Journal of Computational Science 5 (2014) 653–663

Contents lists available at ScienceDirect

Journal of Computational Science

journal homepage: www.elsevier.com/locate/jocs

Assuring consistency in mixed models

Ramzi A. Haraty*, Mirna F. Naous, Azzam Mourad

Department of Computer Science and Mathematics, Lebanese American University, Beirut, Lebanon

ARTICLE INFO

Article history:
Received 10 September 2013
Received in revised form 8 January 2014
Accepted 21 February 2014
Available online 3 March 2014

Keywords:
Role based access control
Clark Wilson
Consistency
Alloy

ABSTRACT

Information systems security defines three properties of information: confidentiality, integrity, and availability. These characteristics remain major concerns throughout the commercial and military industry. Ordinary users have taken these features as basis for their businesses. Furthermore, users may find it necessary to combine policies in order to protect their information in a suitable way. However, inconsistencies may arise as a result of implementing multiple secrecy and privacy models; and therefore, render these services unsecure. In this paper, we propose an approach to detect and report inconsistencies when choosing mixed models for integrity and security. It is based on specifying the policies in first order logic and applying formal analysis. We demonstrate the feasibility of our proposition by applying it to the Clark Wilson and role based access control models. We use the Alloy language and analyzer to formalize the mixed model and check for any inconsistencies.

1. Introduction

The goal of information systems is to control or manage the access of subjects (users, processes) to objects (data, programs) [11]. This control is governed by a set of rules and objectives called a security policy. Data integrity is defined as "the quality, correctness, authenticity, and accuracy of information stored within an information system" [4]. Systems integrity is the successful and correct operation of information resources. Integrity models are used to describe what needs to be done to enforce the information integrity policies. There are three goals of integrity:

- Prevent unauthorized modifications,
- Maintain internal and external consistency, and
- Prevent authorized but improper modifications.

Combining multiple secrecy and privacy policies in the same service may cause inconsistencies. An inconsistency or contradiction may undermine the proper functioning of the service creating instances where some users may be allowed access to information according to a rule, while not allowed according to another. Clearly, it is important that inconsistencies be detected before the online deployment of the system. Furthermore, users may not understand the implications of adding or removing policies, which may grant rights that threaten privacy, or revoke rights that are instrumental to information sharing.

Before developing a system, one needs to describe formally its components and the relationships between them by building a model. The model needs to be analyzed and checked to figure out possible bugs and problems. Thus, formalizing integrity security models helps designers to build a consistent system that meets its requirements and respects the three goals of integrity. This objective can be achieved through the Alloy language and its analyzer.

Alloy is a structural modeling language for software design. It is based on first order logic that makes use of variables, quantifiers and predicates (Boolean functions) [6]. Alloy, developed at MIT, is mainly used to analyze object models. It translates constraints to Boolean formulas (predicates) and then validates them using the Alloy Analyzer by checking code for conformance to a specification [18]. Alloy is used in modeling policies, security models and applications, including name servers, network configuration protocols, access control, telephony, scheduling, document structuring, and cryptography. Alloy's approach demonstrates that it is possible to establish a framework for formally representing a program implementation and for formalizing the security rules defined by a security policy, enabling the verification of that program representation for adherence to the security policy.

There are several policies applied by systems for achieving and maintaining information integrity. In this paper, we focus on the role based access control and Clark Wilson and to show how they can be checked for consistency or inconsistency using the Alloy language and the Alloy Analyzer.

* Corresponding author. Tel.: +961 1867620; fax: +961 1867098.
E-mail address: rharaty@lau.edu.lb (R.A. Haraty).

http://dx.doi.org/10.1016/j.jocs.2014.02.009
1877-7503/

The remainder of this paper is organized as follows: Section 2 provides the literature review. Section 3 discusses the role based access control security model. Section 4 discusses the Clark Wilson security model. Section 5 present the mixed model and Section 6 concludes the paper.

2. Literature review

Modeling and validation has been used extensively in the literature [3–13]. Hassan and Logrippo [20] proposed a method to detect inconsistencies of multiple security policies mixed together in one system and to report the inconsistencies at the time when the secrecy system is designed. The method starts by formalizing the models and their security policies. The mixed model is checked for inconsistencies before real implementation. Inconsistency in a mixed model is due to the fact that the used models are incompatible and cannot be mixed.

Zao et al. [9] developed the RBAC schema debugger. The debugger uses a constraint analyzer built into the lightweight modeling system to search for inconsistencies between the mappings among users, roles, objects, permissions and the constraints in a RBAC schema. The debugger was demonstrated in specifying roles and permissions according and verifying consistencies between user roles and role permissions and verifying the algebraic properties of the RBAC schema.

Hassan et al. [21] presented a mechanism to validate access control policy. The authors were mainly interested in higher level languages where access control rules can be specified in terms that are directly related to the roles and purposes of users. They discussed a paradigm more general than RBAC in the sense that the RBAC can be expressed in it.

Shaffer [1] described a security Domain Model (DM) for conducting static analysis of programs to identify illicit information flows, such as control dependency flaws and covert channel vulnerabilities. The model includes a formal definition for trusted subjects, which are granted privileges to perform system operations that require mandatory access control policy mechanisms imposed on normal subjects, but are trusted not to degrade system security. The DM defines the concepts of program state, information flow and security policy rules, and specifies the behavior of a target program.

Misic and Misic [8] addressed the networking and security architecture of healthcare information system. This system includes patient sensor networks, wireless local area networks belonging to organizational units at different levels of hierarchy, and the central medical database that holds the results of patient examinations and other relevant medical records. In order to protect the integrity and privacy of medical data, they proposed feasible enforcement mechanisms over the wireless hop. Haraty and Naous [15–17] also modeled the clinical information systems policy. The authors used the Alloy formal language to define these models and the Alloy Analyzer to validate their consistency.

Haraty et al. [16] showed how secrecy policies can be checked and inconsistency by modeling the Chinese Wall Model [5], Biba Integrity Model [10], Lipner Model [19] and the Class Security Model [2]. In their work, they listed the ordered security classes (Top Secret, Secret, Confidential, and Unclassified) and their compartments (nuclear, technical, and biological) and defined those using signatures. Then, the possible combinations and the relationships between classes and compartments were specified. Facts were used to set the model constraints and to prove that the model is consistent. In the Biba Integrity model, the authors listed the subject security clearance and the object security classes and then modeled the constraints of how subjects can read/write objects based on "NoReadDown" and "NoWriteUp" properties. In [14], the author presented a comparison of different secure systems with their access control policies.

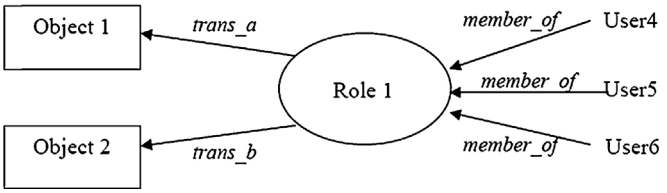

Fig. 1. Roles relatioship.

3. Role-based access control

In many organizations, "the end users do not own the information for which they are allowed access" [7]. However, the company is the actual owner of system objects as well as the programs that process it. Control is often based on employee functions and access control decisions are often determined by the users' roles. This suggests associating access with particular role of the user.

Since access to system modules is not tied to particular individual, role-based access control (RBAC) is used by many organizations to protect data integrity and restrict access to information based on users' roles. RBAC model defines the following entities:

- Role *r* is a collection of job functions. It is a set of transactions that a user or set of users can perform within an organization.
- Transactions are allocated to roles by a system administrator. Each role is authorized to perform one or more transactions *t*.
- Subject s has roles in the system. The active role of a subject is the role that is currently performing.

Fig. 1 shows Role 1 containing users 4, 5 and 6 as members. Users of Role 1 can execute transactions trans_a and trans_b on object 1 and object 2.

Ferraiolo in [7] determined the formal description of RBAC in terms of sets and relations as follows:

- For each subject, the active role is the one that the subject is currently using: *AR*(*s*: *subject*) = {*the active role for subject s*}.
- Each subject may be authorized to perform one or more roles:
- RA(s: subject) = {authorized roles for subject s}.
- Each role may be authorized to perform one or more transactions: *TA(r: role)* = {*transactions authorized for role r*}.
- Subjects may execute transactions. The predicate *exec(s,t)* is true if subject *s* can execute transaction *t* at the current time, otherwise it is false: *exec(s: subject, t: tran)* = *true iff subject s can execute transaction t.*

Accordingly, three basic rules are required in RBAC model:

- *Role assignment*: A subject can execute a transaction only if the subject has selected or been assigned a role. However, all active users are required to have some active role.
- *Role authorization*: A subject's active role must be authorized for the subject with role assignment; this rule ensures that users can take on only roles for which they are authorized.
- *Transaction authorization*: A subject can execute a transaction only if the transaction is authorized for the subject's active role.

Therefore, and according to RBAC model, a subject *s* can execute a transaction *t* if it has an active role *r* and its role is authorized to execute the transaction *t*. Also, RBAC can model the separation of duty rule since users in some roles cannot enter other roles. This means that users cannot perform the job functions of other roles.

Table 1
Procurement management system roles.

Role		Job description
Employee	RE	• Fills new purchase request and submits it to procurement officer for processing.
Procurement Officer	RP	• Checks submitted requests, contacts suppliers then fills purchase orders.
Supervisor	RS	• Reviews purchase orders then submit them to manager for approval. • Reviews the delivery of purchased items then submits it to manager for approval.
Manager	RM	• Approves purchase orders • Approves deliveries. • Approves Payments.
Store Keeper	RK	• Issues deliveries of purchased items.
Accountant	RA	• Inserts payments.

Moreover, some roles subsume others. This defines a hierarchy of roles. Granting access to a role *r* implies that access is granted for all roles containing *r*.

3.1. Role-based access control implementation

In order to implement the RBAC model, a procurement management system (PMS) is used to demonstrate model consistency. Table 1 summarizes available roles in the system and their main job functions regardless the name of users playing these roles.

Table 2 lists the transactions that can be performed in the procurement management system:

There are six roles in the system as described in Table 1. The manager role is the top role in the system. This role subsumes all other system roles. Thus, the manager is allowed to perform the tasks assigned to supervisor, accountant, procurement officer, store keeper, and employee roles. Furthermore, the supervisor role can do the work of procurement user, store keeper and the employee roles. However, it cannot handle the accountant tasks since accountant role is directly under the manager role in the hierarchy. Accountant, procurement officer and store keeper are three different roles in the system and they have the ability to perform their jobs functions as well as the employee functions.

Table 3 specifies system users, their roles and the transactions that are authorized to execute based on the defined roles. For example, in the PMS, Mirna is given the employee role only. The employee role can execute TIR (insert purchase request transaction). Accordingly, Mirna can execute TIR on behalf of her role. Any other active user given the RE role can execute TIR since the execution is assoicated with the particular job not the user. However, the supervisor role represented by Fadi Feghali is authorized to execute maily TRPo and TRP. But since Fadi's role is above the store keeper, procurement officer and employee roles, Fadi or any active user given this role can execute the transactions authorized by RP, RK and RE (i.e., TIP, TIPo, TID).

Table 2
Procurement management system transactions.

Transaction	Description
TIR	Insert purchase request.
TIPo	Insert purchase order.
TRPo	Review purchase order.
TAPo	Approve purchase order.
TID	Issue delivery
TRD	Review delivery
TAD	Approve delivery
TIP	Insert payment
TAP	Approve payment

Table 3
Procurement management system user roles.

User name		Role	Transaction
Mirna Naous	UM	RE	TIR
Hossam Abu Laban	UH	RP	TIR – TIPo
Fadi Feghali	UF	RS	TIR – TIPo – TRPo – TID–TRD
Nagy Karkour	UN	RM	TIR – TIPo – TRPo – TAPo – T ID – TRD -TIP – TAP
Rehab Salloum	UR	RA	TIR–TIP
Jaafar Hashem	UJ	RK	TIR–TID

The procurement management system can be represented using the Alloy language. The following sections provide detailed description of Alloy code.

- Section 1 declares the set of roles and transactions. According to the system there are six roles defined in Table 1. The supervisor role RS is under the manager role RM, the accountant role RA is under the RM, the procurement officer role RP is under RS, the store keeper role RK is under RS and RE is under RA, RK and RP.

```
//declaration of roles
abstract sig Roles{}
abstract sig Trs{executedby:some Roles}

one sig RM extends Roles{}//Role:Manager
one sig RS extends Roles{under:RM} //Role: Supervisor
one sig RA extends Roles{under:RM}//Role: Accountant
one sig RP extends Roles{under:RS}//Role:Procurement Officer
one sig RK extends Roles{under:RS}//Role:Store keeper
one sig RE extends Roles{under:RA+RK+RP} //Role: Employee
```

Section 1 – Procurement Management System Declaration.

- Section 2 defines the procurement system transactions as part of Trs set.

```
//declaration of transactions
one sig TIR,TIPo,TIP,TID,TRPo,TRD,TAPo,TAD,TAP extends Trs {}
```

Section 2 – Procurement Management System Transactions

- Section 3 specifies system users and their roles. For instance user Mirna UM is playing the employee role in the system, Hossam UH is playing the role of procurement officer RP, Fadi UF is playing the supervisor role RS, and so on.

```
//declaration of users
one sig UM in RE{}//user:Mirna
one sig UH in RP{}//user:Hossam
one sig UF in RS{}//User Fadi
one sig UN in RM{}//User Nagy
one sig UJ in RK{}//user Jaafar
one sig UR in RA{}// user rehab
```

Section 3 – Procurement Management System – System Users.

- Section 4 restricts the execution of transactions to certain roles. Table 3 determines the roles and the transactions that are allowed to execute in the procurement system. The transaction approve purchase order TAPo, approver payment TAP and approve delivery TAD need to be executed by the manager role only, other roles are not allowed to perform such action.

```
fact {

// Transaction approve PO, approve delivery, approve payment can be accessed by RM only
TAPo.executedby!=RE
TAPo.executedby!=RK
TAPo.executedby!=RP
TAPo.executedby!=RA
TAPo.executedby!=RS

TAD.executedby!=RE
TAD.executedby!=RK
TAD.executedby!=RP
TAD.executedby!=RA
TAD.executedby!=RS

TAP.executedby!=RE
TAP.executedby!=RK
TAP.executedby!=RP
TAP.executedby!=RA
TAP.executedby!=RS
```

Section 4 - Procurement Management System Transactions Constraints (Part 1)

- Section 5 states that the review purchase order transaction TRPo and the review delivery TRD cannot be accessed by employee RE, store keeper RK, procurement officer RP and accountant RA. Thus, TRPo and TRD can be executed by RS and since RS is under RM in the hierarchy then RM can execute them also.

```
//transaction Review PO and review delivery can be accessed by RS only or RM since RS under RM
TRPo.executedby!=RE
TRPo.executedby!=RK
TRPo.executedby!=RP
TRPo.executedby!=RA

TRD.executedby!=RE
TRD.executedby!=RK
TRD.executedby!=RP
TRD.executedby!=RA
```

Section 5 - Procurement Management System Transaction Constraints (Part 2)

3.2. Role-based access control and Alloy analysis

As shown previously, the Alloy analyzer helps checking system validity by generating system Meta model and by generating instances of the system based on its facts and predicates. Fig. 2 displays the procurement management system Meta model generated by the Alloy system.

The model maps the codes written in the previous sections into a graphical model. It shows the different roles of the system. Additionally, the figure displays the system users attached to their roles, as well as the system transactions. However, the meta model does not show any constraints. Executing the system using the Alloy analyzer will generate instances based on defined constraints. Testing system consistency is done by running the system predicates and generating possible instances then validating them. In order to test the constraints specified in the fact procedure, a predicate is written and executed.

- Section 6 declares an empty predicate used to test the system consistency based on the defined facts. Executing the example yields the output shown in Fig. 3 which shows that "an instance is found" and "Predicate is consistent.

```
// run example to show consistencies/ inconsistencies
pred example(){
}
run example
```

Section 6 - Procurement Management System Predicate

Clicking the link "Instance" will yield Fig. 4. The generated instance demonstrates the consistency of the system.

However, specifying a wrong predicate, such as stating that employee role RE can execute the insert Po transaction will cause inconsistency.

4. Clark Wilson security model

The Clark Wilson Model (CW) is a security model which ensures the integrity of objects. Clark and Wilson stated in [4] that "prevention of unauthorized disclosure of sensitive information, which is vitally important in military security, is less important in commercial applications where integrity of information and prevention of unauthorized data modification is essential". Therefore this model is mainly applied in commercial applications where integrity of data is essential and there is less care about authentication as in military applications.

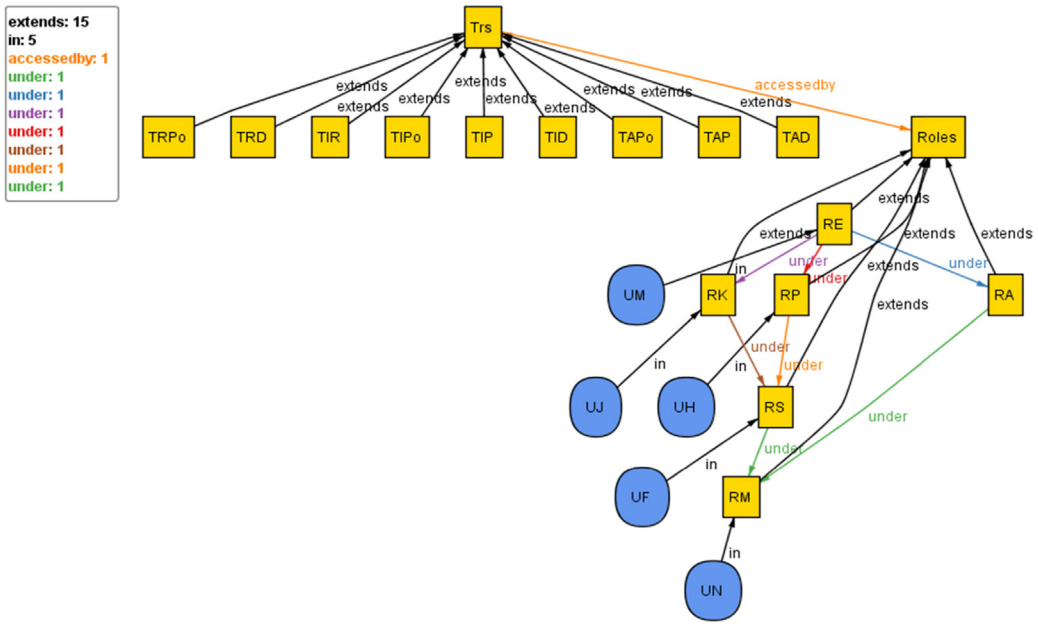

Fig. 2. Procurement management system meta model.

CW defines constrained data items (CDI) as objects requiring integrity and unconstrained data items (UDI) as objects outside the system not requiring integrity. The model also defines integrity verification procedures (IVP) as applications procedures which provide confirmation that all data adheres to security specifications on the system and transformation procedures (TP) as transactions which change the state of data in the system from one valid state to another. Additionally, the model addresses two main principles and a set of rules as follows:

The CW model principles are:

1. *The principle of separation of duty* states no single person should perform a task from beginning to end, but the task should be divided among two or more people to prevent fraud by one person acting alone.
2. *The principle of well-formed transaction* where user is able to manipulate data only in constrained ways. Thus only legitimate actions can be executed in a security system where transactions are well-formed. This will ensures that internal data is accurate and consistent to what it represents.

The CW model rules as stated in [12] are categorized into certification (CR) and enforcement (ER) rules as follows:

- *Certification Rule 1*: when any IVP is run, it must ensure that all CDIs are in a valid state.
- *Certification Rule 2*: For some associated set of CDIs, a TP must transform those CDIs in a valid state into a (possibly different) valid state.

```
Executing "Run example"
  Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
  148 vars. 66 primary vars. 137 clauses. 110ms.
  Instance found. Predicate is consistent. 109ms.
```

Fig. 3. Procurement management system consistent Alloy analyzer output.

- *Enforcement Rule 1*: The system must maintain the certified relations, and must ensure that only TPs certified to run on a CDI manipulate that CDI.
- *Enforcement Rule 2*: The system must associate a user with each TP and set of CDIs. The TP may access those CDIs on behalf of the associated user. The TP cannot access that CDI on behalf of a user not associated with that TP and CDI.
- *Certification Rule 3*: The allowed relations must meet the requirements imposed by the principle of separation of duty.
- *Enforcement Rule 3*: The system must authenticate each user attempting to execute a TP.
- *Certification Rule 4*: All TPs must append enough information to reconstruct the operation to an append-only CDI.
- *Certification Rule 5*: Any TP that takes as input a UDI may perform only valid transformations, or no transformations, for all possible values of the UDI. The transformation either rejects the UDI or transforms it into a CDI.
- *Enforcement Rule 4*: Only the certifier of a TP may change the list of entities associated with that TP. No certifier of a TP, or of an entity associated with that TP, may ever have an execute permission with respect to that entity.

Therefore, and based on CW model rules, the system to be implemented in the following section must ensure that only TPs certified to run on CDIs manipulate those CDIs and only users authorized to

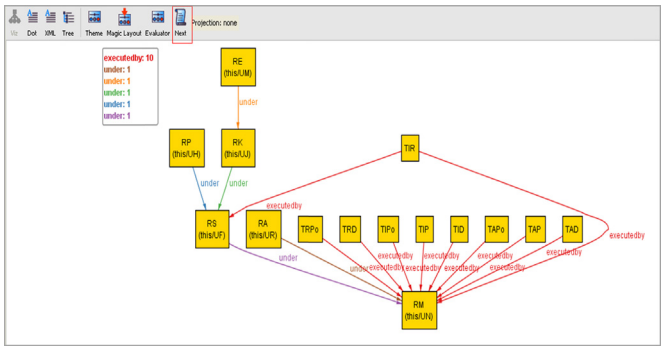

Fig. 4. Procurement management system instance.

Table 4
Transactions.

Transaction	Description
TD	Deposit money into account
TW	Withdraw from account
TT	Transfer money among accounts
TO	Open new account
TC	Close account
TB	Check account balance
TF	Check customer gift balance
TR	Register daily customers

Table 5
System users groups.

Group	Description
CG	Customers Group
TG	Tellers Group
JG	Janitors Group
MG	Accounts Managers Group

Table 6
Constrained data items sets.

CDI Set	Description
CAS	Customers' Accounts Set
LAS	Ledgers Accounts Set
PAS	Petty Cash Accounts Set

Table 7
Unconstrained data items sets.

UDI Set	Description
GBS	Gift Balances Set
CLS	Daily Customers Logs Set

execute the certified TPs can manipulate the CDIs. Moreover, no single user should perform a task from beginning to end, but the task should be divided among two or more users.

4.1. Clark Wilson model implementation

In order to implement the model, a bank system is used as example to demonstrate consistency with respect to CW certification and enforcement rules. For instance, in a bank the balances of accounts are considered CDIs because they are subject to integrity controls; however the gifts selected by accounts holders upon opening new account are UDIs because their integrity is not important with respect to the bank operations. Depositing, withdrawing and transferring money among accounts are examples of TPs. Regarding the framework used for implementation, Alloy analyzer and based on its features is considered a suitable tool for implementation

Table 4 lists the bank transactions, Table 5 defines the groups of system users, Table 6 determines the constrained data items available in the bank system and Table 7 lists the system unconstrained data items.

Table 8
Authorization.

Group	TP	CDI Set	UDI Set
Customers Group (CG)	TW	CAS	
	TB	CAS	
Accounts Managers Group (MG)	TO	CAS, LAS,PAS	
	TC	CAS, LAS,PAS	
	TB	CAS, LAS,PAS	
Tellers Group (TG)	TD	CAS, LAS,PAS	CBS
	TW	CAS, LAS,PAS	
	TT	CAS, LAS,PAS	
	TF		
Janitors Group (JG)	TR	–	CLS

Following the CW model, Clark and Wilson define the triple (user, TP, {CDI set}) such as each TP has a set of CDIs that can execute on. The user authorized to execute the TP on CDIs is placed into a group containing other users authorized to execute TPs on CDIs. Thus, Table 8 illustrates the authorization of each group with respect to available transactions, CDIs and UDIs sets. In the example, I assume that the customer can withdraw money only through ATM, check his/her personal account balance and cannot deposit or transfer money except through the bank tellers.

According to the above table, a customer belonging to customers group (CG) is able to execute only the withdraw money transaction (TW) and check account balance transaction (TB) of his/her account. The users belonging to accounts managers group are able to open new account (TO), close customer account (TC) and check an account balance (TB). These transactions affect the customer accounts (CAS), ledger accounts (LAS) and petty cash accounts set (PAS).Teller group users are able to deposit (TD), withdraw (TW), transfer (TT) money between accounts and check customer gift balance (TF). Their actions affect the customers (CAS), the petty cash accounts (PAS) and the gift balances accounts (GBS). Janitors are the users with the lowest authority in the system. Their main role is monitoring the bank gate and registering daily customers visiting the bank (TR). Thus, their action affect the daily customers logs sets (CLS). Although the customers gift balances (CGB) and daily customers logs (CLS) are accessible only tellers and janitors, these set of data are considered unconstrained data items since their integrity are not crucial to the bank operations. Therefore, any user regardless of his/her group can access the CBS and CLS without system prevention if he/she has the opportunity to perform such action. Moreover, and based on the concept of separation of duty, I assume that the group owner users can change the entities associated with each transaction and any other entities belonging to the group is not authorized to perform such change.

The bank system can be represented using the Alloy language. First, the system main entities represented in the above tables are declared as shown in the following sections of code. Section 7 declares the main entities of the bank system:

- BankDs: bank data set.
- BankCDIs: bank constrained data items.
- BankUDIs: bank unconstrained date items.
- BankUsers: bank users.
- BankTrs: bank transactions.

```
//System Entities declarations -- for simplicity we use one instead of some
abstract sig BankDS {} //Bank data sets
abstract sig BankCDIs extends BankDS{accessedby: one BankTrs} //CDIs set as part of BankDS
abstract sig BankUDIs extends BankDS{accessedby: one BankTrs} //UDIs set as part of BankDS
abstract sig BankUsers {}//Bank Users
abstract sig BankTrs {}//Bank Transactions
```

Section 7 - System Entities declaration

The Alloy model can contain comments by beginning a line with the character sequence "//". After the comment, comes the first signature declaration, "**abstract sig BankDS"**. It defines a set named BankDS representing the set of all data in the bank system. The "**abstract**" keyword is analogous to the abstract keyword of object oriented languages like Java. An abstract signature only contains atoms which are also contained in one of its extending signatures. However, if there are no extending signatures, then the abstract keyword has no meaning. For example, BankCDIs and BankUDIs extend the BankDS, therefore the atoms in BankDs belong either to BankCDIs to BankUDIs and not in both.

The keyword "**extends**" indicates that both BankCDIs and BankUDIs are subtypes of bank system datasets. This means two things: (1) the set of all BankCDIs is a subset of the set of all BankDS, and (2) BankCDIs are disjoint from other subtypes of BankDs (i.e. it is disjoint from BankUDIs). Anything can be done to BankDs can also be done to BankCDIs or BankUDIs. Following the signature declaration is its body, surrounded by curly braces. In the signature body, I can define series of relations that have this signature as their domain. Relations defined here behave like fields of an object in the object oriented paradigm. For instance in the body of BankCDIs I defined "**accessedby**" (section 1, line 2) which relates BankCDIs with BankTrs. It states that BankCDIs are accessed by bank transaction "BankTrs" declared in (section 1, line 5). Note that the two signature BankCDIs and BankUDIs extending the same signature BankDS are disjoint subsets of BankDS. The keyword "**one**" indicates the existence of one element in a signature. This makes sure that every reference to this signature refers to the same atom. The apposite keyword is "**some**" which indicates the existence of more than one element in a signature.

- Section 8 declares the system available transactions as part of bank transaction (BankTrs) defined in (section 7, line 5). Table 4 gives brief description of TD, TW, TT, TO, TC, TB, TF and TR.

```
//declaration of system transactions - Refer to table 1 in the report
//TD:deposit money,TW:withdraw money, TT:Transfer money,TO:open account,
//TC:close account, TB: check balance,TF: check account gift balance
//TR:Register daily customers visitor log
one sig TD,TW,TT,TO,TC,TB,TF,TR extends BankTrs{runby: one BankUsers,
                                                changedby:one BankUsers}
```

Section 8 - System Transactions

- Section 9 defines the bank system groups. As described in Table 5, there are 4 groups: CG, TG, JG and MG.

```
// declaration of users groups - Refer to table 2 in the report
one sig CG extends BankUsers{}//CustomersGroup
one sig TG extends BankUsers{}//TellersGroup
one sig JG extends BankUsers{}//JanitorsGroup
one sig MG extends BankUsers{}//AccountsManagersgroup
```

Section 9 - Users Groups

- Section 10 declares instances of users belonging to the groups defined in the previous section. C1, C2 and C3 are 3 different users in the customers group (CG). Similarly, T1 and T2 are 2 tellers belonging to the tellers group (TG), J1 is one janitor in janitors group (JG) and M1 is a manager belonging to accounts managers group (MG).

```
//declaration of users groups instances
one sig C1,C2,C3 in CG{}//3 customers
one sig T1,T2 in TG{}//2 tellers
one sig J1 in JG{}//1 janitor
one sig M1 in MG{}//1 account manager
```

Section 10 - Users Groups Instances

The keyword "**in**" of line 1 of section 10 means that C1, C2 and C3 are subsets of CG but they are not necessarily disjoints. To make them disjoints I can replace "in" by "extends". However, I used "in" to provide different usage of Alloy terms.

- Section 11 declares the owners groups containing users responsible of changing the list of CDIs/UDIs associated with a transaction TP. The following groups are defined as part of the implementation of CW principle of separation of duty where no single user can perform all tasks. Tasks must be divided among users:

- ODG: the group of owners who can change the CDI of the deposit money transaction (TD).
- OWG: the group of owners who can change the CDI of the withdraw money transaction (TW).
- OTG: the group of owners who can change the CDI of the transfer money transaction (TT).
- OOG: the group of owners who can change the CDI of the open new account transaction (IO).
- OCG: the group of owners who can change the CDI of the close account transaction (TC).
- OBG: the group of owners who can change the CDI of the check account balance transaction (TB).
- OFG: the group of owners who can change the UDI of the account gift balance transaction (TF).
- ORG: the group of owners who can change the UDI of the register daily customers' transaction (TR).

```
// declaration of groups owners
one sig ODG,OWG,OTG,OOG,OCG,OBG,OFG,ORG extends BankUsers{}
```

Section 11 - Users Groups Owners

- Section 12 determines the constrained data items sets as part of the BankCDIs. You can refer to Table 6 to find the description of CAS, LAS and PAS terms. Also, it defines different instances of CDIs available in the bank:

- CA1, CA2, CA3: 3 customers' accounts.
- LA1, LA2: 2 ledger accounts.
- PA1: 1 petty cash accounts.

```
//declaration of constrained data items sets
one sig CAS extends BankCDIs {} //Customers Accounts Set as part of BankCDIs
one sig LAS extends BankCDIs{} //Ledgers Accounts Set as part of BankCDIs
one sig PAS extends BankCDIs{} //Petty Cash Accounts Set as part of BankCDIs

//declaration of constrained data items instances
one sig CA1,CA2,CA3 in CAS{} //3 customers accounts
one sig LA1, LA2 in LAS{} // 2 ledger accounts
one sig PA1 in PAS{}// 1 petty cash account
```

Section 12 - CDIs sets and Instances

- Section 13 specifies the UDIs sets described in Table 4 and GB1, GB2 as instances of gift balances set (GBS) and CL1 as instance of daily customers logs set (CLS).

```
//declaration of unconstrained data items sets
one sig GBS extends BankUDIs {} //GiftBalances Set
one sig CLS extends BankUDIs{}//Customers Logs Set

//declaration of unconstrained data items instances
one sig GB1,GB2 in GBS {}
one sig CL1 in CLS{}
```

Section 13 - UDIs Sets and Instances

CW certification and enforcement rules can be enforced using "**fact**". The fact is a procedure that forces something to be true. "Fact" is convenient to implement system constraints based on a specific model. The following lines of code specify the constraints imposed by the CW model by placing them in the fact body in the form of negation. For simplicity I will show only sections of code demonstrating that the bank system follows the CW rules to enforce security.

- Section 14 states that customers' accounts set (CAS), ledger accounts set (LAS) and petty cash accounts set (PAS) cannot be accessed by register daily customer transaction (TR) and check customer gift balance (TF). In other words, CAS, LAS and PAS can be accessed by all transactions except TR and TF. Following the same manner, system constraints are defined respectively.

```
// constraints to apply CW model principles, certification and enforcement rules.
// Application of Certified and Allowed relations
//Follow table 5 regarding constraints
fact{

//Constraints on CDIs sets: CAS, LAS and PAS cannot be accessed by TR and TF (no constraints on UDIs sets)
CAS.accessedby!=TR
CAS.accessedby!=TF

LAS.accessedby!=TR
LAS.accessedby!=TF

PAS.accessedby!=TR
PAS.accessedby!=TF
```

Section 14 - System Constraints (example 1)

- Section 15 determines list of user groups not able to run deposit money (TD) and withdraw money (TW) transactions. In this example, deposit money can be run only by the tellers group or the owners deposit group. Similarly, TW can be run only by CG, TG and OWD.

```
//Transactions not run by certain groups
TD.runby!=CG
TD.runby!=MG
TD.runby!=JG
TD.runby!=OWG
TD.runby!=OTG
TD.runby!=OOG
TD.runby!=OCG
TD.runby!=OBG
TD.runby!=OFG
TD.runby!=ORG

TW.runby!=MG
TW.runby!=JG
TW.runby!=ODG
TW.runby!=OTG
TW.runby!=OOG
TW.runby!=OCG
TW.runby!=OBG
TW.runby!=OFG
TW.runby!=ORG
```

Section 15 - System Constraints (example 2)

Therefore, Sections 14 and 15 prove that the bank system respects the enforcement rule 1 (ER1) and 2 (ER2) of the CW model which state that the system must maintain the certified relations, and must ensure that only TPs certified to run on a CDI manipulate that CDI and the TP may access CDIs on behalf of the associated user and the TP cannot access that CDI on behalf of a user not associated with that TP and CDI. Regarding ER3, the system determines the current user group and authenticates him/her accordingly.

- Section 16 determines the group of users allowed to change the CDIs entities associated with the transaction. In this example only the ODG can perform changes and other mentioned group are not allowed to perform such change.

```
TD.changedby!=CG
TD.changedby!=TG
TD.changedby!=JG
TD.changedby!=MG

TD.changedby!=OWG
TD.changedby!=OTG
TD.changedby!=OOG
TD.changedby!=OCG
TD.changedby!=OBG
TD.changedby!=OFG
TD.changedby!=ORG
```

Section 16 – System Constraints (example 3)

This section proves that the system adopts the principle of separation of duty. For example, in the bank system, the customer group can run the withdraw money transaction (TW) but cannot change the CDIs associated with TW. OWD can run and change TW. Therefore, withdraw and change cannot be both handled by CG, they are handled separately by CG and OWD.

4.2. Clark Wilson model and Alloy analysis

After implementing the system using the Alloy language, it is time to analyze its consistency. Alloy analyzer helps checking system validity (1) by generation system Meta model and (2) by generating instances of the system based on its facts and predicates. Fig. 1 displays the bank system Meta model generated by the Alloy system. The Meta model maps the codes written in the previous sections into a graphical model. It shows that BankCDIs and BankUDIs are two subsets of BankDS. PAS, LAS, and CAS extends BankCDIs, GBS and CLS extends the BankUDIs. BankCDIs and BankUDIs are accessed by bank transaction BankTrs. However, the transactions TW, TT, TD, TR, TF, TC and TB are run by Bank Users whereas CDIs entities attached to transactions are changed by some BankUsers.

As you notice in the Fig. 5, the Meta model does not cover any constraints. It acts as system prototype. Testing system consistency is done by running the system predicates and generating possible instances then validating them. In order to test the constraints specified in the fact procedure, I need to write a predicate then run it.

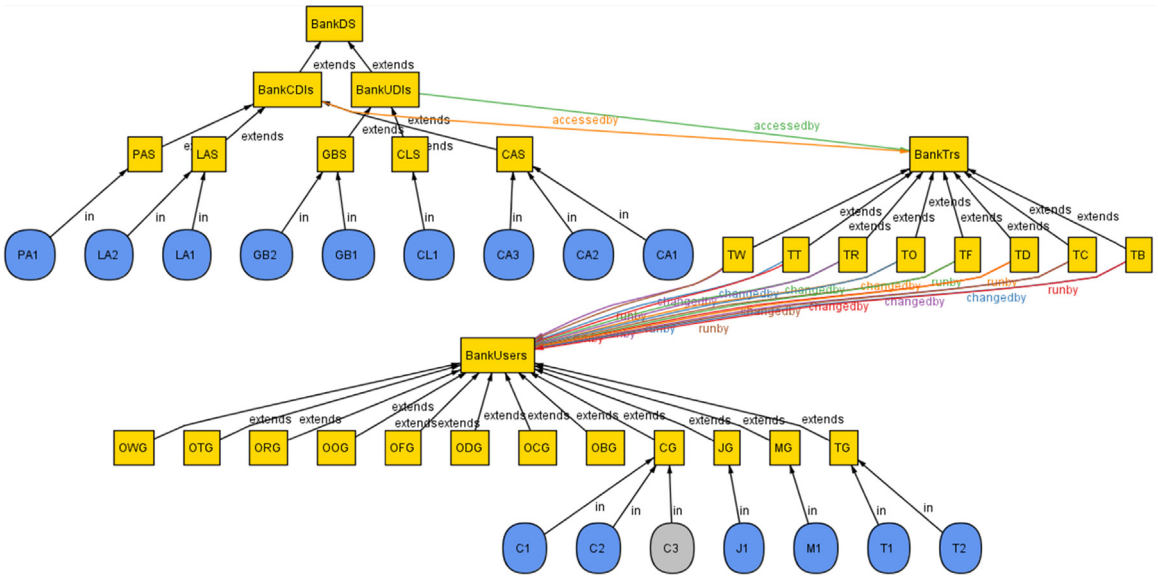

Fig. 5. Bank system meta model.

```
Executing "Run example"
  Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
  708 vars. 248 primary vars. 874 clauses. 125ms.
  Instance found. Predicate is consistent. 62ms.
```

Fig. 6. Alloy analyzer output.

Table 9
Relationships checking (1).

Set/relation	Type	Accessed by	Consistent
GBS	UDI	TF	Yes
CAS	CDI	TT	Yes
CLS	UDI	TR	Yes
PAS	CDI	TW	Yes
LAS	CDI	TW	Yes

- Section 17 declares an empty predicate used to test the system consistency based on the defined facts. Executing the example() will produce the output specified in Fig. 5. The Figure shows that an "instance found" and "Predicate is consistent". It takes around 62 ms to determine consistency and finds an instance (Fig. 6).

```
// run example to show consistencies/ inconsistencies
pred example(){
}
run example
```

Section 17 - Predicate

Clicking the link "**Instance**" will show Fig. 7. More instances can be generated by pressing the "Next" button located at the top of the screen of Fig. 7. The generated instances demonstrate the consistency of the system as shown in Tables 9 and 10.

However, specifying a wrong predicate such as stating that open account transaction (TO) can be run by customers groups (section 18, line 2) will cause inconsistency and the result of running the example() is displayed in Fig. 8.

Table 10
Relationships checking (2).

Trs/relation	Run by	Changed by	Consistent
TB	OBG	OBG	Yes
TC	OCG	OCG	Yes
TD	ODG	ODG	Yes
TO	OOG	OOG	Yes
TF	TG	OFG	Yes
TT	OTG	OTG	Yes
TR	JG	ORG	Yes
TW	TG	OWG	Yes

```
// run example to show consistencies/ inconsistencies
pred example(){
TO.runby=CG
}
run example
```

Section 18 - Inconsistent Predicate

Thus, the bank system adopting the Clark Wilson integrity model is a consistent system. Any misbehavior will results in an inconsistency where no instances are found.

5. Mixed model: RBAC and Clark Wilson

Before we integrate RBAC and Clark Wilson policies, one needs to define the entities and restrictions of the mixed model. The role is one of the entities which defines the set of users; this means that the role defines which user belongs to which group. The other entity is permission, which means that each role is given authorization to read, change or share some data.

To determine whether RBAC and Clark Wilson are compatible and consistent when mixed together, we produced a mixed model. The mixed model gives the users certain roles and then gives access to read, change or share the data according to the level of each user.

After showing that the mixed model is consistent, we need to test for any inconsistencies by specifying a wrong predicate. We run the test predicate again giving an inconsistent data. This showed that no instance is found as shown in Fig. 9, and therefore the mixed models worked harmoniously.

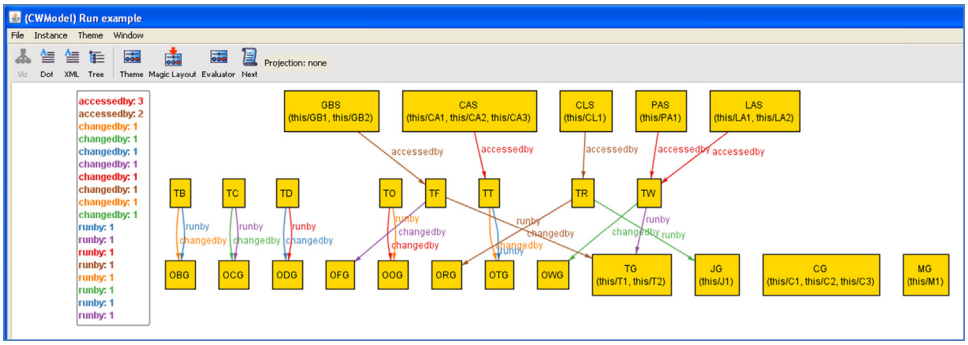

Fig. 7. Bank system instance.

```
Executing "Run test"
  Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
  0 vars. 0 primary vars. 0 clauses. 31ms.
  No instance found. Predicate may be inconsistent. 0ms.
```

Fig. 9. Mixed model inconsistency output using Alloy.

```
Executing "Run example"
  Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
  0 vars. 0 primary vars. 0 clauses. 16ms.
  No instance found. Predicate may be inconsistent. 0ms.
```

Fig. 8. Alloy analyzer output.

6. Conclusion

In this paper, we presented the role-based access control and the Clark Wilson Models. We used system examples based on the defined security models. We formalized the models then checked their consistency. Since Alloy allows expressing systems as set of logical constraints in a logical language based on standard first-order logic, we used it to define the system and its policy. However, when creating the models we specified the system users and subjects then Alloy compiles a Boolean matrix for the constraints, and we asked it to check if the models are valid, or if there are counterexamples. As for future work, we believe that more work can be done to integrate multiple models in a mixed mode, and formalize them to find potential interactions.

References

[1] A. Shaffer, M. Auguston, C. Irvine, T. Levin, A security domain model to assess software for exploitable covert channels, in: Proceedings of the ACM SIGPLAN Third Workshop on Programming Languages and Analysis for Security, Tucson, AZ, USA, 2008, pp. 45–56.

[2] B. Panda, W. Perrizo, R.A. Haraty, Secure transaction management and query processing in multilevel secure database systems, in: Proceedings of the ACM Symposium on Applied Computing, Phoenix, AZ, April 1994.

[3] C. Gonzalez, V. Dutt, C. Lebiere, Validating instance-based learning mechanisms outside of ACT-R, Journal of Computational Science (4) (2013) 262–268.

[4] C. Summers, Computer Security: Threats and Safeguards, McGraw Hill, New York, 1997.

[5] D. Brewer, M. Nash, The Chinese Wall security policy, in: Proceedings of the IEEE Symposium on Research in Security and Privacy, Oakland, CA, USA, 1989, pp. 206–214.

[6] D. Jackson, Alloy 3.0 reference manual, 2014, Retrieved on January 24 from: http://alloy.mit.edu/reference-manual.pdf

[7] D.F. Ferraiolo, D.R. Kuhn, Role-based access control, in: Proceedings of the 15th National Computer Security Conference, Baltimore, 1992.

[8] J. Misic, V. Misic, Implementation of security policy for clinical information systems over wireless sensor networks, Ad Hoc Networks Journal 5 (2006) 134–144.

[9] J. Zao, W. Hoetech, J. Chu, D. Jackson, RBAC schema verification using lightweight formal model and constraint analysis, in: Proceedings of 8th ACM Symposium on Access Control Models and Technologies, Boston, MA, USA, 2003.

[10] K.J. Biba, Integrity considerations for secure computer systems. Technical Report MTR-3153, The MITRE Corporation, 1977.

[11] L. Bell, E. LaPadula, The MITRE Corporation, 1976.

[12] M. Bishop, Computer Security: Art and Science, Addison Wesley, Boston, 2003.

[13] M. Felis, K. Mombaur, H. Kadone, A. Berthoz, Modeling and identification of emotional aspects of locomotion, Journal of Computational Science (4) (2013) 255–261.

[14] R.A. Haraty, C2 secure database management systems – a comparative study, in: Proceedings of the Symposium on Applied Computing, San Antonio, TX, USA, 1999, pp. 216–220.

[15] R.A. Haraty, M. Naous, Modeling and validating the clinical information systems policy using Alloy, in: Proceedings of the Second International Conference on Health Information Science. Lecture Notes in Computer Science, Springer, London, England, March 2013.

[16] R.A. Haraty, N. Boss, M. Naous, Modeling and Validating Confidentiality, Integrity, and Object Oriented Policies Using Alloy, Security and Privacy Preserving in Social Networks, Springer, 2013, ISBN 978-3-7091-0893-2.

[17] R. Anderson, A security policy model for clinical information systems, in: Proceedings of the IEEE Symposium and Security and Privacy, USA, 1996.

[18] R. Seater, G. Dennis, Tutorial for Alloy Analyzer 4.0, 2014, Retrieved on January 24 from: http://alloy.mit.edu/tutorial4

[19] S.B. Lipner, Non-discretionary controls for commercial applications, in: Proceedings of the IEEE Symposium on Security & Privacy, Oakland, 1982.

[20] W. Hassan, L. Logrippo, Detecting inconsistencies of mixed secrecy models and business policies. Technical Report, University of Ottawa, Canada, 2009.

[21] W. Hassan, L. Logrippo, M. Mankai, Validating access control policies with Alloy, in: Proceedings of the Workshop on Practice and Theory of Access Control Technologies, Quebec, Canada, 2005, pp. 17–22.

Ramzi A. Haraty is an associate professor of Computer Science in the Department of Computer Science and Mathematics at the Lebanese American University in Beirut, Lebanon. He serves as the program administrator for the Middle East Program Initiative's (MEPI) Leaders for Democracy Fellowship program. He is also the internship coordinator for MEPI's Tomorrow's Leader program. He received his B.S. and M.S. degrees in Computer Science from Minnesota State University – Mankato, Minnesota, and his Ph.D. in Computer Science from North Dakota State University – Fargo, North Dakota. His research interests include database management systems, artificial intelligence, and multilevel secure systems engineering. He has well over 110 books, book chapters, journal and conference paper publications. He supervised over 110 dissertations, theses and capstone projects. He is a member of the Association of Computing Machinery, Institute of Electronics, Information and Communication Engineers, and the International Society for Computers and Their Applications.

Mirna Naous is a senior solutions developer at Dar el Handasah–Lebanon. She received her B.S. and M.S. degrees in Computer Science from the Lebanese American University – Beirut, Lebanon. Her research interests include computer security.