ERDC/CERL TR-04-14

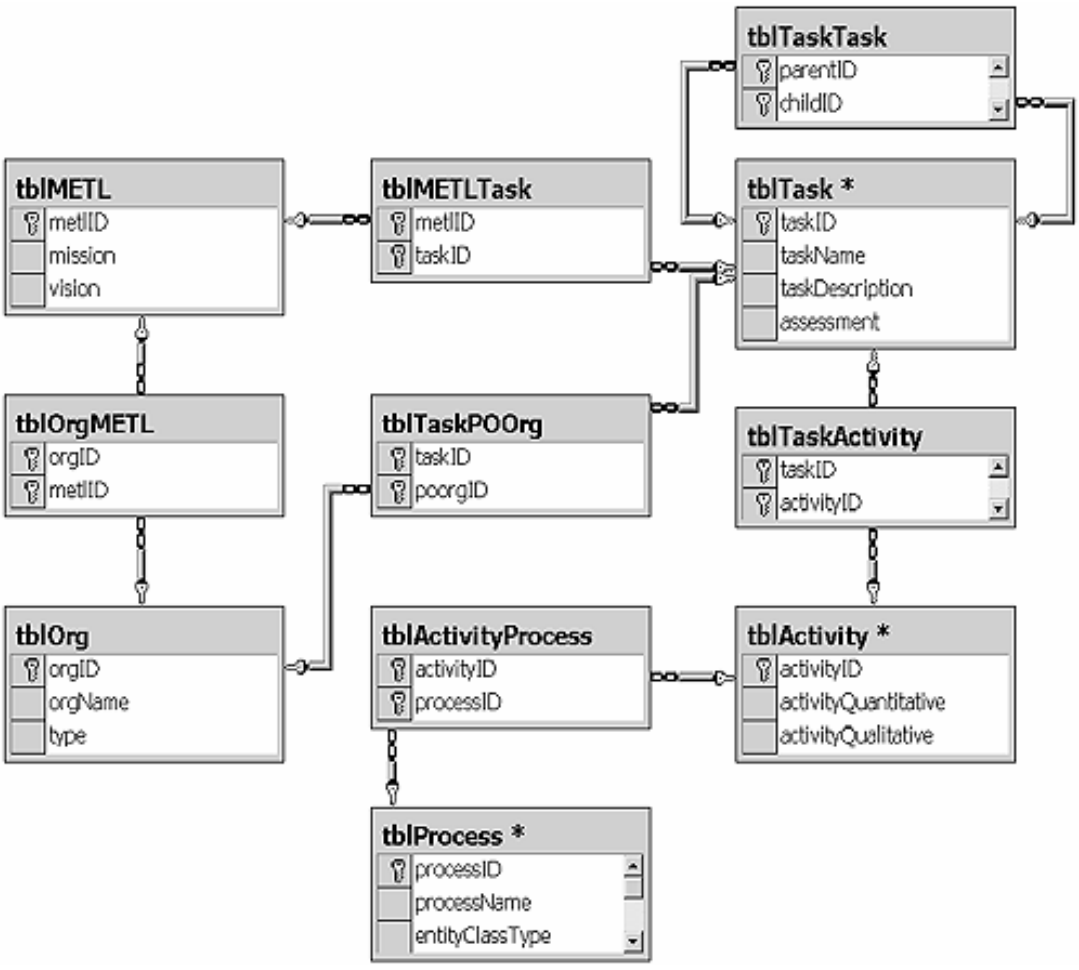
Construction Engineering Research Laboratory

US Army Corps of Engineers®
Engineer Research and Development Center

Knowledge Discovery in the I-METL Application

Todd R. Littell

September 2004



ERDC/CERL TR-04-14
September 2004

Knowledge Discovery in the I-METL Application

Todd R. Littell

Construction Engineering Research Laboratory
PO Box 9005
Champaign, IL 61826-9005

Final Report

Prepared for U.S. Army Corps of Engineers
Washington, DC 20314-1000

Under Work Units LK6K75 and 00S64L

ABSTRACT: The Installation Mission Essential Task List (I-METL) is a software system designed to support the modeling and analysis of garrison capabilities, tenant functions, and installation resources. From a system analyst's point of view, the main focus of the I-METL application is as a means for collecting, sharing, and managing structured data. As the stored data accumulates in size, quality, and richness, stakeholders begin to realize the potential for harvesting new business intelligence from the data store. To this end, a myriad of tools and methods (commonly referred to as Knowledge Discovery from Database [KDD] methods) are available depending on the kind of intelligence pursued.

This research investigated KDD methods that can directly benefit I-METL stakeholders. One goal of this effort was to provide a means for stakeholders to gain an increased understanding of the existing data and data relationships. Another goal was to foster the discovery of new and hidden relationships from the dataset. Methods that will assist with data exploration and cognition were also researched.

The two disparate methodologies investigated produced positive results for the KDD process and offer many potential benefits in a real-world deployment scenario. The prototype software developed validated these results and provided insights into future research possibilities.

Contents

List of Figures and Tables

Figures

Tables

Preface

This study was conducted for Department of the Army under 622784AT41, "Military Facilities Engineering Technology"; Work Unit LK6K75, "Fort Future Facilities"; and Work Unit 00S64L, "Mission Focused Infrastructure Investment."

The work was performed by the Business Processes Branch (CN-B) of the Installations Division (CN), Construction Engineering Research Laboratory (CERL). The CERL Principal Investigator was Todd R. Littell. The technical editor was Linda L. Wheatley, Information Technology Laboratory — Champaign. The technical monitor was William D. Goran, CERL Director of Special Projects. Kay Maguire is Acting Chief, CN-B, and Dr. John T. Bandy is Chief, CN. The Director of CERL is Dr. Alan W. Moore.

CERL is an element of the U.S. Army Engineer Research and Development Center (ERDC), U.S. Army Corps of Engineers. The Commander and Executive Director of ERDC is COL James R. Rowan, EN, and the Director of ERDC is Dr. James R. Houston.

1 Introduction

Background

The Installation Mission Essential Task List (I-METL) is a software system designed to support the modeling and analysis of garrison capabilities, tenant functions, and installation resources. The system defines and implements models for organizations, missions, processes, activities, and resources. The system builds on these data models to assist with business processes such as mission validation and resource planning.

The I-METL system was designed and implemented as a web-deployed, database application. Its 3-tier software architecture is prototypical, where application (business) logic resides in the middle tier. The application uses a Relational Database Management System (RDBMS) for reliable storage and management of application data. Any web browser can serve as the application client since the interface was designed with standard HyperText Markup Language (HTML).

From a system analyst's point-of-view, the main focus of the I-METL application is as a means for collecting, sharing, and managing structured data. This kind of application is commonly referred to as a CRUD application since the primary operations are to "create," "read," "update," and "delete" records. As is common with CRUD applications, the requirement for supporting higher-level analysis often evolves over time as a secondary goal. As the stored data accumulates in size, quality, and richness, the stakeholders typically begin to realize the potential for harvesting new business intelligence from the data store. To this end a myriad of tools and methods are available depending on the kind of intelligence one is pursuing. These methods are commonly referred to as Knowledge Discovery from Database (KDD) methods.

Objective

The objective of this research was to investigate KDD methods that can provide direct benefits to I-METL stakeholders and application. The I-METL data model is complex, and the I-METL dataset is semantically rich. As a result, one goal of this effort was to provide a means for stakeholders to gain an increased understanding

of the existing data and data relationships. Another goal was to foster the discovery of new and hidden relationships from the dataset. Methods that will assist with data exploration and cognition were also to be researched.

Approach

This research effort focused on two concrete KDD methodologies. The first method studied is the process of learning new knowledge from the application of automated reasoning tools. The research of automated reasoning is presented in Chapter 2. The second method studied applies visualization techniques in order to promote the human element of the knowledge discovery process. The topic of KDD through visualization is discussed in Chapter 3. Both efforts are validated through the development of software prototypes. Since these methods complement each other, an integrated approach is discussed in Chapter 4. Chapter 5 summarizes the lessons learned from this research effort. Appendix A includes the source code for the test software discussed in Chapter 2. Appendix B details the criteria used for the software selection as discussed in Chapter 3. Appendix C contains a catalog of screenshot images produced under the visualization research methodology.

Scope

The scope of this research is primarily for the benefit of I-METL stakeholders and application. However, the approaches discussed and lessons learned might potentially benefit any database-driven application. The supporting software prototypes were designed to the largest extent possible as application-independent software.

Mode of Technology Transfer

This report captures the results and lessons learned from this research effort. It will be made accessible through the World Wide Web (WWW) at URL: http://www.cecer.army.mil.

2 Knowledge Discovery from Automated Reasoning

The I-METL application uses a relational database for the reliable storage of I-METL data and relationships. As is common in relational databases, the I-METL database does not support complex query and analysis operations. A complex query operation can exist in multiple forms. It can be in the form of a transitive-closure (recursive) query that needs to re-execute an unknown number of times in order to form a result set. Alternatively, a complex query can be in the form of a reachability query; such as, find all records (objects) that connect record A (in table A) to record Z (in Table Z). Furthermore, a complex query could potentially perform logical deduction; such as, determining if a task scheduling plan exists that can complete a set of tasks within a given time constraint. Relational databases do not provide a means for executing these complex and intelligent queries. In short, relational databases do not provide a mechanism for *reasoning* with the data that it reliably stores.

Defining the Relational Data Model

The limited support for executing complex and intelligent queries emanates from the simplicity of the relational data model. The relational data model is based upon the constructs of *entities*, *attributes*, and *relationships*. Figure 1 illustrates a subset of the relational data model for the I-METL application using an Entity-Relationship (ER) diagram. In a relational data model, entities represent things, people, places, events, and concepts. Attributes are properties or characteristics of entities, such as the name of an organization or the date of an event. Relationships represent the logical or physical connections that exist between entities in the domain of interest. The atomic construct, that allows databases to uniformly manipulate both entities and relationships, is called in mathematics a *relation*. Technically speaking, an *n-ary relation* R, defined over domains $D_1, D_2 \Lambda\ D_n$, is any set $R \subseteq D_1 \times \Lambda\ \times D_n$.

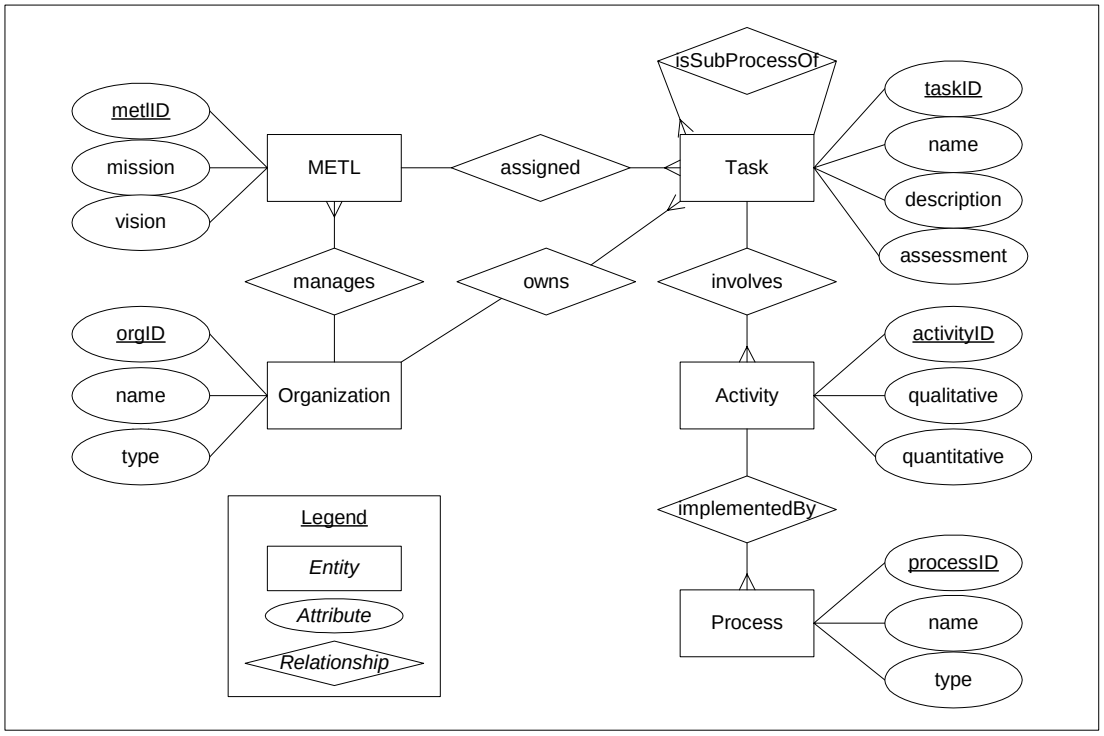


Figure 1. Sample ER diagram.

Limitations of the Relational Data Model

While the relational data model is quite powerful, it cannot model important relationship constraints and properties. Neither the ER diagram nor the underlying relational model is capable of denoting or enforcing certain kinds of constraints. It is common to encounter structural constraints like "tree," "lattice," and "k-partite" when modeling a domain. Similarly, one often needs to be able to capture and enforce relationship properties, such as reflexive, irreflexive, symmetric, asymmetric, or transitive (Odell 1998). An example would be the relationship "X *supports* Y." Equations 1–3 present three first-order logic statements that define the relationship "supports."

$$(\forall X, \forall Y \ni Task(X) \wedge Task(Y) \wedge SubprocessOf(X, Y)) \Rightarrow Supports(X, Y) \quad \textbf{[Eq 1]}$$

Translation: If X and Y are tasks and X is a subprocess of Y, then declare that X supports Y.

$$(\forall X, \forall Y \ni Activity(X) \wedge Task(Y) \wedge Executes(Y, X)) \Rightarrow Supports(X, Y) \quad \textbf{[Eq 2]}$$

Translation: If X is an activity and Y is a task and Y executes X, then declare that X *supports* Y.

$$(\forall X, \forall Y, \forall Z \ni Supports(X,Z) \wedge Supports(Z,Y)) \Rightarrow Supports(X,Y) \quad \textbf{[Eq 3]}$$

Translation: Regardless of what X, Y, and Z represent, if X supports Z and Z supports Y, then declare that X *supports* Y.

The relational data model cannot effectively model the "supports" relationship for two main reasons:

1. The relational model does not provide a mechanism for weak typing and, as such, cannot effectively express the fact that either a task or an activity can fulfill the supporter role.
2. Relational data models are expressible in the *relational calculus* (RC) language. Relational calculus provides the formalized, theoretical framework behind relational databases. Unfortunately, however, RC is not expressive enough to represent transitive (closure) relations such as "supports"* (Aho and Ullman 1979).

Libkin (2001) shows that, while this is true for RC, the same cannot be said with regards to the Structured Query Language 3 (SQL3; a.k.a. SQL99) standard, which includes a few nonrelational features. Having noted the theoretical significance of this result, however, practical implementations of transitive-closure relations are subject to vendor-specific syntax and limitations. If one is using a database that does not include full support of the SQL3 standard, then one can attempt an ad-hoc solution by combining nested triggers and materialized views (Sowa 2000). Both approaches, however, increase the complexity level for the system design and increase the likelihood of error. More sophisticated examples of complex relationships can be readily conceptualized as one considers the infinite possibilities for defining relationships using compound and recursive logical statements.

Modeling With Logic

The inability to perform complex queries, the lack of support for recursive relationships, and the inability to reason logically with data motivated the research team to investigate the potential benefits from utilizing a *logic inference engine*. An inference engine allows one to assert facts and define logical rules. Then, the inference engine will automatically infer new facts (knowledge) from those rules. The rules

* A transitive-closure relation R^* is defined over the binary relation R as follows: if $(a,b) \in R$, then $(a,b) \in R^*$. Likewise, if $(a,b) \in R$ and $(b,c) \in R^*$, then $(a,c) \in R^*$.

are defined in terms of conjunctions, disjunctions, and negations of predicate terms. Perhaps most importantly, the logic language allows one to define recursive rules, and hence, model complex relationships. Each predicate term asserts some quality or aspect of an object, or some binding relationship between objects. For example, Equation 1 uses a unary predicate *Task* and a binary predicate *SubprocessOf* in order to assert: if "X is a task," "Y is a task," and "X is a subprocess of Y," then "X supports Y." As can be seen in Equation 3, an inference engine allows for the definition of implicit relationships via logical operators. A thorough analysis of the expressiveness and reasoning capabilities of inference engines can be found in Russell and Norvig (1995).

Integrating a Logic Inference Engine

While the ability to model complex environments is impressive, the real power of an inference engine stems from its ability to automatically derive new knowledge. In a test environment, the research team used a forward-chaining inference engine called a Java Expert System Shell (JESS), in order to prove the applicability of this technology. The actual program used is presented in Appendix A. Rules were encoded into JESS in support of these complex queries:

- Show all facilities that are needed to support a specified METL (or task).
- Given two METLs (or tasks) show the contention resources that are required by both.
- Show all METLs that depend on the availability of a specific facility or other kinds of resource.

An inference engine such as JESS is particularly well suited for this environment because it allows for recursive queries and complex relationships to be readily modeled with first-order logic rules. As facts are entered into the system, any rule with a matching left-hand side (the antecedent) is fired in order to derive additionally new facts from the right-hand side (the consequents). This process for firing rules continues until no new facts can be derived for the seed fact. This is the procedure by which new knowledge can be consistently derived from existing knowledge.

Future Research Opportunities

While this experimentation successfully illustrated the potential benefits that could be gained, many issues need to be resolved in a real-world deployment. These application-dependent issues would need to be explored:

- Where should the rules be defined? Globally (where all users have access to them), locally (where each user may have their own rule definitions), or some

combination of the two (such as allowing general rules to be specified globally while user-specific rules are defined within a local scope).

- In terms of the system architecture, is it more beneficial to have the logic engine as a separate component that straddles the database, or is it more beneficial to acquire one encompassing product, such as a deductive database? While the former allows for greater flexibility and more deployment options, the latter allows for faster execution and cohesiveness. If the logic engine resides as a separate component, then a method would have to be implemented for bootstrapping the engine with the current fact set.
- What kind of logic inference engine is most beneficial for the intended use? A forward-chaining engine executes faster queries, but it sacrifices space in doing so. While the sacrifice of space for time may be quite suitable on the server-side, it may not be the desirable solution if the engine is deployed on the client-side. On a similar note, are there justifications for pursuing a fuzzy logic inference engine?

These issues would need to be further studied in order to suggest the most suitable deployment scenario for a given application or environment. In any case, if the goal is to enhance the knowledge discovery process, then automated reasoning should be considered as a critical component for any proposed solution.

3 Knowledge Discovery Through Visualization

Chapter 2 described a manner in which intelligence can be harvested from automated reasoning. This chapter addresses a complementary methodology of gaining intelligence through human introspection and analysis of visualized data. Information visualization techniques are studied to support this process of knowledge discovery from databases.

The design of a database-driven application typically proceeds by modeling data and requirements with an ER diagram. An ER diagram provides a visualization of the kinds of entities that need to be stored, as well as their corresponding relationships and constraints. The two-dimensional (2-D) ER diagram is a graphical representation of a data model that is readily understood by system designers. It allows the designers to both micro-analyze individual entities as well as to macro-analyze the overarching design. The designers can easily recognize design anomalies, such as isolated entities and redundant relationships by examining the ER diagram. Hence, even a simplistic visualization can provide insights into the data model that would otherwise be difficult or impossible to achieve. By applying the same reasoning, the premise being investigated is that various visualizations of stored entities and relationships will enable greater understanding of both the application data and the domain. Because the data stored in the I-METL database is mostly textual, as opposed to numerical, the visualization methods deemed applicable are based on rendering stored data with 2-D graph structures (a.k.a. networks)*.

* Here, and throughout the remainder of this report, the term “graph” is used in the discrete mathematics sense of being a network of objects. A graph represents a set of objects and a binary relationship that exists between those objects. Mathematically speaking, a *graph* $G(N, E)$ is defined to be a set of nodes (N) and a set of edges connecting the nodes ($E \subseteq N \times N$).

Visualizing Relationships With Graphs

A graph is a suitable structure for information visualization because humans readily understand their diagrammatic representations and also because they can be studied and analyzed with the powerful theory of discrete mathematics. As an illustration, consider the graph defined in Equations 4–6 and shown in Figure 2. While the text-based representation is more suitable for machine processing, the diagram in Figure 2 is more suitable for humans as it requires less time and effort for us to process and comprehend.

$$G = (N, E)$$
$$N = \{a, b, c, d, e\} \quad \text{[Eq 4,5,6]}$$
$$E = \{(a, e), (b, a), (d, b), (a, c), (e, d), (c, d)\}$$

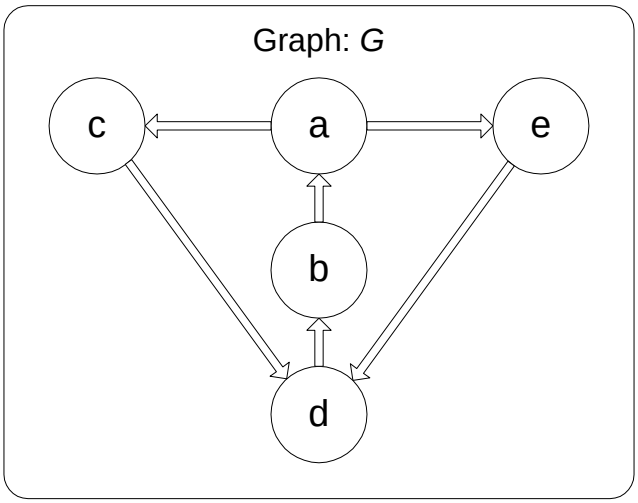


Figure 2. Diagram of graph defined by Equations 4-6.

Visualizing information with graphs is not a new idea. A thorough survey of applied graph visualization methods and issues can be found in Herman et al. (2000). However, a point of interest for this research effort is applying various graph visualization methods to the I-METL database. A goal of the research is to construct a test software system that allows discovery of hidden or unknown relationships that exist in the I-METL data.

Test Software for Database Visualization

To test and validate the hypothesis put forth, the research team designed and engineered a software system for visualizing the I-METL database. This test software is designed to construct and render 2-D object graphs from the data stored in the I-

METL database. It allows for many experimental displays to be created from the test database. Numerous parameters allow the researcher to control various aspects of graph generation, rendering, and viewing.

As shown in Figure 3, the test visualization software was constructed around core components. By using a component-based design, the software could potentially replace or interchange like-components later. These components are explained in detail following Figure 3.

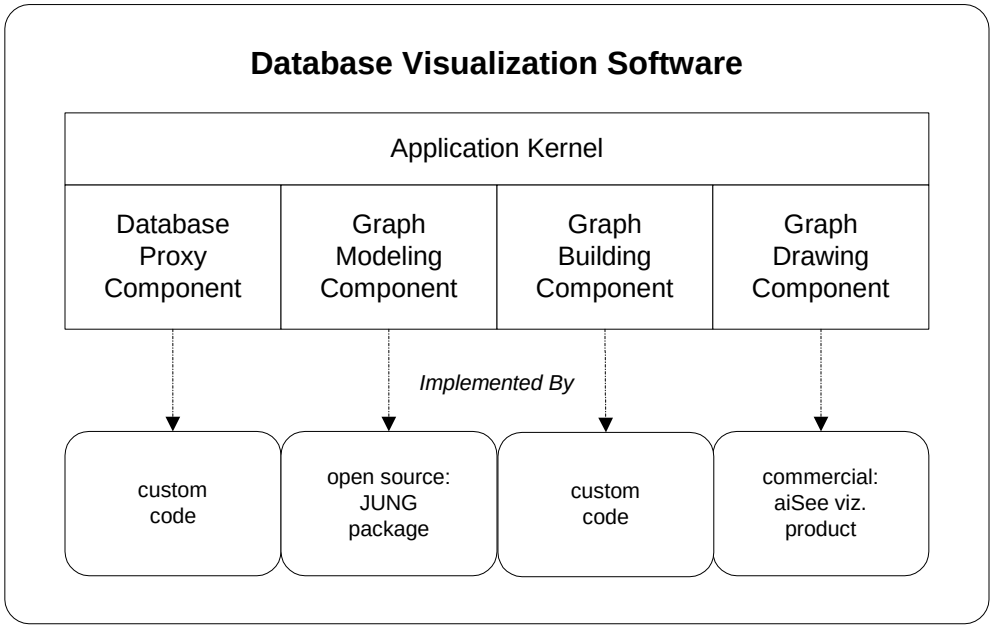


Figure 3. Component-based diagram of Database Visualization Software.

Database Proxy Component

The Database Proxy component allows the software to connect to and communicate with any Java Database Connectivity (JDBC)-compliant database. During the initialization phase, the proxy introspects the database by reading and interpreting the physical schema. Figure 4 shows a sample physical schema. Essentially, an in-memory object network is constructed to reflect the content and relationships defined by the physical schema. Interrelationships that exist between tables are mined from the schema by traversing foreign key constraints, which contain the necessary information for detecting exactly how tables interconnect. After the initialization phase is complete, the proxy serves as an independent moderator between the client application and the database. For example, the proxy can dynamically construct and execute SQL queries for a client that is built without any prior knowledge of the target database. This loose coupling increases the reusability of this component and allows the client application to be written independent of a particular schema.

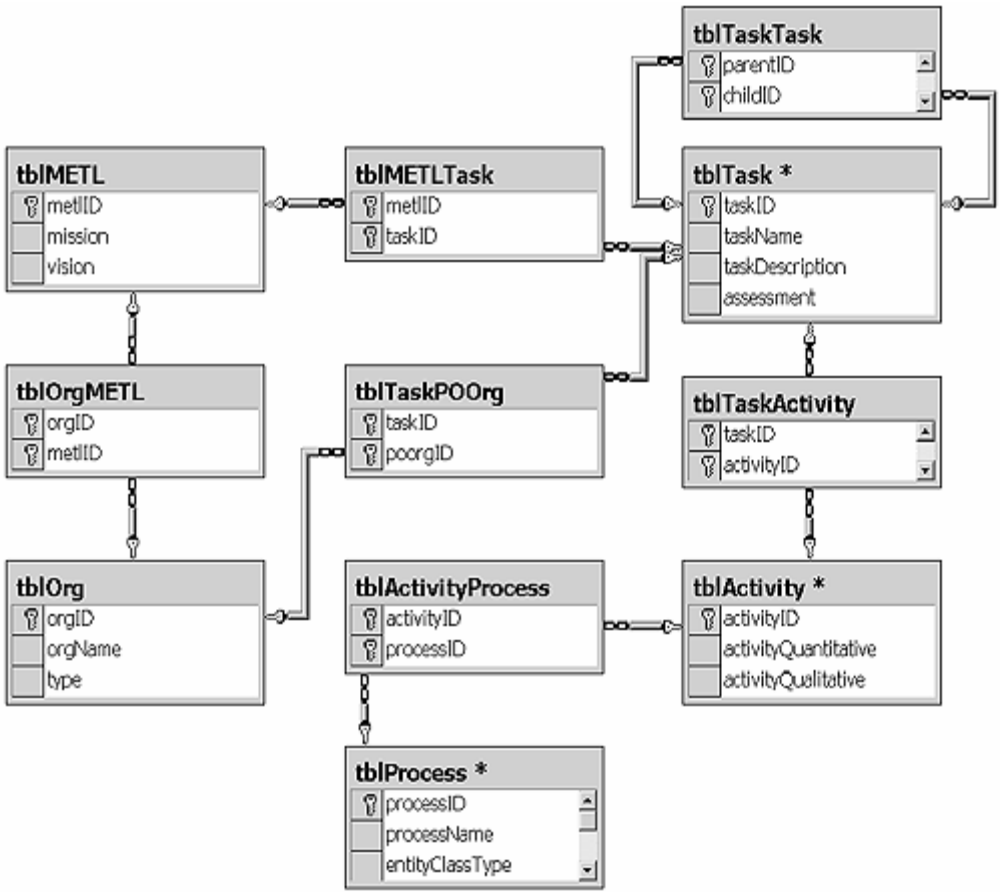


Figure 4. Sample physical schema diagram.

Graph Modeling Component

The Graph Modeling component is responsible for building an in-memory model of a graph and for allowing operations to be performed on that graph. The test software utilized the open source package JUNG (Java Universal Network/Graph Framework) for implementing this component. One of the requirements for the Graph Modeling component is to be able to represent many types of possible graphs. At a minimum, this component needs to be able to represent both directed and undirected graphs, labeled graphs, and distance graphs. While not a strict requirement, it is often desirable to be able to represent and operate on other kinds of graphs such as K-partite, hypergraphs, acyclic graphs, and trees. The JUNG package provides representations and operations for all the required graph types and for several recommended graph types. Additionally, a minimum set of graph algorithms, including breadth-first-search (BFS) and single-source shortest path, should be provided by any implementation for the Graph Modeling component.

Graph Building Component

The Graph Building component is responsible for constructing object graphs, annotating object graphs, and transforming those graphs into a suitable output format for the drawing component. The operations supported by the Graph Building component are going to have a high dependency on the target application. For this reason, it was necessary to custom code the operations supported by the Graph Building component. Some authors refer to these processes as "graph filtering and extraction" (Noik 1992).

The graph-building phase faces several issues. One issue corresponds to the existence of *bridge tables* in the physical schema. A bridge table is an intermediary table in the database design that is used to model many-to-many (and sometimes one-to-many) relationships. A record in a bridge table does not correspond to an actual entity from an ER model, but instead, functions as an implementation artifact. Typically the end-user would not have an interest in viewing these implementation details. During the graph-building phase, therefore, a function must be applied to filter-out bridge table elements. This filtering and extraction process mimics closely the Extraction, Transformation, and Loading (ETL) process that occurs in data warehousing. Furthermore, Noik (1992) identifies additional challenges critical to maximizing the displayable quality of data.

Graph Drawing Component

The Graph Drawing component serves as the Graphical User Interface (GUI) that is responsible for displaying input graphs. Technically speaking, two phases occur during graph drawing. The first phase is to take an input graph and use a layout algorithm for assigning positions to each node and properly routing each edge of the graph. The second phase involves actually rendering the graph onto the user's screen using traditional 2-D graphics. Furthermore, the Graph Drawing component allows the end-user to customize and interact with the displays of these visual graphs.

For this test software system, a third-party commercial-off-the-shelf (COTS) product was selected and acquisitioned for the Graph Drawing component. Generally speaking, graph drawing requires complex procedures and is still an area of active research. Appendix B details the criteria used in evaluating commercially available graph drawing products. For the purposes of this modest research effort, the software product *aiSee Graph Visualization* (AbsInt, Saarbruecken, Germany) was selected and acquisitioned to fulfill the project's needs. While the research team discovered that more capable products do exist (see Appendix B), the need to mediate the cost and risk to the project directed selection of the *aiSee* product. Nonetheless,

the *aiSee* product is quite capable of efficiently generating graph layouts of up to several thousand nodes and does incorporate some unique capabilities that the other products lack.

Application Kernel

The logic that ties all these components together and controls the flow of the application software is the Application Kernel. The first responsibility of the kernel is to accept and validate user input. This application allows the user to specify the search criteria that will be used for selecting the seed set of records. Querying the database through the proxy and interpreting the query results allows the kernel to formulate the seed set of records. The next step for the kernel is to perform a BFS from the seed set.

All records that can be reached from crawling up to N (a user-specified parameter) levels deep are then added to the set. A graph is built over this set utilizing the services provided by the Graph Modeling component. Next, the kernel calls the Graph Building component for traversing the graph and outputting a valid Graph Description Language (GDL) file. Finally, the kernel calls the external program (*aiSee*) for layout and rendering of the visual graph.

The test visualization software allows the user to control many aspects of the graph generation and display. In addition to controlling the graph extraction and filtering process, the user can customize the graphical output with various visual cues (e.g., shapes, colors, line styles, fonts, etc). Furthermore, the user can choose between various graph layout methods, such as hierarchical layout vs. force-directed (energy-based) layout. Figure 5 shows a sample graph display. Appendix C presents additional displays showing various layout and viewing options.

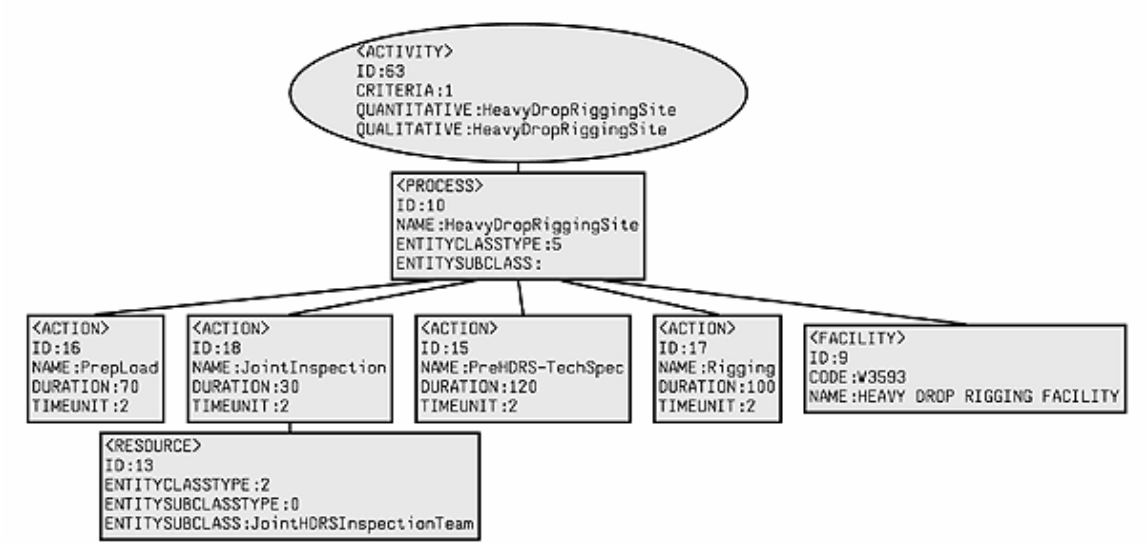


Figure 5. Sample output from Database Visualization Software.

Future Research Opportunities

The test software system provides the research team with a tool that can visualize database information. The benefit of this tool is its ability to generate numerous displays of information and experiment with many variables that control this visualization process. However, a number of research issues still remain that warrant investigation. These issues include:

- The two natural limitations on how much data can be effectively displayed: (1) the amount of available screen space and (2) the amount of information that can be cognitively digested within a given timeframe. Hence, one aspect of research should be to explore what the mean information threshold is for an end-user. For example, one may investigate (in the context of cognition) what the ideal relationship is between the number of displayed entities N and the number of attributes per entity M. Does M decrease inversely proportional to N, such as $M = C / N$ for some constant C?
- Nontraditional user interface controls that warrant investigation for applicability fall under the category of "focus+context" and include techniques such as *semantic zooming, flip zooming,* and *perspective walls.* These controls allow the user to zoom in on detail of an item in focus while retaining other items at their former level of granularity. Semantic zooming can be integrated into the user interface (UI) to create what is commonly referred to as a Zoomable User Interface (ZUI). Some issues related to "focus+context" methods are discussed in Herman et al. (2000).
- If visualization is the more intuitive methodology for presenting information, then it seems natural that visualization would be the more intuitive means

for querying and navigating the information space. A visual query interface would allow the end-user to formulate complex data queries without any prior programming knowledge. Within the context of information visualization, visual querying assumes the form of *graph-based querying*. A notable system in this arena that warrants further investigation is the *G+/Graphlog Visual Query Sytem* (Consens et al. 1992).

- While there are benefits to allowing the end-user to control the style and visual cues of the display, there may be overriding benefits to standardizing the graph display format. A *conceptual graph* (CG) is one proposed standard for formalizing the representation of knowledge. The CG model includes a common display format. For the purposes of consistency in displays and for the purposes of knowledge interchange, conceptual graphs should be further examined. Figure 6 shows a sample query graph presented with the CG display format.

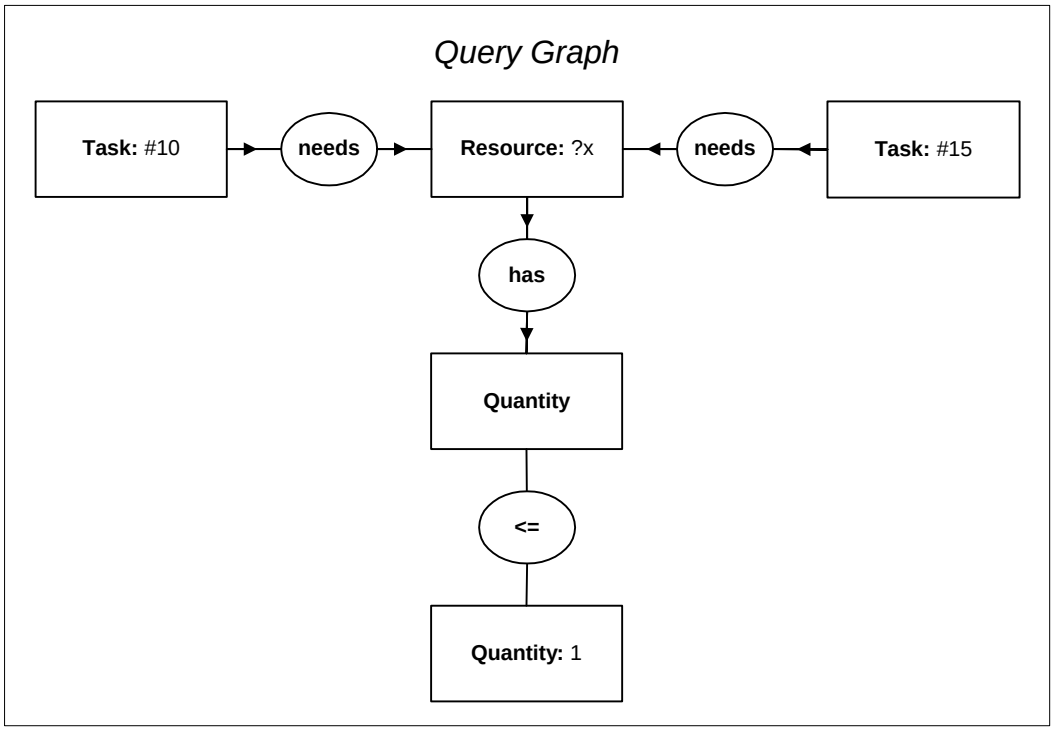


Figure 6. A sample query graph that locates all contention resources for tasks #10 and #15.

4 Integrated Design

The knowledge discovery methods described in this report can be combined into a single high-end analysis tool. Automated reasoning provides a complementary capability to visualization. While visualization allows one to gain a better understanding of semantics and relationships, the automated reasoning allows one to explore those relationships with powerful logic-based queries. Figure 7 shows one possible design of an intelligent visualization tool. In addition to the capabilities previously discussed, this design incorporates the additional capability of data mining. Data mining allows for descriptive and predictive analysis of data based largely on statistical methods. This complements automated reasoning by allowing one to explore inexact relationships as well.

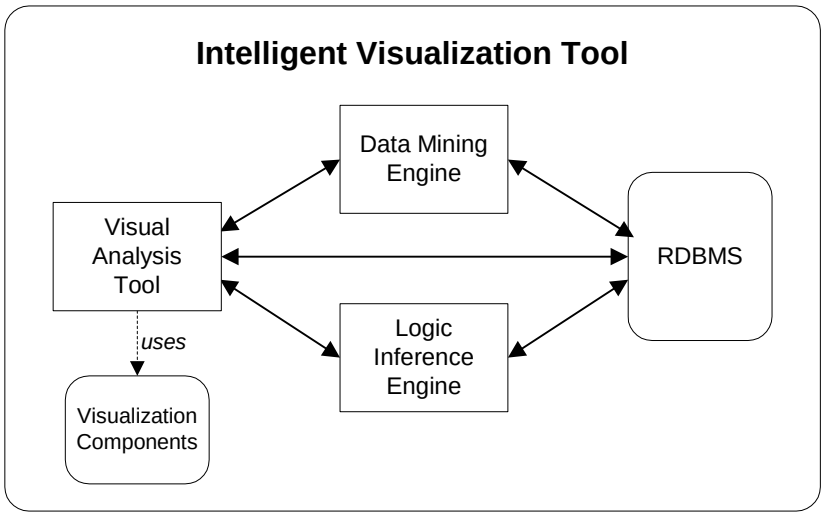


Figure 7. Design of an Intelligent Visualization tool.

5 Conclusions and Recommendations

Conclusions

This research investigated two disparate methodologies in the context of the I-METL application. These methods produced positive results for the knowledge discovery process and offer many potential benefits in a real-world deployment scenario. The prototype software developed validated these results and provided insights into future research possibilities. The integrated design presented in Chapter 4 illustrates how these methodologies can operate cohesively towards the common goal of KDD.

Recommendations

Opportunities exist for extending this research effort in both applied and theoretical directions. By applying the methods reported here to different application contexts, further ground truthing can expose unforeseen user requirements. These methods could potentially benefit application stakeholders by allowing them to improve their understanding of an application's data model and expose previously hidden data relationships. Correspondingly, the researcher would benefit from observing the stakeholder's learning process and being able to identify factors that contribute to or detract from the user's understanding.

Likewise, opportunities exist for continuing research in a theoretical direction. Research should be dedicated to evolving the design of an intelligent visualization tool as described in Chapter 4. Particular emphasis should be placed on evolving a visual query interface that retains the power of intelligent querying while succumbing to imposed limits in user interface complexity. Furthermore, research should be focused on identifying an architecture and implementation that supports an economy of scale, a high degree of adaptation, and the ability to support knowledge-feedback loops. A knowledge-feedback loop would allow users to submit and validate knowledge that they acquire from their KDD process. The goal of such an ideal system would be to (1) accelerate the learning cycle for the user, (2) allow the system to grow and learn from what the user learns, and (3) allow multiple users to cross-validate their knowledge and understanding of the domain of interest.

References

Aho, A.V., and J.D. Ullman, "Universality in Data Retrieval Languages," *Proceedings of the 6th ACM Symposium on Principles of Programming Languages*, January 1979.

Consens, M., I. Cruz, and A. Mendelzon, "Visualizing Queries and Query Visualizations," *SIGMOD Record (21)*, Vol 1, 1992.

Herman, I., G. Melancon, and M.S. Marshall, "Graph Visualization and Navigation in Information Visualization: a Survey," *IEEE Transactions on Visualization and Computer Graphics*, Vol 6, 2000.

Libkin, L., "Expressive Power of SQL," *Proceedings of the 8th International Conference on Database Theory*, Springer LNCS, Vol 1973, 2001.

Noik, E., "Challenges in Graph-based Relational Data Visualization," *Proceedings of the 1992 Conference of the Centre for Advanced Studies*, IBM Centre for Advanced Studies Conference, Nov. 9-12, 1992.

Odell, J.J., *Advanced Object-Oriented Analysis & Design Using UML*, Cambridge University Press, 1998.

Russell, S., and P. Norvig, *Artificial Intelligence: A Modern Approach*, Prentice-Hall, Inc., 1995.

Sowa, J., *Knowledge Representation: Logical, Philosophical and Computational Foundations*, Brooks Cole Publishing Co., 2000.

Appendix A: Using a JESS Program To Execute Transitive-Closure Queries

Below is the Java Expert System Shell (JESS) program used to illustrate the potential benefits that can be obtained from integrating a logic inference engine. Both the syntax and semantics of the JESS language are similar to that of C Language Integrated Production System (CLIPS) from which JESS is a derivative. The first responsibility of the program below is to define fact templates via the "deftemplate" statement. This provides a consistent structure for fact definitions. Secondly, functions are defined for executing custom queries and displaying the results. Thirdly, the inference rules are defined using "defrule" statements. The inference rules used below assume the form of Horn clauses, which are conjunctions of antecedents that implies a consequent. Lastly, both the queries and sample facts are defined in order to complete the testing.

```
;; ---------------------------------------------------------
;; Deftemplates
;; ---------------------------------------------------------

(deftemplate metl
   (slot mission)
   (slot vision))

(deftemplate task
   (slot name)
   (slot descr))

(deftemplate activity
   (slot descr))

(deftemplate process
   (slot descr))

(deftemplate resource
   (slot name)
   (slot quantity (type INTEGER)))

(deftemplate metl-task
   (slot metl)
   (slot task))

(deftemplate task-activity
```

```
   (slot task)
   (slot activity))

(deftemplate activity-process
   (slot activity)
   (slot process))

(deftemplate process-resource
   (slot process)
   (slot resource))

(deftemplate supports
   (slot sup)
   (slot sub))

(deftemplate task-resource-support
   (slot task)
   (slot resource))

(deftemplate metl-resource-support
   (slot metl)
   (slot resource))

;; ------------------------------------------------------------
;; Deffunctions
;; ------------------------------------------------------------

(deffunction mr-list-func ()
   (bind ?it (run-query mr-query (fact-id 1)))
   (while (?it hasNext)
   (bind ?token (call ?it next))
   (bind ?fact (call ?token fact 1))
   (bind ?r (fact-slot-value ?fact resource))
   (bind ?rname (fact-slot-value ?r name))
   (bind ?rquant (fact-slot-value ?r quantity))
   (printout t "METL '" (fact-slot-value (fact-id 1) mission) "'
needs " ?rquant " of RESOURCE " ?rname crlf)))

;;(deffunction task-intersect-func (?x ?y)
;; (bind ?it (run-query task-query-join ?x ?y))
(deffunction task-intersect-func ()
   (bind ?it (run-query tq-join (fact-id 2) (fact-id 3)))
   (printout t "RESOURCES required for both " (call (fact-id 2)
toString) crlf " and " (call (fact-id 3) toString) ":" crlf)
   (while (?it hasNext)
   (bind ?token (call ?it next))
   (bind ?fact (call ?token fact 1))
   (bind ?r (fact-slot-value ?fact resource))
   (bind ?rname (fact-slot-value ?r name))
   (bind ?rquant (fact-slot-value ?r quantity))
   (printout t " " ?rquant " of RESOURCE " ?rname crlf)))

(deffunction task-list-func (?x)
   (bind ?it (run-query tr-query ?x))
   (printout t "TASKS that depend on RESOURCE " (call ?x toString)
":" crlf)
   (while (?it hasNext)
   (bind ?token (call ?it next))
```

```
   (bind ?fact (call ?token fact 1))
   (bind ?t (fact-slot-value ?fact task))
   (bind ?tname (fact-slot-value ?t name))
   (bind ?tdescr (fact-slot-value ?t descr))
   (printout t "TASK(" ?tname ", " ?tdescr ")" crlf)))

;; ------------------------------------------------------------
;; Defrules
;; ------------------------------------------------------------

(defrule mt-support-rule
   (metl-task (metl ?m) (task ?t))
   =>
   (assert (supports (sup ?m) (sub ?t))))

(defrule ta-support-rule
   (task-activity (task ?t) (activity ?a))
   =>
   (assert (supports (sup ?t) (sub ?a))))

(defrule ap-support-rule
   (activity-process (activity ?a) (process ?p))
   =>
   (assert (supports (sup ?a) (sub ?p))))

(defrule pr-support-rule
   (process-resource (process ?p) (resource ?r))
   =>
   (assert (supports (sup ?p) (sub ?r))))

(defrule tran-support-rule
   (supports (sup ?a) (sub ?b))
   (supports (sup ?b) (sub ?c))
   =>
   (assert (supports (sup ?a) (sub ?c))))

(defrule tr-support-rule
   ?tt <- (task (name ?n1) (descr ?d))
   ?rr <- (resource (name ?n2) (quantity ?q))
   (supports (sup ?tt) (sub ?rr))
   =>
   (assert (task-resource-support (task ?tt) (resource ?rr))))

(defrule mr-support-rule
   ?mm <- (metl (mission ?m) (vision ?v))
   ?rr <- (resource (name ?n) (quantity ?q))
   (supports (sup ?mm) (sub ?rr))
   =>
   (assert (metl-resource-support (metl ?mm) (resource ?rr))))

;; ------------------------------------------------------------
;; Defqueries
;; ------------------------------------------------------------

(defquery support-query-1
   "Queries supports by sup"
   (declare (variables ?X))
   (supports (sup ?X) (sub ?Y)))
```

```
(defquery support-query-2
   "Queries supports by sub"
   (declare (variables ?Y))
   (supports (sup ?X) (sub ?Y)))

(defquery support-query-join
   "Join queries supports by sup"
   (declare (variables ?X ?Y))
   (supports (sup ?X) (sub ?Z))
   (supports (sup ?Y) (sub ?Z)))

(defquery mr-query
   "Finds resources that support given metl"
   (declare (variables ?m))
   (metl-resource-support (metl ?m) (resource ?r)))

(defquery tr-query
   "Finds tasks that require a given resource"
   (declare (variables ?r))
   (task-resource-support (task ?t) (resource ?r)))

(defquery tq-join
   (declare (variables ?x ?y))
   (task-resource-support (task ?x) (resource ?z))
   (task-resource-support (task ?y) (resource ?z)))

;; -------------------------------------------------------------
;; Deffacts
;; -------------------------------------------------------------

(deffacts factset1 "initial facts"
   (metl (mission "operate a regional center") (vision "provide a
world-class training center"))
   (task (name "support & enable the missions and readiness") (descr
"primary task"))
   (task (name "command, control & operate installation") (descr
"primary task"))
   (activity (descr "operations & maintenance"))
   (activity (descr "security"))
   (activity (descr "communications))
   (activity (descr "personnel management"))
   (process (descr "daily maintenance schedule"))
   (process (descr "security monitoring"))
   (process (descr "intelligence reviews"))
   (process (descr "QOS monitoring"))
   (resource (name "telecom network") (quantity 1))
   (resource (name "broadband network") (quantity 3))
   (resource (name "routers") (quantity 4))
   (resource (name "security officer") (quantity 2))
   (resource (name "O&M manager") (quantity 1))
   (resource (name "HVAC system") (quantity 2))
   (process (descr "pay bills"))
   (process (descr "train new personnel"))
   (process (descr "re-train existing personnel"))
   (process (descr "install new OS"))
   (process (descr "provide guidance to organization"))
   (resource (name "staff") (quantity 3))
```

```
   (resource (name "safety shelter") (quantity 5))
   (resource (name "wireless satellites") (quantity 4))
   (resource (name "network administrator") (quantity 3)))

(deffacts factset2 "more facts"
   (metl-task (metl (fact-id 1)) (task (fact-id 2)))
   (metl-task (metl (fact-id 1)) (task (fact-id 3)))
   (task-activity (task (fact-id 2)) (activity (fact-id 4)))
   (task-activity (task (fact-id 2)) (activity (fact-id 5)))
   (task-activity (task (fact-id 3)) (activity (fact-id 6)))
   (task-activity (task (fact-id 3)) (activity (fact-id 7)))
   (activity-process (activity (fact-id 4)) (process (fact-id 8)))
   (activity-process (activity (fact-id 4)) (process (fact-id 9)))
   (activity-process (activity (fact-id 4)) (process (fact-id 10)))
   (activity-process (activity (fact-id 4)) (process (fact-id 11)))
   (process-resource (process (fact-id 9)) (resource (fact-id 15)))
   (process-resource (process (fact-id 9)) (resource (fact-id 16)))
   (process-resource (process (fact-id 9)) (resource (fact-id 26)))
   (process-resource (process (fact-id 10)) (resource (fact-id 12)))
   (process-resource (process (fact-id 10)) (resource (fact-id 13)))
   (process-resource (process (fact-id 10)) (resource (fact-id 17)))
   (process-resource (process (fact-id 10)) (resource (fact-id 26)))
   (activity-process (activity (fact-id 6)) (process (fact-id 18)))
   (activity-process (activity (fact-id 6)) (process (fact-id 19)))
   (activity-process (activity (fact-id 6)) (process (fact-id 20)))
   (activity-process (activity (fact-id 6)) (process (fact-id 21)))
   (activity-process (activity (fact-id 6)) (process (fact-id 22)))
   (process-resource (process (fact-id 20)) (resource (fact-id 25)))
   (process-resource (process (fact-id 21)) (resource (fact-id 23)))
   (process-resource (process (fact-id 19)) (resource (fact-id 24)))
   (process-resource (process (fact-id 20)) (resource (fact-id 26)))
   (process-resource (process (fact-id 21)) (resource (fact-id 26)))
   (process-resource (process (fact-id 19))(resource (fact-id 26))))


(reset)
(facts)
```

Appendix B: Feature Comparison of Graph Drawing Components

Numerous graph drawing products are commercially available today. It became crucial for the research team to identify and prioritize the criteria to be used in evaluating these products. Two of the more relevant criteria for this research were (1) the degree of customizability and (2) the number and kinds of layouts supported. The research team utilized available online documentation for evaluating the products. Additionally, the higher-ranking products were obtained as demo versions so further analysis could be performed. Tables B-1 and B-2 give product evaluation details.

According to the specific needs of this research effort, it was deemed that the most suitable product was the *JViews* component suite by ILOG, Inc (Mountain View, CA). The *JViews* product can be deployed as a heavyweight Java application, medium-weight Java applet, or in a lightweight manner as a Java servlet that serves markup and images. Furthermore, *JViews* allows for customizable drawings to be rendered from Cascading Style Sheets (CSS) configuration files. These features made *JViews* the most flexible of the products evaluated; however, its cost and the associated learning curve that often accompanies higher-end technologies eliminated this product from consideration for this research effort.

Table B-1. Graph drawing software comparison (part 1).

Software	Cost for 1 license	Platforms	Input File Formats	Available Layouts
ILOG - Jviews www.ilog.com/jviews	$8,000	Java	SDM	C, E, T, O
yWorks - yFiles www.yworks.com	$5,760	Java	GML, GraphML	C, E, T, O
Tom Sawyer - TSV tomsawyer.com	$7,500	Java	TSGV	C, T, O
AbsInt - aiSee www.aisee.com	$600	Win/Unix/Mac	GDL	E, F, T, O
TouchGraph Link Browser touchgraph.sourceforge.net	free	Java	propietary (xml)	E
IBM - GraphViz www.graphviz.org	free	Win/Unix/Java	DOT	C, E, T
Oreas - GoVisual www.oreas.com	$3,850	Win	GML	C, T, O

Layout Codes
F=fisheye, T=tree/hierarchial, C=circular/radial, E=energy/force, O=others

Table B-2. Graph drawing software comparison (part 2).

Software	Deployable As	Customizable nodes/edges	Subgraph folding (nesting)	Documen-tation	Output Graphics
ILOG - Jviews	servlet, applet, application	Yes	Yes	Yes	Yes
yWorks - yFiles	application	Yes	Yes	Yes	Yes
Tom Sawyer - TSV	applet, application	Yes	Yes	Yes	Yes
AbsInt - aiSee	application	Yes	Yes	Yes	Yes
TouchGraph Link Browser	applet, application	No	Yes	No	No
IBM - GraphViz	application	Yes	Yes	Yes	Yes
Oreas - GoVisual	application	No	No	Yes	No

"Customizable nodes/edges" implies that various shapes, colors, and icons can be used for their display.

Appendix C: Catalog of Graph Drawings for Database Visualization

This appendix includes various graph displays that were generated with the Test Database Visualization software. These displays cover only a small fraction of the large number of graph displays that can be generated. The Graph Drawing component used for this system has more than 100 turnkey controls for customizing the graph layout and rendering. Furthermore, each individual edge and node can be customized with an additional 20 controls.

Figures C-1 through C-7 display hierarchical graphs with increasing diameters. Each of these graphs was built from performing a breadth-first-search (BFS) crawl from the entity (Activity #18). Figure C-5 illustrates the use of a force-directed graph layout. Figures C-6 and C-7 show how fisheye views can be applied to rendered graphs. The intent of these views is to provide context to the entities with maximal focus. Due to the limited amount of screen space available for drawing a graph, the fisheye view displays the focal nodes in complete resolution while displaying nodes farther away with decreasing resolution. The degree to which the warping occurs can be controlled with the graph drawing component's interface.

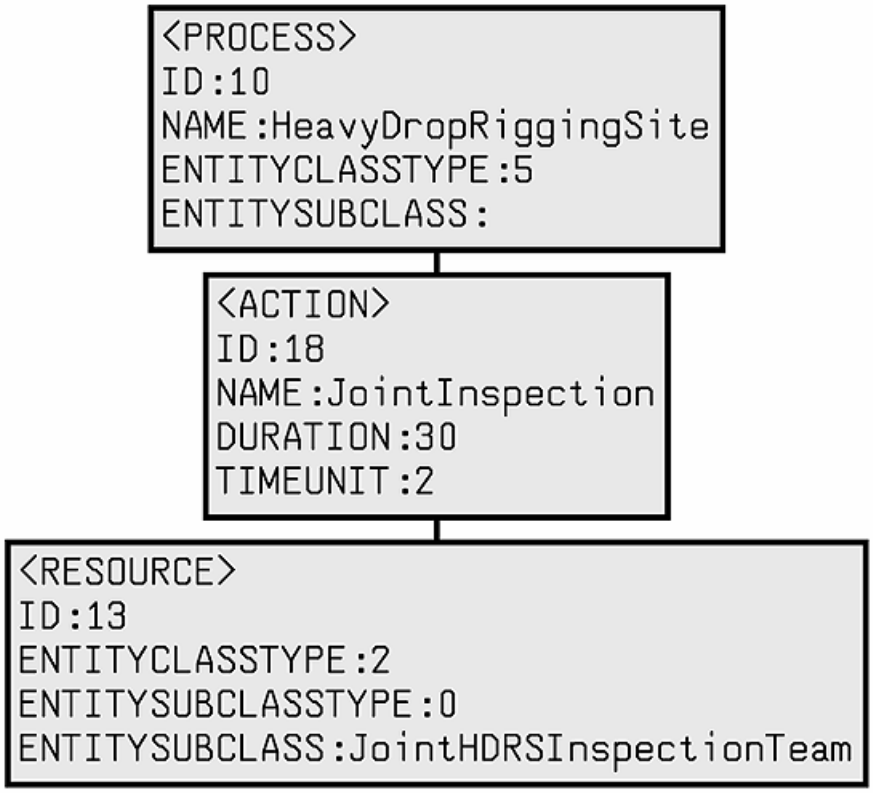


Figure C-1. Displaying a hierarchical object graph with diameter=2.

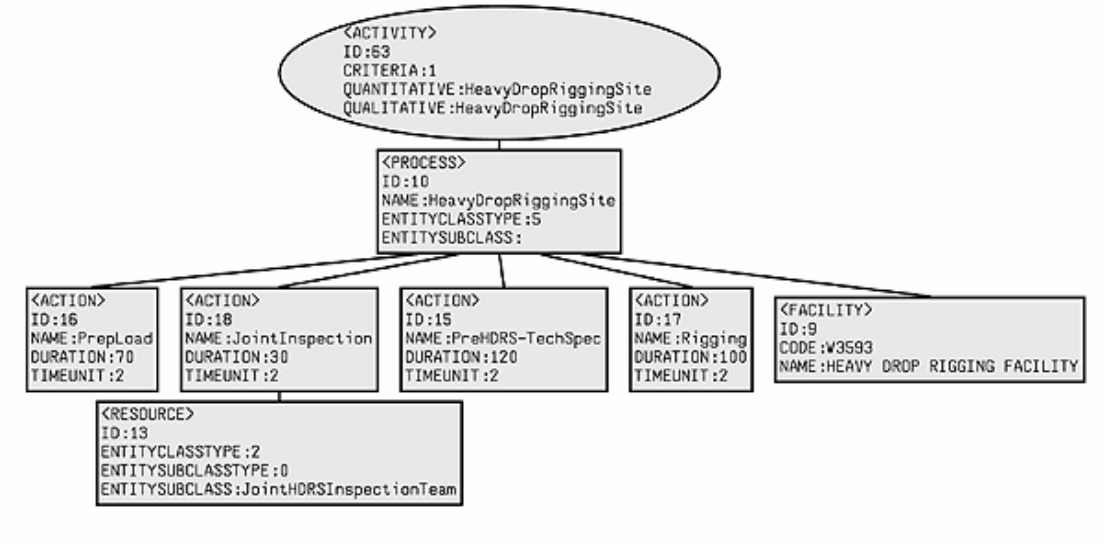


Figure C-2. Displaying a hierarchical object graph with diameter=3.

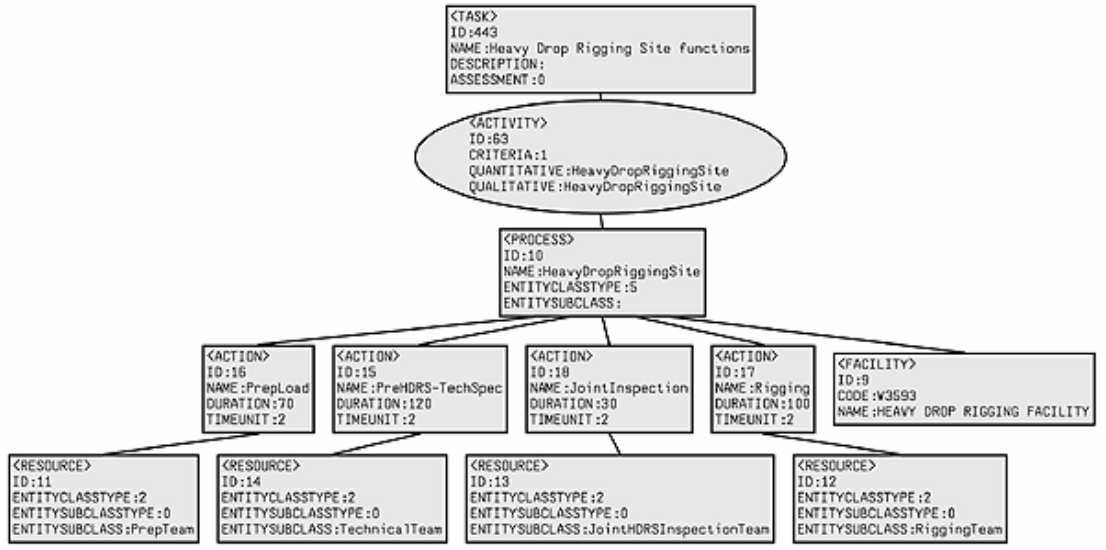


Figure C-3. Displaying a hierarchical object graph with diameter=4.

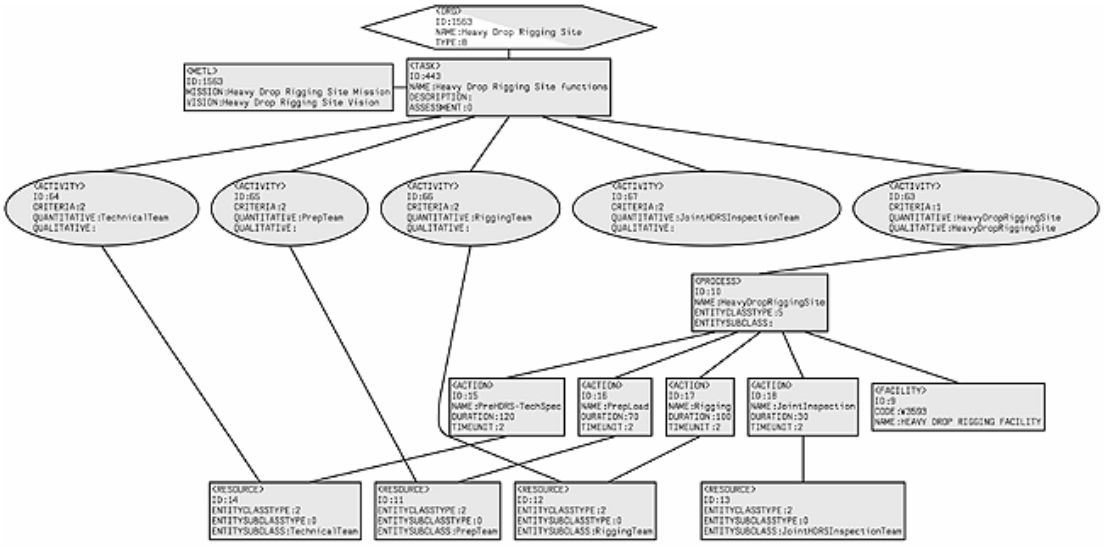


Figure C-4. Displaying a hierarchical graph with diameter=5.

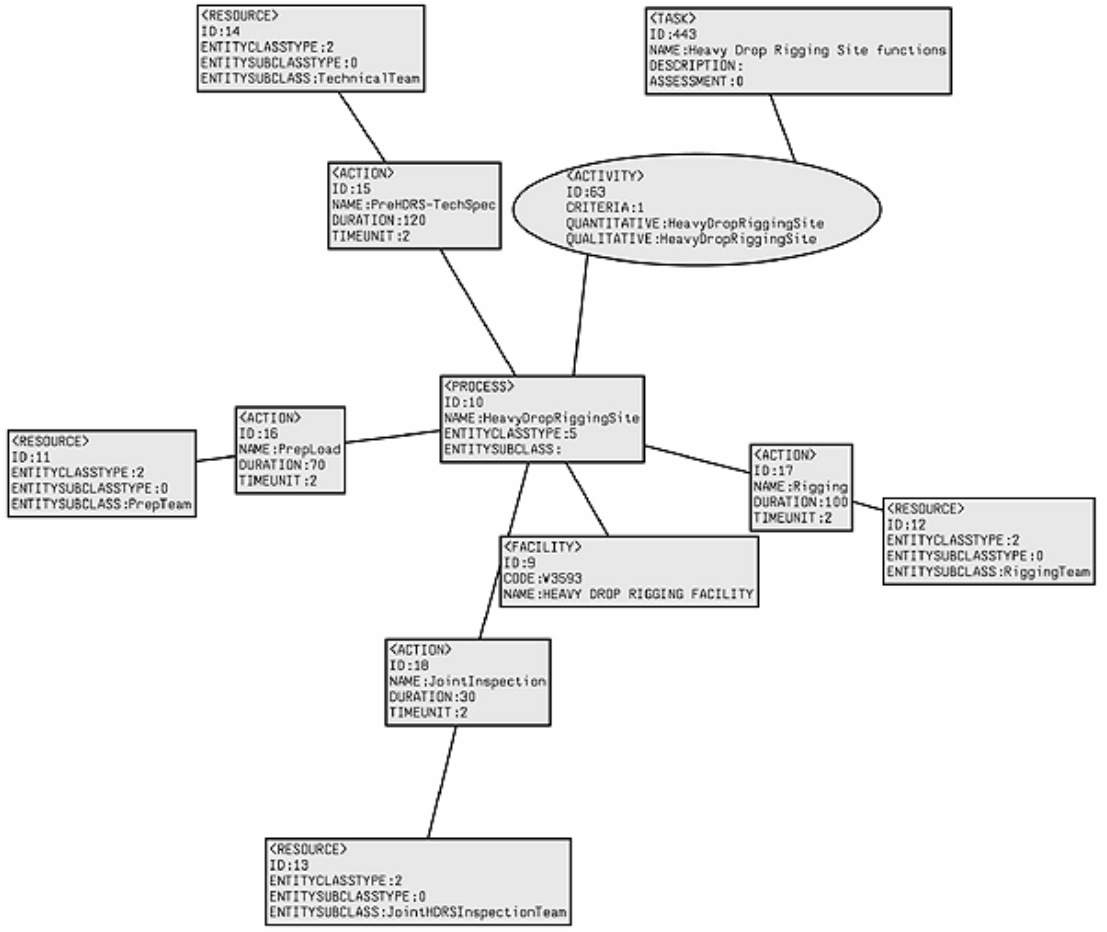


Figure C-5. Displaying a force-directed object graph with diameter=4.

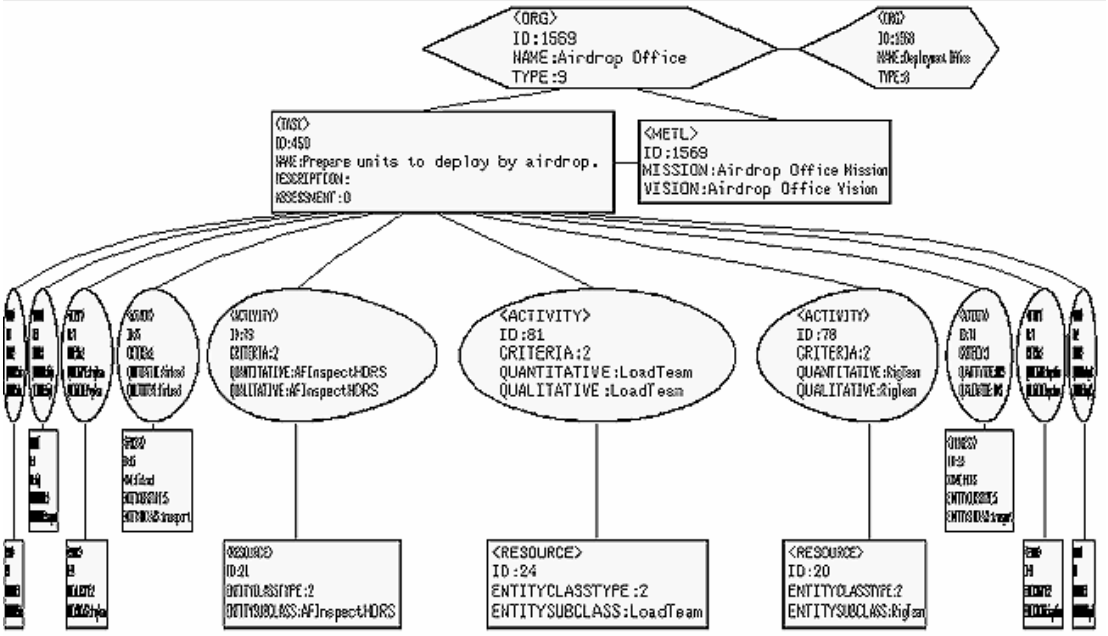


Figure C-6. Displaying a Cartesian fisheye view of a hierarchical object graph.

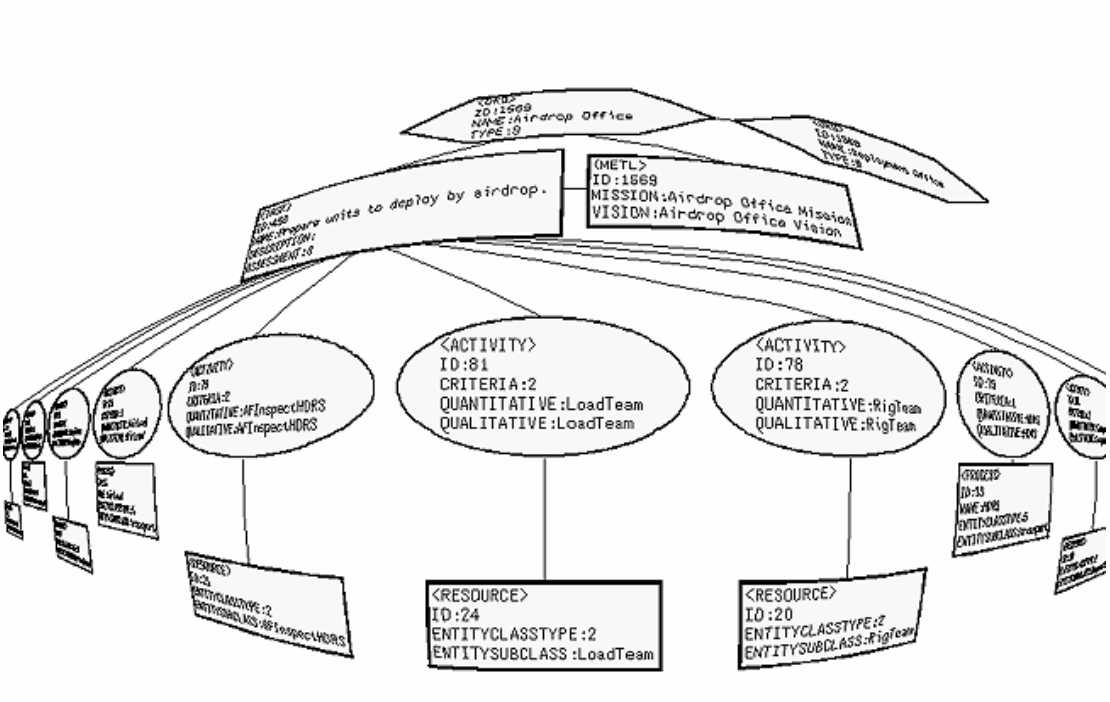


Figure C-7. Displaying a polar fisheye view of a hierarchical object graph.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE (DD-MM-YYYY)	2. REPORT TYPE	3. DATES COVERED (From - To)
09-2004	Final	

4. TITLE AND SUBTITLE
Knowledge Discovery in the I-METL Application

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Todd R. Littell

5d. PROJECT NUMBER
622784AT41

5e. TASK NUMBER

5f. WORK UNIT NUMBER
LK6K75/00S64L

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Engineer Research and Development Center (ERDC)
Construction Engineering Research Laboratory (CERL)
PO Box 9005
Champaign, IL 61826-9005

8. PERFORMING ORGANIZATION REPORT NUMBER
ERDC/CERL TR-04-14

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army Corps of Engineers
441 G Street, NW
Washington, DC 20314-1000

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES
Copies are available from the National Technical Information Service, 5285 Port Royal Road, Springfield, VA 22161.

14. ABSTRACT

The Installation Mission Essential Task List (I-METL) is a software system designed to support the modeling and analysis of garrison capabilities, tenant functions, and installation resources. From a system analyst's point of view, the main focus of the I-METL application is as a means for collecting, sharing, and managing structured data. As the stored data accumulates in size, quality, and richness, stakeholders begin to realize the potential for harvesting new business intelligence from the data store. To this end, a myriad of tools and methods (commonly referred to as Knowledge Discovery from Database [KDD] methods) are available depending on the kind of intelligence pursued.

This research investigated KDD methods that can directly benefit I-METL stakeholders. One goal of this effort was to provide a means for stakeholders to gain an increased understanding of the existing data and data relationships. Another goal was to foster the discovery of new and hidden relationships from the dataset. Methods that will assist with data exploration and cognition were also researched.

The two disparate methodologies investigated produced positive results for the KDD process and offer many potential benefits in a real-world deployment scenario. The prototype software developed validated these results and provided insights into future research possibilities.

15. SUBJECT TERMS
software, knowledge management, KDD, installation management, modeling, data collection, database management

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Todd R. Littell
Unclassified	Unclassified	Unclassified	SAR	39	19b. TELEPHONE NUMBER (include area code) (217)373-5873