An Integrated Toolset for Ontology-Guided Diagnostic Knowledge Discovery

Shane Strasser, Eben Howard, John Sheppard
Department of Computer Science
Montana State University
Bozeman, MT 59717
{shane.strasser, eben.howard, john.sheppard}@cs.montana.edu

***Abstract*—Today's diagnostic systems can generate a large amount data. Data from sources such as onboard reasoners and historical maintenance data are often stored in heterogeneous systems and cannot be collected immediately and aggregated for use. In our previous work we described a software visualization tool that allowed integration of different data sources and displayed the data with elements organized according to maintenance-oriented ontologies. This tool allows users to search quickly through available data to locate interesting relationships in the sequences of maintenance events. Additional previous work described a diagnostic maturation tool, called ModelMat, that updates causal relationships in a Timed Failure Propagation Graph based on historical diagnostic session data. In this paper, we present an update to both of these projects discussing enhancements to each as well as work in progress to create a single, integrated toolset, called Bobcat, to support ontology-guided diagnostic knowledge discovery.**

I. Introduction

Modern diagnostic systems can generate an overwhelming amount of data. For example, many onboard systems record not only test or alarm outcomes and diagnostic results, but also environmental variables, such as temperature, location, and date. In addition, a large amount of data is generated from maintenance events, including information from automatic test equipment (ATE) and maintenance/repair actions taken in response to fault isolation. While in theory this data can be used for knowledge discovery, the actual task of mining the data is non-trivial because the data is distributed across multiple heterogeneous systems [1]. Even after the data is collected, determining what parts of the data are correlated can be challenging. There is the added burden on the user to consider the results of analysis and locate interesting or relevant information. All of these issues highlight the need for a software tool that allows the users to first take data from various sources and aggregate them together in an intelligent way and then enables the users to inspect, analyze, and assess whatever new knowledge has been discovered.

One method that has been proposed to combine data from heterogeneous systems is to use domain ontologies as an "information integrator." Ontologies allow one to associate the semantics of the data across the different data sources. The result is that, from a user's perspective, all of the data looks like a single unified data source containing all of the relevant information within that domain [2]. In prior work, Wilmering and Sheppard proposed a method that used ontologies to focus and filter data that would then be used in the knowledge discovery process. In that work, the authors showed how ontologies could be used to guide the process of maturing diagnostic models. Furthermore, the authors proposed using a method such as the Apriori algorithm [3] to discover new relationships within historical maintenance data that could then be used in diagnostic model maturation [2].

We extended Wilmering and Sheppard's work by mapping diagnostic models and historical diagnostic session data to two ontologies derived from IEEE Standards [4]. In that paper, we mapped the diagnostic reasoner information to the IEEE Std 1232 Standard for Artificial Intelligence Exchange and Service Tie to All Test Environments (AI-ESTATE) [5] and the maintenance history to IEEE Std 1636.2 Software Interface for Maintenance Information Collection and Analysis (SIMICA): Maintenance Action Information (MAI) [6]. We redefined these IEEE models, which were standardized using the EXPRESS language [7], using the Web Ontology Language (OWL) [8]. The main focus of that paper focused on the role ontologies can play in diagnostic model maturation.

Another extension of Wilmering and Sheppard's work involved the use of ontologies in knowledge discovery by visualizing aircraft maintenance data in a meaningful way [9]. In that paper, a visualization tool was presented that takes transactional records detailing each maintenance event that occurred at ground-based maintenance facilities. The tool converts the transactional records into a ontology-based instance graph using the IEEE Std 1636.2. Users can query a database to filter the information, and the tool displays a chronological sequence of maintenance events. The software allows users to click on each maintenance event, which then displays the ontology-based graph of the event.

In this paper, we present the results of our work integrating updated versions of the visualization tool software presented in our previous work, called Event Grapher Gold (EGG) [9] and the model maturation tool, ModelMat [10]. Previously, the EGG software only supported the visualization of data obtained from ground-based maintenance events. EGG now supports displaying sequences of onboard monitoring data conforming to the ontology derived from SIMICA Test Results (IEEE Std 1636.1-2007) [11]. In addition, the software now supports the ability to export data, formatted according to the underlying ontologies, that can be used in knowledge discov-

978-1-4673-0700-0/12/$31.00 ©2012 IEEE

ery algorithms. Finally, there have been several performance upgrades that allows the software to run more smoothly.

Similarly, the ModelMat tool has been updated to incorporate maturing sequence-based relationships, as would be required by a Timed Failure Propagation Graph [12]. In addition, to facilitate integration with EGG, ModelMat was rewritten in Java with several optimizations and a new user interface.

The remainder of the paper is organized as follows. Section II discusses the IEEE standards and the ontologies that were derived from them, followed by Section III, which gives an explanation of Event Graphs and how they are related to ontological graphs in EGG. Section IV provides an overview of the EGG application, followed by a discussion of knowledge discovery algorithms in VI. A discussion of the integrated toolset comprising EGG and ModelMat is then presented in VII. Finally, we give a list of future work along with our conclusions in Section VIII.

II. IEEE Standards and OWL

In information science, an ontology is a type of knowledge model that formally defines concepts, their properties, and the relationships between concepts. These well-defined models enable automated methods of reasoning and analysis focusing on the domain concepts that an ontology describes. Ontologies have been used for several different applications, such as aiding in communication, interoperability of software, and the integration of heterogeneous information sources [13]. In addition, Wilmering and Sheppard suggested using domain ontologies as a means to focus and filter data analysis in the process of knowledge discovery [2]. The specific focus of that work was on utilizing the ontologies to guide the process by which diagnostic models could be matured over time. In this paper we use domain ontologies to join together different data sources and aggregate individual records into more meaningful models.

One issue with domain ontologies is the process by which they are created. In our work, we addressed this issue by basing our ontologies on information models taken from several IEEE standards. Specifically, we utilized IEEE Std 1232 AI-ESTATE and IEEE Std 1636.2 MAI [5], [6]. AI-ESTATE is a set of specifications for exchanging data and defining software services for diagnostic systems. The information models defined for AI-ESTATE are designed to form the basis for facilitating exchange of persistent diagnostic information between two reasoners and to provide a formal typing system for diagnostic services [5]. MAI specifies the means to exchange records of actual maintenance actions performed on a particular system or subsystem, also based on formal information models [6].

In this work, we also utilized IEEE Std 1636.1 Software Interface for Maintenance Information Collection and Analysis (SIMICA): Exchanging Test Results and Session Information via the Extensible Markup Language(XML) (Test Results) [11]. The Test Results standard defines an interface for access, exchange, and analysis of test result information that arises

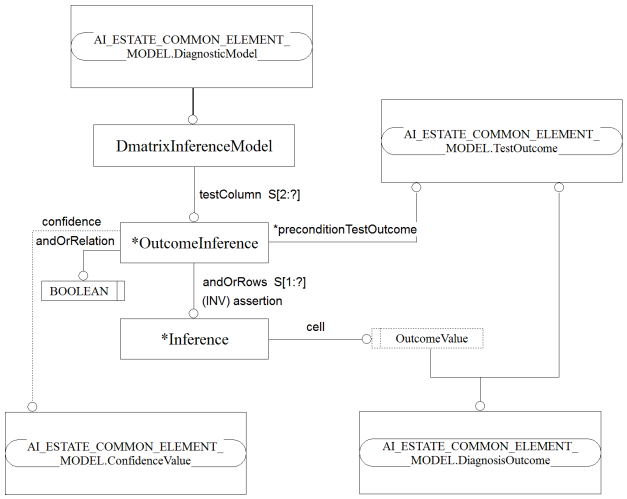

Fig. 1. An EXPRESS diagram for the AI-ESTATE D-Matrix Inference Model (DIM). The lines with circles and labels denote attributes while the lines with circles and no label denote subclass relationships.

```
ENTITY OutcomeInference;
   andOrRows : SET [1:?] OF Inference;
   preconditionTestOutcome : TestOutcome;
   confidence : OPTIONAL ConfidenceValue;
   andOrRelation : BOOLEAN;
UNIQUE
   oneOutcome : preconditionTestOutcome;
WHERE
   conjunctOrDisjunct :
      ((SELF.preconditionTestOutcome.allowedValue =
        Pass) AND
       (SELF.andOrRelation = TRUE)) XOR
      ((SELF.preconditionTestOutcome.allowedValue =
        Fail) AND
       (SELF.andOrRelation = FALSE));
   noUserDefined :
      preconditionTestOutcome.allowedValue <>
        UserDefined;
END_ENTITY;
```

Fig. 2. The EXPRESS code that defines the entity "OutcomeInference." Not only are the attributes for the entity defined, but also any constraints on those attributes, such as the requirement that the attribute "oneOutcome" points to a unique "preconditionTestOutcome."

from executing tests of a Unit Under Test (UUT) via a test program in an automatic test system (ATS) [11]. While generally the Test Results standard is used for exchanging information from an ATS during maintenance, it can also be used to exchange onboard built-in test and monitoring information.

The information models in each of these IEEE standards were defined using EXPRESS, standardized by the International Organization for Standardization (ISO) to support communication of product data between engineering applications. The main purpose of EXPRESS is to define the semantics of information that will be generated by a system [7]. An EXPRESS diagram of the AI-ESTATE D-Matrix Inference Model (DIM) is given in Fig. 1 while the EXPRESS code that defines an "OutcomeInference," as defined by the DIM,

```
<owl:Class rdf:ID="OutcomeInference">
     <rdfs:subClassOf rdf:about="#DMATRIX_MODEL"/>
</owl:Class>
<owl:Class rdf:ID="CEM_ConfidenceValue">
     <rdfs:subClassOf rdf:about="#DMATRIX_MODEL"/>
</owl:Class>
<owl:ObjectProperty rdf:ID="confidence">
     <owl:maxCardinality rdf:datatype=
          "http://.../XMLSchema#nonNegativeInteger">1
     </owl:maxCardinality>
     <owl:minCardinality rdf:datatype=
          "http://.../XMLSchema#nonNegativeInteger">0
     </owl:minCardinality>
     <rdfs:domain rdf:resource="#OutcomeInference"/>
     <rdfs:range rdf:resource="#CEM_ConfidenceValue"/>
</owl:ObjectProperty>
```

Fig. 3. Excerpt from the OWL code defining the class "OutcomeInference" and relationships that exist between "OutcomeInference" and other classes in the ontology, such as the "confidence" relationship with the class "CEM_ConfidenceValue."

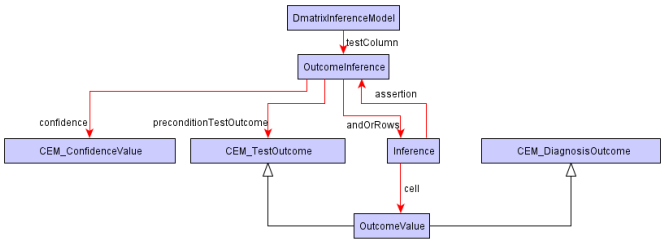

Fig. 4. Subsection of the OWL diagram from the AI-ESTATE DIM model. The large arrows without labels denote parent and child relationships while the smaller arrows denote attributes.

is shown in Fig. 2. The lines with circles and labels denote attributes while the heavy lines with circles and no label denote subclass relationships. For example, in the diagram, the entity "DmatrixInferenceModel" has the attribute "testColumn" which points to the entity "OutcomeInference," which has several relationships with other entities in the standard.

While EXPRESS was not designed to support ontology-based analysis, the semantics defined by EXPRESS models are very rich. Therefore, we used the EXPRESS models as the foundation for defining our ontologies. Recent work in ontology-guided data mining has made use of standard ontology languages such as OWL [8], DAML+OIL [14], and RDF [15]. We decided to use OWL due to its prevalence in the semantic web. OWL is a language for defining and instantiating ontologies based on description logic that also supports a number of automated inference procedures [8]. An OWL ontology may have descriptions of classes, properties, and their instances. The formal OWL semantics then specify how to find logical consequences from the defined entities.

To convert EXPRESS to OWL, we first defined a mapping of EXPRESS concepts to OWL concepts. Once this mapping was completed, we specified the relationships for the OWL ontologies based on the attributes and subtype relationships in the EXPRESS models. After the OWL ontologies had been defined, we mapped the raw data from our maintenance history and diagnostic session data sources to the concepts defined in the corresponding ontologies. In some cases, certain parts of our raw data did not match existing areas of the ontologies; therefore, additional entities and relationships had to be defined. The OWL code and OWL diagram of the same "OutcomeInference" concept from the DIM is shown in Figs. 3 and 4, respectively. In Fig. 4, the large arrows without labels denote parent-child relationships while the smaller arrows denote attributes.

III. Event Graphs

While ontologies can provide an effective way to display data, there is a problem of knowing which ontology instance graphs have interesting information. Displaying all of the ontologies would not be feasible for even a moderate number of graphs, and going through each graph would be far too time-consuming. To overcome this problem, we developed what are called Event Graphs. An Event Graph is a chronological ordering of MAI and on-platform events. For this paper, we refer to an MAI event as a group of records pertaining to a unique set of ground-based maintenance actions while an on-platform event corresponds to a group of records pertaining to a particular set of tests or monitors performed during normal system operation, such as a set of onboard information for an aircraft during a flight [9].

Each MAI event may be composed of one or more transactional database records grouped together based on the MAI model. A 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 element contains a list of MAIs, and is directly connected to the root (MAID) element. For an on-platform event, there are two main concepts of the ontology that are used—Test Result and Test. Specifically, a Test element is the root of the on-platform ontology. All of the test parameters and results that were recorded during system operation connect to the Test element via the Test Result concept.

In an Event Graph, each node corresponds to either an MAI or an on-platform event, and the nodes are ordered chronologically along the horizontal axis. In addition, each node is positioned vertically depending on the event type. The MAI nodes can be ordered by two user-specified date attributes: JCN (Job Control Number) Date, or MA (maintenance action) Completion Date. To improve the readability of the graph, the sequence contains six distinct layers. The topmost layer contains the date when each event occurred. The next layer contains any on-platform event nodes, while the third layer contains the MAI event nodes. Under the MAI event layer are three additional layers, corresponding to the various Maintenance Level (ML) attribute values for each event—Organizational, Intermediate, and Depot levels, respectively (or whatever levels would be implemented for the system being tracked).

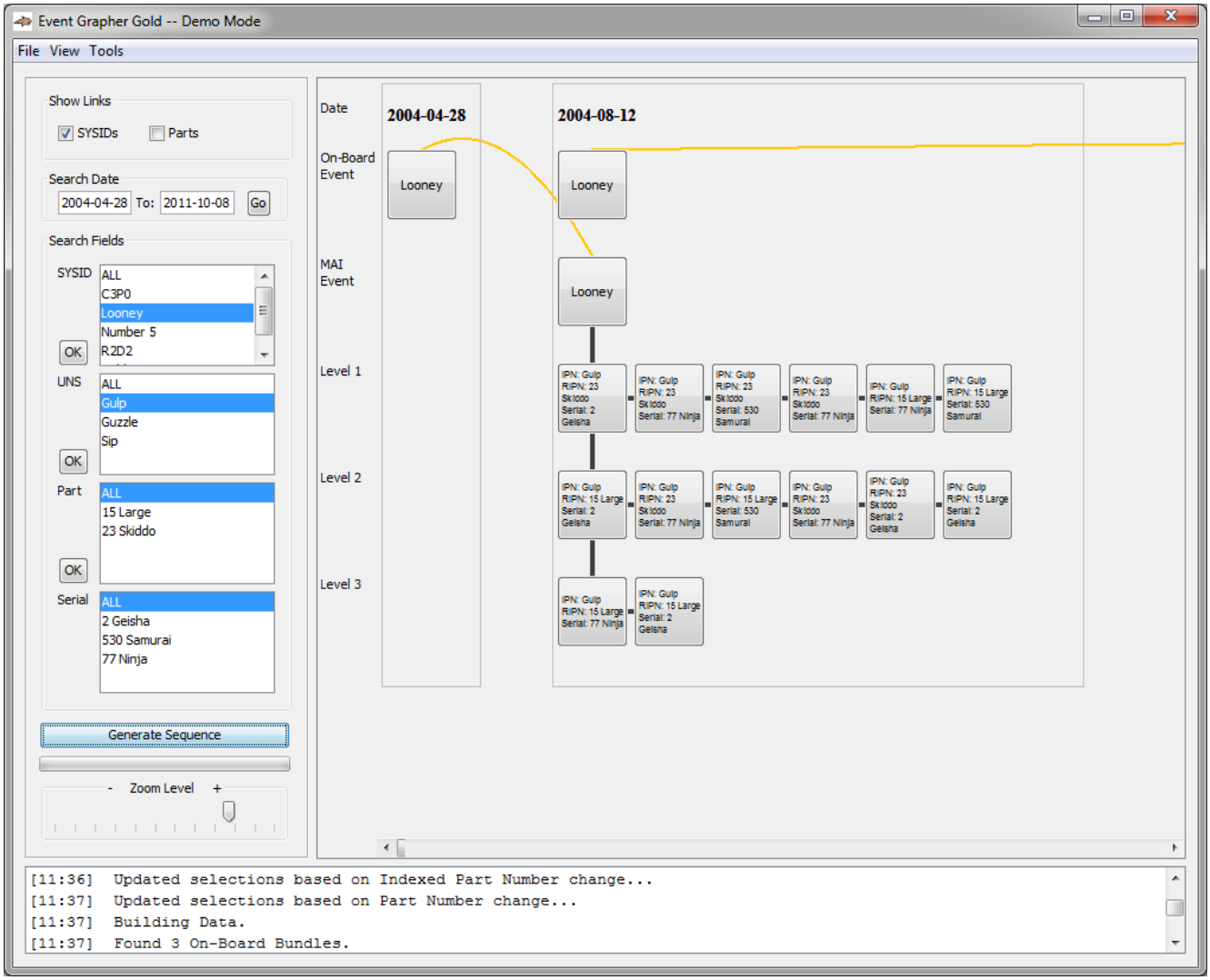

Fig. 5. Event Grapher Gold (EGG) main window. The left panel contains the query options, while the right panel displays all of the Event Graphs in chronological order. The bottom panel displays any messages and status updates.

Finally, we create two sets of links. The first are horizontal links between events that have the same system or item serial number to help the user follow items of interest throughout time. The second set of links connect on-platform and MAI events vertically if they fall under the same date, along with the main MAI event with that of the various nodes representing maintenance actions at different levels. This is a critical capability supporting *post hoc* analysis of maintenance data, as it quickly and easily allows a user to trace the context of specific parts or aircraft through time.

An example of an Event Graph is given in Fig. 5. In this case, two on-platform events are displayed with the system identifier (SYSID) of "Looney," along with a single MAI event that occurred on 2004-08-12. Under that event there are several maintenance actions that occurred at the three maintenance levels.

IV. Event Grapher Gold

Event Grapher Gold (EGG) is a Java application that connects to a MySQL database containing all of the data mapped to the various ontologies. When the program starts, users are presented with the main application window (Fig. 5). There are three main sections to the application window. The left side of EGG shows the search options that the user can use to filter the data displayed. The right side of the application holds the display window, which presents the user with the set of events. Finally, the bottom portion of the window gives the status updates of the tool, such as whether a query was successful or if there were any errors.

To use the program, the user selects the desired search options in the search panel. A database query is then constructed based on the selected options. Currently, EGG allows users to query on dates, system IDs (SYSID), unified number systems (UNS), part numbers (Part), and serial numbers (Serial). Once users have selected their desired parameters, the tool will query

the database and return all of the entries that match the search criteria. This information is then transformed into an ontology-based graph that is displayed in the right panel. To allow users to track events easily throughout time, EGG allows the option of displaying lines between each maintenance event for a particular SYSID and part number.

To display the underlying MAI Event Graphs, the user clicks on an MAI event node, which then generates a new window with the graph displaying all of the information for that particular maintenance event. A window with an MAI graph is shown in Fig. 6. In this example, there is one maintenance action that involved a repair event. From this window, EGG can save the graph as an image, allowing the user to save interesting information they may have discovered.

The database back-end of EGG has received an extensive upgrade. In the previous version, the software required the database to store all of the information as JGraphs, which were held as blobs in the MySQL database. This required a separate data transformation from transactional records representing pieces of maintenance events to entire Event Graphs that aggregate all of those pieces into a common OWL ontology instance. This transformation needed to be performed before the software was run. While this made retrieval of the relevant information straightforward, there were several problems with this approach, such as extending the database to include new data and fields. The current database now only stores the relevant data, and the graphs are generated on demand. While theoretically this does require slightly more overhead when displaying the graphs, the reduced performance is not very noticeable.

One major extension that has been added to EGG is the ability to query and display on-platform data. In previous versions of the visualization tool, only MAI data was displayed. However, the tool now displays information corresponding to Test Results data. For each on-platform event, the tool then attempts to locate the corresponding MAI event. A node is then added to the event sequence graph at its chronological location. The node can then be clicked, which will bring up an ontology-based graph containing all of the data corresponding to that particular on-platform event. The window in Fig. 7 gives an example of an on-platform graph.

Another extension to the tool that is currently being implemented is the ability to export the selected data to an OWL ontology. This new feature converts all of the data that is currently displayed in the Event Graphs into an OWL instance format. The export includes all of the data that is displayed in the MAI and on-platform graph ontologies. Once the data has been exported to the OWL format, it can then be used by ontological guided knowledge discovery algorithms.

V. MODELMAT

Model-based methods have become common when performing system-level diagnosis. The basic idea behind model-based diagnosis is to compare observations from the system with those predicted by a model of the system [12] and derive an explanation for those observations (especially those not expected) based on the properties of the model. One model-based approach to system-level diagnosis is the Timed Failure Propagation Graph (TFPG), developed at Vanderbilt University [16], [17]. The TFPG specifies causal relationships between faults and discrepancies, which are irregular conditions arising as a result of the faults (whether monitored or unmonitored). Consistency-based diagnostic algorithms are then able to diagnose faults by considering those alarms that were triggered, combined with the order in which they occurred [16], [17].

Because creating error-free diagnostic models is difficult, a process is needed to mature the diagnostic models over time. If system engineers or technicians know when a model is misdiagnosing a fault, they should be able to use that information to modify the model, resulting in more accurate diagnoses. To determine whether the reasoner diagnosed the correct fault, one must compare the reasoner's diagnosis with the actual fault found by alternative means. By storing past maintenance history, detailed engineering analysis can often be performed to determine the actual fault that occurred. The maintenance information can then be compared to the reasoner history and searched for any discrepancies between the two data sources. If there is a discrepancy between the two histories, then the users know that the reasoner misdiagnosed a fault [1].

We developed a toolset called ModelMat that implements our maturation algorithms, DMat (which identifies potential errors in the underlying D-matrix) and SeqMat (which identifies potential errors in the expected sequences of alarms), to mature TFPG and other D-matrix models [10]. ModelMat was written in Java and provides the user with an easy-to-use interface for viewing the algorithm output and setting algorithm parameters (Fig. 8). In the left panel, the tool displays information about the diagnostic model being analyzed, including the list of faults, alarms, and the D-matrices representing relationships between faults and alarms. In the right panel, the tool displays results from running the maturation algorithms. To analyze output from the various maturation algorithms, the user clicks on the desired results tab, which instantly displays any recommendations for how to edit the model. When ModelMat locates a possible error in the diagnostic model, the table of results will highlight those potential errors for the user. When there is no conflict found, the rows are colored blue, while potential conflicts that were discovered during the maturation process are highlighted in red.

ModelMat works as follows. First, the user selects a diagnostic model to open. Currently, the user can load D-matrices or TFPGs. If a D-matrix is loaded, the software will first perform logical closure followed by transitive reduction. This will provide an approximation of the type of information the underlying TFPG might have. If a TFPG is loaded, a first-order D-matrix can be extracted directly from the graph. As an option, the software can convert the TFPG to a D-matrix using transitive closure and then perform logical closure and transitive reduction. This allows the user to assess the potential loss of information from working with a D-matrix alone.

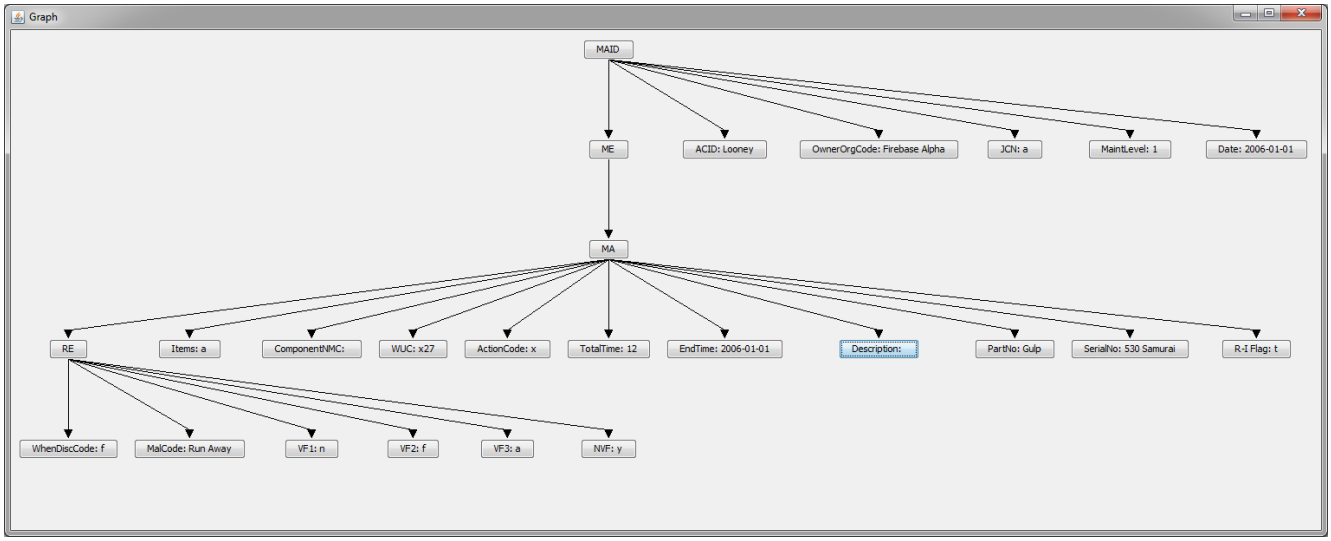

Fig. 6. An EGG MAI window displaying all of the data contained for a particular maintenance event in an ontology-based graph. From this window, the program also allows users to save an image, which can then be referenced later without having to run the query again.

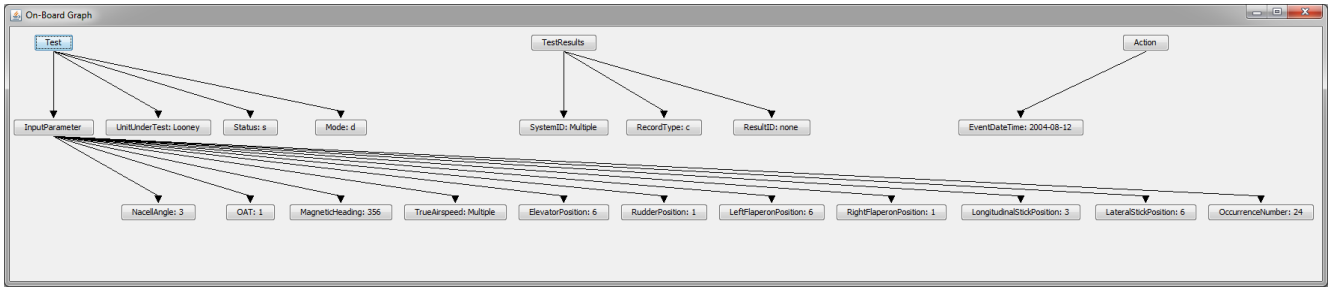

Fig. 7. An EGG window containing an on-platform graph.

The software then gives the user the option to run the alarm sequence maturation algorithms (i.e., either DMat or SeqMat) on the original TFPG or D-matrix.

Next, ModelMat connects to a MySQL database that contains reasoner history data (including what was diagnosed), the ordered list of alarms that fired, and what maintenance determined was the true fault. In addition, system information is also stored, such as the date of the maintenance event, the system ID, and the system type. ModelMat then allows the user to query the database based on ground truth, system type, system ID, and a range of dates. Alarm sequences corresponding to the user's query are loaded into the tool for subsequent analysis. If the user is running the alarm sequence maturation, an option is provided for whether or not to run Warshall's algorithm, which removes the falsely flagged relationships [18], [19]. The software also allows the user to set the various threshold values used by the algorithms.

Note that all of the information stored in the database is in the process of being converted to OWL ontologies corresponding to IEEE Std 1232 (AI-ESTATE) D-matrix models and dynamic context models [5] as well as IEEE Std 1636.2 MAI information [6]. This is a major point of integration for EGG and ModelMat, because now EGG can be used to identify key events that may be of interest to diagnostic maturation. Conversely, potential discrepancies flagged by ModelMat can trigger an analyst to examine event sequences in EGG to see if other problems exist in the maintenance process.

VI. Ontology Mining Algorithms

While EGG allows users to quickly inspect data visually, it does not support the ability for the direct use of knowledge discovery algorithms on the data. The capability to perform knowledge discovery is critical in discovering new relationships in data. A variety of knowledge discovery algorithms can be applied to the data being capture for use by EGG and ModelMat, including algorithms for clustering, feature extraction, association and link analysis, anomaly detection, and classification. Currently, we are in the process of exploring several algorithms, focused specifically on clustering and association analysis, which, when guided by the underlying ontologies associated with the data, provide for a more effective means to discover knew knowledge about the systems being maintained.

A. Apriori Knowledge Discovery

One way to discover new relationships is through mining frequent patterns, which are sets of data items (called itemsets)

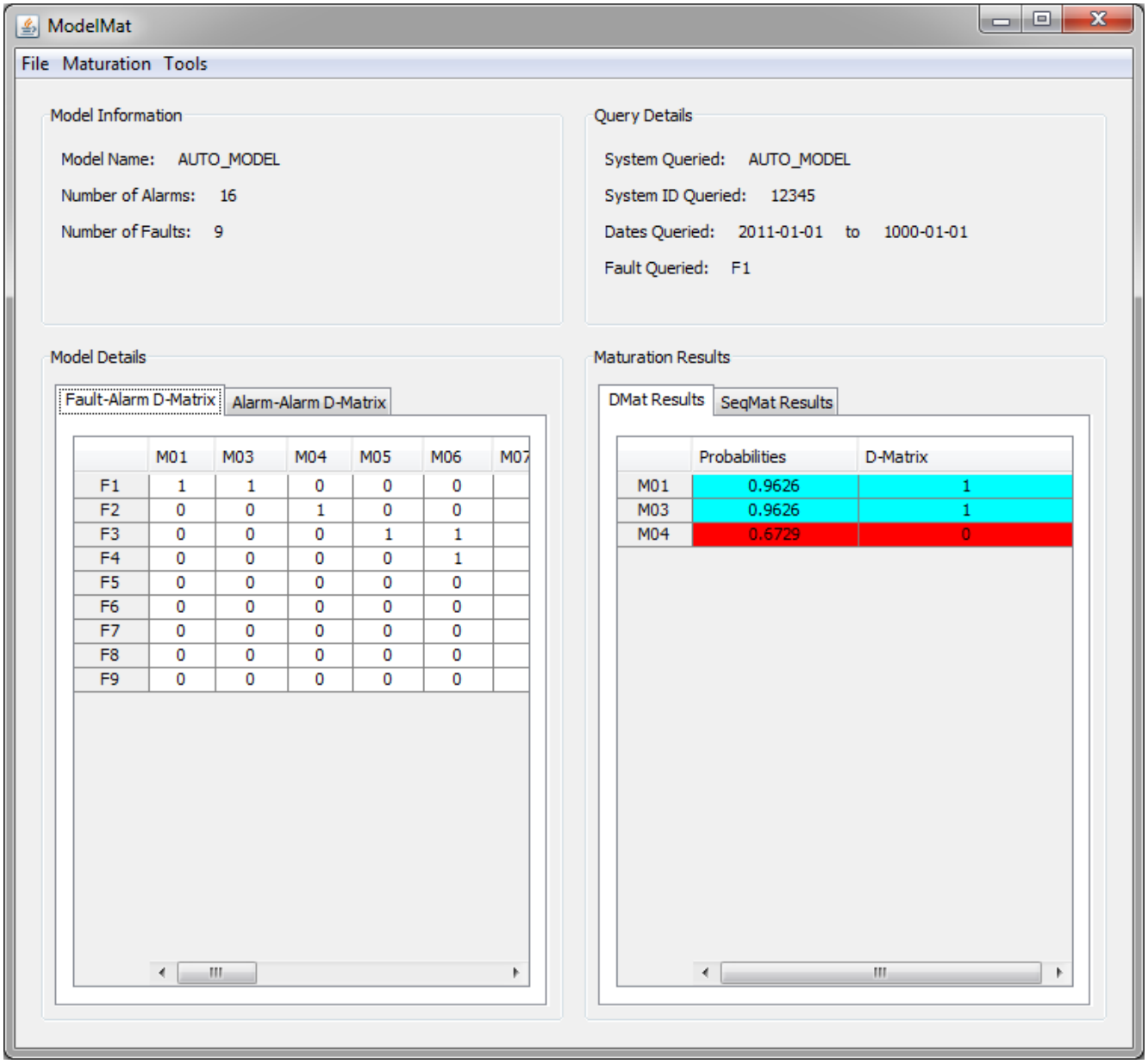

Fig. 8. A screenshot of ModelMat, which is a software program designed to perform maturation on diagnostic models. The left side of the panel shows information of the diagnostic model, while the right panel shows results from the maturation algorithms.

that frequently appear together in a dataset. The Apriori algorithm is one such knowledge discovery algorithm that locates frequent itemsets for creating association rules [3]. An association rule is a rule written as an implication of the form $X \Rightarrow Y$ where X is the conjunction of a set of discrete variables and Y is the conjunction of a different set of discrete variables. Given an association rule $r =_{\text{df}} (X \Rightarrow Y)$ and a dataset D, we say that the support $spt(r)$ of rule r is the percentage of examples in D for which the conjunction $X \wedge Y$ holds:

$$spt(r) = \frac{|X \wedge Y|}{|D|},$$

and the confidence $cf(r)$ of rule r is the percentage of examples in D for which X holds where $X \wedge Y$ also holds:

$$cf(r) = \frac{|X \wedge Y|}{|X|}.$$

The Apriori algorithm finds so-called "large itemsets," which are sets of variables for which the minimum support has been obtained. A large itemset with k variables is referred to as a k-itemset. The algorithm is based upon the Apriori property, which states that all nonempty subsets of a frequent itemset must also be frequent. This property allows us to develop a strategy for building up frequent itemsets from scratch. Using all frequent $(k-1)$-itemsets, we can combine these $(k-1)$-itemsets to build possible k-itemsets.

The pseudocode for the Apriori algorithm can be found in Fig. 9. In this algorithm, let L_k be the set of large k-itemsets, C_k be the set of candidate k-itemsets, d be some subset of "literals" or "items" in the data (called "transactions" in [3]), and s be the user-specified minimal percent of items containing some candidate itemset c. This algorithm then finds the itemsets of various sizes meeting the minimum support threshold. Once the itemsets are found, it is a simple matter to construct the rules where the left-hand sides are itemsets that are subsets of itemsets covering the entire rule.

```
1:  function Apriori (D, minSpt)
2:  L_1 ← set of large 1-itemsets
3:  k ← 2
4:  while L_{k-1} not empty do
5:     C_k ← (L_{k-1}) // Generate itemset candidates
6:     for all d ∈ D do
7:        C_d ← subset(C_k, d) // Candidate contained in data
8:        for all c ∈ C_t do
9:           count[c] ← count[c] + 1
10:       end for
11:    end for
12:    L_k ← {c ∈ C_k | count[c] ≥ minSpt} // Test for support
13:    k ← k + 1
14: end while
15: return ∪_k L_k
```

Fig. 9. The pseudocode for the Apriori algorithm.

The algorithm works as follows. The user passes in a database of transactions D and a value for s, which is the minimum support threshold. In the following steps, the algorithm uses the previous L_{k-1} set of frequent itemsets to build the set of frequent k-itemsets. Line 5 of the algorithm uses the Apriori property to remove itemsets that are not frequent. Next, the database is scanned and compared to the set of possible frequent itemsets (lines 6 and 7). If a single transaction c contains as a subset one of the candidate frequent itemsets, the count for that candidate frequent itemset is increased (line 8 and 9). Finally, in line 12, the count of each possible candidate is checked again the minimum support threshold. If the support is high enough, then the candidate is kept as a frequent k-itemset. The process ends when there are no more frequent itemsets to be constructed (line 14), at which point all of the frequent k-itemsets constructed to that point are returned.

The Apriori algorithm was originally designed to run on raw transactional datasets. However, because EGG is able to export the data in an ontological graph format, we also have additional knowledge that is not present in transactional records. This lets the Apriori algorithm perform more efficiently. One proposed way to use the Apriori algorithm on ontological datasets is to restrict the search of frequent itemsets to that of nodes in the ontology selected by the user [2]. For example, suppose in the MAI data, the user wanted to know the most common occurring repaired items for a set of aircraft. By using ontologies, the user could restrict the analysis to a set of diagnoses that are at the same level of indenture, part of the same repair context, and at the same depth within the lattice structure by constructing ontological queries against the mediated information, thus restricting the information to be limited to the subset of interest. Alternatively, concepts in the ontology could be weighted based upon significance for some issue (e.g., safety, availability, identification of bad actors). The data items could be restricted to those concepts or key related concepts.

B. Graph Mining

Recently, graph mining has become another popular approach to knowledge discovery, especially because a wide variety of complicated structures can be represented as graphs. One of the fundamental graph mining problems is the discovery of frequent subgraphs. Frequent subgraph mining is aimed at locating subgraphs within a set of larger graph structures which appear frequently together. Many of the frequent subgraph mining algorithms utilize the Apriori property to aid in computational complexity [20]. Because ontologies are graphs, one could use a frequent subgraph mining algorithm to locate frequently occurring subgraphs in instance graphs of data mapped to these ontologies. However, because the structure of the ontologies will all be very similar, the algorithm will also have to consider the values in each of the nodes of the ontology. This suggests a hybrid algorithm whereby Apriori would be adapted to consider these values. Alternatively, clustering algorithms, such as k-means, k-medoids, or agglomerative hierarchical clustering could be applied to relate the graph structures together, based upon these feature values.

Another recent graph-based mining approach that has been showing remarkable utility is spectral clustering. Spectral clustering works by finding the top eigenvectors of a matrix representing distances between points. These eigenvectors are then used to derive a scheme for clustering the points [21]. More formally, suppose we have a dataset D that has been mapped to some ontology $\mathcal{O}$. Suppose further that we use a mechanism such as concept weighting (described above) to filter and extract relevant instance data from D. For example, we might only be interested in test measurements that have been taken on inertial navigation systems that have flown on aircraft in high-G situations. We could use information from the AI-ESTATE DCM to find onboard diagnostic sessions for these INS based on the Step.stepContext attribute in the associated sessions. This information points us to the relevant repair items, which then point us to relevant Test Results data. The Test Results instances contain the desired measurements.

At this point, traditional spectral clustering treats the associated data as flat feature vectors and constructs an associated similarity matrix,

$$M = \begin{bmatrix} \delta(x_1, x_1) & \delta(x_1, x_2) & \cdots & \delta(x_1, x_n) \\ \delta(x_2, x_1) & \delta(x_2, x_2) & \cdots & \delta(x_2, x_n) \\ \vdots & \vdots & \ddots & \vdots \\ \delta(x_n, x_1) & \delta(x_n, x_2) & \cdots & \delta(x_n, x_n) \end{bmatrix}$$

where x_i is the i^{th} feature vector and $\delta(x_i, x_j)$ is the similarity between feature vector x_i and feature vector x_j. This corresponds to a weighted adjacency matrix for the graph. Usually, a threshold ϵ is defined such that, should the similarity be less than this threshold, the corresponding entry in the matrix is set to zero. This leads to what is called an ϵ-neighborhood graph. Another approach is to restrict connections to the k nearest neighbors of each vector. These approaches lead to *reduced* weighted adjacency matrices.

After computing the reduced matrix M_{reduced}, the next step is to find the graph Laplacian. The Laplacian can be normalized or unnormalized. For this, we also need to define the degree matrix for the reduced matrix. This matrix, which we denote Δ, is a diagonal matrix where the entries along the diagonal equal the degrees of the vertices $deg(x_i)$ corresponding to each of the feature vectors x_i. specifically, $deg(x_i)$ is the sum of the non-zero entries in the i^{th} row of M_{reduced}.

$$deg(x_i) = \sum_j \delta(x_i, x_j)$$

The unnormalized graph Lagrangian then corresponds to

$$L = \Delta - M_{\text{reduced}}.$$

Given L, the most common spectral clustering algorithm for unnormalized Lagrangians works as follows:

- Construct the reduced similarity graph M_{reduced}.
- Construct the graph Lagrangian $L = \Delta - M_{\text{reduced}}$.
- Find the first k eigenvectors $u_1, \ldots, u_k$ (ranked by eigenvalue in decreasing order) of L.
- Construct a new matrix U where each of the u_i eigenvectors are columns.
- Cluster the rows of U using k-means to find clusters $C_1, \ldots, C_k$.
- Return the row indices of U, grouped according to their k clusters.

To interpret the results, note that, because we are finding eigenvectors of the graph Laplacian, we are constructing a linear transformation into a space where each eigenvector is a linear combination of distances focused on identifying those points that are furthest apart. Then the k-means algorithm finds the components in these new vectors that are most similar, in effect finding ways to "cut" (i.e., partition) the underlying graph. The resulting graph partitions form connected components in the original graph that correspond to clusters of nearby feature vectors.

These clustering methods can be very useful for detecting various fault conditions arising that might not have been expected during system design. For example, as described with the INS flown under high-G conditions, we might be able to use these techniques to detect how different environmental or operational conditions might affect degradation and failure of the system. It is in these types of analyses that coupling state-of-the art knowledge discovery methods with information tied to domain ontologies can yield tremendous benefit.

VII. Integrated Toolset

EGG and ModelMat were both originally designed to be stand-alone applications. For example, one of the primary applications for EGG is for allowing a user to quickly discover interesting events throughout time. Specifically, should a part be removed from a system, we can track the part forward through time to see if it is ever reinstalled on another system. If the same part is installed on a system with a different ID, then we can consider the dates of removal and install, from which we might be able to ascertain that the part was cannibalized from the first to be used in the second. If users wish to investigate this event further, then they are able to click on the MAI events of interest to bring up the specific maintenance events to see if there are any fields that could give users a better understanding of the underlying cause of that event.

Similarly, ModelMat was designed to discover discrepancies between maintenance data and diagnostic reasoner data. However, as mentioned in our previous work on ModelMat, locating the desired data can be difficult. By integrating EGG and ModelMat together, users are able to overcome this difficulty by using EGG to find relevant events for use in ModelMat. Subsequently, a maturation analysis can be performed to determine if any associated discrepancies might be significant.

Conversely, ModelMat may inform users of certain events in the data that they may wish to investigate in EGG. For example, suppose an analyst performs a maturation analysis with ModelMat using all of the historical maintenance data and discovers discrepancies between diagnostic reasoning and maintenance. In doing so, they may wish to investigate what occurred in those maintenance events, which they can do by visually inspecting those events in EGG.

A top-level diagram of the EGG and ModelMat as an integrated toolset and their interactions with the data and one another can be seen in Fig. 10. We call the resulting toolset *Bobcat*. In this diagram, we see that Bobcat draws its information from a semantic repository, consisting of historical instance data that has been mapped to four different domain ontologies through a set of simpler data converters:

- SIMICA Test Results (IEEE Std 1636.1)
- SIMICA MAI (IEEE Std 1636.2)
- AI-ESTATE D-Matrix (IEEE Std 1232)
- AI-ESTATE DCM (IEEE Std 1232)

Both EGG and ModelMat extract the relevant data from the semantic repository to support their respective analyses. Furthermore, as indicated by the arrow from EGG to ModelMat, knowledge discovery algorithms and processes can now be applied to the data, as informed and filtered by these two tools.

VIII. Conclusions

In this paper, we have presented an integrated toolset that consists of two software programs we developed for performing ontology-guided knowledge discovery. The first is a visualization tool called EGG that allows users to inspect

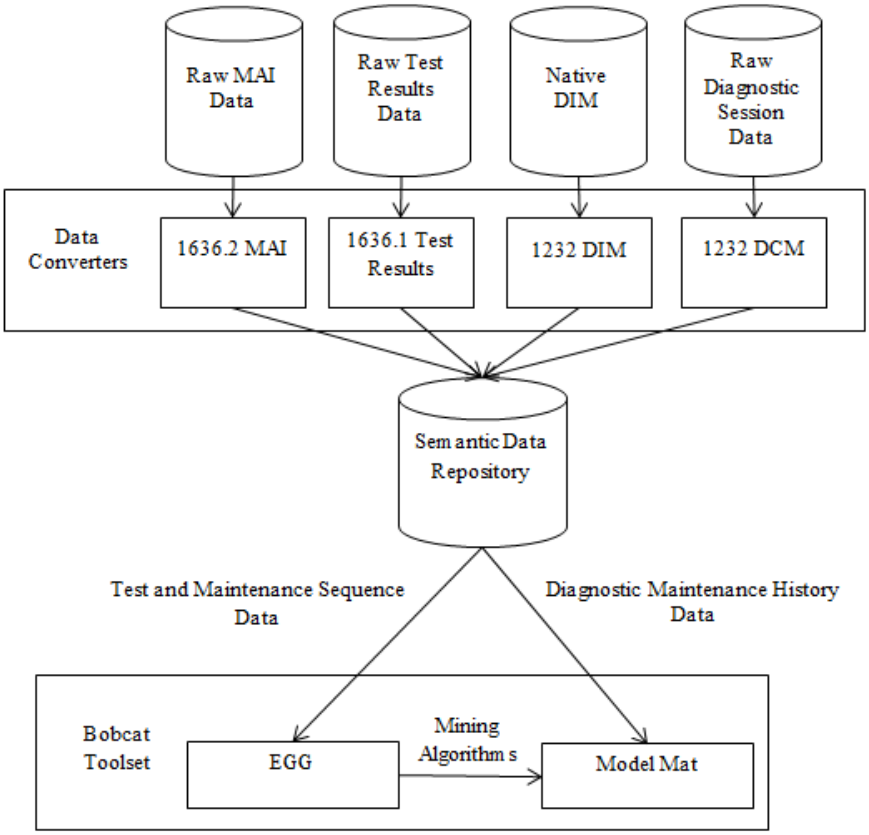

Fig. 10. A top-level view of the toolset presented in this paper. Raw data, such as test results, maintenance histories, and onboard reasoner data, is transformed and sorted into an ontological format. This data is then accessed by EGG and ModelMat for knowledge discovery, visualization, and diagnostic model maturation.

data visually very quickly and efficiently. The second is a diagnostic model maturation tool called ModelMat that engineers use to identify discrepancies between diagnostic sessions and associated maintenance events and then recommend ways to fix those discrepancies. While these two tools were originally designed to be stand-alone, we have begun to integrate them together by exploiting the semantics of the information they process. This is done by mapping the underlying data to several domain ontologies.

For future work, we are continuing to integrate the capabilities of the tools in the toolset. This next step in integration involves redesigning each of their respective databases to use common schemas based on the underlying OWL ontologies. We anticipate additional ontologies (and associated data) being incorporated into the toolset based, for example, on the schemas that have been defined for the Automatic Test Markup Language [22].

Another area of future work is to allow ModelMat and EGG to interact more directly. Currently, EGG can locate desired maintenance event data, but users must manually export the data for analysis by ModelMat by using another external tool. One feature that is being added allows EGG to export the data directly to ModelMat.

Finally, we are in the process of directly integrating the above-mentioned knowledge discovery algorithms into either EGG or the toolset as a whole. One related are of integration that we are considering is incorporating one or more diagnostic reasoners so that we can also use associated inference procedures to evaluate the results of any recommended modifications or to test hypotheses relative to discrepancies that have been detected. We can also use these reasoners to evaluate the effects of modeling discovered relationships and properties, arising from the various knowledge discovery tools.

ACKNOWLEDGMENTS

The authors would like to thank Michael Schuh for his help and guidance in developing the EGG software. Mike was the principal architect and developer of the prototype version of EGG and provided extensive insight into use cases and implementation issues he encountered along the way. We would also like to thank Liessman Sturlaugson for his detailed review of an earlier version of this paper as well as all of the members of the Numerical Intelligent Systems Laboratory and Montana State for several stimulating discussions and ideas along the way. Finally, we thank Tim Wilmering at Boeing for his insight and encouragement as these tools have been developed.

REFERENCES

[1] T. Wilmering, "When good diagnostics go bad—why maturation is still hard," in *Proceedings of the IEEE Aerospace Conference*, vol. 7, 2003, pp. 3137–3147.

[2] 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.

[3] R. Agrawal, R. Srikant *et al.*, "Fast algorithms for mining association rules," in *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, vol. 1215, 1994, pp. 487–499.

[4] S. Strasser, J. Sheppard, M. Schuh, R. Angryk, and C. Izurieta, "Graph-based ontology-guided data mining for D-matrix model maturation," in *Proceedings of the IEEE Aerospace Conference*, 2011, pp. 1–12.

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

[6] IEEE Std 1636.2-2010, *IEEE Trial-Use Standard for Software Interface for Maintenance Information Collection and Analysis (SIMICA): Exchanging Maintenance Action Information via the Extensible Markup Language (XML)*. Piscataway, NJ: IEEE Standards Association Press, 2010.

[7] ISO 10303-11:1994, "Industrial automation systems and integration – Product data representation and exchange – Part 11: Description methods: The EXPRESS language reference manual," 1994.

[8] W3C, "OWL 2 Web Ontology Language Document Overview," http://www.w3.org/TR/owl2-overview/, 2009.

[9] M. Schuh, J. Sheppard, S. Strasser, R. Angryk, and C. Izurieta, "Ontology-guided knowledge discovery of event sequences in maintenance data," in *IEEE AUTOTESTCON Conference Record*, 2011, pp. 279–285.

[10] S. Strasser and J. Sheppard, "Diagnostic alarm sequence maturation in timed failure propagation graphs," in *IEEE AUTOTESTCON Conference Record*, 2011, pp. 158–165.

[11] IEEE P1636.1-2007, *IEEE Trial Use Standard for Software Interface for Maintenance Information Collection and Analysis (SIMICA): Exchanging Test Results and Session Information via the eXtensible Markup Language(XML)*. Piscataway, NJ: IEEE Standards Association Press, 2007.

[12] S. Abdelwahed, G. Karsai, N. Mahadevan, and S. Ofsthun, "Practical implementation of diagnosis systems using timed failure propagationgraph models," *IEEE Transactions on Instrumentation and Measurement*, vol. 58, no. 2, pp. 240–247, February 2009.

[13] M. Uschold and M. Gruninger, "Ontologies: Principles, methods and applications," *Knowledge Engineering Review*, vol. 11, no. 2, pp. 93–136, 1996.

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

[15] W3C, "Resource Description Framework (RDF)," http://www.w3.org/RDF/, 2004.

[16] A. Misra, "Sensor-based diagnosis of dynamical systems," Ph.D. dissertation, Vanderbilt University, Nashville, TN, 1994, electrical Engineering.

[17] A. Misra, J. Sztipanovitz, and J. R. Carnes, "Robust diagnostic system: structural redundancy approach," in *Knowledge Based Artificial Intelligence Systems in Aerospace and Industry, SPIE's Symposium on Intelligent Systems*, vol. 2244, no. 1, April 1994, pp. 249–260.

[18] S. Warshall, "A theorem on Boolean matrices," *Journal of the ACM*, vol. 9, pp. 11–12, January 1962.

[19] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. Cambridge, MA: The MIT Press, 2009.

[20] J. Han, *Data Mining: Concepts and Techniques*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005.

[21] U. Von Luxburg, "A tutorial on spectral clustering," *Statistics and Computing*, vol. 17, no. 4, pp. 395–416, 2007.

[22] IEEE Std 1671-2010, *IEEE Standard for Automatic Test Markup Language (ATML) for Exchanging Automatic Test Equipment and Test Information via XML*. Piscataway, NJ: IEEE Standards Association Press, 2010.