國立交通大學

資訊科學與工程研究所

碩 士 論 文

使用 Alloy 建模驗證系統於存取控制

Access Control Schema Verification Using Alloy SAT Model

研 究 生：陳奕興

指導教授：邵家健　教授

中 華 民 國　九 十 五　年 十 月

使用 Alloy 建模驗證系統於存取控制
Access Control Schema Verification Using Alloy SAT Model Checker

研　究　生：陳奕興　　　　Student：Yi-Hsing Chen

指導教授：邵家健　　　　Advisor：John Zao

國　立　交　通　大　學

資　訊　科　學　與　工　程　研　究　所

碩　士　論　文

A Thesis

Submitted to Institute of Computer Science and Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

Octorber 2006

Hsinchu, Taiwan, Republic of China

中華民國九十五年十月

摘要

在這篇論文中，我們在 Alloy 建模驗證系統中驗證存取控制機制是否滿足線性代數上的某些特質為首要目標。在此論文中，以三個典型的存取控制機制為驗證目標，這三個存取控制分別為(1)Bell-LaPadula Model，驗證以此模型的機制下，安全標籤是否符合代數中的大小關係，如:a>b>c。且安全層級越高者可讀檔案越多。(2) China Wall Security Policy (中國城牆策略)，傳統的 China Wall Security Policy 規定過為嚴格，於此我們將其規定放寬，使得一家公司的顧問可將此公司的資訊流出，但不得讓相同性質公司獲得此公司的資訊，我們以此驗證我們所提出的策略是否符合代數性質中的遞移關係且不會讓資料外流至不開取得此資料之公司。(3)角色型存取控制，我們加上角色責任分離機制，以驗證任一使用者不得獲得兩個以上有衝突的角色。最後再結論時討論我們處理三種存取控制與驗證結果，以及簡略討論 Alloy 建模驗證系統的效能。

Abstract

Throughout the thesis, our main goal is to verify access control schema to see if they satisfy certain algebraic properties. In the thesis, we exercise verifications on three access control policies. The first one is the Bell-LaPadula Model. By exercising Bell-LaPadula Model in Alloy, we verify the order relations between the security label and the user who possesses higher security level can read more files. The second one is the China Wall Security Policy. The Brew and Nash model for China Wall Security Policy is too restrictive to practice. We loosen the policy to be less restrictive. One company's information can be written into the other company, as long as there two companies do not belong to the same conflict of interest. We exercise this model and to verify if there's any possible information goes from one to another and the two companies are in the same conflict of interest class. (Belongs to transitive relation). The last one is the Role-Based Access Control with Separation of Duty concept. In the conclusion, we conclude the result of the verification and briefly discuss the effectiveness of the Alloy SAT model checker.

誌謝

感謝我的指導教授邵家健老師兩年辛勤的指導，邵家健老師不止在學業方面教導我許多，從老師身上也學到一些生活上的啟發。他不只是我的老師也是長輩也像個朋友。感謝老師讓我在預官二梯以前口試趕上當兵，再次感謝老師。再來我要感謝的是三位口試老師：王柏堯老師，陳穎平老師與彭文志老師，能在緊急時刻配合學生的口試時間，出席學生口試答辯。

再來謝謝實驗室的夥伴們，大家的聚集使的實驗室歡樂不斷，是在壓力極大時候最佳的紓解壓力方式，其中的黃為霖學長、張哲為學長與黃國晉學長的鼓勵與支持，曾輔國學弟的協助，都是感動在心裡。在次感謝大家。

最後我想說: 老媽!我畢業了! 感謝我的家人在精神上的支持與鼓勵。交大我愛你!

Contents

List of Figures

List of Tables

Chapter 1: Introduction

1.1 Project Objective

Access control is the basis of computer security. What is an access control? Access control is a mechanism to determine the usage (such as, read, write) of target object (which is a passive entity, such as files) by the subject (which is an active entity, such as users). Access control is the basic security policy and there are certain amounts of application based on this concept. From traditional access control, such as, Discretionary Control Access Control (DAC), Mandatory Access Control (MAC), Latticed-Based Access Control (LBAC), such as Bell-LaPadula Model [3], Role-Based Access Control (RBAC)[7] and Usage Control Policies, such as, UCON$_{abc}$ [9].

Access Control is no longer been applied only in computer security field. Access control has been used in other fields, such as business world, healthcare system, e-home system etc ….Like the China Wall Security Policy [4] for the business world as to Bell-LaPadula Model [3] for the military.

Every access control policy has its own principals. Are every proposed access control policies accurate? How do we prove that the proposed access control policy has no lack of consideration? Scientist and researchers use to prove the policies either by mathematical aspect or by model checker aspect.

What is the objective of the project? We are going to use Alloy SAT[10] model checker[1,2] to exercise three cases which are Bell-LaPadula Model[3] , China Wall Security Policy[4] and the Role-Based Access Control[7] in order to achieve to goals of two level. First level is the general case which is to verify access control schema [8] on the satisfaction of certain algebraic properties. For example, in Bell-LaPadula Model [3] the four labels with the security order, “Top Secrete (TS)” is higher than “Secrete (S)”, “Secrete (S)” is higher than “Secrete but Unclassified (SBU)” and “Secrete but Unclassified (SBU)” is higher than “Unclassified (U)”. The security order contains the algebraic property which is TS>S>SBU>U.

The second level is the specific case which is to find instance that satisfy a specific scheme. For example, we input a set of element with assigned values and to find instances that satisfy our scheme. From the two goals, we hope to achieve the debugging of the control schema.

1.2 Project Approach

Our research approaches are the following steps:

1. From the first access control scheme" Bell-LaPadula model"[3], the four security labels are defined to have the orders among the. The order relations between each of the two labels lead to a algebraic property which is TS>S>SBU>U. And we apply the partial order relation to achieve the property and follow the read-down and write-up rule to see if the Bell-LaPadual model does fulfill this property.
2. The traditional China Wall Security Policy by Brew and Nash [4] is too restrictive. We loosen its write-rule to make it more adequate to the system. We call the less restrictive China Wall Security Policy as China Wall Security Policy Extension. To let company information able to flow to the others. The information flow involves in the checking of transitive relations. In the second case, we focus on verifying if the extension schema satisfies the transitive relations.
3. In role-based access control, we model the essential components of RBAC structure and consider the concept of Separation of Duty (SoD). By considering the SoD in our modeling structure. The SoD involves two ideas which are inheritance and conflict roles. We focus on verifying if the RBAC with SoD will cause two roles with conflict to be assigned to the same user. And later to model Bell-LaPadula Model with RBAC concept
4. In the three access control schema, we try to find the redundant statement in each code and eliminate it, in order to minimize the size of the state bits of verification process. We program the above three cases on the basis of the paper "RBAC Schema Verification Using Lightweight Formal Model and Constraint Analysis" [8] by Prof. John Zao, Daniel Jackson and their research team.
5. We briefly analyze the Alloy model checker by the running result of the above three schema.

1.3 Chapter Overview

Chapter 1 is the introduction, and we briefly introduce Alloy model checker. In chapter 3, we proceed to the first access control model Bell-LaPadula Model and chapter 4 is the China Wall Security Policy Extension and transitive closure verification .Chapter 5 is the Role-Based Access Control schema and its verification and with RBAC applying on Bell-LaPadula Model .The chapter 6 is the conclusion.

Chapter 2: Introduction to Alloy

In this chapter, we briefly introduce the modeling language named Alloy that we use throughout the thesis.

2.1 What is Alloy?

Alloy is a modeling tool which is developed by MIT Professor Daniel Jackson and his research group. The latest version of Alloy is Alloy 3.0-bata. Alloy can model data structure, such as abstract data type which is the basis of OO programming. Alloy model checker has two properties (1) Satisfiability (SAT) [10] check. (2) 1st order predicate logic

2.2 Basic Syntax of Alloy

2.2.1 File Name and Path

Every alloy file is name as file_name.als .In each file, the first part of the program begins with the path of the file, for example,

module models/research/alloy/file_name If user needs to include other file (maybe utility file, test file, marco library etc …), he can use the " open " file function , for example: ***open* util/boolean** .Once we open the file, we can apply all the declared sets, functions which are included in the file(in the example is boolean.als and bolean.als is located under the folder util)

2.2.2: Sets, Subsets and Relation

Set is the basic component of the program, and it is declared as:

sig* Human{}**, ***sig represents signature which tells the alloy that "Human" is a set.

Users can use the key word " ***in* "** or "***extends***" to declare some set(s) is(are) a subset of the other, for example: ***sig* Men *in* Human{}** , which means book is a subset of Object .The difference between ***in*** and ***extends*** is that ***extends*** partitions the sets and all the subset extends the set are disjoint, for example,

***sig* Men *extends* Human{}**

***sig* Women *extends* Human{} :**

It means Men and Women partition the set Human and the subset Men and Women are disjoint and their union is the set Human. But if we write

***sig* Men *in* Human{}**

***sig* Women *in* Human{}**

, it doesn't mean Men and Women partition Human and their union is the set Human. In alloy, everything is a relation, such as unary, binary and ternary relation. In the braces, users can define relation .For example:

sig **Human{ eat : some Food , beLoved : set Human }**

eat is a relation mapping from Human to Food which means "Every human being eats food." and beloved is a relation maps Human to Human that means "Every Human being has zero or more human being like this Human". Here we apply multiplicity key words ***some*** (one or more) and ***set*** (zero or more) .We will introduce multiplicity markings in the next section.

2.2.3: Multiplicity

In Alloy, the following are the multiplicity key words.

lone:	less than or equal to one
one:	exactly one
some:	one or more
set:	zero or more
all:	all

If a user what to declare there's only one element in the set, he can write "***one*** **sig Book {}**". From the previous sector, if we rewrite "***sig*** **Book** ***extends*** **Object{} ,** ***sig*** **Pen** ***extends*** **Object{}**" into" ***one sig*** **Book** ***extends*** **Object{}** ***one sig*** **Pen** ***extends*** **Object{}**", it means the two subset Book and Pen both have only one element in it and the set Object only has two elements. Multiplicity marking can be used in set declaration, fact, predicate, function, assertion. We will introduce fact, predicate, function, assertion in the following section.

2.2.4: Fact

Facts are the constraint statements that always hold through out the simulation and verification process. Every model has its own specification and according to the specification, we apply the specification into the ***fact*** for Alloy to generate all cases under the constrain statements of ***fact***. According to the listed-***fact*** statement, Alloy uses it for simulation and verification. For example,

fact **{ some h: Human | h.beLoved = none}:**

It means there are some human being(s) that no one loves.

2.2.5 Predicate and Function

(a)Predicate

If all the inputs satisfy the listed constraints, then the predicate will be be true.

pred **loveTest (m:Men , w:Women){ w in m.beLoved},**

It means giving two input argument m which belongs to Man and w which belongs to Women. If the w loves the m, the predicate returns true otherwise returns false. Predicate has another function. When it comes to simulation, the listed statement in the brace will be combined (conjoint) with all the ***fact*** listed in the program and generate one of the instances satisfies all the constraints statement.

(b) Function

Function can be used to simulate like the Predicate does and it has another function which can be used to return a set of element satisfying the listed statement in the brace (which is it body). For example,

fun **loveSet (m:Men , W : set Women) : set Women{ m.beLoved & W}:**

It which means giving a m belongs to Men and a set of Women and according to the input set W, function loveSet returns the set of women Who love the m. Here **"&"** means intersection.

2.2.6 Run, Assert and Check

After a user declare all sets, relations, facts, functions and predicates, he can use the command "***run***" to simulate one of cases which meets constraints. For example,

run **loveSet for 10 Human:**

It will generate one of the instances under the scope of 5. Users only specify the scope for top level signature. What is top level signature? It's the set that doesn't extend other sets. Here in our example, Human is the top level signature .Since Men and Women extends Human, these two are not top level signature. The command ***"run"*** is used for simulation, and the command ***"check"*** is used for verification. When a user makes an assertion, he would like to know if the assertion stands or not. That is the command ***"check"*** used for. For example,

assert **famousPeople {some m: Men | m.beLoved = Women }**

check **famousPeople for 10:**

We make an assertion "famousPeople" that in every case, there's at least one man that every women loves. We check the assertion under the scope of 10.

2.2.7 Further References

If reads are interested in understanding more about alloy, please refer to the main page : http://alloy.mit.edu [1] it's reference manual [2] for further acknowledgement .

Chapter 3: Bell-LaPadula Model

In this chapter, we introduce Bell-LaPadla Model and use exercise the model on Alloy for two levels of goals. The first level is to verify the model if it fulfills certain algebraic property and the second level is to find an instance that satisfies the specific scheme. We will use these two level goals throughout the thesis.

3.1 Bell-LaPadula Model (BLP)

The Bell-LaPadula Model was developed by David Elliott Bell and Len LaPadula [1] in 1973 to formalize the U.S. Department of Defense multilevel security policy. The model is a formal model of computer security policy that describes a set of access control rules by the use of security labels which Bell-Lapadula named it as classification on objects, from the most sensitive to the least sensitive, and clearances for subjects. A set of security labels are, from most to least sensitive, “Top Secret”, “Secret”, “Sensitive but Unclassified” and” Unclassified “.

3.2 Access Rule

The access control rules are read-down and write-up rules.

3.2.1 Read-Down Rule

The user with certain clearance have read right to documents with security level lower than or equal to the user’s (*1) clearance. Figure 1 shows the relation that some user with certain clearance and the document(s) that he has read right.

For example, A user named John with its clearance to be Secrete, and there are two document which are document1 (with classification: Top Secrete), document 2 (with classification: Secrete but Unclassified).

Since the user’s clearance is higher than document2, he has the right to access document2 but not document1 due to his clearance (Secret) is in lower security level than document1 (Top Secret). That is how we call it the read-down rule.

---

*1. We simply subject to be user. We don’t distinguish user, subject, principals for simplicity reason.

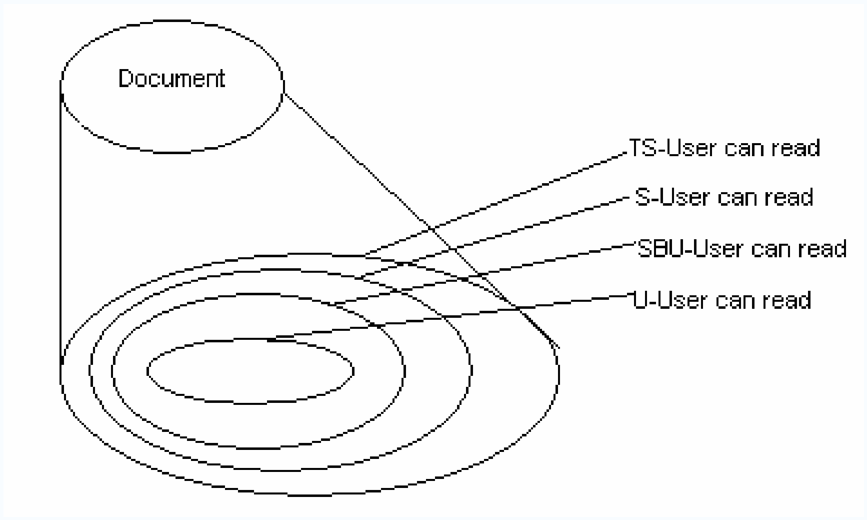


Fig. 1 User Read Right on Document

3.2.2 Write-up rule

The write-up rule is opposite to the read-down rule. A user has the right to write information into the document only when then use's clearance is lower than or equal to the document's classification, he has the right to write information into this document. From the example in section 3.2.1, John has right to write information into document1, because John's clearance (Security) is lower than document1's classification (Top Security). Figure 2 shows the read-down and write-up rules and relations between the security labels.

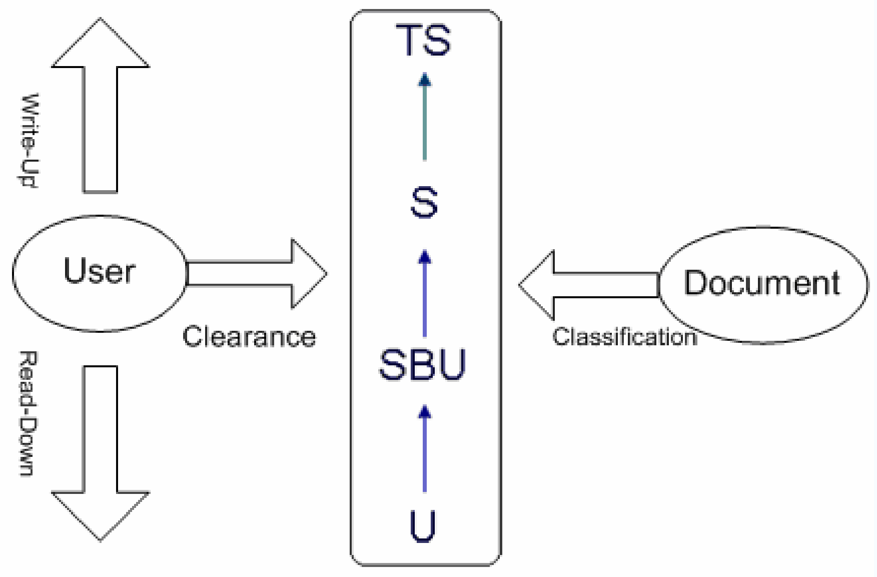


Fig. 2 Security Label Relation

3.2.3 Why Write-Up Rule?

If the write rule is modified to be write-down rule: when a user’s clearance is higher or equal to the document’s classification, he can write information to the document. Here is a possible problem. Let us explain with example. There are two user, say Johan and Mary. John’s clearance is TS, and Mary’s clearance is SBU. There are two documents, say doc1 and doc2. The classification of doc1 is S and the classification of doc2 is SBU. John has read right to both doc1 and doc2. Now John reads doc1 information and write the information to doc2 (because here we say write rule becomes write down rule). In doc2, there’s top secrete information written into it. Then if Mary reads doc2, she can obtain information from doc1. The “write-down” rule would lead to possible information leakage to someone who has lower security label and doesn’t suppose to obtain that information. That is why the write rule is the write up rule.

3.3 Model checking on Bell-LaPadula Model

In this section, we exercise the model checking on Bell-Lapadula Model and to see if we achieve the goals we mentioned in introduction. Since this model has been verified using mathematical way or using model checking. We still do the checking again in alloy for later usage in Role-Based Access Control chapter. We address main part of the program. If reader are interested in reading the whole program , please refer to the Appendix A

3.3.1 Partial Order Relation On security Label

How do we let the TS rank higher than S in security level? The answer is the partial order relation.

3.3.1.1 Partial Order Relation

For a set, say S, and a relation, say α, if α satisfies (a) **reflexive** (b) **transitive** (c)**anti-symmetric** relation , then R is the partial order relation.

(a)**reflexive** $\forall s : S$, *“s α s” then relation α is reflexive*

(b)**transitive:** $\forall s1, s2 : S$ *,” s1 α s2 ⇒ s2α “s1 then α is transitive*

(c) **anti-symmetric:** *∀ s1, s2: S ,”(s1α s2)& (s2α s1) ⇒ s1=s2”*

3.3.1.2 Relations on Security Label

In Bell-LaPadula model, security label TS is higher than S, S is higher than SBU is higher than U. We transfer these relations to be algebraic expression: TS >S >SUB>U.

The following part of the program in Fig. 3 is to define relations between security labels and to define partial order relation in order to achieve "TS>S>SBU>U"

```
fact partialOrderRelation
{
  /*reflexive relation*/
  all la: Label| la in la.geq
  /*transitive relation*/
  all la1 , la2 :Label| la1 in la2.*geq
                => la1 in la2.geq
  /*anti-symmetric relation */
   all la1 , la2:Label |
```

```
fact securityLevel
{
   TS in S.geq
   S in SBU.geq
   SBU in U.geq
}
```

Fig. 3 Parial Order Relation

From Fig.3, alloy generates the relational graph (Fig 4) among these 4 security label.

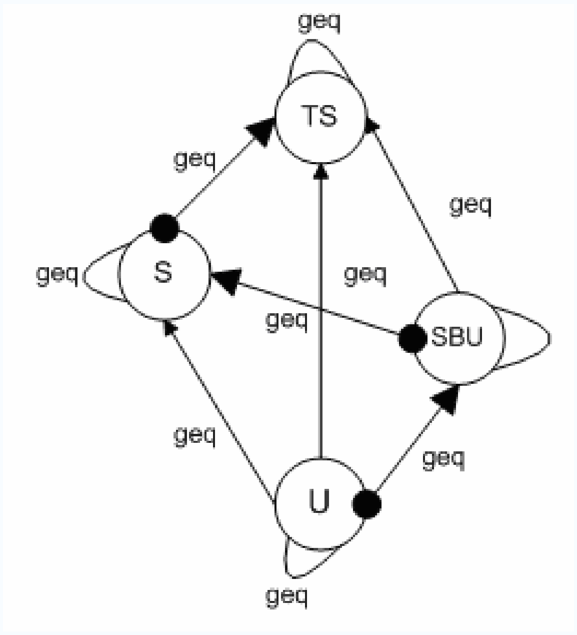
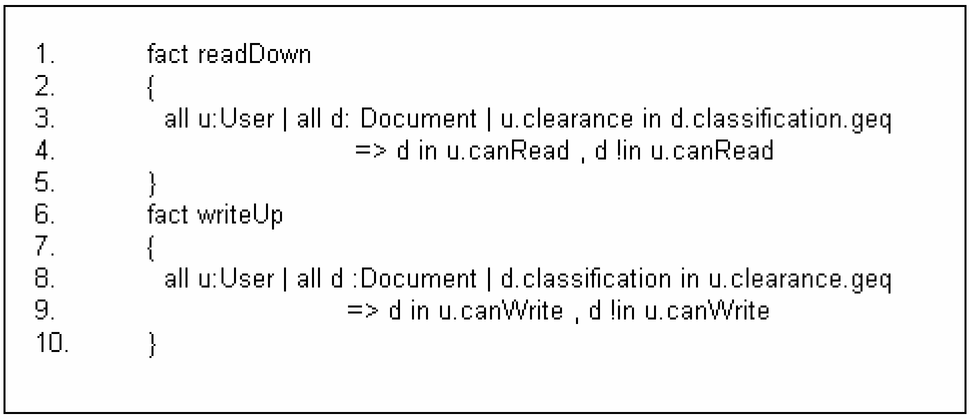


Fig. 4 Graph of Partial Order Relation

The arrows with the black dot of Fig.4 are from because of the fact statement that we declare in Fig.3 The arrows without black dot are derived from the transitive relation. The curve lines are derived from the reflexive relation.

3.3.2 User's access right on documents



```
1.        fact readDown
2.        {
3.          all u:User | all d: Document | u.clearance in d.classification.geq
4.                                 => d in u.canRead , d !in u.canRead
5.        }
6.        fact writeUp
7.        {
8.          all u:User | all d :Document | d.classification in u.clearance.geq
9.                                => d in u.canWrite , d !in u.canWrite
10.       }
```

Fig. 5 Read-down and Writ-up rule

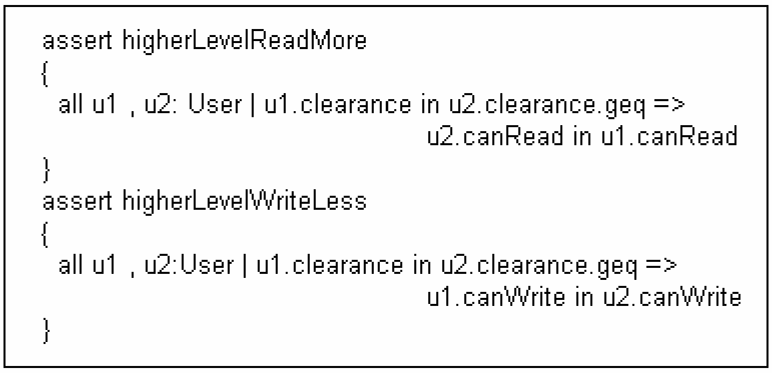
In Fig.5

line 3-4: if a user’s clearance is greater than or equal to the document’s classification he has the right to read the document .This is based on the read-down rule in Bell-LaPadula model.

line 8-9: if a user’s clearance is less than or equal to the document’s classification, he has the right to write the document. This is based on the write-up rule in Bell-LaPadula model.

3.3.3 Verification on the model

In Fig.6 we do the final part of the modeling language which is verification.



```
assert higherLevelReadMore
{
  all u1 , u2: User | u1.clearance in u2.clearance.geq =>
                                        u2.canRead in u1.canRead
}
assert higherLevelWriteLess
{
  all u1 , u2:User | u1.clearance in u2.clearance.geq =>
                                        u1.canWrite in u2.canWrite
}
```

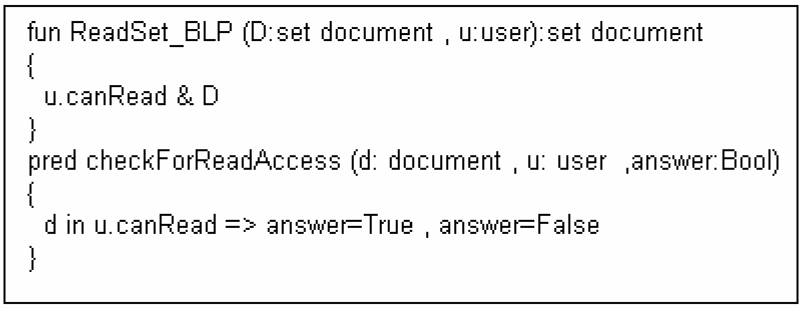
Fig. 6 Assertion for Checking (BLP)

Line 3: This line is used to assert that one user has higher security level status than the other, he has right to read more documents than the other has.

Line 4: This line is used to assert that one user has lower security level status than the other, he has right to write more documents than the other has.

3.3.4 Additional Function Offering

Except for verification on the model, we provide additional functions for user to use. Some user may just want to know that given a set of documents and a user, which documents the user have access right on. Some user may just want to know some particular case but has no intention to figure out all the checking process. This is one of the reasons why we provide the additional function. This helps the user to obtain the answer easily .The additional functions is an important feature throughout the thesis. In Fig.7 shows two of the functions that we provide for users.



```
fun ReadSet_BLP (D:set document , u:user):set document
{
  u.canRead & D
}
pred checkForReadAccess (d: document , u: user  ,answer:Bool)
{
  d in u.canRead => answer=True , answer=False
}
```

Fig. 7 Two Additional Functionls (BLP)

Line 1-4: Given a set of documents and a user as input argument, return a set of document according to the read rule of Bell-LaPadula model. Here we use the expression in line 3 which means the intersection of u.canRead and D and to return result in graph or in text format.

Line 5-8: Give a document and a user as input to check if the user has read right on the documents. If the user can read the document, the predicate output the answer to be true, otherwise false.

The use of additional functions helps users to understand how to use them easily and obtain the test result according to the input arguments that the users give.

3.5 Instance Satisfies the Specific Scheme

The following figure shows one of the simulations case.

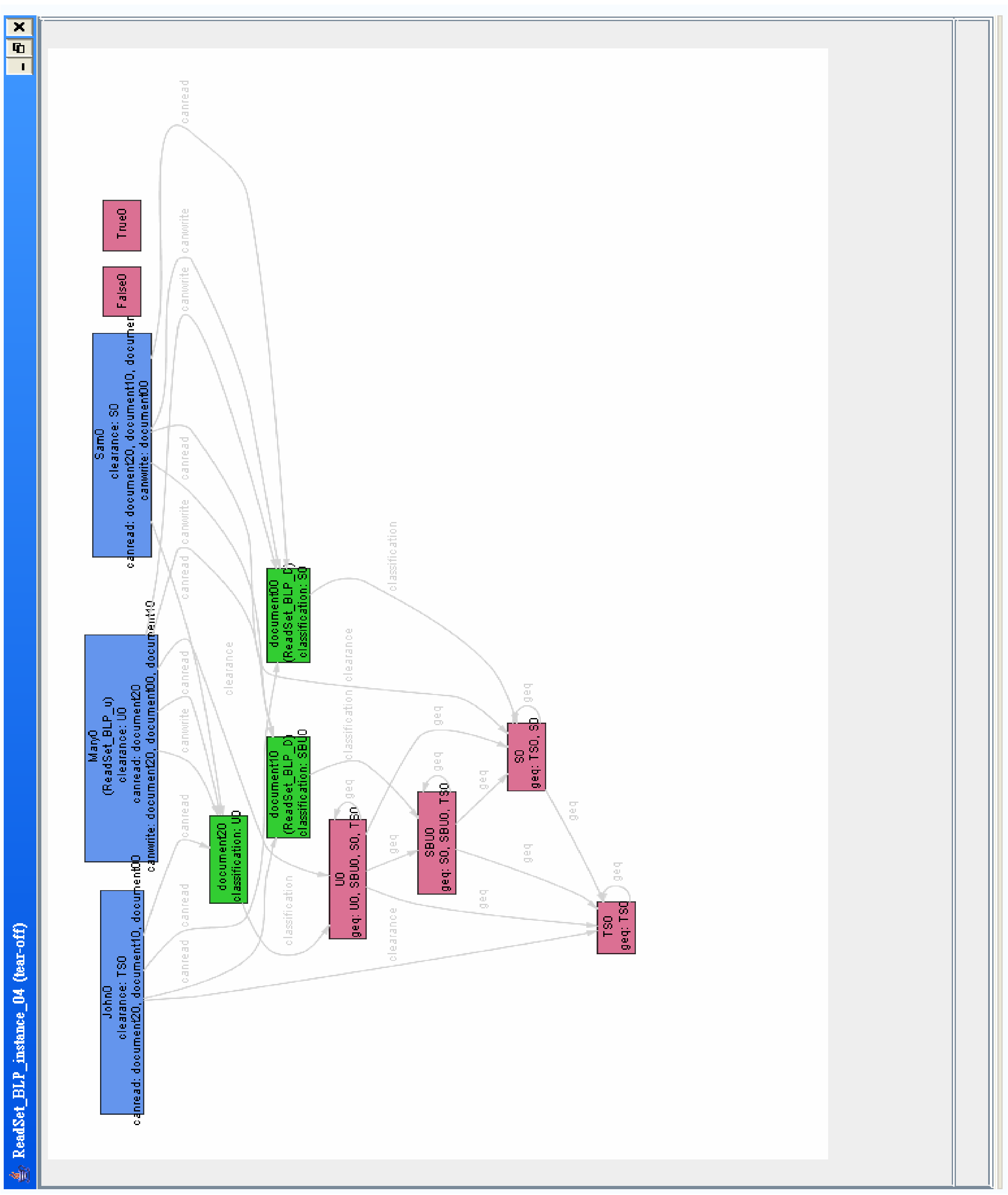


Fig. 8 Instance (BLP)

Chapter 4 China Wall Security Policy Extension

After introducing Bell-LaPadula model in chapter 3, we proceed to another classical access control policy “China Wall policy”. In this chapter, we first introduce the original China wall policy and then we modify its policy to be less restrictive.

4.1 China Wall Security Policy

Securities on business world were hardly discussed and were less relevant to the computer scientist before 1980’s. The issues in business world security were started to attract computer scientist attention. Due to the need for security policies in business world, researchers and computer scientist began to propose security policies for the business world and the China Wall Security Policy [4] was one of the emerging security policies in 1980’s. China wall policy [4] is proposed by David F.C. Brewer and Michael J. Nash.

4.1.1 Conflict of Interest Class (COI)

In the China Wall policy, each company belongs to one conflict of interest class. The companies which have conflict of interest in business world will be categorized to the same conflict of interest class. Fig.9 [4] shows the hierarchical structure among the conflict of interest class and company data set.

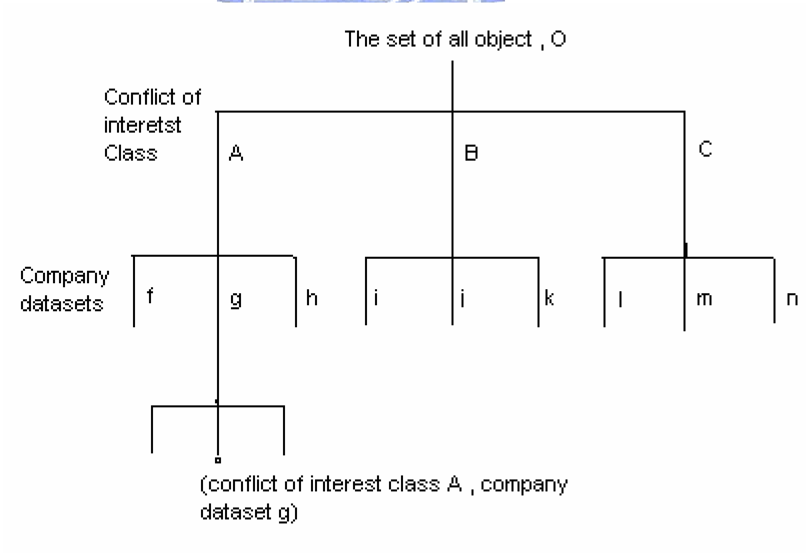

The set of all object , O
Conflict of interetst Class
A
B
C
Company datasets
f
g
h
i
j
k
l
m
n
(conflict of interest class A , company dataset g)

Fig. 9 The Composition of Objects

For example, BankA and BankB have conflict in interest and these two are categorized to into the same conflict of interest class. When it comes to BankA and OilB, these two will not be in the same conflict of interest class.

4.1.2 Access Rule

The Brewer-Nash model [4] for the china wall policy does not distinguish users, principals and subjects. It uses the single concept of subject for all three notions.

(A) BN-Read rule : Subject S can read object O only if

(a) O is in the same company dataset as some object previously read by S (i.e., O is within the wall) , ***or***

(b) O belongs to a COI class within which S has not read any object (i.e., O is outside the wall).

(B) BN-Write rule: Subject S can write object O only if

(a) S can read O by BN read rule , ***and***

(b) No object can be read which is in a different company data set to the one for which that write access is requested.

4.1.3 Drawbacks

In the Lattice-Based Enforcement of Chinese Walls [5], it says “the BN read and write rule is successful in preventing such information leakage. However, it does so at an unacceptable cost. And it is easy to see that the access rule has the following implication.

(A) A subject which has read objects from two or more company datasets cannot write at all

(B) A subject which has read object from exactly one company dataset can write to that dataset.“

From the discussion on the paper [5], the original China wall security policy is too restrictive to really apply in the real world situation a. In the next section, we introduce the modifications on China wall policy to make it less restrictive.

4.2 China Wall Security Policy Extension

4.2.1 Modification of BN-Write Rule

We modify the BN-Write rule to be

(A) A consultant(*2) can read one company’s information and write to another company as long as the two company belongs to different conflict of interest class , ***and***

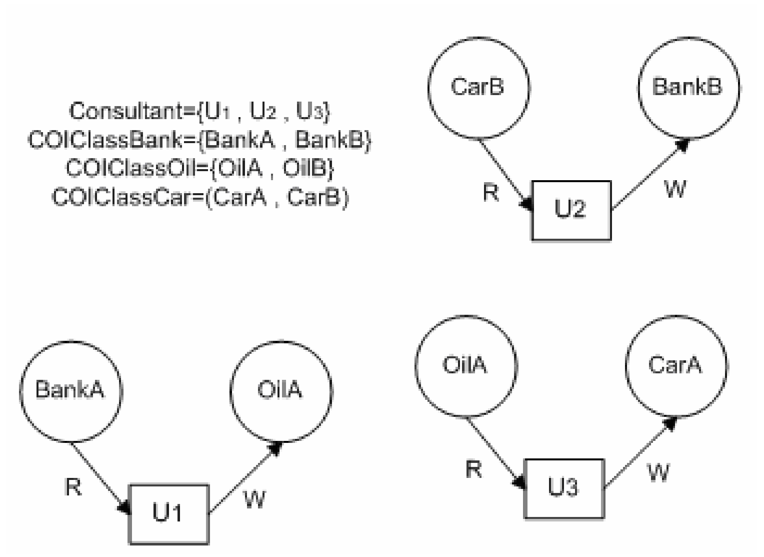
(B) For any two different companies which belong to the same conflict of interest class, there couldn’t be any information flow that flow from one to the other.

In Fig.10, it shows the existing information flow satisfy our modification rule.

In Fig. 10, there are three relations: BankA-> OilA(by U_1) , OilA->CarA(by U_3) , CarB-> BankB(by U_2) . OilA has BankA’s information, meanwhile CarA has OilA’s information, but CarA’s information wasn’t being written into any companies.

---

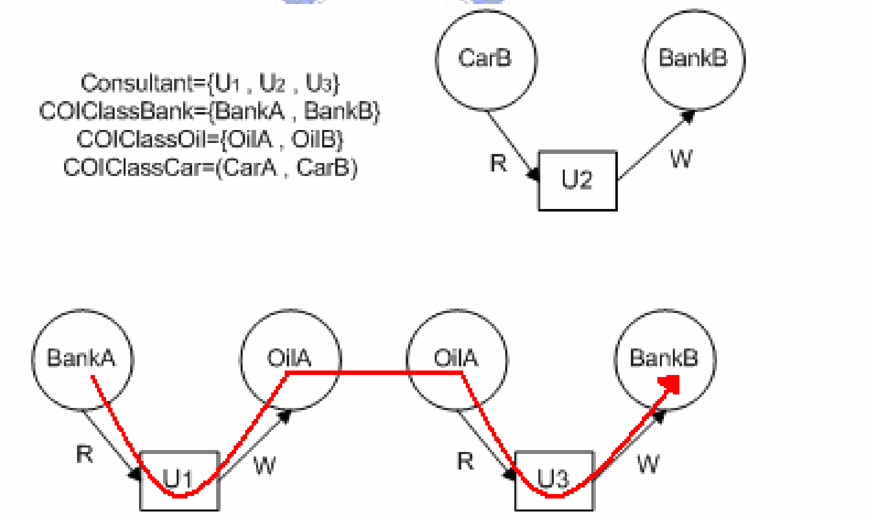
*2: In China wall security policy we simply the subject to be the consultant and the object to be company.



Consultant={U1 , U2 , U3}
COIClassBank={BankA , BankB}
COIClassOil={OilA , OilB}
COIClassCar=(CarA , CarB)
CarB
BankB
R
U2
W
BankA
OilA
R
U1
W
OilA
CarA
R
U3
W

Fig. 10 No Possible Information Flow Violates the Modified Scheme

This information flow satisfies our modification rule. In fact, all the relations in Fig.10 satisfy our modification write rule. Let us show an example in Fig.11that violates our rule.



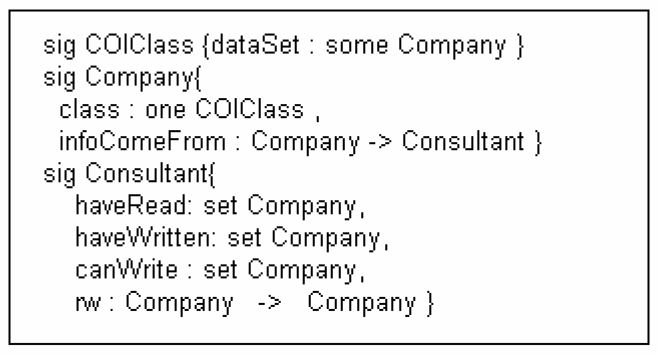
Consultant={U1 , U2 , U3}
COIClassBank={BankA , BankB}
COIClassOil={OilA , OilB}
COIClassCar=(CarA , CarB)
CarB
BankB
R
U2
W
BankA
OilA
OilA
BankB
R
U1
W
R
U3
W

Fig. 11 Example that Violates the Modified Scheme

In Fig.11 the red line marks out the case that violates our write rule (B). Here is the information flow of the case: BankA->OilA (by U_1), OilA-> BankB (by U_2). This situation would let BankB to have information which belongs to BankA and the flow is not obeying the write rule and should be eliminated in simulation/verification process.

4.2.2 Logic Specification of the China Wall Security Policy Extension

4.2.2.1 Set and Relation Declaration



```
sig COIClass {dataSet : some Company }
sig Company{
  class : one COIClass ,
  infoComeFrom : Company -> Consultant }
sig Consultant{
    haveRead: set Company,
    haveWritten: set Company,
    canWrite : set Company,
    rw : Company  ->  Company }
```

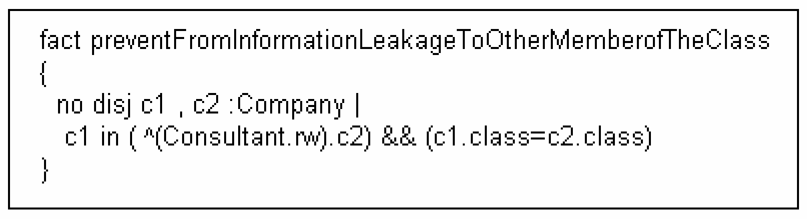
Fig. 12 Set and Relation declaration (CWall)

In Fig.12, Line 1: It declares conflict of interest set (COIClass) and each element in COIClass has a relation named dataSet. DataSet is a relation that maps each COIClass to some company(s) which the company(s) belongs to this COIClass.

Line2-4: Each company belongs to exactly one COIClass (*3) and each element in Company has a relation named infoComeFrom. InfoComeFrom is a relation that records that which company information has been written into the company and by whom did the action.

Line5-8: Each consultant has 4 relations which are haveRead, haveWritten, canWrite and rw. Rw relation represents which company(s) has been read by the consultant and the company information was written to the other company.

4.2.2.2 Apply Modified Write Rule



```
fact preventFromInformationLeakageToOtherMemberofTheClass
{
  no disj c1 , c2 :Company |
   c1 in ( ^(Consultant.rw).c2) && (c1.class=c2.class)
}
```

Fig. 13 Modified Write Rule (CWall)

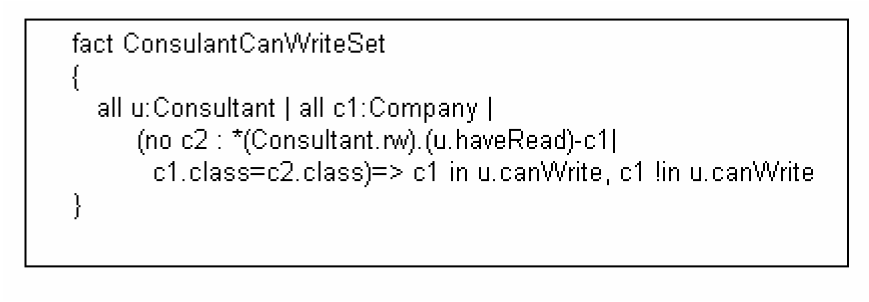
In Fig. 13, we apply the modified write rule into the model.

Line3: There are no two different company elements which can make the two formulas happen simultaneously. First formula is that if the two companies belong to the same conflict of interest class, and the second formula is if one company information flow to another company.

*3. Actually not every company can only belong to one conflict of interest class, i.e., A company could be both bank and insurance company, such as Cathy united bank.It has both bank business and insurance business. In the exercise “A Chinese Wall Approach to Privacy Policies for the web” [6], the researcher has such point of view that each information group can belong to more than one conflict of interest class.

4.2.2.3 Other Facts

After the read and write relation declarations, what companies can a user still have write access right?



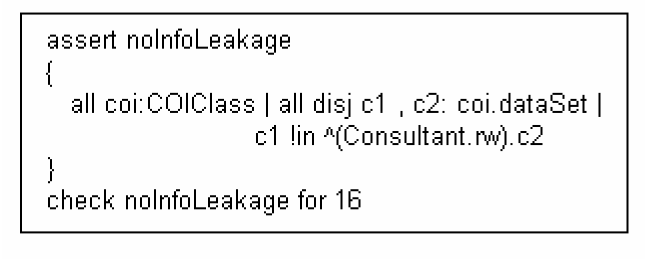
```
fact ConsulantCanWriteSet
{
  all u:Consultant | all c1:Company |
      (no c2 : *(Consultant.rw).(u.haveRead)-c1|
        c1.class=c2.class)=> c1 in u.canWrite, c1 !in u.canWrite
}
```

Fig. 14 Consultant's CanWrite Set (CWall)

In Fig.14:

Line3-5: A user's canWrite set is judged based on what company he has read and the existing rw (read-write) relations. Based on the haveRead and rw relation, we use the transitive and reflexive closure which generates a set of company, say Com1. For any company, say c2, if there's no company belongs to (Com1-c2) that c1 and c2 are in the same conflict of interest class , then the consultant has the write access right on c2.

4.2.2.4 Assertion and Verification on Transitive Relation for Information Leakage



```
assert noInfoLeakage
{
  all coi:COIClass | all disj c1 , c2: coi.dataSet |
                   c1 !in ^(Consultant.rw).c2
}
check noInfoLeakage for 16
```

Fig. 15 Assertion for checking possible violation (CWallExt)

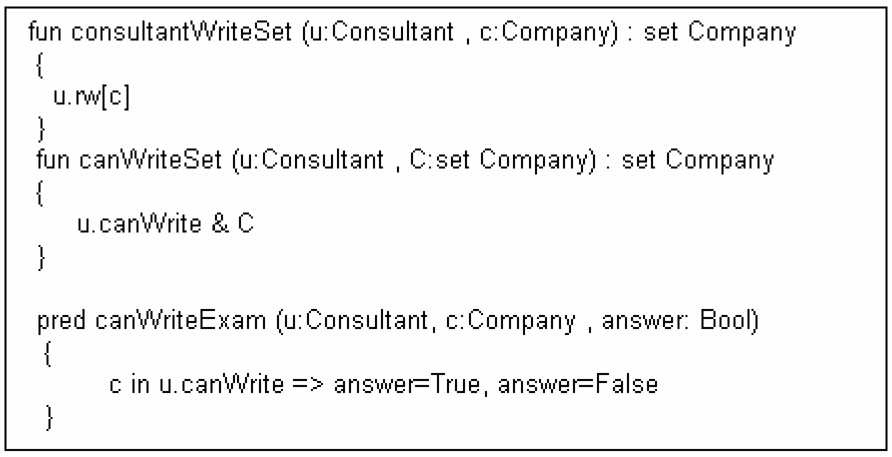
In Fig.15, we assert that there's no information flow which would cause the situation like the example in Fig.12. Indeed, there's no counter example in the verification process to show that the modeling program will lead to the violation cases.

4.2.2.5 Additional Functions Offering

In the code, we still offer some functions for user to use. We list 3 of the functions in Fig.16,

(A) The consultantWriteSet function: Input a consultant, say John, and a company, say BankA and according to the rw relation, it returns the set of document that was written by the John and John wrote BankA's information into the set of document.

(B) The canWriteSet function: Input a consultant and a set of company, it returns the set of company that the consultant still has write access right.



```
fun consultantWriteSet (u:Consultant , c:Company) : set Company
 {
   u.rw[c]
 }
 fun canWriteSet (u:Consultant , C:set Company) : set Company
 {
      u.canWrite & C
 }

 pred canWriteExam (u:Consultant, c:Company , answer: Bool)
  {
          c in u.canWrite => answer=True, answer=False
  }
```

Fig. 16 Three Additional Functions in China Wall Policy Extension

(C) The canWriteExam predicate is to test that giving a consultant, and a company if the company has the write access right.

4.2.2.6 Instance Satisfies the Specific scheme

We show readers one the simulation case in Fig.17 in next page.

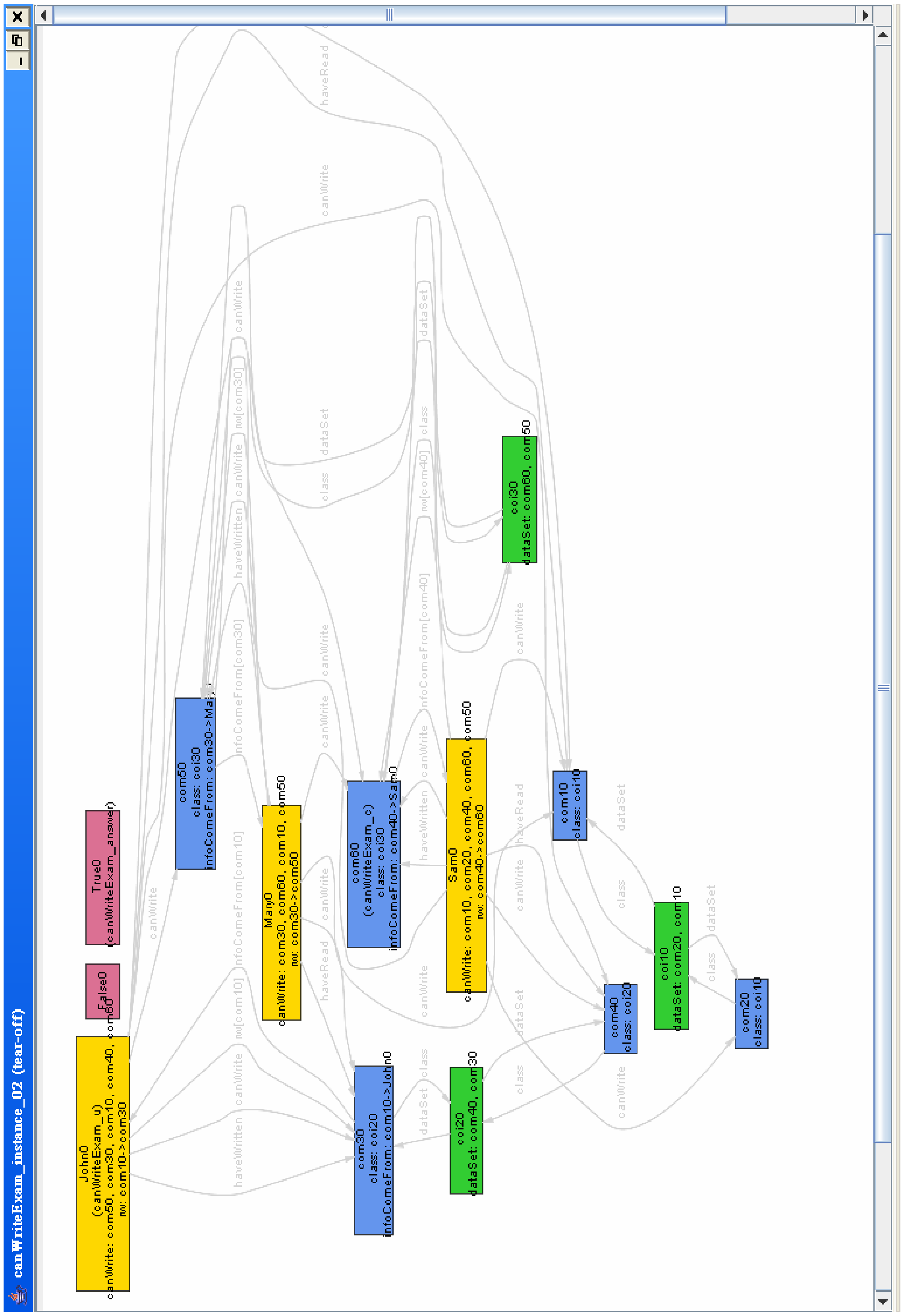

canWriteExam_instance_02 (tear-off)
John0 (canWriteExam_u) canWrite: com50, com30, com10, com40, com60 rw: com10->com30
False0
True0 (canWriteExam_answer)
com50 class: coi30 infoComeFrom: com30->Mary0
Mary0 canWrite: com30, com60, com10, com50 rw: com30->com50
com60 (canWriteExam_c) class: coi30 infoComeFrom: com40->Sam0
com30 class: coi20 infoComeFrom: com10->John0
coi20 dataSet: com40, com30
Sam0 canWrite: com10, com20, com40, com60, com50 rw: com40->com60
coi30 dataSet: com60, com50
com10 class: coi10
com40 class: coi20
coi10 dataSet: com20, com10
com20 class: coi10

Fig. 17 Instance (CWall Ext)

4.3 Conclusion

Unlike the Brew-Nash model for China wall security policy, there couldn’t be any information flow to the other companies. Our China wall security policy extension makes the situation less restrictive, more reasonable and make it possible for one company to have another company’s information. But the extension policy still obeying the principal that for all company belong to the same conflict of interest class, no information could flow from one company to another.

Chapter 5: Role-Based Access Control Verification Schema

In this chapter, we first introduce role-based access control policy and its structure with separation of duty. We apply role base access control policy on Bell-LaPadula model in the next section.

5.1 Role-Based Access Control (RBAC)

Access is the ability to do something with a computer resource (e.g. read, write, modify …etc). Access control is the means by which the ability is explicitly enabled or restricted in some way. Traditional access control, such as DAC, MAC, is the mechanism by which a system constrains the actions of a user. When a new user account is created, the system has to assign access permission right to the new user. In the system, maybe there are new users who are assigned to the same access permission, but the system has to assign same permissions to different user twice. Situations like this will require extensive and huge loads of system administration work. This is main drawback of the traditional access control policy.

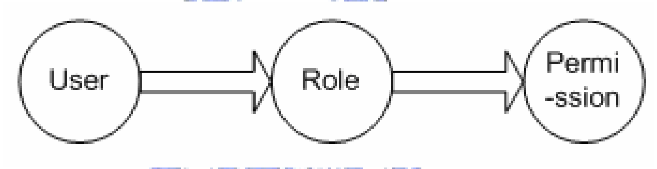


Fig. 18 RBAC Basic Structure

In the 1990's David Ferraiolo and Richard Kuhn proposed an access control policy which named The Role-Based Access Control [7]. Fig.18 shows the basic structure of the role-base access control. The role-based access control policy is different from the traditional access point of view. Instead of assigning permission to users, the system assigns permission to roles. The process of defining roles should base on a thorough analysis of how the organization operates. And the access decisions are based on roles. Each user will be the members of the roles which means, every user can be assigned more than one role. Once a person joins the organization, the only task the system has to do is assign roles to the new member. This improves the system effectiveness and decreases the system administration work.

5.1.1 User and Role

Under the framework of RBAC, each user is granted membership into role(s) based on their responsibilities in the organization. The permission that a user can perform is based on the user's role(s).User membership into roles can be revoked and assigned easily. Role associations can be established when a new permission is created.

5.1.2 Role Hierarchy and Separation of Duty (SoD)

Role hierarchy defines roles that may contain other roles, that is one role may include other role's permissions .Fig. 19 [8] shows one example of role hierarchy.

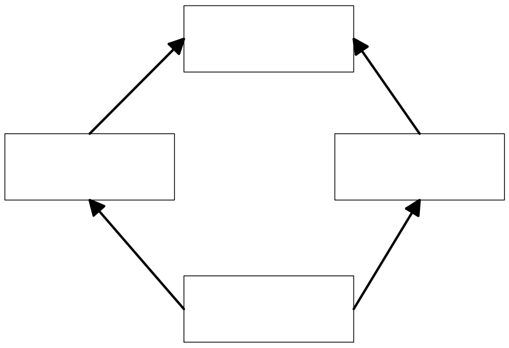


Fig. 19 Role Hierarchy of a Project team

The role hierarchy is a way of organizing roles to reflect each role's responsibilities toward the organization. The role in which the user is assigned is not mutually exclusive with another role for which the user already possesses membership .But when it comes the idea of separation of duty (SoD), two roles may be in conflict, that is, the two roles cannot be provided to the same user .They have to be provided to different user to avoid permission abuse. For example, in fig.19, the role Engl and the role Test Engr are in conflict .If one user are assigned to both roles, and if the user does not pay full responsibility to his job, he may want to forge the test data to conceal the possible bugs in his design. This circumstance leads the possible failure of the project. That's why the concept of separation of duty was proposed. An organization will function even better under the idea SoD.

5.1.3 Roles and Permissions

In role-based access control, each role is assigned certain permissions. A permission consists of two parts which are operations and target objects. See Fig.20.

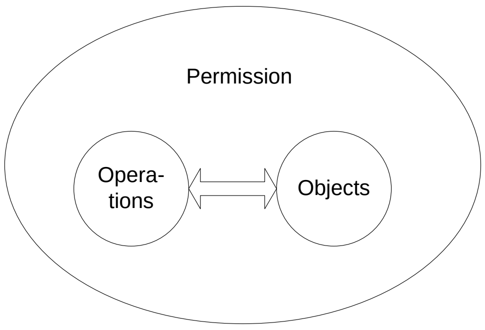


Fig. 20 The Components of a Permission

Organizations can establish rules for the association of permission with roles. For example, in a healthcare organization, the role pharmacist can be provided the permission to dispense but not to prescribe, medication [11].

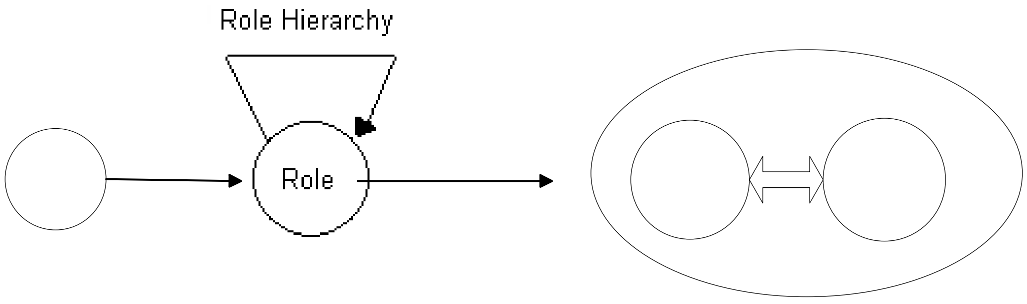
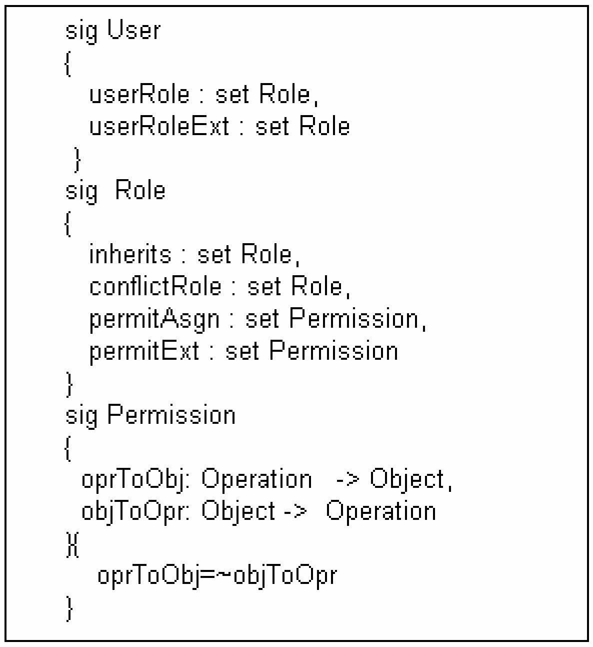


Fig. 21 Entity Relation of Diagram of RBAC

5.2 Logic Specification of RBAC

5.2.1 Set and Relation Declaration



```
sig User
{
   userRole : set Role,
   userRoleExt : set Role
 }
sig  Role
{
   inherits : set Role,
   conflictRole : set Role,
   permitAsgn : set Permission,
   permitExt : set Permission
}
sig Permission
{
  oprToObj: Operation   -> Object,
  objToOpr: Object ->  Operation
}{
    oprToObj=~objToOpr
}
```

Role-Per
Assig

Fig. 22 Set and Relation Declartaion (RBAC)

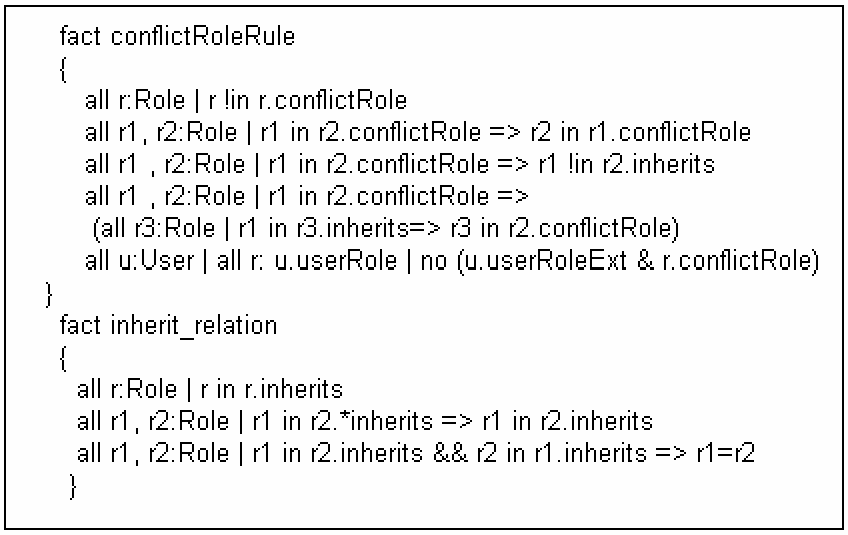
In fig. 22,

Line1-5: Each user has two relations which are userRole and userRoleExt. UserRole maps form user to roles that system assigned. UserRoleExt are the role collections of userRole and the role which userRole inherits.

Line6-12: Each role has four relations which are inherits, conflicts, permitAsgn, permitExt. Inherits relations represent that which role(s) that the role inherits. Conflict relations are being used for SoD function. PermitAsgn relation represents what permission(s) this role possesses.

Line13-19: Each permission has two relations which are oprToObj and objToOpr. OprToObj relation means what operation can perform on the object under permission. ObjtoOpr and oprToObj are inverse relation.

5.2.2 Constraints of Separation of Duty and Role Hierarchy



```
fact conflictRoleRule
{
   all r:Role | r !in r.conflictRole
   all r1 , r2:Role | r1 in r2.conflictRole => r2 in r1.conflictRole
   all r1  , r2:Role | r1 in r2.conflictRole => r1 !in r2.inherits
   all r1  , r2:Role | r1 in r2.conflictRole =>
    (all r3:Role | r1 in r3.inherits=> r3 in r2.conflictRole)
   all u:User | all r: u.userRole | no (u.userRoleExt & r.conflictRole)
}
fact inherit_relation
{
  all r:Role | r in r.inherits
  all r1 , r2:Role | r1 in r2.*inherits => r1 in r2.inherits
  all r1 , r2:Role | r1 in r2.inherits && r2 in r1.inherits => r1=r2
 }
```

Fig. 23 Constraints on SoD

In fig.23, first fact statement define the constraints on conflictRole and the second one defines the inherit relation which is a partial order relation.

Line3-4: Every role does not in its conflictRole relation. For any two roles, say r1 r2, if r1 is in r2's conflict role, it implies that r2 is in r1's conflictRole.

Line5: If two roles are in conflict, it impliesy no one can inherit the other.

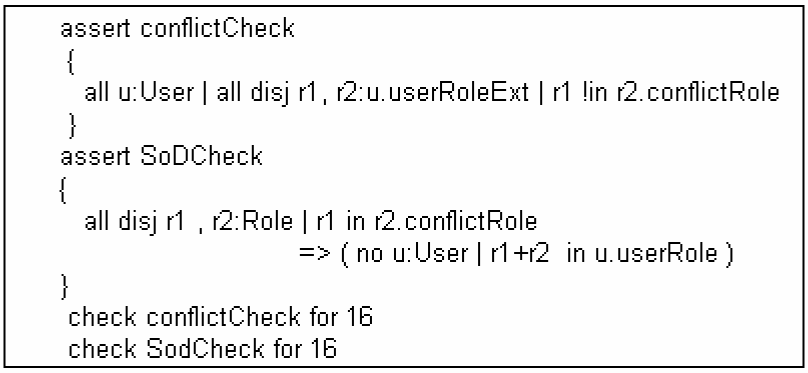
Line6-7: If two roles, say r1 r2, if they are in conflict, it implies for any r3 that inherits r1, r3 and r2 are in conflict.

Line8: For all roles, say r1, belongs to user's role set, the intersection of user's role extension and r1's conflict role set is empty set.

Line10-15: The fact describes the partial order relation among roles.

5.2.3 Assertion and Verification

In fig. 24, we assert that the role assignment and inheritance relation will not generate any cases that violate the principal of SoD. After we checked all the cases under certain scope, there are no counter-examples.

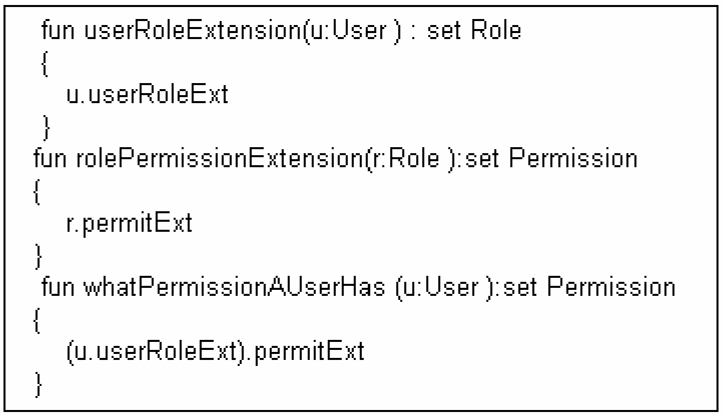


```
assert conflictCheck
 {
   all u:User | all disj r1, r2:u.userRoleExt | r1 !in r2.conflictRole
 }
assert SoDCheck
{
   all disj r1  , r2:Role | r1 in r2.conflictRole
                         => ( no u:User | r1+r2  in u.userRole )
}
 check conflictCheck for 16
 check SodCheck for 16
```

Fig. 24 Conflict Check on SoD

5.2.4 Additional Functions Offering

Like we mentioned in previous chapter, some lazy user may just want to obtain certain answer .In RBAC, we still offer three functions for user to use. In Fig.25 A user may just want to know what role extension he has, he may use the function"userRoleExtension".The function "rolePermissionExtension" returns the set of the permission extension that the roles possess. The last function "permissionThatAUserHas" gives the permission set according to the relation from user-role and role-permisson. It helps the user to realize what permissions he has.



```
 fun userRoleExtension(u:User ) : set Role
 {
    u.userRoleExt
 }
fun rolePermissionExtension(r:Role ):set Permission
{
    r.permitExt
}
 fun whatPermissionAUserHas (u:User ):set Permission
{
    (u.userRoleExt).permitExt
}
```

Fig. 25 Additonal Function Offering (RBAC)

5.2.5 Instance Satisfies the Specific Scheme

According to the specific scheme (Fig.26) that we apply on RBAC code, alloy generates one of the instances in fig.27 and fig.28.

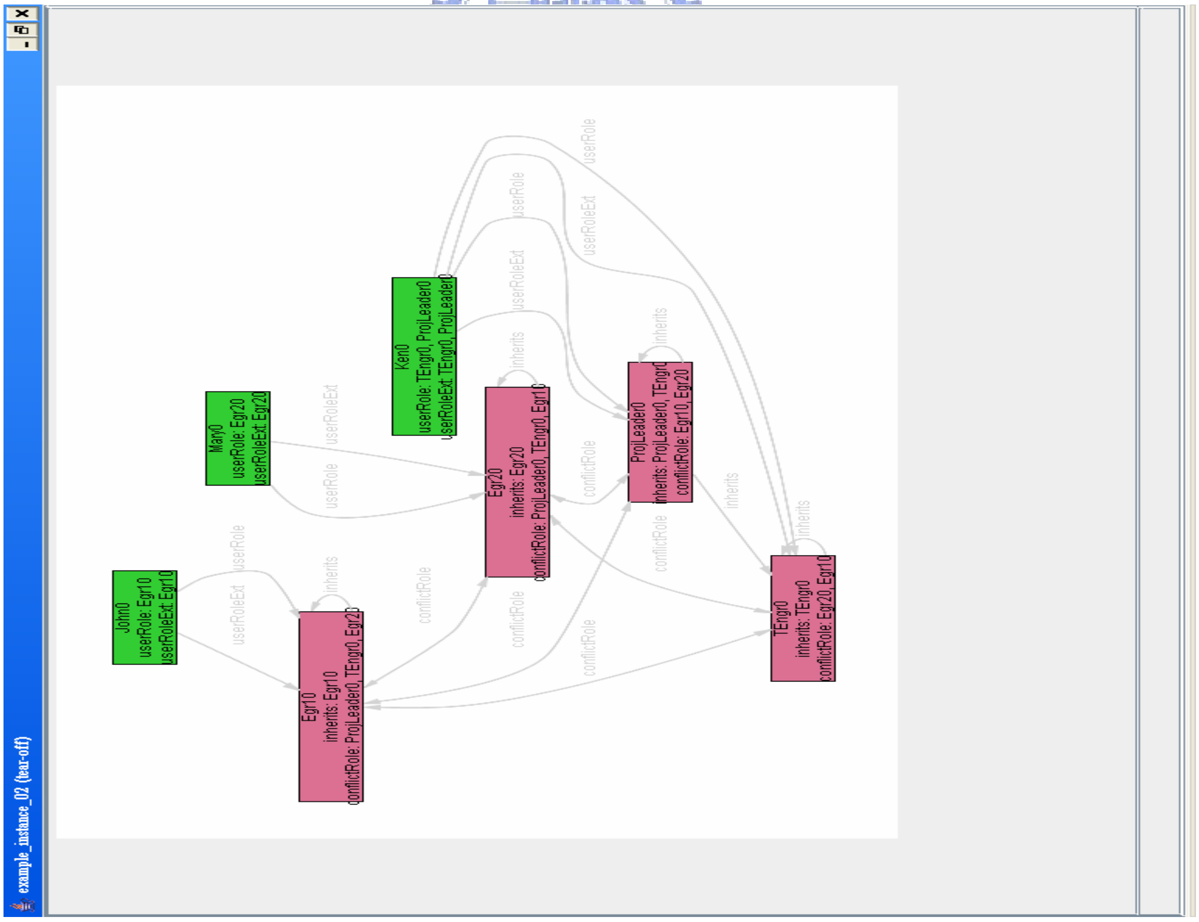
```
fact
 {
  p1.oprToObj =Write-> MgmtFile
  p2.oprToObj=Write-> DesignFile
  p3.oprToObj=Write-> TestFile
  p4.oprToObj=Read->MgmtFile
  p5.oprToObj=Read->DesignFile
  p6.oprToObj=Read->TestFile

  ProjLeader.permitAsgn=p1+p4+p5+p6
  Egr1.permitAsgn=p2+p5+p6
  Egr2.permitAsgn=p2+p5+p6
  TEngr.permitAsgn=p3+p6
 }
```

```
fact
{
 Egr1 in Egr2.conflictRole
 Egr2 in TEngr.conflictRole
 TEngr in Egr1.conflictRole
 TEngr in ProjLeader.inherits
 }
```

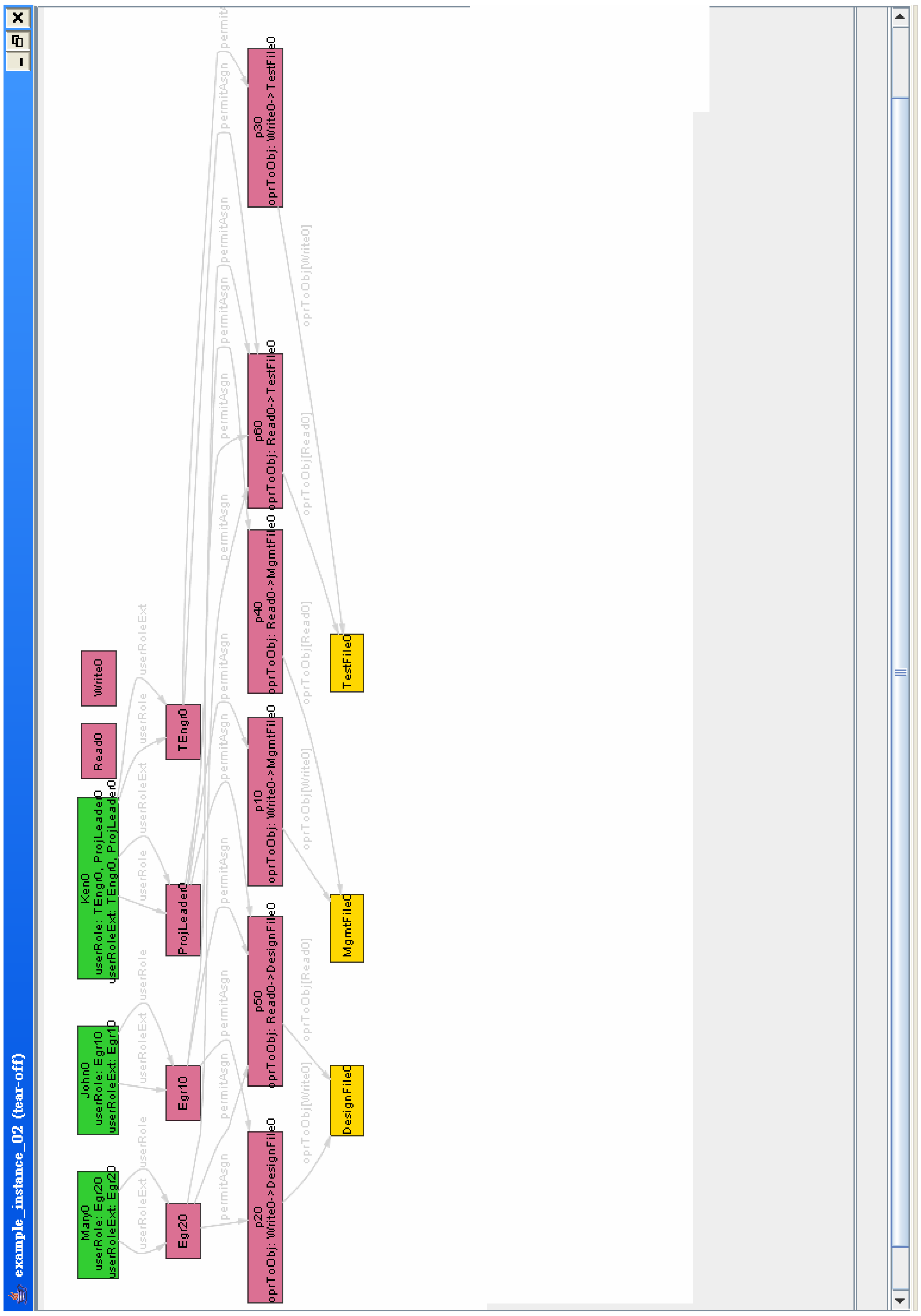
Fig. 26 Specfic Scheme (RBAC)

In Fig.27, it shows that the four roles only require 3 users to accomplish the task. In the instance, John is assigned with role "Engr1', Mary is assigned with role "Engr2" and Ken is assigned with role "ProjLeader" and role "TEgnr". In fig.28, it shows the relation among user, role and permission.



example_instance_02 (tear-off)
Ken0
userRole: TEngr0, ProjLeader0
userRoleExt: TEngr0, ProjLeader0
Mary0
userRole: Egr20
userRoleExt: Egr20
John0
userRole: Egr10
userRoleExt: Egr10
Egr20
inherits: Egr20
conflictRole: ProjLeader0, TEngr0, Egr10
ProjLeader0
inherits: ProjLeader0, TEngr0
conflictRole: Egr10, Egr20
Egr10
inherits: Egr10
conflictRole: ProjLeader0, TEngr0, Egr20
TEngr0
inherits: TEngr0
conflictRole: Egr20, Egr10
userRole
userRoleExt
inherits
conflictRole

Fig. 27 Instance- Inheritance vs. Conflict Roles (RBAC)



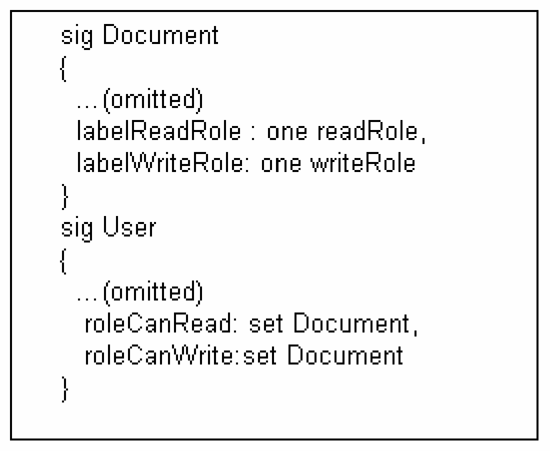
example_instance_02 (tear-off)
Mary0
userRole: Egr20
userRoleExt: Egr20
John0
userRole: Egr10
userRoleExt: Egr10
Ken0
userRole: TEngr0, ProjLeader0
userRoleExt: TEngr0, ProjLeader0
Read0
Write0
Egr20
Egr10
ProjLeader0
TEngr0
userRole
userRoleExt
permitAsgn
p20
oprToObj: Write0->DesignFile0
p50
oprToObj: Read0->DesignFile0
p10
oprToObj: Write0->MgmtFile0
p40
oprToObj: Read0->MgmtFile0
p60
oprToObj: Read0->TestFile0
p30
oprToObj: Write0->TestFile0
oprToObj[Write0]
oprToObj[Read0]
DesignFile0
MgmtFile0
TestFile0

Fig. 28 Instance with User Role and Permission

5.3 Verification on Bell-LaPadula Model using RBAC Concept

The Bell-LaPadula model is a Lattice based access control policy, in this section, we apply RBAC schema on the Bell-LaPadula model to do verification on it. Some part of the codes is the same as the code in Appendix A and Appendix C. We omit this part and explain how we apply RBAC schema on Bell-LaPadula model in the following sections.

5.3.1 New Relation and Set Declaration



```
sig Document
{
  ...(omitted)
  labelReadRole : one readRole,
  labelWriteRole: one writeRole
}
sig User
{
  ...(omitted)
   roleCanRead: set Document,
   roleCanWrite:set Document
}
```

Fig. 29 New Set and Relation Declaration (RBAC on BLP)

In Fig.29,

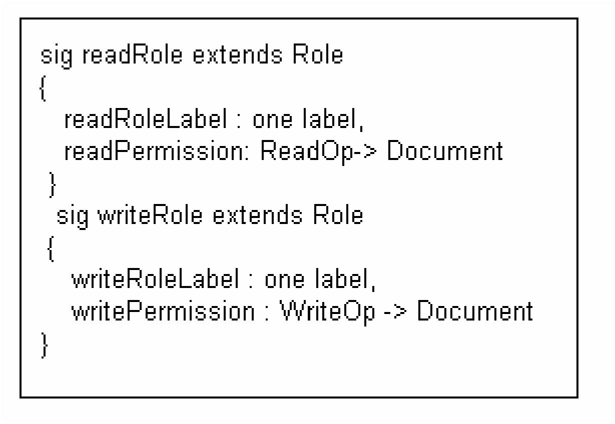
(1) In the set Documents, two new relations which are labelReadRole and labelWriteRole were added.The labelReadRole is a relation maps from label to one readRole , the labelWriteRole maps from label to writeRole.

(2) In the set User, two new relations are roleCanRead and roleCanWrite.The roleCanRead maps user to documents so does the roleCanWrite relation.

In Fig.30,

(1) In the set readRole, two relations are readRoleLabel and readPermission. The readRoleLabel maps readRole to one label and the readPermission represents permission this readRole possesses.

(2) In the set writeRole, two relations which are writeRoleLabel and writePermission were defined.The writeRoleLabel maps writeRole to one label and the writePermission represents permissions the writeRole possesses.

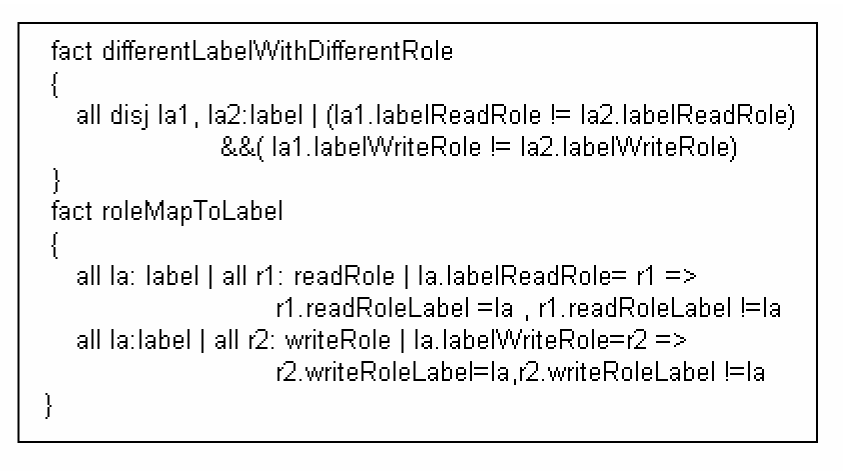


```
sig readRole extends Role
{
   readRoleLabel : one label,
   readPermission: ReadOp-> Document
 }
  sig writeRole extends Role
 {
    writeRoleLabel : one label,
    writePermission : WriteOp -> Document
}
```

Fig. 30 Declaration (RBAC on BLP) Part 2

5.3.2 Mechanism on Applying RBAC to BLP

5.3.2.1 Different Label with Different Read, Write Role



```
fact differentLabelWithDifferentRole
{
   all disj la1, la2:label | (la1.labelReadRole != la2.labelReadRole)
                  &&( la1.labelWriteRole != la2.labelWriteRole)
}
fact roleMapToLabel
{
   all la: label | all r1: readRole | la.labelReadRole= r1 =>
                        r1.readRoleLabel =la , r1.readRoleLabel !=la
   all la:label | all r2: writeRole | la.labelWriteRole=r2 =>
                        r2.writeRoleLabel=la,r2.writeRoleLabel !=la
}
```

Fig. 31 Different Label with Different Read, Write Role

In fig.31,

Line3-4: For any two different labels, their labels on readRole are different, so are the labels on writeRole.

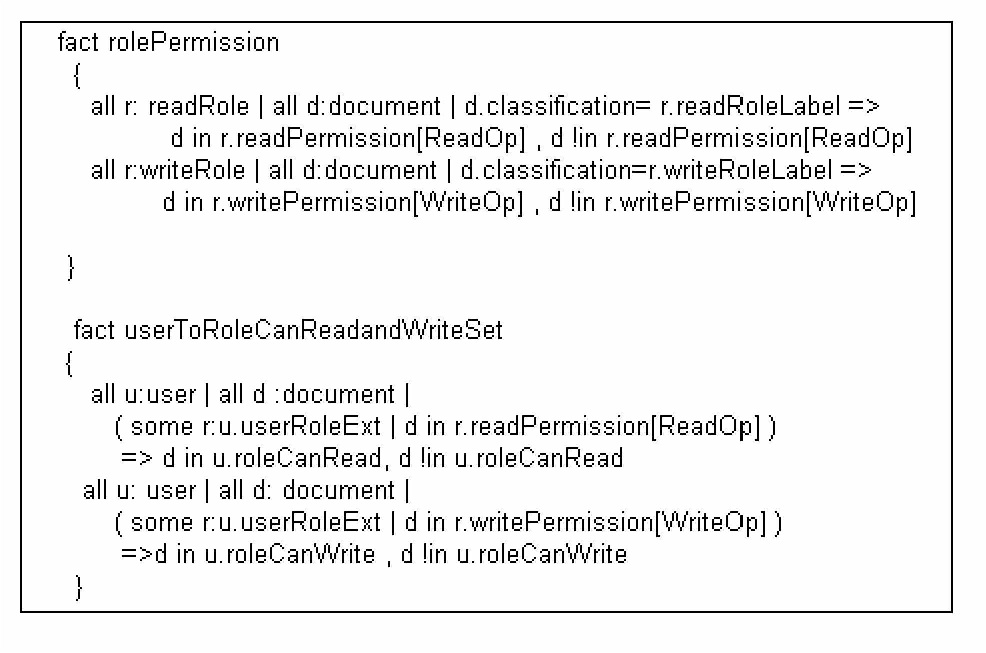
Line8-11: Each role maps to a readRole label and writeRole label.

5.3.2.2 Role Permission

In fig.32,

Line 1-7: We define the set of documents each readRole can read, and set of documents each writeRole can write.If the document's classification is the same as the role's readRole label then this document can be read by the role.Same method applies on writeRole.

Line 9-17: We define the set of documents that each user can read and can write according to the role that was assigned to them.



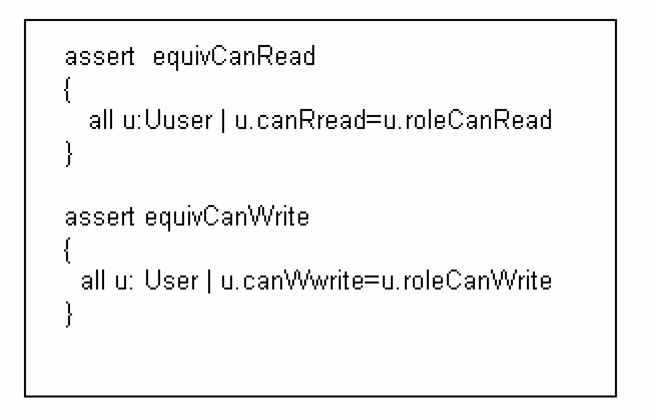
```
fact rolePermission
 {
   all r: readRole | all d:document | d.classification= r.readRoleLabel =>
            d in r.readPermission[ReadOp] , d !in r.readPermission[ReadOp]
   all r:writeRole | all d:document | d.classification=r.writeRoleLabel =>
           d in r.writePermission[WriteOp] , d !in r.writePermission[WriteOp]

 }

 fact userToRoleCanReadandWriteSet
{
   all u:user | all d :document |
      ( some r:u.userRoleExt | d in r.readPermission[ReadOp] )
       => d in u.roleCanRead, d !in u.roleCanRead
  all u: user | all d: document |
      ( some r:u.userRoleExt | d in r.writePermission[WriteOp] )
       =>d in u.roleCanWrite , d !in u.roleCanWrite
 }
```

Fig. 32 Role Permission(RBAC on BLP)

5.3.2.3 Assertion and Verification on the Role-based BLP Model



```
assert  equivCanRead
{
   all u:Uuser | u.canRread=u.roleCanRead
}

assert equivCanWrite
{
  all u: User | u.canWwrite=u.roleCanWrite
}
```

Fig. 33 Verification (Role-Based BLP)

In Fig33, in line3, we assert that for any user, say u, u’s canRead set is equal to the user’s roleCanRead set. Same assertion was made on equivCanWrite.

5.3.2.4 Instance

After the verification process, we found out these two assertion hold thorough out the model .It means we successfully apply the role-based access control schema on the Bell-LaPadual model.Fig.34 (in next page) shows one of the instanecs (Due to the size of the graph , we chose not to show attributes in order to let the graph fit in the page).

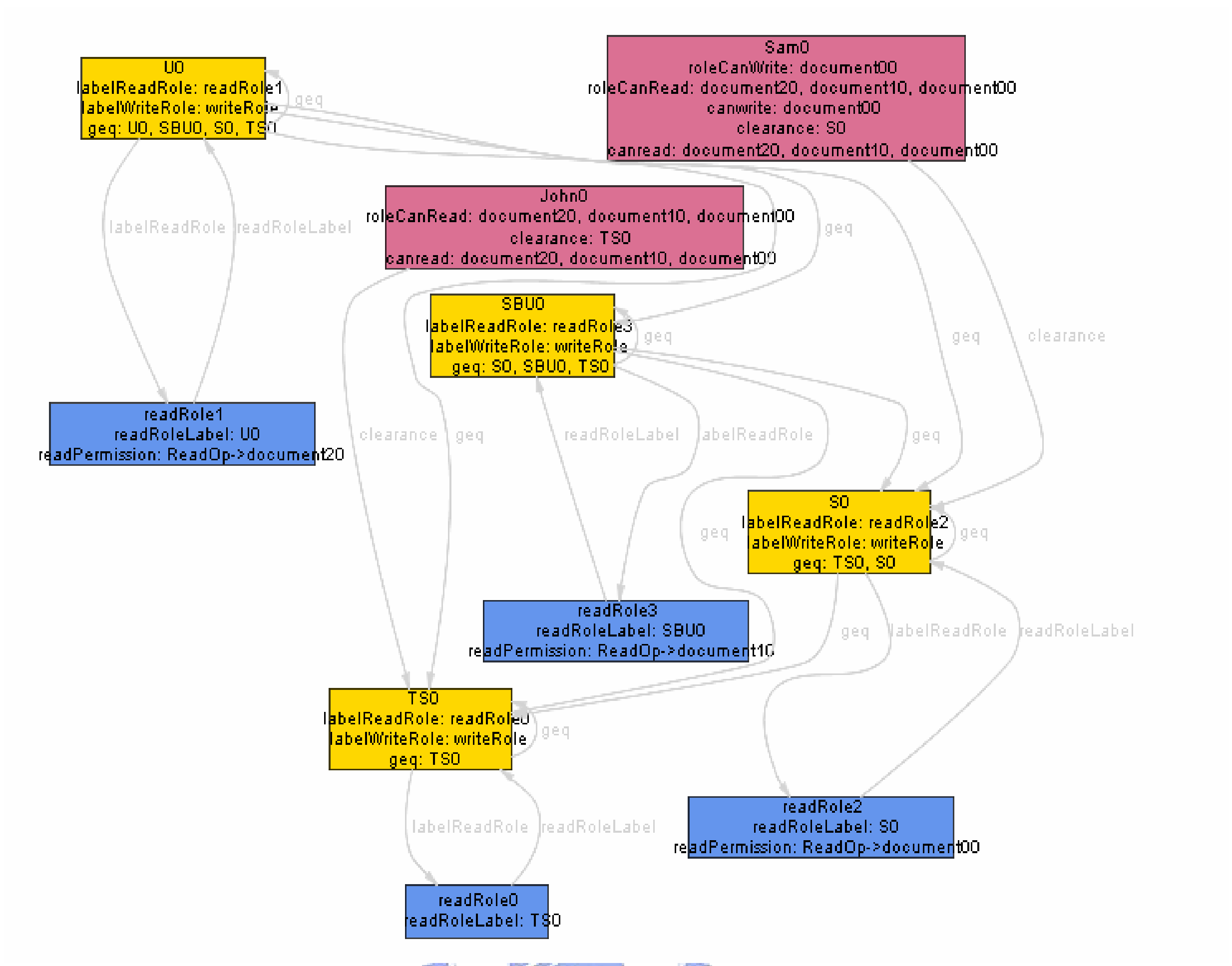


Fig. 34 Instance (RBAC on BLP)

Chapter 6: Conclusion and Discussion

From the exercise of verification of the three access control policy, we successfully verify that each access control has met the fulfillment of certain algebraic properties. For example, the China Wall Security Policy Extension has been verified under the transitive relation that one company's information doesn't leak to the other companies that belong to the same conflict of interest class of that company.

In table 1 and table 2, we show the run time/bits of state vs. size of model of the two assertions in RBAC code.

Size of Model	Run Time(min:sec)/ Bits of State
2^5	0:01/54
4^5	0:02/252
8^5	0:15/1472
16^5	3:08/9840

Table 1: Run Time and Bits of the Assert-permitExtCheck

Size of Model	Run Time(min:sec)/ Bits of State
2^5	0:01/54
4^5	0:02/256
8^5	0:14/1472
16^5	3:10/9856

Table 1: verification data of RBAC assertion-conflictCheck

References

[1] Alloy main page: http://alloy.mit.edu

[2] “Alloy Reference Manual “Daniel Jackson, May 2004

[3] “Secure Computer System: Mathematical Foundations and Models”, D. E. Bell and L. J. LaPadula , Mitre Corp., BedFord ,MA , 1975

[4] “The Chinese Wall Security Policy “, David F.C. Brewer and Michael J.Nash, IEEE symposium, May 1989, page 206-214

[5] “ Lattice-Based Enforcement of Chinese Walls “ Ravi S. Sandhu, Computer& Security , Volumn11, Number 8, December 1992 , page 753-763

[6] “A Chinese Wall Approach to Privacy Policies for the Web “Frans A. Lategen and Martin S. Oliver, COMPSAC ’02

[7] “Role-Based Access Control”, David Ferraiolo and Richard Kuhn , 15th National Computer Security Conference ,1992

[8] “RBAC Schema Verification Using Lightweight Formal Model and Constraint Analysis”, John Zao, HoetTech Wee, Johnathan Chu, Daniel Jackson, SACMAT, 2003

[9] “The UCON ABC Usage Control Model”, J PARK, R SANDHU - ACM Transactions on Information and System Security, 2004

[10] “Algorithm for the Satisfiability (SAT) Problem”, J. Gu, P. W. Purdom, J. Franco and B. W. Wah, DIMACS Series in Discrete Mathematics and Theoretical Computer Science , 1996

[11] "An Introduction to Role Based Access Control" NIST CSL Bulletin on RBAC, December, 1995

Appendix

Appendix A :Bell-LaPadula

```
module   models/BLP/InputSet

sig label
{
 geq : some label
}
one sig   S , TS , SBU , U extends label {}
fact priority
{
   TS in S.geq
   S in SBU.geq
   SBU in U.geq
}
sig user
{
   clearance : one label,
   canread :    set document,
   canwrite : set document
 }
sig document
{
   classification : one label
}
fact PartialOrderRelation
{
   /* transitive and reflexive : reflexive and transitive closure*/
     all   la1 , la2 : label | la1 in la2.*geq => la1 in la2.geq
   /*antisymetric*/
     all la1 , la2 : label | la1 in la2.geq && la2 in la1.geq => la1=la2
}
 /* all x:X | formula - every x of type X satisfies formula. */
fact readdown
{
     all u: user | all d: document      |      u.clearance   in d.classification.geq
                                                 =>d in u.canread, d !in u.canread
}
```

```
fact writeup
{
  all u: user | all d: document | d.classification in (u.clearance).geq
                                        =>d in u.canwrite , d !in u.canwrite
}
    fun ReadSet_BLP (D:set document , u:user):set document
    {
        u.canread & D
    }

    pred checkForReadAccess (d: document , u: user    ,answer:Bool)
    {
      d in u.canread => answer=True , answer=False
    }

    fun WriteSet_BLP (D:set document , u:user):set document
    {
         u.canwrite & D
    }

    pred checkForWriteAccess (d: document , u: user    ,answer:Bool)
    {
        d    in u.canwrite => answer=True , answer=False
    }

    pred example(){}


    one sig John , Sam, Mary extends user{}
    fact
    {
      John.clearance=TS
      Sam.clearance=S
      Mary.clearance=U
    }
    one sig document0 , document1 , document2 extends document{}
    fact
   {
     document0.classification=S
     document1.classification=SBU
     document2.classification=U
   }

run ReadSet_BLP for exactly 4 label,exactly 3 user , exactly 3 document,exactly 2 Bool
run checkForReadAccess for 4 label , 3 user , exactly 3 document , exactly 2 Bool
run WriteSet_BLP for exactly 4 label,exactly 3 user , exactly 3 document,exactly 2 Bool
run checkForWriteAccess for exactly 4 label,exactly 3 user, exactly 3 document, exactly 2
Bool
run example for 4 label , 3 user , exactly 5 document,exactly 2 Bool
```

Appendix B: China Wall Security Extension

```
module models/MCW
open util/boolean

sig COIClass
{
     dataSet : some Company
}
sig Company
{
     class : one COIClass ,
     infoComeFrom : Company -> Consultant

}

sig Consultant
{
     haveRead: set Company,
     haveWritten: set Company,
     canWrite : set Company,
     rw : Company     ->     Company
}


fact eachCOIClassHasAtLeastOneComBelongsToTheCOIClass
{
  all coi: COIClass | all c:Company |c.class=coi => c in coi.dataSet , c !in coi.dataSet
}
fact preventFromInfoLeakage
{
   no disj c1 , c2 :Company |c1 in ( ^(Consultant.rw).c2) && (c1.class=c2.class)
  }
fact consultantHaveReadandWrittenSet
{
   all u:Consultant | u.haveRead    = u.rw.Company
   all u:Consultant | u.haveWritten=u.rw[Company]
}
```

```
fact ConsulantCanWriteSet
{
 all u:Consultant | all c1:Company | (no c2 : *(Consultant.rw).(u.haveRead)-c1|
                                              c1.class=c2.class)=> c1 in u.canWrite, c1 !in u.canWrite
 }
fact companyInfoComeFromSetbySomeUser
{
 all u: Consultant| all c1 , c2 :Company |
 c1 in u.rw.c2 => c1->u in c2.infoComeFrom , c1->u !in c2.infoComeFrom
}
fact
{
   all u:Consultant | u.rw !=none-> none
}
fun consultantWriteSet (u:Consultant) : set Company
{
     u.rw[Company]
}
fun canWriteSet (u:Consultant , C:set Company) : set Company
{
       u.canWrite & C
}

pred canWriteExam (u:Consultant, c:Company , answer: Bool)
{
       c in u.canWrite => answer=True, answer=False
}
assert notwoComBelongsToSameCOIClass
{
 all u:Consultant | no disj c1, c2:u.rw.Company +u.rw[Company] | c1.class=c2.class
}
assert noInfoLeakage
{
   all coi:COIClass | all disj c1 , c2: coi.dataSet | c1 !in ^(Consultant.rw).c2
}
pred example(){}
check noInfoLeakage for 15
check notwoComBelongsToSameCOIClass for 10
```

```
run example for exactly 3 COIClass , exactly 6 Company , exactly 3 Consultant
run consultantWriteSet for    3 COIClass ,    6 Company ,    3 Consultant
run canWriteSet for 3 COIClass , 6 Company , 3 Consultant
run canWriteExam for 3 COIClass , 6 Company , 3 Consultant

one sig John , Sam , Mary extends Consultant{}
one sig com1 , com2 , com3, com4 , com5 , com6 extends Company{}
one sig coi1 , coi2 , coi3 extends COIClass{}
fact
{
   com1.class=coi1
   com2.class=coi1
   com3.class=coi2
   com4.class=coi2
   com5.class=coi3
   com6.class=coi3
}
Fact
{
   John.rw= com1->com3
   Mary.rw=com3-> com5
   Sam.rw=com4->com6
}
```

Appendix C: Role-Based Access Control

```
module models/RBAC/RBAC2

sig User
{
    userRole : set Role,
    userRoleExt : set Role

}
sig   Role
{
    inherits : set Role,
    conflictRole : set Role,
    permitAsgn : set Permission,
    permitExt : set Permission
 }
sig Permission
{
   oprToObj: Operation     -> Object,
   objToOpr: Object ->    Operation
}{
   oprToObj=~objToOpr
   }

sig Operation{}
sig Object{}
fact conflictRoleRule
{
    all r:Role | r !in r.conflictRole
    all r1, r2:Role | r1 in r2.conflictRole => r2 in r1.conflictRole
    all r1 , r2:Role | r1 in r2.conflictRole => r1 !in r2.inherits
    all r1 , r2:Role | r1 in r2.conflictRole =>
    (all r3:Role | r1 in r3.inherits=> r3 in r2.conflictRole)
    all u:User | all r: u.userRole | no (u.userRoleExt & r.conflictRole)
}
fact inherit_relation
{
    all r:Role | r in r.inherits
```

```
      // all r:Role | r.*inherits in r.inherits
       all r1, r2:Role | r1 in r2.*inherits => r1 in r2.inherits
       all r1, r2:Role | r1 in r2.inherits && r2 in r1.inherits => r1=r2
}
fact permissionExt
{
   all r1:Role | all p:Permission|
   (some r2: r1.inherits   | p in r2.permitAsgn )=> p in r1.permitExt , p !in r1.permitExt
}
fact userRoleExtDef
{
   all u: User | all r1:Role | ( some r2:u.userRole | r1 in r2.inherits )
                                        => r1 in u.userRoleExt , r1 !in u.userRoleExt
}
assert permitExtCheck
{
   all r1:Role | all p: r1.permitExt | p in r1.inherits.permitAsgn
}
assert SoDCheck
{
     all disj r1 , r2:Role | r1 in r2.conflictRole => ( no u:User | r1+r2 in u.userRole )
}

assert conflictCheck
{
  all u:User | all disj r1, r2:u.userRoleExt | r1 !in r2.conflictRole
}

fun userRoleExtension(u:User ) : set Role
{
     u.userRoleExt
}
fun rolePermissionExtension(r:Role ):set Permission
{
        r.permitExt
  }
fun whatPermissionAUserHas(u:User ):set Permission
{
    (u.userRoleExt).permitExt
}
pred example (){}

check permitExtCheck for 10
check conflictCheck for 10
check SoDCheck for 16
run example for 3 User , 4 Role    ,    2 Operation ,3 Object,6 Permission
one sig ProjLeader, Egr1, Egr2 , TEngr extends Role{}
one sig John    , Mary , Ken extends User{}
one sig Read , Write extends Operation{}
one sig MgmtFile , DesignFile , TestFile extends Object{}
one sig p1 , p2 ,p3 ,p4,p5,p6 extends Permission{}

fact
{
        Egr1 in Egr2.conflictRole
        Egr2 in TEngr.conflictRole
        TEngr in Egr1.conflictRole
        TEngr in ProjLeader.inherits
}
fact
{
    p1.oprToObj =Write-> MgmtFile
    p2.oprToObj=Write-> DesignFile
    p3.oprToObj=Write-> TestFile
    p4.oprToObj=Read->MgmtFile
    p5.oprToObj=Read->DesignFile
    p6.oprToObj=Read->TestFile

    ProjLeader.permitAsgn=p1+p4+p5+p6
    Egr1.permitAsgn=p2+p5+p6
    Egr2.permitAsgn=p2+p5+p6
    TEngr.permitAsgn=p3+p6
}

fact
{
        all r:Role | r in    User.userRole
}
```