The Multilevel Relational (MLR) Data Model

Ravi Sandhu and Fang Chen
George Mason University

Many multilevel relational data models have been proposed, and different models offer different advantages. In this paper, we adapt and refine several of the best ideas from previous models and add new ideas 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 from the literature (notably 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 to 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 respectively show that any of the five database manipulation operations will keep database states legal (i.e., satisfying all integrity properties), that every legal database state can be constructed, and that MLR is non-interfering. The expressive power of MLR is also compared 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 - *Security, integrity, and protection*

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 respectively. Accesses by subjects are restricted by mandatory access controls, roughly expressed as "no read up, no write down", to follow the well-known Bell-LaPadula model [1; 11]. However, from a security standpoint, this restriction should be strengthened by a further requirement: operations from any given level should not be accepted or rejected due to existence or absence of any higher level data, otherwise there will be some signaling channels for leakage of high-level data (i.e., indirect methods of communication from higher-level processes to lower-level ones [7]).

Many multilevel relational data models have been proposed in the literature, for example SeaView [3; 8], LDV [5], and those proposed by Sandhu-Jajodia [13; 14], Jajodia-Sandhu [6; 7], Smith-Winslett [16], etc. Each of these models has its innovations and strong points, e.g., the integrity properties in SeaView and Sandhu-Jajodia model, the belief-based semantics of Smith-Winslett model, the multilevel manipulation of 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? This paper directly confronts this question by the 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.

This research was supported in part by the National Security Agency through contract MDA904-94-C-6119 at George Mason University.
A preliminary version of this paper appeared under the title "The Semantics and Expressive Power of the MLR Data Model" in *Proceedings of the IEEE Symposium on Research in Security and Privacy*, Oakland, California, May 8-10, 1995, pages 128-142.
Author's address: Ravi Sandhu, Information and Software Engineering Department, Mail Stop 4A4, George Mason University, Fairfax, VA 22033; email: sandhu@isse.gmu.edu, URL: www.list.gmu.edu.

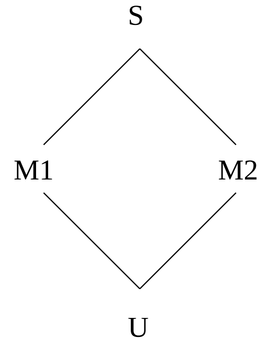

Fig. 1. A partially ordered lattice

MLR is substantially based on the data model proposed by Sandhu and Jajodia [14]. Many aspects of that model in turn are derived from SeaView [3]. The most significant difference between the Sandhu-Jajodia and the SeaView models is the requirement, introduced in [14], 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.

However, there are two major problems left unsolved in the Sandhu-Jajodia model of [14], *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. We assume the reader is familiar with basic aspects of lattice-based access control [11], and we use the conventional labels TS>S>C>U. Let us say that the data in tuples with TC at U and S are respectively are 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 existing in this relation. Do subjects at TS take the values from S or from U? Is there any necessity 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 that at 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 will 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, there is no way in previous models 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.

In this paper, we fully define the MLR data model. 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 respectively show that any of the provided database manipulation operations will keep the database state legal (i.e., satisfying 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 this 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 data manipulation operations of the model. Sections 6, 7 and 8 respectively prove 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 to 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 until 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 sub-lattice 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 $c_i \in \{L_i \ldots H_i\} \cup \{null\}$, 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 3.1 we will 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 having 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 [14]. Firstly, the domain of TC is different from that in [14], which is $\{\text{lub}\{L_i : i = 1 \ldots n\} \ldots \text{lub}\{H_i : i = 1 \ldots n\}\}$. Secondly, the model of [14] 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

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

[3] As we will see in section 3.1, there will always be some c_i which is non-null, therefore $\text{lub}\{c_i \mid c_i \neq \text{null} : i = 1 \ldots n\}$ is always well defined.

[4] Scheme classification in multilevel relations remains an open research issue.

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.

Also, in MLR every relation has only one relation instance at any time. As we will see in section 2.2, 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.

Same as [14], we define $tc \geq \text{lub}\{c_i \mid c_i \neq \text{null} : i = 1 \ldots n\}$, by which

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

is allowed and has different meanings from

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

In fact, the former one 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 one 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 by coincidence. The data interpretation is discussed at length in section 2.2.

The following definition is directly taken 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 ones can be changed by the lower-level subjects who are owning 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 level. Other access controls, such as discretionary or role-based, are omitted for simplicity. These basic ideas come from combining belief-based semantics [16] and LDV model [5]. In sections 3 and 5 we will see 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, identified by (Enterprise, S) and (Enterprise, U) respectively. (Enterprise, S) and (Enterprise, U) are respectively created by a U-subject and an S-subject, and can only be respectively deleted by U-subjects and S-subjects. 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 be deleted only 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 having 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 such as the entire entity is deleted.[5]
 —When $t[TC] = t[C_1]$, t is the *base tuple* of the entity, which means 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 is 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 for attribute A_i, tc-subjects are expecting to borrow data owned by c_i-subjects, however, no data is currently owned by them.
 —Both $t[A_i, C_i]$ =[null, null] and $t[A_i, C_i]$ =[null, tc] mean for A_i no data is available at level tc. The [null, null] case applies when $tc \notin \{L_i \ldots H_i\}$, otherwise the [null,$tc$] case applies.

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? Let us consider a traditional database example first. What does the following tuple mean?

SHIP	OBJ	DEST
Enterprise	Exploration	Talos

One answer is, it means there is a ship in the real world called Enterprise that will go to Talos for exploration. Another answer can be, it stands for an opinion of some people who think a ship Enterprise will go to Talos for exploration. These two answers are very different. The first one says the tuple

[5]It should be noted that discretionary or non-discretionary access controls can be used to control which subject can delete the tuple.
[6]This issue is to be discussed in greater detail in section 3.3 and 5.6.

represents a fact in the real world which should definitely be true, whereas the second one says 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 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 these data are obtained. It is ultimately a problem of cognition. We can also say that the second answer is more general than the first one in the sense it includes both true cognition and false cognition of the real world.

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 a similar approach as 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, respectively.[7] For brevity we denoted these three groups of people as TSs, Ss and Us respectively. Ss may think that their data are more correct than that of Us, however, for some data, e.g. DEST, TSs may nevertheless trust Us rather than Ss. Here both TSs and Ss trust Us on the existence of Enterprise, which means all TSs, Ss and Us are talking about the same Enterprise, therefore TSs can trust either Ss or Us 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 TSs trust Us on the existence of a ship Enterprise, therefore they clearly know that their Enterprise is exactly the same as what Us mean. On the other hand, Ss have their own opinion which is totally independent from that of Us, even on the existence of a ship Enterprise. There could be a case that Ss address their opinion before Us doing so and therefore have no idea about if and what Us might say later. Of course, Ss can change their opinion if subsequently they find out they can trust Us on the existence of Enterprise and their Enterprise is exactly what Us mean. Generally, if there are polyinstantiation tuples with same A_1 but different C_1, they could refer to the 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 TSs feel there exist two different Enterprises, mentioned by Us and Ss respectively, and they want to trust Us and Ss accordingly. However, in this case, if later on TSs want to say that "Kirk is the captain of Enterprise", it is not clear that Kirk is the captain of which Enterprise, or maybe of both. Theoretically, there could be a way to deal with this problem: TSs give a nick name to each of these two ships, say, Enterprise-U for the Enterprise mentioned by Us and Enterprise-S for that by Ss, using "Kirk is the captain of Enterprise-U" and/or "Kirk is the captain of Enterprise-S" accordingly. However the problem is, if we assume there are some other groups at levels even higher than TS, they have to know all the nick names defined by TSs before they can understand and/or use some opinions of TSs. The problem will become even worse if, say, Enterprise-U needs to be used in some other situations such as "The maintenance of Enterprise-U is in July" identified by both "maintenance" and "Enterprise-U", because for the same reason there could be several different "maintenance of Enterprise-U" accepted by some specific group at the same time, leading to recursive nick names. Section 3.1 will further discuss this issue.

[7]So far we have talked about subjects accepting 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 will get a different view in each case. 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 users perspective. The TS user is trusted to behave like a S user in this context and not leak TS data deliberately or inadvertently.

2.4 Practicality

As we have seen, the 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 are belonging to the subjects below c. Actually, this is the crucial requirement to eliminate semantic ambiguity.

This requirement would be quite acceptable if 95% entities had different data at different levels. Unfortunately, often only, say, 5% entities will have secret data at levels higher than unclassified; which means for the other 95% entities, high level tuples just repeat unclassified data. If there are m levels higher than unclassified, in order 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, because the repetition is explicitly done by subjects at each level.

By this consideration, we must face practical issues of the physical model under this reasonable logical model. Fortunately there are several techniques that can be used to 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 $A_1, C_1, \ldots, A_n, C_n$. 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 for all entities in R, all data owned by c'-subjects ($c' < c$) are accepted. This is to say, whenever a c'-subject insert an entity into R, a c-tuple of the entity will be automatically created and all data of the c-tuple are borrowed from c'-subjects (possibly, we can just put c into the TC set of the c'-tuple). For single level relations, i.e., for any i, j such that $1 \le i, j \le n$, $L_i = L_j = H_i = H_j$, 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 make the feasibility argument sketched out here.

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 newly introduced. In particular, the Polyinstantiation Integrity property given here is much more general than that of either SeaView [3] or Sandhu-Jajodia model [14], in that it takes care of both entity polyinstantiation 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 [3], 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) $A_i \in AK \Rightarrow t[A_i] \neq null$
(2) $A_i, A_j \in AK \Rightarrow t[C_i] = t[C_j]$
(3) $A_i \notin AK, A_j \in AK \Rightarrow t[C_i] \geq t[C_j]$.

The first requirement is exactly the definition of entity integrity from the traditional relational model, and 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 therefore 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 the non-key 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, say, at level S addresses an independent opinion, all the data are classified at its level. Later on if another subject, say, at 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, therefore 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 has been required by many models. However, our definition is much more general than previous ones. It is the first one that treats 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) $A_1, TC \rightarrow C_i$
(2) $A_1, C_1, C_i \rightarrow A_i$.

The second requirement is the original polyinstantiation integrity required by both SeaView and Sandhu-Jajodia model, which says 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) $A_1, TC \rightarrow C_1$;
(b) $A_1, C_1, TC \rightarrow A_i, C_i$.

Here, (b) is called tuple-class polyinstantiation integrity in [15][8], which says that every entity in a relation can have at most one tuple for every access class; whereas (a) is newly introduced and is discussed below.

The new property (a) can be called as 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, because at each level, either TS, S or U, subjects only accept one entity with AK value being 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, because 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 [14]. In our case, there can be no downward information leakage, because entity polyinstantiation is allowed across security levels, and therefore no insertion will be rejected due to existing entity polyinstantiation at higher level (see section 5.2 for detail). Also, there is no ambiguity, because no entity polyinstantiation is allowed in what is accepted by subjects at any particular security level.

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

MLR does not support nick name definition (Section 2.3) because we feel using nick name is very confusing. Actually, this confusion exists in the real world. Sandhu and Jajodia [13] 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 of 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 of 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 one 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, must exist. This is because absence of a U-tuple means that U-subjects do not accept the entity Enterprise at this time; 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 yet satisfies PI.

3.4 Foreign Key Integrity

The following Foreign Key Integrity was originallyproposed by SeaView [3].

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 $t \in r$*

item Either $(\forall A_i \in FK)[t[A_i] = null]$ or $(\forall A_i \in FK)[t[A_i] \neq null]$

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

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

3.5 Referential Integrity

Referential Integrity appears both in SeaView and in Sandhu-Jajodia model. The main issue with referential integrity is to avoid semantic ambiguity, as discussed at length in [14]. The definition given here consists of two parts. The first one is similar to the original SeaView definition and the Sandhu-Jajodia definition. However, referential ambiguity is eliminated by our data-based semantics. The second part 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 of 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* $t_{11}[FK_1] = t_{21}[AK_2] \wedge t_{11}[TC] = t_{21}[TC] \wedge t_{11}[C_{FK_1}] \geq t_{21}[C_{AK_2}]$;

(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 will 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 non-null 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 only allow 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, absence of a c-tuple means to c-subjects the entity does not exist.

Let us consider the two examples described in [14] 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	S	Spying	S	Rigel	S	S

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

had some ambiguity 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 of CS can only reference the S-tuple of SOD, there is no referential ambiguity. As 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 be allowed in previous models if the restriction $t_1[C_{FK}] \geq t_2[C_{AK}]$ was 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 that is 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

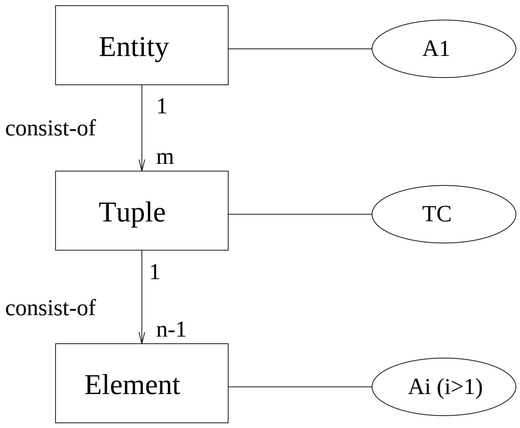

Fig. 2. The expressive power of the tuple-level labeling data model

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. We thus 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 anamolous case violates RI(2), because the two referencing tuples in CS, at level TS and S respectively, are of the same entity (Kirk, U) and the TS-tuple borrows the foreign key SHIP from the S-tuple, 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 respect to several other data models. This is done 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, because it is the simplest data model and has been theoretically analyzed in [17]. 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 context of figure 2, because here an entity is identified only 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, because this allows entities whose attributes are labeled at different classes; whereas entity polyinstantiation would require all attributes of an entity to be uniformly classified.

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, because each entity is identified by both A_1 and C_1, and both tuple and elements have their own classification levels.

What we called the semi-tuple-level labeling data model has schemes as follows.

A_1	C_1	A_2	...	A_n	TC

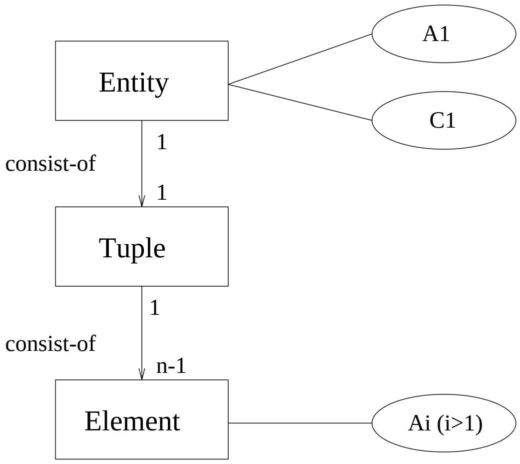

Fig. 3. An alternate interpretation of the tuple-level labeling data model

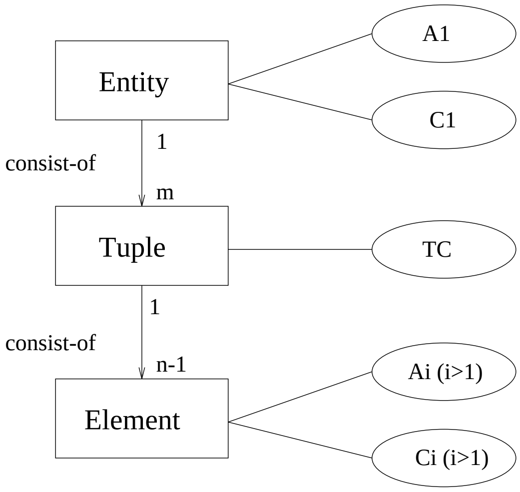

Fig. 4. The expressive power of the MLR data model

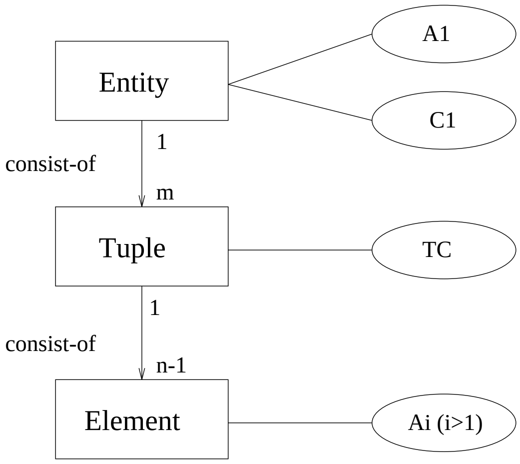

Fig. 5. The expressive power of the semi-tuple-level labeling data model

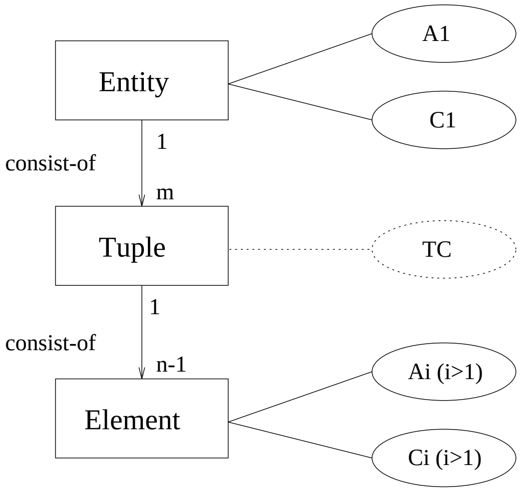

Fig. 6. The expressive power of the SeaView data model

This is exactly the same as the model provided by Smith and Winslett [16]. 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 let us compare MLR with SeaView. Although the SeaView model is not grounded on 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, it is attached by a dotted line in the diagram.

What SeaView can do to counter the semantic ambiguity problem addressed in section 1 is very limited. The model-theoretic semantics of [9] is slightly different from the SeaView's original "fact-based" semantics [10], 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

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 data Talos can be changed by U-subjects. Neither SeaView nor semi-tuple-labeling model can express this case, because it concerns data-borrow. In SeaView, TS-subjects should accept 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 of 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-Jajodia model of [13], we have redefined the semantics of the four traditional SQL statements, and have replaced PUPDATE by UPLEVEL. The SELECT statement given here is similar to that of Smith-Winslett [16], and our intent is to provide a friendly interface upward-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 [6], 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 to an instance a tuple 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 into SOD, whose OBJ and DEST values are, respectively, borrowed from the data owned by M_1-subjects and M_2-subjects.

Next, we illustrate how there is 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. This is 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 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 will be

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.

To the instance above, an M_2-subject issuing

```
SELECT   *
FROM     SOD
```

(∗ 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 M_2-tuple. If the issued statement 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 will be rejected, or PI would be 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%=M_2 (SHIP% stands for the classification attribute of SHIP) into the WHERE clause.

Finally, let us consider the two examples from [14] mentioned in section 3.5. To both of them, 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
```

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:

$$\begin{array}{ll} \text{INSERT} & \\ \text{INTO} & R[(A_{j_1}[, A_{j_2}]\ldots)] \\ \text{VALUES} & (a_{j_1}[, a_{j_2}]\ldots) \end{array}$$

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 $A_1, \ldots, A_n$; a_{j_1}, a_{j_2}, $\ldots$ are data values for A_{j_1}, A_{j_2}, $\ldots$ respectively. (In this paper, as a syntax description we use [ ] to stand for option and ... for repetition.)

Values specified must be from appropriate domains, i.e., $a_{j_1} \in D_{j_1}$, $a_{j_2} \in D_{j_2}$, and so on. 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 INTO clause,
  (a) if $c \in \{L_i \ldots H_i\}$, $t[A_i, C_i] = (\text{null}, c)$;
  (b) if $c \notin \{L_i \ldots H_i\}$, $t[A_i, C_i] = (\text{null, null})$;

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 was 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:

$$\begin{array}{ll} \text{DELETE} & \\ \text{FROM} & R \\ \text{[WHERE} & p] \end{array}$$

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

5.3.2 *Semantics.* Only tuples $t \in r$ with $t[TC] = c$ will be 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 will be changed as follows:

(1) t will be deleted;
(2) if $t[C_1] = c$, all $t' \in r$ with $t'[A_1, C_1] = t[A_1, C_1] \wedge t'[TC] > c$ will be 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 data manipulation is rejected and the original database state is left unchanged.

In case that 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 $t_1[FK_1] = \text{null}$, 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 deletions.

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 the 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 these subjects fix them. These variations are important in practice and should be explicitly added to the semantics in an actual implementation. In case that 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 5.5). User-defined integrity, e.g. functional dependency, can be handled in a very similar way as that of 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 can not be set as having FK_1 being null, because 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:

$$\begin{array}{ll} \text{SELECT} & B_1[, B_2] \ldots \\ \text{FROM} & R_1[, R_2] \ldots \\ \text{[WHERE} & p] \\ \text{[AT} & c_1[, c_2] \ldots] \end{array}$$

Symbol explanation: R_1, R_2, ... are relation names; B_1, B_2, ... are attribute names in R_1, R_2, ... either data attribute or classification attribute or tuple-class attribute (Wildcards are available, $*$ for all data attributes, % for all classification attributes and tuple-class attribute, $*$% for all attributes); p is a predicate expression which may include conditions involving the classification attributes, in addition to the usual case of data attributes; c_1, c_2, ... are values of classification levels (There is wildcard $*$ standing 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 $t[TC]$ being c if there is no AT clause, or otherwise included in AT clause, will be taken into the calculation of p. If there are 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 those tuples t satisfying p, the data of t for those attributes listed in SELECT clause will be 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 would be difficult to interpret the returned result.

It is possible to combine AT clause into 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 that are not accepted by them, can omit the AT clause. Also, with wildcard $*$ different from $*\%$, these users do not have to even see access classes.

5.5 The UPDATE Statement

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

$$\begin{array}{ll} \text{UPDATE} & R \\ \text{SET} & A_{j_1} = s_{j_1} [, A_{j_2} = s_{j_2}] \ldots \\ \text{[WHERE} & p] \end{array}$$

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 $A_1, \ldots, A_n$; s_{j_1}, s_{j_2}, ... are scalar expression for A_{j_1}, A_{j_2}, ... respectively; p is a predicate expression which may include conditions involving the classification attributes, in addition to the usual case of data attributes.

Values specified must be from appropriate domains, i.e., $|s_{j_1}| \in D_{j_1}$, $|s_{j_2}| \in D_{j_2}$, and so on ($|\ |$ stands for 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 $t[TC] = c$ will be taken into the calculation of p. For those tuples $t \in r$ that satisfy the predicate p, r will be 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 SET clause,
   (a) if $t[C_1] = c$, all tuples $t' \in r$ that have $t'[A_1, C_1] = t[A_1, C_1] \wedge t'[TC] > c$ will be deleted;
   (b) if $t[C_1] < c$,
      i. for $2 \leq i \leq n$, if A_i is not in SET clause and $t[C_i] < c$, $t[A_i, C_i] =$ (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 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) In case that some attribute of A_1 is in 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 data manipulation is rejected and the original database state is left unchanged.

In case that RI(1) is not satisfied at levels c' ($c' > c$), same techniques as that for the DELETE statement (section 5.3.2) 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 get 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:

$$\begin{array}{ll} \text{UPLEVEL} & R \\ \text{GET} & A_{j_1} \text{ FROM } c_{j_1} [, A_{j_2} \text{ FROM } c_{j_2}] \ldots \\ \text{[WHERE} & p] \end{array}$$

Symbol explanation: R is a relation name; A_{j_1}, A_{j_2}, ... 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 apparent key; c_{j_1}, c_{j_2}, ... are values of classification levels for A_{j_1}, A_{j_2}, ... respectively; p is a predicate expression which may include conditions involving the classification attributes and tuple-class attributes, in addition to the usual case of 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, $c_{j_1} \leq c$, $c_{j_2} \leq c$, and so on.

5.6.2 *Semantics.* Only tuples $t \in r$ with $t[TC] \leq c$ will be 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 will be 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 $t[A_i, C_i] =$ (null,c_i);
    (b) if A_i is not in GET clause,
        i. if $c \in \{L_i \ldots H_i\}$, $t[A_i, C_i] =$ (null, c);
        ii. if $c \notin \{L_i \ldots H_i\}$, $t[A_i, C_i] =$ (null, 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 $t'''[A_i] =$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 using UPLEVEL statements. Having established these connections, certain kind 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 the data-based semantics. Note that UPLEVEL also allows a c-subject to get data from the original c-tuple as well as borrow from lower level tuples, and replace the original c-tuple by the constructed one (with relative 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 security point of view, S trusts M_1 and M_1 trusts U do not necessarily mean that S should trust U. If S trusts U, S should explicitly express that 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 as that in UPDATE, because it is quite possible that a subject needs to add a tuple with some data taken from lower levels and others 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 are going to prove that the MLR model is a sound data model. To clarify the essential meaning of soundness, we give the following two definitions first.

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 will 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 with MLR, they are much simpler. Also, the operational semantics of the traditional relational data model is very straightforward; therefore its soundness proof is really a trivial matter. For the MLR data model, because of 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 will 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.

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 inserting 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 with 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 have $t''[A_1, C_1]$ being changed.
   (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 be either deleted or set as $t'[A_i] = \text{null}$;

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

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

(d) FKI is satisfied, because

i. all tuples in the original r satisfy FKI;

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

(e) RI(2) is also satisfied with the same reason as 1b.

(2) for relation R_1 and so on, 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 $t'[A_1, C_1] = t[A_1, C_1] \wedge t'[TC] > c \wedge t'[C_i] = c$ $(2 \leq i \leq n)$ will be either deleted or set as $t'[A_i] = \text{null}$;

B. for the resulting entity $t[A_1, C_1]$, the proof is similar to that of 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 $t'[A_1, C_1] = t[A_1, C_1] \wedge t'[TC] > c \wedge t'[C_i] = c$ will be set as $t'[A_i] = t[A_i]$;

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 $t'[A_1, C_1] = t[A_1, C_1] \wedge t'[TC] > c \wedge t'[C_i] = c$ $(2 \leq i \leq n)$ will be either deleted or set as $t'[A_i] = \text{null}$;

B. for the resulting entity $t[A_1, C_1]$, the proof is similar to that of 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 $t'[A_1, C_1] = t[A_1, C_1] \wedge t'[TC] > c \wedge t'[C_i] = c$ will be set as $t'[A_i] = t[A_i]$.

(2) for relation R_1 and so on, 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) in case that 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 tuples t''' at levels c' $(c' > c)$ with $t'''[C_i] = c$ will have $t'''[A_i]$ set to null.

Hence, all these five operation statements will transform any legal database state to a legal database state. Therefore, any sequence of these provided operation statements 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 following 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, because we can delete or insert any specific tuple without much limitations. However, it becomes an issue in element-level labeling multilevel data models. Recall the last example in section 1, which illustrates the incompleteness problem. That 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 see 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.

By deleting all entities in all relation instances, we can therefore 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 the 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 being 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 $t_m[C_i]$ that have $t_m[C_i] < t_m[TC]$ 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 will be 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 the 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 those t_1 and t_m's.

We now have the proof of Theorem 7.1 as follows.

Proof: [Theorem 7.1] From Lemma 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 concerns 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 [4].

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 which is non-interfering, 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 to the term "output". One is that we do not take care of the response timing problem. Our security proof therefore does not deal with timing covert channels (as is typical of non-interfering proofs). 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 statement; the output is 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 $TH(c')$ will be 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 $s \in SV(c)$ ($c' \leq c$),

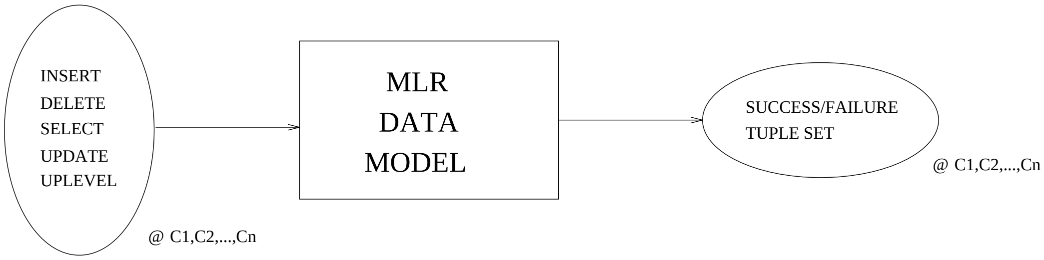

Fig. 7. The interface of the MLR data model

(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 is not existing.
(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) in case that some attribute of A_1 is in 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) in case that 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 is not existing.
(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 is not existing.

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 $s \in SV(c)$. Therefore, 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 $t', t'', t''_1, t'''_1 \notin TV(c)$.
(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 $t', t'', t''' \notin TV(c)$.

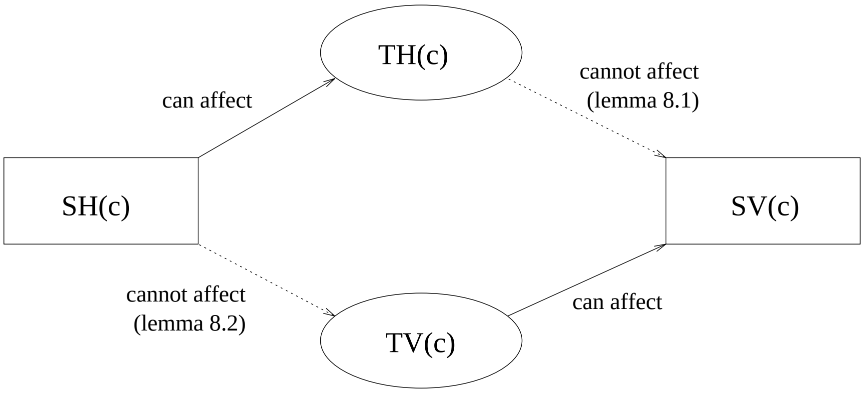

Fig. 8. Can Affect and Cannot Affect Relations

From all of these, 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. Now we can prove Theorem 8.1.

Proof: [Theorem 8.1] From Lemma 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 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 the downward information leakage arising from rejecting lower-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 into the MLR data model. In the MLR model, all these traditional constraints are applied at each classification level, upon 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 the 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 entirely hide some attributes from lower-level subjects, i.e., subjects at different levels have different scheme. 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 that appear as "scheme polyinstantiation"? How about those user-defined integrity constraints with respect to scheme classification?

REFERENCES

[1] Bell, D.E. and LaPadula, L.J. "Secure Computer Systems: Unified Exposition and Multics Interpretation." MTR-2997, MITRE (1975).

[2] Chen, F. and Sandhu, R.S. "The Semantics and Expressive Power of the Multilevel Relational Data Model." *Proc. IEEE Symposium on Security and Privacy*, 1995, pages 128-142.

[3] Denning, D.E., Lunt, T.F., Schell, R.R., Shockley, W.R. and Heckman, M. "The SeaView Security Model." *Proc. IEEE Symposium on Security and Privacy*, 1988, pages 218-233.

[4] Goguen, J.A. and Meseguer, J. "Security Policies and Security Models." *Proc. IEEE Symposium on Security and Privacy*, 1982, pages 11-20.

[5] Haigh, J.T., O'Brien, R.C. and Thomsen, D.J. "The LDV Secure Relational DBMS Model." *Database Security IV: Status and Prospects*, S. Jajodia and C. E. Landwehr (editors), North-Holland, 1991, pages 265-279.

[6] Jajodia, S. , Sandhu, R.S., and Sibley E. "Update Semantics of Multilevel Relations." *Proc. 6th Annual Computer Security Applications Conf.*, Tucson, AZ, December 1990, pages 103-112.

[7] Jajodia, S. and Sandhu, R.S. "Toward A Multilevel Secure Relational Data Model." *Proc. ACM SIGMOD*, Denver, Colorado, May 1991, pages 50-59.

[8] Lunt, T.F., Denning, D.E., Schell, R.R., Heckman, M. and Shockley, W.R. "The SeaView Security Model." *IEEE Transactions on Software Engineering*, 16(6):593-607 (1990).

[9] Qian, X. "A model-theoretic semantics of the multilevel relational model." *Advances in Database Technology – EDBT'94, Lecture Notes in Computer Science 779*, Jarke, M., Bubenko, J. and Jeffery, K. (editors), Springer-Verlag, pages 201-214, 1994.

[10] Qian, X. and Lunt, T.F. "Tuple-level vs. element-level classification." *Database Security IV: Status and Prospects*, Thuraisingham, B.M. and Landwehr, C. (editors), North-Holland, pages 301-315, 1993.

[11] Sandhu, R.S. "Lattice-based access control models." *IEEE Computer*, 26(11):9-19 (1993).

[12] Sandhu, R.S. and Jajodia, S. "Honest Databases That Can Keep Secrets." *14th NIST-NCSC National Computer Security Conference*, Washington, D.C., October 1991, pages 267-282.

[13] Sandhu, R.S. and Jajodia, S. "Polyinstantiation for Cover Stories." *Proc. European Symposium on Research in Computer Security*, Toulouse, France, November 1992, pages 307-328. Published as *Lecture Notes in Computer Science, Vol 648, Computer Security—ESORICS92* (Deswarte, Y., Eizenberg, G., and Quisquater, J.-J., editors), Springer-Verlag, 1992.

[14] Sandhu, R.S. and Jajodia, S. "Referential Integrity in Multilevel Secure Databases." *16th National Computer Security Conference*, Baltimore, MD, Sept. 20-23, 1993, pages 39-52.

[15] Sandhu, R.S., Jajodia, S. and Lunt, T. "A New Polyinstantiation Integrity Constraint for Multilevel Relations." *Proc. IEEE Workshop on Computer Security Foundations*, Franconia, New Hampshire, June 1990, pages 159-165.

[16] Smith, K. and Winslett, M. "Entity Modeling in the MLS Relational Model." *Proc. of the 18th VLDB Conference*, Vancouver, British Columbia, Canada, 1992, pages 199-210.

[17] Thuraisingham, B.M. "A Nonmonotonic Typed Multilevel Logic for Multilevel Secure Database/Knowledge-Base Management Systems." *Proc. of the Fourth IEEE Workshop on Computer Security Foundations*, 1991, pages 127-138.