An IEEE Standards-Based Visualization Tool for Knowledge Discovery in Maintenance Event Sequences

Michael Schuh, John Sheppard, Shane Strasser, Rafal Angryk, Clemente Izurieta
Montana State University

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 easy access to data, much of it is uninteresting to the user's specific inquiry. This puts a large burden 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. Our tool facilitates this data mining process and generates relevant sequences of data in a fraction of the time it would take domain experts to retrieve and display similar information.

We present this extended update from our previously published work at AUTOTESTCON 2011 [1]. As development continues on our application, it has now grown into a full-fledged tool that incorporates additional visualizations and analysis methods, including the necessary inputs required for the method of diagnostic maturation developed in [2]. The tool further removes the user's burden for data lookup and aggregation, and it greatly increases the potential for knowledge discovery from data (KDD) within the maintenance community.

This work primarily focused on aircraft data collected at ground-based maintenance facilities. The data are composed of transactional records detailing each maintenance action (MA) 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. Importantly, this data also contains the necessary corrective actions that were performed as well as insightful text-fields detailing the results of these actions. Also, we now integrate onboard (in-flight) data that indicates which faults were observed to prompt the performed maintenance event (ME) actions.

Authors' current address: M. Schuh, J. Sheppard, S. Strasser, R. Angryk, C. Izurieta, Department of Computer Science, 357 EPS Building, Montana State University, Bozeman, Montana 59717, USA, e-mail: michael.schuh@cs.montana.edu. Manuscript SYSAES-2012-0023 received January 25, 2012, and ready for publication March 11, 2013. Review handled by W. Walsh.
0885-8985/13/ $26.00 ©2013 IEEE

We used existing data model standards, defined by the Institute of Electrical and Electronics Engineers (IEEE), and derived new ontological models to better represent the data. We then transform the raw data into MEs and provide the user with a query interface to filter on specific event attributes. The filtered query generates a chronological sequence of MEs, and the user can optionally display links between events that belong to the same aircraft or share the same remove/install part(s). Additionally, the user now has the ability to view these sequences to scale of the actual time elapsed between events. Each event in a sequence can also be inspected individually, displaying the entire ontology-based graph from interconnected data sources.

By combining ME sequences with their underlying faults and corrective actions, the user is provided with a much more complete context of maintenance history. The model maturation tool uses this data in conjunction with timed failure propagation graph (TFPG) models [3], allowing the user to go beyond standard diagnostic procedures. The maturation process provides recommendations about possible errors in the diagnostic models used to guide MAs. If errors exist in a model, the resulting diagnosis and corrective actions may be incorrect, causing wasted time, money, and effort during the maintenance process [4].

The ongoing tool development provides increased accessibility of existing (and related) data sources to the experts who need them. This increases novel data exploration and KDD by integrating additional existing standards and data collection formats. Through continued integration, the user will waste no time switching to different tools and analysis methods for the same event sequence of interest.

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, ontology is a type of knowledge representation that formally defines concepts, their properties, and the relationships between them [5]. 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 [6].

```
  <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>
```

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 [7]. EXPRESS is an information modeling language defined by the International Organization for Standardization 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.

Models in EXPRESS are defined using a hierarchy partitioned along schemata, entities, and attributes [8]. The EXPRESS 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 nontrivial 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 [8].

The tool uses ontologies derived from the IEEE Std. 1232 Artificial Intelligence Exchange and Service Tie to All Test Environments (AI-ESTATE) [9] and IEEE Std 1636 Software Interface for Maintenance Information Collection and Analysis (SIMICA) [10]. 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) [9].

MEs are primarily represented by the SIMICA MA information (MAI) model, which was designed to capture records of actual MAs performed on a particular system or subsystem [11]. The AI-ESTATE D-Matrix Inference Model and Dynamic Context Model are used to define the diagnostic models and associated information used and produced by diagnostic reasoners, including recommended corrective actions.

Recent work in ontology-guided data mining has made use of standard ontology languages (e.g. OWL [12], DAML+OIL [13], and RDF [14]). 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 [12].

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. In some cases, the data did not match the entire EXPRESS models, and the newly created OWL ontologies had to be extended beyond the defined standards. With the incorporation of additional models and data source, we also have to ensure a unified ontology that links each piece accurately given the available data fields. Figures 1 and 2 present a small sample of this conversion process for the ME component. Notice the similarity between each format, such as the "actionTaken" and "delayReason" relationships present in both.

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

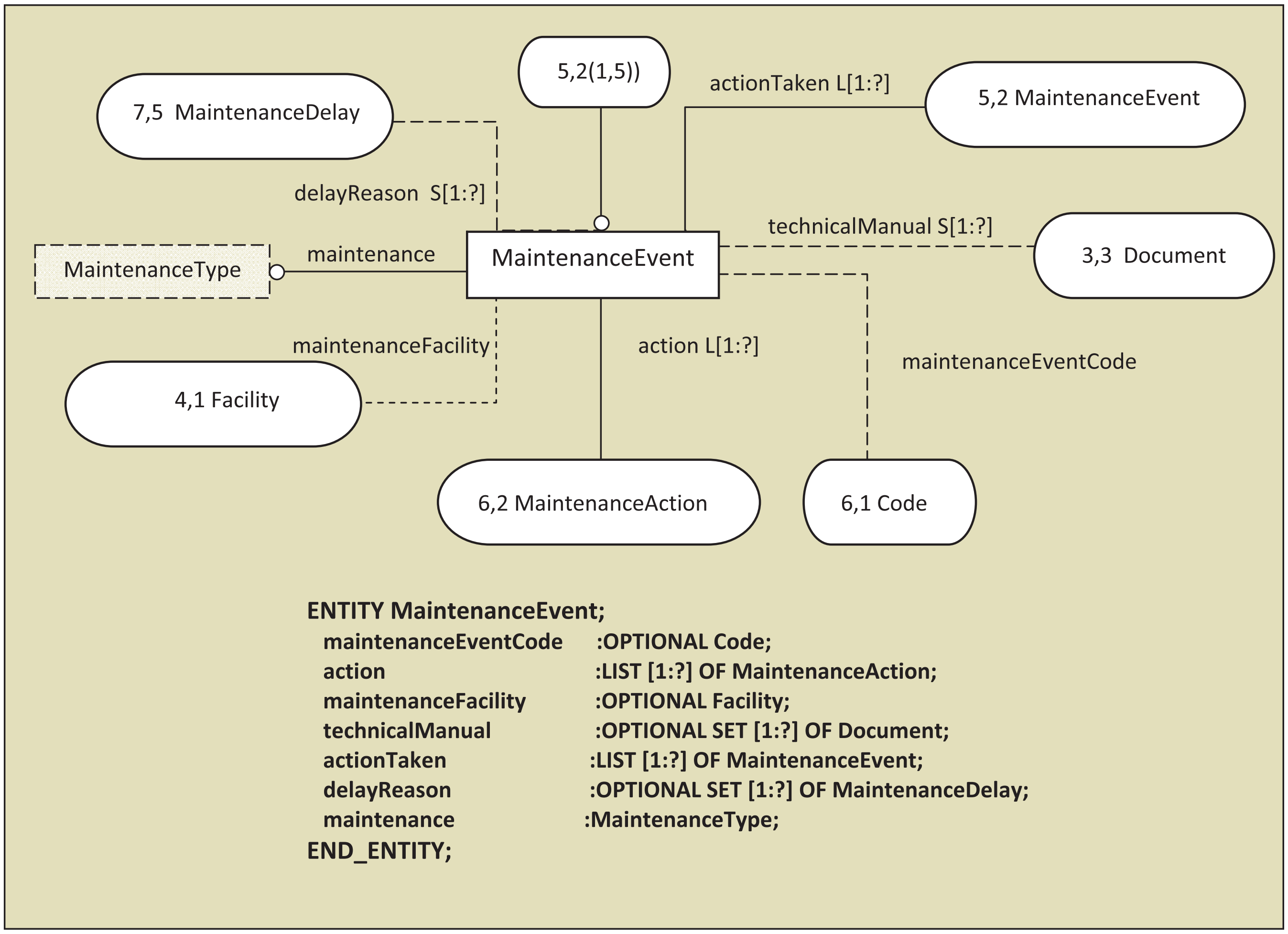
ACID: aircraft identification

JCN: job control number

UNS: unified numbering system

PartNo: item (removed or installed) part number

SerNo: item (removed or installed) serial number



```
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;
```

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

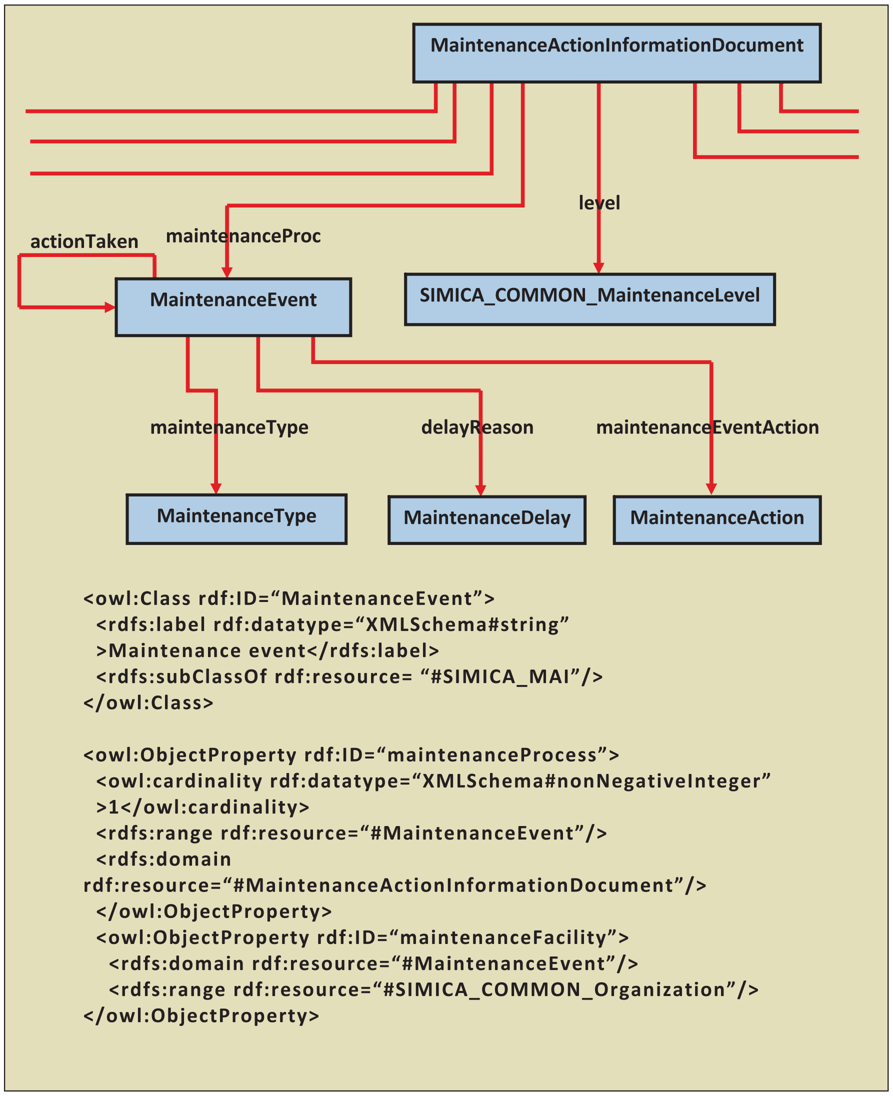
EVENT GRAPHS AND SEQUENCES

We define an *event* as a group of records pertaining to a unique set of ground-based MAs. 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 MA information document (MAID) element is the root of our MAI ontology, and its attributes are specified as unique, representing the superkey of each event. The existence of multiple records with the same superkey value indicates multiple MAs were performed for a single 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.

The tool also incorporates onboard diagnostic data from the aircraft. This data contains a chronological list of diagnostic tests (and results) that were performed during the course of a flight. Faults that were observed during a flight lead to ground-based MAs that are meant to fix and resolve the faults [21]. The TFPG model maturation approach analyzes the statistical discrepencies between the reasoner's action recommendations and the corrective actions (on the ground) that actually fixed the problem [2].

Because the data sources have transactional records, these multiactioned corrective events typically contain a pair of remove/install actions or a list of timely inspection actions. In other words, each ground-based record is essentially an MA element, stored with its MAID attributes. Similarily, each record of onboard data contains the identifying attribute values shared in common by all, such as aircraft identification (ACID). By aggregating these events into ontological graph structures, we are performing something similar to a database conversion from first normal form to third normal form, where each key now returns only a single event graph [16].

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: job control number (JCN) 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



```
<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>
```

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

as all unified numbering systems (UNSs) and item serial numbers (SerNos).

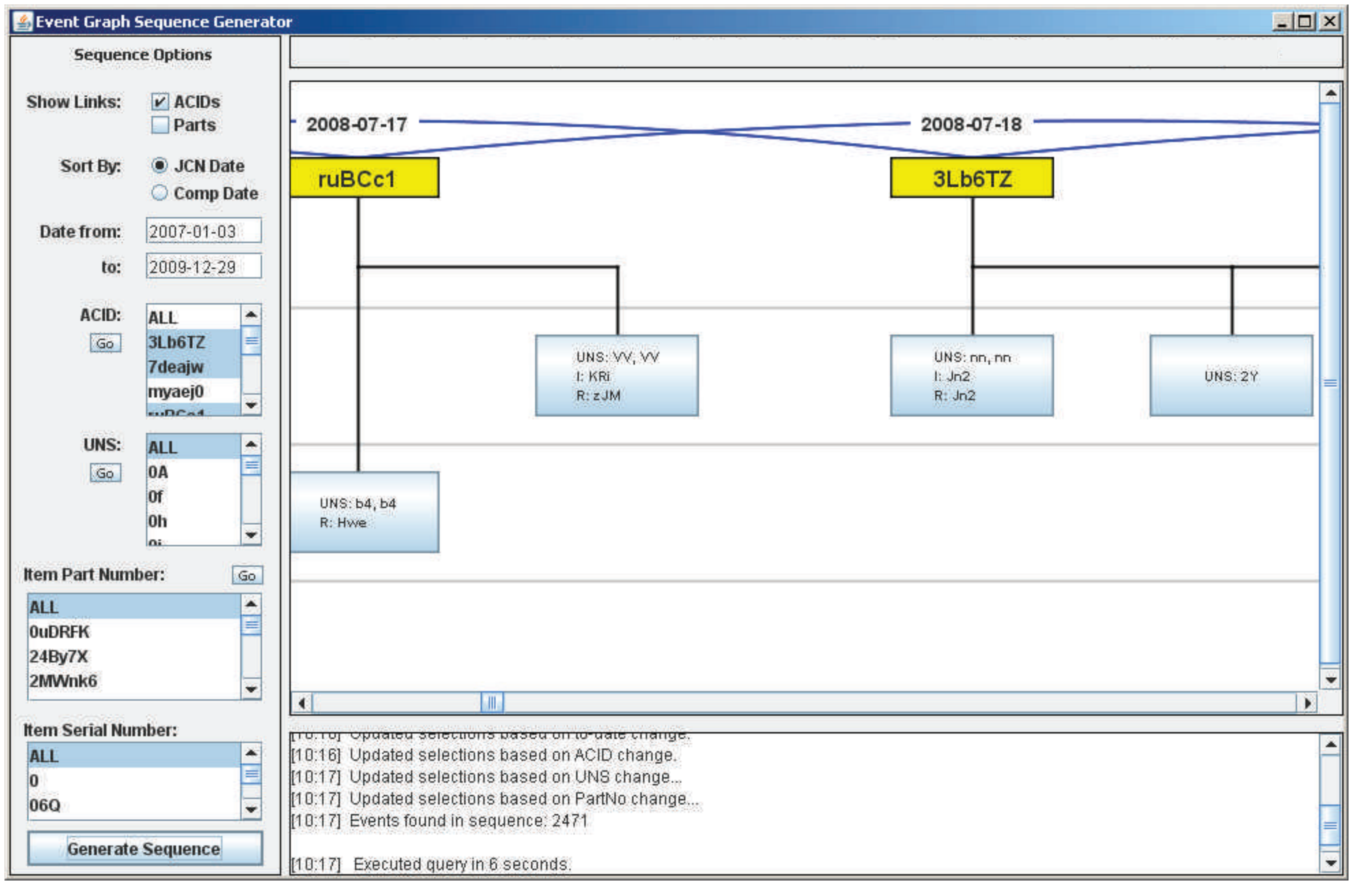
These additional attributes (UNS and SerNo) are not part of the unique key but were suggested by domain experts as the most valuable information to display. The entire application window is shown in Figure 3, with the sequence being displayed in the large panel on the right. To simplify the readability of the interface, the sequence contains three distinct layers. The topmost 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 topmost/closest to the aircraft and ML 3 at the bottom). Finally, we create links between events that have the same aircraft, or SerNo, 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.

The default sequence view is not scaled in proportion to event time/date occurrences. Instead, each event has uniform separation between all actions and other events in the generated sequence. This makes the most sense when a user is interested in following specific links or looking for specific events, regardless of when they occurred. Alternatively, the user can choose to view the sequence in a true time-proportional scale. This can be more beneficial when investigating the history of specific parts or aircraft, where a quick glance can give easy indication of time between events.

An additional utility-turned feature was inspired by the nature of the data. We developed a data preprocessor that 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. Unfortunately, because of the inherent (and necessary) randomness built into our anonymization algorithm, the resultant attribute values make little logical sense to the human observer. Especially important attributes, such as text-based corrective actions, are not even retained, as their only benefit is being able to actually read and infer more information from them. This also means presenting interesting case studies is difficult, especially for the TFPG model maturation, which we encourage the reader to refer to [2] for a previously published example.

IMPLEMENTATION

The application implementation can be separated into five parts: an initial (optional) raw data anonymization, a necessary data transformation into ontology instances, attribute value filtering and querying, the display of event sequences, and the display of individual event graphs. While the first two parts represent one-time preprocessing 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.



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

```
[10:16] Updated selections based on to-date change.
[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.
```

Figure 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.

The optional first step is a preprocessing anonymization function that transforms the data into clean and safe content for general presentation. 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.

The second step is a required data transformation from transactional records representing pieces of MEs to entire event graphs that aggregate all those pieces into a common OWL ontology instance. Because we are only concerned with connection-based relationships among the data, defined by the ontology, we can more 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 ground-based data record. The MAID element also links to the onboard diagnostic records from the previous flight(s) that match the data keys.

In our graph representation, we define nodes in a treelike context, either as internal nodes (e.g., MAID, ME, MA), or as leaf nodes that 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 entire ontology, and that identification provides further information about the node, such as its name and attribute value type used during visualization and analysis methods. We use the JGraphT library [17] 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 (PartNo), and SerNo associated with each event (a many-to-many relation). At this point, the original data is no longer needed because the tool runs entirely on the two new tables. This provides a convenient mechanism for dumping the data into the backend database as it is accumulated incrementally. Also note that both preprocessing steps are self-contained within the tool 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 program 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 autofill 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, UNS, PartNo, and SerNo 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 that 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 using the tool, information, such as query history and important messages, are 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. As discussed earlier, the sequences presented in figures are not scaled by time and instead use the original default graphical view. There is currently active development on the GUI framework to more easily support adding additional views and user interactions.

Finally, when an event is clicked, a separate window is opened that 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 [18], which provides an on-the-fly layout and visualization of the graph object. These event graphs can be further manipulated (dragged, reshaped, etc.) to highlight attributes of interest, while hiding the unimportant ones. At any time, the visualized graphs can be saved as a image (.png format) for presentation and discussion beyond the confines of the tool and the computer system it is running on.

DISCUSSION

The primary objective of this software tool is to minimize wasted time and effort during aircraft maintenance by providing a meaningful display and integration of data collect-

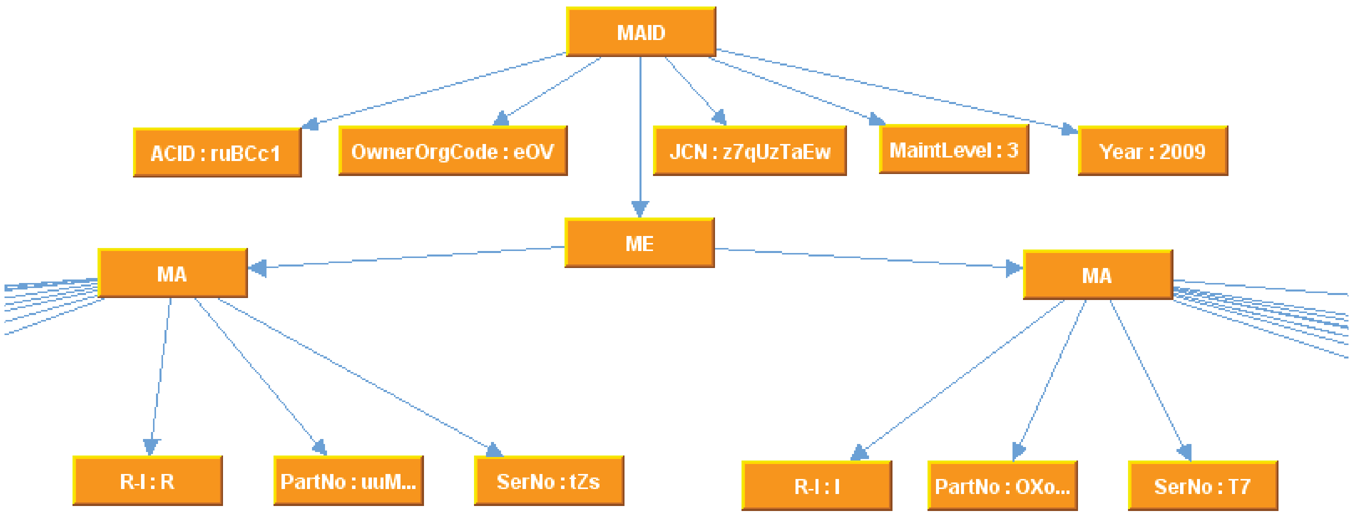


Figure 4.
An individual event window containing an auto-generated layout of all the attributes (and their values) for a given event. The unique event ID is displayed in the window title bar, and the graphical visualiztion can be saved as an image.

ed from ground-based and onboard maintenance records. This was achieved with the grouping and ordering of records and the summarized display of only certain key information. The application facilitates detailed query selections based on the most useful parameters identified by domain experts. Because of the large (and continually growing) set of data being accessed, this allows the user to select only the appropriate data that they are interested in.

The potentially large volume of stored data is a point of concern, as it could have effects on maintaining timely queries. Our current implementation provides no safeguards toward checking for manageable and effective user-generated queries. Through MySQL, we maintain a separate index for all queryable attributes, but the sheer volume of data combined with an ill-minded query can still bring the application to a brief standstill. For example, suppose we have four successive filters, each with 100 possible items to choose from—so if we choose ten from each, then we have (100 choose 10) times 4, which is nearly 1053 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.

"Links are created between events that have the same aircraft, or SerNo, to help the user follow specific parts or aircraft through time."

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.

In a similar argument, the readability of our dynamic links between neighboring events that share aircraft or parts in common can become muddled in an overly complicated query. However, if someone is attempting to follow links for a part or aircraft from one event to another, they are probably not looking at a lot of parts or aircraft. If for some reason the user deems a large or overly complicated query is 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

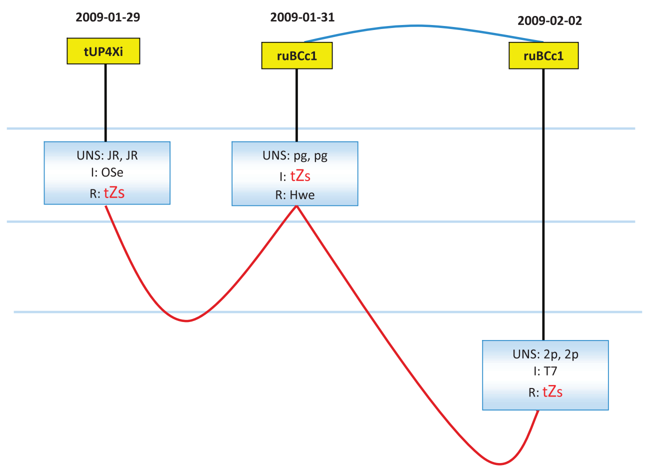


Figure 5.
A sequence of events.

item links displayed on the sequence. With these features, a user can, for example, generate a summary of specific items through time, discover a reoccurring 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 these novel data visualizations.

We provide an example use case in Figures 5 and 6, and we walk through the important knowledge discovery abilities displayed. Presented in Figure 5 is a sequence of three events. 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 2009, the *tZs* item was removed from aircraft *tUP4Xi* during a Level 1 MA. Then, only days later, the same item was installed on aircraft *ruBCc1*, only to be removed a few more days later during a Level 3 MA. According to domain experts, this most likely indicates a failed part that 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 Figure 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.

The investigation of MEs is dramatically increased with the aid of TFPG diagnostic model maturation recommendations through the method described in [2]. The maturation process requires a combined analysis of the diagnosed faults found onboard, and the corrective actions reported

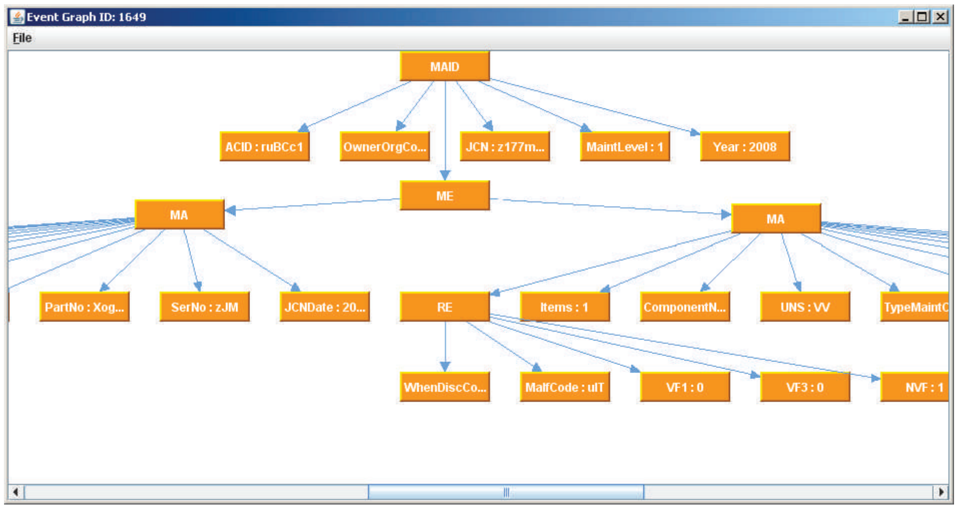


Figure 6.
A user-modified event graph.

to fix the problem—both of which are available in our ME graphs. Essentially, the recommendations arise when a corrective action is performed more often than expected, given the reasoner's diagnosis from observed faults or vice versa, when a suggested corrective action rarely fixes the actual problem. These indicate a potential problem with the underlying diagnosis model, and the user could be better informed by knowing in advance these discrepancies may exist.

FUTURE WORK

The tool provides a promising and exciting framework for continued data mining and knowledge discovery research. Interesting continuations of this work include adding more data sources and tools, enhanced GUI interactions and visualizations, 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. While we have already added an onboard diagnostic data source to the application, additional data sources will continue to enhance KDD through more comprehensive analyses and incorporated tools. By filling out the existing ontologies with more available information sources, we can provide the user a more complete context surrounding an interesting event or sequence of events. Similar to the facilitated TFPG model maturation recommendations, further integration with independent tools will increase the efficiency of aircraft maintenance and minimize costly mistakes that would be otherwise unavoidable.

As we mentioned earlier, current and ongoing work has been focused on enhancing the GUI for the user. Additional views of aggregated maintenance data, such as the time-scaled event sequences, provide unique benefits for an overall better understanding of the data. New visualizations also create the opportunity for new user interactions with the system, and entirely new tools may arise during such development. Predefined layouts of individual event graphs would also speed up the detailed investigation of event attributes, minimizing the need for the user to extensively manipulate these graphs every time one is viewed.

Because 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 [19] 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 [20].

"The graphical visualization of an event can be manipulated to highlight the important attributes before being saved for discussion and investigation outside the application."

Similar to Angryk's use of an ontology as a *master document graph* in text-mining [19], we can use our ontology as a "master event graph," retaining the necessary computational speedups 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. 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, such as "routine inspection," "see job #," "cannibalization of item #," and provide great opportunities to explore clustering on these phrases as an alternative view to the chronological sequence display.

CONCLUSION

This paper described the initial and ongoing development of a tool that facilitates improved knowledge discovery within maintenance data by transforming data records into ontology-based event graphs and providing several filterable visualizations of event sequences through time. We accomplish several major preprocessing objectives, such as records-to-ontology event mapping and the resolution of date conflicts in the aggregated records of events.

The most beneficial aspect of our software is the quick look up and display of filtered event sequences. Rather than take a domain expert hours to coalesce and display the relevant data records, our tool can generate a comprehensive sequence of contextual MEs from several data sources in a matter of seconds. This work continues to highlight a variety of research topics that could greatly benefit the maintenance community. ❖

REFERENCES

[1] Schuh, M., Sheppard, J., Strasser, S., Angryk, R., and Izurieta, C. Ontology-guided knowledge discovery of event sequences in maintenance data. In *Proceedings of the IEEE AUTOTESTCON 2011 Conference*, Baltimore, MD, Sep. 2011, 279–285.

[2] Strasser, S., Sheppard, J., Schuh, M., Angryk, R., and Izurieta, C. Graph-based ontology-guided data mining for D-matrix model maturation. In *Proceedings of the 2011 Aerospace Conference*, Big Sky, MT, Mar. 2011.

[3] Abdelwahed, S., Karsai, G., Mahadevan, N., and Ofsthun, S. Practical implementation of diagnosis systems using timed failure propagation graph models. *IEEE Transactions on Instrumentation and Measurement*, Vol. 58, 2 (2009), 240–247.

[4] Wilmering, T. J. Semantic requirements on information integration for diagnostic maturation. In *Proceedings of the 2001 AUTOTESTCON. IEEE Systems Readiness Technology Conference*, Valley Forge, PA, Aug. 2001, 793–807.

[5] Gruber, T. Toward principles for the design of ontologies used for knowledge sharing. *International Journal of Human-Computer Studies*, Vol. 43, 5–6 (1995), 907–928.

[6] Wilmering T., and Sheppard, J. Ontologies for data mining and knowledge discovery to support diagnostic maturation. In *Proceedings of the 18th International Workshop on Principles of Diagnosis (DX-07)*, Nashville, TN, May 2007, 210–217.

[7] International Organization for Standardization, Industrial automation systems and integration—Product data representation and exchange—Part 11: Description methods: The EXPRESS language reference manual. ISO 10303-11:2004, 2004.

[8] Sheppard, J., Kaufman, M., and Wilmering, T. Model based standards for diagnostic and maintenance information integration. In *Proceedings of the 2007 IEEE AUTOTESTCON*. Baltimore, MD, Sep. 2002, 304–310.

[9] Standard for Artificial Intelligence Exchange and Service Tie to All Test Environments (AI-ESTATE). IEEE Std 1232-2010. Piscataway, NJ: IEEE Standards Press, 2011.

[10] Trial-Use Standard for Software Interface forMaintenance Information Collection and Analysis (SIMICA). IEEE Std 1636-2010. Piscataway, NJ: IEEE Standards Press, 2010.

[11] Trial-Use Standard for Software Interface forMaintenance Information Collection and Analysis (SIMICA): Maintenance Action Information (MAI). IEEE Std 1636.2-2010. Piscataway, NJ: IEEE Standards Press, 2010.

[12] World Wide Web Consortium. OWL 2 Web Ontology Language document overview, http://www.w3.org/TR/owl2-overview/, 2009.

[13] Agent Markup Language Committee. DAML+OIL, http://www.daml.org/2001/03/daml+oil-index, 2001.

[14] World Wide Web Consortium. Resource Description Framework (RDF), http://www.w3.org/RDF/, 2004.

[15] Hitzler, P., Krötzsch, M., and Rudolph, S. *Foundations of Semantic Web Technologies* (1st ed.). Boca Raton, FL: Chapman & Hall/CRC, 2009.

[16] Elmasri R., and Navathe, S. B. *Fundamentals of Database Systems* (5th ed.). Boston: Addison-Wesley Longman Publishing Co., Inc., 2006.

[17] JGraphT. JGraphT a free Java graph library, http://www.jgrapht.org/, 2011.

[18] JGraph. JavaScript and Java diagram library components, http://www.jgraph.com/, 2011.

[19] Hossain M. S., and Angryk, R. A. GdClust: a graph-based document clustering technique. In *Proceedings of the Seventh IEEE International Conference on Data Mining Workshops*, Omaha, NE, Oct. 2007, 417–422.

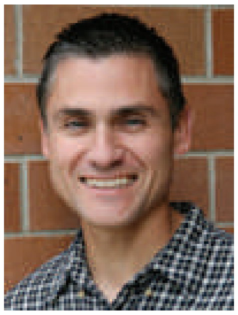
[20] Han J., and Kamber, M. *Data Mining: Concepts and Techniques,* (2nd ed.). Waltham, MA: Morgan Kaufmann, 2006.

[21] Byington, C. S., Kalgren, P. W., and Donovan, B. P. Portable diagnostic reasoning for improved avionics maintenance and information capture & continuity. In Proceedings of the 2004 *IEEE AUTOTESTCON 2004 Conference*, Sep. 2004, 518–524.

ABOUT THE AUTHORS

Michael Schuh received a BS in Computer Science with math and business minors from the University of Wisconsin in 2009. He has since been a graduate student researcher in Computer Science at Montana State University where he received his MS in 2012 and continues to pursue his PhD. His primary research interests revolve around large scale data mining with a focus on high-dimensional data indexing, content-based image retrieval, and similarity-based kNN search. He also has a strong emphasis on interdisciplinary research with solar physics and the massive solar image repositories they produce, archive, and access. Michael has won several awards in recent years, including the 2011 AUTOTESTCON Best Student Paper Award, a 2012 outstanding MSU Computer Science PhD researcher award, and a 2013 MSU College of Engineering Benjamin Fellowship.

Dr. John Sheppard is a Professor and RightNow Technologies Fellow in the Department of Computer Science at Montana State University and an Adjunct Professor in the Department of Computer Science at Johns Hopkins University. In 2007, he was elected as an IEEE Fellow "for contributions to system-level diagnosis and prognosis." Prior to joining Hopkins, he was a Fellow at ARINC Incorporated in Annapolis, MD where he worked for almost 20 years. Dr. Sheppard performs research in Bayesian classification, dynamic Bayesian networks, evolutionary methods, and reinforcement learning and is active in IEEE Standards activities. Currently, he serves as a member of the IEEE Computer Society Standards Activities Board and is the Computer Society liaison to IEEE Standards Coordinating Committee 20 on Test and Diagnosis for Electronic Systems. He is also the co-chair of the Diagnostic and Maintenance Control Subcommittee of SCC20 and has served as an official US delegate to the International Electrotechnical Commission's Technical Committee 93 on Design Automation.



Clemente Izurieta is an associate research professor in the Computer Science Department at Montana State University. Born in Santiago, Chile, his research interests include empirical software engineering, design and architecture of software systems, design patterns, the measurement of software quality and ecological modeling. Dr. Izurieta has approximately 16 years experience working for various R&D labs at Hewlett Packard and Intel Corporation.

Rafal Angryk is an Associate Professor in the Computer Science Department at Montana State University (MSU). He is also the founding Director of MSUís Data Mining Laboratory, and holds the title of Affiliate Professor of Physics due to the interdisciplinary research he is conducting on massive repositories of solar data. He received his M.S. and Ph.D. in Computer Science from Tulane University. Dr. Angryk's research and teaching interests lie in the areas of Very Large Databases (Spatial and Spatio-temporal Databases, and kNN Indexing), Data Mining (Frequent Patterns Discovery, Clustering and Classification of real-life large-scale data), and Information Retrieval (Text and Image data). He has published over 60 journal articles, book chapters and peer-reviewed conference papers in these areas.

Shane Strasser received a BS in computer science and mathematics from the University of Sioux Falls in Sioux Falls, South Dakota. Afterwards he went on to obtain an MS in Computer Science at Montana State University, where he is currently working on his PhD in Computer Science. While at Montana State, he has received several awards, such as the 2012 outstanding PhD MSU Computer Science researcher and the 2011 AUTOTESTCON Best Student Paper Award. In the spring of 2013, he also joined Oracle Right Now as a software developer intern working in the areas of search and artificial intelligence. His research interests are primarily in artificial intelligence and machine learning with a focus on prognostics and health management systems.