

Genie: A Database Generator for Testing Inference Detection Tools *

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

Abstract

This paper describes a system called Genie, which generates databases suitable for testing inference detection tools. In order to provide the inter-relationships that must exist among data instances if the database is actually to have inferences, Genie uses a simulator to mimic "real world" activity and captures data from the simulator. Since the data is based on a simulation, it will have the necessary inter-relationships. When simulator-based data cohesiveness is not required, Genie provides a means to generate instances that are not related to the simulator. It also provides a means to associate external semantics with the data by renaming data to associate it with desired "real-world" objects and activities. The paper describes the database that is currently generated by Genie and then shows how a set of inferences that have been identified by the AERIE inference research project can be supported by the database. These inferences are organized in terms of the inference targets specified by the AERIE inference model. The paper describes a language called FGL (Fact Generation Language), which can be used to program Genie to generate various databases, including the one presented in this paper. It then presents a description of the Genie architecture. Finally, the paper concludes with observations of our experience to date in using Genie to support the development of inference detection tools.

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

1 Introduction

An organization protects data that is valuable to its mission and the release of which to one's adversaries or competitors would cause harm to the mission of the organization. This protection can be provided by a number of means. Physical protection will always be required in order to prevent the physical removal of the media on which the data is stored and the protection of the computers that process the data. Computer-based protection such as secure database management systems can provide controlled access by permitting users to access only that data to which they are authorized. Unfortunately, even if each individual unit of data is correctly protected, it is possible that the accessible data can be used to deduce data that is more sensitive than the data used in the deduction. This is called the "database inference problem".

A number of research groups have been working on the development of techniques and tools to detect whether a database is vulnerable to an inference attack. Researchers at TRW [7, 8], SRI International [16], the U.S. Department of Defense [1, 2] and the University of Alabama in Huntsville (UAH) under its AERIE inference research project [10, 11] have developed tools that can analyze a relational database schema to detect whether it is vulnerable to an inference attack. For example, at the schema-level an inference detection tool can determine whether a sequence of joins can be used to build data associations whose classification-level exceeds that of the data used in the joins.

At the schema level, inference detection tools can be tested using schemas that contain a reasonably large number of relations and have embedded within them some "unauthorized" inferences. To test its schema-level inference tool, the AERIE project has used a database schema developed by SRI International, based on information from Rome Laboratory [6]. This schema is specified in MSQL (a version of SQL developed by SRI International that permits security levels to be associated with attributes of the schema) [13]. This test schema consists of 50 relations. To support its research, the AERIE project has also developed its own test schema, consisting of approximately 30 relations drawn from the manufacturing and distribution domain. For this level of inference detector, there is no need to have actual instances, since only the names of the relations and the names of each of the attributes of the relations are required to test schema-level inference detection tools.

The AERIE project has been working to carry its inference research beyond the schema-level to the instance level. At this level, inference attacks are launched by using the actual data instances in the database to build inference chains that can be used to deduce more sensitive data. To test an instance-level inference detection tool, the test database must obviously include instances. However, in contrast to test databases that are used for benchmark testing [3, 4], an inference test database cannot be just a random collection of data. In order for a database to be useful for inference testing, it must contain real inferences that can be detected. In addition, to stress-test an inference detection tool adequately, the database should be scalable to a significant size. In this way, the size of the

test database will be similar in size to what an inference analysis tool would be expected to encounter in the real world.

There are at least two conventional options for acquiring an inference-based database: Acquire real data from some enterprise, or hand-tailor a synthetic database. There are three major problems with the first possible database source. The first is that one must find an organization that will reveal its own real data for use by a research project. It was our feeling that organizations, in general, would not want to make their real data available.¹ The second problem is that the data acquired must be sufficiently rich to contain real database inferences. The final problem is that the data acquired must be accompanied by extensive background information, such as that which would be available to an adversary who could be expected to be very knowledgeable in the domain of his attack. The approach we are using in the development of an instance-level inference detection system requires a deep level of knowledge about the domain of the attack.

The second option considered was to develop a synthetic application database, based around a manufacturing company using products with which we were familiar. Under this option, the data would be carefully hand-tailored to reflect some underlying inferences. However, inferences result from data because the data itself “tells a story” resulting from some human endeavor. This is the reason that seemingly innocuous data from one area of the database can reveal sensitive data that may be protected in another area. To hand-tailor a reasonable inference-based database of small size would be a difficult undertaking to ensure that the various parts of the database “told a coherent story”. However, to hand-tailor a large database that could stress-test an inference analysis tool would be extremely difficult to do correctly. Because of the problems inherent in both of these approaches, we rejected each as an unreasonable option for providing an inference-based database that would be useful for our research needs.

As an alternative, we embarked on an approach involving a database generator centered around a simulator. The simulator would provide the coherence that was required and the database generator could be configured to generate databases of various sizes. Initially these databases would be small in order to support an initial prototype system. The size of the database could then be increased as the inference detection technology was perfected and the focus of the research turned to performance issues. To support the deep knowledge required for our research, we structured the simulation around a common manufacturer, supplier and customer example. Many of our examples are selected from familiar activities (such as those involving outdoor recreation), although we still have some vestiges of a farm equipment manufacturer, since that provided a useful example in the initial stage of this project.

Genie is the name of this inference-based database generation tool, and its background

¹Subsequent to making our database source decision, we have found that SRI International has acquired data to accompany the schema that was previously described. However, this addresses only one of the three problems inherent in real data.

and design are the topic of this paper. The remainder of this paper will describe the types of inferences that the Genie database is designed to support, the database generated by Genie and the Genie database generation system. The next section will describe the Genie database, since this will be used to illustrate the remainder of the paper. Section 3 will then present the set of inferences that provided the motivation for the database. Section 4 will describe the overall design of Genie. Section 5 will present the conclusions from our initial experience in using Genie. The Appendix includes a more detailed description of the Genie database than that provided in Section 2.

2 Genie Database

This section will describe the Genie database by first laying out the overall simulation scenario on which the data is based. Following this we will present a summary of the relations that comprise the current Genie database. A more detailed description of this database can be found in the Appendix.

2.1 Genie Simulation Scenario

The application area selected for our test database is a relational database for a business environment. The Genie database contains information about three simulator-supported companies: a focus company, a supplier company and a customer company. The focus company is the focus of the inferencing activity; the supplier company acts to provide supplies that the focus company is responsible for delivering to the customer but which it is inherently unable to produce directly, and the customer is the company that consumes what the focus and the supplier provide. The focus company produces equipment primarily on demand. It also conducts training classes, with the goal of improving the skills of the employees as a means of furthering the production goals of the focus company. A data-flow model of the Genie simulation is shown in Figure 1.

Each simulator-supported company is composed of six different divisions: management, sales, purchasing, production, personnel and training. All six divisions are active in the focus company however sales is inactive in the customer (because the customer does not have a customer), and purchasing is inactive in the the supplier (because the supplier requires no supplier).

The purchasing department of each company is in charge of ordering needed parts from another company. The customer buys only from the focus, and the focus buys only from the supplier.² The sales department in each company is responsible for responding to orders. The focus company sells only to the customer, and the supplier sells only to

²If required, multiple additional companies could easily be supported.

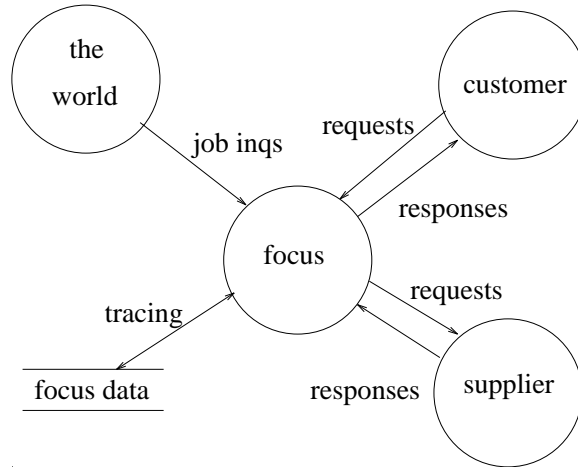


Figure 1: Data-flow Model for Genie

the focus. The production department is responsible for producing new parts. These new parts are added to an inventory of all the parts that a company has. Satisfying a parts order requires accessing this inventory of available parts to fill the present order.

The activities of departments are constrained by their preset capacity to handle the workload and by the abilities of the departments' employees. During a simulator time tick ³, the pre-set capacity of a department could be exceeded. This could occur if the sales department receives more orders than it can possibly satisfy even if it had unlimited personnel available (which it does not have). To request additional workers or to request training for its employees, the department sends a personnel request to management.

The sole function of the management department is to decide what to do with these personnel requests. If the number of personnel requested vastly exceeds the present capacity of the requesting department, management then approves the request and has the personnel department hire new employees for that department. If the number of personnel requests is relatively small compared to the capacity of the department, then management directs the training department to retrain the employees of the slightly overloaded department so they become more competent and can handle the slight overload. With the exception of three special employees, each employee works in only one department. These special employees work in all departments and are known as the focus_boss, the supplier_boss, and the customer_boss. At simulator start-up, these are the only employees. The demand for parts causes other, normal employees to be hired.

³The smallest period of time during which activity is recorded

2.2 Genie Relations

The names and a short description of the relations currently generated by Genie are shown in Table 1. More detail on each of these relations is presented in the Appendix.

3 Inference Targets For Database

In [9] we presented an inference model called AERIE (Activities, Entities, Relationship Inference Effects), which was an extension of Chen’s ER-model [5]. The AERIE model consists of entities, which (as in the ER-model) are just *things* that have a distinguishable identity. Activities represent actions that could be undertaken by the entities or that could involve entities. Relationships represent the association of entities with other entities, activities with other activities or entities with activities. Another type of relationship is the rule, which represents some required relationship among entities, activities or relationships.

The AERIE model can be used to represent the potential inference targets of an adversary. Various types of inference targets were identified during the first phase of the AERIE project. These inference targets have been classified into various target classes using the AERIE model. The database generated by Genie will support a subset of these example inferences. The example inferences and how they can be reflected within the Genie database are described in this section. For completeness, we have also included some example inferences that are not currently supported.

For each inference, the Genie relations that support it are indicated. Each inference is labeled with one or more letters that indicate its inference target class and a number that is used to designate the various members of each particular inference target class. Thus the first inference of the *entity* target class is labeled “E1” and the second inference is labeled “E2”. The first inference of the relationship between entity and entity target class is labeled “REE1”. A similar notation is used throughout the inference list.

1. Inference Target Is an Entity

- (a) E1 – There Exists a Cotton Picker. Either the *Request* or *Response* relations, in conjunction with the *Parts* relation, can be used to support this inference. If a company requests a spindle that is a part used only in a cotton picker, then it can be assumed that a cotton picker exists at the requesting company.
- (b) E2 – Infer Growing Season from Sales Figures. Monthly sales figures can enable one to infer what a particular region’s growing season is, assuming that there are certain items that are purchased during a growing season. The *Request* relation, in conjunction with the *Divloc* relation, support this inference.

Genie Database Relations	
Relation Name	Description
Employee	Employee information
Hiring	Employee hiring information
Personwhatactivity	Activities performed by employees
Training	Results of employee training
Divisionlocation	Physical location of company divisions
Persondivision	Employee work assignment
Partwhatactivity	Activity performed by part
Partmakeswhat	Parts used to make other parts
Courses	Courses taken by employees
Parts	Parts explosion data
Schedule	Shows scheduled activities for companies
Response	Parts shipment in response to order
Activityobject	Object acted upon by activity
Inventory	Parts inventory
Produce	Parts production capacity
Request	Requested orders for parts
Language	Language skills of employees
Professionalorganization	Employee's professional organizations
Countrylanguage	Languages spoken in countries
Environment	Environmental factors associated with activity
Activitylocation	Locations associated with activity
Divisionproject	Projects supported by companies
Projectactivity	Activity associated with project
Activityactivity	Temporal relationship between activities
Activityinterval	Length of time taken for activity
Activitypart	Association of parts with activity
Activityperson	Association of employees with activity
Personreatment	Employee medical treatment
Treatmentdisease	Association of medical treatment with illness
Carsticker	Employee car sticker data
Personcompany	Employee company association
Partisland	Parts found on island data
Personpart	Parts associated with missing people
Partserial	Serial number of parts
Subtype	Subtype to supertype relationships

Table 1: Relations in Genie Inference-test Database

- (c) E3 – Months with Increased Sales of Planting Equipment Implies Start of Growing Season, and Months with Increased Sales of Harvesting Equipment Implies End of Growing Season. This inference is supported by the *Request* and *Divloc* relations in the same fashion that inference E2 is supported. The *Partwhatactivity* relation would indicate the use of the ordered equipment for either planting or harvesting.

2. Inference Target Is an Activity

- (a) A1 – A Winter Mountain-Climbing Expedition is About to Occur. This inference is based on the ordering of equipment that is relevant to mountain climbing. The *Request* relation would be used to determine that equipment with certain properties has been ordered. The *Partwhatactivity* relation can be used to determine that an ice axe is used for mountain climbing, and that sleeping bags are for camping. In particular, it is possible to define an activity of “sleeping-in-the-cold”. The *Environment* relation could be used indicate that “sleeping-in-the-cold” is associated with a cold environment. The various *Project*-related relations could be used to define a winter mountain-climbing expedition.
- (b) A2 – The Activity of Cotton Picking Exists. The fact that a spindle is being ordered is assumed to imply the existence of a cotton picker. The *Request*, *Parts* and *Partmakeswhat* relations would be used to indicate the above prerequisite facts.
- (c) A3 – A Construction Project Activity Exists. A construction project could consist of the following activity sequences: Financing occurs before digging, and digging occurs before framing. If we note that financing occurs and framing occurs, then we can infer that digging has occurred, and that a construction project is occurring. The *Divisionproject*, *Projectactivity* and *Activityactivity* relations will be used to support this inference. The *Divisionproject* relation will indicate which company is performing the project. The *Projectactivity* relation links together the activities that will occur during the project. The *Activityactivity* relation orders pairs of activities in time. This allows the fact that financing occurs before digging and the fact that digging occurs before framing to be entered into the system. It is possible to create the *Schedule* relation instances that show that financing and digging are scheduled to occur.
- (d) A4 – An Object’s Physical Movement. The facts upon which the inference are based is: object A is at location A, and object A is moved to location B. The *Inventory* relation provides this information. The natural activity of the simulator is to cause parts to be ordered, produced and delivered. This causes the inventories of the companies to change during the simulation.
- (e) A5 – Time of Situation is After Action A. The target is time of situation is after action A. The given is action A causes situation A. This inference is an

example of a causality-type inference in the general area of temporal inferences. The *Activityobject* and *Partmakeswhat* relations support this inference. For example, one must pick cotton in order for cotton to be available. In this case the action of picking must occur prior to the situation in which cotton that has been picked exists.

3. Inference Target Is a Relationship Between Entity and Entity

- (a) REE1 – Organization Working on a Project. This inference is based on the association of a person working for an organization with some aspect of a project. The fact that a part of an organization is associated with a project implies that the organization as a whole is associated with the project. This inference is supported by the *Persondivision*, *Divisionproject*, *Projectactivity* and *Activityperson* relations. the *Persondivision* relation indicates that an individual works for a particular division of a particular company. The *Activityperson* relation would indicate that this same individual is working on a certain activity. The *Projectactivity* relation would show that the activity is part of a project. The *Divisionproject* relation would indicate that the project is being worked on by a company different than the one for which the specified individual works.
- (b) REE2 – Person Has a Disease. The inference connects person to disease, based on the association of some characteristic of the disease (namely a treatment) with the person. Treatment for the disease applied to the person implies that the person has the disease. The relations *Persontreatment* and *Treatmentdisease* support this inference. The fact that a person is receiving a treatment and that treatment is associated with the disease delineates the inference.
- (c) REE3 – Site Has Equipment. This inference is based on connecting some part associated with another part to the site. Movement of the subpart to the site implies that the whole good exists at the site. This inference is supported by the *Parts* and *Request* relations. The fact that a company requests a part that is unique to another, larger part establishes this inference.
- (d) REE4 – Organization's Area of Operation. If an organization orders cross-country skis, then they are planning on operating in the snow. From a part and the normal environment of use for a particular part, we can infer the probable environment of use. The relations *Environment*, *Partwhatactivity* and *Request* support this inference. If a company orders cross-country skis, if it is known that skis are for skiing, and that skiing occurs in the cold, then the inference can be made.
- (e) REE5 – Any Relationship Between a Company and a Project. An example would be a relationship between company Alpha and Project AERIE. Assume that Susan works at company Alpha, and Mike works at Company Gamma. Mike escorts visitor Susan and Mike works on Project AERIE. Therefore, it can be

inferred that Susan’s company Alpha is possibly supporting project AERIE (it is also possible that that she is meeting him for lunch). This inference is supported in a fashion similar to the way that inference REE1 is supported. The relations needed are *Persondivision*, *Divisionproject*, *Projectactivity* and *Activityperson*. The *Persondivision* relation is used to place Susan and Mike at divisions in their respective companies. The *Divisionproject* relation is used to indicate that Mike is working on the project in question. The *Projectactivity* relation is used to create an “escort” activity that is part of the project. The *Activityperson* relation is used to place both Mike and Susan in the escort activity.

- (f) REE6 – Infer a Relationship Between a Company and a Project. This is similar to inference REE5, except the connection between the AERIE project and company Alpha is derived based on Susan attending an AERIE meeting. The setup for this inference is very similar to REE5, except that the *Activityperson* relation is used to place Susan in a “meeting” activity created in the *Projectactivity* relation.
- (g) REE7 – Mission Is Going to Switzerland. This inference is based on the fact that 50 percent of the mission personnel speak Romansch. This inference is supported by the *Language*, *Countrylanguage*, *Projectactivity* and *Activityperson* relations. Projects and people can be associated using the relations *Projectactivity* and *Activityperson*. Projects can be associated with languages, based the languages spoken by people. The languages, if they are very distinctive, can then provide an indication of the target of the project using the *Language* and *Countrylanguage* relations which indicate which employees speak languages used in certain nations.
- (h) REE8 – Company Parking Stickers on Visitor’s Car. This inference is based on a person with a company parking sticker for company X visiting a site of project Y. The inference target is a relationship between company X and project Y. The relations used would be *Projectactivity*, *Activityperson* and *Carsticker*. The *Projectactivity* relation would establish a “visiting” activity, and the *Activityperson* relation would establish the visitor as engaging in a visiting activity. The *Carsticker* relation would identify the corporate sticker on the visitor’s car.
- (i) REE9 – Amelia Earhart’s Crash Site Based on the Type of Part Found. The inference target is the island on which the famous flier, Amelia Earhart, crashed. This inference will use the information that a part was discovered on an island and the part is of a type used on her aircraft. This inference is supported by the *Parts*, *Partisland* and *Personpart* relations. *Parts* indicate which part is composed of which other parts, which links the discovered part to the plane. *Partisland* is a relation that links people to the island. *Personpart* is a relation that links people to parts.

- (j) REE10 – Amelia Earhart’s Crash Site Based on the Serial Number of the Part Found. The inference target is the same as for inference REE10. The only difference is that now a part with a serial number that was known to have been used on her aircraft was found. The relations needed are *Partserial*, *Partisland* and *Personpart*. The *Partserial* relation is used to link a serial number to a type of part. This allows the subpart to be linked to a single, specific part. It is also used to link that same serial number to a whole good (the plane). The *Partisland* relation shows that the part is on the island, and the *Personpart* relation places Amelia on the plane.
- (k) REE11 – Battery Is a Part of a Radio. The target is, a battery is part of a radio. The givens are, BP205 part of BC100XLT, BP205 subtype of Battery and BC100XLT subtype of Radio. This relation is supported by the *Parts* and *Subtype* relations. The *Parts* relation would show that BP205 is a subpart of BC100XLT. The *Subtype* relation would show that BP205 is a subtype of Battery, and BC100XLT is a subtype of Radio.

4. Inference Target is a Relationship Between Entity and Activity

- (a) REA1 – Company Orders Unusual Chemical, Which Indicates That the Company Is Adopting a New Process. The target is, company is using a new process. This is a relationship between a company and a process. This inference is supported by the relations *Request* and *Partwhatactivity*. The filling of an order for an unusual part (chemical) and the association of that part with the process (activity) in question completes the inference.

5. Inference Target is a Relationship Between Activity and Activity

- (a) RAA1 (NOT SUPPORTED) – California Example. One can infer that more people are leaving California than are moving to California by reviewing the rental rates for truck rental companies. To support this inference, the rate for renting trucks out of California will be higher than the rate to rent trucks into California, because of the difference in demand.

6. Inference Target is a Relationship Between Relationships

- (a) RERE1 – Infer Student Grade. A grade is an attribute of a student. Infer this attribute by associating a list of grade attributes of anonymous students with a listing of specified students, where both lists have the same sort order. The *Courses* relation is necessary for this inference. This relation provides a list of courses, students taking the courses and their grades. Accessing by primary key will provide the ordering implied for this inference.

7. Inference Target Is a Rule

- (a) RU1 (NOT SUPPORTED) – Infer the Rule that a Credit Card Company Will Approve All Charges for Meals. This inference is based upon not wanting to embarrass customer for food already consumed.

The current datagen file that has been developed for Genie generates the relations required to support all of these inference targets, with the exception of RAA1 and RU1. Having described the Genie database and the set of inferences that can be supported, the next section will provide an overview of the Genie design.

4 Genie System

The Genie system has three components: fact generator, simulator and postprocessor. The fact generation component is used to create the initial facts required by the simulator. These facts represent the initial state of the company, supplier and customer, in terms of such factors as employees, parts which can be used to manufacture other parts and inventory. It also creates facts, which will result in instances, that are needed in the final database, but are not affected by the simulator. The simulator, written in the CLIPS (C Language Integrated Production System) production system [12, 14, 15] simulates a business environment and records data resulting from the simulation in an archival file. The postprocessing component transforms data supplied by the simulator, as well as data supplied by the fact generation, into a number of files whose format is suitable for ready importation into a standard relational database. The following sections will describe each of the Genie components.

4.1 Fact Generator

The purpose of the fact generation component (factgen) is to create facts (e.g. objects o1, o2, o3 exists), which will be used to create relations and instances for these relations. Factgen is controlled by the factgen input file, which is called *datagen*. Datagen includes various commands written in Genie's fact-generation language (FGL). These language statements abstractly define the initial facts used by the simulator. A simplified usage of factgen generates an initial state of the simulator that provides employees and various activities (such as those involved in orders for products or training of employees). This usage requires only that the user select the quantities for four categories of data. The following is a datagen file that would generate a usable set of initial facts:

```
poolsize 30  
overall_population 10  
desired_training 10  
desired_orders 10
```

The *poolsize* statement generates a pool of randomly generated people. In this example, we are generating a pool of 30 people. The *overall_population* statement indicates the initial number of people that are to be assigned to the focus company. Also, since the customer company must have a similar set of people, this statement also indicates the initial number of people that will be assigned to the customer company as well. In this example, 10 people will be assigned to work for the focus and customer companies. The *desired_training* statement indicates that 10 people will be trained initially. In the course of the simulation, additional people may also be trained. This will be triggered if the number of orders slightly exceeds the current production capacity. In this case, this slight overage will be handled by training, rather than hiring. The *desired_orders* statement indicates that 10 orders will be processed by the simulator. These 10 orders may generate hiring or training activity in the course of the simulation. The simulation activity will result in the production of instances for the database. Section 4.3 will show the various sources of instances in the final database, and indicate those that flow directly from the simulation.

The second method for achieving instance generation is to create instances from various attribute domains. Once attribute-domain instances are created, then these instances may be linked together with instances from other domains to form more complex instances. For example, the target database is to have a relation *Language*. This relation indicates the languages spoken by various employees. To be able to create this relation, it is necessary first to create instances for the two domains that comprise this relation – person and language. Then these domains need to be composed into instances that will form the instances of the language relation.

In order to create a number of instances of people in the *person* domain, we can use the following FGL statements:

```
person team  
card 6  
corpname focus  
endperson
```

The general syntax for creating domain instances is that the first line will contain the type of the domain instances that are being created and a name to identify this particular group of instances. In this example, the keyword *person* indicates that domain instances of type person are being created. This keyword is followed by the word *team* which will be used to identify this group of person instances. The second line indicates the cardinality of the set of instances created. In this particular case six instances are being created. Note that for some of the FGL domain types, the keyword *card* is no longer required. The next line uses the keyword *corpname* to indicate to which company (focus, supplier or customer) these people are to be associated. In this case, they are to be associated with the focus company. These FGL domain-creation statements are ended with a terminating keyword that is keyed to the particular domain being created. In this case, since it is instances of type

person that are being created, the terminating keyword is *endperson*. This set of statements will create the domain instances pe0, pe1, pe2, pe3, pe4 and pe5 that are assigned to the *team* domain.

The next two sets of FGL statements create two domains of attributes that are used to represent various languages. In the first set of statements, the keyword *attribute* indicates that instances from the *attribute* domain are being created. The second word *languages* indicates that the created instances are to be associated with a group called *languages*. The next line, indicates the cardinality of the set of instances being created, which in this case is one. Note that these statements do not require the keyword *card*. This set is terminated with the keyword *endattribute*. The first group (*languages*) will consist of a single language, *at0*. It uses the *at* prefix since we are using the *attribute* domain-type to represent languages. The second group of languages, called *languages2*, will also have a single language. Since this language also uses the attribute construct, it will be named *at1*, since *at1* is the next sequential attribute that follows *at0*, that was created for the languages group.

```
attribute languages  
1  
endattribute
```

```
attribute languages2  
1  
endattribute
```

The following statement uses the first three instances from *team* and associates them in a nonexclusive way with the one instance from *languages*. This statement will result in the following associations: (pe0, at0), (pe1, at0) and (pe2, at0).

```
linkperatt language  
nonrandom  
exclusive  
team  
3  
nonrandom  
nonexclusive  
languages  
1  
endlinkperatt
```

As with the previous example, this set of FGL statements begins with a keyword indicating the type of statements that are to follow. In this case the keyword *linkperatt*

indicates a linkage between persons and attributes. The resulting group formed by this linkage is called *language*. The *nonrandom* term means that the attributes are assigned in their sort order. Currently only *nonrandom* is supported. The *nonexclusive* term means that the attribute does not have to be assigned exclusively to a single person instance, and in this case at0 is associated with each of the first three people in team. The name *team* followed by the number 3 indicates that the first three members of the team group are to participate. The name *languages* followed by the number 1 indicates that only a single member of the group *languages* (which is the only member of this group) is to participate in the linkage.

The next set of FGL statements associates the next three members of the *team* group with an instance from *languages2* group. This will result in the following people, attribute pairs: (pe3, at1), (pe4, at1) and (pe5, at1).

```
linkperatt language  
nonrandom  
exclusive  
team  
3  
nonrandom  
nonexclusive  
languages2  
1  
endlinkperatt
```

In order to provide instances that have meaning in the “real world,” such that real-world semantics can be associated with them, Genie provides a name substitution capability. The following FGL statements establish a group called *real_language*, which consists of the single language of Romansch.

```
stringname real_language  
Romansch  
endstringname
```

The following FGL statements will perform name substitution, using the FGL *namelink* statement, on the language instance in *languages2*, to associate the Romansch language with the single instance of *languages2*.

```
namelink real_language  
nonrandom  
nonexclusive
```

```
1
attribute languages2
nonrandom
1
endnamelink
```

This will replace attribute value at1 with the string “Romansch” wherever it occurs within any of the instances.

Rules used by the simulator require that certain facts be created by factgen. One such fact is *whatactivityobj*, which associates an activity with an object. To be able to provide this *whatactivityobj* fact, it is necessary first to create instances for the two domains – activity and object – that comprise this fact, and then secondly, to compose these domain instances in order to create activity-object facts that will satisfy *whatactivityobj*. To create three domain instances that belong to the activity domain, factgen needs to be provided with the following FGL statements:

```
activityname winterexpacts
numactivities 3
end winterexpacts
```

This will create the three activities a0, a1 and a2, which are assigned to the *winterexpacts* (winter expedition activities) domain. This domain could reflect the types of activities that are associated with a winter expedition. The symbols used to represent these domain instances will be unique and sequentially numbered. They need to be unique for the same reason that a primary key needs to be unique. They need to be representative, (that is, a1 to stand for activity 1), because they need to attain a minimal degree of readability. They need to be sequentially numbered in order to aid in creating uniqueness and readability. Scaleability is provided, because the three in the above example could easily be replaced by a much higher number. Note that at this stage, the instances created do not have meaningful names. In order to provide meaningful names, the FGL renaming feature can be used.

The creation of a set of objects that belongs to the *object* domain is performed by the following FGL statements:

```
objname winterobjs
numobjs 3
end winterobjs
```

These statements would create the three objects o0, o1 and o2 and assign them to an object domain called *winterobjs*. These could be a group of objects associated with winter activities. This linkage of winter expedition activities with winter objects is accomplished with the following FGL *link* statement:

```
linkactobj winterexpects to winterobjs  
onetoone  
endlinkactobj
```

These object-activity pairs can then be used to satisfy the object-activity conditions of the simulator. This would create the following instances (a0,o0), (a1,o1) and (a2,o2). These instances would appear both as facts (in CLIPS fact syntax) to be seen by the simulator and as instances (in a format suitable for use by a database loader) in the test database.

This paper has illustrated four of the domains supported by Genie: persons, attributes, objects and activities. These are just some of the domains that are supported by Genie; the following is the complete list:

Persons: defines groups of persons without any accompanying information.

Attributes: defines groups of attributes.

Netnames: defines groups of parts. A netname group is more sophisticated than an attribute group definition or a person group definition, in that a netname group links the parts it just generated into a controlled has-part structure. This is an example of a sequential group that has more associated semantics than others.

Activitynames: defines groups of activities.

Environment: defines groups of environments.

Objnames: defines groups of objects that are the direct objects of activities.

Project: defines groups of projects.

Division: defines groups of company divisions.

Location: defines groups of locations.

The *netname* group is somewhat different from the groups that were previously illustrated in this paper, since it creates a set of parts that are related to each other by the subpart relationships, and, in a sense, provides for synthetic parts explosions. This follows similar ideas for parts explosions provided in [4]. The following set of FGL statements will create a parts explosion, which is illustrated in Figure 2:

```
netname test  
numparts 8  
numdisjoint 1  
numwholes 3
```

```

numarcs 13
numduplicates 3
end test

```

The name of this group is *test*. It consists of one set of connected parts. If multiple, disjoint sets of connected parts are required, then the *numdisjoint* option can be set to a number that is two or more. In this example, the total number of distinct parts is eight, of which three are whole parts. A whole part is not a subpart of any other part. The number of arcs is a set to be 13. Note that each connected group of parts must form a directed, acyclic graph; thus the number of arcs must be at least one less than the number of parts. The number of duplicate parts is set to three in this example. This means that three randomly selected parts will be duplicated in the assembly. Note that each duplicate part will require an additional arc. If, as in this case, the number of arcs exceeds the minimum required to support connectivity and duplicate parts, these will be randomly assigned.

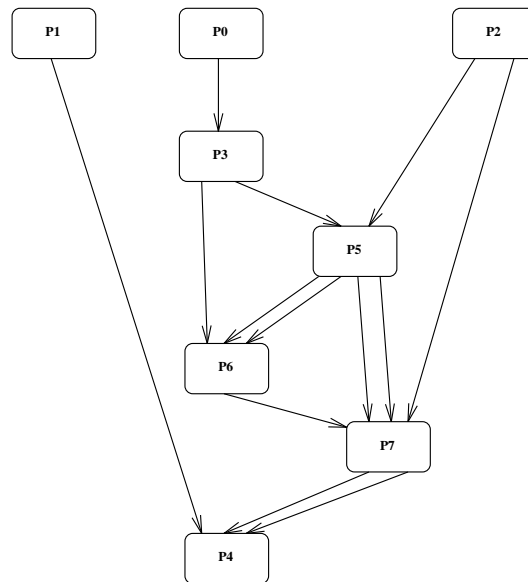


Figure 2: Factgen-Generated Parts Explosion

Genie supports both sequential and nonsequential groups. A sequential group consists of an ordered list of entries of the same type. For instance, a netname group consists of parts that are ordered. A particular group might have five entries, and the entries might be p0, p1, p2, p3 and p4. The ordering is indicated by the numeric suffix of the name, the “0” in p0; and the type of the entry is indicated by the prefix, the “p” in p0. Factgen keeps track of what the next entry to be created should be. If the previous parts group declaration had created parts numbered through p4, then factgen would know that p5 is the next part that should be generated if it encounters a new *netname* definition. Factgen retains this knowledge about every type of sequential group. The following groups listed with their

associated prefix are sequential: person(pe), attribute(at), netname(p), activityname(a), environment(e), objname(o), project(pr), division(d) and location(l).

The non-sequential groups are the following:

Produce: Defines company-independent information about part-makers. This type of group sets the production rate of part-makers.

Schedule: Defines a series of quantities that will be used to define a production schedule.

Inv: Associates a quantity with an inventory price, a production price and a purchase price.

Interval: Defines a time value.

Stringname: Defines a list of strings to be used later during name substitution operations.

These groups do not create a series of entries of the same type. Each occurrence of one of these groups creates a named group that has a set number of attributes and initializes these attributes to values specified in the group definition.

The following seven lines define a produce group, called produce1:

```
produce produce1  
make 3  
per 1  
maint 1  
tmaint 36  
life 210  
end
```

This group does not consist of multiple domain instances. It consists of a single domain instance that defines how rapidly a given part can produce something. The “make 3” term indicates that the part-makers associated with this statement make three parts during each time period in the simulation. These parts take one time period to perform maintenance on the created parts, and the time between required maintenance periods is 36 periods of simulation time. The parts that are made have a lifetime of 210 time ticks of the simulator. The name of the produce group is used to allow reference to this production definition. Scalability is provided, because if many parts share these production characteristics, then those parts need to reference only this definition.

As has been noted in the examples, sequential and non-sequential statements are operated upon by linkage statements, in order to create facts used by the simulator and instances that comprise relations. Figure 3 shows the linkages created between groups.

4.2 Simulator

The Genie simulator is implemented as a production-system-based simulator[17]. It is implemented using the CLIPS production system. In order to understand how the simulator operates, it is first necessary to understand how a production system operates, and then to understand the nature of the facts and rules making up the Genie CLIPS-based simulator.

A production system consists of facts and rules. The production system is in an infinite loop, looking for facts that match the pattern of a rule. The form of a rule is pattern \Rightarrow actions. The pattern is defined as a list of patterns that must be matched by facts in the factbase before the rule will fire. When a rule fires, the actions associated with the pattern are performed.

The facts in the simulator are partitioned into facts that deal with only one of the three companies and facts that are general in nature and apply to all three companies. Each company has particular inventory. That is, it owns a certain quantity of specific parts. Some parts can be used to generate other parts and objects.

Each company has a number of production schedules that specify the quantity of parts and objects to be produced during each time period. The fact that a company has a production schedule to produce a set quantity of a type of part at a certain time does not mean that it will actually be able to do so, since it may be cheaper to order parts from a supplier or retrieve the parts from inventory. It may also be the case that the company is unable to meet the required production quantity. In this case, it must take action to be able to produce the desired quantity of parts. This may require that employees be trained to make them more productive or that employees be hired to increase the production capability of the organization.

A more detailed description of the chain of rule firings that lead to a request, starts with the examination of the production schedules for a company. If the production quantity for the present time tick is nonnegative, then a check is made of the number of part-makers for the part/object. If part0 makes part1's, then in order to make a part1 the company must have a part0. For example, in the real world, a corporation may be in the business of making nails. This corporation would need a machine or group of machines that would make nails. In the simplified Genie simulation, nails (part1's) would be made by a nail-making machine (part0). If the partmaking capacity of the number of part-makers in the company exceeds the scheduled production quantity, then only the scheduled quantity is produced.

If there are some part-makers, but their combined capacity does not meet the scheduled production quantity, then the part-makers make the maximum amount that they can and a request is generated to the company's source of the part-makers for the number of part-makers necessary to meet the demand. If the company has none of the required type of part-maker, then the only action that is taken is for the company to order the required number of part-makers from its supplier.

When a company receives a request, it checks the relation among the various prices for the part. If the inventory price is the lowest, then the company looks to see if the present inventory can meet the order. If it can, then the company responds with the filled order and adjusts its own inventory to reflect the fact that the parts have shifted locations. If the company cannot satisfy the order from its inventory, then the company checks for unused capacity of the parts that make the ordered part. If there is unused capacity, then the company boosts its production to meet the demand. If it cannot boost production sufficiently, then the company generates a request to itself to make more of that part during the next period.

If the production price is the lower, then the company will attempt to satisfy the order purely from newly produced parts and will not touch its present inventory. Either the present production will meet the order or not. If the present production of the required parts meets or exceeds the demand, then the amount of the part in the order is sent to the company that placed the order. If the present production cannot satisfy the request, then the company will send what it can from the production for the present time period and generate a request to itself for the next time period for the remaining parts.

If the purchase price is the lower, then the company will pass along the order to its supplier. In the present configuration, this will happen only when the customer company makes a request to the focus company for a part that the focus company does not manufacture. Focus will pass along a request to the supplier company. Once this request reaches the supplier, it is treated like any request to a company. The only difference between a request to the supplier and a request to the focus company is that a supplier is not permitted to pass the request to yet another supplier. A potential upgrade to Genie would be to break this present dependence upon a strictly defined set of three companies and make the number of companies and their roles variable. This might increase the ability of the Genie-generated database to support additional real-world inferences.

The flow of requests and responses to requests is affected by the employees of the company and how they are assigned to departments. A request, a response or any of the internal intermediate dataflows cannot occur unless there is an employee who performs the dataflow. Each type of dataflow is associated with a particular department.

The number of transactions that employees can perform is limited. Each company has a number of employees who are assigned to certain departments. Each company always starts with one employee. Focus has a predefined employee known as `focus_boss`, customer has a predefined employee known as `customer_boss`, and supplier has a predefined employee known as `supplier_boss`. These special employees are assigned to every department and can perform every activity. Subsequently hired employees are assigned to only one department and perform only that department's associated task.

Every time an employee performs a task, the number of transactions that she/he can handle during the present time period is decreased by one. When an employee runs out

of available transaction capacity for the time period, she/he can do no more work. When a department runs out of employees who are able to process transactions, the department comes to a standstill and sends a personnel request to management. When the next time period begins, each employee's transaction capacity is reset to its default value. Different employees have different, randomly determined capacities.

A similar constraint is placed upon the departments as a whole. Whenever a transaction is performed by a department, not only does the available transaction count of a member of the department go down, but the departmental available transaction count also decreases. This constraint models capacity limitations that apply to a department as a whole. It is possible for a department to run out of transactions while employees still have transactions available and vice versa.

Management decides what to do with personnel requests. If the number of personnel requests from a department exceeds the capacity of the department by 50 percent, then management orders the personnel department to hire more employees. If the number of personnel requests from a department does not exceed the capacity of the department by 50 percent, then management merely orders the employees in the department to be trained to a higher level of competency. This training raises the default transaction capacity of the personnel.

Upon receipt of a command to hire personnel, the personnel department will find the appropriate number of new workers from the process *theworld*, which in the CLIPS component is a pool of PERSON objects. PERSONs who meet minimal requirements are hired. Each PERSON has a defined transaction capacity in each of the six possible departmental activities.

Upon receipt of a command to train personnel in a given department, the training department will randomly select personnel from that department and increment by one their transaction capacity in their departmental activity.

As has been noted, as the simulator progresses, data emanating from the simulation is written to a file that is processed by the postprocessor. This will be described in the next section.

4.3 Postprocessing and Instance Production

Various methods are used for producing instances in Genie. These methods range from one extreme, in which the simulator is the sole source of the instances, to the opposite extreme, in which the simulator has no role in instance production. In between these extremes are various strategies that involve some participation by the simulator. Figure 4 indicates the various sources of these instances in terms of the Genie components that produce or modify them and the ultimate relation for which the instances are destined. The nature of the

processing that is associated with each of these instance groups will be described in this section.

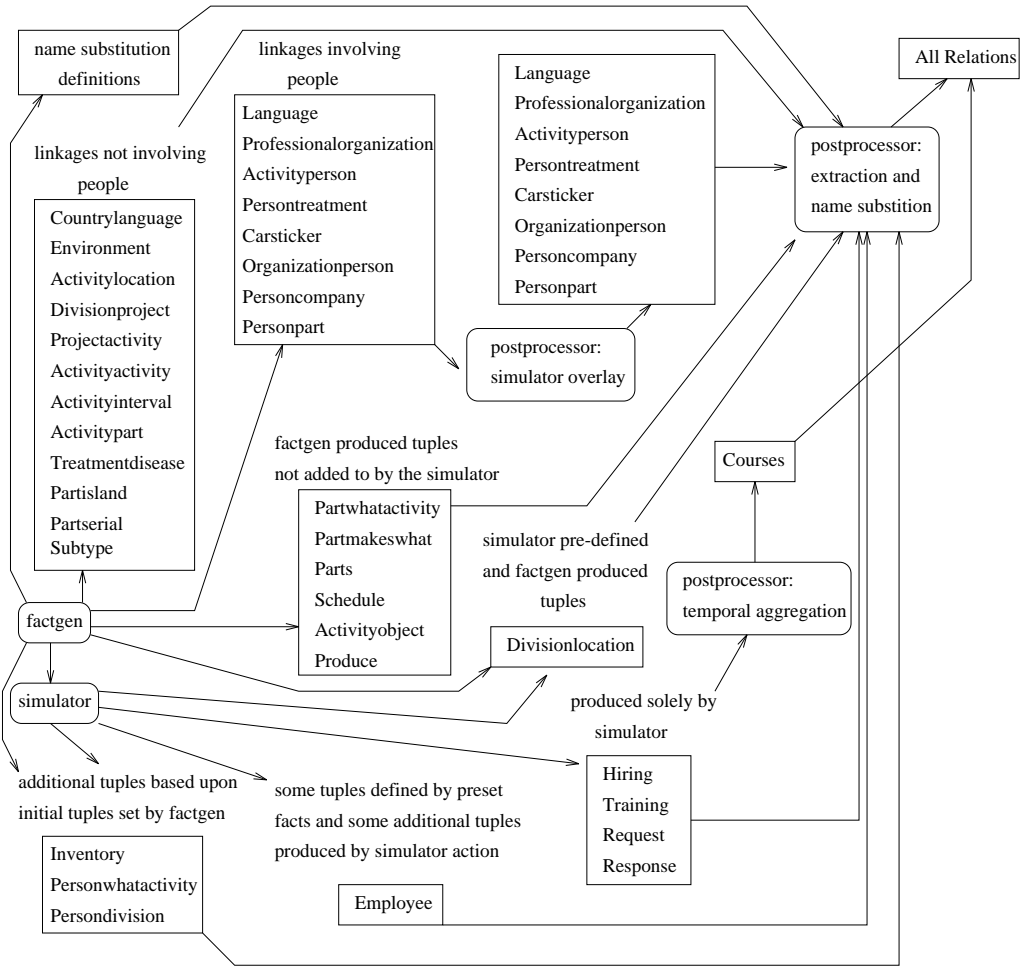


Figure 4: Source of Relations in Genie

At one extreme are the relations whose instances are produced entirely by the simulator. These are the relations *Hiring*, *Training*, *Request* and *Response*. One relation, *Courses*, is produced based on the training activities generated by the simulator. The postprocessor groups training instances by their time stamp and assumes that training activities occurred in the form of a class. This class is then added to the *Courses* relation.

In order to support the simulated work of the various companies, the simulator creates instances for the *Employee* relation. The simulator begins with the initial three bosses, which are predefined. As required, additional employee instances are then created by the simulator.

The *Divisionlocation* relation is a union of predefined simulator instances and factgen-produced instances. The three relations *Inventory*, *Personwhatactivity* and *Persondivision* are produced by the simulator based on some initial instances set by factgen. In a sense, the factgen instances provide the seed for additional instance production, and the nature of these additional instances is dependent upon the nature of the initial seed.

There are six relations that are used by the simulator to produce instances for other relations, but the simulator makes no additions to these six. These include the relations *Partwhatactivity*, *Partmakeswhat*, *Parts*, *Schedule*, *Activityobject* and *Produce*.

There are eight relations that involve people in which factgen produces all of the instances. These instances are overlaid over the instances produced by the simulator to provide some additional semantics to these simulator-produced instances. These include the relations *Languages*, *Professionalorganizations*, *Activityperson*, *Persontreatment*, *Carsticker*, *Organizationperson*, *Personcompany* and *Personpart*. All of these provide additional details about people, and each is expressed in terms of some person, such as *pe1*, *pe15*, *pe47*, that is produced by factgen. The postprocessor associates each of these *pe* terms with some person who was generated by the simulator out of the pool. In this way, some person that was assigned to a particular department by the simulator or received training can also be associated with the ability to speak a particular foreign language, such as Romansch, or have a membership in some professional society, such as IEEE. The professional society could be used to make an association between a person and a particular technology area.

The factgen program also produces twelve relations that do not involve people and are not associated with the simulator. These include *Countrylanguage*, *Environment*, *Activitylocation*, *Divisionproject*, *Projectactivity*, *Activityactivity*, *Activityinterval*, *Activitypart*, *Treatmentdisease*, *Partisland*, *Parserial* and *Subtype*.

As shown in the chart, most of the relations (with the exception of *Courses*) can be processed with name substitutions. As was illustrated in the example in Section 4.1, this permits internally generated domain instances to be given names that have meaning in the “real world,” and thus provides a means of supporting inferences such as those involving cold-weather expedition gear. This name substitution task is performed by the postprocessor.

In addition, the postprocessor also extracts instances from the various files produced by factgen and the simulator and populates various output files that are related to the final relations.

5 Conclusions

This paper has made a number of contributions to database research – specifically security-oriented database inference research. In this paper we have presented the first database generation system that produces an inference-rich test database. By structuring this system around a simulator, we believe that we have an approach that ensures that the database will have sufficient coherence of data to support a set of desired inferences.

A second contribution of this paper is the description of the AERIE test database that we are currently using in our inference research. This database could become a standard by which inference detection tools are tested and evaluated.

A third contribution of this paper is the presentation of various inference targets, organized according to the inference classes of the AERIE inference model. This collection of inferences provides a useful collection of different types of inference that could be used to test inference detection tools. As is noted, most of these examples are currently supported by Genie.

The final contribution of this paper is the description of the Genie system itself. We have validated the usefulness of Genie in the development of Wizard, a tuple-level inference detection tool that is under development as part of the AERIE project. We have used the Genie-generated database with our Wizard inference detection tool. Genie has met our expectations and validated our decision to embark upon the path of using an automated database generation tool. Genie has generated the necessary instances for Wizard to detect a number of potential inference paths. In fact, the value of the Genie database has been demonstrated in the fact that Wizard has found a large number of inference paths. This has resulted in the project having to focus on methods to group the paths, much as we have done with our schema-level tool Merlin [11], in order not to overwhelm the inference analyst. The Genie database has thus stretched the Wizard tool and opened up promising areas for additional research.

The value of a scalable, automatic database generator has also been validated with our experience. Due to some memory limitations in our current development system, we found it necessary to reduce the size of the database that was being used for testing Wizard. We found that this was an easy task to do with Genie. This would have been a very difficult task if we had used a hand-generated test database, since in the course of reducing the size of the database we might have eliminated some useful inferences. With Genie, this was not the case, since we were able to scale down the size of the database, but preserve all of the inferences.

The extensibility of Genie has also been validated. Over the last few months, we have sought to incorporate into Genie most of the inference examples that we have acquired over the first two years of the AERIE inference project. We found that adding these additional inferences required a minimal amount of effort. As we continue to explore

instance-level inference detection, we are confident that Genie will be able to provide the necessary inference-rich database that will be needed to test Wizard.

References

- [1] Leonard J. Binns. Inference Through Secondary Path Analysis. In *Proceedings of the Sixth IFIP 11.3 Working Conference on Database Security*. IFIP, August 1992.
- [2] 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.
- [3] Dina Bitton and Carolyn Turbyfill. A retrospective on the wisconsin benchmark. In Michael Stonebraker, editor, *Readings in Database Systems*. Morgan Kaufman Publishers, Inc., 1994.
- [4] R. G. G. Cattell. An engineering database benchmark. In Michael Stonebraker, editor, *Readings in Database Systems*. Morgan Kaufman Publishers, Inc., 1994.
- [5] Peter Pin-Shan Chen. The entity-relationship model – toward a unified view of data. *ACM Transactions of Database Systems*, March 1976.
- [6] Thomas D. Garvey. SRI RADC Database. Private communication, 1993.
- [7] Thomas H. Hinke. Inference Aggregation Detection In Database Management Systems. In *Proceedings 1988 IEEE Symposium on Security and Privacy*, April 1988.
- [8] Thomas H. Hinke. Database Inference Engine Design Approach. In Carl E. Landwehr, editor, *Database Security II: Status and Prospects*. North-Holland, 1990.
- [9] 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.
- [10] 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.
- [11] 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).
- [12] 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.

- [13] Teresa F. Lunt. Toward a Multilevel Relational Data Language. In *Proceedings of the Fourth IFIP Aerospace Computer Security Applications Conference*, December 1988.
- [14] 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.
- [15] 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.
- [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*, May 1993.
- [17] 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.

A Genie Relations

This section will present each of the relations currently supported by Genie. For each relation, we will present a brief description of the nature of data contained in the relation and the schema, along with some sample data. Only those attributes whose meaning is not self-evident will be described.

Employee This relation, shown in Table 3, contains some basic information about an employee. Every time a department performs a transaction, an employee associated with that department must perform the activity. The *Personneltrain* attribute is the number of transactions that this employee could perform in the Personnel Department during a single simulator time tick. Each person has a potentially variable ability to perform different departmental activities. The attribute *Trainingtrain* represents the transaction capacity of the employee in training activities. Similarly, the attributes *Purchasingtrain*, *Managementtrain*, *Salestrain* and *Prodtrain* represent, respectively, the transaction capacities of the employee in purchasing, management, sales and production activities. The attribute *time* is the time tick that this entry was made.

Hiring This relation, shown in Table 4, reflects the hiring of an employee for a particular company and division at a particular time.

Personwhatactivity This relation, shown in Table 5, is used to indicate the types of activities that people perform in each of the companies.

FGL Link Commands	
Link Command Name	Description
linkperatt	This produces a linkage from a person to an attribute.
linkattatt	This linkage is the most general FGL linkage command available. Attributes may be linked to attributes at will. Once attributes have been linked, pairs of stringname and namelink linkages can be used to string-substitute meaningful names for attributes. This linkage makes Genie a schema generator as well as a tuple generator.
linkpartact	This linkage indicates what activity a part performs or to which it is associated
linkenv	This linkage allows an environment to be associated with an activity.
linkactobj	This linkage allows an activity to be defined to produce or act upon an object or a part. This is how the simulator is informed about what part manufactures what other part, and what part performs what activity in order to produce what object.
threelink	This linkage is the only 3-ary relation that is absolutely necessary. It creates a number of source, verb and object (SVO) triples. The source is always a part, and the verb is always an activity. The object may be either a part or an object.
prodlink	This command links a produce group to a part group, using threelink information. This sets the production rate of part-makers.
schedlink	This command links a schedule group to a part via threelink information.
invlink	This command links inventory definitions to parts through threelink information.
linkcorpdiv	This linkage is somewhat different from the other linkages. It links from a division group to a division group to a location group. This linkage is intended to allow the user to create arbitrary corporations that are not linked to the simulator, to add new divisions to the three existing corporations, and to add new locations to the existing corporations.
linkactloc	This command links an activity with a location.
linkdivproject	This linkage allows a corporation (which is a division) to be associated with a project.
linkprojectact	This linkage allows an activity to be associated with a project.
linkactact	This linkage allows the first listed activity (the from activity) to be defined to occur before the second listed activity (the to activity).
linkactint	This linkage allows a time interval to be linked to an activity. The time interval is measured in simulator time ticks.
linkactpart	This linkage indicates that an activity needs certain parts.
linkactper	This linkage allows people to be associated with a project activity.
namelink	This command associates "real world" names with factgen-generated names.

Table 2: FGL Link Statements

Employee Relation									
Name	Address	Phone	Personnel-train	Training-train	Purchasing-train	Management-train	Sales-train	Prod-train	Time
person1	gen1	gen2	4	3	1	2	3	5	1
person2	gen3	gen4	1	1	2	4	3	3	1

Table 3: Employee Relation

Hiring Relation			
Name	Company	Division	Time
person1	focus	sales	1
person2	supplier	production	3

Table 4: Hiring Relation

Personwhatactivity Relation		
Company	Name	Activity
customer	person1	a0
supplier	person2	a1

Table 5: Personwhatactivity Relation

Training This relation, shown in Table 6, is used to indicate the results of training. The attribute *Division* indicates the skill type in which the employee is being trained, with *Newlevel* indicating the new level of skill that the employee has after training. The attribute *Time* indicates the time period when the training occurred.

Training Relation			
Name	Division	Newlevel	Time
person0	sales	3	1
person1	training	5	3

Table 6: Training Relation

Divisionlocation This relation, shown in Table 7, shows the physical location of each of the divisions of each of the companies.

Divisionlocation Relation		
Company	Division	Location
focus	management	10
supplier	production	dcity

Table 7: Divisionlocation Relation

Persondivision This relation, shown in Table 8, is used to indicate the company and division to which each person is assigned and the time when the assignment was made.

Persondivision Relation			
Company	Name	Division	Time
focus	person2	sales	0
focus	person1	sales	1

Table 8: Persondivision Relation

Partwhatactivity This relation, shown in Table 9, indicates the type of activity that a particular part performs.

Partwhatactivity Relation	
Part	Activity
p0	a33
p10	a2

Table 9: Partwhatactivity Relation

Partmakeswhat This relation, shown in Table 10, is used to indicate which parts are used for making other parts, or which parts are used in conjunction with some task. The activity that is performed could be the keyword “manufacturing,” in which case the part is one that manufactures the direct object. If the activity is not manufacturing, then it could be some other activity, such as ‘picking’. The *Directobject* attribute is the direct object for the activity.

Partmakeswhat Relation		
Part	Activity	Directobject
p1	manufacturing	p4
p4	a2	o9

Table 10: Partmakeswhat Relation

Courses This relation, shown in Table 11, is used to indicate courses taken by employees and the grades received.

Courses Relation		
Course	Name	Grade
c0	person3	A
c3	person2	F

Table 11: Courses Relation

Parts This relation, shown in Table 12, is used to indicate parts explosions, i.e., the components that comprise a part. This uses an approach analogous to that proposed in [4]. The attribute *Subpart* holds the part number that is a subpart component of part. The attribute *Quantity* indicates how many instances of the subpart are used in the part.

Parts Relation		
Part	Subpart	Quantity
p2	p4	2
p3	p6	1

Table 12: Parts Relation

Schedule This relation, shown in Table 13, is used to indicate the company for which the scheduled activity is occurring. The attribute *Activity* holds the activity that the schedule causes to occur. *Directobject* is the direct object of the activity; *Time* is the time slot during which this activity is scheduled to occur, and *Quantity* is the number of times that this activity is scheduled to occur during that time period.

Schedule Relation				
Company	Activity	Directobject	Time	Quantity
focus	manufacturing	p3	11	2
customer	a0	o9	2	2

Table 13: Schedule Relation

Response This relation, shown in Table 14, is used to indicate the shipment of parts between companies. The attribute *Tocompany* is the company receiving the quantity of parts shipped, and *Fromcompany* is the company supplying the parts. The attribute *Time* is the time the shipment occurred.

Response Relation				
Tocompany	Fromcompany	Part	Quantity	Time
customer	focus	p3	2	2
focus	supplier	p9	33	44

Table 14: Response Relation

Activityobject This relation, shown in Table 15, is used to indicate the object acted upon by a particular activity.

Activityobject Relation	
Activity	Object
a1	o9
a2	o8

Table 15: Activityobject Relation

Inventory This relation, shown in Table 16, is used as a parts inventory relation. The attribute *Quantity* stores the quantity of parts that the company has on hand. The *Inventoryprice* attribute stores the value of a part in inventory, and the *Productionprice* specifies the cost of producing a part. The *Purchaseprice* attribute specifies how much it costs to purchase a part. The *Time* attribute indicates the time that this inventory level was established.

Inventory Relation						
Company	Part	Quantity	Inventoryprice	Productionprice	Purchaseprice	Time
focus	p1	2	2	3	5	1
customer	p1	0	9999	9999	1	2

Table 16: Inventory Relation

Produce This relation, shown in Table 17, is used to indicate the number of parts produced and a projected maintenance schedule for them. The attribute *Time* is the number of simulator time ticks it takes to produce a certain quantity. *Timebeforemaint* is the time before the part needs maintenance, and *Mainttime* is an estimate of how long maintenance takes. *Lifetime* is the overall lifetime of the part.

Produce Relation					
Part	Quantity	Time	Timebeforemaint	Mainttime	Lifetime
p2	2	1	2	1	111
p3	1	3	22	22	2222

Table 17: Produce Relation

Request This relation, shown in Table 18, represents orders. *Tocompany* is the company that is being requested to supply the part, and *Fromcompany* is the company making the request. *Object* is the part or object being requested, and *Quantity* is the quantity of parts that are being requested.

Request Relation				
Tocompany	Fromcompany	Object	Quantity	Time
focus	customer	p1	3	1
customer	focus	p1	3	2

Table 18: Request Relation

Language This relation, shown in Table 19, is used to indicate indicate foreign language skills of the employees.

Language Relation	
Name	Language
person1	Romansch
person3	English

Table 19: Language Relation

Professionalorganization This relation, shown in Table 20, is used to indicate the professional organizations to which employees belong.

Professionalorganization Relation	
Name	Professionalorg
person4	ACM
person2	IEEE

Table 20: Professionalorganization Relation

Countrylanguage This relation, shown in Table 21, is used to indicate the languages spoken in the various countries with which the companies do business.

Countrylanguage Relation	
Country	Language
United States	English
Switzerland	Romansch

Table 21: Countrylanguage Relation

Environment This relation, shown in Table 22, is used to indicate significant environmental factors that are associated with each type of activity. The value of the attribute *Environment* could be either an adjective (such as “hot”) or a geographical designation (such as “north”).

Environment Relation	
Environment	Activity
hot	picking
e2	a3

Table 22: Environment Relation

Activitylocation This relation, shown in Table 23, is used to indicate the normal locations where an activity might occur.

Activitylocation Relation	
Activity	Location
a1	ll
digging	farm

Table 23: Activitylocation Relation

Divisionproject This relation, shown in Table 24, is used to indicate the projects supported by each company. The attribute *Division* is a corporate division or an entire corporation.

Divisionproject Relation	
Division	Project
focus	alpha
customer	alpha

Table 24: Divisionproject Relation

Projectactivity This relation, shown in Table 25, is used to indicate the activity associated with each project.

Projectactivity Relation	
Project	Activity
pr5	a0
aerie	deduction

Table 25: Projectactivity Relation

Activityactivity This relation, shown in Table 26, is used to indicate the temporal relationships between activities. *Activity0* is the preceding activity, and *Activity1* is the subsequent activity.

Activityactivity Relation	
Activity0	Activity1
a0	a1
a1	a2

Table 26: Activityactivity Relation

Activityinterval This relation, shown in Table 27, is used to indicate the time length of the activity. It is not specifically the time that the activity has taken or the time that the activity is projected to require. It is simply a time interval in terms of a specific simulation time period (such as time tick 12) associated with the activity.

Activityinterval Relation	
Activity	Interval
a0	1
a1	2

Table 27: Activityinterval Relation

Activitypart This relation, shown in Table 28, is used to associate a part, such as a piton, with a particular activity, such as rock climbing.

Activitypart Relation	
Activity	Part
a0	p0
a0	p1
a1	p1

Table 28: Activitypart Relation

Activityperson This relation, shown in Table 29, is used to associate an activity with a particular person.

Activityperson Relation	
Activity	Name
a0	person1
a0	person2

Table 29: Activityperson Relation

Persontreatment This relation, shown in Table 30, is used to associate a person with a particular medical treatment or drug used to treat the employee.

Persontreatment Relation	
Name	Treatment
person1	AZT
person2	chemotherapy
person3	at22

Table 30: Persontreatment Relation

Treatmentdisease This relation, shown in Table 31, is used to associate particular medical treatments with particular illnesses or diseases. This represents specialized knowledge that may not be in a database but could be available to an adversary.

Treatmentdisease Relation	
Treatment	Disease
AZT	HIV
chemotherapy	cancer

Table 31: Treatmentdisease Relation

Carsticker This relation, shown in Table 32, is used to indicate the company car sticker that is associated with each employee.

Carsticker Relation	
Name	Company
person3	Ajax
person4	AlphaAlpha

Table 32: Carsticker Relation

Personcompany This relation, shown in Table 33, is used to indicate the company of employment of each employee.

Personcompany Relation	
Name	Company
person3	Ajax
person4	Ajax

Table 33: Personcompany Relation

Partisland This relation, shown in Table 34, is used to indicate the results of accident investigations conducted by the company in the Pacific Ocean. It indicates various parts that were discovered and the island on which they were discovered.

Partisland Relation	
Part	Island
enginepart	Crashisland
MaryCeleste'slog	PuertoRico

Table 34: Partisland Relation

Personpart This relation, shown in Table 35, is used to associate particular types of parts with particular people that have disappeared in accidents involving company products. Such a relation could, for example, include all of the parts on Amelia Earhart's missing aircraft.

Personpart Relation	
Name	Part
Amelia	plane

Table 35: Personpart Relation

Partserial This relation, shown in Table 36, is used to associate particular part serial numbers with particular types of parts.

Partserial Relation	
Serialnumber	Part
PP1256	enginepart
PP1	plane

Table 36: Partserial Relation

Subtype This relation, shown in Table 37, is used to indicate subtype relationships, such as that a cotton picker is a type of farm equipment. This relation represents common knowledge that could be available to an adversary who is mounting an inference attack against a database.

Subtype Relation	
Part	Subtypepart
plane	enginepart
battery	BP205

Table 37: Subtype Relation