21

ILIAD: An Integrated Laboratory For Inference Analysis and Detection

Thomas H. Hinke, Harry S. Delugach and Randall P. Wolf
Computer Science Department
University of Alabama in Huntsville
Huntsville, Alabama, 35899, U.S.A.
E-mail:
thinke@cs.uah.edu
delugach@cs.uah.edu
rwolf@cs.uah.edu

Abstract

This paper describes a laboratory for exploring inference detection and analysis techniques.* This laboratory is called ILIAD (Integrated Laboratory for Inference Analysis and Detection). It has been implemented at the University of Alabama in Huntsville as part of our AERIE inference research project. The paper describes the overall architecture of the laboratory and then describes the major components: a database generation tool called Genie and an inference analysis tool called Wizard. Genie is used to generate test databases that can be analyzed by Wizard. The basis for Genie's database generation function is a simulator that provides the coherence necessary for the database to include actual inferences. Genie also supports a fact generation language called FGL, which permits the user to define a large database with only a relatively small number of FGL statements. The Wizard tool operates on data descriptions called facets, which encode the results of an inference-directed microanalysis of a database. Wizard can detect second-path inference channels within a single facet as well as inference channels that span multiple facets. The paper concludes with some observations from our initial use of ILIAD.

Keywords

Database Inference, Database Generator

*This work was supported under Maryland Procurement Office Contract No. MDA 904-92-C-5146 and MDA 904-94-C-6120.

D. L. Spooner et al. (eds.), *Database Security IX*

1 INTRODUCTION

The security-oriented database inference problem is concerned with the ability of an adversary to use accessible data from a database to deduce data that is more sensitive than the data used to perform the deduction. A number of research organizations have taken an important step in addressing this problem by developing tools to analyze the schema of a relational database. Building on the initial work of Hinke (1988, 1990), researchers within the U.S. Government (Binns, 1992, 1993), SRI International (Qian et al., 1993) and the University of Alabama in Huntsville under its AERIE Inference project (Hinke, Delugach and Chandrasekhar, 1994, 1995) have developed inference tools to perform schema-level inference analysis.

The AERIE inference research project has sought to carry inference research from the schema to the database instances, using an inference detection tool called Wizard, which is based on the Merlin second-path detection engine (Hinke, Delugach and Chandrasekhar, 1994, 1995). A significant problem that this research had to address was the source of the instances that could be used to test Wizard. The acquisition of a real database from some public or private organization was considered, but then rejected. A real database was viewed as a problem for three reasons:

1. An organization might be reluctant to divulge real data without extensive sanitization that could remove the desired inferences.
2. If an organization was willing to provide the data, it might not be willing to invest the considerable amount of time that would be needed by the AERIE team to capture the deep knowledge of the application area that instance-level inference detection would require.
3. The real data might not have the wide variety of inference problems that are needed to test the inference detection tools adequately.

In addition to these problems, there is the problem of appropriate scale. The real data might be so voluminous that the embryonic inference detection tools would be overwhelmed from the beginning. However, if the test database were too small, then – as the inference tool development proceeds – a means would have to be found to increase the size of the database in order to stress the performance of the evolving tool.

Because of the potential problems with real data, the AERIE project rejected this approach and considered the idea of a hand-generated inference database. However, this approach was also rejected because of the anticipated difficulty of ensuring that the database instances would "tell a coherent story." Inferences exist because of relationships among the data instances. These relationships exist because the data all flows from some real-world activity that ties these instances together. If the inference database is to have real inferences for testing inference detection tools, then the instances within the database must be carefully designed to provide the necessary relationships. This was viewed as a difficult task for small databases and a very difficult, if not impossible, task for the larger databases that would be required to stress-test the inference detection tools adequately as they become more mature.

For this reason an automatic inference detection tool called Genie was developed. This tool provides a scalable, relational database in which the various tuples "tell a coherent story." Genie has been integrated with the Wizard inference detection tool to provide the Integrated Laboratory for Inference Analysis and Detection (ILIAD). Just as Homer's *Iliad* describes a battle, ILIAD provides an integrated arena for a battle between the Genie inference database generation tool (which seeks to provide challenging inferences) and the Wizard inference detector (which seeks to discover these embedded inferences).

The goals of ILIAD are the following:

- To facilitate acquiring inference-relevant knowledge from a database.
- To assist database designers and administrators in determining possible inference problems in their databases.

- To characterize different groups of inferences in a well-defined manner using the tool.
- To provide a "plug-in" architecture whereby new inference detection tools can be easily incorporated into an open framework.
- To automatically populate the Wizard layers and facets using Genie-generated database instances so that inference detection strategies can be explored and compared.
- To develop techniques that can optimize the detection of inferences in the interest of improving the system's efficiency.

In the next section of this paper we will provide an overview of the ILIAD system. Section 3 describes our approach to inference analysis, beginning with our knowledge acquisition approach and then describing the Wizard inference detection system. Section 4 describes Genie and the final section presents the conclusions to date from the use of ILIAD.

2 OVERVIEW OF ILIAD

This section describes the major components of the ILIAD architecture and their general purpose within the Laboratory. As has been noted, ILIAD is comprised of Wizard and Genie.

Wizard supports the analysis and detection of certain database inference problems using conceptual graphs (Hinke and Delugach, 1992; Thuraisingham, 1991). 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. The overall architecture is shown in Fig. 1. Ovals represent processes; squares represent bodies of information. We will summarize each of the components here.

Inference Analyst. Wizard operates in support of a human inference analyst's desire to detect and prevent unwanted inferences in a database. The analyst uses real-world knowledge about the database and about the real-world domain which contains sensitive information

Real-World Knowledge. This comes from users' personal experience, consultation with others, information from reference materials, etc. We refer to this as "deep knowledge" in that inference problems may require rich knowledge sources to be detected.

Inference Targets. These are provided by the analyst. They serve as a list of sensitive information that is to be protected against unwanted inferences. These provide a "check" whereby we can compare the results of analysis and determine whether our detected inference paths in fact constitute meaningful threats to our database's security.

Microanalysis. This is the knowledge acquisition process. One or more human analysts prepares a conceptual graph for each relation's schema, using steps we have developed. This process is to be supported by automated tools that (a) manage a graphical user interface, (b) are supported by the contents of the conceptual canon and (c) perform conceptual graph consistency checks during graph construction. Microanalysis captures the inference-relevant knowledge about the database schema and instances. Its results are stored in conceptual graph form.

Inference Layers/Facets. The results of microanalysis are combined into a single conceptual graph, which we have called the GEM. The contents of the GEM are partitioned into inference-relevant domains called *layers* and further subdomains called *facets*. Each facet consists of just those concepts that are related via a prescribed conceptual relation; e.g., the "part-of" facet has only those concepts that are related through a "part-of" relation.

Single-Facet Tool. For each facet, the Single-Facet Tool called Merlin (Hinke, Delugach and Chandrasekhar, 1995) analyzes that facet according to the relations specified for that facet. The algorithms for analysis are based

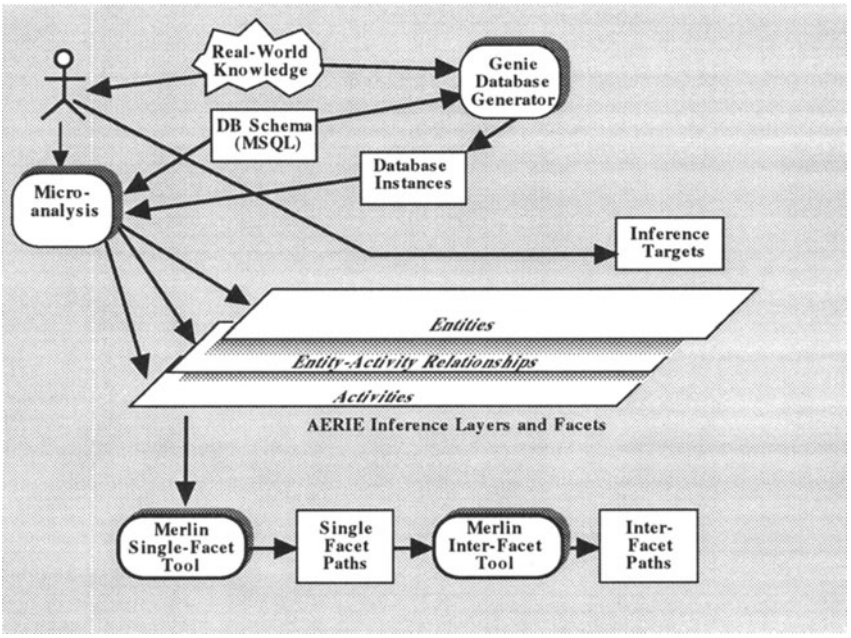

Figure 1 Overview of ILIAD Architecture

on transitive associations and are described in (Hinke, Delugach and Chandrasekhar, 1994, 1995) This tool also categorizes the results of its analysis according to their potential importance to inference problems.

Single-Facet Paths. These are produced by the Single-Facet Tool and constitute secondary inference channels (second paths) for which a possible inference vulnerability exists. The single facet paths can be summarized in a table such as Fig. 2 where each dot represents the existence of a meaningful inference path between (in this case) two entities.

Inter-Facet Tool. Once the single facet paths are obtained, our multi-facet tool uses these paths as input to a determination of multi-facet paths.

Inter-Facet Paths. These paths, along with the single facet paths, permit the analyst to determine if any of the sensitive targets are inferable from lower-classified information in the database. Our work has revealed several ways in which such potential inference problems can be solved.

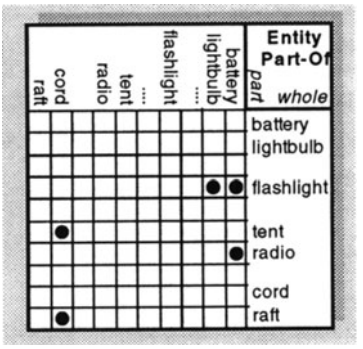

Figure 2 Single Facet Inference Paths

3 WIZARD

This section describes the Wizard tool in more detail. The Wizard tool uses the results of knowledge acquisition to build the microanalyzed knowledge chunks† that are used as the basis for inference detection (Hinke, Delugach and Chandrasekhar, 1994). The Wizard tool then uses the Merlin single-facet tool to generate the second paths within each fact. The results of this single facet analysis are then used by Wizard to detect inter-facet paths. The remainder of this section will provide more details of this process.

3.1 Knowledge Acquisition Through Microanalysis

We have developed a knowledge acquisition technique we call *microanalysis* that allows us to capture the semantics of a database with human assistance. The technique begins with a human analyst having on hand the following:

- A database schema in either SQL (Ullman, 1988) or MSQL, a form of SQL with security levels (Lunt, 1988), in which a classification level (or possibly a classification range) is specified for each attribute in each relation.
- Database instances (tuples), either from an existing database, or automatically generated by the Genie tool.
- Real-world knowledge of the domains covered by the database.
- Real-world knowledge of some sensitive targets that are to be protected.
- A conceptual canon, which is a collection of definitions and formation rules describing "common-sense" knowledge.

The database schema and instances are used in three distinct ways: *(i)* to extract 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, *(iii)* to automatically instantiate the database graphs using attribute values from the database.

†In an earlier paper these were called layered knowledge chunks.

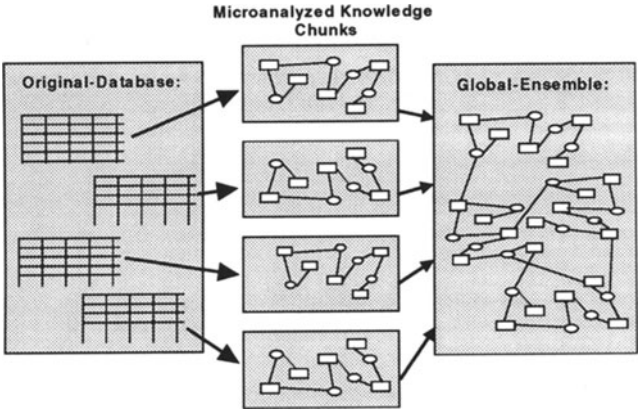

Figure 3 Microanalyzed Knowledge Chunks And Global Ensemble

Microanalysis is a human-aided process whereby each relation schema is intensively analyzed for its meaning in the context of the enterprise that is using the database. The process results in one conceptual graph for each relation. Each graph is called a *microanalyzed knowledge chunk* or MKC.

The collection of MKC's is combined through conceptual graph join operations. The Instantiator Tool creates instances of each relation's schema graph from the database instances. The resulting set of conceptual graphs are called the *global ensemble of MKC's* or GEM.

The Global Ensemble is divided into broad divisions we call *layers*: an entity layer, an activity layer, and a relationship layer between entities and activities. These three analysis layers correspond roughly to the inference classes identified in our previous work (Hinke and Delugach, 1993). In this work, entities are *things*, usually expressed by nouns in the English language. Activities are the verbs in that they describe some action or state of being. Relationships represent various ways in which entities and activities can be associated. These associations can be entity-entity, activity-activity, entity-activity or relationships between relationships themselves.

Each layer is further partitioned into subdivisions we call facets. A *facet* is a single inference domain within a layer, such as "entity composition" which comprises all the part-of relations (and the concepts they connect) within the entity layer. 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. We treat each facet as viewing the GEM through a kind of conceptual "polarizing lens" whereby only certain knowledge "passes through".

Entity Layer – Entities or relationships between entities		
	Entity Subtype (Is-A) Facet	Concepts that are subtypes of entities.
	Functional Dependency Facet.	Subgraphs which represent functional dependency relationships between two entities.
	Entity Composition Facet.	Entities that are part of another entity.
Activity Layer – Activities or relationships between activities.		
	Activity Subtype (Is-A) Facet.	The subtype relationship between activities.
	Activity Composition Facet.	Activities that are a part of another activity.
	Temporal Facet.	Information about time-dependent relationships between activities; e.g., "before", "after", "cause".
Entity-Activity Relationship Layer – Relationships between activities and entities.		
	Used-For Facet.	Entities that are used for particular activities.
	Producer/Consumer Facet.	Activities that produce or consume entities.

3.2 Wizard Inference Detection

A major focus of our work is to obtain inference paths that lead from unclassified knowledge to classified or sensitive information. There are two main categories of these paths:

Single-Facet Paths. This is a set of paths between two items in a single facet.
Inter-Facet Paths. This is a set of paths between items that lie in two or more different facets.

Our path detection techniques have been described in (Hinke and Delugach, 1993; Hinke, Delugach and Chandrasekhar, 1994). We have found that through such methods, there are many paths detected; some *bona fide* inference problems; some irrelevant for purposes of meaningful inferences. We therefore grade the detected paths by their severity as determined by the classification levels of the two endpoints of the path.

Inter-Facet Analysis We use the paths obtained from our single-facet analysis as the basis for detecting paths that involve multiple facets. The strategy used is illustrated in Figure 4.

Figure 4 shows two things; first, we can derive additional paths from components in the same facet, and second, there are some pairs of facets between which it is not possible to find inter-facet paths.

Wizard is thus 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.

4 GENIE

The general requirements that must be satisfied by an inference-oriented test database were described in the introduction. The difficult problem is to ensure that the data "tells a coherent story," such that it is seeded with inferences that can test the ability of inference detection tools. The strategy used for Genie is to structure the instance generation process around a simulator that provides the necessary coherence. The instances are then extracted as a by-product of the simulation.

The simulation that forms the basis for Genie is a microworld, involving a supplier, a manufacturing company and a customer. The customer orders parts that are created by the manufacturing company. To fill the order, the

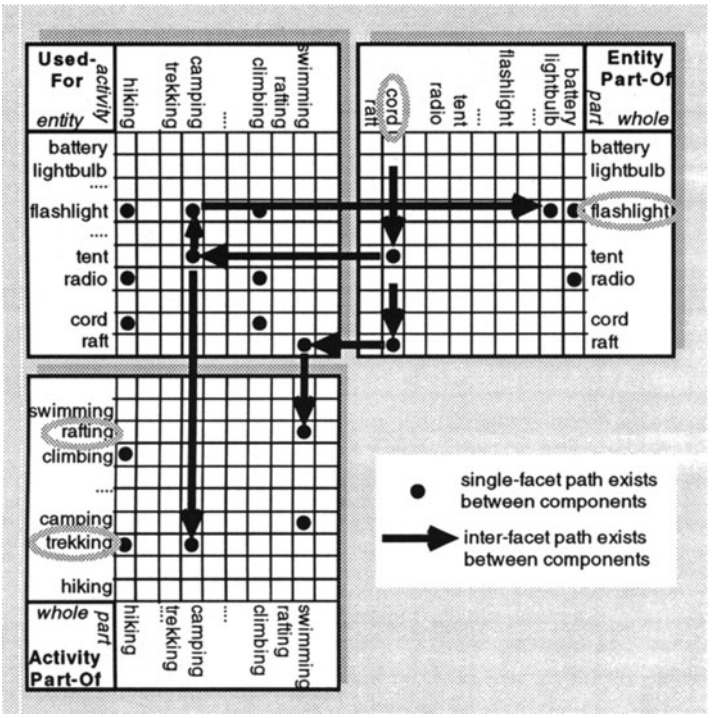

Figure 4 Inter-Facet Inference Paths

company must either pull the parts from inventory, manufacture the parts or order the parts from its supplier. If the manufacturer lacks the parts to perform the required manufacturing operation, these can be ordered from the supplier. If the manufacturer lacks the necessary personnel to perform the manufacturing operations, additional employees can be hired or existing employees can be trained to increase their productivity.

Table 1 lists the 35 relations currently generated by Genie. To provide some feeling of the size of the schema, the degree (number of attributes) for each relation is indicated. A brief description of each relation is also included to provide some feel for the application area of the current test database.

Genie generates instances through a blending of two different approaches to instance generation. The first approach is an instance compiler, called *factgen*, that processes fact and instance generation statements written in FGL, Genie's fact-generation language. The second approach is based on the extraction of facts as a by-product of the microworld simulation. For some relations, instances are generated using each of these approaches in isolation. Other relations will use a blending of these two approaches. The remainder of this section will first describe the nature of the instances that can be generated with FGL. This will be followed by a description of the simulator.

Table 1 Genie Inference Test Database

Relation Name	Degree	Description
employee	10	Provides basic employee information, as well as some information required by simulator, such as the capability to perform particular tasks.
hiring	4	Provides hiring information
person_what_activity	3	Types of activities performed by employees
training	4	Increased employee capability resulting from training
division_location	3	Division locations
person_division	4	Work assignment of employee
part_what_activity	2	Type of activity a part performs
part_makes_what	3	The parts used for making other parts
courses	3	Employee classes taken
parts	3	Parts/subparts descriptions
schedule	5	Work schedule for company activities (e.g., manufacturing)
response	5	Parts shipped to fill orders
activity_object	2	Activities and associated object target
inventory	7	Inventory or parts
produce	6	Parts produced and anticipated maintenance schedule
request	5	Part orders
language	2	Languages spoken by employees
professional_organization	2	Professional organizations of employees
country_language	2	Languages of customer countries
environment	2	Environment normally associated with activity (e.g., cold, skiing)
activity_location	2	Locations normally associated with activity (e.g., mountain, lake)
division_project	2	Company projects
project_activity	2	Project activities
activity_activity	2	Temporal relationships of activity pairs
activity_interval	2	Simulator time-tick associated with activity
activity_part	2	Parts association with normal activity
activity_person	2	Activities of employees
person_treatment	2	Medical treatment of employee
treatment_disease	2	Medical treatment association with illness
car_sticker	2	Parking sticker of employee's car
person_company	2	Employee's company of employment
part_island	2	Accident investigation results indicating types of parts found on various islands
person_part	2	Accident investigation results indicating people associated with particular types of parts (e.g., Amelia Earhart's aircraft parts)
part_serial	2	Serial numbers of types of parts associated with accident investigation
subtype	2	Subtype to supertype relationships

Finally, this section will conclude with a description of the various types of blending that are used for each of the relations that comprise the current Genie-generated database.

4.1 Fact-Generation Language FGL

The FGL language provides a means for creating domains of various types of instances, where a domain is a set of values that represent some type. Domains that can be generated include those that have the following types of values:

1. Persons;
2. Attributes, which can be used to represent any non-key attribute of a relation;
3. Part assemblies consisting of whole parts (those that are not a subpart of any other part) and their associated subparts;
4. Environments, such as hot or cold environments;
5. Activities, which are actions;
6. Objects, which are used with activities;
7. Projects;
8. Divisions, of a company; and
9. Locations, where customers or projects can be located.

Relation instances are constructed by first defining the necessary domains, and then linking them using a variety of link statements supported by FGL. The various types of link commands differ in the nature of the domains that they link. For example, the *linkperatt* statement will link a person to an attribute. Table 2 lists the various types of linkages supported by FGL and the types of domains that are linked.

To provide a flavor of how FGL provides an ability to create instances, we will consider two examples. The first will create the instances for the professional_organization relation, which is shown in Figure 3. In order to create instances for this relation, two domains must be created to represent the two fields shown in the relation. The two domains are a person domain (to represent the name field) and an attribute domain (to represent the professional_organization field). The following FGL statements create a domain of person instances:

person team
card 6
corpname focus
endperson

These statements will create six instances (*cardinality* 6) of type person and associate them with a group that is named team. The *corpname* focus statement will ensure that all of these people are assigned to the focus company, which is the manufacturing company. These statements result in the creation of the following six instances: pe0, pe1, pe2, pe3, pe4 and pe5.

We next want to create a domain of three attributes: at0, at1 and at2. These instances should be associated with a domain named proforgs. The following FGL statements will create these instances:

attribute proforgs
3
endattribute

Table 2 FGL Link Commands

Link Command Name	First Component	Second Component	Third Component
linkperatt	person	attribute	
linkattatt	attribute	attribute	
linkpartact	activity	part	
linkenv	environment	activity	
linkactobj	activity	object or part	
threelink	part	activity	part or object
prodlink	produce	part	using threelink information
schedlink	schedule	part	using threelink information
invlink	inventory	parts	using threelink information
linkcorpdiv	division	division	location
linkactloc	activity	location	
linkdivproject	division	project	
linkprojectact	activity	project	
linkactact	activity	activity	
linkactint	time	activity	
linkactpart	activity	parts	
linkactper	person	project	
namelink	"real world" names	internal names	

Table 3 Professional_organization Relation

Name	Professional_Organization
person0	ACM
person0	IEEE
person1	ACM
person1	IEEE
person2	ACM
person2	IEEE

These two domains must now be joined together to create instances of the type required by the Professional_Organization Relation. This is accomplished with the following link FGL statements:

linkperatt professional_orgs
nonrandom
nonexclusive
team

```
3
nonrandom
nonexclusive
proforgs
2
endlinkperatt
```

The FGL link statement used is linkperatt. The resulting set of instances is grouped under the name professional_orgs. This statement results in three members of the team domain's being grouped with two members of the proforgs domain. Since the *nonexclusive* option is selected, this means that domain instances can be reused. The nonrandom option, which is the only one currently supported, means that each domain instance is selected in its sort order. The result of this particular type of linkage command is the cartesian product consisting of the following instances: (pe0,at0), (pe0,at1), (pe1,at0), (pe1,at1), (pe2,at0) and (pe2,at1).

Name substitution, the FGL *namelink* statement, can be used to substitute names that have more meaning for these generated names. Thus, ACM can be substituted for at0 and IEEE for at1. Similar substitutions can be used to rename the person instances.

The second FGL example illustrates the ability of FGL to create synthetic parts with associated subparts. Synthetic parts were also used in a computer-aided-design-oriented database benchmark (Cattell, 1994). However, we are using an algorithm developed by the AERIE project that is more amenable to our inference needs. These synthetic parts explosions can be used to represent those types of inferences in which the delivery of a component part to a customer indicates that the customer also has the assembly that contains the component. The FGL statements to generate a parts assembly follows:

```
netname test
numparts 8
numdisjoint 1
numwholes 3
numarcs 13
numduplicates 3
end test
```

Figure 5 shows, in graphical form, the parts assembly that these FGL statements generate. These statements use the *netname* FGL linkage construct. The name of the parts assembly generated is called "test". In this example, the FGL statements create three whole parts, which are parts that are not components of any other parts. These are parts 0, 1 and 2 in Figure 5. The number of such whole parts is determined by the *numwholes* statement. The *numduplicates* statement indicates the maximum number of parts that can have duplicate subparts. In this case, three duplicates are to be used. In Figure 5, there are two part 4's in part 7, two part 7's in part 5 and two part 6's in part 5. The *numarcs* statement indicates that 13 arcs are to be used in generating the parts assembly. These arcs are allocated using the following approach: They are first allocated to connect all of the parts randomly. For N parts, this requires N-1 arcs. For eight parts, this will require seven arcs. Then arcs are allocated to support the duplicates. For three duplicates, this will require three arcs, or a total of 10 arcs for both part connectivity and duplicates. This leaves three additional arcs that can be randomly allocated. Note, that the initial set of seven arcs that are used to establish connectivity are also randomly allocated.

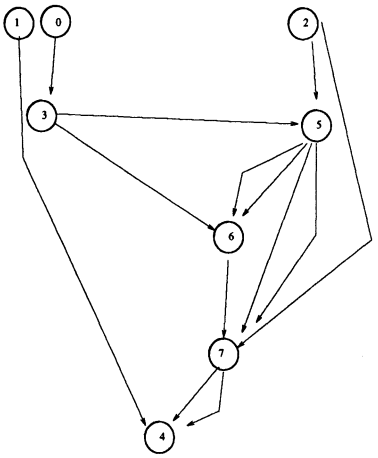

Figure 5 Factgen-Generated Parts Explosion

4.2 Simulator

The simulator is written using a production system (Round, 1989). The specific production system selected for Genie is the CLIPS production system (Giarratano, 1993; NASA, 1993, 1994). The simulator is implemented with a set of CLIPS rules that operate on facts. The initial facts are either provided by factgen or are built into the simulator. In the course of the simulation, new facts are generated that can then be used to fire new production rules.

The simulator is oriented around the customer's ordering of parts from the manufacturer. The manufacturer can fill the order from inventory, can produce parts to fill the order or can order parts from a supplier. This latter case occurs only if the company does not make the desired parts itself. The choice between supplying the parts from inventory or manufacturing them is based on part costs and availability. The least expensive parts will be selected first.

If parts are to be manufactured, the company must have the necessary equipment (represented by parts) to support the manufacturing. If the company does not have sufficient production parts these are then ordered from a supplier.

The manufacturing company also requires people to support its manufacturing functions as well as other functions of the company, such as order processing. If a particular function has insufficient personnel, the simulator has two options. If the shortfall is small, this can be covered by providing training for the existing employees. A training request by the simulator will be interpreted by the postprocessor as indicating a course, which will trigger the addition of a course to the courses relation. If the shortfall in personnel is too large, additional employees will be hired.

The next section will show how the simulator is integrated into the instance-production process.

4.3 Relation Instance Production

Genie uses seven different approaches to generate instances for the 35 relations it currently produces. These approaches range from one that relies totally upon the simulator for the production of instances, to one that relies totally on the fact-generation compiler, factgen. As will be shown, the instances for a number of the relations are generated through a collaboration between factgen and the simulator. In one case a postprocessing stage actually produces the relation. Each of these approaches will be briefly discussed. The approaches are identified in terms of the role played by the simulator and the nonsimulator software.

Sim: Produces Instances, Nonsim: No Role In this approach, the simulator is the sole source of the tuples. For this case, the postprocessor extracts the instances for each of the relations from the file generated in the course of running the simulation. The following relations are produced in this fashion: hiring, training, request and response.

Sim: Produces Instances, Nonsim: Factgen Input In this case, the fact-generation compiler provides some initial facts to the simulator. The simulator then generates instances based on the input provided. The employee relation is produced in this way; the factgen input controls the total number of employees that are hired.

Sim: Produces Instances, Nonsim: Factgen Produces Instances In this case, factgen produces instances and, in the course of the simulation, the simulator adds to them. The following relations are produced in this fashion: division_location, inventory, person_what_activity and person_division.

Sim: Uses Factgen Instances, Nonsim: Factgen Produces Instances In this case, the fact-generation compiler produces instances that are used by the simulator. These facts help drive the simulation, but the simulator does not add to these initial instances. The following relations are produced in this fashion: part_what_activity, part_makes_what, activity_object, schedule, produce and parts.

Sim: Results Involved in Instance Production, Nonsim: Postprocessing This is used for the courses relation. In this case, the training results from the simulator are used by the postprocessor to create synthetic classes. The logic is that if training occurs, then there must have been a course, so the postprocessor will create one.

Sim: Results Involved in Instance Production, Nonsim: Factgen Produces Instances In this case, factgen produces instances that involve attributes of people, such as the professional organizations to which they belong. These people instances are then mapped by the postprocessor to the people that the simulator has created in the course of hiring additional employees. In this way, characteristics of various people can be associated with people that have been involved in the simulation. However, since this association is made after the simulation has run, the simulator does not have to be cognizant of these characteristics during the simulation. In this way, new characteristics of people can be easily added without affecting the simulator. Note that the instances produced by factgen are used in conjunction with simulator results to produce the final instances, not that the simulator actually composes these relations. The following relations are produced in this fashion: language, profession_organization, activity_person, person_treatment, car_sticker, organization_person, person_company and person_part.

Sim: No Role, Nonsim: Factgen Produces Instances Some relations are completely independent of the simulator. These are produced by factgen. The following relations are produced in this fashion: country_language, environment, activity_location, division_project, project_activity, activity_activity, activity_interval, activity_part, treatment_disease, parts_island, parts_serial and subtype.

All of the instances, except those for courses, can have a name substitution applied to them. In this way, the internal names generated by Genie can be mapped to external names that have some meaning, as was shown with the ACM and IEEE mappings previously described.

5 CONCLUSIONS

The various pieces of ILIAD have only recently been assembled such that they can work together. Our initial experience is that it is performing as expected. Two observations of ILIAD are especially relevant to this paper. The first is that the initial application of Wizard to the Genie database resulted in memory problems. Our current set of workstations did not have sufficient memory for the size of database generated by Genie. However, because the size of the database could be easily tailored through Genie's FGL, the Genie database was reduced in size so that it could be processed by Wizard on one of our available workstations. This was accomplished without the loss of any desirable inferences from the Genie database.

The second observation is that the Genie database resulted in real inference channels that were detected by Wizard. However, even with the reduced size of the Genie database, the number of channels identified by Wizard was quite large. While this would be an undesirable result in a production system, it is a desirable one for the current stage of development of Wizard. We have only recently integrated Merlin into the Wizard architecture, and thus we have only a very embryonic version of Wizard. By having an instance-level database of adequate size, we are achieving results that indicate where we must spend our time. We faced a similar problem with our early Merlin work, and spent some time finding ways to organize the inference paths generated, in order to identify those that should be reviewed first (Hinke, Delugach and Chandrasekhar, 1995). With Wizard, we are finding similar challenges, and we are focusing our research to provide a means to group the paths by severity level.

From these limited, early results, we believe that the concept of an inference-database-generation tool has been validated. Thus, a significant contribution of this paper is in presenting how such a tool can be developed and integrated with an instance-level inference detection tool such as Wizard within the ILIAD.

REFERENCES

Leonard J. Binns. Inference Through Secondary Path Analysis. In *Proceedings of the Sixth IFIP 11.3 Working Conference on Database Security*. IFIP, August 1992.

Leonard J. Binns. Implementation considerations for inference detection: Intended vs. actual classification. In *Proceedings of the IFIP WG 11.3 Seventh Annual Working Conference on Database Security*, September 1993.

R. G. G. Cattell. An engineering database benchmark. In Michael Stonebraker, editor, *Readings in Database Systems*. Morgan Kaufman Publishers, Inc., 1994.

Thomas H. Hinke. Inference Aggregation Detection In Database Management Systems. In *Proceedings 1988 IEEE Symposium on Security and Privacy*, April 1988.

Thomas H. Hinke. Database Inference Engine Design Approach. In Carl E. Landwehr, editor, *Database Security II: Status and Prospects*. North-Holland, 1990.

Thomas H. Hinke and Harry S. Delugach. AERIE: Database Inference Modeling and Detection For Databases. In *Proceedings IFIP WG 11.3 Sixth Working Conference on Database Security*, August 1992.

Thomas H. Hinke and Harry S. Delugach. AERIE: Database Inference Modeling and Detection For Databases. In Bhavani M. Thuraisingham and Carl E. Landwehr, editors, *Database Security VI: Status and Prospects*. North-Holland, 1993.

Thomas H. Hinke, Harry S. Delugach, and Asha Chandrasekhar. Layered Knowledge Chunks for Database Inference. In *Proc. 7th IFIP WG 11.3 Working Conference on Database Security*, Lake Guntersville State Park Lodge, Alabama, Sept. 12-15 1993.

Thomas H. Hinke, Harry S. Delugach, and Asha Chandrasekhar. Layered Knowledge Chunks for Database Inference. In T.F. Keefe and C.E. Landwehr, editors, *Database Security VII: Status and Prospects*. North-Holland, 1994.

Thomas H. Hinke, Harry S. Delugach, and Asha Chandrasekhar. A Fast Algorithm For Detecting Second Paths in Database Inference Analysis. *Journal of Computer Security*, 1995. (Accepted).

Joseph C. Giarratano. *CLIPS User's Guide, CLIPS Version 6.0.* NASA Lyndon B. Johnson Space Center, Information Systems Directorate, Software Technology Branch, May 1993.

Teresa F. Lunt. Toward a Multilevel Relational Data Language. In *Proceedings of the Fourth IFIP Aerospace Computer Security Applications Conference*, December 1988.

NASA Lyndon B. Johnson Space Center, Information Systems Directorate, Software Technology Branch. *CLIPS Reference Manual, Volume I,] Basic Programming Guide, CLIPS Version 6.0*, June 1993.

NASA Lyndon B. Johnson Space Center, Information Systems Directorate, Software Technology Branch. *CLIPS Reference Manual, Volume II,] Advanced Programming Guide, CLIPS Version 6.0*, January 1994.

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*, May 1993.

Alfred Round. Knowledge Based Simulation: D4 Rule-driven Simulation. In Avron Barr, Paul R. Cohen, and Edward A. Feigenbaum, editors, *The Handbook of Artificial Intelligence*. Addison Wesley, 1989.

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*, November 1991.

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

Biographical Information

Thomas H. Hinke Thomas H. Hinke is an Associate Professor of Computer Science at the University of Alabama in Huntsville. He received his Ph.D. degree in Computer Science in 1991 from the University of Southern California. He joined UAH in 1990 after a 16-year career in industry, where he led research projects in computer security at System Development Corporation (now part of UNISYS) and TRW. His current research interests include database inference and data mining.

Harry S. Delugach Harry S. Delugach received a B.A. in Chemistry from Carleton College, Northfield, Minnesota in 1976, an M.S. in Computer Science from the University of Tennessee in 1982 and a Ph.D. in Computer Science from the University of Virginia in 1991. Since then he has been an Assistant Professor of Computer Science at the University of Alabama in Huntsville, where his research focuses on conceptual modeling, software requirements engineering and object-oriented software development.

Randall P. Wolf Randy Wolf is a computer professional who has worked in a variety of the subfields of computer science including databases, pattern recognition, network interactive graphics, computer security, software engineering, and image analysis. The environments in which this work has been performed have ranged from commercial to research-oriented. Upon completion of his M.S.in Computer Science in 1993, Mr. Wolf entered the doctoral program at UAH. Presently, he is working on his doctoral research in the area of knowledge acquisition via the integration of personal constructs and conceptual graphs.