Secure Databases: An Analysis of Clark-Wilson Model in a Database Environment

Xiaocheng Ge[1], Fiona Polack[1], and Régine Laleau[2,⋆]

[1] Department of Computer Science, University of York
York, YO10 5DD, UK
{xchge,fiona}@cs.york.ac.uk
Fax: +44 1904 432767
[2] Research Laboratory LACL, IUT Fontainebleau, Université Paris 12
Route forestière Hurtault 77300 Fontainebleau, France
laleau@univ-paris12.fr

Abstract. Information systems are vulnerable to accidental or malicious attacks. Security models for commercial computer systems exist, but information systems security is often ignored or added at or after implementation. The paper explores common security models, and their relevance to databases. It demonstrates how security-relevant concepts can be extracted during a conventional database development.

Keywords: Databases, security models, access control, data integrity, development methods

1 Introduction

This paper considers security models for information systems (ISs); the work is part of ongoing research into a development process for commercial databases that incorporates security. The research objective is to incorporate, in a formally verifiable way, the fundamental requirements of commercial security. For simplicity, we assume a target implementation of a relational DBMS and SQL3[13].

This section introduces key security concepts, outlines the overall research plan, and summarises existing security models. Section 2 explores the Clark-Wilson security model in the context of ISs. Section 3 looks at designing for security with a conventional database development and SQL3 implementation. The case study is necessarily brief, and does not cover the formal verification of the security content, or conventional verification techniques such as normalisation. Section 4 compares our approach to existing work, whilst section 5 presents our conclusions in the context of our ongoing research.

1.1 Background of Database Security

ISs are important to the modern society. Information stored in databases is a valuable resource that enables an organisation to operate effectively. Modern

⋆ Prof. Laleau's contribution is supported by an EPSRC visiting fellowship, grant 006R02664.

A. Persson and J. Stirna (Eds.): CAiSE 2004, LNCS 3084, pp. 234–247, 2004.

organisations are so dependent on the proper functioning of their ISs that corruption or loss of data has serious consequences.

Database security is concerned with ensuring the *confidentiality (or secrecy), integrity,* and *availability* of stored data. *Confidentiality* is the protection of data from unauthorised disclosure either by direct retrieval or by indirect logical inference; it also concerns the possibility that information may be disclosed by legitimate users acting as an information channel, passing secret information to unauthorised users. *Integrity* requires data to be protected from invalid modification, insertion or deletion. Integrity constraints are rules that define the correct states of a database, and maintain correctness under operation. *Availability* ensures that data is available to authorised users. Availability is very closely related to integrity because service denial may cause or be caused by integrity violations.

Database security is not an isolated problem; it is affected by other components of a system, such as the operating system (OS). The security requirements of a system are specified by means of a security policy and enforced by security mechanisms. For databases, [21, 5, 20] classify the secure database requirements.

Our research focuses on database integrity, and those aspects of confidentiality that relate to data protection, namely access control. Of the following security requirements, which are the minimum that need to be supported by the IS, the first relates directly to integrity; the other two relate to confidentiality.

1. *Integrity, Consistency.* Semantic integrity constraints are rules defining the correct states of the system during operation; they exist to protect against malicious or accidental modification of data, and ensure the logical consistency of data. Rules can be defined on the static state of the database, or on transitions (as conditions to be verified before data is modified).
2. *Identification, Authentication, Audit.* Before accessing a system, every user is identified and authenticated, both for the audit trail and for access permission. Auditing is the process of examining all security relevant events.
3. *Authorisation (access control).* Authorisation applies a set of rules that defines who has what type of access to which information. Access control policies govern the disclosure and modification of information.

In the context of requirement engineering, security aspects should not be afterthoughts of database design process. We cannot simply *'firewall* databases, because firewalls cannot do anything against invalid modification by authorised users. We need an IS development method that incorporates security aspects.

1.2 Secure Database Design Process

Our design process (Figure 1) extends a conventional database design process[10] with security; it includes a formalisation (referred to as OAZIS), to support verification of the integrity of state and operations.

The first step, **requirements collection and analysis**, documents users' data and functional requirements (ie required transactions). There are three types of security requirement: logical, physical and organisational. We concentrate on logical requirements, derived by analysis of threats and risks.

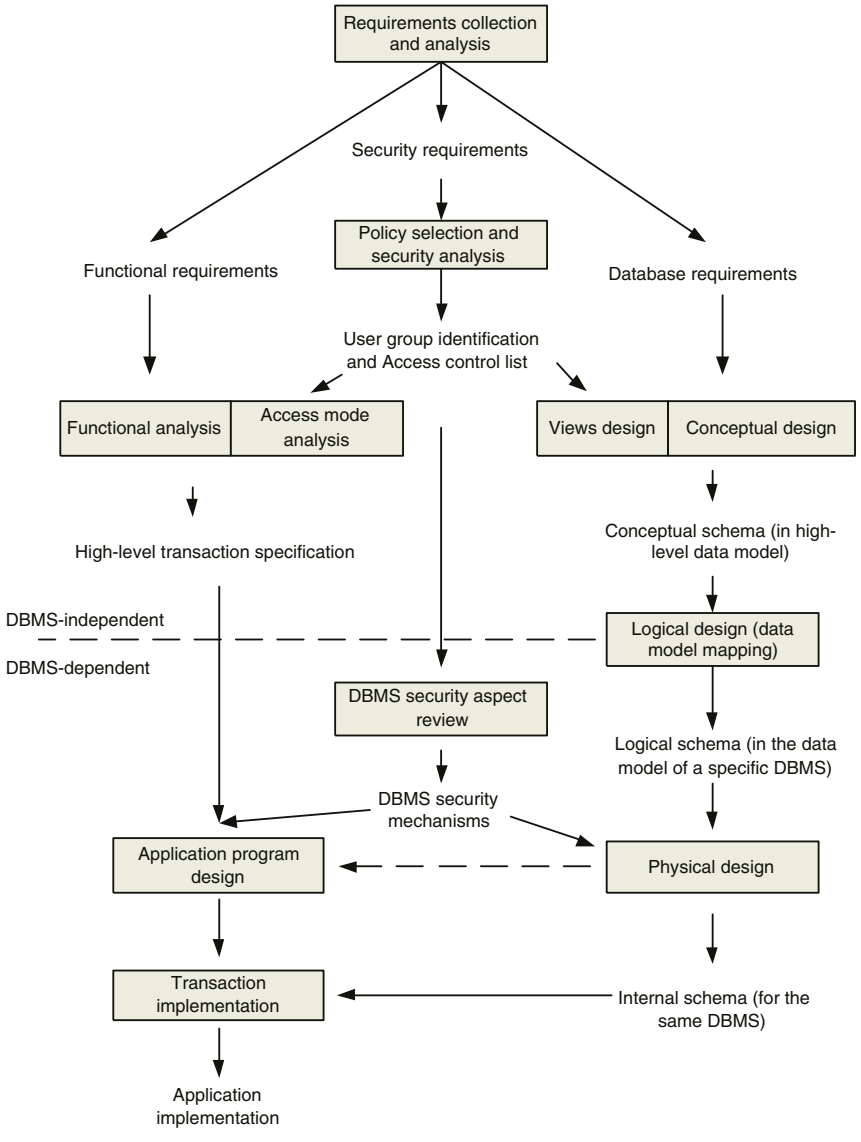


Fig. 1. The development process of OAZIS method

The logical security requirements are the basis for **security analysis and policy selection**. This step is crucial. The security policy determines the access mode for each subject (or role) on each object (data, operations). The permissions of each user role are specified, and the access control list determined.

Once requirements have been analysed, **functional analysis** and **conceptual design** produce a conceptual schema, a concise description of data types, relationships, and constraints. Sub-schemas are identified to aid the expression

of security constraints and access control. Basic data operations (create, delete, and update) are used to specify transactions – both the user-required functional transactions and those relating to the chosen security policy. The results of these steps need to be verified against the security requirements, so development iterates between conceptual modelling and model verification.

Logical design translates the conceptual model into a logical model for a specific DBMS. Analysis of the security features in the conceptual model establishes which security requirements can be achieved by OS and DBMS security mechanisms, or by specific security packages, resulting in a logical security model. If any security requirements in the conceptual model cannot be addressed using available mechanisms, the developer should design further specific mechanisms.

Finally, **physical design** implements the internal database structures, including security mechanisms. Application programmes are coded for those parts of transactions that cannot be implemented directly on the chosen DBMS.

1.3 Literature of Security Models

There are many security policies and models in the literature, relevant to various environments. In a military environment, confidentiality is critical – all classified information shall be protected from unauthorised disclosure or declassification – so models focus on mandatory classification. For example, Bell-LaPadula [3] and its derivatives describe models for confidentiality, whilst Biba [4] defines a similar level-oriented integrity model. In the commercial environment, the goal is to prevent fraud and errors – no user, even if authorised, should be able to modify data in an invalid way – so models focus on integrity enforcement and authorisation mechanisms to prevent illegal modification. The seminal work is Clark-Wilson's integrity model.

Policies and models are implemented by security mechanisms, which can be either *discretionary* or *mandatory*. *Discretionary* models include mechanisms for granting and delegating access permissions by (some) system users. *Mandatory* security is built-in and cannot be changed by system users. These models govern information access on the basis of classifications of subject and object[1].

For our commercial security requirements, the Clark-Wilson model results in a conceptual security model that is defined by identification of,

- data items for which security enforcement is crucial (CDIs);
- transformation procedures (TPs) that can access data;
- user roles, in terms of authorisation to use particular TPs.

The access control is specified as an access triple, $< user, tp, data >$, stating that a *user* has permission to execute *tp* on *data*. Implementation is usually discretionary, but there is no fundamental reason why a Clark-Wilson triple could not be implemented as a mandatory security mechanism.

[1] A subject is a person or application that actively accesses data/processes; objects are passive data or processes stored in the IS. For military systems, the implementation of mandatory security mechanisms is described in the U.S. Department of Defence Trusted Computer System Evaluation Criteria (the Orange Book) [9].

To enforce basic access control and integrity mechanisms, Clark-Wilson identifies two principal mechanisms. The **well-formed transaction** preserves data integrity and prevents users from arbitrarily manipulating data. **Separation of duty** dictates that each critical operation comprises two or more subparts, each of which has to be executed by a different user role. Our research is concerned with how these mechanisms can be established during the database design process.

2 Clark-Wilson and Information Systems

The Clark-Wilson security model derives from commercial data processing practices. It is based on time-tested business methods; thus it represents a real-world approach, rather than an academic exercise. Furthermore, the Clark-Wilson model can be used to evaluate the security of a complete system, rather than just the subject-object accesses [11]. The focus on data integrity and well-formed transactions makes it particularly attractive for database systems.

2.1 The Clark-Wilson Model

In 1987, Clark and Wilson proposed their commercial security model [6]. It can be used for systems where integrity is enforced across both the OS and the application. Clark-Wilson was extended to cover separation of duty in 1993 [1].

Clark-Wilson is not the first approach to model the integrity aspect of security. Biba [4] defined an integrity model based on the security classifications of subjects and objects, using *integrity level* for classification. Later, Lipner [17] tried to describe integrity using the Bell-LaPadula model. In his model, a list of users is attached to transactions and data separately, to ensure that data can only be manipulated by certified transactions. These are all *lattice* models, with security verification based on the mathematical theory of lattices and relations. For IS, they are inadequate as they do not restrict data manipulation to programs that implement well-formed transactions.

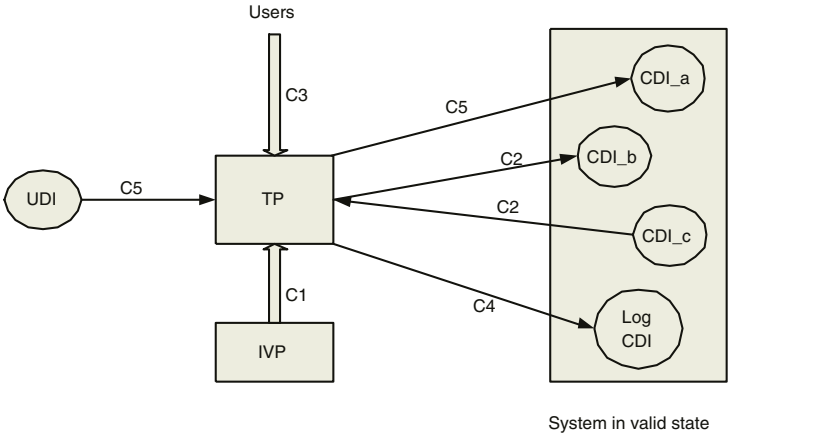
In Clark-Wilson, each datum in the system is classified as either a *constrained data item* (CDI) or an *unconstrained data item* (UDI). CDIs must be protected, whilst UDIs are conventional data objects whose integrity is not assured under the model. No datum can be in both classes:

$$Data = CDI \cup UDI \wedge CDI \cap UDI = \varnothing$$

Operations on CDIs are performed by TPs and *integrity verification procedures* (IVPs)[2]. IVPs ensure that all CDIs conform to some application-specific model of integrity. TPs change the state of the set of CDIs.

Appendix A lists Clark-Wilson rules for certification, enforcement, and separation of duty. *Enforcement rules* specify security requirements that should be

[2] Although it is tempting to think of a TP as a user transaction, the analogy is unsound, as we will see later.



Certification Rules

C1: IVP validates CDI state

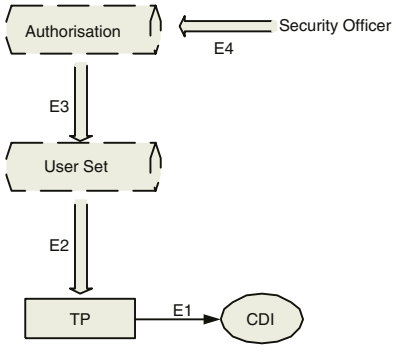
C3: Suitable separation of duty

C5: TPs validate UDI

C2: TPs preserve valid state

C4: TPs write to log file

(a) Certification Rules



Enforcement Rules

E1: CDIs changed only by authorised TP

E3: Users are authorised

E2: Users authorised for TP

E4: Authorisation lists changed only by security officer

(b) Enforcement Rules

Fig. 2. Certification and Enforcement Rules of the Clark-Wilson Model

supported by the protection mechanisms in the underlying system. *Certification rules* specify security requirements that the application system should uphold.

Figure 2 (derived from [6]) shows how these rules apply to data management. UDIs represent data that exists outside the secure system. Certification rules ensure that such data is properly validated on entry to the system – for

example, rule C5 requires that well-formed TPs that convert UDIs to CDIs perform only the complete, certified transformations; rules C1 and C2 require that CDIs conform to the IVPs on entry and under subsequent transformations. Rule C4 requires the logging of all transactions, as is normal for databases – though database logging is for rollback, whilst Clark-Wilson logging is for audit; rule C3 requires appropriate separation of duties. Since data can only be entered in accordance with the certification rules, for the systems in which we are interested, it follows that all data in the database are CDIs. The enforcement rules prevent modification of CDIs in ways that contravene the IVPs. Rules E2 to E4 relate to TP authorisation of access, whilst remaining rules ensure that only well-formed, certified TPs can be used to modify CDIs.

2.2 Applying Clark-Wilson Using a DBMS

Conventional DBMSs support many of the Clark-Wilson mechanisms for access authorisation and control. However, implementations based on standard SQL require some compromises. SQL3 access control mechanisms are primarily on data not transactions, so the access-control triples cannot be directly or fully implemented for user transactions. Inspiration for implementation mechanisms comes from Lee [16] and Shockley [23], who independently developed an implementation of the Clark-Wilson model, using the Biba model categories and trust subjects to interpret access triple authorisations at the data level.

Figure 3 shows a classical DBMS and the related OS and programming functions. The fundamental database principle, that data can only be accessed via the DBMS, is assumed, and the DBMS provides authorisation checking, transaction and data management and logging. The OS authentication also applies as normal and can be extended at the DBMS interface, for example with extra access rules. We now consider how these concepts can be related to the Clark-Wilson rule-application in Figure 2.

First, we consider validation of UDIs. In figure 3, the application object, outside the DBMS box, represents a UDI. Following rule C5, the application object is processed by an application program, invoking integrity enforcement procedures such as procedure preconditions or an integrity contract. During execution, connection to the DBMS server is established, and the user who is executing the application program is authenticated by DBMS-level authorisation. If authentication succeeds, then a database transaction takes over, applying its own checks on data integrity via the DBMS integrity enforcement. The transaction is logged, part in the DBMS transaction log and part by the OS (rule C4).

Secondly, we review the implementation of the enforcement rules on CDIs. At login, users are checked by OS authentication; they can then access either application programs or database transactions according to the relevant access rules (rule E3). Under rule E4, access permissions can only be modified by a specific user role (security officer). Enforcement rule application is strongest if as much processing as possible takes place under the control of DBMS access rules. This is the case with *Stored Procedures* (supported by, for instance, ORACLE[19], IBM DB2[12], and Microsoft SQL server [18]) and DBMS programming facilities such

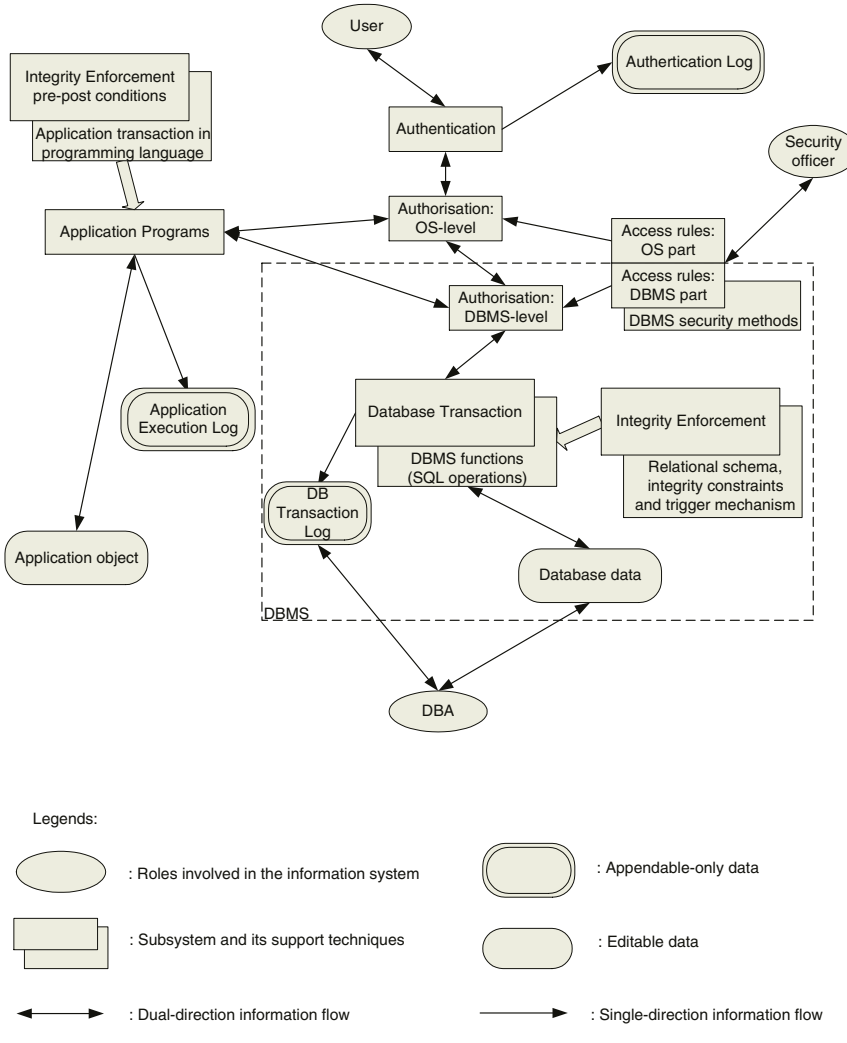


Fig. 3. DBMS classical architecture

as Oracle's PL/SQL. For complex algorithms, library procedures, graphics, and access to other systems, it is necessary to use program code managed by the OS; each time a CDI is exported to an embedding program, it reverts to UDI status.

A typical database transaction is made up of a number of separate TPs, some of which convert UDIs into DBMS CDIs, and some of which update CDIs. Transactions must also implement IVPs. Most database transactions (and their associated access control) must therefore conform to the Clark-Wilson rules. Access triples grant access to whole TPs. However, most SQL authorisation mechanisms are defined on data and simple commands using the **GRANT** statement (coupled with views):

GRANT *list of privileges*
ON *data object*
TO *list of users*
[**WITH GRANT OPTION**]

Access is given **TO** specified users and roles, **ON** specified data structures. The access can be via any of the basic commands listed in the **GRANT** statement. The basic commands are **SELECT**, **DELETE**, **INSERT**, **UPDATE**, **REFERENCES**, **TRIGGER** and **EXECUTE**. One side-effect of SQL access control is to reduce the likelihood that transactions commit. For instance, consider transaction T:

```
BEGIN T
  SELECT * FROM X;
  DELETE FROM X WHERE ...;
COMMIT T
```

On table X, role U has permission for **SELECT** and **DELETE**, whilst role V has only **SELECT** permission. If U executes T there is no problem, provided that integrity constraints are not violated. If, however, V executes T, the transaction always aborts. Here, a solution would be to implement T as a stored procedure, and grant **EXECUTE** permission on T only to U. This is not a general solution, as some transactions cannot be defined as stored procedures.

Application programs are beyond the scope of SQL access control, and can violate confidentiality. For example, if transaction T had additional program commands to store or pass on the value of `X.*`, the values stored in `X` would be available to users who might not have **SELECT** access to `X`.

A final problem with SQL authorisation is **WITH GRANT OPTION**. Although this is a popular concept, as it allows distributed management of authorisations, it contradicts Clark-Wilson rule E4.

Our development process addresses these limitations by considering security mechanisms during design, as well as at implementation. Our eventual goal is to use a formal language to specify both the security policy and the functional requirements, and to check their mutual consistency. We can derive implementations that meet the specification, using SQL integrity and access control, stored procedures, and application code.

3 Designing for Security: A Case Study

We now illustrate some aspects of our process. Our research shows that,

- access triples, TPs and separation of duty can be analysed in use case models;
- data details can be checked via class models, extended for modelling ISs to include the constraints needed to enforce data integrity;
- well-formedness can be designed in to transactions, checked using interaction diagrams, and implemented by the usual embedded SQL approaches;
- IVPs can be modelled as operation preconditions, event guards etc; these can be implemented in SQL constraint and trigger statements.

The scenario is a system for processing university examination papers.

Each academic year, thousands of students sit examinations. The papers have to be set by the lecturer, then checked by an Exam Board (EB). Students' scripts are marked by lecturers. Marks are checked and entered by administrators. The examination, processing and marks achieved are reviewed by EB, which has authority to modify marks. Finally, students are given access to their marks and degree grades.

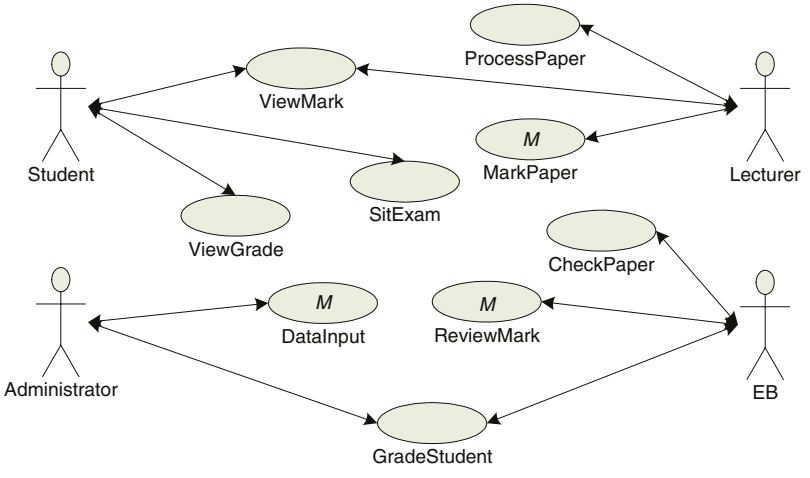


Fig. 4. Use cases of the examination management system

Figure 4 shows use cases for the system. Because of the characteristics of UML use case diagrams, each link between an actor and a use case presents an access triple in form of $< actor, usecase, data >$ – that is, each use case represents a TP. The implementation must enforce rule E3, that only programs that implement an access triple can be executed on the data.

Each use case models the processing of data into a valid final state (rule C2). For example, *Mark*, is accessed and modified by each of the use cases labelled *M*. The link between the actor Administrator and use case DataInput generates the access triple $< Administrator, DataInput, Mark >$; the implementation must check the integrity rules on *Mark* before it becomes a CDI in the database.

In order to make sure that the design meets the separation of duty requirement (rule C3), we can list all the access triples relating to the modification of each CDI. For the data item *Mark*, these are:

$$< Lecturer, MarkPaper, Mark >$$
$$< EB, ReviewMarks, Mark >$$
$$< Administrator, DataInput, Mark >$$

Three different roles are involved in processing *Mark* before a student can access it – separation of duty is preserved, at least at the conceptual level.

The conceptual model class diagram, defining structural integrity, is not illustrated here. In our approach, data constraints are expressed in a suitably formal language (UML recommends OCL; we use Z; elsewhere, we also recommend B [22]). Transactions are specified using UML interaction diagrams, with well-formedness checked by ensuring that structural integrity is maintained; we can also translate the models to a formal language for analysis. Rule C1 says that all IVPs must properly ensure that all CDIs are in a valid state when an IVP is executed – the modelled constraints effectively specify Clark-Wilson IVPs.

IVPs exist for all three mark-processing transactions (some related to the wider organisation):

- the lecturer's marking must conform to published marking criteria;
- at entry, values are checked against data domains and other constraints;
- the EB checks human aspects of the examining system – illness, academic misconduct, exam irregularities – and adjusts marks accordingly, but within the data constraints, plus time constraints imposed by the university.

Part of the IVP controlling data input relates to the constraint that the value of *Mark* must be an integer on the university mark scale, 0 and 100. In SQL, we can implement this either as a **CHECK** statement, or as a **TRIGGER**:

```
CREATE TRIGGER EnforceMark BEFORE INSERT
  ON achievement
  REFERENCING NEW ROW AS new
BEGIN
  IF new.mark > 100 THEN ROLLBACK
  ELSE COMMIT
END IF
```

A well-formed transaction combines such clauses and TPs, enforcing integrity.

4 Related Work

The main recent work on designing secure systems is *UMLsec* [14, 15], an extension of UML to include standard security requirements for critical systems, targeted at general security-critical systems design. UMLsec extends use case diagrams, activity diagrams, class diagrams, sequence diagrams, statechart diagrams, and deployment diagrams. It covers a wide area of information security, providing a rich set of security semantic in UML diagrams.

Like UML, the UMLsec graphical notation can be used with any development process, but does not directly represent IS characteristics such as keys and transactions. UMLsec's philosophy is based on the lattice models' multi-level security classification; Clark-Wilson is not level-oriented, and separation of duty is outside the scope of UMLsec. We cannot use UMLsec as the basis for our development process.

In terms of the application of security models to IS, Cuppens et al [8, 7] reviewed applicable models, and have formally specified, in deontic logic, rules

and obligations for database confidentiality, integrity and availability. Prolog implementation is used to check rules for contradiction. Cuppens' work is more extensive than ours, and expresses many of the security aspects covered by the Clark-Wilson rules. However, publications do not address the completeness or consistency of the formal security models. Implementation of security mechanisms does not relate to commercial DBMSs and SQL, focusing instead on object-oriented databases with a novel prolog-based query language.

5 Conclusion

In this paper, we summarise support for the Clark-Wilson security model in a conventional DBMS context, and an approach to database design that builds security requirements into the design.

The discussion of security models suggests that the Clark-Wilson focus on well-formed transactions makes it appropriate for ISs. Indeed, the main disadvantage normally cited for Clark-Wilson, that the IVPs and associated techniques are not easy to implement in real computer systems [2], is largely overcome in the database context. For relational database, some integrity constraints are inherent in the theory (entity and referential integrity); others can be stated as static constraints using SQL. Some dynamic integrity constraints can be implemented using the SQL3 triggers, and others can be stated in code. These enforce the integrity of CDIs accessed and modification by TPs.

Although conventional DBMSs have most of the security features needed to implement the Clark-Wilson rules, access triples are not fully supported; a combination of OS and DBMS facilities is required. A verification of security can be achieved by calculating the overall data accesses of the implemented transactions and ensuring that these match triples constructed in design for each required transaction.

The case study extract presents part of an approach for building security requirements into the development process. The conventional conceptual models used for ISs specification and design provide the basis for expressing, checking and implementing the necessary security features. Work is ongoing on the detail of the development process, incorporating the formal analysis of system integrity, and dynamic TP/IVP enforcement of integrity by transactions.

The ability to map designed security features to the SQL concepts supported by current DBMSs is also critical to the success of our approach. The case study shows just one aspect of this – the derivation of a trigger to enforce a simple data constraint. We are devising template translations from our conceptual and formal models to SQL, and are working on a prototype of a tool that can express formally-verified integrity rules as appropriate SQL constraints and triggers.

References

1. M. Abrams, E. Amoroso, L. LaPadula, T. Lunt, and J. Williams. Report of an integrity research study group. *Computers and Security*, 12:679–689, 1993.
2. E. Amoroso. *Fundamentals of Computer Security Technology*. Prentice Hall, 1994.
3. D. E. Bell and L. J. LaPadula. Secure computer systems: Mathematical foundations and model. Technical Report MTR 2547 v2, MITRE Corporation, 1973.
4. K. J. Biba. Integrity constraints for secure computer systems. Technical Report EST TR-76-372, Hanscom AFB, 1977.
5. S. Castano, M. Fugini, G. Martella, and P. Samarati. *Database Security*. Addison-Wesley, 1994.
6. D. D. Clark and D. R. Wilson. A comparison of commercial and military computer security policies. In *IEEE Symposium on Security and Privacy*, pages 184–194, Oakland, April 1987.
7. F. Cuppens. *Modélisation formelle de la sécurité des systèmes d'informations*. Habilitation, Paul Sabatier University, Toulouse, France, 2000.
8. F. Cuppens and C. Saurel. A logical formalization of integrity policies for database management systems. In S. Jajodia, W. List, G. W. McGregor, and L. Strous, editors, *Integrity and Internal Control in Information Systems*. Kluwer, 1998.
9. DOD. TCSEC: Trusted computer system evaluation criteria. Technical Report 5200.28-STD, U.S. Department of Defense, 1985.
10. R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems*. Benjamin Commings, 2nd edition, 1994.
11. S. N. Foley. The specification and implementation of "commercial" security requirements including dynamic segregation of duties. In *4th ACM Conf. on Computer and Communications Security*, pages 125–134. ACM Press, April 1997.
12. IBM. DB2 universal database: SQL reference, release 7. IBM Corporation, 2000.
13. ISO. International standard – SQL. Technical report, ISO/IEC 9075-1, 1999.
14. J. Jürjens. Towards development of secure systems using UML. In *FASE 2001, Genova, Italy*, volume 2029 of *LNCS*, pages 187–201. Springer Verlag, April 2001.
15. J. Jürjens. UMLsec: Extending UML for secure systems development. In *UML 2002, Dresden, Germany*, volume 2460 of *LNCS*, pages 412–425. Springer Verlag, Sept-Oct 2002.
16. T. M. P. Lee. Using mandatory integrity to enforce "commercial" security. In *IEEE Symposium on Security and Privacy*, pages 140–146, Oakland, April 1988.
17. S. B. Lipner. Non-discrentionary controls for commercial applications. In *IEEE Symposium on Security and Privacy*, pages 2–10, Oakland, May 1982.
18. Microsoft. SQL server, version 7.0. Microsoft Corporation, 1999.
19. Oracle. Oracle8i SQL reference, release 8.1.6. Oracle Corporation, 1999.
20. G. Pernul, W. Winiwarter, and A. Min Tjoa. The entity-relationship model for multilevel security. In *Int. Conf. on Conceptual Modeling / the Entity Relationship Approach*, pages 166–177, 1993.
21. C. P. Pfleeger and S. L. Pfleeger. *Security in Computing*. Prentice Hall, 3rd edition, 2003.
22. F. Polack and R. Laleau. A rigorous metamodel for UML static conceptual modelling of information systems. In *CAiSE 2001, Interlaken, Switzerland*, volume 2068 of *LNCS*, pages 402–416. Springer Verlag, June 2001.
23. W. R. Shockley. Implementing the Clark/Wilson integrity policy using current technology. In *11th National Computer Security Conference*, pages 29–37, Baltimore, October 1988.

A Clark-Wilson Certification Enforcement, and Separation of Duty Rules

The following rules are directly quoted from [6]:

C1: All IVPs must properly ensure that all CDIs are in a valid state at the time the IVP is run.

C2: All TPs must be certified to be valid. That is, they must take a CDI to a valid final state, given that it is in a valid state to begin with. For each TP, and each set of CDIs that it may manipulate, the security officer must specify a "relation", which defines that execution. A relation is thus of the form: $(TP_i, (CDI_a, CDI_b, CDI_c, \ldots))$, where the list of CDIs defines a particular set of arguments for which the TP has been certified.

E1: The system must maintain the list of relations specified in rule C2, and must ensure that the only manipulation of any CDI is by a TP, where the TP is operating on the CDI as specified in some relation.

E2: The system must maintain a list of relations of the form:

$$(UserID, TP_i, (CDI_a, CDI_b, CDI_c, \ldots))$$

which relates a user, a TP and the data objects that TP may reference on behalf of that user. It must ensure that only executions described in one of the relations are performed.

C3: The list of relations in E2 must be certified to meet the separation of duty requirement.

E3: The system must authenticate the identity of each user attempting to execute a TP.

C4: All TPs must be certified to write to an append-only CDI (the log) all information necessary to permit the nature of the operation to be reconstructed.

C5: Any TP that takes a UDI as an input value must be certified to perform only valid transformations, or else no transformations, for any possible value of the UDI. The transformation should take the input from a UDI to a CDI, or the UDI is rejected.

E4: Only the agent permitted to certify entities may change the list of such entities associated with other entities: specifically, those associated with a TP. An agent that can certify an entity may not have any execute rights with respect to that entity.

The following rules are from [1]:

SP1: User roles should be administered by two different agents: one agent assigns roles to users, but is constrained by information in the system that defines the roles. The other agent can define roles.

SP2: The use of so-called "primary CDIs" is recommended to support separation of duty. Primary CDIs have values that require corroboration by two or more different users. A primary CDI should change only as a result of the last TP in an enabling sequence.

SP3: To apply integrity to mechanisms that implement integrity, access triples can be protected from unauthorised modification by storing them in *CDI-triples* and restricting access to the *Triple Manager* by a role assignment. Similarly, TPs and IVPs can be protect from unauthorised modification by assigning roles and including TP/IVP management TPs in appropriate access rights.