Proceedings of the 2004 IEEE
Workshop on Information Assurance
United States Military Academy, West Point, NY June 2004

Fuzzy Dependency and Its Applications in Damage Assessment and Recovery[!]

Yanjun Zuo and Brajendra Panda, *Member, IEEE*

Abstract

Fuzzy dependency in a database delineates a loose dependency relationship between two sets of attributes. It describes logical relationships among attributes in a database relation and those relationships can't be fully specified by functional dependencies, which focus on database schema and data organization. This characteristic of the database schema can be used to perform damage assessment and also to build fuzzy recovery model. In this paper, we formally define the concept of fuzzy dependency and introduce several inference rules. Then we focus on recovery from information attacks. An architecture for fuzzy value generation during recovery, based on fuzzy dependency relationships, is also presented. Fuzzy dependency can accelerate the post attack recovery process because it can generate acceptable values for damaged data quicker compared to that in traditional recovery schemes. Although the generated fuzzy values may not offer the absolute accuracy, they are acceptable for many real-time applications, which require prompt response more than the data accuracy.

Index Terms: Damage Assessment, Recovery, Fuzzy Dependency

I INTRODUCTION

Database damage assessment and recovery after intrusion detection can be very time consuming. The conventional database damage assessment and recovery mechanisms often need to scan the database logs, perform either transaction or data dependency analysis, and schedule REDO and UNDO operations. In many cases, the delay during damage assessment and recovery is unacceptable for real-time military and commercial applications. The data items in a database have logical relationships among them and can be used to promptly provide estimated values of damaged data items in case of recovery from cyber attacks. Fuzzy dependency, which is similar to functional dependency ([1][2][3], to cite a few) describes these logical relationships and can be utilized in recovery methods after the damage assessment phase.

In a database relation, while a primary key value can uniquely determine other non-key values in a tuple, some non-key values can "fuzzily" determine other non-key values. For instance, in a database relation, the value of attribute *rank* may not uniquely determine the value of another attribute *salary* in a tuple. But for each *rank* value, in many cases, there would be a corresponding range of salary. Let's consider a University database, which stores the payroll information for faculty and staff. Based on the university payroll policy, an assistant professor's salary range is between \$50,000-75,000 and an associate professor's salary is between \$65,000 – 95,000. In a tuple of this relation, if we are given a value for the *rank*, we can't determine the *salary* value but we can provide a reasonable range for *salary*. Fuzzy dependency describes this kind of dependency relationship between two attributes of an entity. Like a functional dependency, a fuzzy dependency is a property of the relation schema and not of a particular legal relation state (extension) of the schema. Hence, fuzzy dependency cannot be inferred automatically from a given relation extension but must be defined explicitly by someone who knows both the semantics of the attributes and some "extra" properties of data relations, i.e., other requirements.

In this paper, we show that based on the nature of fuzzy dependency and its inference rules, a recovery model can be built. This model uses the stored fuzzy rules and performs fuzzy reasoning to provide fuzzy values in a timely manner. In addition, we also describe how fuzzy dependency is useful in enforcing system integrity and performing intrusion detection.

The remaining of this paper is organized as follows. Section 2 introduces some related work. Section 3 defines fuzzy dependency and discusses several fuzzy rules. Section 4 presents the fuzzy recovery

Yanjun Zuo and Brajendra Panda are with the University of Arkansas, Fayetteville, AR 72701, USA (e-mail: {yzuo, bpanda}@uark.edu)

! *This work was supported in part by US AFOSR under grant F49620-01-10346.*

model based on fuzzy dependency and semantic rules. Section 5 concludes the paper.

II. Related Work

Traditional centralized database damage assessment and recovery methods use either data dependency [4] or transaction dependency models [5]. Liu et al. developed a model to recover corrupted data in distributed database systems [6]. But all these models require intensive scanning of database logs. Panda et al. developed log segmentation methods to reduce the amount of scanning to be done [7] [8]. However, all these methods also need to either totally shutdown the database or block transactions during damage assessment and recovery. Using replications of data or services [9][10][11] to make data available during system damage assessment and recovery is quite expensive. In many cases, it is not realistic due to extensive resource usages or difficulty in synchronization of multiple copies. Valsangkar et al. developed an architecture to make data available ceaselessly during recovery [12]. But that model fell short of providing any systematical approximation scheme. The fuzzy dependency relationships presented in this paper can be used along with other approximation algorithms to give reasonable fuzzy values. These values can be quickly supplied to those vital transactions needed to perform essential services. Our main contribution in this paper is the notion of fuzzy dependency and its applications in damage assessment and recovery. We argue that this type of data dependency is useful in generating estimated values of damaged data items in a prompt manner so that critical transactions requiring these data can continue with their operations.

III. Fuzzy Dependency

In this section, we define fuzzy dependency and introduce two lemmas and seven inference rules, which are essential in fuzzy reasoning.

Definition: For two sets of attributes X and Y of a relation R, Y is fuzzily dependent on X (or X fuzzily determines Y) if and only if for every value a_i in the domain of X, $a_i \in D(X)$, there is an uniquely determined subset S_i' in the domain of Y, $S_i' \subseteq D(Y)$, such that a tuple T_i in a relation instance of R with value a_i for X should have a value $b_i \in S_i'$ for Y. But for any value $b_j \in D(Y)$, $b_j \notin S_i'$, b_j can not be the Y value for T_i. We denote this relation as X =>(F) Y and abbreviate fuzzy dependency as FZD. The set of attributes X is called left-side attributes of FZD and Y is called right-side attributes of FZD. S_i' can be expressed as a consecutive range, e.g., (1-10), a discrete set, e.g., ("NY", "AR", "CA"), or a regular expression. In this paper, we focus on the first two formats and use the general term "range".

Lemma 1: Let *r* be an instance of relation R. If X =>(F) Y, then for any two tuples T_i and T_j in *r* such that they agree on the same X value, $T_i[X] = T_j[X]$, we must have:
(1) $T_i[Y] \in S_i' \subseteq D(Y)$ and $T_j[Y] \in S_j' \subseteq D(Y)$.
(2) $S_i' = S_j'$

Proof: For (1) from the definition we can directly infer $T_i[Y] \in S_i' \subseteq D(Y)$ for tuple T_i and $T_j[Y] \in S_j' \subseteq D(Y)$ for tuple T_j. For(2), let's assume that $S_i' \neq S_j'$. In tuple T_i, we have one subset S_i' for $a_i = T_i[x]$ and in tuple T_j we have another subset S_j' for $ai = T_j[X]$. Based on the fuzzy dependency definition, any value $a_i \in D(X)$ can have only an unique corresponding $S_i' \subseteq D(Y)$. So $S_i' \neq S_j'$ is impossible.

Lemma 2: For two sets of attributes X and Y in a relation R, if Y is functionally dependent on X, i.e., $X \rightarrow Y$, then Y must be fuzzily dependent on X, i.e., X =>(F) Y.

Proof: Given $X \rightarrow Y$. For an instance *r* of R, if there exist two tuples T_i and T_j such that $T_i[X] = T_j[X]$, we must have $T_i[Y] = T_j[Y] \subseteq D(Y)$. If we let a subset $S_i' = \{T_i[Y]\} \subseteq D(Y)$ and another subset $S_j' = \{T_j[Y]\} \subseteq D(Y)$. Then $S_i' = S_j'$. Thus we have X =>(F) Y.

Rule 1: (Reverse rule): If X =>(F)Y, then Y =>(F)X.

Proof: Given X =>(F) Y. For every $a_i \in D(X)$ $(0 < i \leq m)$, there is an unique subset $S_i' \subseteq D(Y)$. Now consider every $b_j \in D(Y)$. For every S_i', if $b_j \in S_i'$, we put S_i' in a new set SS_j' such that SS_j' contains all the subsets which has b_j as their members, $SS_j' = \{ \ldots S_k' \ldots S_j' \ldots\} \subseteq D(Y)$. Since there is one-to-one relationship between every a_i and S_j', if we replace every S_j' with a_j, we have a new subset $SS_j'' = \{ \ldots a_k \ldots a_j \ldots \} \subseteq D(X)$. Since every b_j in D(Y) has such a corresponding subset in D(X), we have Y =>(F) X.

Rule 2: (Reflexive rule): If $X \supseteq Y$, then X =>(F) Y.

Proof: Given $X \supseteq Y$, we have $X \rightarrow Y$. According to lemmas 2, we have X =>(F) Y.

Rule 3: (Augmentation rule): If X =>(F) Y, then XZ=>(F) YZ.

Proof: Given X =>(F) Y. For two tuples Ti and Tj in an instance *r*, if $T_i[X] = T_j[X]$, we must have two subsets $S_i' = S_j' \subseteq D(Y)$. For the set of attributes XZ, if $T_i[X] \cup T_i[Z] = T_j[X] \cup T_j[Z]$, then we have $T_i[Z] = T_j[Z]$. So $(S_i' \cup T_i[Z]) = (S_j' \cup T_j[Z]) \subseteq (D(Y) \cup D[Z])$. Thus, XZ=>(F) YZ must be held.

Rule 4: (Transitive rule): If X=>(F) Y and Y=>(F) Z, then X=>(F) Z.
Proof: Given X=>(F) Y. For every $a_i \in D(X)$, we have a unique $S_i' \subseteq D(Y)$. Now suppose S_i' has m elements, $S_i' = \{b_{i1}, b_{i2}, \ldots, b_{im}\}$. Given Y=>(F) Z. For every element b_{ik} $(0 < k \leq m)$ in S_i', there is an unique subset $S_{ik}' \subseteq D(Z)$. By taking union of all such subsets in D(Z), we have a new set $SS_i' = \{S_{i1}', S_{i2}', \ldots S_{ik}' \ldots S_{im}'\} \subseteq D(Z)$. This set SS_i' in Z is for the subset S_i' in Y which in turn corresponds to a value $a_i \in D(X)$. So we have X=>(F) Z.

Rule 5: (Decomposition (or projective) rule): If X=>(F) YZ, then X=>(F) Y.
Proof:
1. X=>(F) YZ. (Given)
2. YZ=>(F) Y (Based on reflexive rule since YZ $\supseteq$ Y)
3. X=>(F) Y. (Transitive rule on 1 and 2)

Rule 6: (Union (or additive) rule): If X=>(F) Y and X=>(F) Z, then X=>(F) YZ.
Proof:
1. X=>(F) Y. (Given)
2. X=>(F) Z. (Given)
3. XX=>(F) XY. (Augmentation rule on 1)
   X=>(F) XY. (Since XX = X)
4. XY=>(F) YZ. (Augmentation rule on 2)
5. X=>(F) YZ. (Transitive rule on 3 and 4)

Rule 7: (Pseudo transitive rule): If X=>(F) Y and WY=>(F) Z, then WX=>(F) Z.
Proof:
1. X=>(F) Y. (Given)
2. WY=>(F) Z. (Given)
3. WX=>(F) WY. (Augmentation rule on 1)
4. WX=>(F) Z. (Transitive rule on 3 and 2)

It is observed that if X =>(F) Y and the domain of X has m possible values $D(X) = \{a_1, a_2, \ldots a_i, \ldots a_m\}$ $(0 < i \leq m)$, we have a set of subsets $S_1', S_2', \ldots S_i', \ldots S_m'$ in domain D(Y), which corresponds to $a_1, a_2, \ldots a_i, \ldots a_m$ respectively. If for any two subset S_k' and S_i' such that $S_k' \cap S_i' = \emptyset$, we conclude that X is not only fuzzily dependent on Y, i.e.,Y =>(F) X, but also functionally dependent on Y, i.e., $Y \rightarrow X$.

Unlike functional dependencies, fuzzy dependencies in a database should be specified not only at the attribute level but also for each instance of the relevant attributes. Only one-way fuzzy dependency needs to be stored since the other way can be deduced based on the reverse rule. In addition, only those fuzzy dependencies that are not functional dependencies are stored for the reason of storage efficiency. Finally, fuzzy dependency storage does not save indirect or deduced fuzzy relationships. A template of fuzzy dependency storage for X=>(F)Y is shown in Figure 1. The fuzzy instance stores fuzzy range for each instance of a fuzzy dependency.

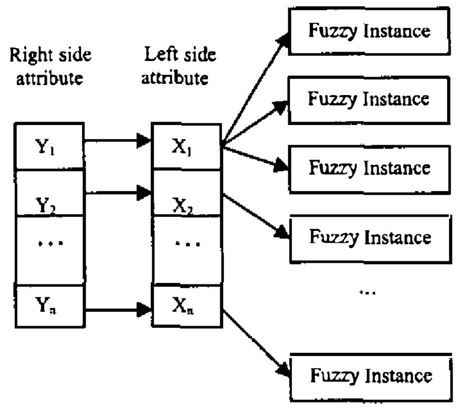

Figure 1: A Template for Fuzzy Dependency Storage

IV. THE APPLICATIONS OF FUZZY DEPENDENCY AND AN ARCHITECTURE OF FUZZY RECOVERY MODEL

Fuzzy dependency can have three major applications. First, it can be used to specify a constraint on possible tuples that can form an instance r of a relation R and to enforce database polices (fuzzy integrity checking). Policy enforcement using fuzzy dependency works as a functional dependency does and the fuzzy rules are checked whenever an instance of an attribute is created, inserted or updated. Any value that violates the fuzzy dependency rules are prohibited and hence the operations performed on that value are aborted. This is to ensure data integrity and aims at preventing normal users from mistakenly entering illegal data.

Secondly, fuzzy dependency can be used in database intrusion detection, which can be run periodically. If X =>(F)Y and an instance value of Y for a tuple does not agree on the fuzzy value range decided by the corresponding instance of X value, then the Y instance data value is considered as corrupted.

Finally, fuzzy dependency can be used to generate fuzzy values for damaged data items in a database in order to minimize the "denial of service" caused by intrusion. Some models (such as [7]) have been developed to provide fuzzy values but they usually involve significant overhead due to the requirements of collecting and summarizing database properties and using mathematical functions or other statistical means to generate fuzzy values. On the other hand, use of fuzzy dependencies is a quicker way to generate fuzzy values and can be implemented via pre-built-in rules and data specifications. Due to limited information in some cases, however, the fuzzy dependency method can't guarantee to generate satisfied values all the time. Under certain situations, it even can't generate a fuzzy value based on the current information. Hence, we use a semantic reasoning method to supplement the fuzzy dependency method for our fuzzy recovery scheme. In this hybrid model, fuzzy dependency reasoning is the major component. Only when it needs more information, does it consult the semantic reasoning unit. The average time (t) to generate a fuzzy value is based on the formula:

$$t = t_1 + t_2 * p + p * t'$$

where t_1 and t_2 are the average time consumed in fuzzy dependency reasoning and semantic reasoning respectively; p is the probability that semantic reasoning is used; t' is the time associated with communications between the fuzzy reasoning unit and the semantic reasoning unit as well as the combination of results between the two units.

A. Fuzzy Recovery Architecture

We developed a fuzzy recovery model to provide fuzzy values as quickly as possible. The core component of this model is the fuzzy value generator, which works primarily based on the stored fuzzy dependency relationships. It consists of Fuzzy Check and Data Fetch Unit, Fuzzy Reasoning Unit, Supplemental Fuzzy Rule Unit, Usage Identifier, Value Finalizer and a supplemental Semantic Reasoning Unit. This system starts after a database system has been intruded and is under repair. The fuzzy recovery model offers the promptness in generating acceptable values of damaged data items needed for real-time applications. But, it can't guarantee the absolute accuracy of the supplied values. The conventional recovery systems performing the repair work must catch up and restore the database accurately.

The key requirement is that the fuzzy value generation process should be quick enough (at least faster than the conventional database recovery techniques). Since this model looks up fuzzy dependency storage (and performs fuzzy reasoning), which is considerably smaller than the database log, this process can provide fuzzy values quicker. The framework of the fuzzy recovery model is shown in Figure 2. Various components of this model are described next.

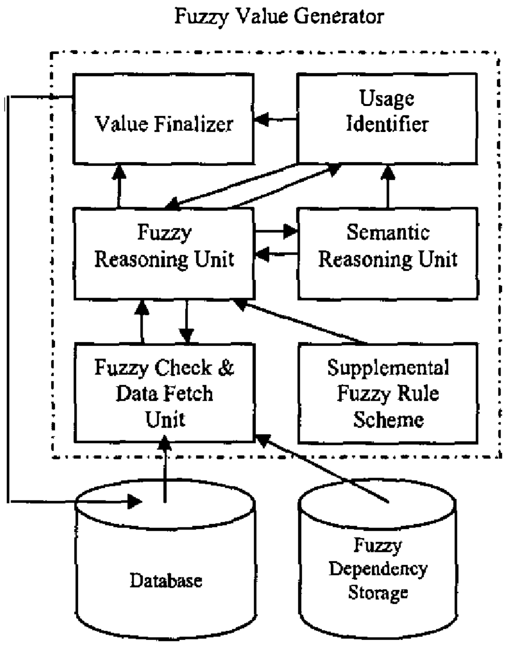

Figure 2: Framework of Fuzzy Recovery Model

1. Fuzzy Check and Data Fetch Unit

This unit is to interact with both the fuzzy dependency storage and database and execute instructions from the Fuzzy Reasoning Unit. It has two major functions. One is to identify fuzzy dependencies with other attributes for the attribute with a damaged data item in a tuple by searching the fuzzy dependency storage. The other function is to fetch relevant data items from database related to a given tuple for fuzzy reasoning. As an example for the latter case, consider a given tuple with two attributes, *salary* and *rank*. This unit is to query the database and find the values of *salary* in other tuples with close *rank* values with that in the given tuple.

2. Fuzzy Reasoning Unit

This unit uses fuzzy inference rules and supplemental fuzzy rules to reduce the range of the possible values for a given corrupted data item. It provides the best known estimated value (or a small set of values). Fuzzy reasoning can be conducted in many reasonable ways. Several cases are given below to show this process. These cases are not ordered and can be combined to use. Some of the reasoning rules can be applied to continuous and ordered sets while others may be applied to discrete sets. In the following cases, we assume the value $b \in D(Y)$ in a database tuple T_b is detected as corrupted (unless specified otherwise).

Case 1: X =>(F)Y and Z =>(F)Y. Let $a \in D(X)$ and $c \in D(Z)$ be two values corresponding to b in T_b. From the fuzzy dependency storage, we learn that c and a determine a range C and A for b respectively. So the correct b value should be in the set of intersection of C and A. This narrows the set of possible values for b. For a certain value of the right-side attribute of FZD, if the subsets based on multiple fuzzy dependencies have no common values, we may define a priority for each fuzzy dependency relation and pick up the subset based on the highest priority of a fuzzy dependency in this case.

Case 2: Suppose X_1=>(F)Y, X_2=>(F)Y, ..., X_k=>(F)Y, and tuple T_b has its Y value as b, which is considered as damaged. If we can identify that X_1 value in T_b agree on (equal or close to) corresponding X_1 values in tuple $T_{b1}, T_{b2}, \ldots, T_{bj}$ and corresponding Y value in those latter tuples are $b_{11}, b_{12}, \ldots, b_{1j}$, then we compute the average b_1 = average ($b_{11}, b_{12}, \ldots, b_{1j}$). Furthermore, for X_2 value, we calculate average b_2 (based on a set of tuples), for X_3 value, we compute b_3, and so on. Then, by averaging all these values, the final average b = average ($b_1, b_2, \ldots, b_k$) is used as the fuzzy value for the damaged Y value in tuple T_b.

Case 3: For a given fuzzy dependency instance, if we can "plug in" an external third value, we could further narrow down the fuzzy range. This third factor might not be an attribute of the database or may be present in a different table and can be provided as needed. Consider an example of the fuzzy dependency relationship, *salary*=>(F) *living standard*. Let's assume that in a database tuple T_b for an employee E, the *salary* value is $50,000 and its corresponding *living standard* value is detected as damaged. Based on the salary level and stored fuzzy dependency instance, we identify the range for *living standard* as (average, middle high, high). If we can find a third factor, *region*, which denotes an employee's living region and has a value of "New York", then we might prefer the value "average" because the salary value ($50,000) in New York region is considered average. However, if the region is a small town in an undeveloped country, then for *living standard* the value "high" may be used as the fuzzy value for the salary of $50,000. Here the third factor "region" is used although it is not a database attribute.

Case 4: In the situation where cascaded damaged data items are detected, the pseudo transitive rule and transitive rule can be used to make substitutions. For the first case, consider two fuzzy dependency relationships *rank*=>(F) *salary* and (*region, salary*)=>(F)*living standard*. We can make substitution and deduce (*region, Rank*)=>(F)*Living standard*. For instance, suppose that for a tuple T_b with attributes *salary, rank*, and *living standard*, the *salary* and *living standard* values are both identified as damaged. We want to first provide a fuzzy value for the latter attribute (*living standard*). We can't rely on the *salary* value since it is damaged. By using this rule, we can use the value for the *rank* attribute in T_b. A range can be retrieved in order to give a fuzzy value for the damaged value of *living standard*. For the transitive rule, X=>(F)Z and Z=>(F)Y, then X=>(F)Y (once again, we assume that Y value has been detected as damaged), if the Z value is not available (for such reasons as data $z \in D(Z)$ is being updated by other transactions or locked by the system or z is considered as inaccurate), the fuzzy reasoning unit may use X=>(F)Y rather than Z=>(F)Y.

Case 5: According to the union (or additive) rule, we know X =>(F)Y and X =>(F)Z, then X=>(F)YZ, where X, Y, Z are three attributes in a database. In some cases, although Y and Z have no stored fuzzy dependency relationship, Y and Z should approximately agree with each other. If the value for Y is detected as damaged, we might observe the Z value in the range determined by the corresponding X value and estimate Y value based on that observation. Consider the following example. Suppose in a database, the fuzzy rules are specified as shown below. Also assume that in a tuple the value of the attribute *power* is damaged and the value of the corresponding *rank* in that tuple is 2.

Rank	**Salary ($)**	**Rank**	**Power**
1	20,000-65,000	1	2-5
2	40,000-130,000	2	3-10
3	55,000-200,000	3	6-15

We assume that both the attributes, *salary* and *power*, have direct fuzzy dependency relationships with *rank*, but the database does not specify the fuzzy dependency relationship between the *salary* and *power* because either their relationship is not obvious or they can be deduced from the other stored fuzzy rules, or they can be observed from the data patterns.

The tuple shows *rank* value is 2 and a range (3-10) is identified for the *power* value. The *salary* value associated with *rank* 2 in that tuple is $40,000, which is at the low end of the range ($40,000-130,000); we would pick up 3 (or 4) as the estimated value for *power*, which is also at the low end of the *power* range (3-10).

Case 6: The fuzzy reasoning unit can use direct observations or supplemental fuzzy rules to generate fuzzy values. For instance, in a database where there is a fuzzy dependency, *years of services* =>(F) *salary*, for tuple T_i, suppose the value $b \in D(salary)$ = $90,000 with a value of *years of services* as 5 was detected damaged. From other tuples in the same relation we observe the consecutive values of *years of services* corresponding to *salary* values as below:

Tuple	**Year of Services**	**Salary ($)**
T_k	2	28,000
T_j	4	32,000
T_m	6	34,000
T_n	8	36,000

This pattern shows a nearly linear relationship between the values of two attributes, *years of services* and *salary*. So a good estimation value to select for the *salary* value from the fuzzy range is some value, which is close to $33,000.

In summary, fuzzy reasoning is flexible and should work on a case by case basis. If the size of resulting range after the above fuzzy reasoning is small enough or the variances of the elements in the range is small enough, then a value close to the average of all the elements in that range can be used as a fuzzy value. Otherwise, the fuzzy reasoning unit needs input from the Semantic Reasoning Unit in order to give useable estimated value.

3. Supplemental Fuzzy Rule Unit

Fuzzy reasoning unit may need additional information in order to generate the fuzzy values. The rules stored in this unit fulfill this need and those rules can be some supplementation or explanation for the fuzzy dependency relationships. These specifications are consulted by the fuzzy reasoning unit to get more insights into the logical connections among data items in a database. These rules must be associated with fuzzy dependency attributes and can be very flexible. One rule, for example, may state that for a fuzzy dependency X=>(F)Y, different Y values in different tuples tend to be close to the same end of the range for the corresponding X value. This is different from case 6 in the previous section, where two attributes in the same tuple have values at the same end of their ranges respectively. Here, one attribute in multiple tuples demonstrates a certain pattern. For instance, if the *salary* value for an *associate professor* in a tuple was detected as damaged and we observe from other tuples in the same database table that the salary values for *assistant professors* and *professors* tend to be at their high ends of the ranges, we will pick up a value (or a small set of values) in the range of *salary* for the *associate professor*, which is close to the high end to be an approximated value because we assume our target employee is not an exception.

Another explanation may indicate the relationship between the changes of two attribute values. For example, for a fuzzy dependency relationship between *years of service* and *salary*, a stored observation may indicate the increase of *years of service* by 1 (year) usually leads to *salary* increase by $1,000, $1,500, and $2,000 for an assistant professor, associate professor, and professor respectively. This observation would help in selecting a value from the *salary* range for given values of *years of service* in a tuple.

4. Semantic Reasoning Unit

This unit works together with the fuzzy dependency unit to provide usable values. Reasoning can take many forms based on available information, ranging from simple calculation, to statistical deduction, to trend predication, to simply backup data lookup. The Semantic Reasoning Unit is an independent entity built on existing techniques and it contains rules and statistical data summarized from the database. We separate these rules and supplemental fuzzy rules as discussed earlier because these rules have broader coverage while the fuzzy rules must be associated with certain fuzzy dependency relationships. These rules or schemes could be familiar patterns or properties in the database. For instance, in a database the rule for the purpose of semantic reasoning may specify that any student ID must start with the last two digits of the year when the student was enrolled and the following four digits of the ID are related to the student's birthday or social security number. In another commercial product table, the product ID takes the form of five digits followed by a dash then followed by another digit. The rule specifies that the sum of the first five digits must be divisible by the last digit.

If we observe the student's classification (freshman, junior, etc.,), we might be able to calculate the student's enrollment year (based on classification and current year) and hence will be able to provide the first two digits of the student ID. For the following four digits of the Student ID, we might search for information about the student's birthday or social security number.

Another approach is to analyze the static properties of the corrupted data items. In such a case, for example, let's assume that a value such as a person's social security number is damaged. This value is relatively very static over the time. In very rare situation, do people change their social security numbers. An estimation scheme for such a damaged data item would be to access the backup copies to retrieve the value.

Our last example is about the salary value in a database tuple, which is considered damaged. If from other sources we can retrieve information about the employer total expense on employee payroll, a simple calculation using all other employees' salaries would generate a very good approximation for the damaged value.

5. Usage Identifier

This unit is to evaluate the usage level of each generated fuzzy value. This usage parameter has two contributions: the accuracy of a fuzzy value and the risk of using the value if it is inaccurate. The first one is the accuracy probability primarily based on the size of narrowed fuzzy range and other semantic probability information. This value is in the range of 0-1, with 1 indicating 'utmost accuracy' and 0 denoting 'definite inaccuracy'. Regarding the second contribution, one must note that the risk of using a fuzzy value comes from not only its nature of lacking accuracy but also the potential of further propagations of damage by valid transactions as they read the estimated but inaccurate data. Risks of using inaccurate values are determined by such factors as data dependency intensity on the approximated value by other data items and actual read frequencies of the approximated value. This risk value of using an inaccurate data item can be quantities in the range of 0-1, with 0 denoting 'most risky' and 1 representing 'no risk at all'. For a given data item, the confidence level and risk level are not totally separated. In many cases, they are related to each other. For instance, if a value has an accuracy level of 1 and risk level of 0, then the usage is still 1 (and not 0 if calculated solely based on the product of the given accuracy and risk values). We introduce a parameter to adjust the product of the confidence level and risk value to calculate a usage value for a fuzzy data b using the following formula:

Usage (b) = Conf (b) * Risk (b) * p

Where conf (b) is the usage level of b and Risk (b) is the risk level of using b, and P is an adjusting parameter, which can be supplied by an expert such as the database administrator.

6. Value Finalizer

This unit accepts usage levels of the generated fuzzy values and compares the latter with the administrator-supplied cutoff value. If the usage level is greater than the threshold, the unit supplies the fuzzy value (along with its usage level) to the database scheduler (or awaiting transactions). This unit can also be equipped with fuzzy value testing capability to further analyze the usage of the generated values.

B. Flow Chart for the Fuzzy Value Generator

The following steps are usually observed in a fuzzy value generator. We assume that a data item $b \in D(Y)$ in a tuple T_b has been detected as damaged.

1. The Fuzzy Check and Data Fetch Unit looks up the fuzzy dependency storage and identifies the attributes on which Y fuzzily depends. For each identified attribute, corresponding values in T_b are fetched from the database.
2. The Fuzzy Check and Data Fetch Unit sends relevant data items to the Fuzzy Reasoning Unit. If there are more instructions from the unit, it repeats steps 1 and 2.
3. The Fuzzy Reasoning Unit performs fuzzy reasoning and, if possible, interacts with the Semantic Reasoning Unit to generate a set small enough to be able to pick a reasonable value. If it establishes that there is no need to contact the Semantic Reasoning Unit, the workflow skips to step 5.
4. The Semantic Reasoning Unit does reasoning and sends its result to the Fuzzy Reasoning Unit. At the same time, it sends confidence parameter about its reasoning to the Usage Identifier. This unit has its own mechanisms to collect database characteristic data and store and generate rules.
5. The Fuzzy Reasoning Unit generates a fuzzy value (if possible) based on the narrowed range and sends this data to the Value Finalizer. It

also sends the confidence parameter about its reasoning to the Usage Identifier.

6. The Usage Identifier evaluates the risk of the value being used if that value is not the exact value. Then along with the received confidence information from the Fuzzy and Semantic Reasoning units, it generates a usage parameter for the fuzzy value. Then it sends this information to the Value Finalizer.
7. The Value Finalizer compares the usage parameter with a pre-defined threshold for a generated fuzzy value to determine if it is usable. If not, it discards the generated fuzzy value and informs the system that no suitable value can be provided for the damage data item.

V. CONCLUSION

In this paper, we have defined the notion of fuzzy dependency and used it as reasoning foundation to build a fuzzy recovery model. We have argued that our model helps perform quick recovery after an electronic attack on a database system. The reason for our model achieving faster recovery than any existing damage recovery model is that the latter requires intensive search of database logs based on data or transaction dependencies. Our model uses pre-derived rules to provide satisfactory values of damaged items. Querying fuzzy rules is much quicker than using other statistical tools and mathematical functions. This quick recovery is desired in many real-time applications, which do not require precise values of data items rather need faster data access. However, as with any fuzzy model, the fuzzy dependency method cannot guarantee absolute accuracy of data values and, hence, must be supplemented with semantic reasoning.

ACKNOWLEDGEMENT

We are thankful to Dr. Robert. L. Herklotz for his support, which made this work possible.

VI. REFERENCES

[1] H. Darwen, C. Date, "The role of functional dependencies in query decomposition." In Relational Database Writings 1989-1991, Addison Wesley, 1992

[2] R. Elmasri, S. B. Navathe, "Fundamentals of Database Systems", Addison-Wesley Publishing Company, 1994

[3] B. Thalheim, "Dependencies in Relational Databases", Teubner Leipzig, 1991

[4] B. Panda and S. Tripathy, "Data Dependency Logging for Defensive Information Warfare", In proceedings of the 2000 ACM Symposium on Applied Computing, Como, Italy, 2000

[5] P. Amman, S. Jajodia, C. D. McCollum, and B. Blaustein, "Surviving Information Warfare Attacks on Databases", In proceedings of the IEEE Symposium on Security and Privacy, 1997

[6] P. Liu and X. Hao, "Efficient Damage Assessment and Repair in Resilient Distributed Database Systems", In proceedings of the 15th Annual IFIP WG 11.3 Working Conference on Database and Applications Security, Canada, June 11-13, 2001

[7] B. Panda and S. Patnaik, "A Recovery Model for Defensive Information Warfare", In proceedings of the 9th International Conference on Management of Data, India, December 1998

[8] P. Ragothaman and B. Panda, "Modeling and Analyzing Transaction Logging Protocols for Effective Damage Assessment", In proceedings of the 16th Annual IFIP WG 11.3 Working Conf. on Data and Application Security, King's College, Univ. of Cambridge, UK, July 2002

[9] Ciprian Tutu Yair Amir, "From total order to database replication", In 22nd International Conference on Distributed Computing Systems (ICDCS'02), Vienna, Austria, 1992

[10] Anish Arora, Sandeep Kulkarni, "Designing masking fault-tolerance via nonmasking fault-tolerance", IEEE Transaction on Software Engineering, June 1998

[11] G. Morgan, P. Ezhilchelvan, "Polices for using replica groups and their effectiveness over the Internet." In Proceedings 2nd International COST264 Workshop on Networked Group Communication (NGC 2000), Palo Alto, California, 2000

[12] A. Valsangkar and B. Panda, "An Architecture for Making Data Available Ceaselessly During Recovery", In proceedings of the 2003 IEEE Workshop on Information Assurance, United States Military Academy, West Point, NY, June 2003