The Multilevel Relational (MLR) Data Model

RAVI SANDHU and FANG CHEN
George Mason University

Many multilevel relational models have been proposed; different models offer different advantages. In this paper,we adapt and refine several of the best ideas from previous models and add new ones to build the new Multilevel Relational (MLR) data model. MLR provides multilevel relations with element-level labeling as a natural extension of the traditional relational data model. MLR introduces several new concepts (notably, data-borrow integrity and the UPLEVEL statement) and significantly redefines existing concepts (polyinstantiation and referential integrity as well as data manipulation operations). A central contribution of this paper is proofs of soundness, completeness, and security of MLR. A new *data-based* semantics is given for the MLR data model by combining ideas from SeaView, belief-based semantics, and LDV. This new semantics has the advantages of both eliminating ambiguity and retaining upward information flow. MLR is secure, unambiguous, and powerful. It has five integrity properties and five operations for manipulating multilevel relations. Soundness, completeness, and security show that any of the five database manipulation operations will keep database states legal (i.e., satisfy all integrity properties), that every legal database state can be constructed, and that MLR is noninterfering. The expressive power of MLR also compares favorably with several other models.

Categories and Subject Descriptors: H.2.0 [**Database Management**]: General—*Security, integrity, and protection*; H.2.7 [**Database Management**]: Database Administration

General Terms: Security

Additional Key Words and Phrases: Access control, confidentiality, multilevel security, polyinstantiation

1. INTRODUCTION

In multilevel data models, data items and subjects have their own access classes (or levels), e.g., TS (Top Secret), S (Secret), U (Unclassified), etc., known as classifications and clearances. Access by subjects is restricted by mandatory access controls, roughly expressed as "no read up, no write down," to follow the well-known Bell and LaPadula [1975] model, and its variations described by Sandhu [1993]. However, from a security standpoint, this restriction should be strengthened by a further requirement:

Authors' address: Information and Software Engineering Department, George Mason University, Mail Stop 4A4, Fairfax, VA 22033; email: sandhu@gmu.edu; URL:www.list.gmu.edu.

operations from any given level should not be accepted or rejected due to existence or absence of any higher level data, otherwise some signaling channels for leakage of high-level data (i.e., indirect methods of communication from higher-level processes to lower-level ones) will occur [Jajodia and Sandhu 1991]).

Many multilevel relational data models have been proposed in the literature, for example, SeaView [Denning et al. 1988;Lunt et al. 1990]; LDV [Haigh et al. 1991]; and those proposed by Sandhu-Jajodia [Sandhu and Jajodia 1991; 1993]; Jajodia-Sandhu [1991;1993]; and by Smith-Winslett [1992], and so on. Each of these models has its innovations and strong points (e.g., the integrity properties in SeaView and in the Sandhu-Jajodia model, the belief-based semantics of the Smith-Winslett model, the multilevel manipulation of the Jajodia-Sandhu model, the derive option of LDV, etc.).

How can the best features of these models be reconciled and unified into a secure, unambiguous, and powerful multilevel relational data model? We confront this question directly by presenting the Multilevel Relational (MLR) data model. MLR introduces several new concepts, notably data-borrow integrity and the UPLEVEL statement. Other concepts adapted from previous models are significantly redefined, notably polyinstantiation and referential integrity, as well as data manipulation statements. The end result is a simple, flexible, and powerful model. It contains five integrity properties and five operations for manipulating multilevel relations. Moreover, MLR eliminates ambiguity and retains upward information flow.

MLR is substantially based on the data model proposed by Sandhu and Jajodia [1993]. Many aspects of that model are in turn derived from SeaView [Denning et al. 1998]. The most significant difference between the Sandhu-Jajodia and SeaView models is the requirement, introduced in Sandhu and Jajodia [1993], that there can be at most one tuple in each access class for a given entity. This gives us the simplicity of tuple-level labeling, combined with the flexibility of element-level labeling. There are also several other subtle, but very important, differences in the precise formulation of various properties.

There are two major problems left unsolved in the Sandhu and Jajodia [1993] model, *semantic ambiguity* and *operational incompleteness*. To illustrate semantic ambiguity, consider the following relation SOD(SHIP, OBJ, DEST) where SHIP is the primary key and the security classifications are assigned at the granularity of individual data elements. OBJ and DEST are abbreviations for OBJECTIVE and DESTINATION, respectively.

SHIP		OBJ		DEST		TC
Enterprise	U	Spying	S	Talos	U	S
Enterprise	U	Exploration	U	Talos	U	U

TC is an abbreviation for TUPLE-CLASS. The label in the TC attribute applies to the entire tuple. It must dominate the labels or each element of

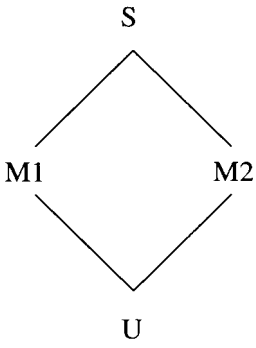

Fig. 1. A partially ordered lattice.

the tuple. We assume the reader is familiar with basic aspects of lattice-based access control [Sandhu 1993], and we use the conventional labels TS $>$ S $>$ C $>$ U. Say that the data in tuples with TC at U and S are, respectively, accepted as the real data by subjects at levels U and S.[1] However, what is the data accepted by subjects at level TS? In the Sandhu-Jajodia model, absence of a tuple at TS means there is no additional data at this level. But there are both tuples at U and S in this relation. Do subjects at TS take the values from S or from U? Is it necessary to force them to choose the one at the higher security class? Are there situations in which a subject at TS should accept values from the tuple at U rather than S?

Taking another more general example of semantic ambiguity, let M_1 and M_2 be incomparable labels whose least upper bound is S and greatest lower bound is U, as shown in Figure 1 (we use this lattice throughout the paper). Consider the following relation.

SHIP		OBJ		DEST		TC
Enterprise	U	Mining	M_1	Talos	U	M_1
Enterprise	U	Spying	M_2	Talos	U	M_2
Enterprise	U	Exploration	U	Talos	U	U

Which OBJ value is accepted by subjects at S? Mining or Spying or even Exploration? Again, is it necessary to force S-subjects to accept data from any specific level?

As for operational incompleteness, how can a tuple whose individual classification attributes are at U, M_1, and M_2 be instantiated by a subject at S? In fact, in previous models, there is no way for a subject at, say, S to add tuples like the following.

[1]Note that the same individual user can have subjects at U and at S, so it is proper to talk about data accepted by subjects at a given level rather than by users.

SHIP		OBJ		DEST		TC
Enterprise	U	Mining	M_1	Sirius	M_2	S

In order to solve these problems we integrate ideas from a number of previous models to establish the new MLR data model. To complete this integration, we introduce several new concepts and refine many old ones.

We fully define the MLR data model in this paper. The new *data-based* semantics for both data and operations is also given. Moreover, we prove that the MLR model is a sound, complete, and secure data model. These proofs show that any of the database manipulation operations provided will keep the database state legal (i.e., satisfy all integrity properties), that every legal database state can be constructed, and the MLR data model is secure, in that all information flow is upwards in the security lattice. The expressive power of the MLR model is discussed by comparing it with several other models.

The rest of the paper is organized as follows. In Sections 2 and 3 we define the MLR model and its data semantics. The expressive power of MLR is discussed in Section 4. Section 5 addresses the model's data manipulation operations. Sections 6, 7, and 8 prove, respectively, the soundness, completeness, and security of the MLR data model. In Section 9, we summarize our major contributions and give our conclusions.

2. THE BASIC MODEL

We define the MLR model in three parts. In this section we formally define the so-called basic model. We give a data interpretation for the basic model as a part of the data semantics and describe how the MLR data model corresponds to the real world. We also discuss the practicality of the model. The next section describes the five integrity properties of MLR. Discussion of data manipulation in MLR is deferred to Section 5.

2.1 Model Definition

A multilevel *relation* consists of the following two parts.

Definition 2.1 A multilevel *relation scheme* is denoted by $R(A_1, C_1, A_2, C_2, \ldots, A_n, C_n, TC)$, where each A_i is a *data attribute* over domain D_i, each C_i is a *classification attribute* for A_i, and TC is the *tuple-class attribute*. The domain of each C_i is specified by a set $\{L_i \ldots H_i\}$ containing all access classes of a sublattice ranging from L_i up to H_i ($H_i \geq L_i$).[2] The domain of TC is $\cup_{i=1}^{n} (\{L_i \ldots H\})$, where H is system high and $\cup$ stands for set union.

Definition 2.2 A *relation instance*, denoted by $r(A_1, C_1, A_2, C_2, \ldots, A_n, C_n, TC)$, is a set of distinct tuples of the form $(a_1, c_1, a_2, c_2, \ldots, a_n, c_n, tc)$, where each $a_i \in D_i$ and $c_i \in \{L_i \ldots H_i\}$, or $a_i =$ null and $c_i \in$

[2]More generally, the domain of C_i can be any arbitrary subset of access classes. This is a straightforward generalization.

$\{L_i \ldots H_i\} \cup$ null, and $tc \geq$ lub $\{c_i \mid c_i \neq$ null: $i = 1 \ldots n\}$. Here lub denotes the least upper bound.[3]

We assume that there is a user-specified *apparent primary key* AK consisting of a subset of the data attributes A_i. In general AK will consist of multiple attributes. In Section 3 we see that all attributes in AK are required to have the same classification level. Meanwhile, each A_i can also be seen as a group of attributes with an identical classification level. In order to simplify our notation, we use A_1 as synonymous to AK, i.e., A_1 and AK both denote the apparent primary key. We also assume that the relation scheme is itself classified at the greatest lower bound of L_i ($i = 1 \ldots n$).[4] A tuple whose tuple class is c is said to be a c-tuple, while a subject whose clearance is c is said to be a c-subject.

There are two subtle differences in these definitions relative to Sandhu and Jajodia [1992]. First, the domain of TC is different from that in Sandhu and Jajodia [1993], which is {lub $\{L_i : i = 1 \ldots n\}$ ... lub $\{H_i : i = 1 \ldots n\}$}. Second, the Sandhu and Jajodia [1993] model does not allow classification attributes to be null. These changes are primarily for the INSERT semantics (Section 5.2), because now the tuple

SHIP		OBJ		DEST		TC
Enterprise	U	Exploration	U	null	null	U

can be inserted into the relation by a U-subject, even if the domain of the classification attribute for DEST is limited to, say, S ... TS, in which case,

SHIP		OBJ		DEST		TC
Enterprise	U	Exploration	U	null	U	U

is not allowed.

In MLR every relation has only one relation instance at any time. As we see in Section 2, subjects at different levels may have different views of the instance. Previous models have defined a relation as having a different relation instance at each level. This modification is simply a technical one for convenience of semantic description.

We define $tc \geq$ lub $\{c_i \mid c_i \neq$ null: $i = 1 \ldots n\}$ in the same way as Sandhu and Jajodia [1993], by which

[3]As we will see in Section 3, there will always be some c_i that is nonnull, therefore, lub $\{ c_i \mid c_i \neq$ null: $i = 1 \ldots n\}$ is always well defined.
[4]Scheme classification in multilevel relations remains an open research issue.

SHIP		OBJ		DEST		TC
Enterprise	U	Exploration	U	Talos	U	S
Enterprise	U	Exploration	U	Talos	U	U

is allowed and has different meanings than

SHIP		OBJ		DEST		TC
Enterprise	U	Exploration	S	Talos	S	S
Enterprise	U	Exploration	U	Talos	U	U

In fact, the former says S-subjects borrow from U-subjects the data currently owned by U-subjects, and the data could be changed by U-subjects; whereas the latter means S-subjects have their own data for OBJ and DEST, which could not be changed by U-subjects. The value equivalence of OBJ and DEST in the latter case is just coincidental; data interpretation is discussed at length in Section 2.2.

The following definition is taken directly from the traditional relational data model.

Definition 2.3 A *database* is a collection of relations. A *database state* is a collection of all relation instances of a database at a particular time.

2.2 Data Interpretation

The intuitive ideas of our *data-based* semantics are as follows.

(1) The data accepted by subjects at one level consist of two parts: the data owned by them and the data borrowed from lower-level subjects. The latter can be changed by the lower-level subjects who own them.

(2) The data that a subject can see are those accepted by subjects at its level or at levels below it.

Here data is treated as a common resource shared by subjects at a given level. For simplicity, other access controls, such as discretionary or role-based, are omitted. The basic ideas come from combining belief-based semantics [Smith and Winslett [1992] and the LDV model [Haigh et al. 1991]. In Sections 3 and 5 we will show how these ideas lead to a complete semantics for MLR.

We consider the following example first, where SHIP is assumed to be AK.

SHIP		OBJ		DEST		TC
Enterprise	S	Spying	S	Rigel	S	S
Enterprise	U	Exploration	U	null	S	TS
Enterprise	U	Exploration	U	Talos	U	U

This example describes two entities, (Enterprise, S) and (Enterprise, U) which are, respectively, created by a U-subject and an S-subject, and can only be deleted by U-subjects and S-subjects, respectively. The TS-tuple, i.e., the tuple with TC as TS, is added by a TS-subject and all data in it are accepted by TS-subjects. Let us take a closer look at the entity (Enterprise, U). The U-tuple, i.e., the tuple with TC as U, is the base tuple, which can only be deleted when the entire entity is to be deleted, including the TS-tuple. TS-subjects can see both the TS-tuple and the U-tuple, whereas U-subjects can only see the U-tuple. The OBJ value of the TS-tuple, Exploration, is borrowed from U-subjects, and is subject to change when that of the U-tuple is changed or deleted. The null in the TS-tuple means that TS-subjects are expecting to borrow DEST data from S-subjects, but there is no DEST data currently owned by them for this entity. Also, absence of an S-tuple in this entity indicates that this entity is not accepted by S-subjects.

Entity polyinstantiation and element polyinstantiation are also illustrated here. They are two types of *polyinstantiation*, a technique to prevent inference violations. Entity polyinstantiation occurs when a relation contains multiple tuples with the same AK values but different C_{AK} values. With element polyinstantiation, a relation contains two or more tuples with identical AK and C_{AK} values, but with different values for one or more A_i's ($2 \leq i \leq n$). In this example, for the AK value Enterprise, there are two C_{AK} values S and U; while for the same (Enterprise, U), there are two different DEST values: null and Talos.

We now give a formal description of the above intuitive ideas. For all instances $r(A_1, C_1, A_2, C_2, \ldots, A_n, C_n, TC)$ and for all tuples $t \in r$, the data are interpreted as follows.

(1) Apparent Primary Key A_1 and its Classification Attribute C_1
   —$t[A_1, C_1]$ identifies an *entity* in r and also gives the class level of the entity.
   —$t[C_1] = c_1$ means the entity is created by a c_1-subject and can only be deleted by c_1-subjects. The entity is called a c_1-entity.

(2) Tuple-Class Attribute TC
   —$t[TC] = tc$ with $t[C_1] = c_1$ means that
      —t is added by a tc-subject and all data in t are accepted by tc-subjects. Absence of t means the c_1-entity is not accepted by tc-subjects.
      —t can only be seen by subjects with level $c' \geq tc$. In other words, all a c'-subject can see are tuples t' with $t'[TC] \leq c'$.
      —t can be deleted either by tc-subjects, or by c_1-subjects in cases where the entire entity is deleted.[5]

[5] Note that discretionary or nondiscretionary access controls can be used to control which subject can delete the tuple.

—When $t[TC] = t[C_1]$, t is the *base tuple* of the entity, all tuples $t' \in r$ such that $t'[A_1, C_1] = t[A_1, C_1]$ are based on t, and t can only be deleted when the entire entity is to be deleted.

(3) Data Attribute A_i and Classification Attribute C_i ($2 \leq i \leq n$)
—$t[A_i, C_i]$ with $t[C_i] = c_i$ and $t[TC] = tc$ ($c_i \leq tc$) indicates that
—the data $t[A_i]$ accepted by tc-subjects are currently owned by c_i-subjects.
—$t[A_i, C_i]$ can be maintained (updated) either by c_i-subjects or by tc-subjects.
—When $t[C_i] < t[TC]$, $t[A_i] \neq$ null is borrowed from the $t'[A_i]$ of t' which has $t'[A_1, C_1] = t[A_1, C_1] \wedge t'[TC] = t'[C_i] = t[C_i]$,[6] and is subject to change when $t'[A_i, C_i]$ is changed or t' is deleted.

(4) Null Value
—$t[A_i, C_i] =$ [null, c_i] ($c_i < tc$) means that for attribute A_i, tc-subjects expect to borrow data owned by c_i-subjects, however, no data are currently owned by them.
—Both $t[A_i, C_i] =$ [null, null] and $t[A_i, C_i] =$ [null, tc] means that for A_i no data are available at level tc. The [null, null] case applies when $tc \notin \{L_i \ldots H_i\}$; the [null, tc] case applies otherwise .

2.3 MLR Model vs Real World

How a data model corresponds to the real world is an essential issue, both in evaluating the model and using the model to design databases. A basic question is: What do data in a model represent? Consider a traditional database example first. What does the following tuple mean?

SHIP	OBJ	DEST
Enterprise	Exploration	Talos

One answer is that in the real world there is a ship called Enterprise that will go to explore Talos. Another answer is that it stands for an opinion held by some people who think a ship called Enterprise will go to explore Talos. These two answers are very different. The first one states that the tuple represents a fact in the real world, which should definitely be true; whereas the second states that it is just somebody's idea, which may or may not be true in the real world. Although in many cases these two answers may not conflict with each other, still they are different, and we feel the second one is more reasonable. The correctness of data in a database really depends on how the data are obtained. It is ultimately a problem of cognition. We can also say that the second answer is more general than the first, in the sense it includes both true cognition and false cognition of the real world.

[6]This issue will be discussed in greater detail in Sections 3 and 5.

Now let us consider an MLR example. What does the following instance mean?

SHIP		OBJ		DEST		TC
Enterprise	U	Spying	S	Talos	U	TS
Enterprise	U	Spying	S	Rigel	S	S
Enterprise	U	Exploration	U	Talos	U	U

If we use an approach similar to the second answer above, we can interpret this relation as representing opinions of three different groups of people with security levels TS, S, and U.[7] For brevity, we denote these three groups of people as *TS*s, *S*s, and *U*s. *S*s may think that their data are more correct than that of *U*s, however, for some data, e.g. DEST, *TS*s may nevertheless trust *U*s rather than *S*s. Here both *TS*s and *S*s trust *U*s on the existence of Enterprise, which means all *TS*s, *S*s, and *U*s are talking about the same Enterprise, so *TS*s can trust either *S*s or *U*s for some of the data about Enterprise, say, OBJ or DEST. Generally, if there are polyinstantiation tuples with same A_1, C_1 value, they refer to the same real world entity. This is called *element polyinstantiation*.

As an *entity-polyinstantiation case*, consider the following instance.

SHIP		OBJ		DEST		TC
Enterprise	U	Spying	TS	Talos	U	TS
Enterprise	S	Spying	S	Rigel	S	S
Enterprise	U	Exploration	U	Talos	U	U

Here *TS* trusts *U* that a ship called Enterprise exists, therefore it clearly knows that its Enterprise is exactly the same as that meant by *U*. On the other hand, *S* has its own opinion, which is totally independent from that of *U*, even on the existence of a ship called Enterprise. There could be a case where *S* addresses its opinion before *U* does, and therefore has no idea about what *U* might say later. Of course, *S* can change its opinion if it finds out subsequently it can trust *U* on the existence of Enterprise, and that its Enterprise is exactly what *U* means also. Generally, if there are polyinstantiation tuples with same A_1 but different C_1, they could refer to the

[7]So far we have talked about subjects who accept data at a given level. We can extend this concept to users or people by understanding the opinion of a user to be that of the subject at the user's clearance, i.e., the highest labeled subject that this user can invoke. For example, a user cleared to TS can access data as a TS, S, U, or C subject and get a different view each time. The opinion of a TS user is therefore the opinion of TS subjects. However, while logged in as a S subject, the TS user will see data from a S user's perspective. In this context, the TS user is trusted to behave like an S user and not leak TS data deliberately or inadvertently.

same real world entity or to different ones. Resolving this is beyond the scope of a data model, and is basically a problem of cognition.

It is possible that TS feels there exist two different Enterprises, mentioned by U and S, and it wants to trust U and S. However, if later on TS wants to say that "Kirk is the captain of the Enterprise," it is not clear which Enterprise Kirk is the captain of, or maybe of both. Theoretically, there could be a way to deal with this problem: TS gives a nick name to each of the two ships, say Enterprise-U to the Enterprise mentioned by U and Enterprise-S to that mentioned by S, using "Kirk is the captain of Enterprise-U" and/or "Kirk is the captain of Enterprise-S" accordingly. But if there are some other groups at levels even higher than TS, they will have to know all the nick names defined by TS before they can understand and/or use TS's data. The problem becomes even worse if Enterprise-U is used in some other situation such as "Maintenance of Enterprise-U is in July" identified by both "maintenance" and "Enterprise-U", because there could be several different versions of "maintenance of Enterprise-U" accepted by some specific group at the same time, leading to recursive nick names. (This issue will be discussed further in Section 3.1.)

2.4 Practicality

As we have seen, data-based semantics takes the following ideas from belief-based semantics: for an entity at level c_1, absence of a c-tuple ($c > c_1$) means the entity is not accepted by c-subjects. In other words, in order to reference an c_1-entity, c-subjects should add a c-tuple of the entity into the relation first, even though all the data accepted by c-subjects belong to the subjects below c. Actually, this is the crucial requirement for eliminating semantic ambiguity.

This requirement would be quite acceptable if 95% of entities had different data at different levels. Unfortunately, often only 5% of entities have secret data at levels higher than unclassified; which means that for the other 95% of entities, high level tuples just repeat unclassified data. If there are m levels higher than unclassified, to reference unclassified data at every level, all these data are logically repeated m times. A naive implementation would not just waste space but also waste user time, since the repetition is explicitly performed by subjects at each level.

Under this reasonable logical model, we must face practical issues of the physical model. Fortunately, there are several techniques that deal with these problems.

To avoid wasting space, we can physically (not logically) expand the tuple-class attribute TC to be a tuple-class attribute set, containing several levels whose data are exactly the same in all A_1, C_1, . . . , A_n, C_n. Hence, instead of keeping several repeated tuples in the database, we can physically just keep one tuple and indicate all levels that accept it in the TC set. For example, $(a_1, c_1, \ldots, a_n, c_n, \{tc_1, \ldots, tc_m\})$ can stand for $(a_1, c_1,$

$\ldots, a_n, c_n, tc_1), \ldots, (a_1, c_1, \ldots, a_n, c_n, tc_m)$. If later on some subject needs to change some data in it, it may be divided to several tuples with some different A_i or C_i values.

To save effort in explicitly accepting a tuple at each level above its tuple class, we can allow subjects, or the database administrator, to set some defaults. For example, c-subjects can set defaults such as that: for all entities in R, all data owned by c'-subjects ($c' < c$) are accepted. That is to say, whenever a c'-subject inserts an entity into R, a c-tuple of the entity is automatically created and all data of the c-tuple are borrowed from c'-subjects (we may possibly just put c into the TC set of the c'-tuple). For single-level relations, i.e., for any i, j such that $1 \leq i, j \leq n$, $L_i = L_j = H_i = H_j$, this approach could be very useful. For example, a single-level relation could be used to keep common knowledge such as the city name, location, area, climate, etc, which can be used by subjects at any level. This needs to be done with some care to make sure that inconsistencies do not arise (Section 3).

It is easy to see that there are many variations of these two basic approaches. Physical issues may also depend on the storage strategies used to construct the DBMS. Since the MLR model is a logical data model, further discussion about physical issues is outside the scope of this paper. Our objective here is to simply sketch the feasibility argument.

3. INTEGRITY PROPERTIES

There are five integrity properties in the MLR data model, of which entity integrity and foreign key integrity are taken from the original SeaView model; polyinstantiation integrity and referential integrity are significantly redefined by the authors; and data-borrow integrity is introduced. In particular, the polyinstantiation integrity property given here is much more general than that of SeaView[Denning et al. 1988] or the Sandhu and Jajodia [1993] model, in that it takes care of both entity and element polyinstantiation. The redefined referential integrity property also deals with some new reference problems arising from data-borrow.

3.1 Entity Integrity

The Entity Integrity property given here was first proposed by SeaView [Denning et al. 1988], and has stayed unchanged in most work since then.

Property 1. **[Entity Integrity (EI)]** Let AK be the apparent primary key of R. An instance r of a multilevel relation R satisfies entity integrity if and only if for all $t \in r$:

(1) $A_i \in AK$ f $t[A_i] \neq$ null;

(2) $A_i, A_j \in AK$ f $t[C_i] = t[C_j]$; and

(3) $A_i \notin AK, A_j \in AK$ f $t[C_i] \geq t[C_j]$ or $t[C_i] =$ null.

The first requirement is exactly the definition of entity integrity in the traditional relational model, which ensures that no tuple in r has a null value for any attribute in AK. The second requirement says that all attributes in AK have the same classification in a tuple, i.e., AK is *uniformly classified*, and so we can define C_{AK} to be the classification of the apparent primary key AK. The final requirement states that in any tuple the class of nonkey attributes must dominate C_{AK}.

The requirement of uniformly classified AK in MLR is consistent with the intuitive ideas of data-based semantics. When a subject at, say, level S addresses an independent opinion, all the data are classified at its level. Later on, if another subject at, say, level TS needs to borrow some data from S, the TS-subject should address the same entity as that of S. In MLR we use AK attributes as well as their classification attributes to identify an entity, so in order to address the same entity the TS-subject has to use exactly the same values for the classification attributes of AK as that of S, leading to uniformly classified AK.

3.2 Polyinstantiation Integrity

Polyinstantiation integrity is required by many models. However, our definition is much more general than previous ones. It is the first to treat both entity polyinstantiation and element polyinstantiation in a unified manner.

Property 2. **[Polyinstantiation Integrity (PI)]** An instance r of a multilevel relation R satisfies polyinstantiation integrity if and only if for $1 \leq i \leq n$,

(1) $A_1, TC \ 3 \ C_i$; and

(2) $A_1, C_1, C_i \ 3 \ A_i$.

The second requirement is the original polyinstantiation integrity necessary for both the SeaView and Sandhu-Jajodia models, which says that the real primary key of the relation is $A_1, C_1, C_2, \ldots, C_n$. The first requirement is derived from the following two conditions (as well as the second requirement above):

(a) $A_1, TC \ 3 \ C_1$; and

(b) $A_1, C_1, TC \ 3 \ A_i, C_i$.

In Sandhu et al. [1990], (b) is an example of tuple-class polyinstantiation integrity,[8] which says that every entity in a relation can have at most one tuple for every access class; whereas (a) is new (and is discussed below).

[8]It has been misexpressed in many places as $A_1, C_1, TC \ 3 \ A_i$, which is too weak, since, for example, it allows tuples <Enterprise, S, Spying, S, Rigel, S, TS> and <Enterprise, S, Spying, TS, Rigel, S, TS> in the same relation.

The new property (a) can be called an example of entity polyinstantiation integrity. The intuitive idea of this property is that there could be several entities in a relation with the same *AK* value, but subjects at any security level can accept at most one entity with that *AK* value. For example,

SHIP		OBJ		DEST		TC
Enterprise	U	Spying	TS	Talos	U	TS
Enterprise	S	Mining	S	Rigel	S	S
Enterprise	U	Exploration	U	Talos	U	U

is allowed when at each level TS, S, or U, subjects accept only one entity with *AK* value as Enterprise. However,

SHIP		OBJ		DEST		TC
Enterprise	S	Spying	S	Rigel	S	S
Enterprise	U	Mining	S	Talos	U	S
Enterprise	U	Exploration	U	Talos	U	U

is not allowed when there are two entities with the same *AK* value Enterprise (Enterprise, S) and (Enterprise, U), accepted at level S. S-subjects could choose either of them, but not both, to avoid semantic confusion.

This is very different from the no entity polyinstantiation integrity of Sandhu and Jajodia [1993]. In our case, there can be no downward information leakage, since entity polyinstantiation is allowed across security levels, and therefore no insertion is rejected due to existing entity polyinstantiation at higher levels (see Section 5 for details). Also, there is no ambiguity, since no entity polyinstantiation is allowed in what is accepted by subjects at any particular security level.

MLR does not support nick name definition (Section 2) because we feel using nick names is very confusing. This confusion exists in the real world. Sandhu and Jajodia [1992] list several approaches to eliminate entity polyinstantiation. For example, a TS user is allowed to log in as an S-subject and rename the lower level entity. This can solve some problem with entity polyinstantiation but may leak some information.

3.3 Data-Borrow Integrity

The newly introduced data-borrow integrity is a key property in our data-based semantics. Allowing data-borrow ensures that the MLR data model can retain upward information flow. Changes to data at a lower level can be automatically propagated to higher levels. This is somewhat like a "write-up" operation, except that existing higher level data elements cannot be overwritten.

The data-borrow integrity property is expressed as follows.

Property 3. **[Data-Borrow Integrity (DBI)]** An instance r of a multilevel relation R satisfies data-borrow integrity if and only if for all $t \in r$ and $1 \leq i \leq n$, if $t[A_i] \neq$ null $\wedge$ $t[C_i] < t[TC]$, there exists $t' \in r$ such that $t'[A_1, C_1] = t[A_1, C_1] \wedge t'[TC] = t'[C_i] = t[C_i] \wedge t'[A_i] = t[A_i]$.

This is based on the following idea in data-based semantics: c-tuple contains all the data accepted (but not necessarily owned) by c-subjects; absence of a c-tuple means that the entity is not accepted by c-subjects.

Consider the following relation instances:

SHIP		OBJ		DEST		TC
Enterprise	U	Exploration	U	Rigel	S	S
Enterprise	U	Exploration	U	Talos	U	U

SHIP		OBJ		DEST		TC
Enterprise	U	Exploration	U	Rigel	S	S

The former instance satisfies DBI but the latter one does not. Here DBI requires that the U-tuple, which is the source of the data Enterprise and Exploration, exist. This is because absence of a U-tuple means that U-subjects did not accept the entity Enterprise at the moment; which implies that the data once owned by U-subjects is invalid and, of course, can no longer be used by S-subjects.

Note that DBI cannot be derived from PI. For example, the second instance above does not satisfy DBI but does satisfy PI.

3.4 Foreign Key Integrity

Foreign key integrity was originally proposed by SeaView [Denning et al. 1988].

Property 4. **[Foreign Key Integrity (FKI)]** Let FK be a foreign key of the referencing relation R. An instance r of a multilevel relation R satisfies foreign key integrity if and only if for all $t \in r$, either $(\forall A_i \in FK)[t[A_i] = \text{null}]$ or $(\forall A_i \in FK)[t[A_i] \neq \text{null}]$ and

(1) $A_i, A_j \in FK$ f $t[C_i] = t[C_j]$.

The first part of this property arises from traditional relations. The motivations for the second part are similar to those for the uniform classification of apparent primary keys in EI; and similarly we can define C_{FK} to be the classification of the foreign key FK.

3.5 Referential Integrity

Referential integrity appears both in SeaView and in the Sandhu-Jajodia models. The main issue in referential integrity is avoidance of semantic

ambiguity, as discussed at length in Sandhu and Jajodia [1990]. The definition given here consists of two parts. The first part is similar to the original SeaView and the Sandhu-Jajodia definitions. However, referential ambiguity is eliminated by our data-based semantics. The second part of the definition is to rule out some abnormal cases arising from simultaneous reference and data-borrow.

Property 5. **[Referential Integrity (RI)]** Let FK_1 be a foreign key in the referencing relation R_1 with apparent primary key AK_1. Let R_2 be the referenced relation with apparent primary key AK_2. Instances r_1 of R_1 and r_2 of R_2 satisfy referential integrity if and only if

(1) for all $t_{11} \in r_1$ such that $t_{11}[FK_1] \neq$ null, there exists $t_{21} \in r_2$ such that $t_{11}[FK_1] = t_{21}[AK_2] \wedge t_{11}[TC] = t_{21}[TC] \wedge t_{11}[C_{FK_1}] \geq t_{21}[C_{AK_2}]$; and

(2) for all $t_{11}, t_{12} \in r_1$ and $t_{21}, t_{22} \in r_2$ if $t_{11}[AK_1, C_{AK_1}] = t_{12}[AK_1, C_{AK_1}] \wedge t_{11}[TC] = t_{21}[TC] \wedge t_{12}[TC] = t_{22}[TC] = t_{11}[C_{FK_1}] = t_{12}[C_{FK_1}] \wedge t_{11}[FK_1] = t_{21}[AK_2] = t_{22}[AK_2]$, then $t_{21}[AK_2, C_{AK_2}] = t_{22}[AK_2,] C_{AK_2}$.

For convenience, we refer to these two parts of RI as RI(1) and RI(2), respectively.

In traditional relations, the referential integrity property precludes the possibility of dangling references. In other words, a nonnull foreign key must have a matching tuple in the referenced relation. The requirement $t_{11}[C_{FK_1}] \geq t_{21}[C_{AK_1}]$ in the first portion of our definition is proposed by SeaView to allow only downward references. We require $t_{11}[TC] = t_{21}[TC]$ as well, which means for any level c, c-tuples can only reference c-tuples. This follows naturally from our data-based semantics: c-tuple contains all the data accepted by c-subjects; to c-subjects, absence of a c-tuple means the entity does not exist.

Consider the two examples described in Sandhu and Jajodia [1993] as an impasse between referential ambiguity and modeling power. In the first example, references between relation instances SOD and CS (CAPTAIN, SHIP) (CAPTAIN is the primary key and SHIP is a foreign key)

SHIP		OBJ		DEST		TC
Enterprise	U	Exploration	U	Talos	U	U
Enterprise	U	Spying	S	Rigel	S	S

CAPTAIN		SHIP		TC
Kirk	U	null	U	U
Kirk	U	Enterprise	S	S

were somewhat ambiguous in previous models because the S-tuple of CS referenced both the U- and the S-tuple of SOD, and there is no way to determine which one is correct. In MLR, since the S-tuple in CS can only reference the S-tuple in SOD, there is no referential ambiguity. In the second example, references between relation instances SOD and CS

SHIP		OBJ		DEST		TC
Enterprise	U	Exploration	U	Talos	U	U
Enterprise	U	Spying	S	Rigel	S	S

CAPTAIN		SHIP		TC
Kirk	U	null	U	U
Kirk	U	Enterprise	S	S

would not, in previous models, have been allowed if the restriction $t_1[C_{FK}] \geq t_2[C_{AK}]$ were replaced by $t_1[C_{FK}] = t_2[C_{AK}]$ to eliminate referential ambiguity because Enterprise in SOD has classification U rather than S in CS. In MLR, the S-tuple of CS references the S-tuple of SOD without loss of any modeling power. In both cases the meaning of (Enterprise, S) in the S-tuple in CS is that Kirk is assigned to the Enterprise accepted by S users.

The second portion of RI is to rule out anomalous cases such as

SHIP		OBJ		DEST		TC
Enterprise	U	Spying	TS	Talos	U	TS
Enterprise	S	Mining	S	Rigel	S	S
Enterprise	U	Exploration	U	Talos	U	U

CAPTAIN		SHIP		TC
Kirk	U	Enterprise	S	TS
Kirk	U	Enterprise	S	S
Kirk	U	Enterprise	U	U

There are two entities in the referenced instance SOD, (Enterprise, U) and (Enterprise, S). Within SOD, Enterprise at TS means entity (Enterprise, U) and at S means entity (Enterprise, S). However, for the referencing instance CS, Enterprise at both TS and S means entity (Enterprise, S) because TS-subjects borrow SHIP from S. Thus we have a problem that, in this database state, Enterprise at level TS has two conflicting meanings, (Enterprise, U) and (Enterprise, S)— which is certainly unacceptable.

It is clear that this anomalous case violates RI(2), because the two referencing tuples in CS at level TS and S are of the same entity (Kirk, U); and because the TS-tuple borrows the foreign key SHIP from the S-tuple,

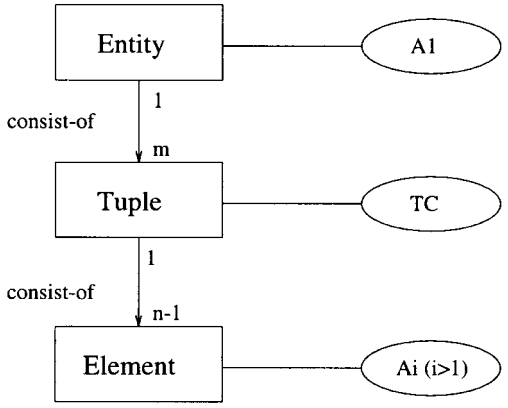

Fig. 2. The expressive power of the tuple-level labeling data model.

but the two referenced tuples in SOD, at level TS and S, respectively, are of two different entities, (Enterprise, U) and (Enterprise, S), respectively.

4. EXPRESSIVE POWER

In this section we compare the expressive power of MLR with several other data models by giving ER (Entity-Relationship) style diagrams to show how entities, tuples, and elements are related in different models. We address the tuple-level labeling model first, since it is the simplest data model and has been theoretically analyzed in Thuraisingham [1991]. The tuple-level labeling model has simple schemes, as follows:

A_1	A_2	. . .	A_n	TC

where there is only one label on the entire tuple. The expressive power of the tuple-labeling data model is shown in Figure 2. Every entity is identified by A_1 and can have at most one tuple for each classification level. Each tuple has its class level recorded in TC, and consists of $n - 1$ elements associated with A_1. The value of every element is kept in A_i ($2 \leq i \leq n$).

Note that entity polyinstantiation is not possible in the context of Figure 2 because here an entity is only identified by A_1. Fundamentally, tuple-level labeling can directly provide either element polyinstantiation or entity polyinstantiation, but not both. An alternate interpretation is shown in Figure 3, in which every entity, identified by A_1 and C_1=TC, can only have one tuple. It is obviously more useful to opt for element polyinstantiation, since this allows entities whose attributes are labeled at different classes; whereas entity polyinstantiation requires all attributes of an entity to be uniformly classified.

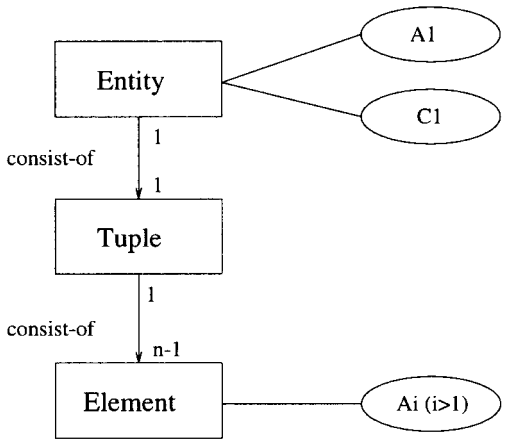

Fig. 3. An alternate interpretation of the tuple-level labeling data model.

The expressive power of the MLR data model is illustrated in Figure 4. Here an entity is identified by A_1 and C_1, and may have at most one tuple for each classification level. Each tuple has its class level recorded in TC, and consists of $n - 1$ elements. The value and owner's level of every element are kept in A_i and C_i ($2 \leq i \leq n$). It is clear that both entity polyinstantiation and element polyinstantiation are allowed here, since each entity is identified by both A_1 and C_1, and both tuple and elements have their own classification levels.

What we call the semi-tuple-level labeling data model has the following schemes:

A_1	C_1	A_2	. . .	A_n	TC

This is exactly the same as the model provided by Smith and Winslett [1992]. The expressive power of the semi-tuple-level labeling data model is shown in Figure 5. Both entity polyinstantiation and element polyinstantiation are allowed here, but there is no data-borrow. The semi-tuple-level labeling model can also be seen as a restricted case of MLR in such a way that C_i=TC for $2 \leq i \leq n$. By this restriction, the model can no longer account for the source of element data, and thereby no upward information flow by data-borrow can exist. In other words, no "write up" is allowed.

Now compare MLR with SeaView. Although the SeaView model is not grounded in belief-based semantics, we can still establish some relationship between SeaView and MLR. The expressive power of the SeaView data model is shown in Figure 6. The TC in SeaView is redundant, and can be calculated from $C_1, \ldots, C_n$. Hence, in the diagram, it is attached with a dotted line.

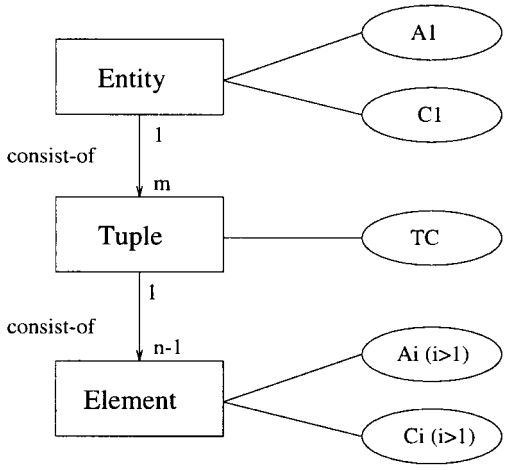

Fig. 4. The expressive power of the MLR data model.

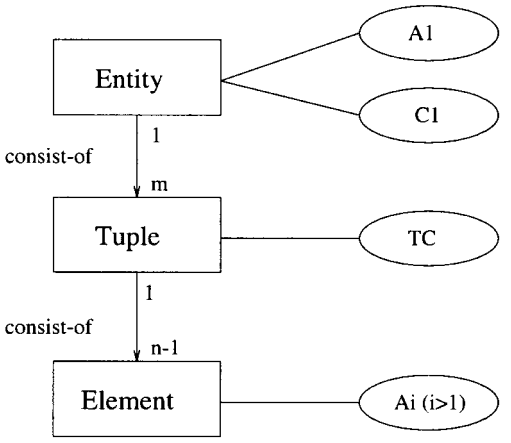

Fig. 5. The expressive power of the semi-tuple-level labeling data model.

What SeaView can do to counter the semantic ambiguity problem addressed in Section 1 is very limited. The model-theoretic semantics of Quian [1994] is slightly different from the SeaView's original "fact-based" semantics [Quian 1993], in that it integrates some ideas from the belief-based semantics to overcome semantic ambiguity. However, as long as believability is equated to visibility, it is quite possible that either believability is maximized or visibility is minimized.

As an example, let us consider an MLR instance:

SHIP		OBJ		DEST		TC
Enterprise	U	Mining	S	Talos	U	TS
Enterprise	U	Mining	S	Rigel	S	S
Enterprise	U	Exploration	U	Talos	U	U

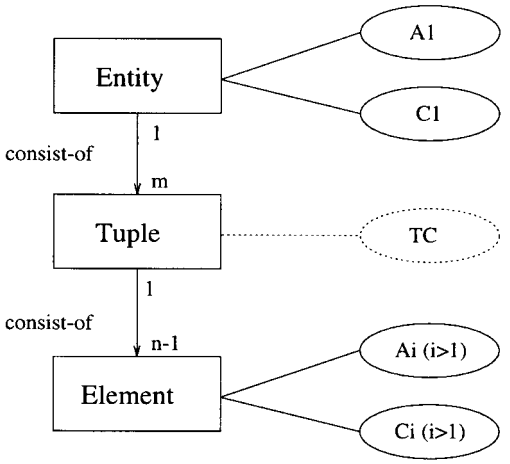

Fig. 6. The expressive power of the SeaView data model.

In this case, all data accepted by TS-subjects are borrowed from lower-level subjects. Furthermore, TS-subjects take DEST from U-subjects instead of S-subjects, and the Talos data can be changed by U-subjects. Neither SeaView nor the semi-tuple-labeling model can express this case because it concerns data-borrow. In SeaView, TS-subjects should accept the S-tuple if they do not have their own data. In semi-tuple-labeling model, TS-subjects can only accept the data owned by themselves.

In summary, we have shown that MLR is a very powerful multilevel relational data model, which can be used as a unified data model to support general MLS database design.

5. MANIPULATION

There are five data manipulation statements in the MLR data model. Four of them are the traditional SQL statements INSERT, DELETE, SELECT, and UPDATE. The fifth statement is UPLEVEL, and is new to MLR. The UPLEVEL statement is introduced to solve the manipulation incompleteness problem discussed in Section 1. Compared to the Sandhu and Jajodia [1992] model, we have redefined the semantics of the four traditional SQL statements, and have replaced PUPDATE with UPLEVEL. The SELECT statement given here is similar to that of Smith and Winslett [1992], and our intent is to provide a friendly interface that is backward-compatible to the standard SQL.

We first give several examples to show how these statements are used. After that, we present formal syntax and semantics for all these data manipulation statements and explain some important aspects. The syntax is similar to that in Jajodia et al. [1990], and the explanation given here concentrates on data-borrow and operation propagation.

5.1 Examples

First of all, we illustrate how a subject can add a tuple to an instance with some data taken from lower levels. Suppose there is a relation instance, as follows:

SHIP		OBJ		DEST		TC
Enterprise	U	Mining	M_1	Talos	U	M_1
Enterprise	U	Exploration	U	Sirius	M_2	M_2
Enterprise	U	Exploration	U	Talos	U	U

An S-subject applying to the instance the following UPLEVEL command

```
UPLEVEL    SOD
GET        OBJ FROM M1,DEST FROM M2
WHERE      SHIP = "Enterprise"
```

will give us the following result:

SHIP		OBJ		DEST		TC
Enterprise	U	Mining	M_1	Sirius	M_2	S
Enterprise	U	Mining	M_1	Talos	U	M_1
Enterprise	U	Exploration	U	Sirius	M_2	M_2
Enterprise	U	Exploration	U	Talos	U	U

An S-tuple is added to SOD, whose OBJ and DEST values are borrowed from the data owned by M_1-subjects and M_2-subjects, respectively.

Next, we illustrate the upward propagation of changes due to UPDATE, DELETE, and UPLEVEL statements. Suppose an S-subject executes the following UPDATE statement:

```
UPDATE     SOD
SET        DEST = "Rigel"
WHERE      SHIP = "Enterprise"
```

the result is

SHIP		OBJ		DEST		TC
Enterprise	U	Mining	M_1	Rigel	S	S
Enterprise	U	Mining	M_1	Talos	U	M_1
Enterprise	U	Exploration	U	Sirius	M_2	M_2
Enterprise	U	Exploration	U	Talos	U	U

Only the S-tuple is changed. Now the S-tuple has an OBJ value taken from a lower level and a DEST value from its own level.

After that, if an M_1-subject issues

```
UPDATE    SOD
SET       OBJ = "Spying"
WHERE     SHIP = "Enterprise"
```

the relation instance will be

SHIP		OBJ		DEST		TC
Enterprise	U	Spying	M_1	Rigel	S	S
Enterprise	U	Spying	M_1	Talos	U	M_1
Enterprise	U	Exploration	U	Sirius	M_2	M_2
Enterprise	U	Exploration	U	Talos	U	U

The M_1-tuple is changed, and the change is propagated to the S-tuple because the OBJ value accepted by S-subjects is currently owned by M_1-subjects.

Furthermore, a DELETE statement from an M_1-subject

```
DELETE
FROM      SOD
WHERE     SHIP = "Enterprise"
```

will change the relation instance to

SHIP		OBJ		DEST		TC
Enterprise	U	null	M_1	Rigel	S	S
Enterprise	U	Exploration	U	Sirius	M_2	M_2
Enterprise	U	Exploration	U	Talos	U	U

The M_1-tuple is deleted and the OBJ value in the S-tuple is set to null since no data is currently owned by M_1-subjects.

If the M_1-subject issues an UPLEVEL instead of the DELETE,

```
UPLEVEL   SOD
GET       OBJ FROM U, DEST FROM U
WHERE     SHIP = "Enterprise"
```

the result is

SHIP		OBJ		DEST		TC
Enterprise	U	null	M_1	Rigel	S	S
Enterprise	U	Exploration	U	Talos	U	M_1
Enterprise	U	Exploration	U	Sirius	M_2	M_2
Enterprise	U	Exploration	U	Talos	U	U

The M_1-tuple is changed and all its values are borrowed from U-subjects. The OBJ value of the S-tuple is null because there is no OBJ data currently owned by M_1-subjects.

An M_2-subject issuing

```
SELECT    *
FROM      SOD
```

to the instance above (* stands for all data attributes) will get the following result:

SHIP	OBJ	DEST
Enterprise	Exploration	Sirius

(i.e., the tuple at level M_2 with neither classification attribute nor tuple-class attribute). This looks like a traditional SELECT statement applied to a traditional relation, consisting of all data in the M2-tuple. If the statement issued is

```
SELECT    *%
FROM      SOD
```

(*% stands for all attributes) the result is

SHIP		OBJ		DEST		TC
Enterprise	U	Exploration	U	Sirius	M_2	M_2

All data attributes, classification attributes, as well as a tuple-class attribute are included.

For entity-polyinstantiation cases, to the following instance:

SHIP		OBJ		DEST		TC
Enterprise	M_2	Spying	M_2	Sirius	M_2	M_2
Enterprise	U	Exploration	U	Talos	U	U

a statement

```
UPLEVEL    SOD
GET        OBJ FROM M2
WHERE      SHIP = "Enterprise"
```

issued by an S-subject is rejected, or PI is violated, since there are two entities with SHIP as Enterprise satisfying the WHERE condition. In this case the S-subject should add either SHIP%=U or SHIP%=M2 (SHIP% stands for the classification attribute of SHIP) into the WHERE clause.

Finally, consider the two examples from Sandhu and Jajodia [1993] mentioned in Section 3.5. If a TS-subject issues the following SELECT statement

```
SELECT     CS.CAPTAIN, CS.CAPTAIN%, SOD.DEST, SOD.DEST%, SOD.TC
FROM       CS, SOD
WHERE      CS.SHIP=SOD.SHIP
AT         S
```

to both of them, where CAPTAIN% and DEST% stand for the classification attributes of CAPTAIN and DEST respectively, the results returned to the TS-subject are the same.

CAPTAIN		DEST		TC
Kirk	U	Rigel	S	S

This is because in both cases the S-tuple of CS only joins with the S-tuple of SOD. Note that the returned results do not have to satisfy the DBI property, since they are just a portion of those two relation instances.

We now give the formal syntax and operational semantics of the INSERT, DELETE, SELECT, UPDATE, and UPLEVEL statements.

5.2 The INSERT Statement

5.2.1 *Syntax.* The INSERT statement executed by a *c*-subject has the following general form:

```
INSERT
INTO       R[(A_j1[, A_j2]. . . )]
VALUES     (a_j1[, a_j2]. . . )
```

Symbol explanation: R is a relation name; A_{j_1}, A_{j_2}, . . . are data attribute names, $1 \leq j_1, j_2, \ldots \leq n$, assuming relation R has data attributes A_1, . . . , A_n; a_{j_1}, a_{j_2}, . . . are data values for A_{j_1}, A_{j_2}, . . . , respectively. (In this

paper we use [ ] as a syntax description, to stand for option and . . . for repetition.)

The values specified must be from appropriate domains, i.e., $a_{j_1} \in D_{j_1}$, $a_{j_2} \in D_{j_2}$, etc. Also, $c \in \{L_{j_1} \ldots H_{j_1}\}$, $c \in \{L_{j_2} \ldots H_{j_2}\}$, and so on.

5.2.2 *Semantics.* Each INSERT data manipulation can insert at most one tuple into the relation R. The inserted tuple t is constructed as follows: for $1 \leq i \leq n$,

(1) if A_i is in the attribute list of INTO clause, $t[A_i, C_i] = (a_i, c)$;

(2) if A_i is not in the attribute list of the INTO clause,
  (a) if $c \in \{L_i \ldots H_i\}$, $t[A_i, C_i] = (\text{null}, c)$; and
  (b) if $c \notin \{L_i \ldots H_i\}$, $t[A_i, C_i] = (\text{null}, \text{null})$;

Also, $t[TC] = c$.

The insertion is permitted if and only if

(1) there is no $t' \in r$ such that $t'[A_1] = a_1 \wedge t'[TC] = c$; and

(2) the resulting database state satisfies EI, FKI, and RI(1).

If so, the tuple t is inserted into r. Otherwise the data manipulation is rejected and the original database state is left unchanged.

5.2.3 *Commentary.* The INSERT semantics is quite straightforward, except the (null, null) case, which is discussed in Section 2.1.

5.3 The DELETE Statement

5.3.1 *Syntax.* The DELETE statement executed by a c-subject has the following general form:

```
DELETE
FROM      R
[WHERE    p]
```

Symbol explanation: R is a relation name, assuming relation R has data attributes $A_1, \ldots, A_n$; p is a predicate expression that may include conditions involving classification attributes, in addition to the usual case of data attributes.

5.3.2 *Semantics.* Only tuples $t \in r$ with $t[TC] = c$ are considered in the evaluation of p. That is, p is effectively changed to $p \wedge t[TC] = c$. For those tuples $t \in r$ that are selected, r is changed as follows:

(1) t is deleted;

(2) if $t[C_1] = c$, all $t' \in r$ with $t'[A_1, C_1] = t[A_1, C_1] \wedge t'[TC] > c$ are deleted from r;

(3) if $t[C_1] < c$, for $t' \in r$ with $t'[A_1, C_1] = t[A_1, C_1] \wedge t'[TC] > c \wedge t'[C_i] = c$ ($2 \leq i \leq n$), $t'[A_i]$ is set to null.

The DELETE statement is successful if at level c the resulting database state satisfies RI(1), i.e., the subview of the database state at level c, which consists of tuples with tuple classes equal to c, satisfies RI(1). Otherwise the deletion is rejected and the original database state is left unchanged.

In case where RI(1) is not satisfied at some levels c' ($c' > c$), for the relation R_1 with relation instance r_1 containing the referencing tuple t_1 and with apparent primary key AK_1 and the foreign key FK_1:

(1) if $FK_1 \cap AK_1 = \emptyset$, for $t_1[C_{FK_1}] = c'$, t_1 is set as $t_1[FK_1] =$ null, and for $t_1' \in r_1$ with $t_1'[AK_1, C_{AK_1}] = t_1[AK_1, C_{AK_1}] \wedge t_1'[TC] > c' \wedge t'[C_{FK_1}] = c'$, $t_1'[FK_1]$ is set to null;

(2) if $FK_1 \cap AK_1 \neq \emptyset$, t_1 (at level c') should also be deleted, which appears as recursive deletion.

5.3.3 *Commentary.* Note that deleting lower-level tuples may lead to deletion of tuples or setting data attributes to null at higher levels. This propagation is consistent with other issues of data-based semantics because what a borrower borrows is the value currently owned by the owner. Therefore, in case some changes happen to the owner, corresponding changes should happen to the borrower. Some variations are possible, such as instead of deleting higher-level tuples or setting higher-level data attributes to null, the system could just "freeze" and mark them, show subjects at these levels some exceptions or warnings, and let the subjects fix them. These variations are important in practice. and should be added to the semantics explicitly in an actual implementation. Where higher-level subjects do not want the values to be changed by lower-level subjects, they should set data classified at their levels by using INSERT or UPDATE (Section 5). User-defined integrity, e.g.. functional dependency, can be handled in a very similar way to data reference.

The reason for adopting different policies in RI(1) for level c and levels c' ($c' > c$) is that the DELETE statement issued by a c-subject should not be rejected due to violation of RI(1) at levels c', otherwise there would be downward information leakage. In case of $FK_1 \cap AK_1 \neq \emptyset$, the referencing tuple cannot be set where FK_1 is null, since by EI, null is not allowed within an apparent primary key. These issues arise with the UPDATE statement also.

5.4 The SELECT Statement

5.4.1 *Syntax.* The SELECT statement executed by a c-subject has the following general form:

```
SELECT    B1[, B2] . . .
FROM      R1[, R2] . . .
[WHERE    p]
[AT       c1[, c2] . . . ]
```

Symbol explanation: R_1, R_2, . . . are relation names; B_1, B_2, . . . are attribute names in R_1, R_2, . . . , each B_i is a data attribute or classification attribute or tuple-class attribute. (Wildcards are available: * for all data attributes, % for all classification attributes and tuple-class attributes, *% for all attributes.) p is a predicate expression that may include conditions involving the classification attributes, in addition to the usual data attributes; c_1, c_2, . . . are values of classification levels (there is wildcard * for all levels lower than or equal to c).

Values specified must be from appropriate domains. Also, $c_1 \leq c$, $c_2 \leq c$, and so on.

5.4.2 *Semantics.* Only those tuples $t \in r_1, r_2, \ldots$ that have $t[TC]$ as c,—and if there is no AT clause and they are not otherwise included in an AT clause—will be taken into the calculation of p. If there is more than one relation included in FROM clause, the predicate p is implicitly substituted by $p \wedge (R_1.TC = R_2.TC = \ldots)$. For tuples t satisfying p, the data of t for attributes listed in the SELECT clause are included in the result. A SELECT statement is assumed to always succeed, although the returned tuple set may be an empty set.

5.4.3 *Commentary.* Replacing p with $p \wedge (R_1.TC = R_2.TC = \ldots)$ serves to enforce that c-tuples in one relation only join with c-tuples in other relations. This is based on the idea that a c-tuple contains all the data accepted by c-subjects, and therefore should be only joined with other c-tuples. Otherwise, it is difficult to interpret the returned result.

It is possible to combine the AT clause with the WHERE clause. However, it is more natural to separate them, in the sense that most users, who have no need to see the lower-level data not accepted by them, can omit the AT clause. Also, with wildcard * different from *%, these users do not even have to see access classes.

5.5 The UPDATE Statement

5.5.1 *Syntax.* The UPDATE statement executed by an c-subject has the following general form:

```
UPDATE    R
SET       Aj1 = sj1[, Aj2 = sj2] . . .
[WHERE    p]
```

Symbol explanation: R is a relation name; $A_{j_1}, A_{j_2}, \ldots$ are data attribute names, $1 \leq j_1, j_2, \ldots \leq n$, assuming relation R has data attributes $A_1, \ldots, A_n$; $s_{j_1}, s_{j_2}, \ldots$ are scalar expressions for, respectively, $A_{j_1}, A_{j_2}, \ldots$; p is a predicate expression that may include conditions involving classification attributes, in addition to the usual data attributes.

Values specified must be from appropriate domains, i.e., $|s_{j_1}| \in D_{j_1}$, $|s_{j_2}| \in D_{j_2}$, etc., (|| stands for the calculated result). Also, $c \in \{L_{j_1} \ldots H_{j_1}\}$, $c \in \{L_{j_2} \ldots H_{j_2}\}$, and so on.

5.5.2 *Semantics.* Only tuples $t \in r$ with $t[TC] = c$ are taken into the calculation of p. For tuples $t \in r$ that satisfy the predicate p, r is updated as follows:

(1) if no attribute of A_1 is in SET clause, for $2 \leq i \leq n$, if A_i is in SET clause:
  (a) $t[A_i, C_i] = (|s_i|, c)$;
  (b) for tuples $t' \in r$ with $t'[A_1, C_1] = t[A_1, C_1] \wedge t'[TC] > c \wedge t'[C_i] = c$, $t'[A_i] = |s_i|$.

(2) if some attribute of A_1 is in the SET clause,
  (a) if $t[C_1] = c$, all tuples $t' \in r$ that have $t'[A_1, C_1] = t[A_1, C_1] \wedge t'[TC] > c$ are deleted;
  (b) if $t[C_1] < c$,
    (i) for $2 \leq i \leq n$, if A_i is not in SET clause and $t[C_i] < c$, $t[A_i, C_i] = (\text{null}, c)$;
    (ii) for tuples $t' \in r$ with $t'[A_1, C_1] = t[A_1, C_1] \wedge t'[TC] > c \wedge t'[C_i] = c$, $t'[A_i] =$ null;
  (c) for $1 \leq i \leq n$, if A_i is in the SET clause, $t[A_i, C_i] = (|s_i|, c)$.

The UPDATE data manipulation is successful if and only if:

(1) the resulting database state satisfies EI, FKI, and RI(1) (at level c); and

(2) where some attribute of A_1 is in the SET clause,
  (a) there is no $t' \in r$ such that for the resulting $t[A_1]$, $t'[A_1] = t[A_1] \wedge t'[TC] = c$;
  (b) t is not referenced by any other tuple.

Otherwise the update statement is rejected and the original database state is left unchanged.

Where RI(1) is not satisfied at levels c' ($c' > c$), the same techniques as for the DELETE statement (Section 5) are used. If RI(2) is violated, for the referencing tuples $t_1, t_2 \in r$, where $t_1[TC] = c \wedge t_2[TC] > c$, $t_2[FK, C_{FK}]$ is set as (null, c).

5.5.3 *Commentary.* When the A_1 of a base tuple is to be changed, all higher level tuples of the entity will be deleted rather than acquiring a new

A_1 value. This is because changing A_1 means changing the entity, lower-level subjects should not have the privilege of having higher-level subjects to accept any new entity beyond their willingness to do so by UPLEVEL statements.

Note that the data set by UPDATE is classified at the subject's level and will not be changed by any subject below it, unless the entire entity is deleted.

The UPDATE semantics given here is a natural extension of the traditional one, in the sense that no extra tuple is generated. In the MLR model, there are two ways to add tuples, INSERT and UPLEVEL.

5.6 The UPLEVEL Statement

5.6.1 *Syntax.* The UPLEVEL statement executed by a c-subject has the following general form:

```
UPLEVEL     R
GET         A_j1 FROM c_j1[, A_j2] FROM c_j2 . . .
[WHERE      p]
```

Symbol explanation: R is a relation name; $A_{j_1}, A_{j_2}, \ldots$ are data attribute names, $2 \leq j_1, j_2, \ldots \leq n$, assuming relation R has data attributes $A_1, \ldots, A_n$ and A_1 is an apparent key; $c_{j_1}, c_{j_2}, \ldots$ are values of classification levels for $A_{j_1}, A_{j_2}, \ldots$, respectively; p is a predicate expression that may include conditions involving the classification attributes and tuple-class attributes, in addition to the usual data attributes.

Values specified must be from appropriate domains, i.e., $c_{j_1} \in \{L_{j_1} \ldots H_{j_1}\}$, $c_{j_2} \in \{L_{j_2} \ldots H_{j_2}\}$, and so on. Also, $c_{j_1} \leq c$, $c_{j_2} \leq c$, etc.

5.6.2 *Semantics.* Only tuples $t \in r$ with $t[TC] \leq c$ are taken into the calculation of p. For every entity that has at least one tuple $t' \in r$ satisfying the predicate p, a c-tuple t is constructed as follows:

(1) $t[A_1, C_1] = t'[A_1, C_1]$;

(2) for $2 \leq i \leq n$,
  (a) if A_i is in GET clause,
    (i) if there is a tuple t'' with $t''[A_1, C_1] = t[A_1, C_1] \wedge t''[TC] = t''[C_i] = c_i$, set $t[A_i, C_i] = t''[A_i, C_i]$;
    (ii) if there is no tuple t'' with $t''[A_1, C_1] = t[A_1, C_1] \wedge t''[TC] = t''[C_i] = c_i$, set $t[A_i, C_i] = (\text{null}, c_i)$;
  (b) if A_i is not in the GET clause,
    (i) if $c \in \{L_i \ldots H_i\}$, $t[A_i, C_i] = (\text{null}, c)$;
    (ii) if $c \notin \{L_i \ldots H_i\}$, $t[A_i, C_i] = (\text{null}, \text{null})$.

After that,

(1) if there is a tuple t'' with $t''[A_1, C_1] = t[A_1, C_1] \wedge t''[TC] = c$,

(a) replace t'' with t;
(b) for any tuple t''' and any $2 \leq i \leq n$ such that $t'''[A_1, C_1] = t[A_1, C_1] \wedge t'''[TC] > c \wedge t'''[C_i] = c$, if $t[A_i, C_i] \neq t''[A_i, C_i]$, set t''' as $t'''[A_i] = \text{null}$.

(2) if there is no tuple t'' with $t''[A_1, C_1] = t[A_1, C_1] \wedge t''[TC] = c$, add t into r.

If RI(2) is violated, $t[FK, C_{FK}]$ is set as (null, c).

The UPLEVEL data manipulation is successful if and only if the resulting database state satisfies PI, FKI, and RI(1). Otherwise the data manipulation is rejected and the original database state is left unchanged.

5.6.3 *Commentary.* In the MLR data model, the only way to establish or reestablish connections between lower level data and higher level data is by using UPLEVEL statements. Having established these connections, certain kinds of "write up" can be done by subsequent UPDATE and DELETE statements at lower levels. The connection established by UPLEVEL is exactly the data-borrow relationship, which is the fundamental aspect of data-based semantics. Note that UPLEVEL also allows a c-subject to get data from the original c-tuple, as well as to borrow from lower level tuples, and replace the original c-tuple by the constructed one (with changes propagated to higher level tuples).

If a subject, say at level S, wants to borrow data from M_1, but for that attribute M_1 borrows data from U, MLR will set (null, M_1) in the S-tuple instead of getting data from U for S. From the security point of view, when S trusts M_1 and M_1 trusts U, it does not necessarily mean that S should trust U. If S trusts U, S should explicitly express this by issuing a statement to borrow data from U. Using automatic "recursive get" seems more dangerous than not. Practically, it might be useful for UPLEVEL to have a SET clause just like the one in UPDATE, because it is quite possible that a subject needs to add a tuple with some data taken from lower levels and other data provided by itself. However, this is redundant in theory, because the subject can use an UPLEVEL statement followed by an UPDATE statement to carry out this function.

6. SOUNDNESS

In this section we will prove that the MLR model is a sound data model. To clarify the essential meaning of soundness, we give the following two definitions.

Definition 6.1 In the MLR data model, a *legal* database state is one in which all relation instances satisfy the five integrity properties.

Definition 6.2 A *sound* data model is one in which any sequence of provided operational statements will transform any legal database state to a legal database state.

From these two definitions, we can see that integrity properties and operational semantics play important roles in the soundness proof. In fact, the traditional relational data model also has some integrity properties regarding primary key, foreign key, and reference control. However, in comparison to MLR, they are much simpler. The operational semantics of the traditional relational data model is also very straightforward; its soundness proof is therefore a trivial matter. For the MLR data model, due to its element-level labeling and data-based semantics, the soundness proof is much more complicated.

THEOREM 6.1 *The MLR model is sound.*

PROOF. To prove the soundness of the MLR model, we need to prove that each of the INSERT, DELETE, UPDATE, and UPLEVEL statements transform any legal database state to a legal database state. Note that any SELECT statement will leave any database state unchanged. So there are four cases to prove:

Case (1). For an INSERT statement, since EI, FKI, and RI(1) are enforced by the INSERT semantics, we only need to show that the resulting instance of relation R satisfies PI, DBI, and RI(2).

(1) PI is satisfied because
   (a) there is no t'' in the original r with $t''[A_1, C_1] = t[A_1, C_1] \wedge t''[TC] = c$, since insertion t is permitted only if there is no $t' \in r$ such that $t'[A_1] = t[A_1] \wedge t'[TC] = c$;
   (b) there is no t'' in the original r with $t''[A_1, C_1] = t[A_1, C_1] \wedge t''[TC] > c$, since the original r satisfies DBI;
   (c) there is no t'' in the original r with $t''[A_1, C_1] = t[A_1, C_1] \wedge t''[TC] < c$, since the definition of relation instance requires $tc \geq c_1$.

(2) DBI is satisfied because there is no $t[A_i]$ ($1 \leq i \leq n$) in t with $t[C_i] < t[TC]$.

(3) RI(2) is also satisfied by the same reason as (1).

Case (2). For a DELETE statement, since RI(1) is enforced by the DELETE semantics, we only need to show that the resulting database state satisfies EI, PI, DBI, FKI, and RI(2):

(1) for relation R
   (a) EI is satisfied because no tuple t'' in the original r, which satisfies EI, will change $t''[A_1, C_1]$;
   (b) PI is satisfied because
      (i) no new tuple is added;
      (ii) all tuples t' in the original r with $t'[A_1, C_1] = t[A_1, C_1] \wedge t'[TC] > c \wedge t'[C_i] = c$ ($2 \leq i \leq n$) will either be deleted or set as $t'[A_i] =$ null;

(iii) by the definition of relation instance, there is no tuple t'' with $t''[A_1, C_1] = t[A_1, C_1] \wedge t''[TC] < c \wedge t''[C_i] = c$ ($2 \leq i \leq n$).

(b) DBI is satisfied because all tuples t' in the original r with $t'[A_1, C_1] = t[A_1, C_1] \wedge t'[TC] > c \wedge t'[C_i] = c$ ($2 \leq i \leq n$) are either deleted or set as $t'[A_i] =$ null;

(d) FKI is satisfied because

(i) all tuples in the original r satisfy FKI;

(ii) all tuples t' in the original r with $t'[A_1, C_1] = t[A_1, C_1] \wedge t'[TC] > c \wedge t'[C_{FK}] = c$ are either deleted or set as $t'[A_{FK}, C_{FK}] = (\text{null}, c)$.

(e) RI(2) is also satisfied for the same reason as (1);

(2) for relation R_1 etc., either $FK_1 \cap AK_1 = \emptyset$ or $FK_1 \cap AK_1 \neq \emptyset$; the proof is similar to that for R.

Case (3). For an UPDATE statement, since EI, FKI, and RI are enforced by the UPDATE semantics, we only need to show that the resulting database state satisfies PI and DBI.

(1) for relation R:

(a) PI is satisfied because

(i) no new tuple is added;

(ii) if $t[A_1]$ is updated,

(A) all tuples t' in the original r with $t'[A_1, C_1] = t[A_1, C_1] \wedge t'[TC] > c \wedge t'[C_i] = c$ ($2 \leq i \leq n$) are either deleted or set as $t'[A_i] =$ null;

(B) for the resulting entity $t[A_1, C_1]$, the proof is similar to that for an INSERT statement;

(iii) if $t[A_1]$ is not updated, for every updated $t[A_i]$ ($2 \leq i \leq n$), all t' in the original r with $t'[A_1, C_1] = t[A_1, C_1] \wedge t'[TC] > c \wedge t'[C_i] = c$ is set to $t'[A_i] = t[A_i]$;

(iv) by the definition of relation instance, there is no t'' in the original r with $t''[A_1, C_1] = t[A_1, C_1] \wedge t''[TC] < c \wedge t''[C_i] = c$ ($2 \leq i \leq n$).

(b) DBI is also satisfied because

(i) every updated $t[A_i]$ ($1 \leq i \leq n$) has $t[C_i] = c$;

(ii) if $t[A_1]$ is updated:

(A) all tuples t' in the original r with $t'[A_1, C_1] = t[A_1, C_1] \wedge t'[TC] > c \wedge t'[C_i] = c$ ($2 \leq i \leq n$) are either deleted or set to $t'[A_i] =$ null;

(B) for the resulting entity $t[A_1, C_1]$, the proof is similar to that for an INSERT statement;

(iii) if $t[A_1]$ is not updated, for every updated $t[A_i]$ ($2 \leq i \leq n$), all

t' in the original r with $t'[A_1, C_1] = t[A_1, C_1] \wedge t'[TC] > c \wedge t'[C_i] = c$ are set as $t'[A_i] = t[A_i]$.

(2) for relation R_1 etc., the proof is similar to that of a DELETE statement.

Case (4). For an UPLEVEL statement, since PI, FKI, and RI are enforced by the UPLEVEL semantics, we only need to show that the resulting instance of relation R satisfies EI and DBI.

(1) EI is satisfied because for each added tuple t there is t' in the original r, which satisfies EI, such that $t[A_1, C_1] = t'[A_1, C_1]$.

(2) DBI is also satisfied because
   (a) in the constructed c-tuple t,
      (i) all $t[A_i]$ ($2 \leq i \leq n$) with $t[C_i] = c$ are either null or taken from the original c-tuple that has $t[C_i] = c$;
      (ii) all $t[A_i]$ ($2 \leq i \leq n$) with $t[C_i] = c' < c$ are directly taken from tuples at levels c';
   (b) where the original c-tuple t'' is replaced by t, for any $2 \leq i \leq n$ such that $t''[A_i, C_i] \neq t[A_i, C_i]$, every tuple t''' at levels c' ($c' > c$) with $t'''[C_i] = c$ will have $t'''[A_i]$ set to null.

Hence, all five operation statements will transform any legal database state to a legal database state. And any sequence of the operation statements provided here will transform any legal database state to a legal database state. □

7. COMPLETENESS

The completeness of the MLR data model is proved in this section. Again, we give the definition first.

Definition 7.1 A *complete* data model is one in which any legal database state can be transformed by a sequence of the provided data manipulation statements to any other legal database state.

The completeness of the traditional relational data model is very obvious, since we can delete or insert any specific tuple without much limitation. However, it becomes an issue in element-level labeling multilevel data models. Recall the last example in Section 1, which illustrates the incompleteness problem. This is why the MLR data model introduces a new data manipulation statement UPLEVEL to complement the other four SQL statements.

THEOREM 7.1 *The MLR model is complete.*

To prove the completeness of the MLR model, we need to prove the following lemmas.

LEMMA 7.1 *Any legal database state can be transformed by a sequence of the provided data manipulation statements to an empty database state.*

PROOF. From the DELETE semantics, we can make the following points:

(1) Any entity can be entirely deleted totally by deleting the base tuple of the entity.

(2) In deleting an entity, if the values of A_1 and C_1, which identify the entity, are given in the WHERE clause, the operation will not change any other entity, except those with tuples referencing this A_1, C_1.

(3) If we delete entities with referencing tuples before we delete entities with referenced tuples, RI will always be satisfied and the DELETE operation will not be rejected.

So by deleting all entities in all relation instances, we can get the empty database state. □

LEMMA 7.2 *An empty database state can be transformed by a sequence of provided data manipulation statements to any legal database state.*

PROOF. Any legal database state can be constructed as follows:

(1) Entities with referencing tuples are added before adding entities with referenced tuples.

(2) A multilevel entity can be added as follows:
   (a) all tuples of the entity are added in reverse topologically sorted order of their TC values;
   (b) each tuple is added by a subject at the level equal to the TC value of the tuple, as follows.
      (i) The base tuple t_1 is added by an INSERT statement with all A_i that have $t_1[A_i] \neq$ null listed in the INTO clause, and $t_1[A_i]$ in the VALUES clause.
      (ii) Adding any additional tuple t_m is done by an UPLEVEL statement, possibly followed by an UPDATE statement. All A_i and $t_m[C_i]$ with $t_m[C_i] < t_m[TC]$ are included in the USE clause of the UPLEVEL statement; whereas all A_i and $t_m[A_i]$ that have $t_m[C_i] = t_m[TC] \wedge t_m[A_i] \neq$ null are included in the SET clause of the UPDATE statement. Also, A_1, C_1, and their values are put into the WHERE clauses of both UPLEVEL and UPDATE.

This process is successful because

(1) adding one entity will not change any other entity, since A_1, C_1 and their values are included in the WHERE clauses of both UPLEVEL and UPDATE;

(2) EI, FKI, and PI are always satisfied, since the constructed database state is a legal one;

(3) DBI is always satisfied, since

(a) the constructed database state is a legal one;
(b) all tuples of an entity are added in reverse topologically sorted order of their TC values;

(4) RI is always satisfied, since
(a) the constructed database state is a legal one;
(b) entities with referencing tuples are added before adding entities with referenced tuples.

From the INSERT, UPLEVEL, and UPDATE semantics, we can easily see that the added tuples are exactly t_1 and t_ms. □

We now have the proof of Theorem 2, as follows.

PROOF. [Theorem 7.1] From Lemmas 7.1 and 7.2, any legal database state can be transformed by a sequence of provided operation statements to any other legal database state. □

8. SECURITY

Security is the fundamental issue of the MLR data model, and is the essential difference between the traditional data model and multilevel secure data models. Therefore, the security proof of the MLR data model is concerned with whether or not the MLR data model satisfies the security requirements of no "downward" information flow. Our proof is based on the concept of noninterference [Goguen 1982].

In this section we use the following notation:

—S: all subjects with varying clearance levels;

—T : all tuples with varying tuple classes in a database state.

Given any access level c,

—$SV(c)$: the set of subjects with clearance levels lower than or equal to c;

—$SH(c) : S - SV(c)$;

—$TV(c)$: the set of tuples with tuple classes lower than or equal to c;

—$TH(c) : T - TV(c)$.

It is clear that for any access level c, $S = SV(c) \cup SH(c)$ and $SV(c) \cap SH(c) = \emptyset$; while $T = TV(c) \cup TH(c)$ and $TV(c) \cap TH(c) = \emptyset$.

The security requirement can now be defined as follows.

Definition 8.1 A *secure* data model is one that is noninterfering, i.e., for any access level c, deleting any input from subject $s_1 \in SH(c)$ does not affect the output to any subject $s_2 \in SV(c)$.

There are two assumptions here regarding the term "output." One is that we do not take care of the response timing problem. Our security proof does not deal with timing covert channels (typical in noninterfering proofs).

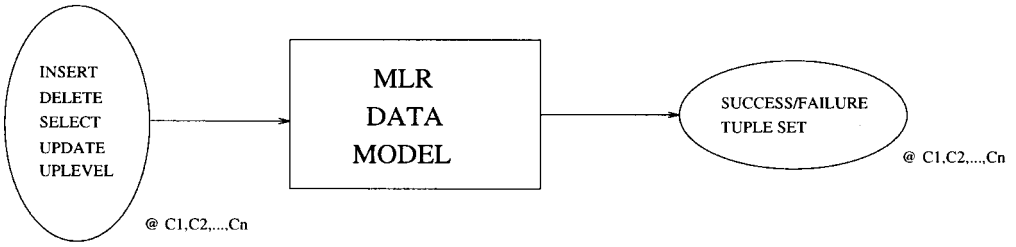

Fig. 7. The interface of the MLR data model.

These channels arise due to implementation-specific details, and are not inherent in the data model at this level of abstraction. The second assumption is that all tuples returned for a SELECT are given in a deterministic order rather than in different orders on different occasions. These two assumptions allow us to concentrate on signaling channels, which are inherent in the deterministic data model.

As shown in Figure 7, the input to the MLR data model is a sequence of operations from subjects with varying clearance levels, which are expressed by the INSERT, DELETE, SELECT, UPDATE, and UPLEVEL statements; the outputs are the results returned to the subjects, including

(1) a set of returned tuples for any SELECT statement;

(2) a SUCCESS or FAILURE (S/F) information for any INSERT, DELETE, UPDATE, or UPLEVEL statement.

THEOREM 8.1 *The MLR model is secure.*

We prove the following lemmas first.

LEMMA 8.1 *For any access level c, changing* $TH(c)$ *does not affect the output to any subject* $s \in SV(c)$.

PROOF. By the SELECT semantics, in processing a SELECT statement issued by a c'-subject $s \in SV(c)$ ($c' \leq c$), no tuples in $TH(c')$ are either taken into the calculation of p or put into the returned tuple set. Since $c' \leq c$ implies $TH(c') \supseteq TH(c)$, changing $TH(c)$ does not affect the tuple set output to $s \in SV(c)$.

By the INSERT, DELETE, UPDATE, and UPLEVEL semantics, for any c'-subject $s \in SV(c)$ ($c' \leq c$):

(1) any INSERT statement issued by s could be rejected if and only if
   (a) there is a tuple $t' \in r$ with $t'[A_1] = a_1 \wedge t'[TC] = c'$; or
   (b) the inserted tuple t violates EI or FKI; or
   (c) t references some c'-tuple t', which does not exist.

(2) any DELETE statement issued by s could be rejected if and only if the deleted tuple t is referenced by some c'-tuple t'.

(3) any UPDATE statement issued by s could be rejected if and only if
   (a) where some attribute of A_1 is in the SET clause, there is a tuple $t' \in r$ with $t'[A_1] = t[A_1] \wedge t'[TC] = c'$, where t is the updated tuple; or
   (b) t violates EI or FKI; or
   (c) where some attribute of A_1 is in SET clause, the original t is referenced by some c'-tuple t'; or
   (d) t references some c'-tuple t', which does not exist.

(4) any UPLEVEL statement issued by s could be rejected if and only if
   (a) there is a tuple $t' \in r$ with $t'[A_1] = t[A_1] \wedge t'[C_1] \neq t[C_1] \wedge t'[TC] = c'$, where t is the constructed tuple; or
   (b) t violates FKI; or
   (c) t references some c'-tuple t', which does not exist.

In all cases, only t or c'-tuple t' needs to be considered. Since $t, t' \notin TH(c') \supseteq TH(c)$, changing $TH(c)$ does not affect the S/F information output to $s \in SV(c)$; so changing $TH(c)$ does not affect the output to $s \in SV(c)$. □

LEMMA 8.2 *For any access level c, deleting any input from subject $s \in SH(c)$ does not change $TV(c)$.*

PROOF. Note that a subject can change database states only by an INSERT, DELETE, UPDATE, or UPLEVEL statement:

(1) an INSERT statement issued by a c'-subject $s \in SH(c)$ ($c' > c$) can only generate a c'-tuple t'. Since $c' > c$, $t' \notin TV(c)$;

(2) a DELETE statement issued by a c'-subject $s \in SH(c)$ ($c' > c$) can only
   (a) delete c'-tuples t', and may propagate changes to tuples t'' at levels c'' ($c'' > c'$);
   (b) may either update or delete referencing tuples t_1'' at levels c'' ($c'' > c'$), and possibly propagate changes to tuples t_1'' at levels c''' ($c''' > c''$).

   Since $c''' > c'' > c' > c$, all these t', t'', t_1'', $t_1''' \notin TV(c)$;

(3) an UPDATE statement issued by a c'-subject $s \in SH(c)$ ($c' > c$) can only
   (a) change c'-tuples t', and may propagate the changes to tuples t'' at levels c'' ($c'' > c'$);
   (b) may either update or delete referencing tuples t_1'' at levels c'' ($c'' > c'$), and possibly propagate changes to tuples t_1'' at levels c''' ($c''' > c''$).

   Since $c''' > c'' > c' > c$, all these t', t'', t_1'', $t_1''' \notin TV(c)$;

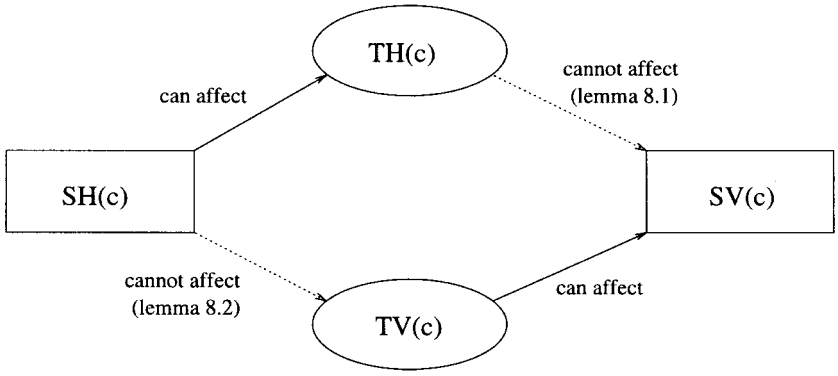

Fig. 8. What can affect and cannot affect relations.

(4) an UPLEVEL statement issued by a c'-subject $s \in SH(c)$ ($c' > c$) can either add a new c'-tuple t' or change an original c'-tuple t'', and propagate the changes to tuples t''' at levels c''' ($c''' > c'$). Since $c''' > c' > c$, all these t', t'', $t''' \notin TV(c)$;

From this we can see that deleting any input from subject $s \in SH(c)$ does not change $TV(c)$. □

These two lemmas are illustrated in Figure 8; we can now prove Theorem 8.1

PROOF. [Theorem 8.1]. From Lemmas 8.1 and 8.2, for any level c, since $S = SV(c) \cup SH(c)$, $SV(c) \cap SH(c) = \emptyset$, $T = TV(c) \cup TH(c)$, $TV(c) \cap TH(c) = \emptyset$, deleting any input from subject $s_1 \in SH(c)$ does not affect the output to $s_2 \in SV(c)$. □

9. CONCLUSION

In this paper we fully defined the MLR model and proved that the MLR model is sound, complete, and free of downward information flows. The redefined polyinstantiation integrity and referential integrity properties as well as the newly introduced UPLEVEL statement and data-borrow integrity strongly support the fact that the MLR model is a secure, unambiguous, and powerful data model.

In the MLR data model, any subject can not only "read down," i.e., get data accepted by subjects at its level or at the levels below it, but also do some kind of "write up," i.e., change data accepted by subjects at levels above it, provided the data are owned by subjects at its level.

At any given level an apparent primary key value identifies only one entity, which avoids referential ambiguity. However, in general, an apparent primary key value can indicate different entities at different levels, therefore entity polyinstantiation is allowed across classification levels, which prevents downward information leakage arising from rejecting low-

er-level insertion. Some other approaches may be useful to supplement our polyinstantiation integrity property, especially for some specific cases.

Since the MLR data model has a clear and unambiguous semantics, some further extensions are fairly straightforward. For example, many user-defined integrity constraints in the traditional data model, some of which even include aggregation functions, can also be included in the MLR data model. In the MLR model, all these traditional constraints are applied at each classification level on those tuples accepted by subjects at that level. Of course, some techniques must be used to prevent downward information leakage, similar to what is done in referential integrity checking.

Another interesting issue is scheme classification. In MLR, a scheme is classified at the greatest lower bound of L_i ($i = 1 \ldots n$). However, there may be some cases where one wants to hide entirely some attributes from lower-level subjects, i.e., subjects at different levels have different schemes. There are several questions that arise: Who is in charge of scheme creation and maintenance? What is the relationship between schemes at different levels? Would this appear as "scheme polyinstantiation"? What about those user-defined integrity constraints with respect to scheme classification?

REFERENCES

BELL, D. E. AND LAPADULA, L. J. 1975. Secure computer systems: Unified exposition and multics interpretation. Tech Rep. MTR-2997. MITRE Corp., Bedford, MA.

CHEN, F. AND SANDHU, R. S. 1995. The semantics and expressive power of the multilevel relational data model. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*. IEEE Computer Society Press, Los Alamitos, CA, 128–142.

DENNING, D. E., LUNT, T. F., SCHELL, R. R., SHOCKLEY, W. R., AND HECKMAN, M. 1988. The SeaView security model. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*. IEEE Computer Society Press, Los Alamitos, CA, 218–233.

GOGUEN, J. A. AND MESEGUER, J. 1982. Security policies and security models. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*. (Oakland, CA), IEEE Computer Society Press, Los Alamitos, CA, 11–20.

JAJODIA, S. AND LANDWEHR, C. E., Eds. 1991. *Database Security IV: Status and Prospects*. North-Holland Publishing Co., Amsterdam, The Netherlands.

JAJODIA, S., AND SANDHU, R. 1991. Honest databases that can keep secrets. In *Proceedings of the 14th NIST-NCSC National Computer Security Conference* (Washington, DC, Oct.), 267–282.

JAJODIA, S., SANDHU, R. S., AND SIBLEY, E. 1990. Update semantics of multilevel relations. In *Proceedings of the 6th Annual Computer Security Applications Conference*. (Tucson, AZ, Dec.), 103–112.

JAJODIA, S. AND SANDHU, R. S. 1991. Toward a multilevel secure relational data model. In *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data* (Denver, CO, May 29–31, 1991), J. Clifford and R. King, Eds. ACM Press, New York, NY, 50–59.

LUNT, T. F., DENNING, D. E., SCHELL, R. R., HECKMAN, M., AND SHOCKLEY, W. R. 1990. The SeaView security model. *IEEE Trans. Softw. Eng. 16*, 6 (June), 593–607.

QIAN, X. 1994. A model-theoretic semantics of the multilevel relational model. In *Proceedings of the 4th International Conference on Extending Database Technology: Advances in Database Technology* (Cambridge, UK, March 28–31, 1994), M. Jarke, J. Bubenko, and K. Jeffery, Eds. Lecture Notes in Computer Science, Springer-Verlag, New York, NY, 201–214.

QIAN, X. AND LUNT, T. 1993. Tuple-level vs. element-level classification. In *Database Security VI. Results of the Sixth Working Conference of IFIP Working Group 11.3* (Simon Fraser

Univ., Vancouver, B.C., Canada, Aug. 19–21, 1992), B. Thuraisingham and C. E. Landwehr, Eds. Elsevier Science Inc., New York, NY, 301–315.

SANDHU, R. S. 1993. Lattice-based access control models. *IEEE Computer 26*, 11, 9–19.

SANDHU, R. S. AND JAJODIA, S. 1992. Polyinstantiation for cover stories. In *Proceedings of the European Symposium on Research in Computer Security* (Toulouse, France, Nov.) Y. Deswarte, G. Eizenberg, and J.-J. Quisquater, Eds. Lecture Notes in Computer Science, Springer-Verlag, Berlin, 307–328.

SANDHU, R. S. AND JAJODIA, S. 1993. Referential integrity in multilevel secure databases. In *Proceedings of the 16th National Computer Security Conference*. (Baltimore, MD, Sept. 20–23), 39–52.

SANDHU, R. S., JAJODIA, S., AND LUNT, T. 1990. A new polyinstantiation integrity constraint for multilevel relations. In *Proceedings of the IEEE Workshop on Computer Security Foundations* (Franconia, NH, June), 159–165.

SMITH, K. AND WINSLETT, M. 1992. Entity modeling in the MLS relational model. In *Proceedings of the International Conference on Very Large Data Bases* (Vancouver, Canada, Aug. 1992), IEEE Computer Society Press, Los Alamitos, CA, 199–210.

THURAISINGHAM, B. M. 1991. A nonmonotonic typed multilevel logic for multilevel secure database/knowledge-base management systems. In *Proceedings of the Fourth IEEE Workshop on Computer Security Foundations*, 127–138.

Received: December 1997; revised: March 1998; accepted: March 1998