Wizard: A Database Inference Analysis And Detection System

Harry S. Delugach and Thomas H. Hinke

Abstract—
The database inference problem is a well-known problem in database security and information system security in general. In order to prevent an adversary from inferring classified information from combinations of unclassified information, a database inference analyst must be able to detect and prevent possible inferences. Detecting database inference problems at database design time provides great power in reducing problems over the lifetime of a database. We have developed and constructed a system called Wizard to analyze databases for their inference problems. The system takes as input a database schema, its constituent instances (if available) and additional human-supplied domain information, and provides a set of associations between entities and/or activities that can be grouped by their potential severity of inference vulnerability. A knowledge acquisition process called microanalysis permits semantic knowledge of a database to be incorporated into the analysis using conceptual graphs. These graphs are then analyzed with respect to inference-relevant domains we call facets using tools we have developed. We can determine inference problems within single facets as well as some inference problems between two or more facets. The architecture of the system is meant to be general so that further refinements of inference information subdomains can be easily incorporated into the system.

Keywords—**information security, conceptual graphs, database inference, inference detection, inference analysis, transitive associations.**

I. Introduction

As more and more databases are made accessible to the public, many new problems have arisen in information security. One important problem is the database inference problem which is characterized as follows: *Information that is classified at level L_1 or below can be used to infer database information classified at level L_2 where the level L_2 is either higher than that of L_1 or is not comparable to L_1.*

The problem is illustrated as follows: Suppose we wish to keep secret the fact that the Smith Expedition is operating in a cold climate. While we may be able to prevent an adversary from obtaining this particular fact directly, the leader of the expedition Jane Smith may be shown in a database as the customer to whom several zero-degree-Fahrenheit sleeping bags were recently shipped. We could therefore assume the expedition was operating in some location where the temperature might reach zero degrees Fahrenheit.

Manuscript received February 1995; revised August 1995. This work was supported through Maryland Procurement Office Contract No. MDA904-92-C-5146.

Harry S. Delugach and Thomas H. Hinke are with the Computer Science Department of the University of Alabama in Huntsville, Huntsville, AL 35899 U.S.A. (e-mail: delugach@cs.uah.edu); (e-mail: thinke@cs.uah.edu).

IEEE Log Number ????????

Detecting such inference vulnerabilities is the job of the security analyst. Although such detection is possible at times of querying and updating the database, but we do believe that detection at design time is valuable since performing the analysis once at the start helps reduce inference vulnerabilities over the lifetime of a database. We have developed and constructed a system called "Wizard" to analyze databases for their inference problems. The system takes as input a database schema, its constituent instances (if available) and additional human-supplied domain information, and provides a set of associations between entities and/or activities that can be grouped by the severity of their inference potential. Such associations represent secondary inference channels and therefore represent a database vulnerability that could be exploited by a potential adversary.

We use conceptual graphs, as popularized by Sowa [17] as our internal representation. Conceptual graphs have previously been used to show inference-relevant information with semantic networks [19]. For details of conceptual graphs, the reader is directed to [12], [17], [18]; for a concise introduction, see [15].

Wizard supports the analysis and detection of certain database inference problems using conceptual graphs. It will advise the inference analyst as to whether certain specified pieces of information can be inferred from a given database, and if so, what information was used to perform the inference. Our goals in developing the system are as follows. The first three goals are met in the present work; the last two will be addressed within an inference analysis laboratory that we are currently developing.

1. To facilitate the acquisition of inference-relevant knowledge in the database.
2. To assist database designers and administrators in determining possible inference problems in their databases.
3. To characterize different groups of inferences in a well-defined manner using the tool.
4. To provide a "plug-in" architecture whereby new inference detection tools can be easily incorporated into an open framework.
5. To develop techniques that can optimize the detection of inferences to improve the system's efficiency.

This paper describes the general principles behind Wizard and how it operates. The Wizard system provides a technique for acquiring rich database semantics, partition-

ing that knowledge into inference-relevant domains, analyzing those domains for inference problems, and categorizing the severity of the problems detected. Wizard has three main subsystems, as shown in Fig. 1: (i) tools to support manual knowledge acquisition for the purposes of analysis, (ii) projection of database semantic knowledge into subdomains, and (iii) inference analysis of subdomains through automated tools. The first subsystem (knowledge acquisition) is described in [3] and [10]; the other two are described in this paper.

Fig. 1. Wizard subsystems.

The organization of the paper is as follows. First we briefly describe how we represent inference-relevant knowledge, both from the database itself and from the domain. Next we tell how we partition that knowledge for analysis purposes. We then describe how the analysis is done. We outline our prototype implementation and show excerpts from a worked example run. Finally we discuss some open issues and conclude the paper.

II. Capturing Database Semantics

Analysis of database inference problems begins by obtaining a relational database schema and instances in either SQL [20] or MSQL (a form of SQL with security levels [11], although using SQL alone restricts the usefulness of the analysis, since SQL lacks any security classification information. We use the database in three distinct ways: (*i*) to provide the basis for extraction of functional dependencies, automatically from primary keys, and manually where required, (*ii*) to form the basis for manual microanalysis by the inference analyst in creating graphs to capture its semantics, and (*iii*) to automatically instantiate the database graphs using attribute values from the database.

We use conceptual graphs as our representation for several reasons: (a) it is a well-known form of knowledge representation, (b) it is one of the base representations for the ANSI Knowledge Sharing Effort [1], and (c) some tools for automatically manipulating them are already being developed [4], [5]. In this work, we do not yet exploit the full power of conceptual graphs (e.g., in reasoning and retrieval); in the future we intend to support knowledge acquisition, an explanation facility for inference paths, and path-trimming based on common type information.

We represent the database's semantics in the form of conceptual graphs, with knowledge added by the analyst during a process we call *microanalysis*. This process relies on human-aided analysis whereby each relation schema is intensively analyzed for its meaning in the context the enterprise that is using the database. Such meaning is represented in one conceptual graph for each relation[1] schema; we call each graph a *microanalyzed knowledge chunk* or MKC.

[1]The words "relation" and "attribute" have two distinct meanings for our work; each has one meaning in the relational database domain, another in the conceptual graph domain. Unless the context is clear, we will use "database relation" or "database attribute" for relational terms; we will use "conceptual graph relation" or "CG relation", etc. to mean a conceptual graph relation, etc.

We then combine the MKC's we have created and use the Instantiator Tool to create an instance of each relation's schema graph from each database instance. The resulting set of conceptual graphs are called the *global ensemble of MKC's* or GEM. Details of the knowledge acquisition process are beyond the scope of this paper; interested readers are referred to [3]. The general process is suggested by Fig. 2.

Fig. 2. Microanalyzed knowledge chunks and global ensemble.

The GEM is a combination of all the knowledge from all subdomains. Once the GEM has been obtained, the inference analyst has available a rich semantic representation of the database under scrutiny. Rather than analyze the GEM *in toto*, we have found it effective to partition the GEM into broad knowledge domains called *layers* and further subdivided into specific domains called *facets*, both of which we will now describe.

III. Layer and Facet Descriptions

Information from the GEM is projected into three *layers*, which are a set of views based upon broad partitions of inference relevant information. There is an entity layer, an activity layer, and a layer in between with relationships between entities and activities. We further subdivide the layers into *facets*, each of which contains information about a subdomain of that layer. We thus have "sliced" the GEM into subdomains. Fig. 3 summarizes these divisions; we explain them below.

Fig. 3. Layers and their facets' constituent relations.

Note that the layers and facets contain the same information as in the GEM, only partitioned for easier analysis; whereas the GEM was constructed from a collection of MKC's, the layers and facets are obtained by slicing the GEM into domains based on the nature of their relationships. The three analysis layers correspond roughly to the inference classes identified in our previous work [2], [8]. Figure 3 shows their general contents. The entity layer contains knowledge of entities themselves and relationships between entities, e.g., entity-part-of, is-a, or functionally-determines. The activity layer contains knowledge of activities themselves and relationships between activities , e.g., activity-part-of, precedes, follows, or causes. The in-between layer contains relationships between entities and activities, e.g., produces, consumes, or used-for.

A *facet* is a single inference domain within a layer, such as "entity composition" which comprises all the entity-part-of relations (and the concepts they connect) within the entity layer. Fig. 3 shows relations in other facets; e.g., a func-dep facet (for functional dependencies), a temporal facet (with before and after relations), etc. Later in the paper we define each of the facets we have so far identified[2].

[2]Earlier papers [8], [10] called these facets *layers* and termed the MKC a Layered Knowledge Chunk (LKC), but the term was changed to *facet* to eliminate any connotation of a hierarchical relationship.

We said that our second subsystem constructs layers and facets. A facet is just a projection of the Global Ensemble to match certain pre-defined patterns. To consider a single facet means to effectively view the world as comprised of only a few certain relationships (usually just one) and to ignore all the others when performing its analysis.

Projection in this case means to view the GEM through a kind of conceptual "polarizing lens." Just as a light polarizing lens allows only light waves in a certain plane to pass through, the conceptual "lens" allows only certain concepts and relations to pass through. In practice, this projection is straightforward; for example, the entity composition facet of the entity layer is obtained by including only those concepts that are subtypes of [ENTITY] and are linked to each other via a (part-of) relation.

The rest of this section describes each layer and facet, telling what kind of information it contains, how that information is projected from the GEM, and how the information supports useful inferences.

A. Entity Layer

This layer comprises information about entities or relationships between two or more entities. Three facets of this layer have been identified so far: a sub-type facet, and functional dependency facet and a composition facet.

Entity Subtype (Is-A) Facet. For this facet, we consider only the subtype relationship between entities; i.e., those concepts that are entities or subtypes of entities. This involves projecting from the GEM all those occurrences of subtypes where either type is a subtype of the general type ENTITY, e.g., EMPLOYEE < PERSON. Subtype information is easily obtained from conceptual graphs. Inferences in this facet are based on the fact that an instance of a subtype implies an instance of its supertype: knowing some employee exists means that some person exists. Note that the reverse is not true; all employees are persons, but not all persons are employees. Since direction is significant (not all associations are reflexive), we discuss it further in Sec. 4.1.

Functional Dependency Facet. In this facet, we use the definition of functional dependency in its general use in relational databases (e.g., see [20]). In performing the microanalysis earlier, we supplied CG relations labeled (func-dep) that are incorporated into the GEM. In conceptual graph terms, this facet is obtained by projecting from the GEM only subgraphs that can be generalized to

$$[\,T\,] \rightarrow (\text{func-dep}) \rightarrow [\,T\,]$$

where [T] represents any concept. This effectively pulls out all pairs of concepts that are related by a functional dependency.

Functional dependencies are important because (1) they are based entirely at the database schema level and (2) in general they can be automatically extracted from the schema itself. From them, we can analyze the entire database for the presence of second paths (i.e., transitively derived functional dependencies) between pairs of attributes. The complete approach is described in [8] and [9].

Knowledge about functional dependency originates from any of the following:

1. Database schema with primary keys shown. We use this to derive implicit functional dependencies between a primary key and all other non-key attributes in its relation.
2. Foreign key information. For the purposes of our work, we treat foreign keys as synonyms for the primary key to which they correspond.
3. Non-key functional dependencies[3].
4. Subtype information (e.g., BOOT is a subtype of SHOE). The supertype can be considered a synonym of the subtype (e.g., SHOE is a synonym for BOOT).
5. Populated database with instances (if available) from which are derived empirical dependencies and cardinality (see Sec. ?? below).

Items 1 and 2 are known from the database schema. Items 3 and 4 are supplied by the inference analyst upon consultation with the database designer(s). Item 5 is obtained from the actual database.

Entity Composition Facet. Knowledge in this facet represents information about entities that are part of another (composite) entity. Its information is obtained by projecting only subgraphs that can be generalized to

$$[\text{ENTITY}] \rightarrow (\text{part-of}) \rightarrow [\text{ENTITY}]$$

Such associations support inferences as follows. If a certain kind of secret electric coil is part of a radar unit, then knowing where such a radar unit was being shipped would infer that a secret coil was also located there.

B. Activity Layer

This layer is comprised of facets having to do with activities themselves or with relationships between activities. We have identified three facets so far in this layer: a subtype facet, a composition facet, and a temporal facet. We now briefly describe each of them.

Activity Subtype (Is-A) Facet. For this facet, we consider only the subtype relationship between activities; i.e., those concepts that are either activities or subtypes of activities. This means projecting from the GEM all those occurrences of subtypes where either type is a subtype of the general type ACTIVITY, e.g., SWIMMING < WATERSPORT. Just as in entity subtypes, these associations support inference of an activity.

Activity Composition Facet. Here we have information about an activity that is a part of another entity. This facet is obtained by projecting only subgraphs that can be generalized to [ACTIVITY] → (part-of) → [ACTIVITY]. Inferences are supported since the existence of a whole activity infers the existence of each of its parts. For example, if BOATING is a part of DEEP-SEA-FISHING, then knowing that DEEP-SEA-FISHING is occurring infers that BOATING is also taking place.

[3] If the relation is in Boyce-Codd Normal Form, then all of the single-relational functional dependencies can be automatically extracted. There could, however, be some inter-relation functional dependencies. These would have to be entered explicitly by the inference analyst.

Temporal Facet. Information here is concerned with time-dependent relationships between activities. This facet is obtained by projecting only subgraphs that can generalize to any of the following:

[ACTIVITY] → (before) → [ACTIVITY]
[ACTIVITY] → (after) → [ACTIVITY]
[ACTIVITY] → (cause) → [ACTIVITY]

This information supports inferences in several ways. If activity IGNITION is before LAUNCHING, then knowing about LAUNCHING means that IGNITION must have preceded it. Similarly if activity SHIPPING causes the activity RECEIVING, then knowing about a SHIPPING activity infers a RECEIVING activity.

C. Entity-Activity Relationship Layer

This layer contains information used for inferences of relationships between activities and entities.

Used-For Facet. Information here is about entities that are used for particular activities. This facet is obtained by projecting only subgraphs that can be generalized to

[ACTIVITY] ← (used-for) ← [ENTITY]

This information supports inferences in that if, e.g., HAMMER is used for NAILING, then the existence of a NAILING activity infers that HAMMER exists.

Producer/Consumer Facet. This facet is obtained by projecting only subgraphs that can generalize to either of the following:

[ACTIVITY] → (produce) → [ENTITY]
[ACTIVITY] ← (consume) ← [ENTITY]

This facet is important for inferences because if a certain activity is known to have occurred, then whatever entity it produces is known to have been created. For example, if FIRE produces SMOKE, then if we know about a fire, then we can infer smoke.

IV. Path Analysis

In our work, we are interested in inference paths that lead from unclassified knowledge to classified or sensitive information. The links along an inference path represent relationships between entities or activities. After inference-relevant knowledge layers and facets have been identified, we use a fast algorithm we have developed to find such paths in each facet [9]. There are two main categories of these paths:

Single-Facet Paths. This is a set of paths between two items in a single facet that represent potential inference problems in the originating database. Each path represents a means of inferring a relationship that was not originally specified. A given path may or may not be an actual inference problem; we categorize them according to their potential severity (explained below).

Inter-Facet Paths. This is a set of paths between items that lie in two or more different facets. We use the previous single-facet paths to build the set of inter-facet paths. As with the single-facet paths, we categorize them according to severity.

Obtaining these paths is the primary focus of our work. This section gives details of how the paths are obtained and categorized. We have not implemented all of these features; this section provides a framework for generalized path processing for inference purposes.

We base our path analysis on the idea of *associations* that collectively form secondary paths to information. For our purposes, an association is treated simply as a directed relationship. Many useful inference relationships can be expressed as *transitive associations*; e.g., if a certain coil is a part of a microwave generator and a microwave generator is a part of a radar unit in a secret location, then the coil is a part of the radar unit. Even keeping the radar unit's location secret, we could still infer it from the delivery location of the coil. We refer to this secondary inference channel as a *second path* to the information (as in [7]), where the *first path* is the explicit association itself. In this case we could eliminate the inference by either classifying the parts breakdown or by classifying the delivery of the coil.

We originally developed our tools solely to analyze functional dependency in a database. Functional dependency is a well-understood transitive association in that, by definition, if C is functionally dependent on B and B is functionally dependent on A, then C is functionally dependent upon A. Our work in this area has been reported previously [8], [9], [10] Similar work has been done by SRI International [16] based on the original work by Hinke [7] on detecting second paths.

We quickly realized that the notion of functional dependency could easily be extended to transitive associations in general. The idea of detecting unclassified second paths can be applied to several relationships of interest to database security beyond functional dependencies. We have used several of them in this paper: part-of relations, temporal relations, used-for relations, etc. We therefore have extended our methods to handle transitive associations in general, as long as we are careful in describing the semantics of the association.

The Wizard system organizes known inference-relevant information and gleans certain types of transitive associations from it. Each type of transitive association is analyzed, and potential inference paths determined. Those inference paths are then categorized according to their severity; we explain below how the inference analyst may begin by focusing on those paths most likely to pose a practical inference problem. This section outlines how these steps are accomplished. First we make clear what we mean by an association and introduce four characteristics by which they may distinguished.

There are two separate issues involved in analyzing inference paths once they are detected. The first is a determination of the "goodness" of an inference path; i.e., whether we can actually infer information along the path. This corresponds to Morgenstern's function INFER(x,y) which defines how much information x gives us about y [13] [14] The second issue is based on the security classifications of the "good" paths; we must examine the classifications of both x and y to decide whether the path constitutes a real inference vulnerability or not. This section considers both issues; the first, in terms of the semantics of a given associ-

ation, and the second in terms of grading paths according to their severity.

A. Association Semantics

In our work, an association is merely a labeled directed arc between two components (either entities or activities). We represent this arc as one CG relation from the (small) set of inference-relevant CG relations we have identified; e.g., part-of, used-for, is-a, etc. Earlier we said that different analysis techniques were used based on which particular CG relation (or association) was involved. Fig. 4 summarizes the important characteristics of associations that we will use in our analyses. We now explain our basis for choosing the techniques.

Fig. 4. Different types of associations.

Direction

We define each association with an inherent direction. For example, the association part-of is directed from a whole to a part (in keeping with the interpretation of conceptual graphs). We define each association with a particular meaning in one arbitrary direction. The defined association direction we call **forward**, the other we call **backward**. For example, the part-of association as we have define it goes forward from a whole to its part(s); its backward association would be a (different) association composed-of. We will refer to it as the backward part-of relation.

In one direction, some inferences can be made with certainty. If we have an emergency kit, then we have a flashlight; if we have a flashlight, then we have a light bulb. In the reverse direction, inferences can also be made, but they are weaker: given a light bulb, we might have a flashlight, but we might also have a backlit computer display, etc. Given a flashlight we may or may not have an emergency kit.

Some reverse associations might appear fruitless for inference detection. For example, in an employee database relation with the social security number as a primary key, if we are given a particular social security number, we can uniquely determine a person's age, but given some particular age, we are unable in theory to determine whose age it is, unless there happens to be only one person with that age; in that case, we use empirical dependency, which describe next.

Cardinality

We are interested in the number of entities or activities that can be inferred through a given association. Take the entity-part-of association: given a radio that needs a certain kind of battery, we can infer the battery's existence with certainty. (We can also infer other parts of the same radio, but those are other distinct associations.) If we look at the association starting with the battery, a certain kind of battery may be a part of many different kinds of electronic devices. Given a battery, we cannot determine exactly which kind of electronic device it is a part of.

We distinguish four different cardinalities which affect the usefulness of inferences.

Many. This cardinality exists when the number of inferrable items is large enough so that no meaningful inference can be made. Obviously the precise quantity varies from situation to situation; we must use simple heuristic techniques to determine what that quantity is.

Few. This cardinality exists when there is more than one inferrable item, but the number is small enough to narrow down the possibilities to form a meaningful inference (albeit a weak one). For example, in a police investigation, if the number of suspects can be reduced to a small number, then each one can be questioned, etc. Again, the precise quantity that constitutes "few" depends on the situation; we are mostly interested in differentiating it from "many".

One. This cardinality exists when given a particular item and an association, only one inferrable item exists. For example, the association "biological mother of" has only one inferrable item; each person has only one biological mother.

Empty. This cardinality exists when there are no inferrable items from a given item and association. These constitute effective "dead ends" in the inference detection process; however, in some cases, the lack of any inferrable item might itself be a useful inference.

Epistemological Basis

We note two distinct bases for deciding why an association exists, and what is its cardinality:

Predefined Basis. These are definitional characteristics that are known by virtue of the structure of general knowledge (e.g., all persons have one biological mother) or by virtue of the structure in the information domain being analyzed. For example, if all dormitory rooms in a certain college are known to be single-occupant rooms, then knowing a room number permits the inference (with cardinality one) of its occupant's identity (assuming we also have the mapping from room to name).

Empirical Basis. These are analyst-supplied characteristics that are not part of the defined information structure, but comprise *de facto* characteristics that can be determined by inspection of instances in the knowledge base. For example, a hotel's registration database ordinarily allows for two or more persons per room, but it may be the case that on a particular night, all rooms are in fact booked to just a single person each. In that case, knowing a room number permits an unconditional inference of a particular person, even though the database definition would not require it.

The basis of an association is obviously important for practical inference analysis. A clever adversary may make inferences that are based on *de facto* characteristics of a particular set of database instances, rather than on any *a priori* constraints on those instances. The most important interaction between the epistemological basis and cardinal-

ity is that a predefined many-cardinality association can be treated as an empirical few- or one-cardinality once the knowledge base's instances have been inspected.

In our work, if some association has a predefined cardinality of many, we either ignore it during analysis (effectively sacrificing a degree of completeness for soundness), or else we calculate a cardinality (see below). We call the first strategy a *sound* strategy, and the second we call a *completeness-seeking* strategy. This feature is important when we are looking for potential inference problems, because an adversary may be interested even in weak inferences.

Transitivity

Transitive associations are directional relationships between entities or activities. We are interested in whether a given association r is transitive; i.e., given [A] → (r) → [B] and [B] → (r) → [C], can we assume that [A] → (r) → [C].

As an example of a transitive association, consider the "entity-part-of" relationship. If a light bulb is part of a flashlight, and a flashlight is part of an emergency kit, then a light bulb is part of the emergency kit. If we have an emergency kit, we can infer that we have a flashlight and from the flashlight we can infer that we have a light bulb.

We remind the reader that not all associations are transitive within a given facet. An association between two incompatible types might not be transitive. That is, an association from "entity" to "activity" cannot be transitive, since there can be no like association from "activity" to "activity" to connect with.

B. Applying Association Semantics

In this section, we apply the semantics above to the facets we have identified thus far. We have defined each association to conform to conventional conceptual graph usage; here we explain how the directions affect processing.

Entity and Activity Subtype (Is-A) Facet. Characterized as SUBTYPE < SUPERTYPE, this facet is processed in the forward direction; i.e., from subtype to supertype. Its cardinality in this direction is *one*, meaning that given an instance of a subtype, we are certain we also have an instance of the one supertype. This does not mean there cannot be another supertype (e.g., in multiple inheritance) but there is no ambiguity about the existence of the given supertype. This facet is processed as *transitive*.

Functional Dependency Facet. Characterized as [T1] → (func-dep) → [T2], where T2 functionally determines T1, this facet is processed in the backward direction. Its cardinality in this direction is *one*, meaning that given T2, we can infer T1 for certain. It is transitive, in that given some T3 that functionally determines T2, we can say that T3 functionally determines T1.

Entity and Activity Composition Facets. Both these facets are defined as WHOLE → (part-of) → PART. This facet is processed in the forward direction. Its cardinality is defined as *one*, since given a whole, we expect to have its parts[4]. Again, there may be more than one part we can infer for certain. It is transitive, since a part may itself be made up of smaller parts.

Temporal Facet. We have several relations in the temporal facet:

[ACTIVITY] → (before) → [ACTIVITY], or
[ACTIVITY] → (after) → [ACTIVITY], or
[ACTIVITY] → (cause) → [ACTIVITY]

In general, the cardinality of these relations is *many*, since there may be many unrelated activities that cause a certain activity, as well as many unrelated activities that are before a given one. As we said earlier, we can examine the database empirically to discover smaller cardinalities for inference purposes. These are treated in the forward direction. These relations are transitive.

Used-For Facet. Characterizing this facet as [ACTIVITY] ← (used-for) ← [ENTITY], the cardinality is *many*, since a given entity can generally be used for more than one activity and a given activity can be used for more than one entity. We consider this facet in the forward direction (i.e., in the direction of the arrows). It is non-transitive, since entities and activities cannot be mixed; we could not string it together with another used-for relation.

Producer/Consumer Facet. Characterizing this facet as

[ACTIVITY] → (produce) → [ENTITY], or
[ACTIVITY] ← (consume) ← [ENTITY],

its cardinality is *many* in either direction, and it is non-transitive (for reasons similar to the used-for facet).

C. Association Processing

For each facet, we categorize its association(s) according to the four characteristics we have introduced. Fig. 4 summarized how each characteristic affects our processing. We then perform the processing for each facet. The overall process is shown in Fig. 5.

Fig. 5. Association processing.

In our previous work [10] [9] we have described a lossless-join algorithm based on Navathe-Elmasri [6] and Ullman [20] as modified. It allows us to quickly obtain the endpoints of transitive paths given any set of transitive associations, taking into account their initial security classification levels.

There are usually many paths detected by the lossless-join algorithm. Many of these paths represent *bona fide* inference problems; however, many of them are not relevant to inference for various reasons. We grade detected paths by their severity based on the relationship between the classification levels of the ending points of the detected path versus any direct (i.e., single-hop) paths between those same points. This grouping is therefore based on the database schema using its own defined classification levels. Complete details of our severity groupings are beyond the scope of

[4] This assumes all entities are complete entities. We have not yet addressed the issue of incomplete entities; e.g., a car without tires is still considered a car.

this paper; we summarize here the four grades we identified to which a path might belong[5]:

Red Grade - detected paths where one or both endpoints only appear in single-hop paths at a higher (or non-comparable) classification level than the detected path. All such paths are certain to represent an inference vulnerability.

Yellow Grade - detected paths where one or both endpoints appear in single-hop paths at both higher (or non-comparable) and lower classification levels than the detected path. These paths may represent an inference vulnerability.

White Grade - detected paths where neither endpoint appears in any single-hop paths from the original schema. These are detected paths about which we have no other information; they may or may not be inference vulnerabilities.

Green Grade - detected paths where both endpoints appear in single-hop paths at lower classification levels than the detected path. These paths do not represent an inference vulnerability, since the detected path is already known from inspection of the database at a lower level.

The database designer can subsequently adjust classifications or the design as he chooses, using the grades as a guide. Path analysis thus provides the designer with a powerful tool that can advise him as to potential inference problems in a database, and help him decide which ones are most important.

D. Inter-Facet Analysis

In the previous section, analysis was portioned out by facet boundaries so that any interactions between the facets were effectively ignored. For practical inference analysis, such *inter-facet interactions* are an important source of further potential inference problems. We have developed the following technique to detect at least some of these inferences.

A natural extension to single facet analysis is to link endpoints of two single-facet paths to form one inter-facet path. In other words, given some single-facet path P with endpoints p_i and p_j, and another single-facet path Q with endpoints p_j and p_k, we can create a new inter-facet path R from p_i to p_k.

We can efficiently perform inter-facet analysis using the paths we obtained from our single-facet analysis. The advantage of doing the single-facet analysis first (aside from the obvious benefit of obtaining the single-facet inference paths themselves) is that we can use them to exploit their transitivity within a single facet wherever it has been determined – we need only use the path endpoints when crossing facet boundaries; we do not have to re-derive them in combination with all the other facets. Fig. 6 illustrates the process. It shows three facets: entity composition, used-for and activity composition. The dots indicate pairs of endpoints of paths already determined through single-facet analysis. The illustration shows how we establish a path from cord in the entity layer to rafting in the activity layer. It also shows how a path from entity cord to activity trekking can be established. Finally it also shows how a path from cord to flashlight, both in the entity composition facet, can be obtained using information from the used-for facet.

Fig. 6. Establishing inter-facet paths.

The paths in Fig. 6 illustrate two important points about inter-facet analysis. First, the use of multiple facets may allow us to derive paths from two components in the same facet that were not derivable from just the single-facet analysis. In Fig. 6, the path from cord to flashlight (both in entity-composition facet) was not derived just within the facet; it required information from the used-for facet (e.g., tent and flashlight both used for camping) in order to derive the path. Secondly, there are some pairs of facets for which it is not possible to find inter-facet paths. For example, there cannot be direct one-hop inter-facet paths from the entity-composition facet to the activity-composition facet; their axes do not share any common components. On the other hand, the entity-composition and used-for facets can be linked by a single hop since they both share a common axis (namely, entities).

Inter-facet paths will involve concepts and relations that were supplied by the analyst during microanalysis and will likely not have security classification levels assigned to them. Our default therefore will be to consider concepts not from the database as essentially unclassified (i.e., available to any adversary) for the purposes of path grades. It is also possible that the analyst could manually assign higher security levels to particular concepts if he knows that they do, in reality, represent information whose security is to be preserved.

The combinatorial complexity of inter-facet detection is exponential in the worst case, that is, if every concept is related via a single-facet path to every other concept. That is to be expected; however, in real-world databases this worst case is unlikely to occur. We have used a prototype of Wizard to perform the inter-facet analysis. We used a Sun Sparc 5 with 24Mbytes of memory running at 80MHz using our own benchmark database of 36 relations. Fig. 7 shows the run times for obtaining the multi-facet paths for differing number of instances. We would like to perform the analysis on a large existing database to determine its practical utility.

Fig. 7. Performance results in obtaining multi-facet paths.

Our last task is to rank the inter-facet paths according to their severity, much as we did for the single-facet paths. Since the inter-facet paths are not directly based on the database schema (whose original security classifications are known), the inference analyst must provide security classifications to the GEM via each relation's microanalysis.

[5]In an earlier paper, we called these grades *classes* [10]; we changed the name to avoid confusion with security classification.

This provides the "original" paths for purposes of ranking the new-found paths. We are still exploring this aspect.

V. Implementation

The Wizard architecture that we outlined at the beginning of this paper will now be fleshed out in more detail. In Fig. 8 the boxes denote collections of information; the ovals denote processes, either automatic or manually assisted.

Fig. 8. Wizard system overview.

The first step is microanalysis for database semantics using a set of standard form graphs called the *CG canon*, along with real-world knowledge and the database schema. The semantics are represented by one conceptual graph for each relation schema; we call each graph a *microanalyzed knowledge chunk* or MKC. We then combine the MKC's we have created and use the Instantiator Tool to create an instance of each relation's schema graph from each database instance, resulting in a set of conceptual graphs called the *global ensemble of MKC's* or GEM. Since we base the graphs of the GEM on the original database relation schemata, each tuple is used to fill in individual values to form instance graphs. This process is performed automatically by the Instantiator Tool; each concept box originating from the relation schema itself is supplied with an individual identifier (a conceptual graph *referent* that is taken from its corresponding attribute's value. Each instance concept is then labeled with its corresponding security classification level.

We have constructed a Parser/Projector Tool that (1) parses the conceptual graphs of the GEM, and (2) performs the projection of each facet's information by replicating them and using the database's tuples to fill in values. We then use standard parsing techniques on conceptual graphs to glean each facet's contituent relations (e.g., (part-of), (used-for), etc.) for analyzing their paths.

We have developed a tool called Merlin that is adaptable to detect inference problems in the different facets. Merlin's processing is based on the association processing described above. Merlin single-facet tools support the detection of second paths which it then grades according to their potential severity or importance. In the future it will support the empirical basis with some refinements (see Discussion below).

Merlin inter-facet tools find paths between concepts in different facets. In an prototype run of the system we used a database benchmark for inferences that we have developed that has 39 relations and approximately 3000 tuples. In modeling three enterprises: a manufacturer, its supplier and its customer, we have successfully detected both single-facet and inter-facet paths.

Single facet paths detected by the prototype include an inference from company to division to employee in the entity-part-of facet, and an inference from part number to order number to customer number in the func-dep facet. Multi-facet paths include a path from a spindle part to a cottonpicker in the entity-part-of facet and then to the activity of cottonpicking in theused-for facet.

We are developing a complete inference detection and analysis product based on the design in Fig. 8 that will support all parts of the system in an integrated environment.

VI. Discussion

One important issue is why we do not analyze the GEM directly, but instead project it onto facets for analysis. Since many useful inferences reside in a single facet, we have developed the Merlin tool to efficiently analyze transitive associations within each facet separately so that some portions of the analysis can be performed rapidly. Of course, we could deal with the GEM directly, but at the cost of reduced performance of whatever analysis methods we choose, since we not only have many more concepts to deal with, we also must deal with arbitrary combinations of concepts and relations, which increases the combinatorial complexity of any solution. Our solution is a divide-and-conquer approach, and is particularly amenable to concurrent implementation, since each facet can be independently processed.

We have merely begun the identification of useful facets; therefore, our architecture allows for additional facets to be easily included. We clearly have further techniques to explore that are based on associations other than transitive ones. In particular, we need to explore techniques to address inferring relationships between relationships. These do not appear to come under the transitive association paradigm, yet they might form an interesting set of inferences in their own right. Although performing single-facet analysis separately reduces the work needed to detect the multi-facet paths, it is also possible that certain inference paths would be rendered undetectable by our division; we are studying this aspect to determine these inference paths and develop techniques for analyzing them.

We need to further clarify the notion of "few" vs. "many" in dealing with associations that involve database instances. We may want to incorporate additional domain knowledge from a human analyst to make our determination more accurate. We also need to address the issue of incomplete entities (or activities), where some parts are missing but the entity (or activity) is still deemed to exist.

We do not yet recommend particular solutions to the inference problems we have detected. It is our claim that with a repertoire of fast and flexible analysis tools, the designer can explore critical database attributes (possibly reclassifying some of them) and then quickly re-analyze the database to observe any improvements. Given fast feedback, the designer will be able to gain valuable insight as to how changes in classification affect the overall security of the database, at least with respect to inferences.

There are some interesting variations to our approach that can be attempted in further refining the analysis of an existing database, as well as exploring our analysis techniques. With respect to cardinality, we could perform a sensitivity analysis; i.e., if a cardinality is many, then assume none and observe the difference in results, then assume one and observe further differences. This can give

some indication to the analyst as to the reliability of the initial processing. With respect to classification, we can consider the effect of providing security classification levels to knowledge that is strictly outside the database, by attaching those levels to conceptual graph during micro-analysis.

VII. Conclusion

We have developed techniques for handling a complete knowledge acquisition, detection and analysis process for an important set of database inference problems. We have implemented these techniques in a prototype version that has helped us identify many of the issues we have presented here. The results of analysis are a set of practical problems for the database designer, organized by their severity or importance, which can be addressed in actual databases.

We acknowledge that the problem of detecting database inference vulnerabilities requires modeling an amount of information comparable to the information that a determined and intelligent adversary would have. We believe, therefore, that any database inference detection approach must incorporate real-world knowledge in order to address the problem. Our approach uses conceptual graphs to model general knowledge that the analyst deems relevant, and then uses a fast algorithm to quickly detect potential (not always actual) inference problems.

Our system can easily incorporate new tools. Tools that use conceptual graphs can use the results of microanalysis for other purposes related to database design, analysis and implementation. Tools that require some new facet need merely to have the parser/projector glean whatever concepts and relations are relevant to that new facet. The extensibility of the system makes it an ideal environment for exploring database inference.

Acknowledgments

The authors wish to thank Randall W. Wolf for his substantial effort in implementing the tools presented in this paper. Asha Chandrasekhar did the preliminary design and implementation for functional dependency.

References

[1] ANSI, "Information Resource Dictionary System (IRDS) Technical Report, Part 1: Conceptual Schema for IRDS". Technical Report X3/TR-14:1995, American National Standards Institute (ANSI), 1995.

[2] Harry S. Delugach and Thomas H. Hinke. "AERIE: Database Inference Modeling and Detection Using Conceptual Graphs". In Heather D. Pfeiffer and Timothy E. Nagle, editors, *Conceptual Structures: Theory and Implementation*, number 754 in Lecture Notes in Artificial Intelligence, chapter 16. Springer-Verlag, 1993. ISBN 3-540-57454-9, reprinted from *Proc. Seventh Annual Workshop On Conceptual Graphs*, New Mexico State University, Las Cruces, New Mexico, July 8-10, 1992.

[3] Harry S. Delugach and Thomas H. Hinke. "Microanalyzed Knowledge Chunks For Knowledge Acquisition in Database Inference Analysis". Tech. Report 95-01, Dept. of Computer Science, Univ. Alabama in Huntsville, 1995.

[4] Gerard Ellis and Robert A. Levinson, editors. *Proceedings of the Second International Workshop on PEIRCE: A Conceptual Graphs Workbench*, 1993. Held in association with the First Intl. Conf. on Conceptual Structures, Laval University, Quebec, Canada, Aug. 1993.

[5] Gerard Ellis and Robert A. Levinson, editors. *Proceedings of the Third International Workshop on PEIRCE: A Conceptual Graphs Workbench*, 1994. Held in association with the 2nd Intl. Conf. on Conceptual Structures, Univ. Maryland, College Park, Aug. 1994.

[6] Ramez Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems*. The Benjamin/Cummings Publishing Company, Inc., Redwood City, CA., 1989.

[7] Thomas H. Hinke. "Inference Aggregation Detection in Database Management Systems". In *IEEE Symposium on Security and Privacy*, Oakland, CA, USA, April 1988.

[8] Thomas H. Hinke and Harry S. Delugach. "AERIE: An Inference Modeling and Detection Approach For Databases". In B. W. Thuraisingham and C. E. Landwehr, editors, *Database Security, VI: Status and Prospects*, number A-21 in IFIP Transactions, Amsterdam, 1993. Elsevier Science Publ. (North-Holland).

[9] Thomas H. Hinke and Harry S. Delugach. "A Fast Algorithm For Finding Second Paths in Database Inference Analysis". *Jour. Computer Security*, 1995. (in press).

[10] Thomas H. Hinke, Harry S. Delugach, and Asha Chandrasekhar. "Layered Knowledge Chunks For Database Inference Detection". In *Proc. 7th IFIP WG 11.3 Working Conference on Database Security*, Huntsville, Alabama, Sept. 1993.

[11] Donovon Hsieh, Teresa F. Lunt, and Peter K. Boucher. "The Seaview Prototype". Technical Report A012, SRI International, August 1993.

[12] G. W. Mineau, B. Moulin, and J. F. Sowa, editors. *Conceptual Graphs for Knowledge Representation*. Number 699 in Lecture Notes in Artificial Intelligence. Springer-Verlag, 1993.

[13] Matthew Morgenstern. "Security and Inference in Multilevel Database and Knowledge-Base Systems". In *Proceedings of SIGMOD (ACM Special Interest Group on Management of Data*. ACM, 1987.

[14] Matthew Morgenstern. "Controlling Logical Inference in Multilevel Database Systems". In *1988 IEEE Symposium on Security and Privacy*, Oakland, CA, USA, April 1988.

[15] Simon Polovina and John Heaton. "An Introduction to Conceptual Graphs". *AI Expert*, pages 36–43, May 1992.

[16] Xiaolei Qian, Mark E. Stickel, Peter D. Karp, Teresa F. Lunt, and Thomas D. Garvey. "Detection and Elimination of Inference Channels in Multilevel Relational Database Systems". In *Proceedings 1993 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 196–205, May 1993.

[17] John F. Sowa. *Conceptual Structures: Information Processing in Mind and Machine*. Addison-Wesley, Reading, MA, 1984.

[18] William M. Tepfenhart, Judith P. Dick, and J. F. Sowa, editors. *Conceptual Structures: Current Practices*. Number 835 in Lecture Notes in Artificial Intelligence. Springer-Verlag, 1994.

[19] Bhavani Thuraisingham. "The Use of Conceptual Structures for Handling the Inference Problem, and Cover Stories for Database Security". In *Proc. 5th IFIP WG 11.3 Working Conference on Database Security*, Shepherdstown, WV, U.S.A., November 1991.

[20] Jeffrey D. Ullman. *Principles of Database and Knowledge-base Systems, Volume 1*. Computer Science Press, Rockville, MD., 1988.

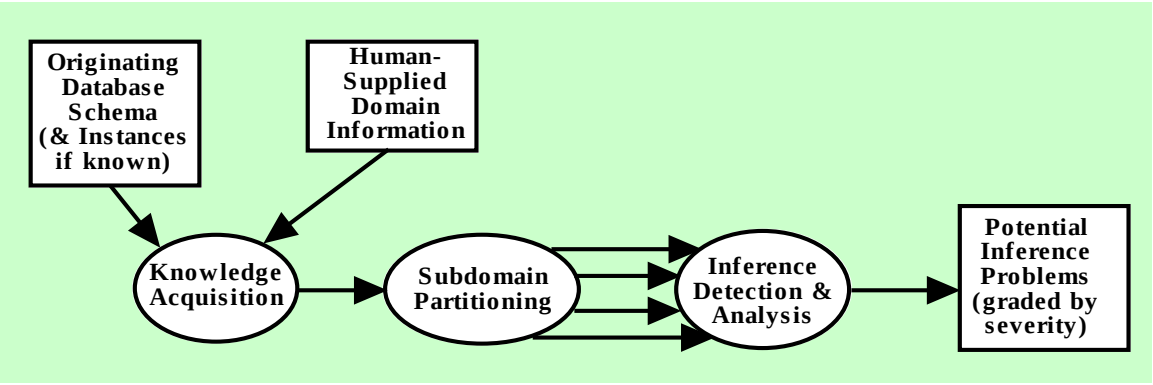

Fig. 1. Wizard subsystems.

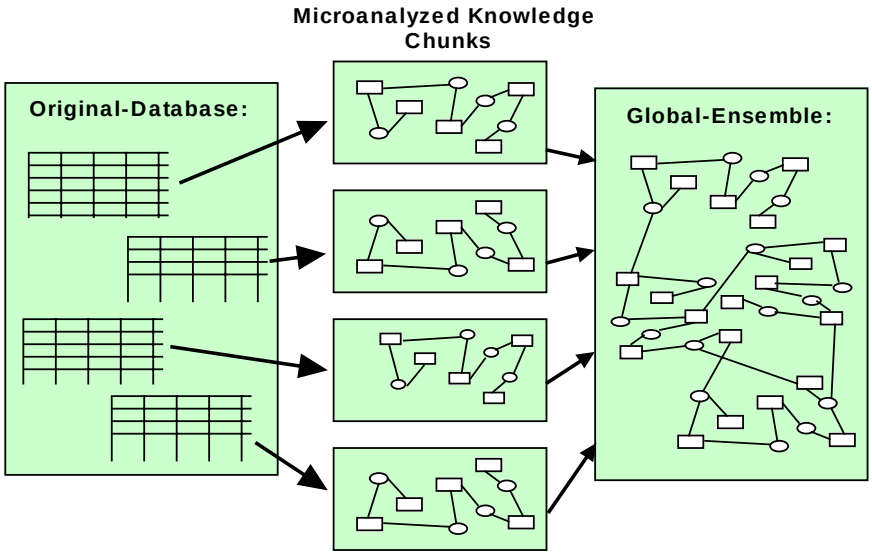

Fig. 2. Microanalyzed knowledge chunks and global ensemble.

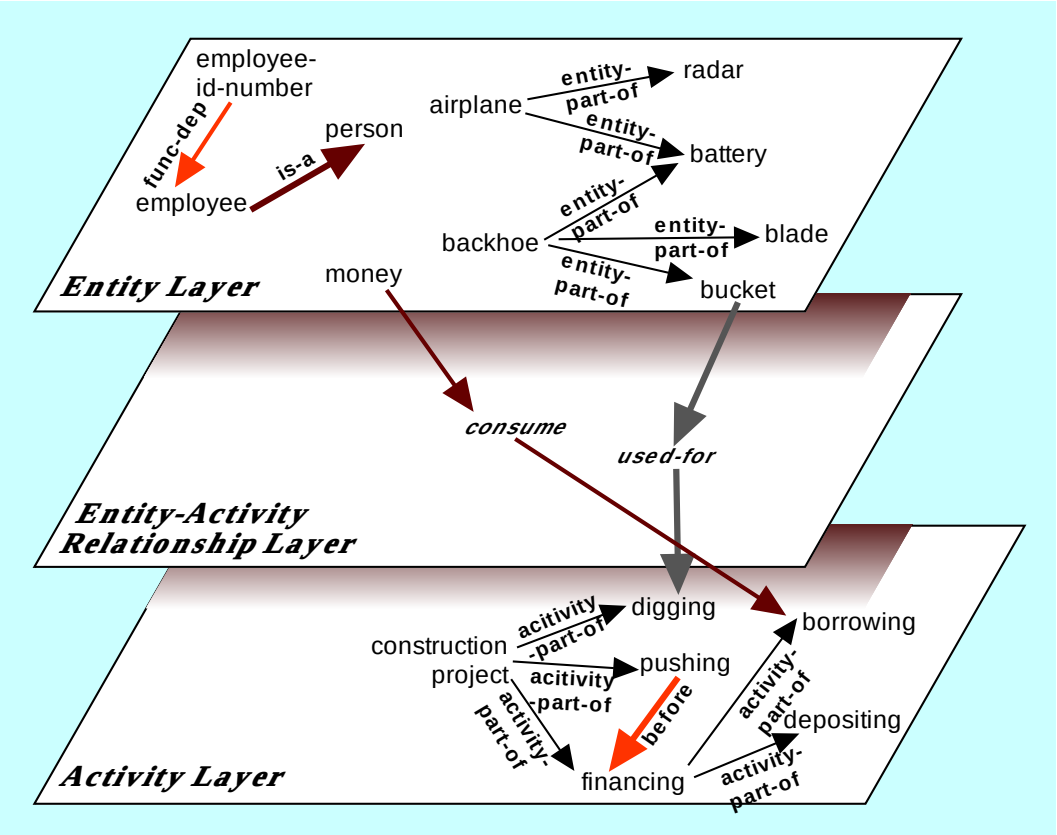

Fig. 3. Layers and their facets' constituent relations.

Characteristic	Values	Operational Definition	Processing Implications
Direction	forward	association as defined (e.g., part-of)	normal
	backward	derived reverse association (e.g., composed-of)	invert connectivity matrix
Cardinality	many	number of associations >> 1	ignored
	few	number of associations > 1 and < many	narrowed inference
	one	number of associations = 1	unique path
	empty	number of associations = 0	no path
Basis	predefined	known from constraints on info structure	deterministic
	empirical	derived from observation of instances	calculate cardinality
Transitivity	transitive	$A \rightarrow B$ and $B \rightarrow C$ implies $A \rightarrow C$	lossless join
	non-transitive		ignored

Fig. 4. Different types of associations.

```
DECIDE whether strategy = completeness-seeking or sound
FOR each facet DO
    FOR specified direction DO
        IF facet is transitive THEN
            IF facet cardinality = many AND strategy = completeness seeking THEN
                Calculate new cardinality
            ELSE
                SET new cardinality = old cardinality
            END IF
            CASE new cardinality OF
                none  :   skip to next facet
                one   :   perform endpoint detection
                few   :   assume one; perform endpoint detection
                many  :   skip to next facet
            END CASE
        END IF
    END FOR
END FOR
```

Fig. 5. Association processing.

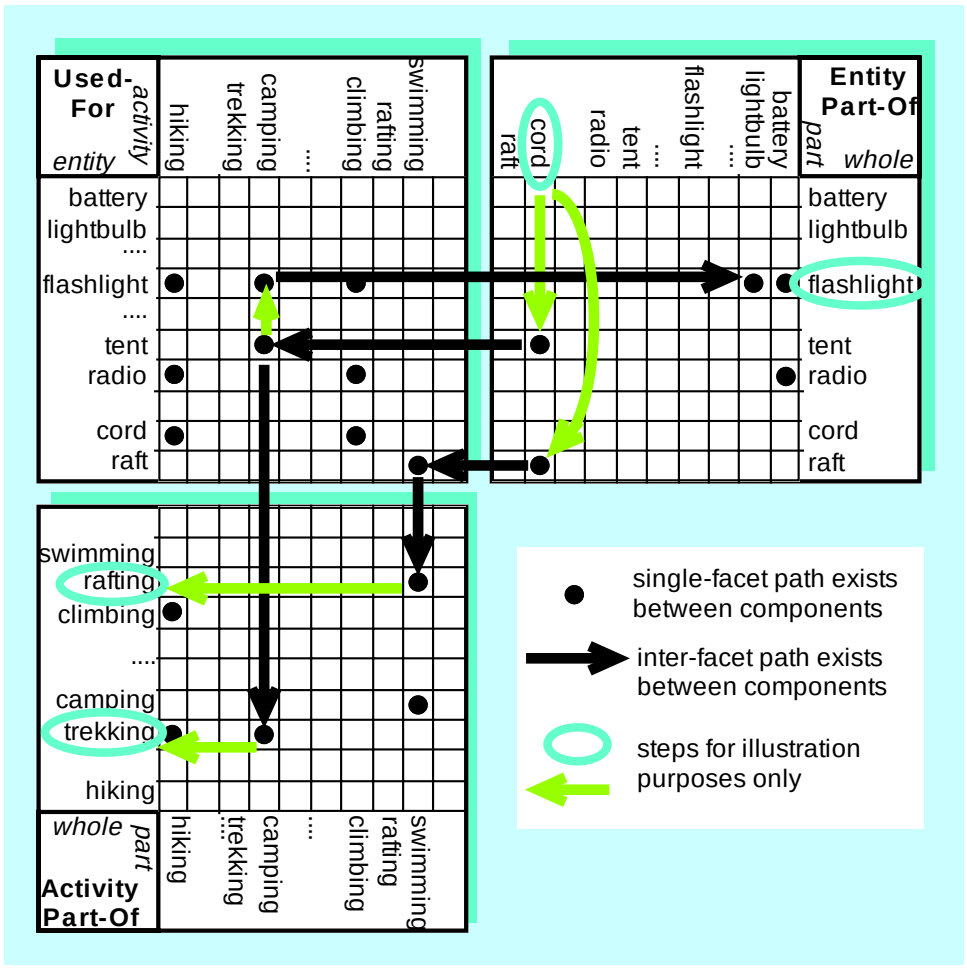

Fig. 6. Establishing inter-facet paths.

Database Tuples	Concepts	Single-Facet Paths	Multi-Facet Paths	Run-time (secs)
2233	217	983	3934	262
3394	299	1417	5370	359
5587	456	2120	6673	821

Fig. 7. Performance results in obtaining multi-facet paths

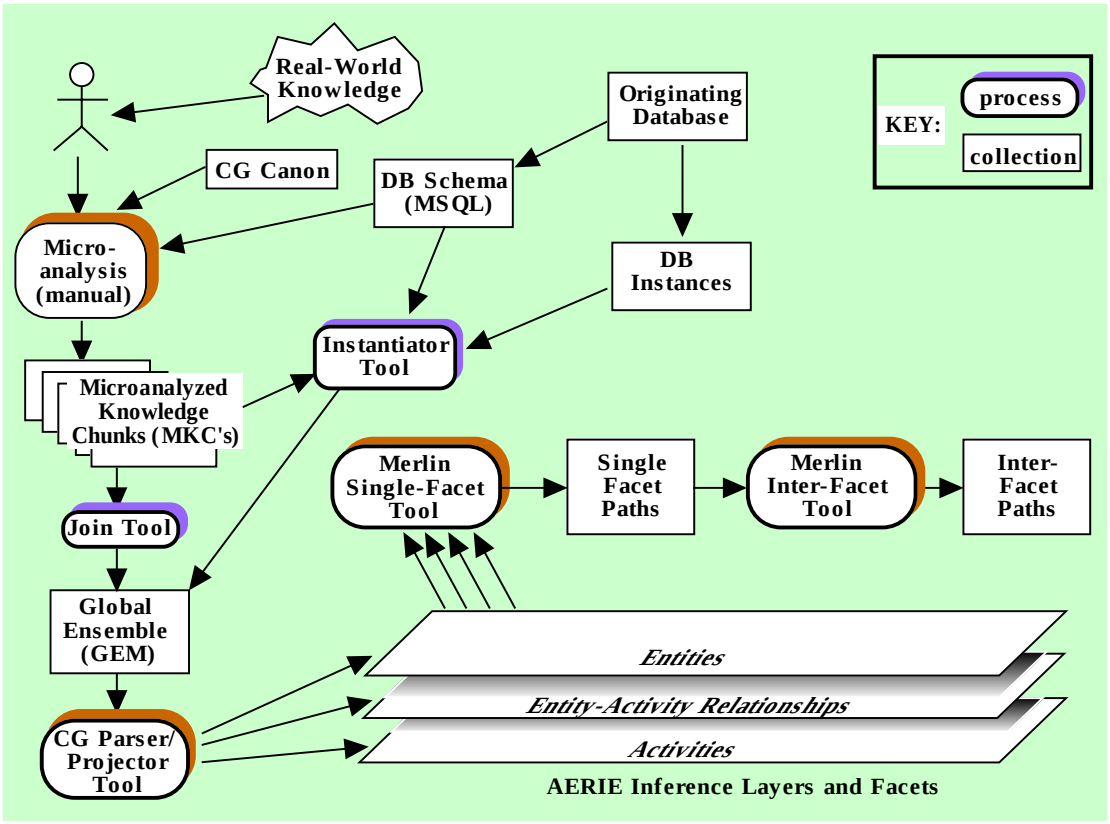

Fig. 8. Wizard system overview.