Ontology-Guided Knowledge Discovery of Event Sequences in Maintenance Data

Michael Schuh, John Sheppard, Shane Strasser, Rafal Angryk, Clemente Izurieta
Department of Computer Science
357 EPS Building
Montana State University, Bozeman, MT 59717
michael.schuh@cs.montana.edu

***Abstract*—We created an application that facilitates improved knowledge discovery from aircraft maintenance data by transforming transactional database records into ontology-based event graphs, and then providing a filterable visualization of event sequences through time. We developed OWL ontologies based on formally defined IEEE standards, and use these ontologies to guide the data mining and data transformation processes. Our application removes much of the users burden for data look-up and greatly increases the potential for knowledge discovery from data (KDD) in this field. We provide an easy-to-use interface that generates relevant sequences of data in a meaningful context in a fraction of the time it would take domain experts to retrieve and display similar information.**

I. Introduction

Modern diagnostic systems can generate an overwhelming abundance of data. Often this data is distributed across multiple heterogeneous systems and cannot be immediately (or easily) collected and aggregated for use. Even with access to data, there can be an enormous amount of automatically logged information with interesting content sparsely intermixed. Therefore, a large burden is put on the user to coalesce the data and mine the interesting bits relevant to their current needs. Depending on the task at hand, this amount of effort may not be justifiable or practical, and the potential for knowledge discovery is lost. We developed a general-use application that visualizes aircraft maintenance data in a meaningful and concise manner, and allows a user to quickly retrieve interesting and relevant subsets of data. Our application removes much of the users burden for data look up and greatly increases the potential for knowledge discovery from data (KDD) within the maintenance community.

This work focused on aircraft data collected at ground-based maintenance facilities. The data are composed of transactional records detailing each maintenance event that takes place, including important fields such as the aircraft tail number, type and date of the event, and possible part(s) being removed or installed. We use existing data model standards, defined by the Institute of Electrical and Electronics Engineers (IEEE), and derive new ontological models to better represent the data. With our ontologies, we then transform the raw data into maintenance events and provide the user with a query interface to filter on specific event attributes. The filtered query generates a chronological sequence of maintenance events and the user can optionally display links between events that belong to the same aircraft or share the same remove/install part(s). Each event in a sequence can be inspected individually to view its entire ontology-based graph of data attributes.

The application we developed provides better accessiblity of existing data sources to the experts who need them. The foundational framework enables novel data exploration and KDD while still integrating with existing standards and data collection formats. While the software can contribute greatly to the productiveness of experts in this specific domain, the underlying data analysis principles and visualization techniques are universal and easily adaptable to other domains.

The remainder of the paper is organized as follows. Section II discusses our use of ontology-guided data mining and data transformation. Section III provides an overview of our application, and the detailed implementation is discussed in Section IV. We highlight aspects worth further discussion in Section V and present several directions for future work in Section VI. Finally, we close with our conclusions in Section VII. As a convenience to readers, Table I provides a list of frequently used abbreviations.

II. Ontology-Guided Data Mining

We utilized domain ontologies to join together different data sources and aggregate individual records into more meaningful models. In information science, an ontology is a type of knowledge representation that formally defines concepts, their properties, and the relationships between them. This well-defined representation enables automated methods of reasoning and analysis into the domain concepts the ontology describes. Previous work by Wilmering and Sheppard suggested using domain ontologies to focus and filter data analysis in data mining [1]. The approach we take in developing ontologies to support the knowledge discovery process is based on a set of standardized semantic models developed in the EXPRESS modeling language [2]. EXPRESS is an information modeling language defined by the International Organization for Standardization (ISO) to support communication of product data between engineering applications. The purpose of the language is to define the semantics of information that will be generated by a system, and it is not meant to define database formats, file formats, or exchange formats.

Models in EXPRESS are defined using a hierarchy partitioned along schemata, entities, and attributes [3]. The EX-

TABLE I
FREQUENTLY USED ABBREVIATIONS

AI-ESTATE	Artificial Intelligence Exchange and Service Tie to All Test Environments
SIMICA	Software Interface for Maintenance Information Collection and Analysis
MAID	Maintenance Action Information Document
ME	Maintenance Event
MA	Maintenance Action
ML	Maintenance Level
ACID	Aircraft Identification
JCN	Job Control Number
UNS	Unified Numbering System
PartNo	Item (Removed or Installed) Part Number
SerNo	Item (Removed or Installed) Serial Number

PRESS language incorporates a number of object-oriented features, such as encapsulation, abstraction, and inheritance, and it additionally allows logical constraints to be placed on attribute values. These constraints, which often define relationships in non-trivial ways, give EXPRESS the ability to define computer-processable semantics, which allows applications to discern if the information being received satisfies the intended meaning when it was generated and transmitted [3].

This application used ontologies derived from the IEEE Std. 1232 Artificial Intelligence Exchange and Service Tie to All Test Environments (AI-ESTATE) and IEEE Std 1636 Software Interface for Maintenance Information Collection and Analysis (SIMICA) [4], [5]. AI-ESTATE is a set of specifications for exchanging data and defining software services for diagnostic systems. Its purpose is to standardize the diagnostic data representations of an intelligent diagnostic reasoner and the interfaces between elements of such reasoners. The information models defined for AI-ESTATE are designed to form the basis for facilitating exchange of persistent diagnostic information between two reasoners through a standardized system for diagnostic services. Additionally, both models make use of a "common" information model (called the Common Element Model) [4]. Our data is primarily represented by the SIMICA Maintenance Action Information (MAI) model, which was designed to capture records of actual maintenance actions performed on a particular system or subsystem [6].

Recent work in ontology-guided data mining has made use of standard ontology languages (e.g. OWL [7], DAML+OIL [8], and RDF [9]). EXPRESS was not designed to support ontology-based analysis; however, the semantics defined by EXPRESS models are rich enough to use as the foundation for defining ontologies in the Web Ontology Language (OWL), which is one of the most widely used ontology languages. An OWL ontology may have descriptions of classes, properties, and their data instances, and the formal OWL semantics then specify how to find logical consequences from the defined entities. Given an OWL ontology, we can then define and instantiate data in OWL format [10], [7]. To convert EXPRESS to OWL, we first had to define a logical mapping from the general EXPRESS concepts to OWL concepts (e.g., an EXPRESS *entity* becomes an OWL *class*). We then used the mapping to create our OWL ontologies from the existing standards in EXPRESS. Figures 1 and 2 present a small sample of this conversion process for the *Maintenance Event* component. Notice the similarity between each format, such as the "actionTaken" and "delayReason" relationships present in both.

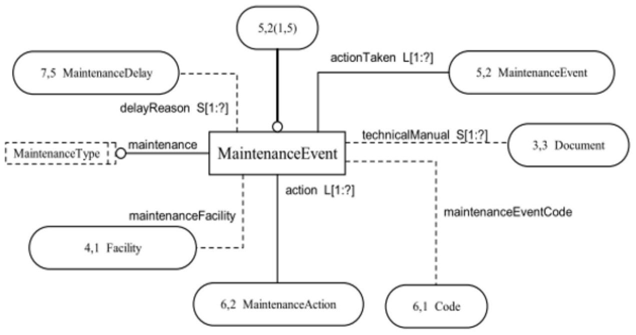
III. EVENT GRAPHS AND SEQUENCES

We define an *event* as a group of records pertaining to a unique set of ground-based maintenance actions. Each event is composed of one or more transactional database records grouped together based on the SIMICA MAI model, which defines the elements and attributes of our OWL ontology. The Maintenance Action Information Document (MAID) element is the root of our MAI ontology, and its attributes are specified as unique, representing the super-key of each event. The existence of multiple records with the same super-key value indicates multiple Maintenance Actions (MA) were performed for a single Maintenance Event (ME). Therefore, the ME ontology element contains a list of MAs, and is directly connected to the root (MAID) element. Refer to the model in Figure 2 to see the direct relationship between MAID, ME, and MA.

Since the raw records are transactional, these multi-actioned events typically contain a pair of remove/install actions or a list of timely inspection actions. In other words, each database record is essentially an MA element, stored with its MAID attributes. By aggregating these events into ontological graph structures, we are performing something similar to a database conversion from 1st normal form to 3rd normal form, where each key now returns only a single event graph [11].

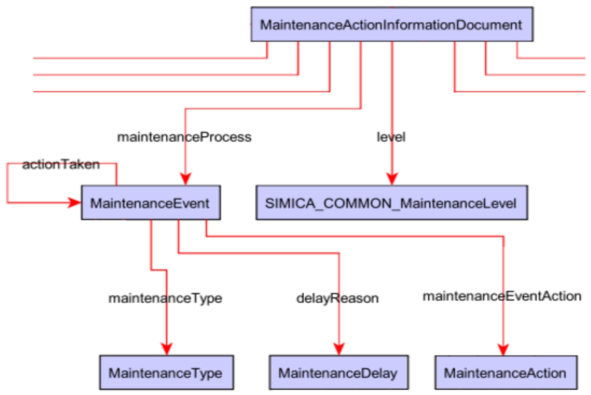
After the data is transformed into events it must then be presented to the user. We develop a method of displaying the summarized events as a sequence through time. Each event is reduced to a single node, and arranged in sequence by one of the user-specified date attributes: JCN (Job Control Number) Date, or MA Completion Date. The sequence can be considered a further abstraction of the data, where each event only displays the date and associated aircraft, as well as all UNS (Unified Numbering System) and item serial numbers. These additional attributes (UNS, and item serial number) are not part of the unique key, but were suggested by domain experts as the most valuable information to display. The entire application window is shown in Figure 3, with the sequence being displayed in the large panel on the right.

To simplify the readability of the interface, the sequence contains three distinct layers. The top-most layer contains the date of each event through time, and the aircraft tail number is displayed in the middle layer directly below the event date. The bottom layer is composed of the single node events, linked vertically to their corresponding aircraft above. This layer of event nodes is further segmented into three separate levels corresponding to the Maintenance Level (ML) attribute value for each event (with ML 1 the top-most / closest to the aircraft,



```
ENTITY MaintenanceEvent;
  maintenanceEventCode     :OPTIONAL Code;
  action                   :LIST [1:?] OF MaintenanceAction;
  maintenanceFacility      :OPTIONAL Facility;
  technicalManual          :OPTIONAL SET [1:?] OF Document;
  actionTaken              :LIST [1:?] OF MaintenanceEvent;
  delayReason              :OPTIONAL SET [1:?] OF MaintenanceDelay;
  maintenance              :MaintenanceType;
END_ENTITY;
```

Fig. 1. A sample of an EXPRESS model and related code.



```
  <owl:Class rdf:ID="MaintenanceEvent">
    <rdfs:label rdf:datatype="XMLSchema#string"
    >Maintenance event</rdfs:label>
    <rdfs:subClassOf rdf:resource="#SIMICA_MAI"/>
  </owl:Class>

  <owl:ObjectProperty rdf:ID="maintenanceProcess">
    <owl:cardinality rdf:datatype="XMLSchema#nonNegativeInteger"
    >1</owl:cardinality>
    <rdfs:range rdf:resource="#MaintenanceEvent"/>
    <rdfs:domain
rdf:resource="#MaintenanceActionInformationDocument"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="maintenanceFacility">
    <rdfs:domain rdf:resource="#MaintenanceEvent"/>
    <rdfs:range rdf:resource="#SIMICA_COMMON_Organization"/>
  </owl:ObjectProperty>
```

Fig. 2. A sample of an OWL model and related code.

and ML 3 at the bottom). Finally, we create links between events that have the same aircraft, or item serial number, to help the user follow items of interest through the larger sequence. This is a critical enhancement to KDD, as it quickly and easily allows a user to trace the context of specific parts or aircraft through time.

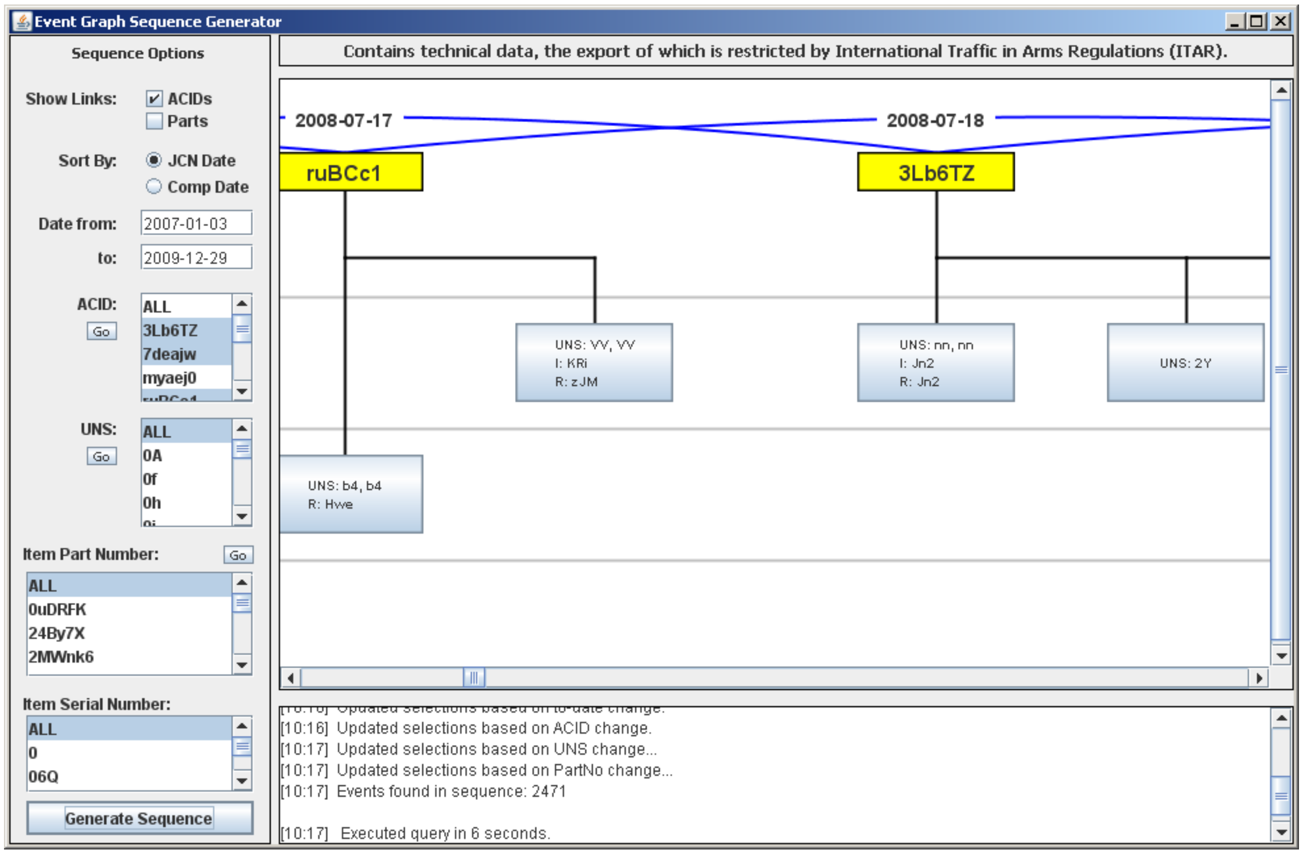
An additional utility-turned-feature was inspired by the clearance restrictions of our dataset. These restrictions made discussion (and presentation) of purely software-related development quite cumbersome because of the data-dependent nature of our application. Therefore, we developed a data pre-processor which anonymizes sensitive attributes' values while retaining the relationships between records. After anonymization, the application is used exactly like before, except the "clean" data now replaces the original (sensitive) data. Furthermore, the anonymized records automatically result in anonymized events and query options, allowing the application to be presented in normal public circumstances – such as the screen shots appearing in this paper. Note however, because of the inherent randomness built into our algorithm, the resultant attribute values make little logical sense to the human observer.

IV. Implementation

The application implementation can be separated into five parts: (1) an initial (optional) raw data anonymization, (2) a necessary data transformation into ontology instances, (3) attribute value filtering and querying, (4) the display of event sequences, and (5) the display of individual event graphs. While the first two parts represent one-time pre-processing steps, the remaining three parts are performed dynamically during normal application use. All code was written in Java (v1.6), and MySQL (v5.1) was used as our database management system.

The optional first step is a pre-processing anonymization function that transforms the data into publicly-viewable content. This algorithm essentially creates a mathematical correspondence mapping between the original data values, and the randomly generated new values. Each attribute retains its specific qualities, such as data type and length, and the correspondence ensures all relationships within the data are preserved. We iteratively process each sensitive attribute, requiring at most O(n) memory at one time (the new-values-set), where n is the largest number of unique values for any attribute. When all possible values of an attribute are found (onto requirement) and given a mapping (one-to-one requirement), the new values are updated in the database and the next attribute is processed. While we casually mention the space complexity here, we add that if necessary, the algorithm could preserve the new-values-set on disk, sacrificing time but reducing space to a constant cost. We found this was not necessary for our initial implementation, as the application is, in general, much more focused on time complexity than space requirements.

The second step is a required data transformation from transactional records representing pieces of maintenance events, to entire event graphs that aggregate all those pieces into a common OWL ontology instance. Since our specific application is only concerned with connection-based relationships among the data, defined by the ontology, we can more


Event Graph Sequence Generator

Sequence Options

Show Links: [✓ ACIDs] Parts

Sort By: [✓ JCN Date] Comp Date

Date from: 2007-01-03

to: 2009-12-29

ACID: ALL, 3Lb6TZ, 7deajw, myaej0

UNS: ALL, 0A, 0f, 0h

Item Part Number: ALL, 0uDRFK, 24By7X, 2MWnk6

Item Serial Number: ALL, 0, 06Q

Generate Sequence

Contains technical data, the export of which is restricted by International Traffic in Arms Regulations (ITAR).

```
[10:16]  Updated selections based on ACID change.
[10:17]  Updated selections based on UNS change...
[10:17]  Updated selections based on PartNo change...
[10:17]  Events found in sequence: 2471

[10:17]  Executed query in 6 seconds.
```

Fig. 3. The main application window. The left panel contains the query options to filter on event attributes, and the right panel contains the generated sequence based on the query. Below the sequence is a small status window that provides information and feedback to the user.

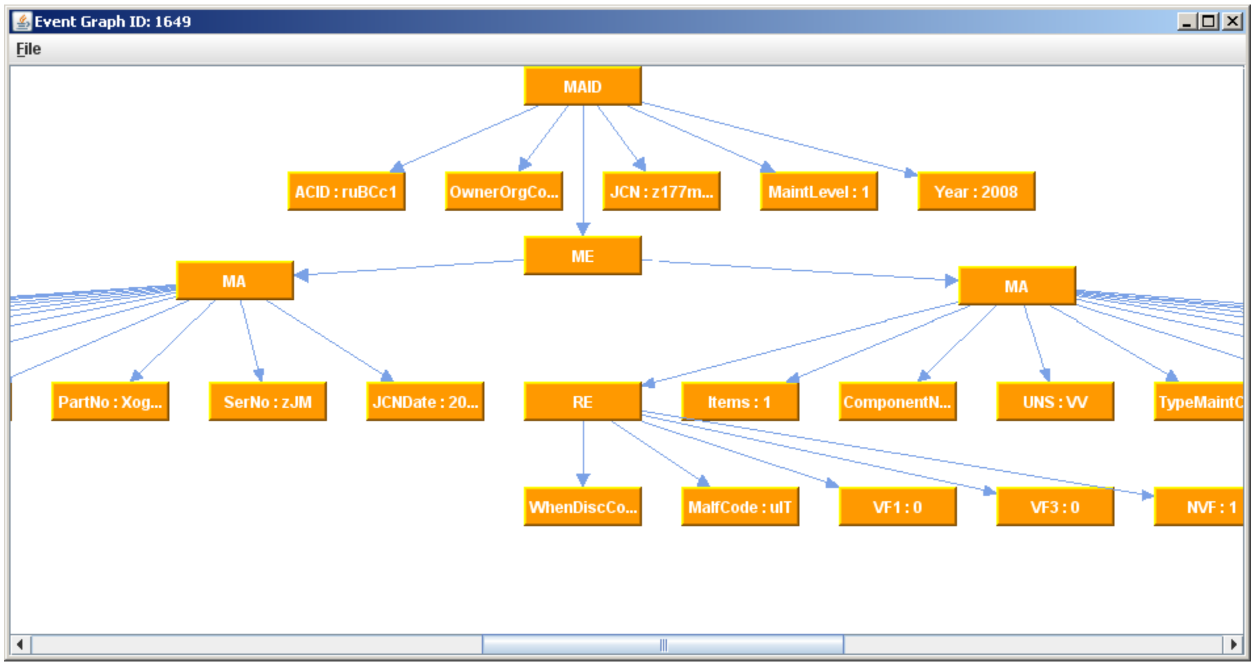


Fig. 4. An individual event window containing an auto-generated layout of all the attributes (and their values) for a given event. The unique event ID is displayed in the window title bar and the graphical visualiztion can be saved as an image.

simply, and efficiently, store the transformed data as graph objects – requiring only a few assumptions. Recall that our ontology defines a root element, MAID, which contains the keys for each instance, as well as an attached ME list of MA elements, each of which essentially encapsulates an original data record. In our graph representation, we define nodes in a tree-like context, either as internal nodes (e.g. MAID, ME, MA), or as leaf nodes which represent attributes associated with an internal node. Therefore, given an internal node, all connected leaf nodes are its attributes, and all connected internal nodes are further extensions of the ontology structure, such as MAID to ME, ME to MA #1, ME to MA #2, etc. Additionally, each node contains an identifier that defines it as a specific element of the ontology, and internal nodes contain a list of connected nodes, while leaf nodes contain their respective data value. The node ID is important, because it allows for quick and easy identification of any node anywhere within the ontology, and that identification provides further information about the node, such as its name and attribute value type used during visualization. We use the JGraphT library [12] to create these graphs with custom nodes, and store them as serialized objects in our database.

After all records for a single event are added to a graph, we store the event keys (which again are the MAID attributes, or leaf nodes), as well as the two date fields used for sorting and the serialized graph object in a new database table – separate from the raw data. We also create a look-up table to store a cross-relation of every UNS, item part number, and item serial number associated with each event (a many-to-many relation). At this point, the original data is no longer needed since the application runs entirely on the two new tables. This provides a convenient mechanism for dumping the data into the application as it is accumulated incrementally. Also note that both pre-processing steps are self-contained within the application and initiated with specific command line arguments.

With the data transformation complete, the user can now access the main application window for interactive KDD by applying a filter to the query and then viewing the event sequence and individual events. When the application is started, the user is presented with several query options in the left panel (refer to Figure 3) which are ordered from top to bottom, and from least to most specific. The only required options are "Sort By", which orders the events by JCN Date or MA Completion Date, and "Date To" - "Date From", which restrict the sequence to only events within the given date range. To aid the user, we provide the default selection of JCN Date, and auto fill the date range with the minimum and maximum JCN dates found in the database.

One complication that arose with our method of record aggregation is the possibility of conflicting date values for actions in the same event. This is a known problem with the data, and we resolved it by storing the earliest found JCN and MA Completion dates as the date fields for an event. The remaining query options: ACID (Aircraft ID), UNS (Unified Numbering System), PartNo (item part number), and SerNo (item serial number) provide further filtering capabilities, and the valid options for each filter are auto-generated based on the previously selected filters. For example, when a user selects a set of ACIDs that are of interest, the subsequent filters (UNS, PartNo, and SerNo) are repopulated to display only valid values found in database records which contain one of the selected ACIDs. This removes the guessing game of identifying which records exist, and it allows the user to gain considerable insight into the data they are investigating – even before the first query is performed.

After the query options are set, the sequence graph is generated and displayed in the right side panel of the application window. While the application executes, important messages, query history, and other information is displayed in the bottom status panel and optionally logged to a file. A sequence can be scrolled (left and right) through time and each event is a click-able object that displays the details of the individual event. The sequence graph is generated using Java's Swing layout mechanism with the semantic links being overlaid by Java's 2D drawing framework. These links connect identical aircraft (ACID) and item parts (SerNo) from one occurrence to the next, as time flows to the right. It is important to note that time is not scaled proportionally, and is used solely as a method of sorting the events sequentially in a sequence.

Finally, when an event is clicked, a separate window is opened which displays all the details of the event, such as in Figure 4. Each clicked event opens a new window, allowing easier side-by-side comparison of multiple events. Viewing individual events is accomplished quite easily by porting objects from JGraphT to JGraph [12], which provides an on-the-fly layout and visualization of the graph object. These event graphs can be further manipulated (dragged, reshaped, etc.) and then saved as an image (.png) for further presentation and discussion outside of the application.

V. Discussion

The primary objective of this software application is to provide a meaningful display of data collected from maintenance records. This was achieved with the grouping and ordering of records, and the summarized display of only certain key information. Beyond this general use case, the application allows detailed query selection based on the most useful parameters identified by domain experts. Because of the large (and continually growing) set of data being accessed by the application, this allows the user to better select only the appropriate data they are interested in.

The large volume of data stored behind this application is a point of concern, as it could have effects on maintaining timely queries. Our current implementation provides no safe-guards towards checking for manageable and effective user-generated queries. Through MySQL, we maintain a separate index for all query-able attributes, but the shear volume of data combined with an ill-minded query can still bring the application to a brief stand-still. It is easy to understand this dilemma by considering the process of successive choice decisions in mathematics. For example, suppose we have four successive

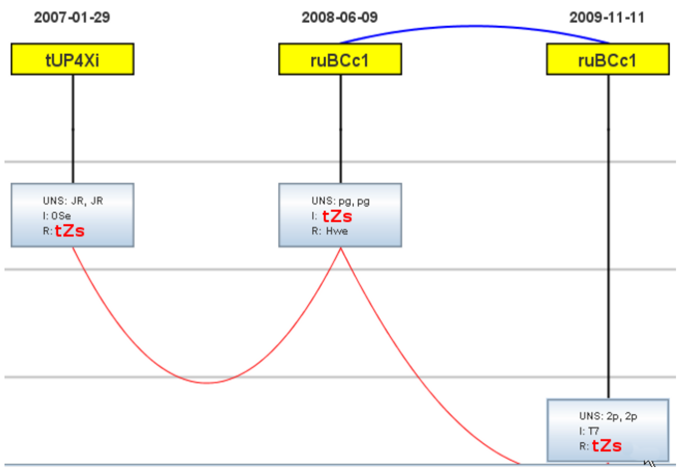


Fig. 5. A sequence of events.

filters, each with 100 possible items to choose from – so if we choose ten from each, then we have $(100 \text{ choose } 10)^4$ which is nearly 10^{53} possible combinations! The problem is that a query like this has no defining attribute to filter on that provides adequate data reduction, whereby MySQL can optimize the query and perform the most selective joins first.

The good news for us is that these vague and indiscriminate queries are rarely helpful to real-world users, and should therefore be encountered minimally. For example, a user rarely needs to see a large set of aircraft, related to a large set of items, over a large span of time; rather, they are more often interested in a specific aircraft and a handful of parts, or a specific part on any aircraft. These practical queries return results in seconds. Likewise, the readability of our dynamic links between neighboring chronological events that share identical aircraft or item serial numbers depend on a defined query that agrees with the users needs. If someone is attempting to follow links for a part or aircraft from one event to another, they probably are not looking at a lot of parts or aircraft. However, if the user deems a large query necessary, these links are easily toggled off for a clean view of the event sequence.

Beyond the idea of just querying items, the application has the added benefit of allowing a user to follow interesting items through time. This is achieved in part by the aforementioned query options, but also by the dynamically generated item links displayed on the sequence. With these features, a user can for example: generate a summary of specific items through time, discover a re-occurring list of similar problems to a specific aircraft, or track a specific part from aircraft to aircraft as it is perhaps repetitively cannibalized or replaced. All of these uses could benefit from this novel data visualization.

We provide an example use case in Figures 5 and 6. Although the data in these figures is anonymized like the rest, we can still walk through the important knowledge discovery abilities. Presented in Figure 5 is a sequence of three events – all quite separate in time. The top arc indicates the same aircraft *ruBCc1* in the second and third events, while the bottom arc indicates the same item is referenced in all three events. The item is identified by its serial number *tZs*, which we simply highlighted for readability. Now we can essentially read the event sequence "story". In early 2007, the *tZs* item was removed from aircraft *tUP4Xi* during a Level 1 Maintenance Action. Then, in the middle of 2008, the same item was installed on aircraft *ruBCc1*, only to be removed over a year later during a Level 3 Maintenance Action. According to domain experts, this most likely indicates a failed part which was later fixed and cannibalized only to fail once again.

To investigate further, the user could then inspect the other attributes of each event in the sequence. The graphical visualization of an event can be manipulated to highlight the important attributes before being saved for discussion and investigation outside the application. Figure 6 shows an example of the modified event graph for the third event in the sequence in Fig. 5. Notice we can now clearly see the event's MAID node connected to the ME node and the ME node connected to two MA nodes. Each MA node has three visible attributes that indicate the item being removed or installed. Important text fields would also be present here (in real data), and they would likely confirm our previous insights.

VI. Future Work

The application we developed provides a promising and exciting framework for continued data mining and knowledge discovery research. Interesting continuations of this work include adding more data sources, applying graph-based data mining algorithms to the data, and performing an in-depth analysis and mining of text field attributes.

A benefit of conforming to the IEEE standards-based ontologies is the well established data model that already provides connections between multiple sources of maintenance data. A common data source with an existing connection in the ontology is on-board (and in-flight) Built-In Test (BIT) data. We received a sample BIT dataset, but did not have enough time to warrant inclusion in the initial implementation. By filling out the existing ontology with available information sources, the application can expand and provide the user a more complete context surrounding an interesting event, or sequence of events.

Since our data is essentially a database of graphs, another interesting research direction would be exploring graph-based data mining algorithms. Angryk's previous work in frequent subgraph mining [13] shared a similar problem formulation, where the goal was to detect all frequently occurring subgraphs (based on a given support threshold) from a larger graph object. This is similar to frequent itemset mining, except instead of a set of items we use a set of edges, representing a subgraph [14].

Similarly to Angryk's use of an ontology as a *master document graph* in text-mining [13], we can use our ontology as a "master event graph", retaining the computation speed-ups gained by this assumption. While the set of frequent subgraphs lends itself to further data mining applications, even simple analysis could provide some beneficial knowledge.

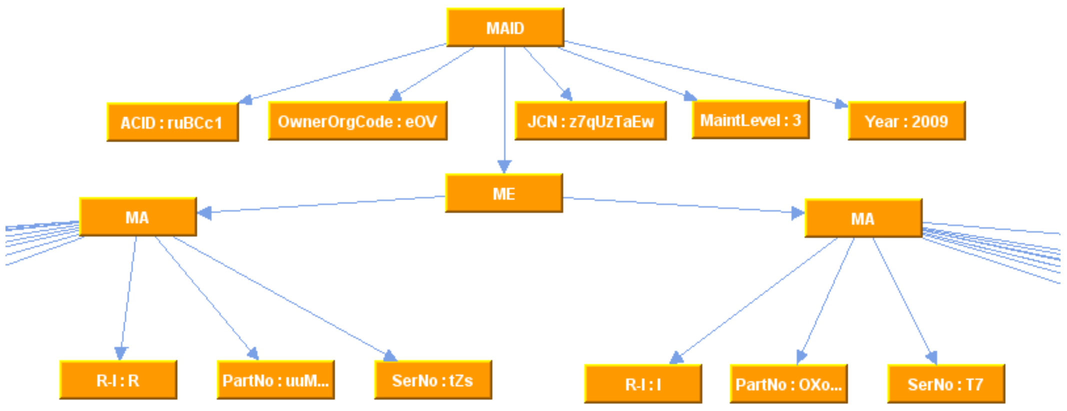


Fig. 6. A user-modified event graph.

For example, a frequent subgraph might indicate that several events always occur together when accompanied by certain attribute values. Perhaps this is a series of items to replace after a specific malfunction, then if the malfunction occurs and only triggers some of the associated events, an operator could be informed that other events commonly occur in these circumstances and they probably deserve attention too.

The ground-based maintenance data we used has two very important text fields – the description narration, which describes details of the task or problem of the event, and the corrective action, which describes the actions taken to fix or complete the event task. Both fields are entered manually by human operators and contain a variety of shorthand and abbreviations – as well as spelling mistakes and input errors – that truly require a domain expert for proper interpretation. However, the benefits of understanding and incorporating these fields would be enormous, as a great deal of information is conveyed solely within the text, including referrals to other events, parts, and problems. Furthermore, simple keyword analysis (and perhaps tagging of events) would detect common phrases like: "routine inspection", "see job #", "cannibalization of item #", etc., and provide great opportunities to explore clustering on these phrases as an alternative view to the chronological sequence display.

VII. Conclusion

This paper describes the creation of an application which facilitates improved knowledge discovery within maintenance data by transforming data records into ontology-based event graphs, and providing a filterable visualization of event sequences through time. We accomplish several major preprocessing objectives, such as data anonymization, records-to-ontology event mapping, and the resolution of date conflicts in the aggregated records of events.

The most beneficial aspect of our application is the timely display of filtered event sequences. Rather than take a domain expert hours to coalesce and display the relevant data records, our application can be used to generate relevant data in a contextual display in a matter of seconds. This work provides the foundations for further investigative research on a variety of topics in this area that could greatly benefit the maintenance community.

References

[1] T. Wilmering and J. Sheppard, "Ontologies for Data Mining and Knowledge Discovery to Support Diagnostic Maturation," in *Proceedings of the 18th International Workshop on Principles of Diagnosis (DX-07)*, 2007, pp. 210–217.
[2] ISO, "10303-11:2004, Industrial automation systems and integration – Product data representation and exchange – Part 11: Description methods: The EXPRESS language reference manual," 2004.
[3] J. Sheppard, M. Kaufman, and T. Wilmering, "Model based standards for diagnostic and maintenance information integration," in *Autotestcon, 2007 IEEE*. IEEE, 2002, pp. 304–310.
[4] *IEEE Std 1232-2010, Standard for Artificial Intelligence Exchange and Service Tie to All Test Environments (AI-ESTATE)*, IEEE Standards Press, Piscataway, New Jersey, 2011.
[5] *IEEE Std 1636-2010, Trial-Use Standard for Software Interface for Maintenance Information Collection and Analysis (SIMICA)*, IEEE Standards Press, Piscataway, New Jersey, 2010.
[6] *IEEE Std 1636.2-2010, Trial-Use Standard for Software Interface for Maintenance Information Collection and Analysis (SIMICA): Maintenance Action Information (MAI)*, IEEE Standards Press, Piscataway, New Jersey, 2010.
[7] W3C, "OWL 2 Web Ontology Language Document Overview," http://www.w3.org/TR/owl2-overview/, 2009.
[8] Agent Markup Language Committee, "DAML+OIL," http://www.daml.org/2001/03/daml+oil-index, 2001.
[9] W3C, "Resource Description Framework (RDF)," http://www.w3.org/RDF/, 2004.
[10] P. Hitzler, M. Krtzsch, and S. Rudolph, *Foundations of Semantic Web Technologies*, 1st ed. Chapman & Hall/CRC, 2009.
[11] R. Elmasri and S. B. Navathe, *Fundamentals of Database Systems (5th Edition)*. Addison-Wesley Longman Publishing Co., Inc., 2006.
[12] JGraphT, "JGraphT a free Java Graph Library," http://www.jgrapht.org/, 2011.
[13] M. S. Hossain and R. A. Angryk, "Gdclust: A graph-based document clustering technique," in *IEEE International Conference on Data Mining*, 2007, pp. 417–422.
[14] J. Han and M. Kamber, *Data Mining: Concepts and Techniques, Second Edition*. Morgan Kaufmann, 2006.