HUGO: Real-Time Analysis of Component Interactions in High-Tech Manufacturing Equipment (Industry Article)

Yuanzhen Ji  
SAP AG  
Chemnitzer Str. 48  
01187 Dresden, Germany  
yuanzhen.ji@sap.com

Thomas Heinze  
SAP AG  
Chemnitzer Str. 48  
01187 Dresden, Germany  
thomas.heinze@sap.com

Zbigniew Jerzak  
SAP AG  
Chemnitzer Str. 48  
01187 Dresden, Germany  
zbigniew.jerzak@sap.com

ABSTRACT

One of the major problems faced by the high-tech manufacturing industry is the need for automated and timely detection of anomalies which can lead to failures of the manufacturing equipment. Failures of the high-tech manufacturing equipment have a direct negative impact on the operating margin and consequently profit of the high-tech manufacturing industry.

Automated and timely detection of anomalies is a difficult problem, the major challenge being the need to understand the interactions between large amount of machine components. Even very experienced system engineers are not aware of all interactions, especially if those need to be derived from high velocity sensor data. This, in turn, makes it impossible to recognize early warning signals and take action before failure happens.

In this paper we present HUGO\textsuperscript{1} – a system for real-time analysis of component interactions in high-tech manufacturing equipment. HUGO automatically discovers (based on the available sensor data) correlations between machine components and helps engineers analyze them in real-time so as to be able to detect deterioration of the manufacturing equipment conditions in a timely fashion.

Categories and Subject Descriptors

J.1 [Administrative Data Processing]: Manufacturing; I.5.4 [Pattern Recognition]: Applications—signal processing

Keywords

Equipment Monitoring; Complex Event Processing

1. INTRODUCTION

High-tech manufacturing equipment, such as photolithography systems or vapor depositions systems, contain thousands of components which are monitored by hundreds of digital and analogue sensors. The sheer size and complexity of the data which is delivered by the sensors monitoring such systems makes it very challenging for system engineers to detect abnormal behavior leading to failures. The two major challenges faced by the system engineers are: (1) the need to know what to monitor and (2) the ability to extract higher level information from raw sensor data in real-time.

It might be surprising for a laymen to learn that a trained professional does not know which parts of a machine should be monitored. It is, however, important to notice that a systems engineer always seeks to monitor interactions of components within a machine. Let us consider a following example: given a sensor monitoring an opening of a valve and a sensor monitoring the operations of a pump, an engineer can observe whether the time between the opening of a valve and the start of the pump operation remains stable or degrades over time. This, in turn, is an indication for an abnormal behavior leading to failures in the future. If we recall that a manufacturing machine contains hundreds of sensors, we can conclude that an engineer is faced with tens to hundreds of thousands of such correlations which he could monitor. The existence of such a big number of correlations makes it impossible, even for experienced engineers, to analyze them manually and detect anomalies timely.

HUGO helps systems engineers to cope with this issue by automatically selecting only meaningful component correlations to monitor. HUGO exposes a set of parameters to drive the selection process. These parameters can be easily adjusted by the system engineers to meet their desired level of insight. HUGO is also able to automatically derive component interaction diagrams with only minimum prior knowledge as to the setup of the machine. Component interaction diagrams represent direct feedback to the engineers tuning the selection process parameters.

Moreover, HUGO allows system engineers to perform real-time analysis of high velocity raw sensor data. Having automatically selected component correlations to monitor, HUGO uses a modified version of the HDDStream \textsuperscript{2} algorithm to group sensor readings within each correlation and monitor statistical properties of each group, focusing on the calculation of the long time trend. Combined with rules defined by the system engineers HUGO provides real-time alerting functionality allowing for a timely detection of anomalies.

We have implemented HUGO on top of a commercial Complex Event Processing (CEP) \textsuperscript{3} system and evaluated the proposed methods using a real world dataset \textsuperscript{4} coming from the manufacturing machine installed at the Infineon Technolo-

\footnotesize{\textsuperscript{1}Hugo Steinhaus was a Polish mathematician and educator, one of the first to propose the method of k-means clustering.}

\footnotesize{\textsuperscript{2}HDDStream is a real-time data stream mining system that automatically discovers correlations between sensors.}

\footnotesize{\textsuperscript{3}CEP systems are designed to process streams of data and automatically detect significant events.}

\footnotesize{\textsuperscript{4}The dataset is provided by Infineon Technologies GmbH.}

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

\textit{DEBS’13, June 29–July 3, 2013, Arlington, Texas, USA.}

Copyright 2013 ACM 978-1-4503-1758-0/13/06 ...$15.00.
In Section 4 we describe how HUGO uses work in Section 6. We conclude with Section 7.

Monitoring sensors produce either digital or analog signals. Analog sensors monitor aspects such as current flowing through and voltage of different parts of the machine. Binary digital sensors produce a state transition (from 0 to 1 or vice versa) whenever a processing action is taken by the corresponding machine component. Detailed description of the sensors can be found in [6]. To reduce the data processing complexity, we first convert analog signals into digital ones using user-specified rules. Without loss of generality, we consider in the following only digital sensors which produce just two different values: 0 and 1. Nevertheless, the proposed approach can easily be extended and applied to digital sensors producing multiple values.

### 3. TEMPORAL ANALYSIS

The goal of the sensor data analysis is to monitor whether the interaction between pairs of related components in the machine adheres to certain time regimes. More specifically, we inspect the interaction in terms of the delay between the processing actions taken by the related components. In this section, we describe in detail how sensors are correlated with each other, how the streaming data clustering method is applied to analyze the temporal dependency between machine components, and most importantly, how we tailor a state-of-the-art clustering algorithm HDDStream [8] to solve the specific interaction monitoring problem.

#### 3.1 Correlate Sensor Data for Analyzing

Interdependence between two related machine components is studied by correlating the sensor data from the two components.

Figure 2 shows an example of the signals from a pair of digital sensors over time. It also illustrates how the two sensors are correlated and analyzed. Each transition from the “off” state to the “on” state (the raising edge) of a sensor indicates an occurrence of a processing action. As described in Section 2, processing steps repeat for each production item processed by the machine. Therefore, we can observe a repeating state transition pattern in the signal series. To study the temporal dependency between two processing actions, each raising edge of Sensor A is correlated with the following raising edge of Sensor B in the same processing step. The delay ($\Delta t$) between these two state transitions is measured and subject to clustering. The same correlation procedure is repeated in each iteration of the corresponding processing step, producing a stream of delay measurements. It can be observed in Figure 2 that the first two delay measurements ($t_2 - t_1$ and $t_4 - t_3$) are almost equal, while the third one ($t_6 - t_5$) is significantly longer. As the result of clustering, the first two delay measurements will be assigned to the same

<table>
<thead>
<tr>
<th>Time</th>
<th>Sensor A</th>
<th>Sensor B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$t_2$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$t_3$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$t_4$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$t_5$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$t_6$</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
sensor (Cluster 1) while the third one will be assigned to a second cluster (Cluster 2).

Delay measurements are continuously fed into the clustering algorithm. After a certain training period (e.g. 24 hours), data points in clusters which contain only a few members (e.g. less than 10) can be considered as erroneous. This in turn, suggests an abnormal interaction between the corresponding pair of machine components. By means of clustering, system engineers are automatically presented only with the data which is already considered to be a manifestation of an abnormal behavior.

3.2 Tailored Clustering Algorithm

We leverage HDDStream, a density-based streaming data clustering algorithm proposed in [8] to perform data analyzing. We choose the density-based rather than the k-means-based clustering method, because k-means-based clustering requires to know the desired number of clusters (k) produced by the algorithm in the final solution, and assumes that the number of clusters remains constant. This assumption does not hold in real-world scenarios of constantly evolving data streams. In contrast, density-based approaches do not demand a priori knowledge of the number of clusters.

Because data streams are unbounded, it is impossible to maintain clusters by keeping all the historical data points. A widely adopted solution is to summarize sets of data points through an appropriate summarizing structure called micro-cluster [1]. A micro-cluster only keeps sufficient statistics of a set of data points, for instance, a number \(N\) which represents the total number of data points in the cluster, and a vector, of the same dimension of data points, which stores the linear sum or square sum of the \(N\) points. Single data points are not maintained by micro-clusters.

HDDStream introduces two types of micro-clusters to cope with the evolving feature of data streams: potential core micro-clusters and outlier micro-clusters. A potential core micro-cluster is a micro-cluster which collects more than \(N_{\text{min}}\) data points within a limited radius \(\epsilon\) in a projected subspace, while an outlier micro-cluster is a micro-cluster whose density has not reached the pre-specified density threshold. An outlier micro-cluster may collect more and more data points over time and evolve into a potential core micro-cluster when its density reaches \(N_{\text{min}}\). Vice verse, a potential core micro-cluster may also lose density over time and degrade to an outlier micro-cluster.

The reason for the density loss is that, to give a higher level of importance to the most recent data, each data point in HDDStream is assigned a weight via an aging function. The weight of a data point exponentially decreases with time \(t\) according to the function \(f(t) = 2^{-\lambda t}\), where \(\lambda(>0)\) is a user-configurable decay factor. For example, given the current time \(t\), the weight for a data point \(p_i\) which arrived earlier at \(t_i\) is \(2^{-\lambda(t-t_i)}\). Since data points are summarized by micro-clusters, the statistics maintained in the micro-clusters decays over time in the same way. Consequently, HDDStream employs a temporal extension of micro-clusters. Each micro-cluster at time \(t\) contains three statistic components for a set \(C\) of \(d\)-dimensional data points:

- \(CF1(t)\): a \(d\)-dimensional vector of the weighted linear sum of the points in each dimension.
- \(CF2(t)\): a \(d\)-dimensional vector of the weighted square sum of the points in each dimension.
- \(W(t)\): sum of the weights of the data points.

3.2.1 Representation of the Radius Threshold

The region covered by a micro-cluster in the dimensional feature space is determined by the current center of the micro-cluster and its radius. The radius threshold \(\epsilon\) defines the boundary of a micro-cluster. A data point \(p\) can be assigned to an existing micro-cluster \(mc\) iff the addition of \(p\) to \(mc\) does not affect the boundary of \(mc\).

For the machine monitoring scenario described in this paper, the target items to be clustered is the temporal delay between pairs of processing actions. Therefore, the data point has only one dimension. As a result, the functions given in [8] for calculating the center and radius of a micro-cluster can be simplified into the following form:

\[
\text{center}(mc) = CF1(t)/W(t)
\]

\[
\text{radius}(mc) = \sqrt{\frac{CF2(t)}{W(t)}} - \left(\frac{CF1(t)}{W(t)}\right)^2
\]

Note that \(CF1(t)\) and \(CF2(t)\) is now a scalar rather than a vector.

In HDDStream, the radius threshold of a micro-cluster is a parameter which can be specialized with an absolute value. In other words, all micro-clusters have the same maximal radius. Whereas, we have the observation that different pairs of related processing actions are usually sensitive to variations in the delay to a different extent. More specifically, if the normal delay between one pair of processing actions is around 10 seconds, then 1 second’s variance is already a critical event of which the system engineer needs to be notified. To that end the clustering algorithm must be able to mark a duration measurement with value 9 or 11 as an outlier instead of assigning it to the same micro-cluster as for the delay measurements with value 10. This implies that the radius threshold of a micro-cluster in this scenario must be smaller than 1. However, for another pair of processing actions whose normal delay is around 100 seconds, the deviation of 1 second is normal and tolerable. But if the radius threshold remains smaller than 1, delay measurements with value 100
We can observe that one raising edge of Sensor X is followed by two raising edges of Sensor Y. To describe the correlation between these two sensors, both delay measurements ($t_2 - t_1$ and $t_3 - t_1$) are required. Although the deviation between these two measurements may be tiny, they represent different interaction semantics and should not be assigned to the same cluster. To guarantee the correct correlation semantics, we assign each delay measurement a cluster. As a result, all micro-clusters of a certain sensor correlation are maintained in a hierarchical way. A set of micro-clusters is maintained under each class ID and no micro-cluster can belong to two different classes. In the current implementation, the class ID is a simple integer value which is incrementally increased according to the number of so-far encountered different edge correlations in a single iteration of the corresponding processing step. Note that this tuning to the HDDStream algorithm is introduced only to facilitate our sensor data analysis scenario. It does not change the principle of the clustering algorithm.

The example in Figure 3 also shows that there might exist multiple “regular” delay values for a given pair of processing actions. Indeed, the lower bound for the number of regular delay values is exactly the number of classes appeared in the given correlation.

Listing 1 gives the pseudo code of the revised clustering algorithm. The input to the clustering algorithm are delay measurements calculated based on the raw sensor data. Information contained in one data point $p$ includes the value of the delay measurement, a timestamp as well as a class ID. For each newly arrived data point, the algorithm first check whether it belongs to an existing class ($p.classID$). If not, a new outlier micro cluster $omc$ is initialized under the new class ID and the data point is assigned to it. If the class already exists, the algorithm tries to add $p$ to its closest micro-clusters in the class. If $p$ cannot be assigned to any existing micro-cluster in the class, a new outlier micro-cluster will be initialized for $p$.

The pseudo code of the add procedure in Listing 1 is shown in Listing 2.

The distance between a one-dimensional data point $p$ and a micro-cluster $mc$ is defined as follows:

$$dist(p, mc) = p - center(mc)$$

### 3.3 Analysis Procedure Summary

The overall data analysis procedure is outlined in Figure 4. In the preprocessing step, analog sensor signals are converted into digital signals using user-specified rules. Next, based
Listing 2: Pseudo code of the add procedure

```java
Input: a data point p, pCoreMC and oMC;
distances: list of distances w.r.t p;
for (mc in pCoreMC ∪ oMC) {
    dist = distance between p and mc;
    distances.add(dist);
}
if (!empty(distances)) {
    mc_closest = getClosestMC(distances);
    Trial = checkRadiusBoundary(mc_closest, p);
    if (Trial == true) {
        Add p to mc_closest;
        return mc_closest.ID;
    }
}
return -1;
```

Figure 4: Overall data analysis procedure

on the timestamps included in the sensor data, temporal delay between related processing actions is measured and analyzed with the online clustering algorithm. The outcome of online clustering is a set of micro-clusters. As described in Section 3.2.1, we tune the HDDStream algorithm to use a relative percentage instead of an absolute value to define the radius threshold. Because the radius threshold represents the maximum deviation that can be tolerated and it is derived from the use case itself, the online extracted micro-clusters can be regarded as final clusters, making the on-demand offline micro-cluster grouping step only optional.

4. SELECTION OF CORRELATIONS

A high-tech manufacturing machine is composed of a large number of mechanical components. However, not every component pair has a strong correlation in terms of the action delay because they may be involved in different, unrelated processing steps. Due to the high complexity of manufacturing processing steps and the machine itself, system engineers usually have little a priori knowledge about which components are indeed involved in the same processing step and thus worth monitoring. To tackle this issue, we propose a heuristic method for automatically selecting meaningful correlations using a training phase.

The rationale behind the proposed method is that if a pair of machine components has strong interdependence, then all delay measurements between the corresponding processing actions over time should form only a few densely populated areas in the domain of the delay value. Any measurement whose value does not fall into any of these areas can be regarded as an outlier. When considering the expected clustering result, we would expect the clustering algorithm to produce only a limited number of clusters which summarize the majority of the delay measurements. These clusters are defined as regular clusters. Clusters which contain only few members are basically formed by outliers and are defined as outlier clusters. In other words, a component pair is meaningful for monitoring only when one could easily distinguish outlier clusters from regular clusters based on its clustering result.

In the training phase, we first divide all sensors into groups based on, if any, priori knowledge about the manufacturing machine. For instance, a SABRE Electrofill system, which is widely used in the semiconductor industry, contains 3 electroplating cells for copper deposition and 3 postplating cells for wafer-postclean. Machine components in one cell do not interact with components in other cells. Therefore, sensors in the machine can be grouped based on cells and sensors in different groups will never be correlated with each other. If there is no such priori knowledge available for grouping sensors, all sensors will be put into the same group. However, grouping sensors based on priori knowledge is only a manual way to reduce the search space of the correlation selection procedure, thereby saving the computation resources. In the end the results with and without pre-sensor grouping will be identical.

For each sensor group the data analyzing approach outlined in Section 3 is applied to every sensor pair. However, we turn off aging. Moreover, due to the lack of knowledge about the actual ordering between the processing actions, both correlation possibilities (Sensor A to Sensor B and Sensor B to Sensor A) need to be studied. We now consider a single sensor group since the same method can be applied to any sensor group. We only investigate the correlation between different sensors. That is, given N sensors in a group, there are N×(N−1) possible correlation pairs. At the end of the training phase, the following information can be obtained for each correlation pair:

- number of generated delay measurements: \(N\)
- number of produced classes: \(l\)
- number of produced clusters: \(k\)
- number of delay measurements summarized by each cluster: \(N_c = \{N_{c_1}, \ldots, N_{c_k}\}\)

To find the most meaningful component correlations, we first sort elements in \(N_c\) in the descending order. The resulting list is denoted as \(N^1_c\). The lower bound for the number of “regular” delay values between a certain pair of processing actions can be derived from the number of produced classes. Ideally, there should be only one regular cluster in each class \((k = l)\), which is also the only cluster in this class, indicating that the corresponding pair of machine components interact without any exception. However, due to the existence of anomalies, usually the number of produced clusters is bigger than the number of classes \((k > l)\). Therefore, we introduce an error factor \(\rho\) to tolerate outliers existing in the dataset. The rule we used to determine whether a component correlation is meaningful for monitoring or not is as follows:

A component correlation is meaningful for monitoring iff:

1. \(k \leq \rho \cdot l\), where \(\rho \geq 1\), or
2. \(\sum (TOP_{\rho} (N^i_c)) \geq N\theta\), where \(\theta \in (0, 1]\)
$\rho \cdot l$ defines the upper bound of the expected number of regular clusters. Chosing a bigger $\rho$ means we allow more clusters produced for a given component correlation to be treated as regular clusters. This, in turn, makes it easier for the component correlation under inspection to pass the first condition. Parameter $\theta$, which we name as coverage parameter, defines the expected proportion of delay measurements that which covered by the regular clusters. For example, $\theta = 1$ indicates that all delay measurements of the given correlation have to be covered by the allowed number $(\rho \cdot l)$ of regular clusters. By setting a smaller $\theta$, we relax this “coverage” requirement, thereby making the second condition easier to be passed. In general, the above rule is stricter when $\rho$ and $\theta$ are set to a value close to 1.

Although this sensor pair selection procedure is mainly applied in the training phase, it can also be applied periodically at runtime to check whether the so-far selected correlations are still valid.

5. EVALUATION

To evaluate our machine health monitoring approach, we used a real-world dataset coming from one manufacturing machine located at the Infineon Technologies AG fabrication plant in Dresden, Germany. The manufacturing machine is monitored by 45 binary digital sensors as described in [6]. However, only 26 of them are active. All active sensors form 3 groups - see Table 1. The monitoring data is generated at the frequency of 100Hz. Each sensor has a unique identifier which is the same as the identifier of machine component being monitored. Each monitoring data is stamped by the time at which the data is produced.

<table>
<thead>
<tr>
<th>Sensor Group</th>
<th># Active Binary Sensor</th>
<th># Possible Correlation Pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group 1</td>
<td>10</td>
<td>90</td>
</tr>
<tr>
<td>Group 2</td>
<td>9</td>
<td>72</td>
</tr>
<tr>
<td>Group 3</td>
<td>7</td>
<td>42</td>
</tr>
</tbody>
</table>

Table 1: Distribution of active binary sensors in groups

In the experiments, raw sensor data is replayed using a custom data generator, which can preserve the original data producing frequency. Moreover, the generator also allows to configure a speedup parameter to further increase the data producing frequency. All the experiments are conducted on a machine with a two-core 2.26GHz CPU and 4GB RAM. We first evaluate the scalability of the proposed solution in terms of the system throughput and the clustering latency with increasing number of correlation pairs under monitoring. System throughput is defined as the number of events that can be processed by the system per second. We then study the influence of the parameter configuration on the result of meaningful correlation selection.

In the scalability test, we only use data from sensor group 1. We set the speedup parameter of the data generator as 100, thereby fixing the input data rate from each sensor at 10,000 events/s. We then vary the number of sensors ($n$) from 5 to 35 and assume that interdependence exists between every two sensors. Although group 1 has only 10 binary sensors, additional sensor streams can be simulated by inverting the sensor value\(^5\) or shifting the timestamps in an original sensor stream. The same dataset is replayed in each test run and the actual input rate is calculated by multiplying 10,000 with the number of sensors being monitored (10,000 $\times n$). The system throughput is then measured under each simulated workload and compared with the actual input data rate. It can be seen from Figure 5 that the system reaches its throughput capacity when monitoring 870 correlation pairs (30 sensors) simultaneously.

Under each simulated workload, the time needed to cluster the correlation samples (clustering latency) is also measured. We select a certain correlation pair which is a member of all the correlation pairs in each test run, and measure the clustering latency for 100 consecutive correlation samples starting from a fixed data time (12 hours after the time represented by the timestamp of the first raw sensor data in the dataset). Latency measurements are summarized and shown in Figure 5 using boxplot. It can be seen that in general the clustering latency is very small when the system is not overloaded, and starts to increase after the system reaches its capacity.

\(^5\)Change value 0 to 1 and vice versa.

5.3. Evaluation of Correlation Selection

As shown in Figure 5, with a given amount of computation resources and under a certain data rate, there exists an upper bound on the number of correlation pairs that can be monitored by the system.

With naive correlation strategy, a lot of computation resources is indeed wasted since not every component pair is meaningful for correlation. To find out the most meaningful component correlations, we apply the method described in

- Figure 5: System throughput and clustering latency
- Figure 6: Influence of parameter setting on the correlation selection result
Section 4 to 24 hours' sensor data from all 3 sensor groups. Due to the existence of outliers in the given dataset, the number of selected correlation pairs varies with different parameter ($\rho$ and $\theta$) configurations. The stricter the selection condition is, the higher possibility that one of the filtering condition is violated, thereby resulting in a less number of component correlations to be selected by the algorithm. Figure 6 shows the influence of the parameter setting on the selection result for sensor group 1.

One thing that is worth explaining is, even with the most relaxed parameter configuration ($\rho = 100$ and $\theta = 0$) the total number of selected correlations is 45 but not 90 (the number of all possible correlation pairs in sensor group 1). This is because, during the training period, the occurrence of the other 45 correlation pairs was not observed. For example,
in each repeating processing step, a signal from sensor pp08 always appears after the signal from sensor pp04, but not the other way around, which is similar to the scenario shown in Figure 2. Therefore, there is only the correlation from pp04 to pp08 but not the correlation in the other direction. Provided with this parameter influence chart shown in Figure 6, system engineers can tune parameters of the algorithm ($\rho$ and $\theta$) to tolerate the existence of outliers in a given dataset.

In addition, for each parameter configuration, we can derive a component interaction diagram for each sensor group based on the selection result. This diagram could help system engineers to understand how machine components interact with each other within the machine. Figure 7a shows the component interaction diagram for sensor group 2 with $\rho = 2$ and $\theta = 0.3$. The number of component correlations appearing in the diagram is given in the brackets below the diagram. Each circle in the diagram represents a machine component. An arrow from component X to component Y indicates that component Y directly depends on component X temporally.

As mentioned above, the number of selected component correlations is influenced by the parameter settings in the filtering rules. We show in Figure 7, 8 and 9, each of which contains a series of 3 component interaction diagrams, how the component interaction diagrams change with the parameter settings. In Figure 7, the error factor $\rho$ is fixed and the coverage parameter $\theta$ is changed stepwise from 0.3 to 0.7 and 1. Figure 8 shows the results when $\theta$ is fixed and $\rho$ is changed from 2 to 5 and 10. Results for changing $\rho$ and $\theta$ at the same time are shown in Figure 9. In all three figures, we see that the component interaction diagrams evolve in a consistent way when we relax or strengthen the selection conditions. Take Figure 8 as an example, when $\rho$ is changed from 2 to 5, more action dependencies between machine components are revealed, but all the existing dependencies shown in Figure 7b can still be found in Figure 8b. Similar behavior can be observed when strengthening the selection conditions: only existing action dependencies are filtered out and no new dependencies can be detected - see Figure 7 and Figure 9.

The meaningfulness of these component interaction diagrams have been verified by the Infineon engineers working with the machine.

6. RELATED WORK

Equipment and machine process monitoring by means of sensor data analysis is not a new concept. In [2], authors propose IDES, a diagnosis and control system for mechanical systems. IDES is based on constructing influence diagrams, which describe the relationship between a potential failure and the signal changes of a monitoring sensor. IDES, in contrast to HUGO, is not designed to monitor the temporal component interactions. Authors in [9] present a hierarchical fabrication plant-wide control strategy for semiconductor manufacturing process. Sensors are used to collect metrology data of the wafers that are processed on the high-tech manufacturing equipment. The collected data is then analyzed using the principle component analysis approach. The failure detection is based on evaluation of quality of the product itself, which is different from the component interaction monitoring approach taken by HUGO. In [4] authors present an oil processing plant behavior analysis model SDAEM, whose goal is to analyze time series data such as temperature and pressure monitored by sensors deployed in the plant. Individual sensor data is analyzed by detecting patterns in historical data using data mining techniques and searching for patterns in real-time data. Similarities between the signal trends of different sensors are detected using the detrended cross correlation method. The approach of HUGO is different from the one of SDAEM in that HUGO detects correlation between machine components which cannot be achieved by comparison of the signal trends. There also exist numerous review papers regarding manufacturing processing monitoring and machinery diagnostics [5, 3]. None of these approaches aims at analyzing the temporal dependencies between related machine components.

7. CONCLUSION

In this paper we have presented HUGO – a system which enables systems engineers to get real-time insight into relevant high-tech manufacturing machine component interactions. HUGO achieves this goal by applying a tailored version of HDDStream [8], a density-based clustering algorithm, on top of the sensor data originating from the manufacturing equipment. The results of clustering are used as the input to the algorithm which automatically filters out meaningless component interactions. HUGO can be easily applied to any type of high-tech machines, providing that they allow for real-time sensor data collection.

We have evaluated HUGO using real world data originating from one of the machines located at the Infineon Technologies AG fabrication plant in Dresden, Germany.

8. ACKNOWLEDGMENT

The authors wish to thank Infineon Technologies Dresden, for providing real-world machine monitoring sensor data. We especially thank Infineon engineers working with the manufacturing machine, for constructive discussions during the work and their feedback to our evaluation results.

9. REFERENCES


