Manufacturing Process Optimization Using Edge Analytics

Louis Nunzio Loizides

Marquette University

Recommended Citation
https://epublications.marquette.edu/theses_open/435
MANUFACTURING PROCESS OPTIMIZATION USING EDGE ANALYTICS

By

Louis Loizides

A Thesis submitted to the Faculty of the Graduate School
Marquette University
in Partial Fulfillment of the Requirements for
the Degree of Master of Science

Milwaukee, Wisconsin
August 2017
Most manufacturing plants contain some amount of time series sensor data – streams of values and time stamps. This data, however, isn’t useful with most types of analytics or machine learning for the purpose of process optimization. This thesis presents a novel and innovative solution to the problem using a software stack leveraging the Predix Complex Event Processing Engine (Edge Analytics) to condition the data, combined with RFID for serialization. Each step in the formation of the solution is documented, from connecting equipment to analyzing and ingesting data produced by the edge analytic. This solution was developed and piloted at the GE Grid Solutions plant in Clearwater, FL.
TABLE OF CONTENTS

1 INTRODUCTION .......................................................................................................................... 1
   1.1 The Clearwater Plant ............................................................................................................. 2

2 MATERIALS AND TECHNOLOGY .............................................................................................. 4
   2.1 System Components ............................................................................................................. 4
   2.2 Protocols ............................................................................................................................. 6
       2.2.1 OPC .......................................................................................................................... 6
       2.2.2 MQTT ....................................................................................................................... 8
   2.3 RFID Implementation ............................................................................................................ 10
       2.3.1 Implementation Functional Requirements ................................................................. 12
       2.3.2 Zoning ....................................................................................................................... 14
       2.3.3 RFID Protocols and Events ....................................................................................... 17
       2.3.4 RFID Printer ............................................................................................................. 18
   2.4 Edge Analytics ..................................................................................................................... 18
   2.5 Historian ............................................................................................................................. 20

3 ARCHITECTURE AND COMPONENTS ....................................................................................... 23
   3.1 Network Security Needs ...................................................................................................... 23
       3.1.1 Wurldtech .................................................................................................................. 24
       3.1.2 Network Address Translators .................................................................................... 24
   3.2 Middlewares ....................................................................................................................... 25
       3.2.1 RFID to MQTT Middleware ...................................................................................... 25
       3.2.2 Software Filtering Techniques .................................................................................. 26
       3.2.3 MQTT to SQL Bridge .............................................................................................. 31
   3.3 Edge Analytics Platform...................................................................................................... 33
6.1 Predictive Modeling Challenges ................................................................. 78
6.2 Solution Transferability to Other Plants ..................................................... 80
6.3 Results ......................................................................................................... 81
  6.3.1 Initial High Voltage Winding Use Case Findings ................................. 81
  6.3.2 High Voltage Dielectric Yield Predictors ............................................ 82
  6.3.3 Low Voltage Insights .......................................................................... 85
6.4 Minds + Machines ..................................................................................... 86
6.5 Sensor Data Control Charts ..................................................................... 86

7 Appendix ...................................................................................................... 88
  7.1 Detailed SQL Schema Descriptions ....................................................... 88

BIBLIOGRAPHY ............................................................................................. 91
DEFINITIONS

- CEP (Complex Event Processor): The name of the Predix Edge Analytics product
- Cloud: A distributed, highly scalable architecture for software applications.
- COQ (Cost of Quality): The cost of quality excursions, measured in dollars. Generally caused by scrap or rework due to manufacturing yields lower than 100%.
- CTQ (Critical to Quality): A measureable characteristic that we believe has an impact on part quality
- DMZ (De-militarized zone): an isolated space on a LAN with minimal security due to its isolation
- Edge: In cloud architecture, the edge is the location of the data being consumed.
- EPCGlobal: An RFID industry association that develops standards for RFID readers and tags.
- FIFO: First-In, First-Out
- Fog: Edge computing controlled by the cloud.
- GUID (Globally Unique Identifier): A random string that’s expected to be statistically unique.
- HMI (Human machine interface): The device that allows an operator to control a machine
- Inventory: Materials kept in order to build products
- IoT (Internet of Things): A term referring to the connecting of devices
- IT (Information technology): Generally refers to corporate supported applications and infrastructure
- JSON (JavaScript Object Notation): A way of representing key-value data in the form of JavaScript objects
- Ladder Logic: A programming language commonly used with PLCs. Ladder logic is meant to emulate pre-PLC hard wired relay logic and the code is written on a series of rungs.
• LAN: Local area network
• LLRP (Low Level Reader Protocol): An RFID reader protocol created by EPCGlobal that most modern RFID readers comply with.
• MES (Manufacturing Execution System): An application used to provide workflow information and guide various manufacturing processes throughout the plant.
• MVP (Minimum Viable Product): The minimum feature set software application that can be used by a customer. MVP is an agile driven methodology that aims to release middleware with minimal feature sets as quickly as possible (MVP1, MVP2, etc.).
• OPC (Object Linking and Embedding for Process Control): A standard protocol and architecture for process data
• Operator: The person that executes a manufacturing process
• OT: (Operational technology): Generally refers to plant supported applications and infrastructure, including those running on machines.
• REST (Representational State Transfer): A standard for transmitting API data over a web interface where each URL is a representation of an object.
• RFID (Radio Frequency Identification): A standard for performing serialization using small transponders that can be written and read to through the use of RFID readers.
• RSSI (Received Signal Strength Indication): A metric indicating the relative signal strength of tags in a radio based serialization system (ex. RFID or Bluetooth).
• SQL (Structured Query Language): A protocol for sending information to and from (usually) relational databases. Commonly used to refer to the databases themselves.
• Time Series Data: A collection of values versus time stamps.
• UTF8: A string encoding scheme that can represent any number of characters through the use of ASCII codes.

• VLAN (Virtual Local Area Network): An isolated portion of a network. Frequently containing a DMZ.

• WIP (Work In Progress): The amount of unfinished product that exists on the shop floor at any time. WIP is essentially borrowed money and is kept to a minimum through the use of lean principles.

• Yield. The percentage of product that passed or failed a process. In this paper the yield referenced is usually at a final product test.
Good manufacturing operations are customer focused. Customer priorities can be categorized as two classifications – quality and lead time. In other words, customers want both a high quality product, and want it as quickly as possible. A plant with a customer focus must keep this in mind.

In addition to this, the manufacturing plants themselves have the same shared interests as their customers. The cost of product quality excursions can be significant and needs to be minimized (Mantri & Jaju, 2013). Furthermore, any business would prefer that the cash used to fund WIP and keep inventory could be invested in other projects. As a result, good manufacturing plants want to build and ship product as quickly as possible. In fact plants often rate themselves in how much WIP exists in the plant at any given time (Goodson, 2002) and common methodologies like lean are primarily aimed at reduce WIP.

One way plant leadership can increase quality and reduce WIP is to optimize the parameters used during a manufacturing process. Optimizing a process requires obtaining large, traceable data sets in which various process characteristics can be correlated to desired outcomes like process times and product quality. Once these relationships are established, various modeling techniques can be applied, and optimal operating parameters can be determined that improve yield (Capodieci, 2017).

Several solutions for generating process characteristics from time series data have been proposed, such as automatically generating these characteristics based on comparisons to sample data (Data, 2005). These automatic methods, however, suffer from inaccuracies due to the exclusion of expert knowledge in the data conversion process and a lack of automatic traceability. In order to achieve fully traceable data from manufacturing processes, I have developed a solution using
RFID and barcoding to provide serialization, combined with various edge analytics to contextualize process data around that serialization. In addition, an application built on GE’s Predix platform serves as a tool for analysis. I have also worked closely with IT security to architect a solution with both IT and OT security in mind.

1.1 The Clearwater Plant

This solution described in this document was piloted at the GE Capacitors and Transformers Plant in Clearwater, FL. It was designed using modular components that could be easily translatable to other plants.

The Clearwater plant builds industrial capacitors that are most commonly used in electrical substations or on top of utility poles. These capacitors absorb excess electrical energy from the grid and release that energy when needed. They essentially serve as buffers for excess electrons, and also for changing the power factor in a grid.

The plant is approximately 175,000 square feet and opened in early 2016. The advantage of using Clearwater as a pilot was due to it having new machines and infrastructure. Not only were these easier to get online, but new machines and processes also provided new optimization opportunities that hadn’t previously existed.
Figure 1 - The GE Grid Solutions plant in Clearwater, FL

Figure 2 - A cross section of a Clearwater industrial capacitor

Note that the Clearwater plant is technically part of Instrument Transformers Incorporated, which is a wholly owned subsidiary of GE Grid Solutions.
2 MATERIALS AND TECHNOLOGY

2.1 System Components

A high-level system architecture diagram is below in Figure 3.

![High Level Solution Architecture Diagram]

Figure 3 - High Level Solution Architecture Diagram

The in house portion (“OT”) of the solution described in this document is a collection of modular components forming a gateway with no persistent back end.

The reason for limiting the solution to a gateway was to keep the solution simple, componentized and maintainable. Cloud solutions are serviceable by a central team, but edge solutions are not and manufacturing plants generally have very little IT support. Keeping the edge component of the solution minimal significantly reduces opportunities for bugs. Furthermore, each individual
component is easily swappable with other components that can serve or consume OPC-UA or MQTT data.

- **RFID Middleware**
  This is a service that consumes events from RFID readers, and republishes those events to an MQTT broker. The Middleware also contains functionality for managing readers and filtering reader events before republishing them. This middleware is described in depth in section 3.2.1.

- **SQL to MQTT Middleware**
  Barcode scans are inserted into an SQL table in the plant’s MES system, and in some cases barcode events are preferable to RFID for serialization. This middleware is a service that subscribes to those SQL inserts and publishes the inserted data to an MQTT broker so that it can be consumed by an edge analytic. The middleware is described in section 3.2.3.

- **MQTT Broker**
  MQTT is a publish/subscribe protocol, which is ideal for communicating discrete events such as barcode or RFID scans. MQTT is implemented with a broker-client architecture, so a broker must be set up in order for clients to send and receive messages. The use of a broker is an advantage for brilliant factory as a single port on a firewall can be opened up to allow messages to travel from a middleware on one side of a firewall to a broker on the other side.

- **OPC Server**
  The OPC server consumes data from various sensors and devices, specifically PLCs, and provides them for consumption through the OPC protocol. We’re using the GE IGS OPC server, which based on the Kepware OPC server.

- **Predix Machine + Foghorn Edge Analytic**
  The edge analytic consumes data from MQTT or OPC sources, and provides real time event-based processing of data. This processing is used to contextualize data. The Predix CEP edge
analytic engine is built by Foghorn and runs in a Docker container on Predix machine, which is GE’s Predix solution for cloud to device connectivity.

• **Predix**

Predix is GE’s cloud-based platform used to develop applications for industrial data. All of our back end and reporting tools exist on Predix. The first iteration of a Predix tool I developed with GE Digital is described in section 5.1.

### 2.2 Protocols

#### 2.2.1 OPC

##### 2.2.1.1 Legacy OPC Standards

OPC stands for OLE (Object Linking and Embedding) for Process Control. OLE is a standard developed in the 1990’s that uses Microsoft Windows COM technology for publishing and subscribing to real time equipment data (Foundation, OPC Common Specifications, 2002). The goal of OPC is to take disparate sources of equipment data, such as PLCs, sensors, etc. and convert that data into a standard format.

OPC solutions are constructed on a client-server model. Servers pull data from devices. Clients consume data from servers and also specify polling rates for devices (the frequency that data updates need to be provided at). Because many commercial OPC servers also have client drivers, OPC servers can usually be linked together with the top level client controlling data polling down to a device.
2.2.1.2 Newer OPC Standards

The original distributed COM based OPC standard has since been designated OPC-DA for “OPC-Data Access”. OPC has evolved into a toolkit of several different technologies. OPC-DA, for instance, now has an XML variant that can publish and subscribe data through REST methods. OPC-UA (OPC-Unified Architecture) is another, newer variant that uses a single port and SSL certificates for security. One goal of OPC-UA is eliminate reliance on Windows.

For our purposes we’re using OPC-UA. OPC-UA is far more secure than OPC-DA as fewer ports need to be opened through our equipment firewall, and it also uses modern SSL certificate based authentication. Furthermore, both our edge analytics and Predix Machine are built on Linux and as a result it can’t use the older COM-based standards which require the Windows operating system. OPC-UA also has support for more advanced features such as different timestamps for device and server, vendor specified data types, and address spaces for nodes (Foundation, OPC UA Part 1: Overview and Concepts Release 1.03, 2015).

OPC maintains real time connections to each sensor stream through its client server architecture. In OPC-DA these streams are referred to as tags. In OPC-UA they’re generally referred to as nodes. Each tag or node contains the following fundamental information:

- **Value**: The value of the node or tag at that time
- **Timestamp**: By default this is the equipment timestamp, but can also be overridden by the server. Note that OPC will not usually transmit a new value unless that value has changed, so the time stamp is generally the last time the value was changed and not the last time the value was read. Furthermore, not all devices contain a timestamp – in the case of many PLCs the time stamp needs to be explicitly coded.
• Quality: This is a flag for whether the value can be trusted or not. If the device was read and the value was updated, this will be marked as good (true). If not the value will still be transmitted but it will be marked as bad (false).

![Figure 4 - OPC Tags viewed through a Client](image)

2.2.2 MQTT

The OASIS MQTT protocol is a light weight message transport protocol that it facilitates publish/subscribe notifications (OASIS, 2014). Publish/subscribe messaging notifications generally come in two flavors – peer to peer and client and server. MQTT is client and server, which means that it requires a central broker to moderate all data. ZeroMQ, on the other hand, would be an example of a peer to peer messaging protocol.
The reason that I’m using a client and server model is because all of our sources of contextual and equipment data are behind a DMZ (de militarized zone) on a virtual LAN. By having a broker with a static IP we’re able to open up a single firewall rule between each middleware and our broker I can allow devices outside of the DMZ to publish and subscribe to the same MQTT message stream.

Because MQTT is an open standard, many open source client libraries exist. For our RFID middleware and services we’re using the C# Micro MQTT library by Paolo Patierno (Patierno). This library provides a very simple MQTT implementation using the following code:

```csharp
MqttClient client = new MqttClient(hostName);
MqttClient client.ConnectionClosed +=
    new MqttClient.ConnectionClosedEventHandler(client_ConnectionClosed);
String clientId = Guid.NewGuid().ToString();
client.Connect(clientId, username, password);
client.Publish(topic, Encoding.UTF8.GetBytes(message),
    MqttMsgBase.QOS_LEVEL_EXACTLY_ONCE, retain);
```

In this case, the following items must be specified:

- **Client ID**

  Each client has a unique ID. In our case we’re generating a random ID using a GUID.

- **Topic**

  A topic must be provided to publish an event to. The topic is essentially an address that other clients can subscribe or push messages to. Topics are created and accessed by clients.

- **Quality of Service**

  MQTT requires that each message publish or subscription has a QOS (Quality of Service) level defined. Quality of service describes whether or not the message will transmit under certain situations. Quality of service zero means that the message is sent, but it’s lost if it doesn’t make it to the destination. Quality of service 1 means that the message will be resent by the broker until
the client receives it, but that the client will receive it only once. Quality of service 2 means that
the message can be sent and received several times to make sure it arrives.

- **Payload**

Each message has a binary payload, the contents of which are arbitrary. In our case the payload is
a UTF8 JSON string.

### 2.3 RFID Implementation

RFID is a standard for encoding small amounts of data on transponders or beacons. When a
transponder is placed in the proximity of an RFID reader and antenna, the reader can read the data
on the transponder. The reader can also write new data to the transponder. Note that most
documents refer to transponders or beacons as tags. I generally prefer to use the term transponder
in order to distinguish between the RFID component and the paper hang tag the transponder is
placed on.

![Figure 5 - Paper Hang Tags With and Without RFID Transponders Attached](image)

---

*Note: The image shows examples of paper hang tags with and without RFID transponders attached.*
Several variants of RFID exist, including:

- **HF (High Frequency)**

  HF has short read distances of no more than a few feet and is generally used for on contact RFID reading like ticketing or badging (Intermec, 2007).

- **UHF (Ultra High Frequency)**

  UHF transponders can have much larger read distances. Passive transponders can be read from as much as 20-30 feet away and active transponders can be read from more than 100 feet away (such as those used in highway tolling systems) (Harmon, 2010).

- **Active Transponders**

  Active transponders contain their own power source, usually in the form of a battery, and can send much stronger signals back to the reader. Active transponders can take the form of beacons, which continually broadcast a location, and transponders which return information in the presence of a reader.

- **Passive Transponders**

  Passive transponders are powered by the device reading the tag. These transponders are generally much less expensive than active transponders, but also have much shorter read range as they’re powered by the reader. They can be as inexpensive as a few cents each and don’t have a power source that can eventually run out. Passive transponders work by receiving a wave from an antenna, modulating it to contain some data, and then bouncing the modulated wave back to the antenna.

Furthermore, in addition to RFID, several other types of real time location and transponder systems exist for this type of serialization – most notably Bluetooth and Ultra-Wideband. Each technology has its own benefits and tradeoffs. RFID was chosen for this solution due to its long history in manufacturing, high temperature compatibility and well established standards between manufacturers.
2.3.1 Implementation Functional Requirements

2.3.1.1 Impact to Manufacturing

One requirement of our RFID implementation was that the implementation had to be completely passive to the existing manufacturing process in order to drive adoption. I didn’t want to force any specific changes that could potentially add manufacturing time, and, therefore cost to our product.

2.3.1.2 Transponder Selection

In our implementation we’re using passive UHF transponders. One reason we’re using passive transponders is because most of our read distances are in the 2-10 foot range and because we’re attaching the transponders to disposable paper tags, making active tags cost prohibitive. The other reasons are due to our process and environmental requirements.

2.3.1.3 Process and Environmental Requirements

Each manufacturing process has its own distinct process and environmental requirements. RFID transponder selection depends on these requirements. In the case of Clearwater, several process and environmental considerations were found to be important:

• Several processes involve serializing capacitor “rolls” created out of aluminum sheets with Mylar insulators. RFID transponders generally experience signal loss once they’re applied very close to metal, so serializing the roll without some kind of mechanism separating the transponder from the roll wasn’t possible
• The maximum temperature process is well in excess of 100 °C, and the transponders would have to be at that temperature for as much as 24 hours. This automatically disqualifies
active tags as batteries that can survive these temperatures for this long of a time are rare and prohibitively expensive.

The first consideration regarding serializing rolls was a process one, and as a result we could define our serialization process around this consideration. The one station where RFID was required was in the winding process. I found that operators, while winding, would place the tag for the pack at a specific location on the machine. As a result I placed the antennas in winding to observe this one location.

![Antenna and Tag with Transponder in Winding](image)

**Figure 6 - RFID Reader and Transponder in Winding**

The second consideration, regarding the maximum temperature, involved significant testing of various tag options from several manufacturers. RFID transponders contain an integrated circuit chip, and that chip must generally be kept below a maximum temperature (most commonly 85 C). After testing more than 50 tags, we found that the 53mm Frog 3D tags from SmartTrac reliably
survived the high temperatures. We ran 20 Frog 3D tags through our process 20 times each with satisfactory results in order to prove that they would survive the process with a statistically significant confidence.

The Frog 3D tags have another added benefit – the signal strength of these tags doesn’t vary much with read angle (Smartrac, 2013). In the case of most RFID tags the signal strength varies significantly as the tag is rotated. This can make RSSI based filtering challenging in a situation like ours where tag location isn’t controlled due to our desire to not change the manufacturing process. With the Smartrac tags, therefore, the distance between tag and antenna becomes the more significant RSSI driver and differentiating between antenna zones becomes easier as a result.

2.3.2 Zoning

With an RFID implementation we generally want to know if tags are in very specific areas, which I refer to as zones. These zones are bounded to some degree by an antenna or reader, but are generally a composite of multiple antenna fields of view and filtering definitions.

For our implementation each RFID zone would have to be distinct from each other (ex. process in, process out). But because RFID is a radio protocol the only way to achieve distinct zones is by physically blocking one zone from another. This isn’t practical in most manufacturing environments where processes are large and have to be accessible to operators.

Aside from physical barriers where possible, three strategies have to be used in conjunction with each other to ensure proper zoning:

- **Adjusting the antenna power and filtering by RSSI (transponder signal strength).**
• **Placing antennas in locations that have line of sight to transponders.**

If line of sight doesn’t exist transponders can frequently still be read due to signal reflections from surrounding areas. But because the path of a reflected signal can reduce the RSSI, the lack of a line of sight can reduce the RSSI of a closer transponder and increase the RSSI of a more distant one. In the case of Figure 7 below, the RSSI between Tag A and Antenna B can be greater than that between Tag A and Antenna A due to the obstruction between Tag A and Antenna A. In this case, if we were to RSSI based filtering techniques to zoning, our RFID implementation would incorrectly associate Tag A with Antenna B.

![Figure 7 - Effect of Reflections on RSSI](image)

• **Use of extra antennas to accurately shape zones.**

Having more antennas allows us to more accurately assess a transponders’ location and allows us to more accurately shape zones. For instance, in Figure 8 the correct zone from Antenna A should only cover the conveyor, but this isn’t possible as radiation is emitted in all directions. Placing a second antenna nearby allows us to remove the portion of the zone that overlaps, allowing us to
more approximately shape the conveyor zone. Note that zone shaping isn’t only necessary due to the ideal shape of the zone – it’s also necessary because in many cases in industrial environments we have to turn up the signal strength in order to get a reliable read. This technique allows us to use higher signal strengths while still keeping zones confined to a small area if necessary.

Figure 9 shows an example of an antenna that was moved to improve line of sight and barriers that we installed to block the RFID signal from bleeding across zones. We originally wanted the antenna above the capacitor as we didn’t control tag placement and felt that the ceramic bushing could obstruct the tags if we moved the antennas behind them. We found out, however, that the ceramic bushings had almost no effect on tag strength. And a set of steel pipes between the antenna and capacitors was causing intermittent bad reads. As a result we moved the antennas below the pipes, and we also added thin aluminum panels to help prevent bleed across zones.
2.3.3 RFID Protocols and Events

Most RFID readers comply with a protocol that allows a user to connect the reader and consume events generated by it. Siemens, for instance, uses its own XML based Simatic protocol (Siemens, 2010). Most readers also comply with a lower level LLRP protocol developed by EPCglobal, an RFID industry consortium (EPCGlobal, 2010).

RFID readers, either through interpretation of LLRP data or through their own proprietary protocols, can generate events. The two fundamental events that can occur with a reader are that a tag goes into an antenna’s field of view or that a tag exits an antenna’s field of view. Siemens refers to a tag going into a field of view as “observed” and a tag existing a field of view as “lost”. Because my initial work was with Siemens readers I generally use the same terminology. Also note that events are generated on a reader level and a reader’s field of view is a composite of the field of views of all attached antenna antennas. The specific antenna that sees the tag is usually communicated in any event data.
2.3.4 RFID Printer

Clearwater was using a Zebra thermal transfer printer for its paper hang-tag (linen tag) printing. See Figure 6 for an example of hang-tags with and without RFID.

In Clearwater’s pre-RFID printing implementation, server would send text files with tag information, including serial numbers, to a server running BarTender software. We ordered a newer RFID-capable R110Xi4 Zebra printer that had the ability to write to RFID transponders while printing to paper tags. We simply added an RFID writing function block to BarTender that would allow the printer encode our transponders while the hang-tag printed.

Note that the printer is used for convenience here as we already had the paper tag printing functionality implemented. In another implementation for a GE Transformers Plant we’re writing serial numbers to permanently attached tags using handheld RFID readers. The handheld reader has a barcode scanner that can read the serial number off of an existing traveler and then write that serial number to a tag on a nearby fixture.

2.4 Edge Analytics

Modern industrial software trends are migrating towards cloud architecture (Columbus, 2013). The main benefit of the cloud is scalability, but modular cloud platforms like Cloud Foundry and Predix also provide other benefits such as being able to update applications without shutting them down.

Cloud architecture is a great way to keep a manufacturing data collection system scalable and supported. But in a manufacturing environment, a full cloud implementation suffers from two key challenges:
• **The need to send large amounts of data to the cloud for processing**

Analyzing a single process can require multiple sensors. These sensors can change in value continuously, which generates a very large amount of data. In order to analyze this data in the cloud, it all needs to be sent there before analysis. This requires a large amount of bandwidth and isn’t possible for older manufacturing sites with limited infrastructure. Getting this data to the cloud and saving it there is expensive for any plant.

• **Delay tolerance**

Data sent to the cloud can be delayed, which is why most edge to cloud connections include some capacity for store and forward. Furthermore, because cloud infrastructure is shared it isn’t always practical to analyze data in real time. As a result insights can be delayed. Such delays are unacceptable in cases where real time analysis is needed, or if immediate insight is needed into a real time manufacturing process (such as the case with SPC).

In order to facilitate real time processing, therefore, an on-site analytics solution can be deployed in the plant. This solution can be an edge analytics solution (meaning it runs at the data source but is managed by a “cloud” component), an MES solution, or even reprogramming the PLC. We compared and ranked these options side by side with a cloud-based rules type of analysis.

<table>
<thead>
<tr>
<th>Factor</th>
<th>Rank</th>
<th>Cimplicity</th>
<th>Rules Based/Cloud</th>
<th>PLC</th>
<th>Edge Analytics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cloud Managed</td>
<td>5</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Known Language</td>
<td>3</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Predix Integration</td>
<td>1</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>MQTT Support</td>
<td>3</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>HDFS Support</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Can Control Machines</td>
<td>3</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Optional Adv. Scripting</td>
<td>3</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>
The edge analytics solution is the clear winner out of all options compared. One of the key benefits of edge analytics is to provide the best of both worlds – a cloud back end with on-site processing. While the edge analytics solution we’ve implemented in Clearwater does perform processing on site, it’s fully managed through a cloud interface. This creates a pseudo-cloud architecture generally referred to as a “fog” (Cisco, 2015). The edge analytics we’re using are integrated into Predix, but the analytics engine is developed by Foghorn, a startup company in Mountain View, CA.

Another use for edge analytics is to reduce the amount of data sent to the cloud. In our case, when monitoring a manufacturing process, an edge analytics implementation might need to monitor hundreds or thousands of data streams. If the analysis is performed in the cloud then all of those streams will have to be transmitted to and stored in the cloud as well, consuming bandwidth and other resources. By analyzing this data on site and only sending the result we’ve significant reduced the amount of data transferred to a small fraction of the original time series data volume.

### 2.5 Historian

Machine data is natively in time series format – a stream of values vs. time stamps. As a result time series data is generally not stored in a relational database. It’s stored in a “historian” database that saves individual streams of data by timestamp and value. New value and time stamp
pairs are only saved when a value changes. In many cases these streams also include a quality value in order to conform to the OPC standard.

In our case the Foghorn installation includes an Influx historian. Influx can be written to and queried from using a REST API. In addition, it allows us to create a relational-like table by adding key-value pairs to time series data points, which is an unusual feature for a historian. It also allows us to attach key-value pairs to those records.

Influx also has various powerful options like data retention policies that allow users to delete or down sample data after some period of time. This is a very valuable feature for use cases like ours where we might want to maintain high resolution data for some period of time, and then down sample it. While the Influx historian exists only for data backup and convenience, connecting Influx to the open source Grafana application creates a simple and powerful time series data visualization stack.
Figure 10 - Influx Time Series Data Viewed Using Grafana
3 ARCHITECTURE AND COMPONENTS

3.1 Network Security Needs

Our implementation contains various devices – sensors, PLCs, PCs, RFID readers, etc.

Connecting these devices can be very risky for several reasons:

- PLCs and sensors in a plant can control machines. These machines to be large pieces of manufacturing equipment and can be exceptionally dangerous. We have to build security that minimizes the risk of someone gaining control of these machines, and also prevents potentially hazardous situations in the event of network connectivity problems.

- Small devices like sensors and RFID readers contain computers that usually have no virus or malware protection due to limited processing power. These devices could be easily hijacked for malicious purposes like gaining access to a company network or building a botnet.

- PCs on an equipment network are usually used as HMIs (Human machine interfaces). These devices frequently contain no virus or malware protections in order to avoid complications with control systems. Automatic updates are usually turned off for the same reason. And because firewalls tend to restrict the complex communication between devices that are required for HMIs to function properly, they tend to be disabled as well.

Because of these risks, we chose to carve out a DMZ (de-militarized zone) on our network for these types of devices. Communication to/from the DMZ is explicitly defined by port and IP, and approved on a case-by-case basis.

To add further protection, we have two tiers of DMZs. One level of DMZ can communicate to the outside GE network and will contain devices such as the OPC server and MQTT broker. The second level DMZ can only transfer data to/from the first level of DMZ.
3.1.1 Wurldtech

We have also implemented Wurldtech Industrial network security devices on the Clearwater LAN. These security devices are smart devices that monitor network traffic for known industrial network vulnerabilities. Wurldtech was acquired by GE, which facilitated the pilot in the Clearwater plant.

3.1.2 Network Address Translators

A critical consideration regarding infrastructure is how separate the machines and plant network need to be. In some cases every switch inside a machine will be replaced with a switch managed by corporate IT. The challenge with this strategy is that if the corporate network goes down, it brings equipment down with it. It also requires changing internal IP addresses, of which a machine can have hundreds. Changing IP addresses on a machine can also risk creating bugs in equipment programming.

For our solution we had to ensure that the equipment could run independently of network issues, and also avoid making potentially hazardous IP address changes. In order to facilitate this we used Allen Bradley 9300-ENA Network Address Translators (NAT). The NAT is a device that sits in between the equipment and plant networks (Rockwell Automation). It allows us to specify a virtual public IP for specific devices on our equipment VLAN, but allows the machine itself keep its own internal IP address range.

Figure 11 shows a conceptual diagram of how the NAT works. In this case the internal PLC 192.168.0.3 IP address is translated to an external 172.168.0.1 address. The two I/O devices are not given an external address and are, therefore, effectively hidden from the external network.
The existing solution is architected using various modular components in the form of middlewares. These middlewares essentially use MQTT as message busses. Because of this, more middlewares can be added as necessary and the system is as scalable as the MQTT broker is.

### 3.2.1 RFID to MQTT Middleware

The RFID middleware I’ve developed has three objectives:

- Consume the events from RFID readers throughout the plant and retransmit them as MQTT messages to any number of MQTT brokers.
- Provide filtering options for tags in order to help distinguish between various read zones.
• Provide a REST API to allow other applications to take tag inventories at any point in
time as well as to assess reader status.

The RFID middleware was developed as a C# console service and uses the SDKs provided by
both Zebra and Siemens in order to connect to each of those readers and consume events. In the
case of Siemens readers the events are provided directly by the Siemens SDK. In the case of
Zebra the consumption of events proved to be unreliable in practice so I continuously polled the
reader and generated events as the tag list changed.

Because the middleware is expected to handle data and polling from a large number of readers
(currently 29) at any time the software is highly parallelized. A separate thread is created to
manage each reader and continually evaluate connectivity. Polling and other events are executed
using a large thread pool. Thread synchronization is achieved both through the use of locks and
by using the concurrent collection classes in C#.

In order to further facilitate thread synchronization, no single tag list is maintained across all
readers. Instead each reader maintains its own tag list and filtering settings. In order to keep tags
distinct between readers, when a new tag becomes visible on a reader, each other reader can
check its own tag list for the same serial number.

3.2.2 Software Filtering Techniques

A number of different software filtering techniques are implemented on a per-reader or per
antenna basis. If a tag is excluded due to a filter it doesn’t generate MQTT events. It will still be
available through the GUI or REST API but will be marked as suppressed. The filtering options
developed are:
• **RSSI (signal strength) high/low limit**

An upper or lower limit can be set for the signal strength of the response from the tag to the antenna.

• **Max tag count**

Set a maximum number of tags that can be under a reader at any time. For instance, in our winding process only one product is being wound at a time so we set the reader filter for a single tag. In our EMF process only two capacitors can exist per station and each station has a single antenna, so we set the limit at two tags per antenna.

• **Gaussian distribution filter**

An RSSI range is specified by the group average +/- some number of standard deviations. This is not exceptionally useful as an outlier can significantly increase the standard deviation. If the range of tag RSSIs is a true Gaussian distribution then this distribution will explicitly filter out some percentage of tags corresponding to the Z score used (Dwiyasa & Lim, 2014).

• **IQR filter**

As an alternative on the RSSI filter, the median and inter quartile ranges are used in place of a mean and standard deviation. Unlike with a Guassian distribution, the median +/- some number of IQRs will not be as susceptible to influence by outliers. The IQR filter has proven useful when trying to section off zones that have large numbers of product, such as a conveyor entrance or exit.

• **Observed and lost timer intervals**

With RFID it’s very possible to have a tag on the edge of a zone that generates rapid observed or lost events. With these filters tags will only begin to generate events after they’ve either existed in the zone for some time, or after they’ve left the zone for some time. This significantly helps reduce the possibility of these rapid events occurring.
• **Regex ID filter**

A regex filter is used to filter out tags that only contain serial number formats we’re expecting.

UHF RFID is fairly common and we don’t want to pick up rogue tags in the room.

• **Antenna moderation time**

When multiple antennas from the same reader are in close proximity a tag can appear to jump from one antenna to the other rapidly. Using this filter we can moderate this possibility by defining a time threshold. If a tag appears on one antenna for most of the time it will continue to appear under that antenna even if it jumps to another for a brief period of time.

• **RSSI smoothing time**

The RSSI (signal strength) of a tag will jump around frequently. In order to improve the usefulness of the above filters, this RSSI smoothing will take a moving average of the tag’s signal strength over time. Note that the moving average isn’t a simple point average, but instead is a moving average weighted by the time in between reads. This helps to mitigate the impact of brief increases or decreases in signal strength. In the example below, for instance, the moving average RSSI for the 7 second window shown in Figure 12 would be calculated as:

$$\frac{(1)(-8-10) + (2)(-10-20) + (1)(-20-15) + (3)(-15-5)}{(1+2+1+3)}$$

![Figure 12 - RSSI Smoothing](image-url)
The FIFO concurrent queue class in C# is used in this implementation. Because moving averages are essentially low pass filters, future implementations can use other low pass filters such as Gaussian filtering as well.

### 3.2.2.1 GUI and REST API

The middleware contains a web server component that serves as both a GUI and REST API. The GUI uses asynchronous JavaScript to poll the middleware and update reader lists through a web based interface.

![Figure 13 - Web interface GUI](image)

- Greyed out text indicates that a tag is filtered out
- Greyed out background indicates that a tag is subject to a lost or observed timer
In order to provide AJAX functionality, the middleware must serve a main page as well as provide information on reader lists and states through a REST API. This rest API serves data in a JSON format that can be consumed by other applications.

One advantage of using REST and a web page for the GUI is that the RFID middleware exists behind a firewall on a secure VLAN. A PC can be set up on the non-secure side of the VLAN to operate as a forward proxy for the purpose of accessing this interface on a public network.

One use of this REST API, for instance, was to help reduce mistakes at the capacitor final test station. Capacitors are filled with oil and after being filled they must sit for some period of time before being tested. A dashboard I placed at final test periodically polls the readers to at final test to observe the capacitors in queue for testing. If a capacitor is detected that hasn’t been in soak for long enough it creates a visible alarm for the operators.
RFID is extremely useful for serializing most processes by providing visibility around the process that doesn’t depend on operator interaction. In some cases, however, serialization benefits from operator interaction. One such case is with test equipment. In many cases with intermediate and final tests an operator has to make a determination as to whether a test was run successfully. One example of this is a leak detection test. If a mistake was made during the leak detection test, the results are ignored. When the test is run successfully the operator will scan a bar code on a linen hang tag. We want to capture the test results from the machine at the time of this success scan.

Each barcode scan generates an SQL insert in an MES database. So in order to facilitate transferring this data to the edge analytic, I developed a simple middleware in C# that subscribes to SQL table inserts using the .Net SqlDependency class and republishes those events to MQTT.
Like with RFID, by keeping this component modularized we’ve improved the scalability of the system and allowed for swapping components as needed.

In addition to bridging SQL to MQTT, the same middleware can also consume MQTT events and insert those events into SQL tables by mapping the keys in the MQTT JSON payload to SQL columns. This allows us to link the middleware components to back ends without maintaining dependencies on specific schemas. Furthermore, by unifying the SQL and MQTT objects in the middleware, the middleware can effectively support four bridge types: SQL to MQTT, MQTT to SQL, SQL to SQL and MQTT to MQTT as shown in the conceptual diagram in Figure 16.

![Figure 16 - MQTT to SQL Bridge Conceptual Diagram](image-url)
One use, for instance, of the MQTT to SQL bridge functionality was to log RFID events at the beginning and end of a conveyor (the treat queue) to SQL. These events are then analyzed in order to determine the number of capacitors on the conveyor at any time in order to compare that count to a limit target.

![Image](image.png)

**Figure 17 – RFID driven treat queue dashboard**

### 3.3 Edge Analytics Platform

As described in section 2.4, we’re using Foghorn for our edge analytics engine. The Foghorn edge analytics platform consists of two types of components – a cloud interface and a number of edge boxes. The edge boxes are generally not interacted with directly. They’re managed from the cloud interface.

For the Clearwater pilot we installed a cloud interface on site in order to keep data in the plant during testing. Furthermore, a single edge box was used to run all scripts in the plant. Scripts
were developed and deployed through the cloud interface. In future implementations one cloud server would likely be able to handle several plants, each with some number of edge devices.

Conceptually in Foghorn the analytics themselves have three steps – data adapters that pull in streams of data from various sources (mainly OPC-UA and MQTT), analytic expressions to process this data, and data publishers to publish results.

![Figure 18 - Foghorn Conceptual Diagram](image)

### 3.3.1 Vel Analytics Language

The analytic expressions are written in a special programming language, Vel (FogHorn Systems, 2016). These scripts used in our use cases are event based models of the actual process. Subsequent sections will examine the operations of the scripts for several different use cases.

The need for a custom programming language for the type of analytics Foghorn handles has to do with the fact that Vel is not a procedural language, it’s flow-reactive. Flow-reactive programming is similar to what happens in an Excel spreadsheet. The Vel script is monitoring several time series streams of data (a stream of sequential data that can be logically selected by time stamp). Once a new point enters the stream, the reaction then fires on all available data in order to create the new output stream.
The concept of treating all data as a time series stream is very useful in manufacturing. All manufacturing data can be viewed in this way. For example, temperature data coming off of a machine is what we would traditionally think of as time series data. But serial numbers coming from an RFID antenna are also time series data. And completed products with data points attached as they come off of an assembly line are streams of data as well.

**Figure 19 - Data Streams**
Figure 20 – EMF use case edge analytic script viewed through the Foghorn cloud interface

Figure 21 - Foghorn GUI For Graphically Developing Logic
The scripts themselves were created by Foghorn, through a joint agreement in which Foghorn leveraged the Clearwater plant for their own pilot. While I spent most of 2016 working with Foghorn on developing their software, their first public version wasn’t released until September of that year.

An example of a simple Vel script is pasted below. This represents two of the data points captured in our oil fill use case:

```vel
# input valve status stream (mapped to data bus)
def stream valve_status is valve_data_type
# input temperature stream (mapped to data bus)
def stream temperature is {timestamp is int, value is real}
# extract temperature data to local stream
def local only_temp = data.value select data from temperature
# define valve closed to open pattern
open_event is (item1:valve_data_type, peek, item2:valve_data_type ->
  item1.is_open == false and item2.is_open == true)
# extract valve open event stream from status stream with pattern
def stream valve_opened = true select state from (open_event from valve_status) when state == true
# data collection constraints... must have at least one value
atleast_one is (data:real .. { >=1 } - data)
# output stream with average temp
def stream temp_output = avg(temp_list)
  select state from (open_event from valve_status)
    with temp_list from (atleast_one from only_temp)
  when state == true
```

**Figure 22 - Vel Programming Example**

### 3.3.2 Data Adapters

Foghorn can consume data from several sources, most importantly OPC-UA and MQTT. This ability drove the requirement for us to use OPC-UA and MQTT in our solution. These data adapters provide data for the analytic expressions, the results of which can be sent to publishers.

Foghorn is directly connected to the MQTT broker we’re using for RFID and barcode events, as
well as the OPC-UA server we’re using for time series data streams. In Clearwater we’re standardizing all of our data sources by forcing them through one of these two sources.

3.3.3 Data Publishers

The native publishers provided by Foghorn consist of various no-SQL database standards, such as Hadoop File System and Kafka. In addition an Influx historian is bundled into the platform and data can be published to that. Throughout our development work with Foghorn, we’ve also added MQTT and OPC-UA publishers as well in order to pitch data to other programs, and also to control machines with the results of the analytics. If a Foghorn script, for instance, finds that a value is trending too far off of a mean, or that any kind of alarm signals, we now have the ability to shut that process down.

4 USE CASES

4.1 Desired Goal

The goal of each of our pilot edge analytics use cases was to generate a characteristic record of a manufacturing process that could be tied to the specific serial numbers of the product associated with that process at this time. Because each process would have its own representative characteristics, and each process record could be associated with any number of serial numbers, having a strict schema for these records isn’t possible. Instead I defined a key-value JSON format for each characteristic record with a number of standard “header” keys that could be used to identify common process characteristics like machine name or line number.
The standard “header” keys in each characteristic JSON are defined below. In addition to these keys, any number of sensor values can be included with arbitrary key names in camel case notation (ex. avgPressure, avgTemperature, etc.).

<table>
<thead>
<tr>
<th>Key Name</th>
<th>Optional/Required</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>processStartTime</td>
<td>required</td>
<td>Unix time</td>
<td>Start time of process data window – used for analytics</td>
</tr>
<tr>
<td>processEndTime</td>
<td>optional</td>
<td>Unix time</td>
<td>End time of process data window – used for analytics</td>
</tr>
<tr>
<td>totalProcessTime</td>
<td>required</td>
<td>Long</td>
<td>This is used for production metrics, instead of data analysis. So unlike the start time and end time, the total process time can also include the time a product is sitting idle for whenever possible. In most cases this value will likely just be startTime-endTime.</td>
</tr>
<tr>
<td>equipment</td>
<td>optional</td>
<td>String</td>
<td>The name of the gauge or equipment associated with the record</td>
</tr>
<tr>
<td>process</td>
<td>optional</td>
<td>String</td>
<td>The name of the process associated with the record</td>
</tr>
<tr>
<td>line</td>
<td>optional</td>
<td>Int</td>
<td>The number of the line associated with the record</td>
</tr>
<tr>
<td>station</td>
<td>optional</td>
<td>Int</td>
<td>The number of the station associated</td>
</tr>
<tr>
<td>Field</td>
<td>Required/Recommended</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>----------------------</td>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>parent</td>
<td>optional</td>
<td>int</td>
<td>A parent serial number to create genealogy. In this case if parentItem doesn’t exist, the record’s parent is assumed to have the same item number as the child.</td>
</tr>
<tr>
<td>parentItem</td>
<td></td>
<td></td>
<td>The item of the parent to create genealogy.</td>
</tr>
<tr>
<td>item</td>
<td>highly recommended</td>
<td>String</td>
<td>An item number or SKU. Serial-SKU combinations are assumed to be distinct. If item number is omitted then a generic “NA” number should be created.</td>
</tr>
<tr>
<td>plant</td>
<td>highly recommended</td>
<td>String</td>
<td>A plant associated with the record.</td>
</tr>
<tr>
<td>serials</td>
<td>highly recommended</td>
<td>Array</td>
<td>An array of serial numbers associated with the record in the following format:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>[{tagId:&quot;serialA&quot;,timestamp:12345}, {tagId:&quot;serialB&quot;,timestamp:6789}]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>tagID is the serial number and is required. timestamp is the unix time that the serial number was scanned, observed, etc. and is not required.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Serial number arrays can also be in</td>
</tr>
</tbody>
</table>
the following format if there’s no logical timestamp associated with the serial number:

```
["serialA",timestamp:12345}, {tagId: "serialB",timestamp:6789}]
```

color| children | optional | Array |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>An array of child records, which are generally assumed to be linked to child parts or assemblies unless a top level serial number is explicitly defined within the child record.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

An example of a complete record with a child is shown in Figure 23 with “header” keys in bold.

```
{
  "avg_mia": 1.752635,
  "avg_pe": 3.123366,
  "avg_te1": 24.227273,
  "avg_te2": 24.325,
  "end_decay_rate": 74.644379,
  "equipment": "treat",
  "line": 1,
  "oil_type": "7a",
  "process": "emf",
  "process_end_time": 1478585159,
  "process_start_time": 1478548026,
  "serials": [
    {
      "tagID": "S420351",
      "timestamp": 1478584900.923
    }
  ],
  "start_decay_rate": 39.644379,
  "station": 1,
  "total_process_time": 37133,
  "vacuum_button_type": "7a",
  "vacuum_farm_state": 0.151081
}
```
4.2 Capacitor Manufacturing Process

The heart of the capacitor is the roll. Rolls, as shown in Figure 2, consist of an insulator film sandwiched in between aluminum foils. These combinations of insulators and foils are rolled up to contain thousands of film-foil layers in a roll. Several rolls are then combined with various poll arrangements in order to produce a stack of rolls known as a pack. This pack is then placed into a metal case. The case and roll combination is then baked under a vacuum to remove moisture.
Once moisture is removed it’s filled with dielectric oil. This dielectric oil impregnates the insulator which increases the capacitance. Once it’s filled and the oil has had a chance to impregnate it, the capacitor is subjected to a battery of tests. If it passes the test it’s painted and shipped to the customer.

For our purposes, final test would provide our pass or fail classification for any number of tests that we wanted to optimize performance against, so the painting process is out of scope and any of the processes before final test are in scope.

The Clearwater capacitors plant produces mostly engineered to order products. A customer will define the size and performance requirements of a capacitor, then the plant will build it to their specifications. As a result Clearwater has thousands of potential capacitor designs. Some designs are repeated, others are not.

**4.3 Dielectric Problem**

We applied the edge analytic solution to several use cases in Clearwater in an attempt to provide insights into our data. In each case we defined the same quality excursion to solve – dielectric test failures. Our goal was collect characterized data from our process, and then to build a binary classification model against whether each unit passed or failed this specific test. The reason we selected dielectric test was due the fact that it was the highest cost of quality for all test failures in the plant. Dielectric failures are essentially shorts between aluminum sheets. Engineers had theorized that dielectric failures could be caused from anything from a speck of dust in the roll to high moisture after drying. Our hope was that collecting manufacturing characteristics in cases where those characteristics varied would provide some insight into the failures.

Whether or not a unit passed or failed dielectric could be determined by the final test code. The final test process is sequential – several tests for various characteristics such as resistance,
dielectric and capacitance are performed at stations. For our classification we ignored all data that failed before dielectric test. Anything that failed dielectric was considered a failure, and anything that passed dielectric (even if it failed a subsequent test) was marked passing. This provided the two pass and fail classes.

Note that of capacitor went through final test more than once we generally used the last past result, but this strategy can be flawed in the case of rework. Future analytics should use the last pass final test result before any rework is done.

4.4 Winding Use Case

The purpose of the winding process is to build the packs of Mylar rolls that provide the electric charge storage for the capacitor. This process is essentially the heart of the capacitor manufacturing operation.

**Figure 25 - Winding Process Diagram**
The winding process consists of three pieces of equipment:

- **Winder**
The winder produces each individual roll by winding several layers of film and foil together.
- **Dry Kap Tester**
The dry kap tester performs a quick pseudo-capacitance measurement on each roll as it comes off of the winder. It also subjects the roll to a high voltage in order to make sure that it doesn’t short.
- **Pack Press**
The pack press squeezes a pack of rolls together in order to get the pack to a height that can fit in the can. Paper bands are then applied to the pack in order to keep it from flexing back to the non-pressed height.

Rolls aren’t serialized. The rolls are created and placed into a pack. The edge analytic can keep track of which rolls go into which pack by watching data from the machines.

Once the pack is created it’s assigned a serial number, and this serial number is traced throughout the rest of the manufacturing process.

## 4.4.1 Initial Investigation

We knew winding should be a focus area for analytics. In order to reach this conclusion we performed some initial data analysis using MES data we had from each of our process areas. We examined the potential influence on various process attributes and how they impacted dielectric yields (determined by the pass/fail classifications). As the highest level the process attribute we examined was the manufacturing line the capacitor was built on. We knew that if we saw significant variation by line that wasn’t related to statistical uncertainty (the data had a large sample size) then this area should be investigated. Our long term hope would be that as we
revealed potential interaction effects and predictors in the data, we could use this knowledge to move towards a full predictive digital twin model for the plant.

Two examples are below in Figure 26 and Figure 27 – winding line and assembly line.

![Dielectric Yield vs Winding Line](image)

**Figure 26 - Dielectric Yield vs. Winding Line Segmentation**
While both have variation, winding line #1 dips to 85% yield. This is a very low yield and is clearly a sign that we need to investigate winding further and understand why, for the same SKU, this one line could be so significantly different. Because assembly also showed some variation we continued to investigate that as well.
Next we investigated the same processes, but isolated a specific operator and SKU in order to reduce uncertainty caused by interaction effects between various design and manufacturing parameters. Conceptually variations in data might not reveal themselves if variations due to those interaction effects are present.

Figure 28 - Segmentation by Winding Line for the Maximum Variation Operator and SKU Combination
While assembly didn’t reveal anything extraordinary, winding showed significant variation in every combination we tested. The data above shows that for the same operator and SKU performance can vary significantly from line to line.

We also found that for the same line and SKU performance by operator varied even more significantly.
One must be cautious in this case. The fact that operator to operator variation exists doesn’t necessarily mean that operators are the cause of the variation. It’s very possible that another characteristic confounding with operators is causing variation.

4.4.2 Edge Analytics Implementation

Even if a critical cause is the operator or the winding line, eliminating an operator or winding line isn’t a practical solution for optimizing a manufacturing process. Our goal is to drill down to the absolute root causes of the variation, which we would hope would be some kind of equipment parameter that can be adjusted to improve yield.

This is the benefit of using edge analytics to capture machine data – we can continue follow the same process and drill down to equipment settings, operating characteristics, environmental
characteristics, etc. But the edge analytic data, since it comes off of a machine, lends itself to optimization.

The first step in developing an edge analytic to contextualize data is determining the various pieces of data we wanted to collect.

4.4.2.1 Characteristics

In our case we had determined that both line and operator could drive variation. So we focused on characteristics that could vary on the winder at the operator’s discretion. When we combined this with engineering knowledge, we decided the two most critical characteristics would be winding tension and winding speed. Neither of these operating parameters are defined by design and they are both available for the operator to tweak.

We also found that pack press height was being varied by the operator outside of specification limits. Height can contribute to force, which we’ve always believed might be a driver of dielectric failures.

4.4.2.2 Trigger Events

Once we knew the characteristics we wanted to gather, we have to define scriptable trigger events around those characteristics for the purpose of developing a Vel edge analytic script.

In our case, before RFID was implemented, I used barcode scans to define the first trigger point in the winding process. When the barcode was scanned we would start collecting roll level data from the winder and dry kap. Eventually we replaced barcode scanning with RFID, for reasons defined in section 4.4.2.3.
Each roll has its own two trigger points in winding, the first of which is defined by an arbor turns counter in the machine, and the second of which is defined by a roll completion flag. In between these two triggers, values from counters are maxed and values from other sensor streams like tension are averaged. These values become the winding roll characteristics.

For dry kap we only need one trigger - the machine provides a single completion flag. When that completion flag changes value the edge analytic captures a snapshot of OPC data that contains the test result.

Next, pack press provides a completion flag as well and this is used as the completion trigger for the entire process. When the operation is complete, this signals the edge analytic to record pack press results and also to take the winder data and dry kap data from the initial barcode scan trigger to the pack press operation trigger.

Figure 31 - Winder Use Case Triggers
4.4.2.3 Vel Implementation

The case described in the previous section was fairly straightforward to model in Vel. Each individual sensor or trigger is a stream. Vel process data according to input streams and then sends the resulting roll level data to an output stream.

These output streams then become input streams for another line-level analytic that aggregates each roll into a serialized pack-level stream, which is outlined in the diagram below.

![Diagram: Roll Level Streams Aggregating to Pack Level]

**Figure 32 - Roll Level Streams Aggregating to Pack Level**

Building a real-time analytic like this in a procedural based programming language would be incredibly difficult as events would have to be created for every individual sensor and roll level completion.
4.4.2.4 *Switch to RFID*

Once we implemented the first scripts using barcode scans we found a major problem with the data. We laid out all events in chronological order to validate the data. We found the operators weren’t reliably scanning barcodes at the beginning of building packs. In many cases they’d build several packs, and then scan all barcodes at once. Because data collection was never a concern, discipline hadn’t been built into the process.

<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
<th>Serial</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:00</td>
<td>Barcode Scan</td>
<td>R00001</td>
</tr>
<tr>
<td>1:10</td>
<td>Pack Press</td>
<td></td>
</tr>
<tr>
<td>1:11</td>
<td>Barcode Scan</td>
<td>R00002</td>
</tr>
<tr>
<td>1:21</td>
<td>Pack Press</td>
<td></td>
</tr>
<tr>
<td>1:22</td>
<td>Barcode Scan</td>
<td>R00003</td>
</tr>
<tr>
<td>1:32</td>
<td>Pack Press</td>
<td></td>
</tr>
<tr>
<td>1:33</td>
<td>Barcode Scan</td>
<td>R00004</td>
</tr>
<tr>
<td>1:43</td>
<td>Pack Press</td>
<td></td>
</tr>
</tbody>
</table>

**Observed Events**

<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
<th>Serial</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:00</td>
<td>Barcode Scan</td>
<td>R00001</td>
</tr>
<tr>
<td>1:10</td>
<td>Pack Press</td>
<td></td>
</tr>
<tr>
<td>1:20</td>
<td>Pack Press</td>
<td></td>
</tr>
<tr>
<td>1:21</td>
<td>Barcode Scan</td>
<td>R00002</td>
</tr>
<tr>
<td>1:22</td>
<td>Barcode Scan</td>
<td>R00004</td>
</tr>
<tr>
<td>1:30</td>
<td>Pack Press</td>
<td></td>
</tr>
<tr>
<td>1:40</td>
<td>Pack Press</td>
<td></td>
</tr>
<tr>
<td>1:41</td>
<td>Barcode Scan</td>
<td>R00003</td>
</tr>
</tbody>
</table>

Figure 33 - Expected vs. Observed Barcode Events Example

In this case we had two options – build discipline into the process, or allow the operators to work as they already did and find another way to serialize the data. Generally a good manufacturing practice is to assume that operators are never wrong and if you don’t design a process to be performed in a specific way it won’t happen. Because of this we went with the latter option.

We found that when building each pack, the operator would place the RFID enabled tag on the pack press machines (see Figure 6 for an image of a tag on a pack press machine). So in order to serialize the process, we placed RFID antennas on the ceiling for each of these machines. Now when the operator places the tag on pack press, it generates a message that’s used as the first
trigger in place of the barcode scan. The tag placement on the machine effectively controls the data collection process. While this isn’t a perfect way of mistake proofing, it’s far more intuitive for the operator.

Note that due to the nature of RFID, the RFID visibility of the tag can go in and out throughout the course of the build. One reason, for example, would be the operator walking between the tag and antenna. Because of this the Vel trigger event used is the first observation of any new serial number, and not a specific RFID event type.

4.4.2.5 RFID Filtering

The hardest part of any RFID deployment is the fact that a single antenna will pick up all tags under its field of view. If someone walks near the antenna with a different tag, there’s a risk that it’ll end up being interpreted as a new Vel trigger event. In the case of winding, we had to place the antennas far away from the machines in order to avoid affecting the operators’ work. Furthermore, we found that the winders were picking up tags that didn’t have serial numbers written to them and as a result they were sending random strings to the antennas.

Because we developed our own middleware, we had the ability to try and implement new filtering options in order to help mitigate these issues. For winding I developed the following new filters:

- The tag data would have to match a specific Regex string in order to send an MQTT message. We set this Regex to our serial number format.
- Only the maximum RSSI tag in scope would be used.
- Tags that had been in scope for only a small amount of time would be ignored. A tag would have to be in scope for more than some number of seconds in order to effectively kick out a lower strength tag. This eliminated events due to transient tags.
4.4.3 Opportunities

The winding edge analytic is now capturing characteristics per roll and per pack. This gives us a clear opportunity for optimizing winding characteristics against dielectric failures. Additionally, we’ve added a vision system and laser thickness measurements to a single pilot line to measure roll dimensions so that we can optimize roll form as well.

Furthermore, now that we have the analytics infrastructure in place we can use the Vel expressions and trigger events in the edge analytic to generate alarms and also prevent operators from making mistakes or running a process incorrectly. For instance, we can post a warning if an operator tries to press a pack with a non-standard displacement or event send a signal back to the machine to lock them out from doing so together.

In the near future we’ll also be combining the analytic with known trends or using anomaly detection to look for irregularities. Anomaly detection can generate alerts that identify whether or not certain process settings appear to be increasing failures over time.

4.5 Treat Oven Use Case

The treat oven is a multi-stage vacuum oven used for drying a capacitor after sealing. Unlike with winding, the treat oven is a batch process. A large pallet containing as many as 35 capacitors moves through the process in stages.

The edge analytic has several novel uses here:

- It can determine which serial numbers are on the pallet by watching RFID events as the pallet moves in and out of the equipment.
• It can follow the pallet through the equipment by watching events such as proximity switches going on and off and doors between stages opening and closing. This is accomplished by considering each stage in the oven to be its own stream. When a stage generates a reaction by seeing the pallet move off of a proximity switch, the result of this reaction feeds the subsequent stream in the next stage.

• It can calculate useful statistics from time series data streams such as time at set point and time to set point, which aren’t saved in the PLC program and aren’t, therefore, exposed to the OPC server. These statistics can also be used for alarms and anomaly detection.

The first point, determining which serial numbers on the pallet, is an excellent example of how edge analytics can be used to improve RFID accuracy. In the case of the treat oven, an entire pallet of capacitors is generating events. When the pallet enters the oven, the RFID signal will be lost and this is the trigger event we want to use to start data collection.

Because the pallet is large, however, and covering the entire pallet of capacitors with antennas is impractical, some number of capacitors will always be on the edge of the RFID zone and will continuously generate both observed and lost events.

The edge analytic, fortunately, can ignore specific events outside of some selected time window. We can add logic to both only consider RFID events from while the door is opened and only select events from some time window around the door opening or closing. This ability to perform signal processing across multiple devices is a feature that makes the edge analytic extremely powerful.
4.6 Oil Fill Use Case

The oil fill process consists of the following actions:

- The operator pulls a vacuum on the sealed and dried the capacitor
• The vacuum is cut and the operator checks the decay rate (loss of vacuum) over a certain period of time
• If the decay rate is low enough, the capacitor can be filled. If it’s not then the capacitor must go back to welding in order for the leak to be repaired.
• The capacitor is filled with oil

Several problems with this process exist:

• We have two different oil types in our designs. The machine isn’t connected to the MES system, so there’s no way to force the operator to use a specific oil type and mistakes can happen.
• There’s no time control on the filling process, and filling times vary by design. As a result we’ve frequently had problems where larger capacitors were taken off of the process with smaller ones and weren’t completely filled as a result.
• Not all capacitors were disposition correctly and not all serial numbers were scanned

Furthermore, we had two analytical needs:

• We also wanted to know whether various oil quality characteristics contributed to dielectric failures – mainly the temperature of the oil during filling and moisture content.
• We’ve frequently had yield excursions at EMF where capacitors frequently failed decay in their first pass. This shouldn’t be happening as we have a leak test at the point of welding and before the capacitors go through the baking process. Because the baking process is very long (> 40 hours) this was creating a lot of extra WIP and rework, as well as delaying customer orders. We wanted to establish transfer functions between EMF characteristics and results and leak test rates so that we could both optimize yields and investigate the issue.
4.6.1 Event Triggers

Unlike in the winding use case, events for oil fill are primarily based on operator button presses. The operator presses a button to pull a vacuum, presses another button to check decay rate, then presses another button to fill with oil. The end trigger is a serial number scan. In each case of the button presses we were able to find out the registers used by inspecting the HMIs.

For data characterization, we captured either the max or average of each value in between the appropriate triggers. For instance, the max decay rate was taken during the decay operation and the average water content was taken during the fill operation.

4.6.2 Oil Alarms

Because the initial Vel script had the serialized trigger point at the end of the process, the only check we were able to do around oil type was to throw an alarm if a capacitor was filled with the incorrect oil. Having an alarm after the fact obviously wasn’t ideal. This prompted the installation of RFID antennas on every fill station. The RFID implementation means that our analytic now knows exactly which capacitor is in each station. This allows us to implement process controls and mistake proofing.

Furthermore, for safe measure, we also record which specific oil type button the operator presses.
The RFID implementation for EMF was extremely difficult for several reasons:

- Each station is in close proximity to the next station, and there’s a high probability the antenna will pick up a tag from the wrong station.
- If we turn down the antenna power too much, it might not read transponders that are too close to the metal case (as they are in some designs)
- There’s a conveyor below each station where the capacitors travel before they raise up to the fill level, so we’d have to ignore the transponders moving along the capacitor.
- There’s a lot of metal in the area, causing signals to bounce around. This can make the closest transponder appear further away than others.
In order to facilitate this, we created a new filtering technique. Antenna changes for a specific tag are moderated for EMF. This means that if an antenna and reader changes briefly for a few seconds for a specific tag then it’s assumed to be with the antenna and reader it’s existing in for the maximum amount of time. This helped eliminate issues due to signals bouncing around.

We also implemented the following existing software filter techniques:

- If a capacitor is in scope for only a short period of time then it’s ignored. This helped resolve the conveyor issue.
- Only the top two RSSIs per station are used (there are two capacitors per station), which helps to eliminate tags from surrounding scopes.
- Only one tag can exist per station across all EMF lines.

In addition to software filtering, we’ve also modified the physical space:

- Aluminum barriers were installed to help block the signals between stations
- The antenna locations were moved in order to optimize line of sight and prevent signals from being blocked or reflected

5 ANALYTICS APP

The combination of RFID with edge analytic has done an excellent job of capturing and contextualizing data from the two use cases described here, as well as others. This solution exists at the edge, now we need a cloud application in order to consume the data and provide analytics and reporting around it. This application effectively completes the RFID-Edge-Predix stack.

The goal of the app is to support the following key functions:

- Evaluate yields vs. discrete pass/fail classifications
• Allow filtering of data sets to remove interaction effects and introduced variation from data (ex. focus on a specific machine, operator, SKU, etc. – or combinations of these)

• Monitor changes and trends of contextualized data over time. This is both to predict future quality excursions and also to track new process optimizations to make sure they’re holding. These trends can be used to create dashboards that indicate overall process health.

5.1 Predix

Predix is GE’s Cloud Platform as a Service. A Platform as a Service is a type of cloud offering that provides a collection of micro-services. These micro-services are essentially modularized tools that one can build an application with. Micro-services can consist of things like user authentication as well as various types of databases and analytical services.

Predix is based on Cloud Foundry, which is also a Platform as a Service. The difference between Predix and Cloud Foundry is that Predix’s collection of micro-services are geared towards industrial use. These services, for instance, contain things like a historian and an ISA-95 compliant asset management service.

Micro-services are independent programs that talk to each other using a combination of REST, environment variables, and message brokers. Applications are designed in such a way so that they can scale up simply by adding more micro-services. File systems are sparingly used - persistent storage isn’t available unless it’s implemented by using one of Predix’s databases.

We’re using the Predix cloud for our analytics app since it provides a key scalability benefit, and also because it can be part of an IT supported infrastructure. This app is expected to sit on the IT side, not the OT side, and plants won’t have the resources to maintain it. Furthermore, having the app in the cloud makes it very easy to deploy to other plants.
5.2 GUI Design

The ultimate goal of connecting and contextualizing processes using RFID and edge analytics is to optimize processes by producing transfer functions and data models that connect process parameters to part performance.

Because these efforts can take time, the Predix application is designed to facilitate immediate benefits by assessing process health and trending various quality characteristics to keep them in control. The GUI of the Predix application has three main components: a plant overall health dashboard, various dashboards driven by a quality analysis and trending tool, and an alarms management screen.

The current state of the application is a functional mockup. It’s connected to the data from Foghorn and functions, but it’s not yet considered to be a minimum viable product.

5.2.1 Quality Analysis Tool

The heart of the application is a quality analysis tool that allows a user to view histograms of test yields vs any of the process characteristics acquired by the edge analytic or through some other linked data source (ex. MES). The goal of the user would be to find characteristics that can cause significant yield variation as those characteristics represent opportunities for process optimization.

In addition to viewing histograms, functionality exists for filtering the data set based on any other characteristics. Clearwater, for instance, produces thousands of different capacitor SKUs and viewing these histograms for all SKUs combined would introduce so much design variation that any actual yield vs. characteristic variation likely wouldn’t be visible. We can filter the data set by key design characteristics in order to reduce this variation. Furthermore, we frequently need to
optimize parameters for each individual piece of manufacturing equipment. An optimal setting on one machine might not apply to another.

Each histogram is given an objective metric to represent the amount of variation in the histogram. In our case, the objective metric is the standard deviation of the yield bars. This standard deviation excludes bars with non-statistically significant sample counts. The ultimate goal of this objective statistic is two-fold. It can be used to assess the changing impact of any characteristic over time as well as optimization efforts. Plus it can serve as a heuristic for automatic searches through various characteristics as the software progresses.

In the future the analysis tool will also include trending of characteristics – specifically anomaly detection that can identify if a characteristic is increasing, decreasing or increasing in variation over time. Future iterations might also include cluster detection.
5.2.2 Alarm Management Tool

In addition to characterizing data, the edge analytics are also very useful for generating alarms. The event based expressions in the edge analytics script can not only generate alarms based on various time-series data triggers, but they can also generate alarms based on characteristic data.

The Predix GUI includes functionality to monitor these alarms in real time, and actually maintains a websocket connection to the edge analytics box in order to view events immediately. Functionality exists in the GUI to either allow the user to ignore an alarm or take an action on it.
Histograms and trends from the quality analysis tool can be compiled into dashboards that are capable of monitoring different areas of the plant. Alarms generated through edge analytic triggers can also be funneled into these dashboards as well. By monitoring histogram heuristics and anomaly detection for changes, each dashboard can provide a holistic view of that particular area.

A top level plant management dashboard provides a summary of each of these individual area dashboards. This top level dashboard can be viewed by plant quality management in order to assess day to day health trends. Note that dashboards have not been implemented yet in the current working mock-up - the view in Figure 38 is a placeholder.
5.3 Original Back End Schema

As described in section 4.1, the JSON format of the characteristic data generated by Foghorn has no strict schema. Any number of keys representing sensor data can be contained within the record. The formats of the records are expected to continually change. Furthermore, depending on the process, a record can be associated with a single capacitor (as in winding), a couple of capacitors (as in EMF), or a large pallet of capacitors (as with the treat oven).

Because of this, key-value document databases are ideal for this type of data. Because, however, a mature key-value database didn’t exist in the Predix toolkit we were forced to use SQL.

Traditionally SQL doesn’t lend itself well to key-value data. If the schema of each record was standard and not subject to change, we could potentially create a separate table for each analytic case. The disadvantage of this is that any stored procedures or reporting tools built on top of such
a database would have to be written with special cases for each table. And in our case, the JSON does change which has the potential to lead to tables with large numbers of NULLs.

For instance, let’s assume a characteristic record has the sensor data keys “Temperature” and “Pressure”:

Table 3 - SQL Table with Values in Separate Columns

<table>
<thead>
<tr>
<th>Record</th>
<th>Serial</th>
<th>Temperature</th>
<th>Pressure</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ABC12345</td>
<td>125</td>
<td>0.0001</td>
</tr>
</tbody>
</table>

Now, if a new key, “Humidity” is added the database will have to rebuild the entire table. And we have to fill in the empty Humidity cells with nulls.

Table 4 - SQL Schema with New Key Introduced

<table>
<thead>
<tr>
<th>Record</th>
<th>Serial</th>
<th>Temperature</th>
<th>Pressure</th>
<th>Humidity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ABC12345</td>
<td>125</td>
<td>0.0001</td>
<td>NULL</td>
</tr>
<tr>
<td>2</td>
<td>DEF67890</td>
<td>130</td>
<td>0.0002</td>
<td>50</td>
</tr>
</tbody>
</table>

The better way to store the data, therefore, is in one very large column. This technique has some disadvantages due to large IDs and inefficiencies with data mining due to inefficiencies in filtering data sets, but the table itself will be far more condensed which will lead to faster scans and queries.

Table 5 - All Keys in a Single Column

<table>
<thead>
<tr>
<th>Record</th>
<th>Serial</th>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ABC12345</td>
<td>Temperature</td>
<td>125</td>
</tr>
<tr>
<td>1</td>
<td>ABC12345</td>
<td>Pressure</td>
<td>0.0001</td>
</tr>
<tr>
<td>2</td>
<td>DEF67890</td>
<td>Temperature</td>
<td>130</td>
</tr>
<tr>
<td>2</td>
<td>DEF67890</td>
<td>Pressure</td>
<td>0.0002</td>
</tr>
<tr>
<td>2</td>
<td>DEF67890</td>
<td>Humidity</td>
<td>50</td>
</tr>
</tbody>
</table>
Using this strategy, I designed a very flexible SQL schema that was capable of storing and appropriately organizing this data. Because newer SQL databases have native JSON processing capability, so I was able to write a stored procedure to parse the JSON from the edge analytic and place the appropriate data from each record into its respective tables. In the case of nested records, the stored procedure parsed each record recursively and generated a serial number with a “-1” appended if no serial number in the child record was defined. For instance, if a winding record had the serial number “S00001” then each child record would have serial numbers like “S00001-1”, “S00001-2”, etc. recursively.

The characteristic database diagram is shown in Figure 39. The large vertical columns of data are in the NumericData and StringData tables. Detailed descriptions of each table schema are provided in the appendix. Note that in this schema “CTQ” refers to “Critical to Quality” which is a six sigma term for the type of characteristic we’re collecting using the edge analytic.
Figure 39 - Characteristic Database Diagram

For instance, the JSON in Figure 40 translates to the tables in Figure 41:
Figure 40 - Example JSON for SQL Ingestion
### SQL Query Results

**Table 1:**

<table>
<thead>
<tr>
<th>id</th>
<th>timestamp 1</th>
<th>start Time 1</th>
<th>end Time 1</th>
<th>processTime 1</th>
<th>line 1</th>
<th>station 1</th>
<th>samplingPlanID 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2016-10-05 15:45:47.370</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>id</th>
<th>equipmentID 1</th>
<th>processID 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>594</td>
<td>48</td>
</tr>
<tr>
<td>2</td>
<td>595</td>
<td>48</td>
</tr>
<tr>
<td>3</td>
<td>596</td>
<td>48</td>
</tr>
<tr>
<td>4</td>
<td>597</td>
<td>48</td>
</tr>
<tr>
<td>5</td>
<td>598</td>
<td>48</td>
</tr>
<tr>
<td>6</td>
<td>599</td>
<td>48</td>
</tr>
<tr>
<td>7</td>
<td>600</td>
<td>48</td>
</tr>
<tr>
<td>8</td>
<td>601</td>
<td>48</td>
</tr>
</tbody>
</table>

**Table 2:**

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>PCTQ</th>
<th>hstaStrID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>avg_m1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>avg_pe</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>avg_ln1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>avg_to2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>end_d</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>ol_type</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>star_d</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>vssu</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

**Table 3:**

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>plantID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>48</td>
<td>-1</td>
</tr>
</tbody>
</table>

**Table 4:**

<table>
<thead>
<tr>
<th>id</th>
<th>itemNumber</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NA</td>
</tr>
</tbody>
</table>

**Table 5:**

<table>
<thead>
<tr>
<th>id</th>
<th>ctaRelationID</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>394</td>
<td>1.96452796459198</td>
</tr>
<tr>
<td>2</td>
<td>394</td>
<td>2.46483111361531</td>
</tr>
<tr>
<td>3</td>
<td>394</td>
<td>27.127735505703</td>
</tr>
<tr>
<td>4</td>
<td>394</td>
<td>27.3050076282895</td>
</tr>
<tr>
<td>5</td>
<td>394</td>
<td>69.23429376086547</td>
</tr>
<tr>
<td>6</td>
<td>394</td>
<td>30.23429376086547</td>
</tr>
<tr>
<td>7</td>
<td>394</td>
<td>0.064979001875...</td>
</tr>
</tbody>
</table>

**Table 6:**

<table>
<thead>
<tr>
<th>assetID</th>
<th>recID</th>
<th>timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>394</td>
<td>1900-01-01 00:00:00.000</td>
</tr>
<tr>
<td>2</td>
<td>394</td>
<td>1900-01-01 00:00:00.000</td>
</tr>
<tr>
<td>3</td>
<td>394</td>
<td>1900-01-01 00:00:00.000</td>
</tr>
</tbody>
</table>

**Table 7:**

<table>
<thead>
<tr>
<th>id</th>
<th>evalueNumber</th>
<th>typeID</th>
<th>parentID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S370277</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>2</td>
<td>S370283</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>3</td>
<td>S370285</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>
The goal of this database structure isn’t only to provide flexibility for characteristic records. The structure provides logical joins for the types of manufacturing applications that use the data. Some examples of joins are provided below:

- **List of data for a serial number**

- **All records for a process within a timespan**

Note that consideration when segmenting must be given to the fact that each record has its own timestamp (ex. we might want all records, regardless of date, corresponding to serial numbers that were *tested* between two dates).

- **All records for a specific CTQ**
5.4 Simplified Back End Schema

The schema described in section 5.3 was considered to be too complex by developers for several reasons:

- Simple queries required extensive table joins
- Inserts required cross referencing several tables to find the correct keys

As a result, a new and significantly simpler schema was developed.

![New Schema Diagram](image)

**Figure 42 - New Schema**

This schema contains only four fundamental tables that are required for most queries, although tables containing meta data can be added as necessary. The SensorRecords table is essentially a header for the json analytic expression result, and the SensorData tables contain the key-value pairs associated with it.

In this schema, the columns are as follows:
### Table 6 - New Schema Table Descriptions

<table>
<thead>
<tr>
<th>Table</th>
<th>Column</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SensorRecords</td>
<td>record</td>
<td>bigint</td>
<td>This is the record number representing a single level json, and the only identity column in the database.</td>
</tr>
<tr>
<td>SensorRecords</td>
<td>series_name</td>
<td>int</td>
<td>This is the hashed name of the analytic script used to generate the data.</td>
</tr>
<tr>
<td>SensorRecords</td>
<td>test_date</td>
<td>datetime</td>
<td>This is the date the analytic record was generated.</td>
</tr>
<tr>
<td>SensorRecords</td>
<td>line</td>
<td>int</td>
<td>The manufacturing line number the analytic corresponds to (if applicable)</td>
</tr>
<tr>
<td>SensorRecords</td>
<td>equipment</td>
<td>int</td>
<td>The hashed name of the equipment (if applicable)</td>
</tr>
<tr>
<td>SensorRecords</td>
<td>parent_record</td>
<td>bigint</td>
<td>The parent record for multi-level analytic expression traceability</td>
</tr>
<tr>
<td>SensorData</td>
<td>record</td>
<td>bigint</td>
<td>The record # associated with this key-value pair</td>
</tr>
<tr>
<td>SensorData</td>
<td>ctq_name</td>
<td>int</td>
<td>The hashed key name for the key-value pair</td>
</tr>
<tr>
<td>SensorData</td>
<td>ctq_value</td>
<td>real</td>
<td>The ctq_value for the key-value pair</td>
</tr>
<tr>
<td>SensorDataStrings</td>
<td>record</td>
<td>bigint</td>
<td>The record # associated with this key-value pair</td>
</tr>
<tr>
<td>SensorDataStrings</td>
<td>ctq_name</td>
<td>int</td>
<td>The hashed key name for the key-value pair</td>
</tr>
<tr>
<td>SensorDataStrings</td>
<td>ctq_value</td>
<td>varchar</td>
<td>The ctq_value for the key-value pair</td>
</tr>
<tr>
<td>SensorSerialNumbers</td>
<td>record</td>
<td>bigint</td>
<td>The record # associated with this serial number</td>
</tr>
<tr>
<td>SensorSerialNumbers</td>
<td>ser_num</td>
<td>int</td>
<td>The hashed serial number</td>
</tr>
</tbody>
</table>
Hashed strings are used for Series, CTQ and Equipment names in order to reduce the size of the data and speed up searches. This was better strategy than using foreign keys as separate tables don’t need to be scanned on every insert to see if matching CTQ names exist, and queries themselves can contain hashed strings, meaning we don’t need to join more than two or three tables together for simple queries.
6 CONCLUSION

The ultimate goal of this solution is a moonshot vision of implementing predictive modeling in a manufacturing line. Predictive modeling would yield two major benefits:

• We’d be able to identify problem product before they fail a final test, and then either take a corrective action or rework a capacitor before it generates larger amounts of scrap.

• Once we have a model, essentially a “digital twin” of the manufacturing process, we’d be able to tweak parameters in order to optimize our processes and then evaluate results in the model. This would allow for continual process optimization and keep our product yields at the absolute theoretical peak.

6.1 Predictive Modeling Challenges

Throughout this pilot we’ve kept the moonshot predictive modeling goal in mind. We’ve also identified several challenges in the effort to establish characteristics vs. product performance relationships and eventually build a predictive model:

• Segmentation vs. Data Volume

In a predictive model such as a Bayes’ Net or decision tree data sets are continually filtered by various characteristics in order to increase variation at each step. In Clearwater, due to our large number of SKUs we have to segment data sets more than we would in a plant that only produces a few different products. Clearwater only builds about 200 capacitors a week, of which yields are generally above 90%. By the time we segment a data set we might only have a handful of failures to analyze every month. This has made predictive modeling very difficult.

• Rework

In any manufacturing process product can be reworked. In Clearwater, for instance, our
capacitors go through a welding process. After welding a vacuum is created in the capacitor and it’s filled with oil. If the vacuum can’t be created then a weld leak exists and the capacitors go back to welding. The vacuum is then pulled again. How this rework impacts a data model isn’t known as the process and other characteristics have changed. We can use data from the first pass, the last pass, or ignore that particular serial number all together (difficult as rework is common). Note that this concern is also extremely important for final test results as well. Final tests are frequently rerun due to equipment, test setup and operator issues. As a result using a last pass test is generally a safe bet, but only if units that have been tested more than once due to rework can be filtered out of the data set.

• Genealogy

Very few manufactured products consist of all top level parts. Our edge analytic, for instance, collects data on a per roll basis. Some number of rolls combine to form a pack with a distinct serial number, and then final test results are generated at the pack level. But we need a single data point to relate X to Y. So how do we aggregate data from the roll level to the pack level in order to get that single X. We can use common aggregates – min, max, average, or a more advanced model. Generally we’ve been averaging, but this will have to be evaluated on a case by case basis.

• Deviations from the Expected Process

For edge analytics to be useful we need to be able to accurately script our process. In winding we set up our RFID filtering to expect one serial number per pack press. For the most part this is true, but in a small number of cases the operators in winding press two packs at the same time. While we can try and detect this, the equipment data is invalid in these cases as pack press force can’t be determined for each individual pack.

• Accuracy of Sensor Data

We generally trust our process and equipment data to be accurate. In many times this isn’t the case. And for edge analytics to be useful we have to keep variation introduced by the measuring
process to a minimum. Gauge R&Rs are a useful study in the six sigma toolkit for evaluating the measurement variation introduced by a specific sensor.

- **Pass/Fail Definition**

Looking for variation of characteristic data vs. yields requires a binary classification of what a pass and a failure is. This isn’t always so black and white. With our final test, for instance, a capacitor is subjected to a barrage of testing – pre-capacitance, dielectric, capacitance, etc. Capacitors can fail before or after dielectric test. Or they can fail for equipment issues. Or they can fail in various ways during dielectric testing itself. Incorrectly classifying passes and failures can remove any chance of seeing variation that’s crucial to solving quality issues.

### 6.2 Solution Transferability to Other Plants

We intended to implement the solution described here in Clearwater, with the intention of transferring it to other plants. Because the solution was implemented using modularized components, it’s very transferrable to other plants. Other RFID management solutions can be used as long as they can publish MQTT messages. Events can be generated from and sent to any MES system. The biggest hurdle in transferring the Clearwater solution to another plant is in connecting machines and building the OPC infrastructure.

The solution also represents a potential service sales opportunity for GE Digital. Most manufacturing plants don’t have the capacity to support or deploy a solution like this. They also don’t have the IT expertise to develop Vel scripts. By leveraging Predix-ready components, GE can deploy and manage this solution in other facilities.
6.3 Results

Note that the findings in the conclusion might be referring to values obtained from edge analytics as “Sensor 1”, “Sensor 2”, etc. This is to protect potentially sensitive company intellectual property. These sensors, for instance, in winding refer to values like machine speed, winding tension and pack press force.

6.3.1 Initial High Voltage Winding Use Case Findings

Capacitors in Clearwater are logically defined as “high” or “low” voltage depending on the application. High voltage units represent 80% of the product volume, so it made sense to analyze these first. The winding edge analytic deployment resulted in the following findings for high voltage capacitors within two weeks of deployment:

- Dielectric yields tend to increase or decrease as pack press force variation increases or decreases. This finding is consistent with the common Six Sigma view that controlling variation improves yields.
- Pack press force variation tends to increase when operators over-press the pack.
- Pack press closing force and winding tension are highly linearly correlated. Four out of the five lines with pack presses have an r-squared value of over 0.7, and we believe that the one with a lower r-squared value is due to inconsistencies in the process.
Over the course of approximately two months we collected data from several hundred units per line. On line 1 we found approximately 148 similar units that appeared to have low dielectric yields. We joined the edge analytic data to test result data to look for shifts between measured characteristics, which are described here as Sensor 1, Sensor 2 and Sensor 3. Not that Sensor 1 and Sensor 2 are averages of each roll in a pack, whereas Sensor 3 does not need to be averaged as the characteristic is measured at the pack level.

Table 7 - Dielectric Failure Means

<table>
<thead>
<tr>
<th>Dielectric</th>
<th>Average of Normalized Sensor 1</th>
<th>Average of Normalized Sensor 2</th>
<th>Average of Normalized Sensor 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pass</td>
<td>0.30964</td>
<td>0.58336</td>
<td>0.11202</td>
</tr>
<tr>
<td>Fail</td>
<td>0.31043</td>
<td>0.55200</td>
<td>0.03430</td>
</tr>
<tr>
<td>All</td>
<td>0.30974</td>
<td>0.57919</td>
<td>0.10170</td>
</tr>
</tbody>
</table>
The initial results show a potential difference between dielectric pass and fail, and Sensor 3. That said, this is just a quick gauge and doesn’t consider outliers and statistical distributions.

A two sample t-test, however, does consider both outliers and distributions. This test on the data set above shows conclusively that there is a statistical difference between distributions with 95% confidence (p=0.005).

![Boxplots of 3 by Dielectric Failure](image)

**Boxplots of 3 by Dielectric Failure**
(means are indicated by solid circles)

**Figure 44 - Pack Press Force Box Plots (1=Dielectric Failure)**

<table>
<thead>
<tr>
<th></th>
<th>N</th>
<th>Mean</th>
<th>StDev</th>
<th>SE Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pass</td>
<td>119</td>
<td>0.104</td>
<td>0.145</td>
<td>0.013</td>
</tr>
<tr>
<td>Fail</td>
<td>19</td>
<td>0.0524</td>
<td>0.0510</td>
<td>0.012</td>
</tr>
</tbody>
</table>

95% CI for mu (0) - mu (1): (0.016, 0.087)
T-Test mu (0) = mu (1) (vs not =): T = 2.89  P = 0.0051  DF = 75

**Figure 45 - Pack Press Force vs. Dielectric Failure T-Test Results**

Furthermore, we found that sensors 2 and 3 appeared to have an interaction effect. The results are encouraging.
Table 8 - Force x Tension vs. Dielectric Failure T-Test Results

<table>
<thead>
<tr>
<th></th>
<th>Avg. of 1 x 2</th>
<th>Avg. of 1 x 3</th>
<th>Avg. of 2 x 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pass</td>
<td>0.18236</td>
<td>0.03562</td>
<td>0.08194</td>
</tr>
<tr>
<td>Fail</td>
<td>0.17429</td>
<td>0.01121</td>
<td>0.02036</td>
</tr>
<tr>
<td>All</td>
<td>0.18129</td>
<td>0.03238</td>
<td>0.07376</td>
</tr>
</tbody>
</table>

Figure 46 - Force x Tension vs. Dielectric Failure Box Plots (1=Dielectric Failure)

Two sample T for 2 x 3

<table>
<thead>
<tr>
<th></th>
<th>N</th>
<th>Mean</th>
<th>StDev</th>
<th>SE Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pass</td>
<td>98</td>
<td>0.082</td>
<td>0.158</td>
<td>0.016</td>
</tr>
<tr>
<td>Fail</td>
<td>15</td>
<td>0.0204</td>
<td>0.0175</td>
<td>0.0045</td>
</tr>
</tbody>
</table>

95% CI for \( \mu (0) - \mu (1) \): (0.029, 0.0944)
T-Test \( \mu (0) = \mu (1) \) (vs not =): \( T = 3.72 \) \( P = 0.0003 \) \( DF = 108 \)

Figure 47 - Pack Press Force x Tension vs. Dielectric Failure T-Test Results

The interaction effect between sensors 2 and 3 by dielectric failure show a very large statistical difference in both mean and distributions (\( p=0.0005 \)) as well as a very low standard deviation in...
the failure set. Note that we did see a very strong correlation between sensors 2 and 3 on most lines which would help to explain this stronger shift.

6.3.3 Low Voltage Insights

While low voltage units make up a much smaller proportion of Clearwater’s volume, they also represent the largest variety in design variation. Because of this, operator skill for building these units isn’t as great, so have traditionally lower yields and, therefore, offer better optimization opportunities.

One specific SKU in Clearwater had a very low yield, approximately 10% below the average yield for the plant. Because the volume of this SKU was relatively high, approximately 450 capacitors, we examined it for potential optimization opportunities.

The analysis had two steps – we first examined the correlation of each sensor variable to dielectric yields using logistic regression. Next, we used the best correlating variables to build a very simple decision tree that fit our data set. Note that due to our limited data set we did not intend to build a predictive model and did not test the decision tree. The purpose of the tree was only to reveal potential opportunities within the data.

The analysis revealed that 25% of failures had a combination of a high value on one sensor as well as a low value on another. Independently, the overall yield for capacitors that had a low value on the first sensor was 42% vs. 82% for the high value set. Optimizing the value on both of these sensors would theoretically lead to a 6% yield improvement if the decision tree is a valid predictive model.
6.4 Minds + Machines

The approach described in this paper, and the Statistical Process Analysis application, was presented at GE’s Minds + Machines Conference in November 2016.

6.5 Sensor Data Control Charts

In addition to driving predictive analytics, this solution also allowed us to create a scheme for sensor driven control charts.

In manufacturing, control charts are statistical charts that draw current measurements, by run, on a chart with +/- N sigma “control limits” generated using historical data. The idea is that if points start trending outside of their historical limits, action can be taken before a problem happens.

Traditionally control charts are created by entering data by hand, but the edge analytics solution provided in this document created an excellent framework for conditioning streams of automatic data for use with control charts. In addition to this, because the data was serialized, each point on the chart could be traced back to a final test yield in order to help prioritize corrective actions.
The control chart in Figure 48 above was created using the D3 JavaScript framework. In order to display sensor data on a chart, it first had to be filtered using an median +/- N IQR statistical filter. Then points were sub-grouped by day and final test yields were displayed to make longer term trends more visible. This generic data preparation process makes any key-value pair collected by edge analytics a candidate for control charts – from winding parameters to capacitor dimensions to process times and WIP counts.

The control charts are very actionable. The chart in Figure 48, for instance, showed that high winding speeds on certain SKUs were lowering dielectric test yields. As a result we were able to lower speeds and counteract the yield reduction. These charts are currently being migrated to the Predix statistical analysis application.
The SQL database contains the following implemented tables, of which some associated with reserved keys:

<table>
<thead>
<tr>
<th>Table</th>
<th>Associated JSON “Header” Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Records</td>
<td>N/A</td>
<td>Contains header data for records</td>
</tr>
<tr>
<td>SerialRecordRelation</td>
<td>serials</td>
<td>Maintains to-many relationship between serial numbers and records</td>
</tr>
<tr>
<td>NumericData</td>
<td>N/A</td>
<td>Stores numeric data associated with CTQs in a record. All data is in reals</td>
</tr>
<tr>
<td>StringData</td>
<td>N/A</td>
<td>Stores string data associated with CTQs in a record. All data is in reals</td>
</tr>
<tr>
<td>CTQs</td>
<td>N/A</td>
<td>Essentially the keys in the key-value pairs</td>
</tr>
<tr>
<td>Items</td>
<td>Item</td>
<td>Item numbers to associate with serials</td>
</tr>
<tr>
<td>Equipment</td>
<td>equipment</td>
<td>Equipment names</td>
</tr>
<tr>
<td>Plants</td>
<td>plant</td>
<td>Plant names</td>
</tr>
<tr>
<td>Processes</td>
<td>process</td>
<td>Process names</td>
</tr>
<tr>
<td>Serials</td>
<td>serials</td>
<td>These are essentially distinct items (ex. capacitors and they’re child parts) that have a serial number and item number</td>
</tr>
</tbody>
</table>

These tables are included in the existing database design but aren’t used yet. They’re added to accommodate future needs. They’re essentially stubs and will not be included in the individual table descriptions below.

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SamplingPlans</td>
<td>SPC sampling plans associated with records (ex. record 6 samples of some measurement once per shift)</td>
</tr>
<tr>
<td>CTQMetaData</td>
<td>Meta data associated with CTQs. Frequently spec limits. Can be used for SPC or to store operating parameters in order to drive alarms. The edge analytic should be able to consume this data as well.</td>
</tr>
<tr>
<td>CTQMetaDataTypes</td>
<td>Types for CTQs – ex. Isl, usl which are commonly defined across many different records</td>
</tr>
</tbody>
</table>
The individual table columns are as follows (FK=Foreign Key, PK=Primary Key):

<table>
<thead>
<tr>
<th>Records</th>
<th>Column</th>
<th>Type</th>
<th>FK</th>
<th>PK</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>id</td>
<td>int</td>
<td></td>
<td>X</td>
<td>Identity</td>
</tr>
<tr>
<td></td>
<td>timestamp</td>
<td>datetime</td>
<td></td>
<td></td>
<td>The time stamp of the record insert</td>
</tr>
<tr>
<td></td>
<td>startTime</td>
<td>datetime</td>
<td></td>
<td></td>
<td>Start time from reserved key</td>
</tr>
<tr>
<td></td>
<td>endTime</td>
<td>datetime</td>
<td></td>
<td></td>
<td>End time from reserved key</td>
</tr>
<tr>
<td></td>
<td>processTime</td>
<td>long</td>
<td></td>
<td></td>
<td>Process time from reserved key</td>
</tr>
<tr>
<td></td>
<td>line</td>
<td>int</td>
<td></td>
<td></td>
<td>Associated line</td>
</tr>
<tr>
<td></td>
<td>station</td>
<td>Int</td>
<td></td>
<td></td>
<td>Associated station</td>
</tr>
<tr>
<td></td>
<td>samplingPlanID</td>
<td>int</td>
<td>SamplingPlans.id</td>
<td></td>
<td>Associated sampling plan</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NumericData</th>
<th>Column</th>
<th>Type</th>
<th>FK</th>
<th>PK</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>recID</td>
<td>int</td>
<td>Records.id</td>
<td>X</td>
<td>Associated record</td>
</tr>
<tr>
<td></td>
<td>ctqRelationID</td>
<td>int</td>
<td>CTQEquipmentProcessRelation.id</td>
<td>X</td>
<td>The CTQ-Equipment-Process combo associated with this record</td>
</tr>
<tr>
<td></td>
<td>value</td>
<td>float</td>
<td></td>
<td></td>
<td>The numeric value</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>StringData</th>
<th>Column</th>
<th>Type</th>
<th>FK</th>
<th>PK</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>recID</td>
<td>int</td>
<td>Records.id</td>
<td>X</td>
<td>Associated record</td>
</tr>
<tr>
<td></td>
<td>ctqRelationID</td>
<td>int</td>
<td>CTQEquipmentProcessRelation.id</td>
<td>X</td>
<td>The CTQ-Equipment-Process combo associated with this record</td>
</tr>
<tr>
<td></td>
<td>value</td>
<td>varchar(20)</td>
<td></td>
<td></td>
<td>The string value</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CTQEquipmentProcessRelation</th>
<th>Column</th>
<th>Type</th>
<th>FK</th>
<th>PK</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>id</td>
<td>int</td>
<td></td>
<td>X</td>
<td>identity</td>
</tr>
<tr>
<td></td>
<td>ctqID</td>
<td>int</td>
<td>CTQs.id</td>
<td></td>
<td>The CTQ id associated with this relation</td>
</tr>
<tr>
<td>equipmentID</td>
<td>int</td>
<td>Equipment.id</td>
<td>The equipment id associated with this relation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------</td>
<td>--------</td>
<td>--------------</td>
<td>-----------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>processID</td>
<td>int</td>
<td>Processes.id</td>
<td>The process id associated with this relation</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Serials

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>FK</th>
<th>PK</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>int</td>
<td>X</td>
<td></td>
<td>identity</td>
</tr>
<tr>
<td>serialNumber</td>
<td>varchar(20)</td>
<td></td>
<td></td>
<td>The text serial number</td>
</tr>
<tr>
<td>itemID</td>
<td>int</td>
<td>Items.id</td>
<td></td>
<td>The item id associated with this serial number</td>
</tr>
<tr>
<td>parentID</td>
<td>int</td>
<td>Serials.id</td>
<td></td>
<td>The parent serial id – used for maintaining genealogy</td>
</tr>
</tbody>
</table>


