating the machine, this number seems to be very high.

Not all alarms, we generated from the machine were useful for our test. Some alarms were not followed by operator events. We assume that these alarms are artifacts of our data preparation algorithm. Other alarms obviously occurred at the end of a production shift and consequently led to the recommendation to shut down the machine. We chose alarm 225 for our evaluation.

Alarm 225 occurred 1949 times. For 277 instances of this alarm, we could create a fingerprint and an operator sequence.

We assume that this difference is partly due to our self-generated alarms and partly due to spurious and short alarms that disappear without user intervention.

For these 277 instances of alarm 225, we generated the 22 operator sequences shown in Table VIII.

13 of these operator sequences corresponding to 46 event sequences occur in only one of the four subsets. Consequently, we reduce the theoretical maximum by 46. Furthermore, we found 33 ambiguous event sequences shown in Table XII. We subtract all but one of these ambiguous sequences except for those that are already marked as not learnable because they are appearing in only one subset. So, we have to subtract further 47 alarms.

TABLE VII. DATASET 2: THE FIVE MOST FREQUENTLY RAISED ALARMS

Alarm number	Alarm count	With op-sequence	Different op-equences
225	1949	277	22
423	1059	70	43
122	763	243	93
123	503	149	69
214	238	142	81
213	109	76	60

TABLE VIII. DATASET 2: OPERATOR SEQUENCES FOR ALARM 225

Frequency	Operator Sequence	Learnable
54	AOut1_1_85#	yes
53	AOut1_1_84#	yes
30	AOut1_1_83#	yes
30	AOut1_1_86#	yes
20	AOut1_1_81#	yes
17	AOut1_1_82#	yes
16	AOut1_1_77#	no
14	AOut1_1_87#	yes
11	AOut1_1_80#	yes
9	AOut1_1_76#	no
7	AOut1_1_78#	no
5	AOut1_1_79#	no
2	AOut1_1_88#	yes
1	AOut1_1_0#AOut2_1_6#	no
1	AOut1_1_83#AOut2_1_64#	no
1	AOut1_1_83#AOut2_1_98#	no
1	AOut1_1_84#AOut2_1_98#	no
1	AOut1_1_85#AOut2_1_98#	no
1	AOut1_1_87#AOut2_1_73#	no
1	AOut1_1_89#	no
1	AOut1_1_91#Loop2_2_SP_230#	no
1	AOut2_1_99#	no

For event based algorithms, we get a maximum score of:

$$Q_{alg_{2ev_{max}}} = \frac{184}{277} \cdot 100\% \approx 66\% \quad (15)$$

For the point based algorithms, we get a maximum score of:

$$Q_{alg_{2p_{max}}} = \frac{231}{277} \cdot 100\% \approx 83\% \quad (16)$$

C. Dataset 4 – Real Data

Dataset 4 was gathered on a new production machine equipped with our system. The machine became fully productive a few weeks ago, so we now have $N_{Rows} = 1,062,541$ of data corresponding to a run time of $\approx 2,951$ hours. During this time period, we found $N_{Alarms} = 10.431$ alarms.

We could also conduct some interviews with operators regarding the recommendations generated from the earlier version of that dataset presented as dataset 3 in [12].

As a result of this discussion we now distinguish between set-up operation and production. For now we focus on production because the set-up operations are deemed too sensitive to environmental influences not included in the process data. We found $N_{Alarms} = 1446$ alarms during operation.

Further investigation of our recommendations showed that the gap between some alarms is less or equal to ten seconds. To get operator sequences for these instances we decided to combine two or more alarms if the gap between them is 10 seconds or less. We also removed alarms with a duration of less than 10 seconds.

This gives us total of $N_{Alarms} = 358$ alarms. For 194 of these alarms we can generate an operator sequence. Table IX lists the recommendations that occur more than once.

1) Dataset 4a – including "no operation": Previously we ignored alarms without an operator event sequence. Our interview partners assured us however, that sometimes the appropriate operator action is to just wait for the alarm to clear

TABLE IX. DATASET 4: OPERATOR SEQUENCES

Frequency	Operator Sequence	Learnable
164	*no operator action*	yes
4	Meltpump.Output.Value_-1.0#	yes
4	Meltpump.Output.Value_-9.0#	yes
4	Meltpump.Output.Value_-4.0#	yes
4	Meltpump.Output.Value_-2.0#	yes
3	Meltpump.Output.Value_-0.1#	yes
3	Meltpump.Output.Value_-3.0#	yes
3	Meltpump.Output.Value_-0.5#	yes
3	Meltpump.Output.Value_-5.0#	yes
3	Meltpump.Output.Value_-10.0#	no
3	Tool.Temp1.SP_5.0#	no
2	Meltpump.Output.Value_-0.3#	yes
2	Meltpump.Output.Value_-0.2#	yes
2	Meltpump.Output.Value_12.0#	yes
2	Meltpump.Output.Value_4.0#	yes
2	Meltpump.Output.Value_-0.4#	yes
2	Meltpump.Output.Value_-5.8#	yes
2	Tool.Temp2.SP_5.0#	yes
2	Extruder.Temp3.SP_5.0# Extruder.Temp4.SP_5.0# Tool.Temp0.SP_5.0#	yes
2	Tool.Temp3.SP_-5.0#	no
2	Meltpump.Output.Value_-7.0#	no
2	Meltpump.Output.Value_-20.0#	no
2	Tool.Temp1.SP_10.0#	no

itself. Therefore, we include a "do nothing"-recommendation for alarms without operator event sequences.

Thus, we get get a maximum score for the point set based algorithms:

$$Q_{alg_{4_{a}pmax}} = \frac{208}{358} \cdot 100\% \approx 58\% \quad (17)$$

And for the event sequence based ones:

$$Q_{alg_{4_{a}evmax}} = \frac{198}{358} \cdot 100\% \approx 55\% \quad (18)$$

2) Dataset 4b – excluding "no operation": The 164 alarms without operator event sequences constitute nearly half of our alarms. Thus any algorithms recommending "do nothing" will achieve a high score. For a more detailed study of the other recommendations we remove these alarms from the test sets. This gives us a total of $N_{Alarms} = 195$ alarms. The learnable operator sequences are the same as in Table IX, only the first line with the empty recommendation is not in the dataset any more

If we determine which operator sequences are learnable we get a maximum score for the point set based algorithms:

$$Q_{alg_{4_{b}pmax}} = \frac{44}{195} \cdot 100\% \approx 23\% \quad (19)$$

And for the event sequence based ones:

$$Q_{alg_{4_{b}evmax}} = \frac{38}{195} \cdot 100\% \approx 19\% \quad (20)$$

VIII. COMPARISON OF THE ALGORITHMS

After processing the three datasets through all of our algorithms, we calculated the score for each dataset according to Equation (12). Table X shows the results of our tests. As discussed in Section VII the resulting values for Q_{alg} are not comparable between the datasets because of unlearnable and ambiguous operator event sequences.

Equations (13) to (20) give the maximum possible score for each dataset. With this score, we can normalize the results as

$$Q_{alg_{rel}} = \frac{Q_{alg}}{Q_{alg_{max}}} \cdot 100\% \quad (21)$$

To put the performance into perspective, we calculate the performance of a simple approach that always recommends the most frequent user sequence in the knowledge base. Such an approach will propose a correct operator sequence for all the alarms associated with the most frequent operator sequence:

$$Q_{simple} = \frac{100\%}{N_{OpSeq}} \cdot N_{mostFreqOpSeq} \quad (22)$$

i.e.,

$$Q_{simple_1} = \frac{100\%}{57} \cdot 36 \approx 63\% \quad (23)$$

$$Q_{simple_2} = \frac{100\%}{277} \cdot 54 \approx 19\% \quad (24)$$

$$Q_{simple_{4_a}} = \frac{100\%}{208} \cdot 164 \approx 79\% \quad (25)$$

$$Q_{simple_{4_b}} = \frac{100\%}{44} \cdot 4 \approx 9\% \quad (26)$$

Figure 5 shows the normalized performance of the different algorithms on the three datasets and for reference the normalized scores of the simple approach.

The mapping algorithm introduced in [1] and V-A1 scored $\approx 63\%$ on dataset 1. On dataset 2 it performed less with only $\approx 12\%$. On dataset 4a it only gives a few correct recommendations. On dataset 4b, however, it did not give any correct recommendations. We attribute this lack of performance to the fact that we have only a few identical event sequences. Only operator sequences which are mapped at least two times to identical event sequences can be discovered by the simple mapping algorithm. In dataset 4a we found 38 operator sequences which could be discovered by the simple mapping, in dataset 4b instead it was only one operator sequence. In total it scores less than the simple approach on all datasets based on real data.

With $\approx 42\%$ on dataset 2 the statistical filter performs better than the simple approach and is almost on par with the point based algorithms. It also shows good performance on dataset 4a together with the Levenshtein String Matching algorithm. However, in this case it is outperformed by the simple algorithm. With 16.7 % on dataset 4b it shows the best performance of all tested algorithms.

The point based algorithms score better on dataset 1 with almost 90% and mostly $> 40\%$ on dataset 2. Dataset 4a, however, proves difficult with $\approx 30 \ldots 37\%$. On dataset 4b the best results are near the simple algorithm or just a few percent above.

A. Computing exigencies

1) Memory requirements: The recommendation algorithms need to store keys and values for all past occurences of an alarm.

In our test environment, we used a table with $N_{Columns} = 51$ data columns in dataset 2 and 3 and $N_{Columns} = 250$ in dataset 1.

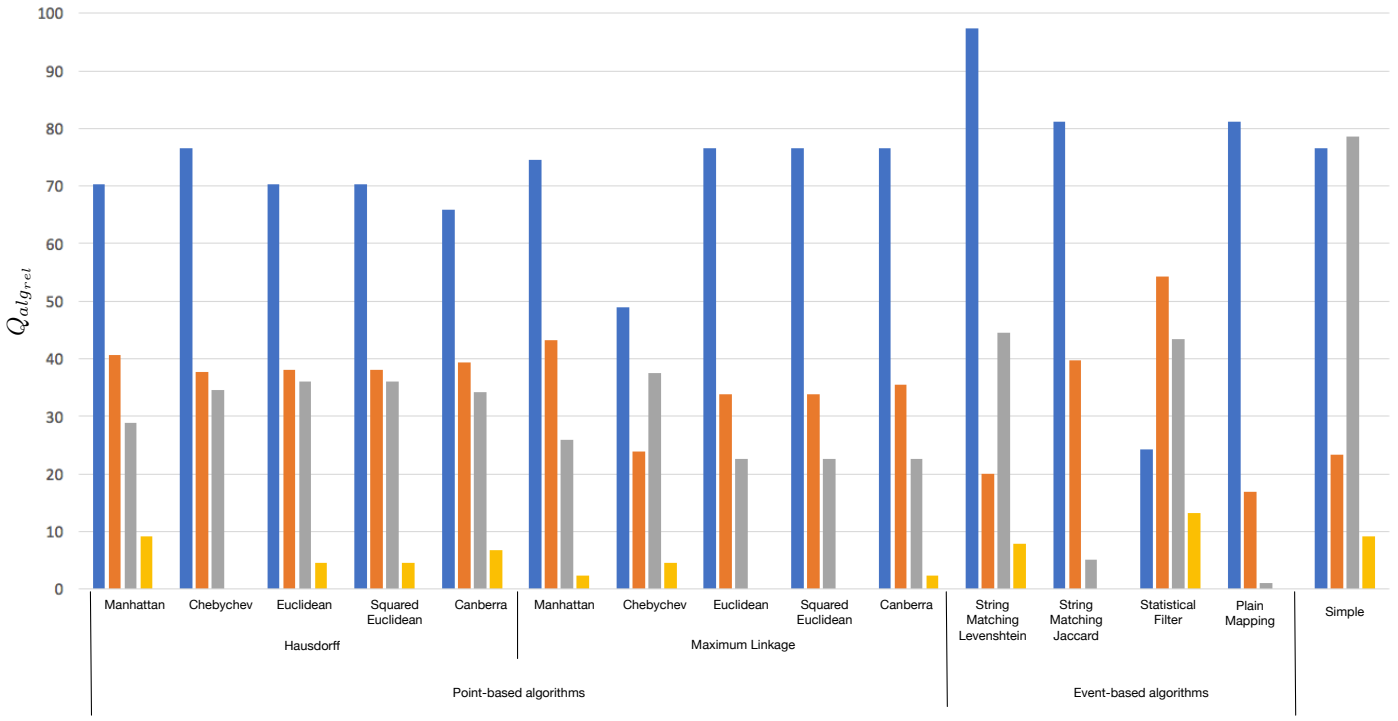

Figure 5. Normalized results for various recommender algorithms (left bars: Dataset 1, second bars: Dataset 2, third bars: Dataset 4a, right bars: Dataset 4b)

For every fingerprint, we use $N_{Rows} = 30$ rows, i.e., $N_{Rows} = 30$ datapoints per set.

For the dataset 1, we used alarm 225 with 276 occurences and split the data into four test sets. The test data for each test set contained approximately $N_{Alarms} = 210$ point sets.

The point based algorithms store $N_{Rows} \cdot N_{Columns}$ double variables as keys and a small number of operator events as values. Thus, in our worst case one fingerprint uses $30 \cdot 250 \cdot 8\,\text{byte} \approx 60\,\text{kbyte}$ of memory for the point set algorithms.

In $1\,\text{Gbyte}$ of memory, we can store data for ≈ 16.000 alarms. At a rate of $2\frac{\text{Alarms}}{\text{h}}$, which is considered a very high rate, this corresponds to $\approx 8000\text{h}$ of operation. A standard computer should have enough memory for two years of operation.

With an efficient coding of the events the event based algorithms will use considerably less memory for the fingerprint.

B. Processing times

Once the data is stored, the algorithms have to iterate through the stored keys to identify the proper record.

To determine the distance between two sets, we have to compare each data point from one set with each data point from the other set. For point based algorithms this results in N^2_{Rows} comparisons. Every dimension has to be considered in the distance measurements, so overall a comparison of two sets will take $N^2_{Rows} \cdot N_{Columns}$ floating point operations. To find a new recommendation the new point set has to be compared to all respective point sets in the stored data resulting in $N^2_{Rows} \cdot N_{Columns} \cdot N_{Alarms}$ floating point operations.

For our test one recommendation will take round about $30^2 \cdot 250 \cdot 276 \cdot \frac{3}{4} \approx 46$ million floating point operations.

A standard computer should need significantly less than $1\,\text{sec}$ to produce a recommendation.

Event based algorithms do not have to compare all points but just the events generated from the points. The processing for the event generation is run only once prior to the iteration through the stored keys and therefore does not significantly increase the total processing time.

IX. Conclusion and Future Work

We presented a system to extract knowledge from data logged during the operatiom of a production machine. Three systems have been linked to actual production machines and have started to collect data.

Based on the scoring introduced in [12] we have shown that our recommender algorithms outperform a simple algorithm by a factor of 2 on some datasets. On the real data in our latest dataset they lack performance.

That being said, we want to point out that all algorithms need the operator sequences, we generate from the logged data. On dataset 4b we found 195 alarms with 159 different operator sequences. This shows that our approach is able to extract operator-knowledge from the logged data. Table IX shows that we still only have a few cases for each operator sequence. We hope that with more data we will also find more cases for these operator sequences, not just more different operator sequences.

Alarm conditions in the real data sometimes overlap each other, e.g., a warning for exceeding a first temperature level (high-alarm) and a second more urgent warning for exeeding a second higher temperature level (high-high-alarm). In these cases we generate several, usually identical, operator sequences and consequently generate multiple recommendations that might confuse the operator. Therefore, we plan to generate operator sequences without considering the alarms.

This approach will also permit us to identify situations that need operator action without the machine raising an alarm. This may help us to incite operators to fix potential problems before the machine detects them. Maybe, we can even indicate problems that the machine itself can not identify.

ACKNOWLEDGMENT

We thank our industry partners Hans Weber Maschinenfabrik GmbH, H.N. Zapf GmbH & Co. KG and Rehau AG & Co. KG for giving us access to their machines and feedback on our software.

This work is funded by Hof University's research center for car infotainment and man-machine-interfaces.

REFERENCES

[1] V. Plenk, "Improving special purpose machine user-interfaces by machine-learning algorithms," in Proceedings of CENTRIC 2016 : The Ninth International Conference on Advances in Human-oriented and Personalized Mechanisms, Technologies, and Services, Rome, August 2016, pp. 24 – 28.

[2] V. Plenk, "A benchmark for embedded software processes used by special-purpose machine manufacturers," in Proceedings of the Third international Conference on Software Engineering Advances. Los Alamonitos: CPS, 2008, pp. 166 – 171.

[3] H. J. C. et al., "Learning styles diagnosis based on user interface behaviors for the customization of learning interfaces in an intelligent tutoring system," in Proceedings of the 8th International Conference on Intelligent Tutoring Systems (ITS 2006). Jhongli, Taiwan: Springer, June 26-30 2006, pp. 513 – 524.

[4] B. D. Davison and H. Hirsh, "Predicting sequences of user actions," in AAAI Technical Report WS-98-07, 1998, pp. 5 – 12.

[5] P. K. Wonga, J. Zhonga, Z. Yanga, and C. M. Vong, "Sparse bayesian extreme learning committee machine for engine simultaneous fault diagnosis," in Neurocomputing, 2016, pp. 331 – 343.

[6] K. Zidek, A. Hosovsky, and J. Dubjak, "Diagnostics of surface errors by embedded vision system and its classification by machine learning algorithms," in Key Engineering Materials, 2015, pp. 459 – 466.

[7] J. Luo, C.-M. Vong, and P.-K. Wong, "Sparse bayesian extreme learning machine for multi-classification," in IIEEE Transactions on Neural Networks and Learning Systems, 2014, pp. 836 – 843.

[8] A. Ziaja, I. Antoniadou, T. Barszcz, W. J. Staszewski, and K. Worden, "Fault detection in rolling element bearings using wavelet-based variance analysis and novelty detection," Journal of Vibration and Control, vol. 22, no. 2, February 2016, pp. 396–411.

[9] C. Challiol, A. Fortier, S. Gordillo, and G. Rossi, "A Flexible Architecture for Context-Aware Physical Hypermedia," in 18th International Workshop on: Database and Expert Systems Applications, 2007. DEXA '07., September 2007.

[10] S. Sigg, S. Haseloff, and K. David, "Context Prediction by Alignment Methods," in Proceedings of the 4th International Conference on Mobile Systems, Applications, and Services (MobiSys 2006), June 2006.

[11] S. Sigg, D. Gordon, G. von Zengen, M. Beigl, S. Haseloff, and K. David, "Investigation of Context Prediction Accuracy for Different Context Abstraction Levels," IEEE Transactions on Mobile Computing, vol. 11, no. 6, June 2012, pp. 1047 – 1059.

[12] V. Plenk, S. Lang, and F. Wogenstein, "Scoring of machine-learning algorithms for providing user guidance in special purpose machines," in Proceedings of CENTRIC 2017: The Tenth International Conference on Advances in Human-oriented and Personalized Mechanisms, Technologies, and Services, Athens, October 2017.

[13] M. P. J. van der Loo, "The stringdist Package for Approximate String Matching," The R Journal, vol. Vol. 6/1, 2014, pp. 111 – 122.

[14] J. Bacher, A. Pöge, and K. Wenzig, Clusteranalyse - Anwendungsorientierte Einführung in Klassifikationsverfahren. München: Oldenbourg, 2010.

[15] J. Lunze, Ereignisdiskrete Systeme. München: Oldenbourg, 2006.

APPENDIX A

TABLE X. RESULTS OF OUR TESTS (NON NORMALIZED)

	Dataset 1			Dataset 2		
Algorithm	C_{pos}	C_{neg}	Q_{alg_1}	C_{pos}	C_{neg}	Q_{alg_2}
Plain Mapping	30	6 (21) [1]	52.6	31	67 (179)[1]	11.2
Statistical Filter	9	37 (11) [1]	15.8	100	133 (44)[1]	36.1
String Matching + Jaccard	30	6 (21) [1]	52.6	73	104 (100) [1]	26.5
String Maching + Levenshtein	36	21	63.2	37	240	13.6
Hausdorff + Manhattan	33	24	57.9	94	183	33.9
Hausdorff + Chebychev	36	21	63.2	87	190	31.4
Hausdorff + Euclidean	33	24	57.9	88	189	31.8
Hausdorff + Squared Euclidean	33	24	57.9	88	189	31.8
Hausdorff + Canberra	31	26	15.8	91	186	32.9
Average Linkage + Manhattan	32	25	56.1	83	194	30
Average Linkage + Chebychev	26	31	45.6	80	197	28.9
Average Linkage + Euclidean	26	31	45.6	83	194	30
Average Linkage + Squared Euclidean	32	25	56.1	83	194	30
Average Linkage + Canberra	32	25	56.1	84	193	30.3
Single Linkage + Manhattan	21	36	36.8	93	184	33.6
Single Linkage + Chebychev	21	36	36.8	81	196	29.2
Single Linkage + Euclidean	21	36	36.8	84	193	30.3
Single Linkage + Squared Euclidean	21	36	36.8	84	193	30.3
Single Linkage + Canberra	21	36	36.8	78	199	28.2
Maximum Linkage + Manhattan	35	22	61.4	100	177	36.1
Maximum Linkage + Chebychev	23	34	40.4	55	222	19.9
Maximum Linkage + Euclidean	36	21	63.2	78	199	28.2
Maximum Linkage + Squared Euclidean	36	21	63.2	78	199	28.2
Maximum Linkage + Canberra	36	21	63.2	82	195	29.6

	Dataset 4 a			Dataset 4 b		
Algorithm	C_{pos}	C_{neg}	$Q_{alg_{4_a}}$	C_{pos}	C_{neg}	$Q_{alg_{4_b}}$
Plain Mapping	2	2 (354) [1]	0.6	0	0 (195)[1]	0
Statistical Filter	86	203 (69) [1]	24.0	5	147 (43)[1]	2.6
String Matching + Jaccard	10	7 (341) [1]	2.8	0	3 (192) [1]	0
String Maching + Levenshtein	88	270	24.6	3	192	1.5
Hausdorff + Manhattan	60	298	16.8	4	191	2.0
Hausdorff + Chebychev	72	286	20.1	0	195	0
Hausdorff + Euclidean	75	283	21.0	2	193	1.0
Hausdorff + Squared Euclidean	75	283	21.0	2	193	1.0
Hausdorff + Canberra	71	287	19.8	3	192	1.5
Average Linkage + Manhattan	44	314	12.3	2	193	1.0
Average Linkage + Chebychev	45	313	12.6	0	195	0
Average Linkage + Euclidean	48	310	13.4	0	195	0
Average Linkage + Squared Euclidean	50	308	14.0	1	194	0.5
Average Linkage + Canberra	87	271	24.3	0	195	0
Single Linkage + Manhattan	45	313	12.6	3	192	1.5
Single Linkage + Chebychev	57	301	15.9	3	192	1.5
Single Linkage + Euclidean	48	310	13.4	1	194	0.5
Single Linkage + Squared Euclidean	48	310	13.4	1	194	0.5
Single Linkage + Canberra	85	273	23.7	1	194	0.5
Maximum Linkage + Manhattan	54	304	15.1	1	194	0.5
Maximum Linkage + Chebychev	78	280	21.8	2	193	1.0
Maximum Linkage + Euclidean	47	311	13.1	0	195	0
Maximum Linkage + Squared Euclidean	47	311	13.1	0	195	0
Maximum Linkage + Canberra	57	301	15.9	1	195	0

[1]The first value is the number of wrong recommendations. The value in parentheses is the amount of cases in which the algorithm is unable to find a recommendation. For scoring both numbers are added, i.e., $C_{neg} = Val + (Val)$.

TABLE XI. DATASET 1: EVENT SEQUENCES AND OPERATOR SEQUENCES

EventSequence	Operator Sequence	Frequency	Learnable
Empty[1]	Loop1_3_SP_225	1	no
	Loop1_3_SP_250	2	no
	AOUT1_1_OutValue_29	1	no
	Loop1_3_SP_170	2	yes
	Loop1_2_SP_170#Loop1_3_SP_170	1	yes
	Loop1_2_SP_250#Loop1_3_SP_250	1	no
	AOUT1_1_OutValue_10	1	no
Loop1_4_PV_B#Loop1_4_PV_X	Loop1_4_SP_170	1	yes
Loop1_3_SP_0.0	Loop1_3_SP_170	28	yes
	Loop1_2_SP_170#Loop1_3_SP_170	6	yes
Loop1_4_PV_H#Loop1_4_SP_50.0# Loop1_4_PV_G#Loop1_4_PV_F	Loop1_4_SP_170	1	yes
Loop1_4_SP_0.0	Loop1_4_SP_170	2	yes
Loop1_2_SP_0.0#Loop1_4_SP_0.0	Loop1_3_SP_170#Loop1_4_SP_170	1	no
	Loop1_3_SP_170	2	yes
Loop1_3_SP_10.0	Loop1_3_SP_170	1	yes
Loop1_4_PV_H#Loop1_5_PV_H# Loop1_6_PV_H#Loop1_7_SP_50.0	Loop1_7_SP_250	1	no
Loop1_3_SP_60.0	Loop1_3_SP_170	3	yes
AIN2_1_Value_G#MP1_Value_G#MT1_Value_F# MP1_Value_F#AIN2_1_Value_F#MP1_Value_B# MT1_Value_E#MP1_Value_C	AOUT1_2_OutValue_9	1	no
MP1_Value_E#MP1_BandMin_E#MP1_BandMax_E# Loop1_6_PV_E#Loop1_6_PV_F#Loop1_6_PV_E# Loop1_6_PV_D#Loop1_6_PV_C#Loop1_6_PV_B# Loop1_6_PV_A#MP1_BandMax_F#MP1_Value_H# MP1_BandMax_H#MP1_BandMin_F	AOUT1_1_OutValue_50	1	no

[1]In our simulation empty event sequences can occur when an alarm is generated by changing the alarm condition. This happens during testing or demonstration.

TABLE XII. DATASET 2: AMBIGIOUS EVENT SEQUENCES

Event-Sequence	Operator-Sequence	Frequency	Learnable	Event-Sequence	Operator-Sequence	Frequency	Learnable
1	AOut1_1_84#	2	yes	18	AOut1_1_86#	2	yes
	AOut1_1_82#	1	yes		AOut1_1_87#	1	yes
2	AOut1_1_86#	3	yes	19	AOut1_1_85#	5	yes
	AOut1_1_87#	1	yes		AOut1_1_85#AOut2_1_98#	1	no
	AOut1_1_85#	1	yes		AOut1_1_84#	1	yes
3	AOut1_1_86#	2	yes	20	AOut1_1_82#	1	yes
	AOut1_1_87#	1	yes		AOut1_1_81#	1	yes
4	AOut1_1_84#	5	yes	21	AOut1_1_85#	1	yes
	AOut1_1_85#	4	yes		AOut1_1_87#	1	yes
5	AOut1_1_83#	2	yes	22	AOut1_1_87#	3	yes
	AOut1_1_85#	2	yes		AOut1_1_85#	1	yes
	AOut1_1_84#	1	yes		AOut1_1_86#	1	yes
6	AOut1_1_80#	2	yes	23	AOut1_1_83#	1	yes
	AOut1_1_79#	1	no		AOut1_1_81#	1	yes
7	AOut1_1_87#	1	yes	24	AOut1_1_81#	2	yes
	AOut1_1_86#	1	yes		AOut1_1_80#	1	yes
8	AOut1_1_85#	1	yes	25	AOut1_1_85#	1	yes
	AOut1_1_84#	1	yes		AOut1_1_86#	1	yes
9	AOut1_1_83#	1	yes	26	AOut1_1_80#	3	yes
	AOut1_1_84#	1	yes		AOut1_1_81#	1	yes
10	AOut1_1_84#	2	yes	27	AOut1_1_84#	1	yes
	AOut1_1_83#	1	yes		AOut1_1_85#	1	yes
11	AOut1_1_82#	1	yes	28	AOut1_1_77#	3	no
	AOut1_1_83#	2	yes		AOut1_1_78#	1	no
12	AOut1_1_86#	2	yes	29	AOut1_1_85#	1	yes
	AOut1_1_88#	1	yes		AOut1_1_86#	1	yes
	AOut1_1_87#	1	yes		AOut1_1_84#	2	yes
13	AOut1_1_87#	1	yes	30	AOut1_1_82#	1	yes
	AOut1_1_86#	1	yes		AOut1_1_81#	1	yes
14	AOut1_1_86#	2	yes	31	AOut1_1_85#	1	yes
	AOut1_1_87#	1	yes		AOut1_1_86#	1	yes
15	AOut1_1_85#	2	yes	32	AOut1_1_83#	1	yes
	AOut1_1_86#	2	yes		AOut1_1_85#	1	yes
16	AOut1_1_84#	1	yes	33	AOut1_1_84#	1	yes
	AOut1_1_83#	1	yes		AOut1_1_83#	1	yes
	AOut1_1_85#	1	yes		AOut1_1_85#	2	yes
	AOut1_1_82#	1	yes				
17	AOut1_1_83#	1	yes				
	AOut1_1_85#	7	yes				
	AOut1_1_84#	3	yes				