การกำหนดรูปแบบการควบคุมการเข้าถึงแบบโรลเบสสำหรับองค์กรขนาดใหญ่โดยใช้หลักการแบ่งแยกหน้าที่แบบสแตติก

AN ENHANCED ROLE-BASED ACCESS CONTROL MODEL FOR LARGE ENTERPRISES USING STATIC SEPARATION OF DUTY CONCEPT

บุรินทร์ เย็นมั่นคง
BURIN YENMUNKONG

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาเทคโนโลยีสารสนเทศ
บัณฑิตวิทยาลัย
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ. 2547

ISBN 974-9708-85-7

การกำหนดรูปแบบการควบคุมการเข้าถึงแบบโรลเบสสำหรับองค์กรขนาดใหญ่โดยใช้หลักการแบ่งแยกหน้าที่แบบสแตติก

AN ENHANCED ROLE-BASED ACCESS CONTROL MODEL FOR LARGE ENTERPRISES USING STATIC SEPARATION OF DUTY CONCEPT

บุรินทร์ เย็นมั่นคง
BURIN YENMUNKONG

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาเทคโนโลยีสารสนเทศ
บัณฑิตวิทยาลัย
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ.2547

ISBN 974-9708-85-7

AN ENHANCED ROLE-BASED ACCESS CONTROL MODEL FOR LARGE ENTERPRISES USING STATIC SEPARATION OF DUTY CONCEPT

BURIN YENMUNKONG

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENT FOR THE DEGREE OF
MASTER OF SCIENCE IN INFORMATION TECNOLOGY
SCHOOL OF GRADUATE STUDIES
KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG
2004

ISBN 974-9708-85-7

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ใบรับรองวิทยานิพนธ์

หัวข้อวิทยานิพนธ์ การกำหนดรูปแบบการควบคุมการเข้าถึงแบบโรลเบสสำหรับองค์กรขนาดใหญ่โดยใช้หลักการแบ่งแยกหน้าที่แบบสแตติก

AN ENHANCED ROLE-BASED ACCESS CONTROL MODEL FOR LARGE ENTERPRISES USING STATIC SEPARATION OF DUTY CONCEPT

ชื่อนักศึกษา นายบุรินทร์ เย็นมั่นคง

รหัสประจำตัว 44067032

ปริญญา วิทยาศาสตรมหาบัณฑิต

สาขาวิชา เทคโนโลยีสารสนเทศ

อาจารย์ผู้ควบคุมวิทยานิพนธ์ ผศ.ดร.จันทร์บูรณ์ สถิตวิริยวงศ์

คณะกรรมการสอบวิทยานิพนธ์		ลายมือชื่อ
ผศ.ดร.จันทร์บูรณ์	สถิตวิริยวงศ์	
ผศ.ดร.นพพร	โชติกกำธร	
ผศ.ดร.โชติพัชร์	ภรณวลัย	
ผศ.อัครินทร์	คุณกิตติ	
ดร.วัชระ	ฉัตรวิริยะ	

วัน/เดือน/ปี ที่สอบ 20 พฤษภาคม 2547 เวลา 10.00 น. เป็นต้นไป

สถานที่สอบ ณ ห้อง M 21 (ชั้นลอย) อาคารเรียนรวมและปฏิบัติการคณะเทคโนโลยีสารสนเทศ

บัณฑิตวิทยาลัยรับรองแล้ว

(ผศ.ดร.จารุวัตร เจริญสุข)

คณบดีบัณฑิตวิทยาลัย

วันที่ 28 **เดือน** พฤษภาคม **พ.ศ.** 2547

หัวข้อวิทยานิพนธ์	การกำหนดรูปแบบการควบคุมการเข้าถึงแบบโรลเบสสำหรับองค์กรขนาดใหญ่โดยใช้หลักการแบ่งแยกหน้าที่แบบสแตติก
นักศึกษา	นายบุรินทร์ เย็นมั่นคง
รหัสประจำตัว	44067032
ปริญญา	วิทยาศาสตรมหาบัณฑิต
สาขาวิชา	เทคโนโลยีสารสนเทศ
พ.ศ.	2547
อาจารย์ผู้ควบคุมวิทยานิพนธ์	ผศ.ดร.จันทร์บูรณ์ สถิตวิริยวงศ์

บทคัดย่อ

องค์กรขนาดใหญ่ให้ความสำคัญต่อการรักษาความปลอดภัยในเรื่องของการควบคุมการเข้าถึงเป็นอย่างมาก ซึ่งการควบคุมการเข้าถึงแบบโรลเบสได้รับความสนใจในปัจจุบัน และมีการพัฒนาอย่างต่อเนื่องดังเช่นแบบดีสครีทกับแบบแมนดาทอรี ถึงแม้ว่าการควบคุมการเข้าถึงแบบโรลเบสเป็นรูปแบบที่ดี แต่การบริหารจัดการ รวมถึงการสร้างและดูแลรักษาข้อมูลการควบคุมการเข้าถึงยังคงเป็นปัญหาใหญ่สำหรับองค์กรขนาดใหญ่ ซึ่งรูปแบบการควบคุมการเข้าถึงไม่ได้บระบุถึงวิธีการแก้ปัญหาในประเด็นนี้ อย่างไรก็ตาม งานวิจัยบางชิ้นประสบความสำเร็จในการแก้ปัญหา โดยการค้นหาข้อมูลเพื่อนำไปพัฒนาองค์ประกอบของการควบคุมการเข้าถึงแบบโรลเบส

วิทยานิพนธ์ฉบับนี้ นำเสนอเนื้อหารูปแบบการควบคุมการเข้าถึงแบบโรลเบสที่มีคุณสมบัติเพิ่มขึ้นสำหรับองค์กรขนาดใหญ่ โดยรูปแบบใหม่นี้ประกอบด้วยคุณสมบัติที่เป็นข้อดีของรูปแบบการควบคุมการเข้าถึงแบบแมนดาทอรีและแบบดีสครีท สำหรับกำหนดโครงสร้างของบทบาทและที่ตั้ง รวมถึงการกำหนดใบอนุญาตให้กับบทบาทอย่างเหมาะสม ตลอดจนการกำหนดบทบาทให้กับผู้ใช้และที่ตั้งอย่างเหมาะสม อีกทั้งมีการประยุกต์ใช้วิธีการลำดับชั้นของที่ตั้งร่วมกับรูปแบบใหม่สำหรับสภาวะแวดล้อมแบบกระจาย รูปแบบใหม่นี้ใช้หลักการแบ่งแยก หน้าที่แบบสแตติก เพื่อความต้องการคงสภาพความปลอดภัยของระบบ และเรียกรูปแบบใหม่นี้ว่า อีอาร์แบคศูนย์สาม (ERBAC03) ซึ่งง่ายในการจัดการการควบคุมการเข้าถึงข้อมูลสารสนเทศและเป็นประโยชน์อย่างยิ่งสำหรับองค์กรขนาดใหญ่ในอนาคต

Thesis Title	An Enhanced Role-Based Access Control Model for Large Enterprises using Static Separation of Duty Concept
Student	Mr. Burin Yenmunkong
Student ID.	44067032
Degree	Master of Science
Programme	Information Technology
Year	2004
Thesis Advisor	Asst.Prof. Dr.Chanboon Sathitwiriyawong

ABSTRACT

Access control is one of the most important security issues for large organizations. Role-based access control (RBAC) has recently received considerable attention as a promising alternative to traditional discretionary and mandatory access control. Though RBAC is a good model, the administration of RBAC including building and maintaining access control information remains a difficult problem in large enterprises. RBAC model itself dose not tell the solution. Some researches were done on practical ways to find the information that fills RBAC components.

This thesis proposes an extended model of RBAC for large enterprises. A new model consists of qualified mandatory and discretionary features for roles and locations, including the assignment of permissions for the appropriate roles and the assignment of roles for the appropriate users and locations. Moreover, location hierarchies approach was applied to the new model for distributed environment. Static separation of duty principles is also used for integrity requirements of security system. This model is called "Enhanced Role-Based Access Control (ERBAC03)". It is easy to manage access control information and makes benefit for large enterprises in the future.

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงได้ด้วยดีด้วยคำแนะนำและคำปรึกษาเกี่ยวกับการกำหนดรูปแบบการควบคุมการเข้าถึงแบบโรลเบสสำหรับองค์กรขนาดใหญ่โดยใช้หลักการแบ่งแยกหน้าที่แบบสแตติกจาก ผศ.ดร.จันทร์บูรณ์ สถิตวิริยวงศ์ ซึ่งเป็นอาจารย์ผู้ควบคุมวิทยานิพนธ์ ผู้วิจัยรู้สึกซาบซึ้งในความอนุเคราะห์จากท่านและขอกราบขอบพระคุณเป็นอย่างสูง

ขอขอบพระคุณครอบครัวผู้วิจัยที่ให้การสนับสนุนการศึกษาและเป็นกำลังใจในการศึกษาให้ผ่านไปได้ด้วยดี

ขอขอบคุณ อารดี โรจนภาสกร ที่ให้คำปรึกษาเป็นอย่างดี โดยทำให้งานวิจัยฉบับนี้มีเนื้อหาที่สมบูรณ์ยิ่งขึ้น

ขอขอบคุณหัวหน้างานที่บริษัท ไปรษณีย์ไทย จำกัด ที่ให้การส่งเสริมการวิจัยในครั้งนี้ให้เป็นไปอย่างราบรื่น

สุดท้ายนี้ผู้วิจัยหวังว่าวิทยานิพนธ์ฉบับนี้มีคุณค่าและประโยชน์ต่อสถาบันหรือองค์กรใดๆ ที่ได้นำไปประยุกต์ใช้ให้เกิดประสิทธิผล

บุรินทร์ เย็นมั่นคง

สารบัญ

หน้า

บทคัดย่อภาษาไทย..... I

บทคัดย่อภาษาอังกฤษ.....II

กิตติกรรมประกาศ.....III

สารบัญ.....IV

สารบัญตาราง.....VIII

สารบัญรูป.....IX

บทที่ 1 บทนำ.....1

1.1 ความเป็นมาและความสำคัญของปัญหา.....1

1.2 ประเด็นปัญหา.....3

1.2.1 การออกแบบรูปแบบการควบคุมการเข้าถึงแบบโรลเบส (RBAC) เหมาะสมกับองค์กรขนาดใหญ่ที่มีการจัดการการควบคุมการเข้าถึงแบบกระจายทำได้อย่างไร.....3

1.2.2 รูปแบบการควบคุมการเข้าถึงแบบโรลเบส (RBAC) มีการจัดการข้อมูล (Data Management) อย่างไร.....4

1.2.3 รูปแบบการควบคุมการเข้าถึงแบบโรลเบส (RBAC) มีการกำหนดใบอนุญาตให้กับบทบาทอย่างไร.....6

1.2.4 รูปแบบการควบคุมการเข้าถึงแบบโรลเบสในลักษณะสภาพแวดล้อมแบบ Workflow มีการระบุคุณสมบัติข้อจำกัดอย่างไร.....6

1.3 ความมุ่งหมายและวัตถุประสงค์ของงานวิจัย.....6

1.4 แผนการดำเนินงานวิจัย.....7

1.4.1 ขั้นตอนการดำเนินงานวิจัย.....7

1.4.2 ระยะเวลาที่ใช้ในแต่ละขั้นตอน.....7

1.5 ขอบเขตของงานวิจัย.....9

1.6 ประโยชน์ที่คาดว่าจะได้รับ.....9

1.7 โครงสร้างของวิทยานิพนธ์.....10

สารบัญ (ต่อ)

หน้า

บทที่ 2 งานวิจัยที่เกี่ยวข้องและทฤษฎี.....11
2.1 งานวิจัยที่เกี่ยวข้อง.....11
2.2 ทฤษฎี.....13
2.2.1 รูปแบบการควบคุมการเข้าถึงแบบโรลเบส.....13
2.2.2 การกำหนดโครงสร้างลำดับชั้นแบบไฮเปอร์โหนด (Hyper node hierarchy).....24
2.2.3 หลักการแบ่งแยกหน้าที่ (Separation of Duty).....25

บทที่ 3 รูปแบบการควบคุมการเข้าถึง ERBAC03.....29
3.1 การนิยามรูปแบบ ERBAC03.....30
3.2 การกำหนดความสัมพันธ์ระหว่างเอนทิที.....33
3.2.1 ผู้ใช้กับบทบาท.....33
3.2.2 ที่ตั้งกับบทบาท.....33
3.2.3 บทบาทกับงาน.....34
3.2.4 งานกับงานย่อย.....34
3.2.5 งานย่อยกับใบอนุญาต.....34
3.3 วิธีการลำดับชั้นที่ตั้ง (Location hierarchy approach).....34
3.4 การกำหนดคุณสมบัติข้อบังคับ (Constraint).....36
3.5 การกำหนดความต้องการความคงสภาพของรูปแบบ ERBAC03.....41
3.6 หลักการอื่นๆ ที่เกี่ยวข้อง.....44
3.6.1 การให้สิทธิ์ (Delegation).....44
3.6.2 การยกเลิกสิทธิ์ (Revocation).....44

บทที่ 4 การออกแบบอัลกอริทึมสำหรับรูปแบบ ERBAC03.....45
4.1 อัลกอริทึมสำหรับรูปแบบ ERBAC03.....45
4.2 อัลกอริทึมสำหรับความสัมพันธ์ของเอนทิที.....46
4.2.1 การเพิ่มความสัมพันธ์.....47

สารบัญ (ต่อ)

หน้า

4.2.2 การลบความสัมพันธ์.....52

4.2.3 การบำรุงรักษาลำดับชั้นของบทบาทและที่ตั้ง.....52

4.3 อัลกอริทึมสำหรับเอนทิทีที่ขัดแย้ง.....55

4.3.1 การเพิ่มเอนทิทีที่ขัดแย้ง.....55

4.3.2 การลบเอนทิทีที่ขัดแย้ง.....63

4.4 อัลกอริทึมสำหรับจัดการเอนทิที.....66

4.4.1 การเพิ่มเอนทิที.....66

4.4.2 การลบเอนทิที.....66

บทที่ 5 การพัฒนาโปรแกรมสำหรับรูปแบบ ERBAC03.....68

5.1 การพัฒนาโปรแกรมแบบทริกเกอร์ในฐานข้อมูล (Database trigger).....68

5.1.1 วิธีการออกแบบทริกเกอร์.....69

5.1.2 หลักการทำงาน.....69

5.2 การพัฒนาโปรแกรมแบบกราฟิกสำหรับการบริหารจัดการระบบฯ.....85

บทที่ 6 การทดลองและผลการทดลอง.....88

6.1 การออกแบบการทดลอง.....88

6.1.1 วิธีการทดลองสมมุติฐานการลดความถี่ในการบุกรุก.....90

6.1.2 วิธีการทดลองสมมุติฐานการเพิ่มความถูกต้องของข้อมูล.....93

6.1.3 วิธีการทดลองการหาเวลาการทำงาน.....94

6.2 ผลการทดลอง.....100

6.2.1 ผลการทดลองสมมุติฐานการลดความถี่ในการบุกรุก.....100

6.2.2 ผลการทดลองสมมุติฐานการเพิ่มความถูกต้องของข้อมูล.....101

6.2.3 ผลการทดลองการหาเวลาการทำงาน.....103

บทที่ 7 สรุปผลการวิจัยและข้อเสนอแนะ.....104

สารบัญ (ต่อ)

หน้า

เอกสารอ้างอิง.....107

ภาคผนวก ก. 112

ผลงานวิจัยที่ได้รับการตีพิมพ์.....113

ภาคผนวก ข. 126

ตัวอย่างโปรแกรมทริกเกอร์.....127

ประวัติผู้เขียน.....150

สารบัญตาราง

ตารางที่ หน้า

1.1 แสดงตัวอย่างตารางบทบาท....4

1.2 แสดงตัวอย่างตารางบทบาท....5

1.3 แสดงตัวอย่างตารางที่ตั้ง....5

1.4 แสดงตัวอย่างตารางการกำหนดบทบาทให้กับที่ตั้ง....5

2.1 แสดงหลักการแบ่งแยกหน้าที่แบบสแตติก....27

3.1 แสดงความแตกต่างระหว่าง $RBAC_0$ กับ $ERBAC03_0$....31

3.2 แสดงตัวอย่างประเภทที่ตั้งของไปรษณีย์ไทย....35

4.1 แสดงถึงโอเปอร์เรชั่นที่แสดงถึงความสัมพันธ์ ความขัดแย้งและเอนทิที....46

5.1 แสดงตัวอย่างตารางผู้ใช้....70

5.2 แสดงตัวอย่างตารางที่ตั้ง....70

5.3 แสดงตัวอย่างตารางบทบาท....70

5.4 แสดงตัวอย่างตารางงาน....70

5.5 แสดงตัวอย่างตารางงานย่อย....71

5.6 แสดงตัวอย่างตารางใบอนุญาต....71

5.7 แสดงการสรุปเหตุการณ์ที่มีในรูปแบบ ERBAC03....71

5.8 แสดงตัวอย่างการกำหนดความสัมพันธ์ระหว่างผู้ใช้กับบทบาท....80

5.9 แสดงตัวอย่างการกำหนดความสัมพันธ์ระหว่างที่ตั้งกับบทบาท....81

5.10 แสดงตัวอย่างการกำหนดความสัมพันธ์ระหว่างงานกับบทบาท....82

5.11 แสดงตัวอย่างการกำหนดความสัมพันธ์ระหว่างงานกับงานย่อย....83

5.12 แสดงตัวอย่างการกำหนดความสัมพันธ์ระหว่างงานย่อยกับใบอนุญาต....84

6.1 แสดงข้อสมมุติฐานทั้งหมดที่จะนำมาทดลอง....89

6.2 แสดงตัวอย่างข้อมูลการล็อกอินและล็อกเอาท์ของผู้ใช้ในระบบควบคุมส่วนกลาง โดยใช้รูปแบบRBAC....91

6.3 ผลการทดลองการหาเวลาการทำงานเฉลี่ยหนึ่งชุดคำสั่งต่อหนึ่งผู้ใช้....103

สารบัญรูป

รูปที่ หน้า

2.1 แสดงความสัมพันธ์ในรูปแบบ RBAC.....14

2.2 แสดงรูปแบบ $RBAC_0$.....14

2.3 แสดงรูปแบบ $RBAC_1$.....17

2.4 แสดงตัวอย่างลำดับชั้นบทบาท (Example of Role Hierarchies).....18

2.5 แสดงการสืบทอดใบอนุญาตแบบมัลติเพิล (Multiple Inheritance of Permissions).....18

2.6 แสดงตัวอย่างลำดับชั้นบทบาทแบบส่วนตัว.....19

2.7 แสดงรูปแบบ $RBAC_2$.....20

2.8 แสดงรูปแบบ $RBAC_3$.....23

2.9 ลำดับชั้นแบบไฮเปอร์โหนด (Hyper Node Hierarchy).....24

2.10 ประเภทของหลักการแบ่งแยกหน้าที่.....26

3.1 แสดงการแบ่งส่วนการควบคุมการเข้าถึง.....29

3.2 รูปแบบ ERBAC03 ระดับศูนย์ ($ERBAC03_0$).....30

3.3 รูปแบบ ERBAC03 ระดับหนึ่ง ($ERBAC03_1$).....35

3.4 ลำดับชั้นที่ตั้ง (Location hierarchy).....36

3.5 รูปแบบ ERBAC03 ระดับสอง ($ERBAC03_2$).....37

3.6 รูปแบบ ERBAC03 ระดับสาม ($ERBAC03_3$).....40

4.1 แสดงอีอาร์ไดอะแกรมของรูปแบบ ERBAC03.....45

4.2 แสดงอัลกอริทึมการเพิ่มความสัมพันธ์ระหว่างผู้ใช้กับบทบาท.....47

4.3 แสดงอัลกอริทึมการเพิ่มความสัมพันธ์ระหว่างที่ตั้งกับบทบาท หรืองานกับบทบาท.....49

4.4 แสดงอัลกอริทึมการเพิ่มความสัมพันธ์ระหว่างงานกับงานย่อย.....50

4.5 แสดงอัลกอริทึมการเพิ่มความสัมพันธ์ระหว่างงานย่อยกับใบอนุญาต.....51

4.6 แสดงอัลกอริทึมการเพิ่มบทบาทหรือที่ตั้งให้กับลำดับชั้นของบทบาทหรือที่ตั้ง.....53

4.7 แสดงอัลกอริทึมการลบความสัมพันธ์ของบทบาทหรือที่ตั้งจากลำดับชั้นของบทบาท หรือที่ตั้ง.....54

4.8 แสดงอัลกอริทึมการสร้างความขัดแย้งของเอนทิที.....56

สารบัญรูป (ต่อ)

รูปที่ หน้า

4.9 แสดงอัลกอริทึมการตรวจสอบความสัมพันธ์ระหว่างผู้ใช้กับบทบาท.....57

4.10 แสดงอัลกอริทึมการตรวจสอบความสัมพันธ์ระหว่างงานกับบทบาทและงานกับงานย่อย...58

4.11 แสดงอัลกอริทึมการตรวจสอบความสัมพันธ์ระหว่างงานย่อยกับงานและงานย่อยกับ ใบอนุญาต.....59

4.12 แสดงอัลกอริทึมการตรวจสอบความสัมพันธ์ระหว่างใบอนุญาตกับงานย่อย.....60

4.13 แสดงอัลกอริทึมการตรวจสอบความสัมพันธ์ระหว่างผู้ใช้กับบทบาท และบทบาทกับที่ตั้ง...61

4.14 แสดงอัลกอริทึมการตรวจสอบความสัมพันธ์ระหว่างบทบาทกับที่ตั้ง.....62

4.15 แสดงอัลกอริทึมการตรวจสอบความสัมพันธ์สำหรับการลบบทบาท.....63

4.16 แสดงอัลกอริทึมการตรวจสอบความสัมพันธ์สำหรับการลบงาน.....64

4.17 แสดงอัลกอริทึมการตรวจสอบความสัมพันธ์สำหรับการลบงานย่อย.....65

4.18 แสดงอัลกอริทึมการลบผู้ใช้ ที่ตั้ง บทบาท งาน งานย่อย และใบอนุญาต.....67

5.1 แสดงการกำหนดรูปแบบทริกเกอร์.....69

5.2 แสดงหน้าจอหลักของโปรแกรมบริหารจัดการสำหรับรูปแบบ ERBAC03.....85

5.3 แสดงตัวอย่างหน้าจอการเพิ่มผู้ใช้ใหม่สำหรับรูปแบบ ERBAC03.....86

5.4 แสดงตัวอย่างหน้าจอการเพิ่มที่ตั้งให้กับลำดับชั้นของที่ตั้งสำหรับรูปแบบ ERBAC03.....86

5.5 แสดงตัวอย่างหน้าจอการกำหนดความขัดแย้งของที่ตั้งสำหรับรูปแบบ ERBAC03.....86

5.6 แสดงตัวอย่างหน้าจอการกำหนดความสัมพันธ์ระหว่างบทบาทกับใบอนุญาต สำหรับรูปแบบ RBAC.....87

5.7 แสดงตัวอย่างหน้าจอการเรียกดูผู้ใช้ในระบบฯสำหรับรูปแบบ ERBAC03.....87

6.1 แสดงตัวอย่างการกำหนดคำสั่งออดิท.....91

6.2 แสดงขั้นตอนการตรวจสอบการล็อกอินเข้าสู่ระบบฯ.....93

6.3 แสดงตัวอย่างการตั้งเวลาการทำงาน.....95

6.4 แสดงผลการทดลองการกำหนดบทบาทให้กับผู้ใช้ด้วยรูปแบบ RBAC ของ Perelson [39].....101

6.5 แสดงผลการทดลองการกำหนดบทบาทให้กับผู้ใช้ด้วยรูปแบบ ERBAC03.....102

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

เนื่องจากระบบคอมพิวเตอร์ในปัจจุบันถูกบุกรุกจากผู้ประสงค์ร้ายเพิ่มขึ้น จึงจำเป็นต้องมีระบบรักษาความปลอดภัยของระบบคอมพิวเตอร์ที่มีประสิทธิภาพที่สามารถป้องกันการกระทำอันมิชอบของผู้บุกรุก โดยวัตถุประสงค์หลักในการรักษาความปลอดภัยของระบบคอมพิวเตอร์คือ รักษาความลับของข้อมูลในระบบ (Confidentiality) รักษาความคงสภาพของระบบ (Integrity) และรักษาการมีอยู่ของระบบ (Availability) โดยในงานวิจัยนี้ได้ให้ความสำคัญในเรื่องของการรักษาความลับของข้อมูลในระบบกับการรักษาความคงสภาพของระบบมากที่สุด จึงจำเป็นต้องมีการควบคุมการเข้าถึงของผู้ที่มีสิทธิ์ใช้ระบบคอมพิวเตอร์ โดยในปี พ.ศ. 2514 แลมสัน (Lampson) [1] ได้เสนอรูปแบบตารางการเข้าถึง (Access Matrix Model) ซึ่งอยู่บนพื้นฐานการกำหนดกฎต่างๆ ที่ใช้ในการควบคุมสิทธิ์การเข้าถึงทรัพยากร โดยระบุให้เจ้าของออปเจ็คท์มีสิทธิ์โดยสมบูรณ์ในการจัดการกับออปเจ็คท์นั้น

ในปี พ.ศ. 2515 เกรแฮม (Graham) และเดนนิ่ง (Denning) [2] ได้เสนอกฎที่แตกต่างกัน โดยให้เจ้าของออปเจ็คท์มีอำนาจในการมอบสิทธิ์ของตัวเองให้กับซับเจ็คท์ได้

ในปี พ.ศ. 2519 แฮริสัน (Harrison) รุซโซ (Ruzzo) และอูแมน (Ullman) [3] ได้พัฒนารูปแบบแฮชอาร์ยู (HRU) ขึ้นมา โดยรูปแบบนี้ไม่ได้เชื่อมโยงกับกฎที่ระบุถึงสิทธิ์การเข้าถึง แต่แฮชอาร์ยูเป็นรูปแบบที่จัดเตรียมไว้สำหรับผู้ออกแบบนโยบายด้านความปลอดภัย โดยสามารถระบุถึงกฎที่เหมาะสมในการนำนโยบายไปใช้งานได้ตามความพอใจ รูปแบบแฮชอาร์ยูง่ายต่อการจัดหานโยบายที่หลากหลาย แต่มีข้อด้อยในเรื่องความปลอดภัยอยู่มาก เพราะควรมีการกำหนดนโยบายลงไปให้ชัดเจน โดยอาจไม่ให้ซับเจ็คท์เข้าถึงออปเจ็คท์ได้

ในปี พ.ศ. 2520 ลิปตัน (Lipton) และสนีเดอร์ (Snyder) [4] ได้พัฒนารูปแบบแบบเทคแกรน (Take-grant model) โดยอธิบายไว้ว่า รูปแบบนี้สามารถขจัดข้อด้อยในเรื่องความปลอดภัยของ HRU Model ซึ่งช่องว่างระหว่างรูปแบบแฮชอาร์ยูกับรูปแบบเทคแกรนนี้ถูกพัฒนาต่อให้มีความสมบูรณ์ขึ้นโดย ซานฮู (Sandhu) [5] ในปี พ.ศ. 2531 และใช้ชื่อรูปแบบว่า สคีมาตีกโปรเทคชั่น (Schematic Protection Model) ซึ่งมีคุณสมบัติเด่นในด้านความปลอดภัยมาก กล่าวคือ จะอนุญาตให้ซิงเกิลพาเรนต์ (Single parent) สร้างการปฏิบัติงานได้

ในปี พ.ศ. 2535 มีการขยายขอบเขตความสามารถของรูปแบบสคีมาตีกโปรเทคชั่น ซึ่งมีประสิทธิภาพเท่ากับรูปแบบแฮชอาร์ยูแบบโมโนโทนิค (Monotomic HRU) และมีความปลอดภัย

ไม่ด้อยไปกว่ารูปแบบสคีมาตีกโปรเทคชั่น เรียกว่า รูปแบบอีเอสพีเอ็ม (Extended SPM: ESPM) [6] โดยทั้งรูปแบบสคีมาตีกโปรเทคชั่นและรูปแบบอีเอสพีเอ็ม ถูกปรับแต่งคุณสมบัติด้านความปลอดภัยให้เหมาะสมและเปลี่ยนชื่อใหม่ว่า ไทพ์แอคเซสเมทริค (Type Access Matrix:TAM) [7] และออกเมินท์ไทพ์แอคเซสเมทริค (Augmented Typed Access Matrix: ATAM) [8] โดยไทพ์แอคเซสเมทริคถูกกำหนดโดยข้อได้เปรียบที่มีอยู่ในรูปแบบแฮซอาร์ยู ซึ่งข้อได้เปรียบนี้ หมายถึงแต่ละซับเจ็คท์และออปเจ็คท์ที่ถูกสร้างขึ้นมาโดยเฉพาะประเภทซับเจ็คท์และออปเจ็คท์นั้น จะไม่สามารถเปลี่ยนแปลงเป็นประเภทอื่นได้อีก ออกเมินท์ไทพ์แอคเซสเมทริคจะคล้ายๆ กับไทพ์แอคเซสเมทริค แต่ออกเมินท์ไทพ์แอคเซสเมทริคจะมีความสามารถเพิ่มขึ้น คือ สามารถตรวจสอบการไม่ได้รับสิทธิ์ให้เข้าถึงได้ [9][10]

ในปี พ.ศ. 2537 ซานฮู (Sandhu) และซามาราติ (Samarati) [11] ได้อธิบายถึงรูปแบบการควบคุมการเข้าถึงแบบดีสครีท (Discretionary access control: DAC) และรูปแบบการควบคุมการเข้าถึงแบบแมนดาทอรี (Mandatory access control: MAC) ซึ่งรูปแบบการควบคุมการเข้าถึงแบบดีสครีท ใช้หลักการของสิทธิ์ความเป็นเจ้าของข้อมูล ส่วนรูปแบบการควบคุมการเข้าถึงแบบแมนดาทอรี ใช้วิธีการติดป้ายความสำคัญให้กับทุกซับเจ็คท์และออปเจ็คท์ที่มีอยู่ในระบบทั้งหมด

ในปี พ.ศ. 2539 ซานฮู (Sandhu) [12][13][14] ได้พัฒนารูปแบบการควบคุมการเข้าถึงแบบโรลเบส (Role-based access control: RBAC) ความมุ่งหมายของรูปแบบการควบคุมการเข้าถึงแบบโรลเบส คือ การระบุสิทธิ์ที่ได้รับให้เหมาะสมกับบทบาทที่ได้ทำของแต่ละคน ซึ่งต้องกำหนดบทบาทให้เหมาะสมกับผู้ใช้งานด้วย เพื่อให้จัดการเกี่ยวกับใบอนุญาตง่ายขึ้น แต่ก็ไม่ได้ให้รายละเอียดเกี่ยวกับการระบุวิธีการกำหนดบทบาทให้กับผู้ใช้และการกำหนดใบอนุญาตให้กับบทบาทไว้อย่างชัดเจน จึงทำให้เกิดปัญหาคือ ผู้บริหารจัดการระบบไม่สามารถทำการกำหนดบทบาทให้กับผู้ใช้ได้อย่างถูกต้องตรงตามหน้าที่ที่ผู้ใช้ต้องปฏิบัติงานจริงๆ ในองค์กร และทำให้ระบบคอมพิวเตอร์ในองค์กรเกิดความไม่ปลอดภัยขึ้น ยกตัวอย่างเช่น นาย ก. ถูกกำหนดให้ได้รับบทบาทเป็นทั้งผู้จัดการฝ่ายจัดซื้อและผู้ตรวจสอบบัญชีการจัดซื้อ จะทำให้นาย ก. สามารถทุจริตต่อองค์กรได้ ทำให้องค์กรเสียผลประโยชน์เป็นอย่างมาก ต่อมาได้มีการพัฒนารูปแบบการควบคุมการเข้าถึงแบบโรลเบสโดยใช้เทคนิคต่างๆ มาประยุกต์ใช้ร่วมกับรูปแบบการควบคุมการเข้าถึงแบบโรลเบส โดยแต่ละวิธีของงานวิจัยที่ผ่านมามีทั้งข้อดีข้อเสียและข้อจำกัดที่แตกต่างกัน ซึ่งสามารถนำมาใช้เป็นแนวทางในการพัฒนารูปแบบการควบคุมการเข้าถึงแบบโรลเบสต่อไปในอนาคตได้

ดังนั้น ในงานวิจัยนี้จึงมุ่งเน้นในการพัฒนารูปแบบการควบคุมการเข้าถึงแบบโรลเบส โดยจะมุ่งประเด็นไปที่การระบุรายละเอียดของการกำหนดใบอนุญาตให้กับบทบาทได้อย่างเหมาะสมและการกำหนดบทบาทให้กับผู้ใช้ ซึ่งจะขึ้นอยู่กับที่ตั้งที่ผู้ใช้ปฏิบัติบทบาทนั้นๆ ด้วย โดยรูปแบบ

การควบคุมการเข้าถึงแบบใหม่นี้จะทำให้ผู้บริหารจัดการระบบคอมพิวเตอร์เกิดความชัดเจนเพิ่มมากขึ้นในการกำหนดบทบาทให้กับผู้ใช้และการกำหนดใบอนุญาตให้กับบทบาท จึงทำให้ลดความผิดพลาดในการปฏิบัติงานลงอย่างมาก ซึ่งสามารถทดสอบโดยการเขียนโปรแกรมเปรียบเทียบทั้งสองรูปแบบระหว่างรูปแบบเดิมกับรูปแบบใหม่ อีกทั้งยังสามารถนำโดยรูปแบบการควบคุมการเข้าถึงแบบใหม่ไปประยุกต์ใช้กับองค์กรในปัจจุบันได้อย่างเหมาะสมเป็นอย่างมาก เนื่องจากโดยรูปแบบการควบคุมการเข้าถึงแบบใหม่ออกแบบอยู่บนพื้นฐานของที่ตั้ง ซึ่งองค์กรขนาดใหญ่ในปัจจุบันจะมีลักษณะสภาวะแวดล้อมแบบกระจาย [42] ซึ่งมีสาขาอยู่มากมายที่กระจายอยู่ตามสถานที่ต่างๆ ที่แตกต่างกันไป

1.2 ประเด็นปัญหา

ปัญหาของรูปแบบ RBAC ที่พบนี้ได้นำมาสู่การวิจัยฉบับนี้ ซึ่งปัญหาเหล่านี้ได้แสดงในรูปแบบของคำถาม

1.2.1 การออกแบบรูปแบบการควบคุมการเข้าถึงแบบโรลเบส (RBAC) เหมาะสมกับองค์กรขนาดใหญ่ที่มีการจัดการการควบคุมการเข้าถึงแบบกระจายทำได้อย่างไร

RBAC จะมีความเหมาะสมได้ต้องออกแบบถึงระดับที่ 2 ซึ่งก็คือมีการกำหนดข้อบังคับ (Constraint) โดยข้อบังคับนี้จะเป็นกฎที่ใช้ควบคุมการใช้สิทธิ์ของผู้ใช้ไม่ให้ละเมิดสิทธิ์ที่ตัวเองมีอยู่ ซึ่งการกำหนดข้อบังคับให้คลอบคลุมนโยบายขององค์กรทั้งหมดนั้นมีความยุ่งยากและซับซ้อนอย่างมาก ยกตัวอย่าง RBAC ระดับ 0 เช่น

กำหนดให้

(x, Chief Post office) $\in$UA ผู้ใช้ x ได้รับบทบาทเป็นหัวหน้าไปรษณีย์

(Chief Post office, Closing the End of Day) $\in$PA บทบาท Chief Post Office มีใบอนุญาตในการปิดบัญชีสิ้นวัน

แสดงว่าผู้ใช้ใดก็ตามที่มีบทบาท Chief Post Office จะสามารถปิดบัญชีได้ ซึ่งถ้าผู้ใช้เป็นหัวหน้าไปรษณีย์ที่จังหวัดกรุงเทพฯ ก็สามารถที่จะปิดบัญชีที่สาขาอื่นๆได้ ซึ่งในความเป็นจริงแล้วการกระทำลักษณะนี้ไม่สามารถกระทำได้จนกว่าจะได้รับมอบหมายจากผู้มีอำนาจระดับที่สูงกว่า

วิธีแก้ปัญหาลักษณะนี้สามารถกระทำได้โดยการเพิ่มคุณสมบัติของข้อบังคับ ซึ่งจะทำให้ RBAC ระดับ 0 เปลี่ยนเป็น RBAC ระดับ 2 ทันที โดยอาจเพิ่ม attribute ที่เกี่ยวข้องกับที่ตั้งให้กับองค์ประกอบของผู้ใช้ (Users) หรือ บทบาท (Roles) ขึ้นอยู่กับการออกแบบของแต่ละองค์กร เช่น

ตารางที่ 1.1 แสดงตัวอย่างตารางบทบาท

ROLE_ID	ROLE_NAME	LOCATION
1	Chief Post Office	Bangkok
2	Cashier	Bangkok
3	Accountant	Trad
4	Chief Post Office	Krabi

จากตัวอย่างตารางสามารถกำหนดกฎการควบคุมการเข้าถึงได้ดังต่อไปนี้

Bangkok(Chief_Post_Office) $\wedge$ Chief_Post_Office(BOB) $\rightarrow$ Closing_EOD(BOB, Chief_Post_Office)

หมายความว่า BOB ได้รับบทบาท Chief_Post_Office สามารถปิดบัญชีสิ้นวันที่สาขากรุงเทพฯได้

1.2.2 รูปแบบการควบคุมการเข้าถึงแบบโรลเบส (RBAC) มีการจัดการข้อมูล (Data Management) อย่างไร

จะเห็นได้ว่าเมื่อมีการกำหนด attribute ให้กับตารางบทบาทแล้วจะทำให้เกิดปัญหาของความซ้ำซ้อนของข้อมูลเมื่อมีการปรับปรุงแก้ไข เพิ่มเติม ลบข้อมูลทิ้ง จึงจำเป็นต้องมีกระบวนการนอร์มัลไลเซชัน เพื่อลดความซ้ำซ้อนของข้อมูล [43]

- **ผลเสียของปัญหาการแก้ไขข้อมูล**
  - สิ้นเปลืองเวลาในการแก้ไขข้อมูล
  - ไม่สามารถรับประกันได้ว่าจะมีการแก้ไขข้อมูลครบถ้วนถูกต้องทุกรายการ
  - ข้อมูลที่แก้ไขอาจจะไม่สอดคล้องกันหรือไม่ถูกต้องทุกแห่ง ซึ่งตามหลักการฐานข้อมูลการแก้ไขข้อมูลควรจะกระทำเพียงแห่งเดียว
- **ผลเสียของปัญหาการเพิ่มข้อมูล**
  - การเพิ่มข้อมูลใหม่จะทำไม่ได้ทันที จนกว่าจะมีค่าข้อมูลคีย์หลัก (Primary Key) ของตารางนั้นเกิดขึ้นเสียก่อน
- **ผลเสียของปัญหาการลบข้อมูล**
  - การลบข้อมูลออกจากตารางอาจจะทำให้สูญเสียข้อมูลที่จะใช้ทำงานหรืออ้างอิงในอนาคตได้

จากผลเสียที่กล่าวข้างต้นจะทำให้การจัดการข้อมูลขาดประสิทธิภาพ และมีผลทำให้การจัดการด้านความปลอดภัยขาดประสิทธิภาพไปด้วย จึงทำให้เจ้าหน้าที่บริหารจัดการด้านความปลอดภัยของระบบฯ ทำงานได้อย่างลำบากมากขึ้น

วิธีแก้ปัญหาลักษณะนี้สามารถกระทำได้โดยการแตกตารางบทบาทโดยแยก attribute ที่ตั้งออกเป็นอีกตารางหนึ่ง เช่น

ตารางที่ 1.2 แสดงตัวอย่างตารางบทบาท

ROLE_ID	ROLE_NAME
1	Chief Post Office
2	Cashier
3	Accountant
4	Chief Post Office

ตารางที่ 1.3 แสดงตัวอย่างตารางที่ตั้ง

LOCATION_ID	LOCATION_NAME
1	Bangkok
2	Trad
3	Krabi

ตารางที่ 1.4 แสดงตัวอย่างตารางการกำหนดบทบาทให้กับที่ตั้ง

ROLE_ID	LOCATION_ID
1	1
2	1
3	2
4	3

เมื่อตารางทั้งหมดมีคุณสมบัตินอร์มัลไลเซชันที่ 5 จะทำให้ความซ้ำซ้อนของข้อมูลหมดไปทันที

1.2.3 **รูปแบบการควบคุมการเข้าถึงแบบโรลเบส (RBAC) มีการกำหนดใบอนุญาตให้กับบทบาทอย่างไร**

เนื่องจาก RBAC เป็นรูปแบบที่มีความง่ายและไม่ซับซ้อนในการออกแบบ ซึ่งในงานวิจัยนี้จะพิจารณา RBAC ระดับ 0 เป็นหลัก ซึ่งความง่ายและไม่ซับซ้อนนี้เองทำให้รูปแบบ RBAC ไม่ได้ให้รายละเอียดในการกำหนดใบอนุญาตให้กับบทบาท และไม่สามารถบอกได้ว่าใบอนุญาตที่ได้รับนั้นมีไว้สำหรับปฏิบัติงานอะไรของบทบาทนั้นๆ มีผลทำให้เจ้าหน้าที่บริหารจัดการด้านความปลอดภัยของระบบฯ มีความเข้าใจที่คลาดเคลื่อนในการกำหนดใบอนุญาตให้กับบทบาท อาจทำให้เกิดความผิดพลาดในการกำหนดได้ ซึ่งทำให้ระบบฯถูกผู้ไม่หวังดีบุกรุกได้

เนื่องจากในปัจจุบันการดำเนินการปฏิบัติงานในองค์กรขนาดใหญ่มีลักษณะสภาพแวดล้อมเป็นแบบ Workflow process [44] จึงทำให้ต้องมีการกำหนดรายละเอียดในการกำหนดใบอนุญาตให้กับบทบาทให้ครอบคลุมกับลักษณะการดำเนินงานที่เป็นแบบ Workflow process การกำหนดรายละเอียดนี้จะใช้วิธีการจำแนกบทบาทออกเป็นงานและงานย่อยเพื่อให้ได้ใบอนุญาตที่ โดยการกระทำวิธีนี้จะทำให้การนำ RBAC ไปประยุกต์ใช้ร่วมกับองค์กรมีประสิทธิภาพมากขึ้น

1.2.4 **รูปแบบการควบคุมการเข้าถึงแบบโรลเบสในลักษณะสภาพแวดล้อมแบบ Workflow มีการระบุคุณสมบัติข้อบังคับอย่างไร**

ในรูปแบบการควบคุมการเข้าถึงแบบใหม่จำเป็นอย่างยิ่งที่ต้องมีการกำหนดคุณสมบัติข้อบังคับ เพื่อให้ระบบฯมีความปลอดภัยสอดคล้องกับนโยบายที่ได้กำหนดไว้ โดยเฉพาะอย่างยิ่งการกำหนดคุณสมบัติข้อบังคับควรกำหนดให้ครบกับทุกๆ องค์ประกอบและความสัมพันธ์ทั้งหมดของรูปแบบการควบคุมการเข้าถึงแบบใหม่

1.3 ความมุ่งหมายและวัตถุประสงค์ของงานวิจัย

1.3.1 เพื่อศึกษารูปแบบการควบคุมการเข้าถึงแบบโรลเบส

1.3.2 เพื่อศึกษาเทคนิคการกำหนดโครงสร้างของบทบาทและที่ตั้ง

1.3.3 เพื่อศึกษาวิธีการกำหนดใบอนุญาตให้กับบทบาท

1.3.4 เพื่อศึกษาเทคนิคของข้อบังคับที่ใช้กับรูปแบบการควบคุมการเข้าถึงแบบโรลเบส

1.3.5 เพื่อนำความรู้ที่ได้ศึกษามาทั้งหมด มาพัฒนารูปแบบการควบคุมการเข้าถึงแบบโรลเบสให้มีประสิทธิภาพมากขึ้นในด้านการกำหนดใบอนุญาตให้กับบทบาท และการกำหนดบทบาทให้กับผู้ใช้ โดยมีที่ตั้งของผู้ใช้เป็นฐาน

1.4 แผนการดำเนินงานวิจัย

1.4.1 ขั้นตอนการดำเนินงานวิจัย

1.4.1.1 ศึกษาบทความและผลงานวิจัยต่างๆ ที่เกี่ยวข้องกับงานวิจัยนี้

1.4.1.2 กำหนดหัวข้อ เป้าหมาย วัตถุประสงค์ และขอบเขตการทำวิทยานิพนธ์

1.4.1.3 ศึกษาทฤษฎีและหลักการที่เกี่ยวข้อง

1.4.1.4 ออกแบบรูปแบบการควบคุมการเข้าถึงแบบโรลเบส โดยใช้เทคนิคของการกำหนดโครงสร้างของบทบาทและที่ตั้ง

1.4.1.5 พัฒนาประสิทธิภาพของรูปแบบที่ออกแบบโดยการเพิ่มคุณสมบัติข้อบังคับ คือ หลักการแบ่งแยกหน้าที่แบบสแตติก (Static Separation of Duty)

1.4.1.6 พิสูจน์ความถูกต้องของรูปแบบที่ออกแบบในด้านความถูกต้องในการกำหนดใบอนุญาตให้กับบทบาทและการกำหนดบทบาทให้กับผู้ใช้ ที่อยู่บนพื้นฐานของที่ตั้ง

1.4.1.7 พัฒนาโปรแกรมเพื่อควบคุมการเข้าถึงตามรูปแบบเดิมกับรูปแบบที่ออกแบบใหม่

1.4.1.8 ทดลองการกำหนดใบอนุญาตให้กับบทบาทและการกำหนดบทบาทให้กับผู้ใช้จากโปรแกรมที่พัฒนาจากรูปแบบที่ออกแบบ

1.4.1.9 สรุปผลการดำเนินงานและรวบรวมจัดทำเอกสารเสนอเป็นงานวิจัย

1.4.2 ระยะเวลาที่ใช้ในแต่ละขั้นตอน

ขั้นตอนการทำงาน	ระยะเวลาที่ใช้ (เดือนที่)											
	1	2	3	4	5	6	7	8	9	10	11	12
1. ศึกษาบทความและผลงานวิจัยต่างๆ ที่เกี่ยวข้องกับงานวิจัยนี้												
2. กำหนดหัวข้อ เป้าหมาย วัตถุประสงค์ และขอบเขตการทำวิทยานิพนธ์												

ขั้นตอนการทำงาน	ระยะเวลาที่ใช้ (เดือนที่)											
	1	2	3	4	5	6	7	8	9	10	11	12
3. ศึกษาทฤษฎีและหลักการที่เกี่ยวข้อง												
4. ออกแบบรูปแบบการควบคุมการเข้าถึงแบบโรลเบส โดยใช้เทคนิคของการกำหนดโครงสร้างของบทบาทและที่ตั้ง												
5. พัฒนาประสิทธิภาพของรูปแบบที่ออกแบบโดยการเพิ่มคุณสมบัติข้อบังคับ												
6. พิสูจน์ความถูกต้องของรูปแบบที่ออกแบบ												
7. พัฒนาโปรแกรมเพื่อควบคุมการเข้าถึงตามรูปแบบเดิมกับรูปแบบที่ออกแบบใหม่												
8. ทดลองการกำหนดใบอนุญาตให้กับบทบาทและการกำหนดบทบาทให้กับผู้ใช้ จากโปรแกรมที่พัฒนาจากรูปแบบที่ออกแบบ												

ขั้นตอนการทำงาน	ระยะเวลาที่ใช้ (เดือนที่)											
	1	2	3	4	5	6	7	8	9	10	11	12
9. สรุปผลการดำเนินงานและรวบรวมจัดทำเอกสารเสนอเป็นงานวิจัย												

1.5 ขอบเขตของงานวิจัย

สำหรับงานของวิธีการออกแบบรูปแบบการควบคุมการเข้าถึงแบบโรลเบสนี้ประกอบด้วยงานหลักๆ อยู่ 4 อย่างคือ การระบุวิธีการกำหนดบทบาทให้กับผู้ใช้ การระบุวิธีการกำหนดใบอนุญาตให้กับ บทบาท การระบุวิธีการกำหนดโครงสร้างของบทบาท และการกำหนดข้อบังคับให้กับรูปแบบใหม่ ซึ่งในงานวิจัยส่วนนี้จะมุ่งประเด็นไปที่การกำหนดข้อบังคับให้กับรูปแบบใหม่ แต่ก็จะปรับปรุงงานอื่นๆ ที่มีอยู่ให้มีความสอดคล้องกัน ที่จะทำให้นำไปประยุกต์ใช้กับองค์กรในปัจจุบันได้เป็นอย่างดี และเครื่องมือที่จะใช้ในการพัฒนางานวิจัยมีดังนี้

1.5.1 ภาษาที่ใช้เขียนโปรแกรมนั้นใช้ภาษา JAVA และ SQL

1.5.2 ฐานข้อมูลที่ใช้ร่วมพัฒนาคือ ออราเคิล (Oracle 8.16)

1.5.3 เครื่องคอมพิวเตอร์ที่ใช้ในการทดสอบผลการทำงานเป็นเครื่อง AMD Athlon ความเร็ว 1.4 MHz หน่วยความจำ 512 MB

1.5.4 ข้อมูลที่ใช้ในการทดสอบเป็นข้อมูลที่ได้จากองค์กรที่ได้ปฏิบัติงานอยู่คือ บริษัท ไปรษณีย์ จำกัด ซึ่งเป็นองค์กรขนาดใหญ่ที่มีความเหมาะสมอย่างยิ่งต่อการนำมาเป็นองค์กรตัวอย่างในการทดลอง

1.6 ประโยชน์ที่คาดว่าจะได้รับ

1.6.1 ทำให้ได้ศึกษาถึงผลงานวิจัยที่เกี่ยวข้องกับการควบคุมการเข้าถึงแบบโรลเบสและปัญหาที่เกิดขึ้นของการควบคุมการเข้าถึงแบบโรลเบส

1.6.2 เพื่อพัฒนารูปแบบการควบคุมการเข้าถึงแบบโรลเบส ในด้านการกำหนดใบอนุญาตให้กับบทบาทและการกำหนดบทบาทให้กับผู้ใช้ให้มีประสิทธิภาพมากขึ้น

1.6.3 สามารถนำรูปแบบการควบคุมการเข้าถึงแบบโรลเบสที่พัฒนาไปประยุกต์ใช้กับองค์กรได้อย่างมีประสิทธิภาพมากขึ้น

1.7 โครงสร้างของวิทยานิพนธ์

โครงสร้างของวิทยานิพนธ์ ประกอบด้วย 6 ส่วนคือ

บทที่ 1 บทนำ ซึ่งบทนี้จะกล่าวถึง ความเป็นมาและความสำคัญของปัญหาที่จะต้องทำงานวิจัยในเรื่องนี้ วัตถุประสงค์ของงานวิจัย แผนการดำเนินงานวิจัย และประโยชน์ที่คาดว่าจะได้รับ

บทที่ 2 ทฤษฎีและหลักการที่เกี่ยวข้อง ซึ่งจะกล่าวถึง งานวิจัยที่เกี่ยวข้อง รูปแบบการควบคุมการเข้าถึงแบบโรลเบส การกำหนดโครงสร้างแบบลำดับชั้น ข้อบังคับในรูปแบบการควบคุมการเข้าถึงแบบโรลเบส และงานวิจัยที่เกี่ยวข้อง

บทที่ 3 กล่าวถึงวิธีการออกแบบรูปแบบการควบคุมการเข้าถึงแบบโรลเบส โดยใช้วิธีการกำหนดโครงสร้างลำดับชั้นให้กับที่ตั้ง และได้เพิ่มองค์ประกอบให้กับรูปแบบขึ้นมาใหม่ อีกทั้งได้เพิ่มคุณสมบัติข้อบังคับให้กับรูปแบบใหม่นี้ด้วย

บทที่ 4 การออกแบบอัลกอริทึมสำหรับรูปแบบ ERBAC03

บทที่ 5 การพัฒนาโปรแกรมสำหรับรูปแบบ ERBAC03

บทที่ 6 การทดลองและผลการทดลอง

บทที่ 7 สรุปผลการวิจัยและข้อเสนอแนะ

บทที่ 2
งานวิจัยที่เกี่ยวข้องและทฤษฎี

2.1 งานวิจัยที่เกี่ยวข้อง

มาริดีฟ (Mavridis) [15] และทีมงานวิจัย ได้พัฒนาการป้องกันข้อมูลของคนไข้ในโรงพยาบาลโดยการกำหนดนโยบายความปลอดภัยที่เรียกว่า ไดมิเดก (DIMEDAC) ซึ่งได้มีการกำหนดองค์ประกอบของ RBAC ใหม่โดยรองรับคุณสมบัติของรูปแบบการควบคุมการเข้าถึงแบบแมนดาทอรีและแบบดีสครีท และกระบวนการควบคุมการเข้าถึงที่ใช้ใน DIMEDAC คือการนำทฤษฎีการกำหนดโครงสร้างลำดับชั้นแบบไฮเปอร์โหนด (Hyper node hierarchies) มาใช้ร่วมกับความสัมพันธ์ระหว่างผู้ใช้ ที่ตั้ง และข้อมูล อีกทั้งยังมีการเก็บข้อมูลการควบคุมการเข้าถึงเป็นแบบตารางสามมิติ (Three dimension access matrix) โดยวิธีการนี้ได้พัฒนาขึ้นเพื่อสนับสนุนกระบวนการการควบคุมการเข้าถึงแบบกระจาย

แอฟสไตน์ (Epstein) [16] ได้สร้างรูปแบบใหม่ที่ขยายขีดความสามารถของรูปแบบ RBAC เดิม โดยได้พัฒนาในส่วนของการกำหนดใบอนุญาตให้กับบทบาท ซึ่งได้มีการกำหนดรายละเอียดเพิ่มขึ้นระหว่างองค์ประกอบของ RBAC คือ บทบาทและใบอนุญาต ซึ่งในการพัฒนานี้มีเป้าหมายคือ การกำหนดรูปแบบใหม่เป็นชั้น (Layered model) เพื่อรองรับรายละเอียดที่กำหนดไว้อย่างมีประสิทธิภาพ และยังนำคุณสมบัติของระบบการไหลของงาน (Workflow system) มาประยุกต์ใช้ร่วมกับรูปแบบใหม่อีกด้วย

เคริน (Kern) [17] ได้พัฒนารูปแบบใหม่โดยใช้วิธีการกำหนดประเภทของบทบาทให้ประยุกต์ใช้งานร่วมกับองค์กรได้อย่างมีประสิทธิภาพ ซึ่งจุดเด่นของรูปแบบนี้คือ รองรับสภาพแวดล้อมทางด้านเทคโนโลยีสารสนเทศขององค์กรขนาดใหญ่ที่มีระบบปฏิบัติการและโปรแกรมที่หลากหลาย

โมเยอร์ (Moyer) [18] ได้ออกแบบระบบรักษาความปลอดภัยที่อยู่บนพื้นฐานของรูปแบบ RBAC โดยกำหนดประเภทของบทบาทออกเป็น 3 ประเภทคือ ซับเจ็กต์ ออบเจ็กต์ และสภาพแวดล้อม ซึ่งการกำหนดประเภทของบทบาทออกเป็นหลายประเภทมีวัตถุประสงค์เพื่อให้ง่ายในการทำความเข้าใจต่อนโยบายรักษาความปลอดภัยโดยปราศจากความรู้ทางด้านเทคนิค

การทำงานของบทบาทในตอนแรก ไม่ได้ต้องการใช้ในการเข้าถึงคอมพิวเตอร์ แต่ทำงานภายใต้กฎระเบียบและสังคมสำหรับทฤษฎีองค์กร [19] ซึ่งบทบาทคือกลุ่มของสิทธิ์และหน้าที่ที่เกี่ยวข้องกับตำแหน่งที่กำหนดให้กับบุคคล ต่อมา ซานฮู [12] ได้พัฒนารูปแบบการควบคุมการ

เข้าถึงแบบโรลเบส (RBAC) โดยได้กำหนดแนวคิดของบทบาทไว้ด้วย และติง (Ting) [20] ได้วิจัยเกี่ยวกับกระบวนการรักษาความปลอดภัยของข้อมูลที่อยู่บนพื้นฐานของผู้ใช้กับบทบาท

ใน ระหว่างการประชุมปฏิบัติการของ ACM ก็ได้มีการกำหนดนิยามที่เป็นมาตรฐานของ RBAC และได้อภิปรายถึงคุณสมมุติของ RBAC บทบาทกับกลุ่ม และแนวโน้มในอนาคตของ RBAC [21] ซึ่งในขณะนั้นก็ได้เกิดแนวคิดเรื่องวิศวกรรมบทบาท (Role engineering) ขึ้นมา [22] โดย คอยน์ (Coyne) ได้นิยามวิศวกรรมบทบาท และใช้ชื่อว่า กระบวนการนิยามบทบาทและกำหนดใบอนุญาตให้กับบทบาท ซึ่งกระบวนการนี้มีขั้นตอนทั้งหมดเจ็ดขั้นตอนในการระบุบทบาท อย่างไรก็ตามในเอกสารก็ยังไม่ได้ให้รายละเอียดถึงวิธีการกำหนดใบอนุญาตให้กับบทบาท

หนานซามา (Nyanchama) และ ออสบอน (Osborn) [23] ได้เปรียบเทียบรูปแบบสองรูปแบบคือรูปแบบการควบคุมการเข้าถึงแบบแมนดาทอรี (Mandatory access control: MAC) และแบบโรลเบส ต่อมาได้กำหนดเครื่องมือที่แสดงผลแบบกราฟฟิก เรียกว่า โรลกราฟ (Role graph) [24] ซึ่งเป็นเครื่องมือที่ช่วยในการกำหนดบทบาทหรือเส้นที่ละเมิดเงื่อนไขของแบบ MAC และต่อจากนั้น ซานฮู [25] ได้แสดงให้เห็นว่าองค์ประกอบของ RBAC ควรจะปรับแต่งอย่างไรให้เหมาะสมกับรูปแบบการควบคุมการเข้าถึงแบบแมนดาทอรีที่มีโครงสร้างเป็นโครงตาข่าย (Lattice-based mandatory access controls) ซึ่งเขาได้สาธิตถึงวิธีการกำหนดคุณสมบัติแบบโครงตาข่ายโดยใช้รูปแบบของ RBAC

RBAC มีประสิทธิภาพในการทำงานร่วมกับรูปแบบการควบคุมการเข้าถึงแบบแมนดาทอรี (MAC) และแบบดีสครีท (DAC) [26][27] มีลักษณะเป็นนโยบาย ซึ่งนโยบายนี้เป็นผลลัพธ์โดยตรงขององค์ประกอบ RBAC และส่วนที่เกี่ยวข้องกัน [11]

แกสเซอร์ (Gasser) และแมคเดอมอทท์ (McDemott) [28] กำหนดการโอนสิทธิ์ระหว่างผู้ใช้และเครื่องคอมพิวเตอร์ โดยผู้ใช้มีสิทธิ์ในการประมวลผลทรัพยากรที่เขาเป็นเจ้าของเท่านั้น ในบางกรณีผู้ใช้อาจโอนสิทธิ์บางอย่างได้ด้วย ข้อบังคับการโอนสิทธิ์ส่วนใหญ่ปรากฏอยู่ในระบบความปลอดภัยแบบลำดับขั้น ที่ผู้ใช้ถูกแบ่งระดับการใช้งาน

ในขอบเขตของการให้สิทธิ์เป็นหลักการหนึ่งในแปดหลักการของการป้องกันข้อมูลสารสนเทศในระบบคอมพิวเตอร์ [29] ในการวิจัยได้มีการพัฒนาระบบคอมพิวเตอร์ร่วมกับสองคีย์หลักคือ มีความเข้มแข้งมากขึ้นและยืดหยุ่นขึ้นมากกว่าระบบที่มีคีย์เดียว ซึ่งทำให้ไม่มีการปลอมแปลง การฝ่าฝืนความน่าเชื่อถือเกิดขึ้น

คาล์ค (Clark) และ วิลสัน (Wilson) [30] ได้ระบุหลักการแบ่งแยกหน้าที่ที่เป็นกระบวนการหนึ่งในสองกระบวนการที่ทำให้เกิดความมั่นใจในเรื่องของการคงสภาพข้อมูล SoD เป็นกระบวนการที่ต่อต้านการหลอกลวงและความผิดพลาดที่อาจเกิดขึ้นได้ ขณะเดียวกันยังรับรองความสอดคล้องกันระหว่างออบเจ็กต์ของระบบกับออบเจ็กต์ในความเป็นจริงด้วย และพวกเขาได้

นำกระบวนการดังกล่าวไปใช้ในระดับนโยบาย ซึ่งได้แบ่งนโยบายออกเป็นหลายขั้นตอน โดยในแต่ละขั้นตอนถูกปฏิบัติโดยต่างบุคคลกัน

ในหัวข้อของหลักการแบ่งแยกหน้าที่ยังได้มีการศึกษากันอย่างกว้างขวางในหลายๆ ประเด็น ยกตัวอย่างเช่นในเอกสาร [31-41] ซึ่งงานวิจัยเหล่านี้ได้นำมาใช้อ้างอิงโดยตรงกับวิทยานิพนธ์เล่มนี้

คุฮ์น (Kuhn) [37] ได้ค้นพบบทบาทแบบ mutual exclusion และได้ให้ความหมายของความต้องการด้านการแบ่งแยกหน้าที่ ซึ่งเขาได้นำเสนอประเภทของหลักการแบ่งแยกหน้าที่โดยอาศัยความต้องการแบ่งแยกหน้าที่เป็นตัวแบ่ง เช่นแบ่งเป็นแบบสแตติกกับไดนามิกหรือการกันออกไปแบบทั้งหมดหรือแบบบางส่วน การกันออกไปแบบทั้งหมดคือ การที่ไม่มีใบอนุญาตหรือผู้ใช้ถูกกำหนดให้กับบทบาทแบบ exclusion ส่วนการกันออกไปแบบบางส่วนคือ บทบาทแบบ mutual exclusion มีการแบ่งปันใบอนุญาตหรือผู้ใช้ร่วมกัน แต่ในแต่ละบทบาทไม่ควรมีการกำหนดใบอนุญาตให้เหมือนกับบทบาทอื่น

หนานชามา (Nyanchama) และ ออสบอร์น (Osborn) [40] ได้ศึกษาประเภทของการขัดแย้งที่หลากหลาย และได้ประเมินผลกระทบที่เกิดขึ้นในลำดับชั้นบทบาทในทางลึกในขอบเขตของรูปแบบบทบาทแบบกราฟ (Role-graph model)

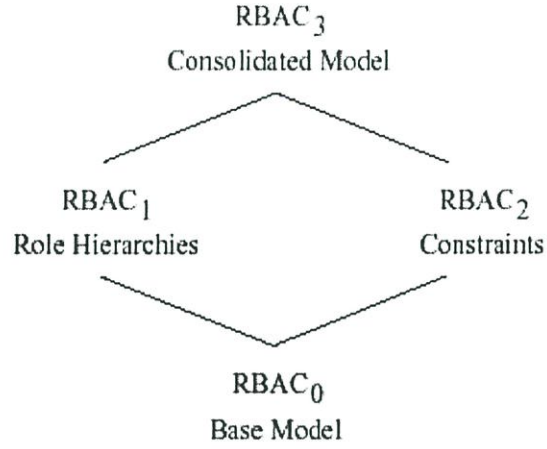
อาฮ์น (Ahn) และ ซานฮู (Sandhu) [41] ได้กำหนดภาษา RSL99 สำหรับระบุข้อบังคับของหลักการแบ่งแยกหน้าที่ ซึ่งได้กำหนดนิยามของการขัดแย้งผู้ใช้ การขัดแย้งบทบาท และการขัดแย้งใบอนุญาต โดยภาษา RSL99 สามารถนำไปใช้ศึกษา SoD ในสภาพแวดล้อมของ RBAC ได้เป็นอย่างดี ภาษานี้ช่วยให้ระบุคุณสมบัติของหลักการแบ่งแยกหน้าที่ ซึ่งทำให้เกิดความเข้าใจมากขึ้น

2.2 ทฤษฎี

2.2.1 รูปแบบการควบคุมการเข้าถึงแบบโรลเบส (Role Based Access Control: RBAC)

แนวคิดของ RBAC มีรูปร่างที่ไม่แน่นอน ซึ่งอาจจะเริ่มจากแนวคิดอย่างง่ายจนดำเนินไปเป็นผลสำเร็จ แต่ก็จะมีความสลับซับซ้อนเกิดขึ้นด้วย และเพื่อให้เข้าใจรูปร่างแนวคิดของ RBAC ซึ่งได้กำหนดรูปแบบระดับแนวคิดไว้ 4 รูปแบบดังรูปที่ 2.1 $RBAC_0$ เป็นรูปแบบพื้นฐานที่ระบุถึงความต้องการที่ต่ำที่สุดสำหรับทุกระบบที่สนับสนุนหรือรองรับ RBAC อย่างเต็มรูปแบบ ส่วน $RBAC_1$ และ $RBAC_2$ จะรวม $RBAC_0$ เข้าไว้ในตัวของมันด้วย อีกทั้งยังมีการเพิ่มคุณสมบัติเพิ่มเติมอีกด้วย ซึ่งเราจะเรียกว่ารูปแบบขั้นสูง โดย $RBAC_1$ จะเพิ่มแนวคิดลำดับชั้นบทบาท (Role hierarchies) ซึ่งเป็นแนวคิดที่บทบาทสามารถถ่ายทอดสิทธิ์ไปให้กับบทบาทอื่นได้ $RBAC_2$ ได้เพิ่มข้อบังคับ (Constraints) ซึ่งทำให้ยอมรับรูปแบบขององค์ประกอบของ RBAC ที่แตกต่างกันได้ ทั้ง $RBAC_1$

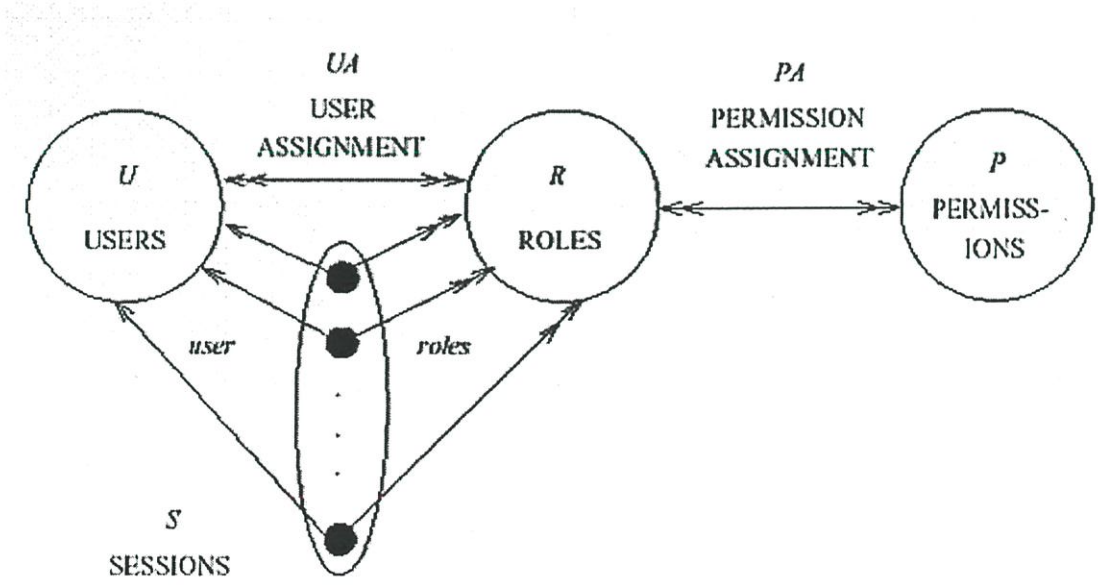
และ $RBAC_2$ เป็นรูปแบบซึ่งไม่อาจจะเปรียบเทียบได้กับรูปแบบที่มีความมั่นคงนั้นก็คือ $RBAC_3$ ซึ่งได้รวม $RBAC_0$ $RBAC_1$ และ $RBAC_2$ ไว้ด้วย ซึ่งเราเรียกรูปแบบทั้งหมดที่ได้อธิบายไว้ว่ารูปแบบ RBAC



รูปที่ 2.1 แสดงความสัมพันธ์ในรูปแบบ RBAC

2.2.1.1 รูปแบบการควบคุมการเข้าถึงแบบโรลเบสระดับศูนย์ ($RBAC_0$)

$RBAC_0$ ไม่ได้รวมคุณสมบัติ Role hierarchies กับข้อบังคับไว้ แต่ $RBAC_0$ จะมีส่วนประกอบย่อยสามส่วนสำคัญคือ ผู้ใช้ (Users: U) บทบาท (Role: R) ใบอนุญาต (Permission: P) ซึ่งจากรูปที่ 2.2 ก็จะแสดงกลุ่มของการติดต่อด้วย (Session: S)



รูปที่ 2.2 แสดงรูปแบบ $RBAC_0$

ผู้ใช้ (Users) ในที่นี้จะหมายถึงความเป็นมนุษย์ ซึ่งแนวคิดของผู้ใช้สามารถรวมถึงตัวแทนที่มีความฉลาดก็ได้ อย่างเช่น หุ่นยนต์ ซอฟต์แวร์ตัวแทน คอมพิวเตอร์ รวมถึงเครือข่ายคอมพิวเตอร์ เป็นต้น แต่เพื่อให้เกิดความง่าย ในที่นี้เราจะขอให้ผู้ใช้คือมนุษย์ ส่วนบทบาทคือ ภาระหน้าที่งานที่ทำในองค์กร ซึ่งเกี่ยวพันกับอำนาจและความรับผิดชอบที่มี

ใบอนุญาต (Permission) จะสนับสนุนวิธีการของการเข้าถึงโดยเฉพาะ ซึ่งจะสามารถกำหนดให้เข้าถึงออบเจ็กต์หนึ่งออบเจ็กต์ หรือมากกว่าหนึ่งออบเจ็กต์ได้ในระบบ และในส่วนของการให้สิทธิ์นั้นจะใช้เอกสารแทนใบอนุญาต ใบอนุญาตในทางบวกนั้นเสมือนกับว่ามีสิทธิ์ปฏิบัติกิจกรรมบางอย่างในระบบได้ ออบเจ็กต์คือ ข้อมูลหรือทรัพยากรที่มีอยู่ในระบบคอมพิวเตอร์ โดยในรูปแบบแบบแนวคิดนี้ยินยอมให้มีการแปลความหมายของคำว่าใบอนุญาตได้หลากหลาย เช่น สามารถให้เข้าถึงเครือข่ายย่อยได้ โดยหน่วยของการเข้าถึงนั้นให้กระทำเฉพาะบางคอลัมน์ บางแถว ในบางเอกสารที่เกี่ยวกับการควบคุมการเข้าถึงจะพูดถึงใบอนุญาตในทางลบด้วย ซึ่งก็คือจะถูกปฏิเสธ โดยการปฏิเสธการเข้าถึงนั้นจะมองในส่วนของข้อบังคับ (Constraint)

ธรรมชาติของใบอนุญาตจะขึ้นอยู่กับการลงรายละเอียดให้กับระบบ รวมถึงชนิดของระบบด้วย รูปแบบโดยทั่วไปสำหรับการควบคุมการเข้าถึงนั้นจะต้องกระทำต่อใบอนุญาตเสมือนสัญลักษณ์ที่ไม่สามารถแปลความหมายได้ และในแต่ละระบบต้องมีการป้องกันออบเจ็กต์ของตัวเองด้วย เช่น ระบบปฏิบัติการมีการป้องกันแฟ้มข้อมูล อุปกรณ์ พอร์ต รวมถึง การอ่าน การเขียน และการรันโปรแกรมด้วย ส่วนระบบการจัดการฐานข้อมูลก็ต้องทำการป้องกันตาราง แถว คอลัมน์ วิว รวมถึงการกระทำคำสั่งพวก SELECT UPDATE DELETE และ INSERT ด้วย ส่วนระบบบัญชีก็ต้องมีการป้องกัน เลขที่บัญชี บัญชีแยกประเภทต่างๆ เช่น บัญชีลูกหนี้ บัญชีเจ้าหนี้ รวมถึงการโอนเงิน การสร้างและการลบบัญชีอีกด้วย

ใบอนุญาตสามารถนำไปใช้กับออบเจ็กต์เดียวหรือหลายออบเจ็กต์ก็ได้ เช่น ใบอนุญาตให้มีสิทธิ์อ่านแฟ้มข้อมูลที่กำหนดให้เท่านั้น หรือให้อ่านแฟ้มข้อมูลทั้งหมดของแผนกที่ระบุไว้

จากรูปที่ 2.2 แสดงให้เห็นถึงการกำหนดผู้ใช้ (User Assignment: UA) และการกำหนดใบอนุญาต (Permission Assignment: PA) ที่มีความสัมพันธ์กัน โดยความสัมพันธ์เป็นแบบหลายต่อหลาย ซึ่งผู้ใช้สามารถเป็นสมาชิกได้ในหลายบทบาท และบทบาทหนึ่งสามารถมีสมาชิกที่เป็นผู้ใช้ได้หลายคน ในทางเดียวกันบทบาทก็สามารถมีใบอนุญาตได้หลายๆแบบ และใบอนุญาตแบบหนึ่งๆสามารถถูกกำหนดไว้ในบทบาทได้หลายบทบาท ซึ่งหัวใจสำคัญของ RBAC จะอยู่ที่ความสัมพันธ์ของทั้งสามส่วนนี้

ในแต่ละการติดต่อ (Session) จะถูกกำหนดให้ผู้ใช้หนึ่งคน เพื่อให้ผู้ใช้ระบุบทบาทที่เป็นไปได้ ส่วนหัวลูกศรสองหัวที่เชื่อมจากกลุ่มการติดต่อไปยังบทบาทในรูปที่ 2.2 หมายถึง บทบาทจะถูก

กระตุ้นพร้อมๆกันได้ และผู้ใช้หนึ่งคนสามารถมีได้หลายการติดต่อ ดังที่ระบุไว้ในรูปที่ 2.2 ซึ่งจะสังเกตจากหัวลูกศรหัวเดียวที่เชื่อมจากกลุ่มการติดต่อไปยังผู้ใช้

ผู้ใช้สามารถเปิดการติดต่อได้หลายครั้งในเวลาเดียวกัน โดยในแต่ละครั้งของการติดต่อผู้ใช้จะต้องกระทำคนละบทบาทกัน ซึ่งคุณสมบัติของ $RBAC_0$ นี้สนับสนุนหลักการจำกัดสิทธิ์ให้น้อยที่สุดด้วย คือว่าเมื่อผู้ใช้เป็นสมาชิกในบทบาทหลายๆบทบาท ก็สามารถทำการยกเลิกงานที่ดำเนินการเสร็จเรียบร้อยแล้วได้ ดังนั้นถ้าผู้ใช้เป็นสมาชิกของบทบาทที่มีอำนาจมากๆ ก็สามารถทำการยกเลิกบทบาทชั่วคราวได้ และเมื่อผู้ใช้ต้องการใช้บทบาทที่ถูกยกเลิกไปแล้วนั้น ก็สามารถกระตุ้นบทบาทให้สามารถนำมาใช้ใหม่ได้ในระหว่างที่การติดต่อยังไม่เสร็จสิ้น หลักการของการติดต่อนี้จะคล้ายๆกับหลักการเดิมของซับเจ็กต์ในเอกสารที่พูดถึงเรื่องการควบคุมการเข้าถึงไม่ว่าซับเจ็กต์หรือการติดต่อก็เป็นหน่วยหนึ่งของการควบคุมการเข้าถึง ผู้ใช้อาจจะมีได้หลายซับเจ็กต์หรือหลายการติดต่อได้ในเวลาเดียวกัน พร้อมกับมีความแตกต่างกันในเรื่องของใบอนุญาต

จากที่กล่าวมาทั้งหมดตอนต้น จึงได้มีการกำหนดนิยามไว้ดังนี้

นิยาม 1 รูปแบบ $RBAC_0$ มีองค์ประกอบดังนี้

- U R P และ S (ผู้ใช้ บทบาท ใบอนุญาต และการติดต่อ)
- $PA \subseteq P \times R$ คือความสัมพันธ์แบบหลายต่อหลายระหว่างใบอนุญาตกับการกำหนดบทบาท
- $UA \subseteq U \times R$ คือความสัมพันธ์แบบหลายต่อหลายระหว่างผู้ใช้กับการกำหนดบทบาท
- ผู้ใช้: $S \rightarrow U$ คือฟังก์ชันที่กำหนดในแต่ละการติดต่อ (s_i) ให้กับผู้ใช้คนเดียว user (s_i) โดย s_i เป็นค่าคงที่สำหรับเวลาที่ใช้ติดต่อกัน
- บทบาท: $S \rightarrow 2^R$ คือฟังก์ชันที่กำหนดในแต่ละการติดต่อ (s_i) ให้กับกลุ่มของบทบาท $roles(s_i) \subseteq \{r \mid (user(s_i),r) \in UA\}$ (ซึ่งสามารถเปลี่ยนบทบาทได้พร้อมกับเวลา) และการติดต่อ Si มีใบอนุญาต $U_{r \in roles(si)}\{p \mid (p,r) \in PA\}$

โดยคาดหวังว่าการกำหนดผู้ใช้รวมถึงการกำหนดใบอนุญาตจะถูกกำหนดไว้ในบทบาทเดียวเท่านั้น ซึ่งมีความเป็นไปได้ที่สองบทบาทมีการกำหนดใบอนุญาตเหมือนกันได้ แต่ทั้งสองบทบาทก็ยังคงแยกจากกัน ซึ่งการกำหนดผู้ใช้ก็จะเป็นในทำนองเดียวกัน

ที่กล่าวมาก่อนหน้านี้ว่า $RBAC_0$ จะปฏิบัติต่อใบอนุญาตเสมือนสัญลักษณ์ที่ไม่สามารถแปลความหมายได้ นั่นเพราะว่าโดยธรรมชาติของใบอนุญาตนั้นจะถูกนำไปใช้งานกับระบบที่มีความอิสระ ซึ่งเราต้องการให้ใบอนุญาตนี้สามารถนำไปประยุกต์กับข้อมูลและทรัพยากรที่มีอยู่ในระบบไม่ใช่นำไปประยุกต์กับองค์ประกอบของ RBAC เอง ใบอนุญาตนี้สามารถปรับปรุงแก้ไขกลุ่มของ U R P และความสัมพันธ์ PA และ UA นี้จะถูกเรียกว่า ใบอนุญาตทางการบริหาร ซึ่งจะกล่าวใน

ภายหลังในเรื่องรูปแบบการจัดการสำหรับ RBAC แต่ในตอนนี้เราจะสมมุติว่ามีเจ้าหน้าที่ด้านความปลอดภัยเพียงคนเดียวสำหรับทำการเปลี่ยนแปลงองค์ประกอบต่างๆใน RBAC

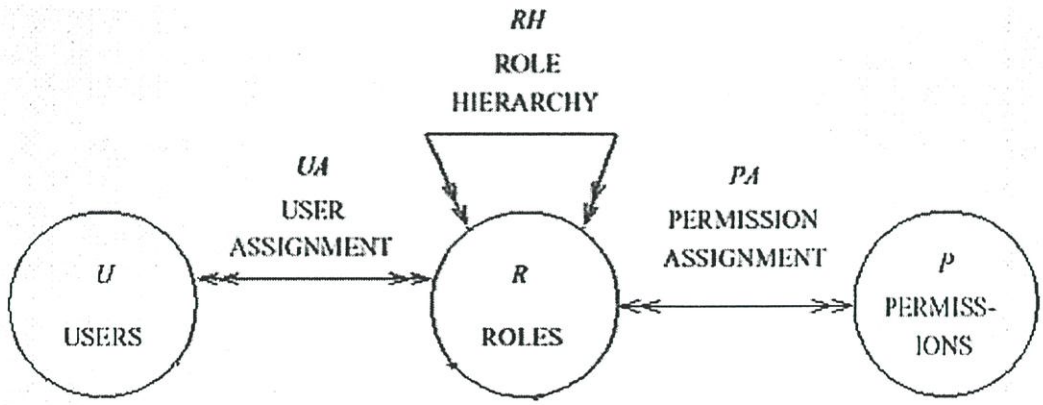
การติดต่อกันนั้นจะอยู่ภายใต้การควบคุมของแต่ละบุคคล โดยผู้ใช้สามารถสร้างการเชื่อมต่อและเลือกที่จะเป็นบทบาทใดบทบาทหนึ่งได้ตามที่ตัวเองได้รับ ซึ่งบทบาทที่ผู้ใช้กระทำอยู่ในการติดต่อครั้งหนึ่งๆสามารถเปลี่ยนแปลงได้ตามแต่ดุลยพินิจของผู้ใช้เอง และการติดต่อจะสิ้นสุดลงก็ต่อเมื่อผู้ใช้ทำการยกเลิกเอง (บางระบบจะยกเลิกการเชื่อมต่อเองถ้าบทบาทของผู้ใช้ไม่ได้ถูกกระตุ้นเป็นเวลานานๆ)

$RBAC_0$ ไม่มีความคิดที่จะให้หนึ่งการติดต่อสามารถสร้างการติดต่ออื่นๆได้อีก โดยการติดต่อจะถูกสร้างขึ้นโดยผู้ใช้เท่านั้น เพื่อไม่ให้เกิดความซับซ้อนขึ้น แต่อย่างที่ทราบกันว่าสถานการณ์เช่นนี้ก็มีทางเป็นไปได้ถ้าต้องการสร้างการเชื่อมต่ออื่นๆ ก็แค่ปฏิบัติบทบาทที่แตกต่างกันออกไปนั้นเอง

นักวิจัยบางท่านได้รวมภาระหน้าที่ (Duties) เข้าไปกับใบอนุญาตด้วยเปรียบเสมือนเป็นคุณสมบัติของบทบาท โดยภาระหน้าที่นี้ก็คืองานที่ผู้ใช้ต้องปฏิบัติในองค์กร ซึ่งในมุมมองของเราในที่นี้ ภาระหน้าที่จะไม่นำมารวมอยู่ใน $RBAC_0$ แต่ก็ได้มีรูปแบบการควบคุมการเข้าถึงขั้นสูงที่กล่าวถึงภาระหน้าที่โดยตรง อย่างเช่น Task-based authorization

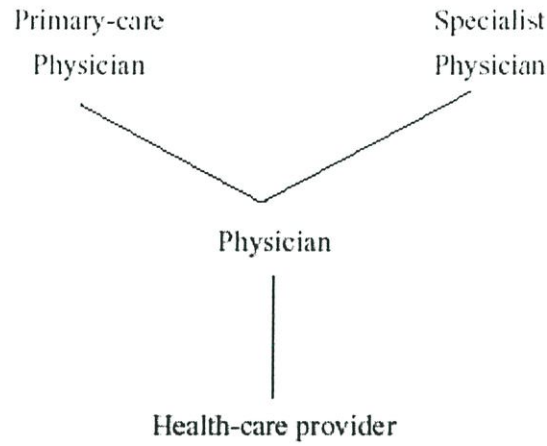
2.2.1.2 รูปแบบการควบคุมการเข้าถึงแบบโรลเบสระดับหนึ่ง ($RBAC_1$)

รูปแบบการควบคุมการเข้าถึงแบบโรลเบสระดับหนึ่งนี้ได้เพิ่มคุณสมบัติลำดับชั้นบทบาทร่วมกับ $RBAC_0$ ดังรูปที่ 2.3



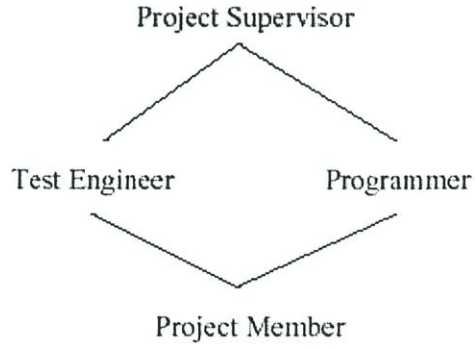
รูปที่ 2.3 แสดงรูปแบบ $RBAC_1$

ซึ่งลำดับชั้นบทบาท หมายถึง โครงสร้างของบทบาทที่สะท้อนให้เห็นถึงสายการบังคับบัญชาและอำนาจความรับผิดชอบขององค์กร ดังแสดงในรูปที่ 2.4 บทบาทที่มีอำนาจมากกว่าจะอยู่เหนือบทบาทที่มีอำนาจน้อยกว่า



รูปที่ 2.4 แสดงตัวอย่างลำดับชั้นบทบาท (Example of Role Hierarchies)

จากรูปที่ 2.4 บทบาทแพทย์ ได้รับการถ่ายทอดใบอนุญาตทั้งหมดมาจากบทบาท Health-care provider ส่วนบทบาท Primary-care กับ Specialist ได้รับการถ่ายทอดใบอนุญาตมาจากบทบาทแพทย์ แต่ในแต่ละบทบาทจะถูกกำหนดให้มีใบอนุญาตที่แตกต่างกัน



รูปที่ 2.5 แสดงการสืบทอดใบอนุญาตแบบมัลติเพิล (Multiple Inheritance of Permissions)

จากรูปที่ 2.5 บทบาท Project Supervisor สืบทอดมาจากบทบาท Test Engineer กับบทบาท Programmer ดังนั้นนิยามของ $RBAC_1$ จะกำหนดไว้ดังนี้

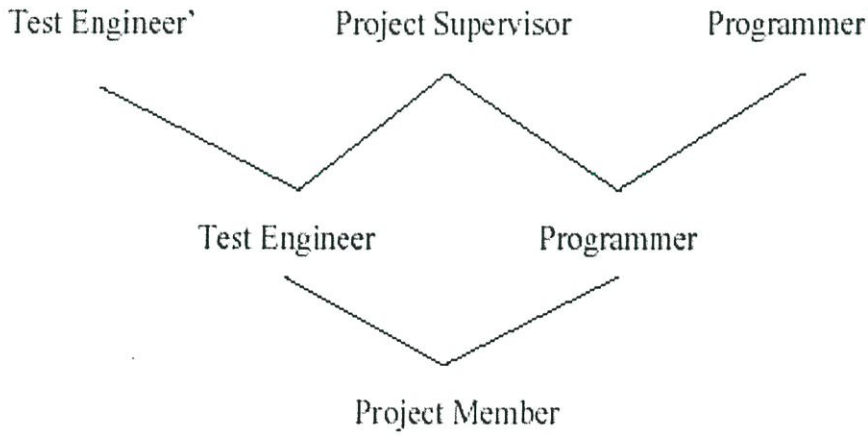
นิยาม 2 รูปแบบ $RBAC_1$ มีองค์ประกอบดังนี้

- U R P S PA และ UA เหมือนกับ $RBAC_0$
- $RH \subseteq R \times R$ คือลำดับชั้นของ R จะถูกเรียกว่า Role Hierarchy หรือ Role Dominance Relation โดยจะถูกเขียนด้วยเครื่องหมาย $\geq$

- Roles: $S \rightarrow 2^R$ ถูกปรับปรุงมาจาก $RBAC_0$ โดยต้องการให้ $roles(s_i) \subseteq \{r \mid (\exists r' \geq r)[user(s_i),r') \in UA]\}$ (ซึ่งสามารถเปลี่ยนแปลงได้) และการติดต่อ s_i จะมีใบอนุญาต $U_{r \in roles(si)}\{p \mid (\exists r'' \leq r)[(p,r'') \in PA]\}$

ผู้ใช้จะถูกให้สามารถทำการติดต่อด้วยบทบาทลูกใดๆก็ได้ที่ผู้ใช้เป็นสมาชิกอยู่ และใบอนุญาตในการติดต่อจะถูกกำหนดให้กับบทบาทที่ใช้อยู่โดยตรง ซึ่งก็คือกำหนดให้กับบทบาทลูกนั้นเอง

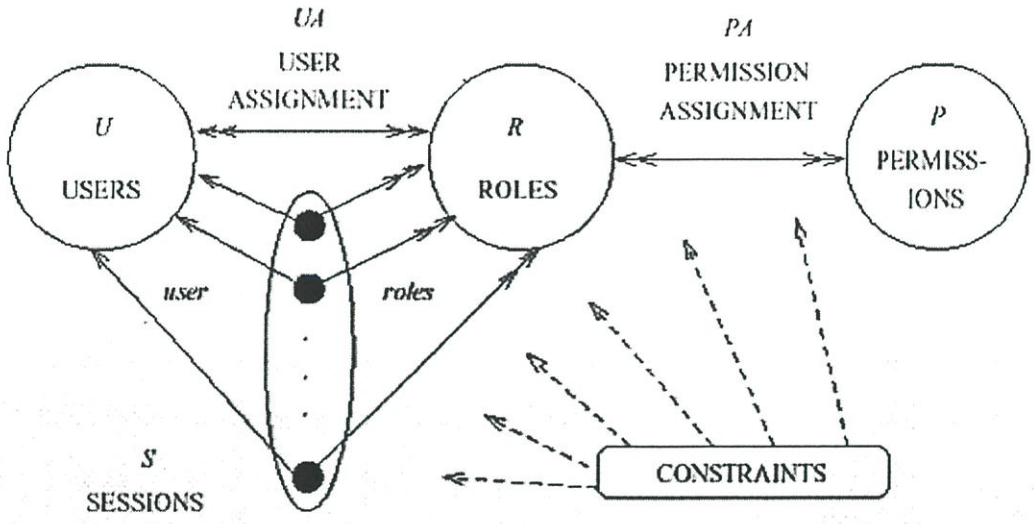
บางครั้งการใช้วิธีลำดับชั้นก็มีข้อจำกัดในการสืบทอดอยู่เหมือนกัน ซึ่งจากลำดับชั้นในรูปที่ 2.6 จะเห็นได้ว่า บทบาท Project supervisor จะสืบทอดมาจากสองบทบาท แต่ละสันนิษฐานได้ว่าบทบาท Test engineer จะเก็บรักษาใบอนุญาตบางอย่างไว้ เพื่อไม่ให้ตกทอดไปถึงบทบาท Project supervisor สถานการณ์เช่นนี้ก็อาจเกิดขึ้นได้โดยไม่ผิดอะไร แต่จากตัวอย่างถ้าการเข้าถึงงานที่ยังไม่สมบูรณ์ โดยอาจจะไม่เหมาะสมหากใช้บทบาทพ่อ ขณะที่ RBAC สามารถกระทำได้โดยการเข้าถึงบทบาท Test engineer ซึ่งสามารถปรับให้เหมาะสมได้โดยกำหนดบทบาทใหม่โดยใช้ชื่อว่า Test engineer′ ซึ่งมีความสัมพันธ์กับบทบาท Test engineer ดังรูปที่ 2.6 ใบอนุญาตบางอย่างของบทบาท Test engineer จะถูกกำหนดให้กับบทบาท Test engineer′ และใบอนุญาตบางอย่างนี้จะไม่ได้ถูกถ่ายทอดไปยังบทบาท Project supervisor เลย เราจะเรียกบทบาท Test engineer′ ว่าบทบาทส่วนตัว



รูปที่ 2.6 แสดงตัวอย่างลำดับชั้นบทบาทแบบส่วนตัว

2.2.1.3 รูปแบบการควบคุมการเข้าถึงแบบโรลเบสระดับสอง ($RBAC_2$)

รูปแบบการควบคุมการเข้าถึงแบบโรลเบสระดับสองนี้ได้เพิ่มคุณสมบัติข้อบังคับ (Constraints)ร่วมกับ $RBAC_0$ ดังรูปที่ 2.7



รูปที่ 2.7 แสดงรูปแบบ $RBAC_2$

$RBAC_2$ มีหลักการของข้อบังคับ (Constraints) ซึ่งจากรูปที่ 2.1 $RBAC_2$ กับ $RBAC_1$ ไม่ได้มีความสัมพันธ์กัน โดยข้อบังคับเป็นหลักการที่สำคัญของ RBAC ซึ่งบางครั้งก็ได้มีการโต้เถียงกันเกี่ยวกับหลักการ ยกตัวอย่างเช่น ผู้จัดการฝ่ายจัดซื้อกับผู้จัดการฝ่ายการเงินในองค์กรส่วนใหญ่จะกำหนดให้เป็นสมาชิกของบทบาทที่แตกต่างกันเพื่อป้องกันการทุจริต ซึ่งหลักการเช่นนี้จะเรียกได้ว่า หลักการแบ่งแยกหน้าที่ (Separation of duties)

ข้อบังคับเป็นกระบวนการที่สำคัญอย่างมากสำหรับการนำไปประยุกต์ใช้กับนโยบายระดับสูงขององค์กร โดยการจัดการ RBAC แบบรวมศูนย์นี้สามารถให้เจ้าหน้าด้านความปลอดภัยจัดการเพียงคนเดียวได้ แต่บางครั้งเพื่อให้งานบรรลุเป้าหมายก็อาจจะให้มีเจ้าหน้าที่ด้านความปลอดภัยหลายๆคน โดยให้แต่ละคนดูแลเฉพาะส่วน อย่างไรก็ตามในที่นี้จะยังไม่พูดถึงการจัดการ RBAC แบบกระจาย โดยข้อบังคับกลายเป็นกระบวนการที่เจ้าหน้าที่ด้านความปลอดภัยใช้ในการจำกัดความสามารถของผู้ใช้ ผู้ซึ่งมีสิทธิ์ในการจัดการ โดยจะมีหัวหน้าของเจ้าหน้าที่ด้านความปลอดภัยคอยกำหนดขอบเขตที่ยอมรับได้ และกำหนดความต้องการที่จำเป็นให้กับเจ้าหน้าที่ด้านความปลอดภัยคนอื่นที่มีส่วนร่วมในการจัดการ RBAC

เมื่อพิจารณา $RBAC_0$ ข้อบังคับสามารถนำไปประยุกต์ใช้ร่วมกับความสัมพันธ์ UA และ PA ได้ อีกทั้งยังนำไปประยุกต์กับผู้ใช้และบทบาทที่มีการติดต่อกันได้ด้วย ข้อบังคับจะคอยวินิจฉัยความสัมพันธ์และการทำงานแต่ละส่วน แล้วจะคืนค่า "ยอมรับได้" หรือ "ยอมรับไม่ได้" กลับมา ข้อบังคับสามารถมีมุมมองเสมือนประโยคที่มีภาษาแบบทางการ แต่ที่จริงแล้ว ข้อบังคับควรมีมุมมองแบบภาษาธรรมชาติจะดีกว่า ซึ่งในที่นี้เราจะศึกษาข้อบังคับแบบไม่เป็นทางการมากกว่าแบบเป็นทางการ ซึ่งมีนิยามดังนี้

นิยาม 3 $RBAC_2$ จะไม่มีการเปลี่ยนแปลงใดๆจาก $RBAC_0$ ยกเว้นการเพิ่มขึ้นมาในส่วนของ ข้อบังคับที่ซึ่งมีการคืนค่ากลับขององค์ประกอบต่างๆของ RBAC ไม่ว่าจะเป็นค่า "ยอมรับได้" หรือ "ยอมรับไม่ได้" ก็ตาม ซึ่งค่า "ยอมรับได้" จะถูกให้ได้รับอนุญาต

ข้อบังคับที่กล่าวถึงส่วนมากในบริบทของ RBAC ก็คือบทบาทแบบ mutual exclusive โดยผู้ใช้คนเดียวกันสามารถถูกกำหนดให้ได้รับบทบาทได้แค่บทบาทเดียวในเซตของ mutual exclusive โดยข้อบังคับแบบนี้จะสนับสนุนหลักการแบ่งแยกหน้าที่ด้วย ส่วนข้อบังคับแบบคู่จะไม่ค่อยถูกกล่าวถึงไว้ในเอกสารเท่าไร แต่ในความเป็นจริงแล้วข้อบังคับแบบ mutual exclusive สำหรับการกำหนดเกี่ยวกับเรื่องใบอนุญาตก็สามารถขยายเพิ่มเติมการรับรองสำหรับหลักการแบ่งแยกหน้าที่ โดยข้อบังคับแบบคู่นี้มีความต้องการให้ใบอนุญาตเดียวกันสามารถถูกกำหนดให้บทบาทมากที่สุดได้เพียงบทบาทเดียวในเซตของ mutual exclusive เมื่อพิจารณาบทบาทแบบ mutual exclusive สองบทบาท คือผู้จัดการฝ่ายบัญชีกับผู้จัดการฝ่ายจัดซื้อ mutual exclusion ในส่วนของ UA จะระบุไว้ว่าผู้ใช้หนึ่งคนห้ามเป็นสมาชิกของทั้งสองบทบาท และ mutual exclusive ในส่วนของ PA ก็จะระบุไว้ว่าใบอนุญาตเดียวกันจะไม่สามารถถูกกำหนดให้กับทั้งสองบทบาท โดยปกติถ้าใบอนุญาตที่ถูกกำหนดให้กับบทบาทผู้จัดการฝ่ายบัญชีแล้ว ในข้อบังคับแบบ mutual exclusion ส่วนของ PA จะระบุไว้ว่าให้มีการป้องกันใบอนุญาตจากเหตุการณ์ที่อาจเกิดขึ้นโดยไม่ได้ตั้งใจหรือเหตุการณ์ที่ประสงค์ร้าย หรือเหตุการณ์ที่มีการกำหนดใบอนุญาตนี้ให้กับบทบาทผู้จัดการฝ่ายจัดซื้อ เพื่อให้ชัดเจนยิ่งขึ้น ข้อบังคับของ PA จะใช้เพื่อกำหนดให้จำกัดการกระจายของใบอนุญาตที่มีอำนาจ ยกตัวอย่างเช่น ไม่มีทางที่บทบาท A กับ B จะได้รับลายเซ็นเดียวกันในบัญชีเฉพาะ แต่ทั้ง A และ B อาจจะได้รับบทบาทหนึ่งบทบาทจากสองบทบาท เพื่อได้รับใบอนุญาตนี้

ตัวอย่างอื่นๆเช่น ข้อบังคับในการกำหนดผู้ใช้สามารถมีการกำหนดจำนวนสมาชิกมากสุดในแต่ละบทบาท เช่น มีบุคคลแค่คนเดียวที่มีบทบาทประธานบริษัท เราเรียกข้อบังคับแบบนี้ว่าข้อบังคับแบบกำหนดจำนวน (Cardinality constraint) แต่การกำหนดจำนวนแบบน้อยที่สุดจะทำได้ยาก เช่น การกำหนดจำนวนน้อยสุดของผู้ใช้ที่เป็นสมาชิกในบทบาท จะทำให้ระบบทำอย่างไรถ้าบทบาทนั้นยังไม่มีสมาชิกอยู่เลย

แนวคิดของบทบาทก่อนหน้า (Prerequisite role) อยู่บนพื้นฐานของความสามารถและการได้ครอบครอง ด้วยเหตุนี้ผู้ใช้สามารถถูกกำหนดให้กับบทบาท A ได้ ถ้าผู้ใช้คนนั้นเป็นสมาชิกของบทบาท B ยกตัวอย่างเช่น ผู้ใช้ที่เป็นสมาชิกของบทบาท project สามารถถูกกำหนดให้ได้รับบทบาท testing task ได้ จากตัวอย่างนี้ บทบาทก่อนหน้าคือบทบาทลูกที่กำลังไปเป็นบทบาทใหม่ได้ถูกสมมุติขึ้น ข้อบังคับที่เหมือนกันที่ใช้ในการกำหนดใบอนุญาตสามารถเกิดขึ้นดังนี้ ข้อบังคับควรมีส่วนช่วยเหลือต่อความสอดคล้องกัน เพื่อต้องการให้ใบอนุญาต p ถูกกำหนดให้กับบทบาท

ถ้าบทบาทนั้นมีใบอนุญาต q อยู่ก่อน เช่นในบางระบบใบอนุญาตให้อ่านแฟ้มข้อมูลต้องการการอนุญาตให้อ่านไดเรกทอรีที่มีแฟ้มข้อมูลนั้นอยู่ด้วยการกำหนดใบอนุญาตแรกโดยปราศจากการกำหนดใบอนุญาตหลังก็จะไม่สมบูรณ์ โดยทั่วไปจะมีการกำหนดเงื่อนไขของบทบาทก่อนหน้าด้วยเหตุนี้ผู้ใช้สามารถถูกกำหนดให้กับบทบาท A โดยผู้ใช้จะเป็นสมาชิกหรือไม่ได้เป็นสมาชิกของบทบาทที่ระบุไว้หรือไม่ก็ตาม และสำหรับการกำหนดใบอนุญาตก็จะคล้ายคลึงกัน แนวคิดนี้จะถูกใช้ในรูปแบบการจัดการ (Administrative model)

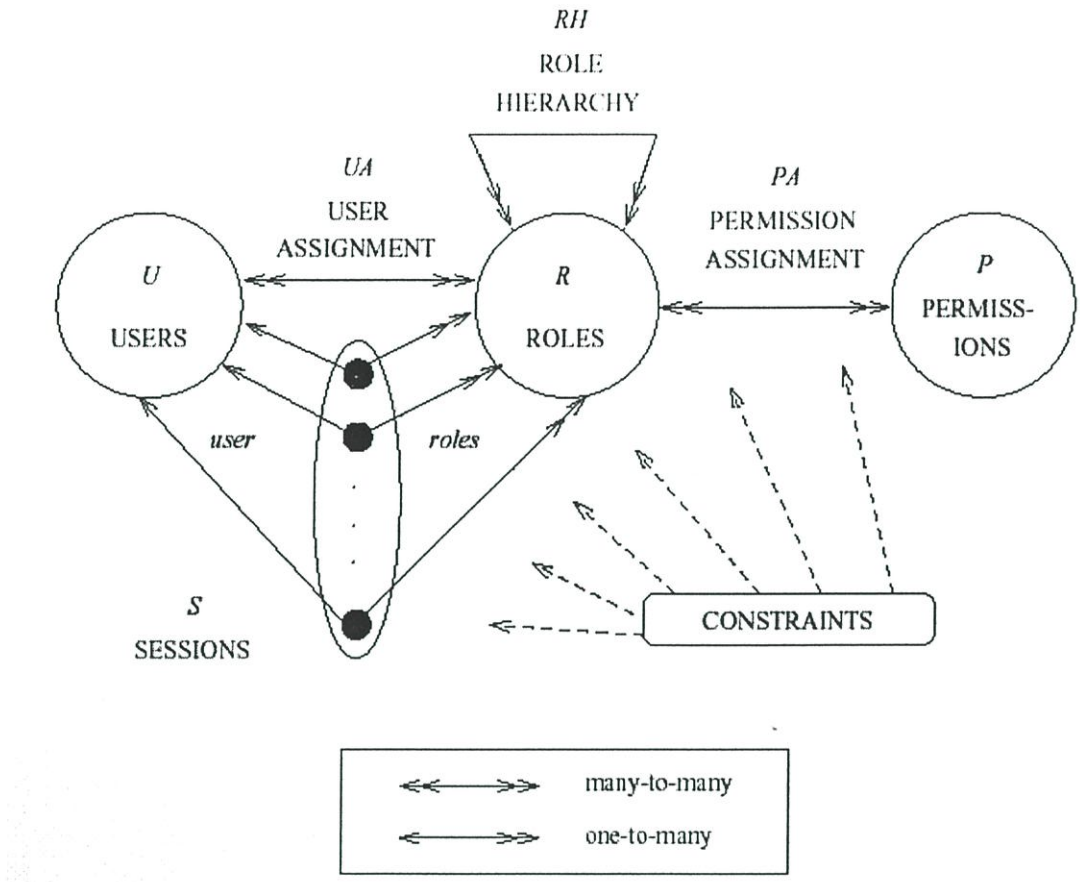
ข้อบังคับการกำหนดผู้ใช้จะมีประสิทธิผล ถ้าการฝึกฝนภายนอกเหมาะสม คือมีการบำรุงรักษาในการกำหนดการระบุผู้ใช้ให้กับงาน ถ้าเกิดมีการกำหนดบุคคลคนเดียวกันเป็นสองบุคคลหรือมากกว่านั้น การควบคุมในเรื่องการแบ่งแยกและเรื่องจำนวนก็จะล้มเลิกไป ดังนั้นควรมีความสอดคล้องกันแบบหนึ่งต่อหนึ่งระหว่างการพิสูจน์ผู้ใช้และบุคคล ข้อพิสูจน์นี้นำไปประยุกต์ใช้กับข้อบังคับใบอนุญาตได้ ถ้าการปฏิบัติการเดียวกันถูกได้รับอนุญาตโดยสองใบอนุญาตที่แตกต่างกัน ระบบ RBAC จะไม่สามารถใช้ข้อบังคับแบบการกำหนดจำนวน และหลักการแบ่งแยกหน้าที่ได้อย่างมีประสิทธิผล

ข้อบังคับสามารถนำไปประยุกต์ใช้กับการติดต่อ (Session) ได้ และในส่วนของผู้ใช้กับบทบาทที่สัมพันธ์กันในการติดต่อแต่ละครั้ง ซึ่งอาจจะยอมรับได้สำหรับผู้ใช้ที่เป็นสมาชิกในสองบทบาท แต่ผู้ใช้ไม่สามารถปฏิบัติทั้งสองบทบาทในเวลาเดียวกันได้ เพื่อให้สอดคล้องกันต้องมีการกำหนดจำนวนของการติดต่อไว้ด้วย ซึ่งทำให้ใบอนุญาตที่ถูกกำหนดสามารถถูกจำกัดได้

ลำดับชั้นบทบาทสามารถนำมาพิจารณาเสมือนข้อบังคับ ได้ ซึ่งข้อบังคับคือใบอนุญาตที่ถูกกำหนดให้กับบทบาทลูกจะต้องถูกกำหนดให้กับบทบาทพ่อทั้งหมดด้วย หรืออีกนัยหนึ่ง ข้อบังคับคือการกำหนดผู้ใช้ให้กับบทบาทพ่อจะต้องถูกกำหนดให้กับบทบาทลูกทั้งหมดด้วย ดังนั้นก็จะวินิจฉัยได้ว่า $RBAC_1$ มีความซ้ำซ้อนเกินความจำเป็น อย่างไรก็ตามเรารู้สึกว่า $RBAC_1$ ยังมีความเหมาะสมในการยอมรับแนวคิดเรื่องลำดับชั้นบทบาท ซึ่งจะเป็นการลดจำนวนข้อบังคับโดยเฉพาะ โดยจะสนใจที่การกำหนดใบอนุญาตหรือการกำหนดผู้ใช้ที่มากเกินความจำเป็น และจะเป็นการดียิ่งขึ้นถ้ามีการสนับสนุนลำดับชั้นบทบาทโดยตรงมากกว่าโดยอ้อม เพื่อลดการกำหนดที่มากเกินไปนั่นเอง

2.2.1.4 รูปแบบการควบคุมการเข้าถึงแบบโรลเบสระดับสาม ($RBAC_3$)

รูปแบบการควบคุมการเข้าถึงแบบโรลเบสระดับสามนี้ได้รวมเอา $RBAC_1$ และ $RBAC_2$ เข้าไว้ด้วยกันดังรูปที่ 2.8 โดยจะมีคุณสมบัติลำดับชั้นบทบาทกับข้อบังคับ ซึ่งได้นำมาใช้ร่วมกัน



รูปที่ 2.8 แสดงรูปแบบ $RBAC_3$

ข้อบังคับสามารถประยุกต์ใช้กับลำดับชั้นบทบาทได้ภายในตัวมันเอง โดยลำดับชั้นบทบาทก็คือคุณสมบัติที่ถูกต้องการให้เป็นลำดับเฉพาะส่วน ส่วนข้อบังคับเพิ่มเติม สามารถจำกัดจำนวนของบทบาทพ่อหรือลูก ได้ ข้อบังคับประเภทนี้นำไปใช้ในสถานการณ์ที่ผู้มีอำนาจเปลี่ยนแปลงลำดับชั้นบทบาทให้เป็นแบบกระจาย แต่หัวหน้าเจ้าหน้าที่ด้านความปลอดภัยต้องการที่จะเข้มงวดในเรื่องวิธีการทั้งหมด อย่างเช่น การเปลี่ยนแปลงที่สามารถกระทำได้

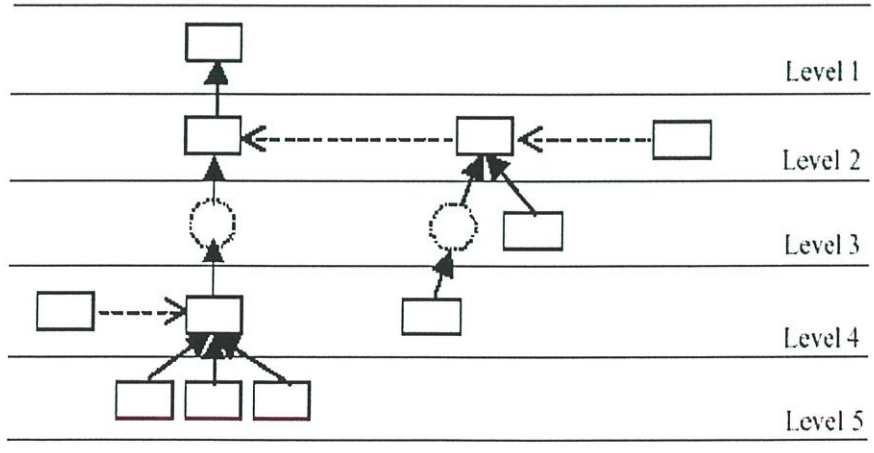
ปฏิสัมพันธ์ที่ซับซ้อนที่ปรากฏขึ้นระหว่างข้อบังคับและ ลำดับชั้น เช่น บทบาท test engineer และบทบาท programmer ได้ถูกกำหนดไว้แยกออกจากกัน ดังรูป 2.5 โดยบทบาท project supervisor ได้ฝ่าฝืนหลัก mutual exclusion นี้ ในบางกรณีการฝ่าฝืนของข้อบังคับ แบบ mutual exclusion โดยบทบาทพ่ออาจจะยอมรับได้ ขณะที่ในบางกรณีอาจจะไม่ได้ ในทำนองเดียวกันกับข้อบังคับแบบกำหนดจำนวน สมมุติให้ผู้ใช้สามารถถูกกำหนดให้กับบทบาทได้มากสุดหนึ่งบทบาท ดังนั้นการกำหนดบทบาท test engineer ในรูปที่ 2.5 จะละเมิดข้อบังคับนี้หรือไม่ หรืออีกนัยหนึ่งข้อบังคับแบบกำหนดจำนวนนำไปประยุกต์กับสมาชิกโดยตรงหรือหรือถูกสืบทอดให้กับสมาชิกอื่นๆต่อไปเรื่อยๆได้หรือไม่

ลำดับชั้นในรูป 2.6 แสดงให้เห็นว่าข้อบังคับนำไปใช้ในบทบาทส่วนตัว (Private role) อย่างไร ในกรณีที่บทบาท test engineer′ programmer′ และ project supervisor สามารถถูกกำหนดให้เป็นแบบ mutual exclusive เพราะว่าไม่มีบทบาทพ่อ ดังนั้นจึงไม่ขัดแย้งกัน โดยทั่วไปบทบาทส่วนตัวจะไม่มีบทบาทพ่อหรือบทบาทอื่นๆ เพราะว่าบทบาทส่วนตัวเป็นส่วนประกอบมากที่สุดในลำดับชั้น ดังนั้น mutual exclusion ของบทบาทส่วนตัวสามารถถูกระบุโดยปราศจากความขัดแย้งกันได้ การแบ่งปันส่วนที่คล้ายคลึงกันของบทบาทส่วนตัวสามารถถูกประกาศให้มีข้อบังคับที่มีเลขมากที่สุด ในที่นี้บทบาท test engineer จะแบ่งปันใบอนุญาตร่วมกับบทบาท project supervisor

2.2.2 **การกำหนดโครงสร้างลำดับชั้นแบบไฮเปอร์โหนด** (Hyper node hierarchy)

เพื่อให้การกำหนดโครงสร้างลำดับชั้นมีคล้ายคลึงกับหลักการของการกำหนดระดับความปลอดภัย (ประกอบด้วยระดับความปลอดภัยและกลุ่มของประเภทบทบาท) รวมกับหลักการของลำดับชั้นบทบาท [14] โดยได้ผลลัพธ์คือลำดับชั้นบทบาทที่มีการกำหนดความสำคัญของบทบาทตามตำแหน่งความลึกของลำดับชั้น อีกทั้งการกำหนดกลุ่มของประเภทบทบาทจะได้รับการสืบทอดมาจากโหนดพ่อในลำดับชั้นที่กลุ่มนั้นๆ เชื่อมต่ออยู่ ดังนั้นจึงเกิดกระบวนการกำหนดลำดับชั้นแบบไฮเปอร์โหนด (Hyper node hierarchy: HNH) [15] ขึ้นมาเพื่อรองรับหลักการข้างต้น

ลำดับชั้นแบบไฮเปอร์โหนด คือกลุ่มของโหนด (Hyper node) ดังรูปที่ 2.9 ซึ่งในแต่ละโหนดจะเชื่อมต่อกับโหนดอื่นๆ ได้โดยกิ่ง (Branch) และเส้นเชื่อม (Link) โดยกิ่งนี้ถูกใช้ในการเชื่อมต่อกับโหนดอื่นๆ ภายในลำดับชั้นจนกลายเป็นลำดับชั้นแบบไฮเปอร์โหนดในที่สุด และลำดับชั้นแบบไฮเปอร์โหนดยังรองรับคุณสมบัติการสืบทอดแบบมัลติเพิลด้วย



รูปที่ 2.9 ลำดับชั้นแบบไฮเปอร์โหนด (Hyper Node Hierarchy)

สัญลักษณ์  คือ โหนด (Hyper node)

 คือ โหนดที่ว่างเปล่า

 คือ กิ่ง (Branch)

 คือ เส้นเชื่อม (Link)

กิ่งที่ใช้เชื่อมของแต่ละโหนดมีการกำหนดทิศทางไปยังโหนดพ่อในลำดับชั้นเดียวกันที่มีระดับสูงกว่าโหนดนั้นๆ ส่วนเส้นเชื่อมก็มีการกำหนดทิศทางการเชื่อมต่อเช่นเดียวกันกับกิ่ง แต่มีความแตกต่างกันตรงที่ เส้นเชื่อมจะใช้เชื่อมต่อกับโหนดอื่นๆ ในระดับเดียวกันเท่านั้น

กระบวนการกำหนดลำดับชั้นแบบไฮเปอร์โหนดนี้จะนำมาประยุกต์ใช้ในการกำหนดโครงสร้างของบทบาทและที่ตั้งของรูปแบบการควบคุมการเข้าถึงแบบโรลเบสที่ได้พัฒนาใหม่นี้ด้วย

2.2.3 **หลักการแบ่งแยกหน้าที่** (Separation of Duty: SoD)

หลักการแบ่งแยกหน้าที่ (Separation of duty: SoD) เป็นหลักการที่มีบทบาทสำคัญอย่างมากทางด้านการรักษาความปลอดภัยในองค์กร โดยมีวัตถุประสงค์เพื่อป้องกันการทุจริตของผู้ใช้ระบบคอมพิวเตอร์ ซึ่งหลักการนี้จะไม่อนุญาตให้ผู้ใช้ระบบมีสิทธิ์ในการใช้ระบบมากเพียงพอต่อการทุจริตต่อองค์กรได้ [35] และเพื่อให้บรรลุเป้าหมายในการป้องกันการทุจริตหรือหลอกลวง จึงจำเป็นต้องมีการแบ่งงานที่ผู้ใช้ต้องปฏิบัติออกเป็นหลายๆ งาน โดยให้ในแต่ละงานถูกปฏิบัติโดยผู้ใช้คนละคนกัน

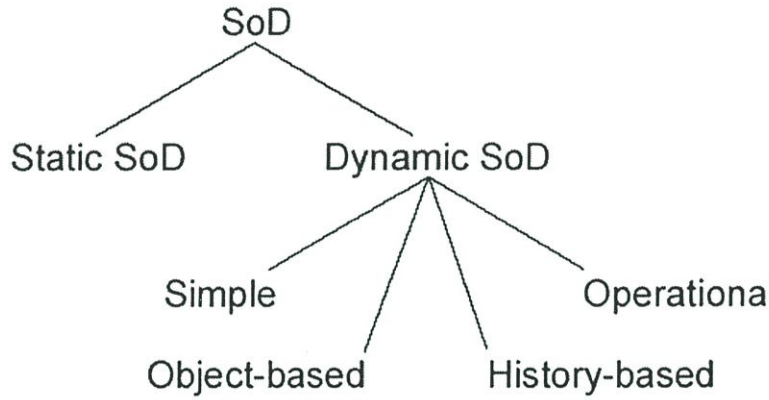
นิยามของหลักการแบ่งแยกหน้าที่ได้ถูกกำหนดไว้ในหลายงานวิจัย โดยมีนิยามหนึ่งที่น่าสนใจคือ SoD เป็นหลักความปลอดภัยที่ใช้นโยบายการควบคุมผู้ใช้มัลติเพิล ซึ่งนโยบายนี้มีความหมายว่า ต้องการบุคคลที่มากกว่าสองคนขึ้นไปมาปฏิบัติงานหรือกลุ่มของงานให้สำเร็จโดยสมบูรณ์ [36] ซึ่งการทุจริตของบุคคลเพียงคนเดียวก็ไม่อาจเกิดขึ้นได้เนื่องจากได้มีการกระจายความรับผิดชอบต่องานให้กับบุคคลหลายๆ คน แต่ก็สามารถเกิดความเสี่ยงในการหลอกลวงได้ถ้าบุคคลหลายๆ คนร่วมมือกันหลอกลวง

ผู้ใช้ในองค์กรจะเกี่ยวข้องเฉพาะงานที่มีขนาดเล็กเท่านั้น ยกตัวอย่างเช่น นาย ก. ได้รับบทบาทเป็นพนักงานนับเงินในไปรษณีย์ มีหน้าที่เพียงแค่นับเงินเมื่อมีการปิดบัญชี ซึ่งเงินจะได้รับมาจาก นาย ข. เท่านั้น โดย นาย ข. มีบทบาทเป็นพนักงานรับฝากพัสดุ ทำให้นาย ก. จะทำการทุจริตเองเพียงคนเดียวไม่ได้ ถ้าจะทำต้องได้รับความร่วมมือกับนาย ข. ด้วย แต่ถ้าไม่มีหลักการแบ่งแยกหน้าที่ ทำให้นาย ก. มีอำนาจทั้งรับเงินและนับเงินทั้งหมดเอง จึงมีโอกาสที่ทำให้เกิดการทุจริตขึ้นได้

หลักการแบ่งแยกหน้าที่สามารถนำมาประยุกต์ใช้ในรูปแบบการควบคุมการเข้าถึงได้โดยการกำหนดข้อบังคับ (Constraint)

2.2.3.1 ประเภทของหลักการแบ่งแยกหน้าที่

จากนิยามที่กำหนดโดย ซิมมอน (Simon) และ ซูโก (Zurko) [36] ได้จำแนกรูปแบบของหลักการแบ่งแยกหน้าที่ไว้หลายรูปแบบ โดยรูปแบบทั้งหมดที่กำหนดถูกจัดไว้เป็นกลุ่มหลักๆ สองกลุ่มคือ หลักการแบ่งแยกหน้าที่แบบสแตติก (Static Separation of Duty: SSoD) และหลักการแบ่งแยกหน้าที่แบบไดนามิก (Dynamic Separation of Duty: DSoD) ซึ่งแสดงดังรูปที่ 2.10



รูปที่ 2.10 ประเภทของหลักการแบ่งแยกหน้าที่

ในวิทยานิพนธ์เล่มนี้ได้กำหนดขอบเขตการวิจัยเฉพาะหลักการแบ่งแยกหน้าที่แบบสแตติกเท่านั้น ซึ่งได้กล่าวถึงในบทที่ 3

2.2.3.1.1 หลักการแบ่งแยกหน้าที่แบบสแตติก (Static Separation of Duty: SsoD)

เอนทิที (Entity) ของ RBAC ทั้งหมดสามารถกำหนดข้อบังคับไว้ภายในได้ [41] ข้อบังคับที่กล่าวถึงนี้จะหมายถึง การไม่อนุญาตให้ผู้ใช้ได้รับบทบาทสองบทบาทที่มีความขัดแย้งกัน (Conflicting roles) ซึ่งถ้าผู้ใช้ได้รับทั้งสองบทบาทที่มีความขัดแย้งกันจะทำให้มีโอกาสเกิดการทุจริตหรือหลอกลวงขึ้นได้ จากตัวอย่างที่กล่าวก่อนหน้านี้ บทบาทพนักงานตรวจสอบเงินจะขัดแย้งกับบทบาทพนักงานรับฝากพัสดุ โดยทั้งสองบทบาทนี้จะต้องไม่ถูกกำหนดให้กับผู้ใช้เพียงคนเดียว

ข้อบังคับในเอนทิทีอีกประเภทหนึ่งคือ ผู้ใช้ไม่ควรได้รับใบอนุญาตที่มีความขัดแย้งกัน (Conflicting permissions) ใบอนุญาตที่ได้รับนี้ได้จากบทบาทที่ผู้ใช้นั้นปฏิบัติอยู่ เพราะฉะนั้นต้องมีความระมัดระวังอย่างมากในการกำหนดใบอนุญาตให้กับบทบาท ยกตัวอย่างเช่น

ใบอนุญาตให้มีสิทธิ์ในการยืนยันความถูกต้องของเงินที่ตรวจสอบกับใบอนุญาตให้มีสิทธิ์ใส่ข้อมูลการเงินลงบัญชีสิ้นวันทำการ (End of day account) โดยใบอนุญาตทั้งสองใบมีความขัดแย้งกัน ดังนั้นจึงไม่ควรกำหนดให้กับผู้ใช้โดยผ่านบทบาท ซึ่งก็หมายความว่าใบอนุญาตที่ขัดแย้งนั้นไม่ควรกำหนดให้กับบทบาทนั้นเอง จากที่กล่าวมาข้างต้นได้สรุปไว้ในตารางที่ 2.1

ตารางที่ 2.1 แสดงหลักการแบ่งแยกหน้าที่แบบสแตติก

เอนทิทีที่ขัดแย้ง	ไม่สามารถเกี่ยวข้องกับ
บทบาทที่ขัดแย้ง	ผู้ใช้
ใบอนุญาตที่ขัดแย้ง	บทบาท

ข้อบังคับแบบอื่นๆ ได้ถูกพิสูจน์ไว้อยู่ในรูปแบบทางคณิตศาสตร์ ซึ่งสามารถศึกษาได้จากเอกสารงานวิจัยเรื่อง ภาษาของข้อบังคับบนพื้นฐานของบทบาท (Role-based constraint language: RCL99) [41] และในบทที่ 4 ได้กำหนดนิยามของข้อบังคับให้กับรูปแบบการควบคุมการเข้าถึงที่พัฒนาขึ้นมาใหม่ด้วย

หลักการแบ่งแยกหน้าที่แบบสแตติกนี้มีประโยชน์อย่างมากต่อผู้บริหารจัดการด้านความปลอดภัย อีกทั้งยังเป็นพื้นฐานสำคัญสำหรับการกำหนดหลักการแบ่งแยกหน้าที่แบบไดนามิกด้วย

2.2.3.1.2 **หลักการแบ่งแยกหน้าที่แบบไดนามิก** (Dynamic Separation of Duty: DsoD)

หลักการประเภทนี้ถูกใช้ควบคุมการใช้บทบาทในระบบของผู้ใช้ ซึ่งถ้าพอใจกับหลักการแบ่งแยกหน้าที่แบบสแตติกอยู่แล้ว ก็ไม่จำเป็นต้องใช้หลักการแบบไดนามิกเลย เพราะหลักการแบบสแตติกมีความเข้มงวดสูงมาก

จากตัวอย่างที่ได้กล่าวไว้แล้ว ถ้านาย ก. ได้รับบทบาทพนักงานตรวจสอบเงินแล้ว นาย ก. จะไม่มีสิทธิ์ได้รับบทบาทพนักงานรับฝากพัสดุอีก แต่ถ้าเป็นหลักการแบ่งแยกหน้าที่แบบไดนามิกจะอนุญาตให้นาย ก. กระทำได้ทั้งสองบทบาทได้ ซึ่งหลักการแบบไดนามิกนี้มีการกำหนดรูปแบบย่อยอีก 4 รูปแบบ คือ

1) **หลักการแบบไดนามิกอย่างง่าย** (Simple Dynamic SoD) เป็นหลักการที่ยอมให้บทบาทมีสมาชิกที่เหมือนกันได้ แต่สมาชิกอาจจะถูกกำหนดให้กับบทบาทเพียงบทบาทเดียว ณ เวลานั้น

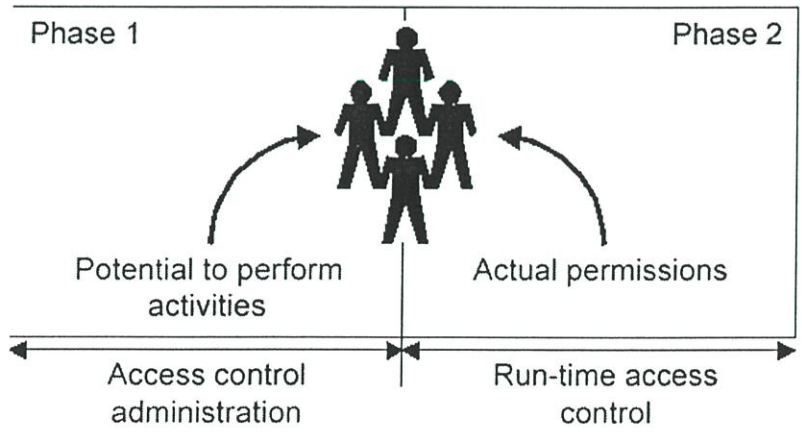
2) **หลักการแบ่งแยกหน้าที่บนพื้นฐานของออบเจ็กต์** (Object-based SoD) เป็นหลักการที่ยอมให้ผู้ใช้ปฏิบัติสองบทบาทได้ในพร้อมๆ กัน แต่มีเงื่อนไขว่าผู้ใช้ไม่สามารถกระทำกับออบเจ็กต์ที่ผู้ใช้ผู้นั้นได้กระทำมาก่อนหน้านี้ได้

3) **หลักการแบ่งแยกหน้าที่บนพื้นฐานของการปฏิบัติงาน** (Operational SoD) เป็นหลักการที่ยอมให้ผู้ใช้ปฏิบัติได้หลายบทบาทได้ โดยบทบาทที่ผู้ใช้ปฏิบัติทั้งหมดนั้นไม่ทำให้กระบวนการปฏิบัติงานเกิดความสำเร็จทั้งหมด เพื่อป้องกันไม่ให้บุคคลใดบุคคลหนึ่งสามารถปฏิบัติงานได้สำเร็จทั้งหมด ซึ่งอาจเกิดการทุจริตหลอกลวงขึ้นได้

4) **หลักการแบ่งแยกหน้าที่บนพื้นฐานของสิ่งที่ทำในอดีต** (History-based SoD) เป็นหลักการที่จัดการการปฏิบัติบทบาทได้เป็นอย่างดี โดยเป็นหลักการที่รวมมาจากหลักการแบ่งแยกหน้าที่บนพื้นฐานของออบเจ็กต์กับหลักการที่รวมมาจากหลักการแบ่งแยกหน้าที่บนพื้นฐานของการปฏิบัติงาน

บทที่ 3
รูปแบบการควบคุมการเข้าถึง ERBAC03

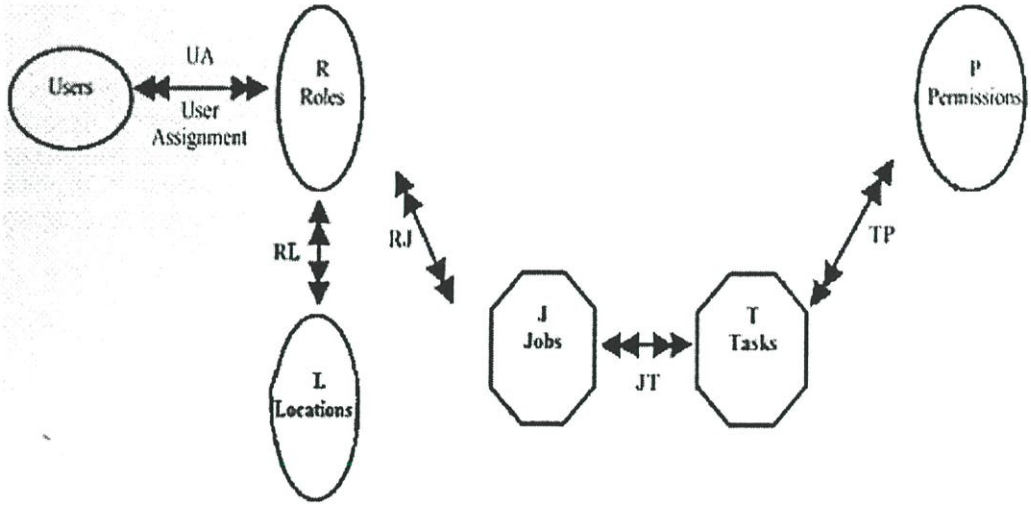
วิทยานิพนธ์เล่มนี้ได้เสนอรูปแบบใหม่ที่พัฒนาต่อจากรูปแบบ RBAC เดิมเรียกว่า ERBAC03 (Enhanced Role-Based Access Control) ซึ่งรูปแบบใหม่นี้ได้เพิ่มคุณสมบัติใหม่ โดยใช้วิธีการลำดับชั้นที่ตั้ง (Location hierarchy approach) เพื่อสนับสนุนองค์กรขนาดใหญ่ที่มีสาขาอยู่มากมาย ซึ่งวิธีการดังกล่าวได้นำข้อดีของรูปแบบการควบคุมการเข้าถึงแบบแมนดาทอรีคือ มีการกำหนดระดับความปลอดภัย (Security level) ให้กับที่ตั้งเพื่อบ่งบอกว่าที่ตั้งใดมีอำนาจเหนือกว่าที่ใด เพื่อใช้เป็นเงื่อนไขในการให้สิทธิ์และการยกเลิกสิทธิ์ของผู้ใช้ต่างที่กันหรือที่ตั้งเดียวกันก็ได้ ซึ่งวิธีการให้สิทธิ์และการยกเลิกสิทธิ์ของผู้ใช้นี้เป็นคุณสมบัติพื้นฐานของรูปแบบการควบคุมการเข้าถึงแบบดีสครีท ซึ่งระดับความปลอดภัยนี้ได้ประยุกต์โดยการพิจารณาจากระดับความสูงต่ำในลำดับชั้นเท่านั้นโดยงานวิจัยฉบับนี้ไม่ได้เน้นถึงการให้สิทธิ์และการยกเลิกสิทธิ์ แต่จะอธิบายถึงหลักการที่เหมาะสมต่อรูปแบบ ERBAC03 เนื่องจากเป็นอีกส่วนหนึ่งที่เกี่ยวข้องกับการควบคุมการเข้าถึงในระดับระยะเวลาดำเนินงาน โดยการควบคุมการเข้าถึงจะแบ่งกระบวนการเป็นสองกระบวนการ ดังรูปที่ 3.1 ซึ่งในส่วนที่หนึ่งเป็นการกำหนดกิจกรรมที่ผู้ใช้สามารถกระทำได้ เรียกว่าส่วนการจัดการการควบคุมการเข้าถึง (Access control administration) และส่วนที่สองจะปรากฏเมื่อมีการใช้งานแอพพลิเคชั่น รวมถึงมีการกำหนดใบอนุญาตที่แท้จริงให้กับผู้ใช้ เรียกว่า ส่วนการควบคุมการเข้าถึงแบบระยะเวลาการดำเนินงาน (Run-Time access control)



รูปที่ 3.1 แสดงการแบ่งส่วนการควบคุมการเข้าถึง

3.1 การนิยามรูปแบบ ERBAC03

รูปแบบการควบคุมการเข้าถึง ERBAC03 ระดับศูนย์ได้มีการกำหนดองค์ประกอบเพิ่มขึ้นคือ ที่ตั้ง (Locations) งาน (Job) และงานย่อย (Task) โดยที่ตั้งหมายถึงลักษณะทางกายภาพขององค์กร เช่นสาขาต่างๆ ขององค์กร เป็นต้น ส่วนงานหมายถึง ประเภทของงาน และงานย่อยหมายถึง ความต้องการที่จะเข้าถึงโปรแกรมประยุกต์เพื่อที่จะนำไปเทียบกับใบอนุญาตที่มีอยู่เพื่อสิทธิ์ในการเข้าถึงทรัพยากรที่ต้องการ รูปแบบ ERBAC03 นี้ไปประยุกต์ใช้กับองค์กรได้อย่างเหมาะสมมากยิ่งขึ้น เพราะองค์กรส่วนใหญ่มีการกำหนดงานที่พนักงานในองค์กรต้องรับผิดชอบ ซึ่งงานในที่นี้จะประกอบด้วยงานย่อยๆมากมาย เพื่อทำให้การปฏิบัติงานของพนักงานสำเร็จลุล่วงไปด้วยดี โดยรูปแบบ RBAC เดิมไม่ได้ให้รายละเอียดของการกำหนดใบอนุญาตให้กับบทบาทชัดเจนเท่าที่ควร และเพื่อให้ระบบที่นำรูปแบบ ERBAC03 ไปประยุกต์ใช้งานได้อย่างดี รูปแบบ ERBAC03 จึงเพิ่มคุณสมบัติข้อบังคับ (Constraint) เพื่อความต้องการทางด้านความคงสภาพของระบบต่อการหลอกลวงของผู้ใช้



รูปที่ 3.2 รูปแบบ ERBAC03 ระดับศูนย์ (ERBAC03$_0$)

จากรูปที่ 3.2 ได้สรุปความแตกต่างระหว่าง RBAC กับ ERBAC03 ระดับศูนย์ดังตารางที่ 3.1 ดังนี้

ตารางที่ 3.1 แสดงความแตกต่างระหว่าง $RBAC_0$ กับ $ERBAC03_0$

$RBAC_0$	$ERBAC03_0$
ผู้ใช้ (Users)	ผู้ใช้ (Users)
บทบาท (Roles)	บทบาท (Roles)
	ที่ตั้ง (Locations)
	งาน (Jobs)
	งานย่อย (Tasks)
ใบอนุญาต (Permissions)	ใบอนุญาต (Permissions)

โดยมีการกำหนดนิยามตามรูปแบบของเอกสาร [41] ให้กับองค์ประกอบในแต่ละส่วนของรูปแบบ ERBAC03 ไว้ดังนี้

นิยาม 3.1.1: สำหรับองค์ประกอบของ ERBAC03

U คือ กลุ่มของผู้ใช้ $\{u_1, u_2, \ldots, u_n\}$

R คือ กลุ่มของบทบาท $\{r_1, r_2, \ldots, r_n\}$

P คือ กลุ่มของใบอนุญาต $\{p_1, p_2, \ldots, p_n\}$

L คือ กลุ่มของที่ตั้ง $\{l_1, l_2, \ldots, l_n\}$

J คือ กลุ่มของงาน $\{j_1, j_2, \ldots, j_n\}$

T คือ กลุ่มของงานย่อย $\{t_1, t_2, \ldots, t_n\}$

นิยาม 3.1.2: สำหรับความสัมพันธ์ของ ERBAC03

$UA \subseteq U \times R$ มีความสัมพันธ์ในการกำหนดบทบาทให้กับผู้ใช้แบบหลายต่อหลาย

$RH \subseteq R \times R$ เป็นระดับของบทบาทเรียกว่า ลำดับชั้นบทบาท (Role hierarchy)

$RL \subseteq L \times R$ มีความสัมพันธ์ในการกำหนดบทบาทให้กับที่ตั้งแบบหลายต่อหลาย

$RJ \subseteq J \times R$ มีความสัมพันธ์ในการกำหนดงานให้กับบทบาทแบบหลายต่อหลาย

$JT \subseteq T \times J$ มีความสัมพันธ์ในการกำหนดงานย่อยให้กับงานแบบหลายต่อหลาย

$TP \subseteq P \times T$ มีความสัมพันธ์ในการกำหนดใบอนุญาตให้กับงานย่อยแบบหลายต่อหลาย

$LH \subseteq L \times L$ เป็นระดับของที่ตั้งเรียกว่า ลำดับชั้นที่ตั้ง (Location hierarchy)

นิยาม 3.1.3: สำหรับหน้าที่ของบทบาทของ ERBAC03 ที่เพิ่มเติมจาก RBAC

roles: $U \cup L \cup J \cup P \rightarrow 2^R$ เป็นฟังก์ชั่นจับคู่อันดับกลุ่มของผู้ใช้ ที่ตั้งและใบอนุญาตให้กับกลุ่มของบทบาท

roles*: $U \cup L \cup J \cup P \rightarrow 2^R$ เป็นการขยายบทบาทเพื่อรองรับลำดับชั้นของบทบาท

roles: $R \rightarrow 2^L$ เป็นฟังก์ชั่นจับคู่อันดับกลุ่มของบทบาทให้กับกลุ่มของที่ตั้ง

$roles(u_i) = \{r \in R \mid (u_i, r) \in UA\}$

$roles(u_i)^* = \{r \in R \mid (\exists r \leq r')[(u_i, r') \in UA]\}$

$roles(l_i) = \{r \in R \mid (l_i, r) \in RL\}$

$roles(j_i) = \{r \in R \mid (j_i, r) \in RJ\}$

$roles(l_i)^* = \{r \in R \mid (\exists r \leq r')[(l_i, r') \in RL]\}$

$roles(j_i)^* = \{r \in R \mid (\exists r' \leq r)[(j_i, r') \in RJ]\}$

หมายเหตุว่านิยามของ roles* สะท้อนให้เห็นถึงการสืบทอดบทบาทที่สัมพันธ์กับผู้ใช้ที่ถ่ายทอดลงมาระดับล่างและสัมพันธ์กับใบอนุญาตที่ถ่ายทอดขึ้นสู่ระดับบน

นิยาม 3.1.4: สำหรับหน้าที่ของที่ตั้งของ ERBAC03

$location(r_i) = \{l \in L \mid (l, r_i) \in RL\}$

$location(u_i) = \{l \in L \mid (\exists r \in roles(u_i))\ [(l, r) \in RL]\}$

$location(j_i) = \{l \in L \mid (\exists r \in roles(j_i))\ [(l, r) \in RL]\}$

$location(p_i) = \{l \in L \mid (\exists r \in roles(jobs(tasks(p_i))))\ [(l, r) \in RL]\}$

$location(r_i)^* = \{l \in L \mid (\exists r' \leq r)[(l, r') \in RL]\}$

$location(u_i)^* = \{l \in L \mid (\exists r \in roles^*(u_i))\ [(l, r) \in RL]\}$

$location(j_i)^* = \{l \in L \mid (\exists r \in roles^*(j_i))\ [(l, r) \in RL]\}$

$location(p_i)^* = \{l \in L \mid (\exists r \in roles^*(jobs(tasks(p_i))))\ [(l, r) \in RL]\}$

locations: $U \cup R \cup J \rightarrow 2^L$ เป็นฟังก์ชั่นจับคู่อันดับกลุ่มของผู้ใช้ บทบาทและงานให้กับกลุ่มของที่ตั้ง

locations*: $U \cup R \cup J \rightarrow 2^L$ เป็นการขยายที่ตั้งเพื่อรองรับลำดับชั้นที่ตั้ง

นิยาม 3.1.5: สำหรับหน้าที่ของงานของ ERBAC03

jobs: $J \rightarrow 2^T$ เป็นฟังก์ชั่นจับคู่อันดับกลุ่มของงานให้กับกลุ่มของงานย่อย

jobs*: $J \rightarrow 2^T$ เป็นการขยายงานเพื่อรองรับลำดับชั้นบทบาทหรือที่ตั้ง

$jobs(t_i) = \{j \in J \mid (t_i, j) \in JT\}$

นิยาม 3.1.6: สำหรับหน้าที่ของงานย่อยของ ERBAC03

tasks: $T \rightarrow 2^P$ เป็นฟังก์ชั่นจับคู่อันดับกลุ่มของงานย่อยให้กับกลุ่มของใบอนุญาต

tasks*: $T \rightarrow 2^P$ เป็นการขยายงานเพื่อรองรับลำดับชั้นบทบาทหรือที่ตั้ง

$tasks(p_i) = \{t \in T \mid (p_i, t) \in TP\}$

นิยาม 3.1.7: สำหรับหน้าที่ของใบอนุญาตของ ERBAC03

perm: $U \cup R \cup L \cup J \cup T \rightarrow 2^P$ เป็นฟังก์ชั่นจับคู่อันดับกลุ่มของผู้ใช้ บทบาท สถานที่งาน และงานย่อยให้กับกลุ่มของใบอนุญาต

perm*: $U \cup R \cup L \cup J \cup T \rightarrow 2^P$, เป็นการขยายใบอนุญาตเพื่อรองรับลำดับชั้นบทบาทหรือที่ตั้ง

$perm(u_i) = \{p \in P \mid (\exists t \in tasks(jobs(roles(u_i)))) [(p, t) \in TP]\}$

$perm(u_i)^* = \{p \in P \mid (\exists t \in tasks^*(jobs^*(roles^*(u_i)))) [(p, t) \in TP]\}$

$perm(l_i) = \{p \in P \mid (\exists t \in tasks(jobs(roles(l_i)))) [(p, t) \in TP]\}$

$perm(j_i) = \{p \in P \mid (\exists t \in tasks(j_i)) [(p, t) \in TP]\}$

$perm(l_i)^* = \{p \in P \mid (\exists t \in tasks^*(jobs^*(roles^*(l_i)))) [(p, t) \in TP]\}$

$perm(j_i)^* = \{p \in P \mid (\exists t \in tasks^*(j_i)) [(p, r) \in TP]\}$

3.2 การกำหนดความสัมพันธ์ระหว่างเอนทิที

3.2.1 ผู้ใช้กับบทบาท

ในส่วนแรกนี้ได้มีการกำหนดบทบาทที่มีอยู่เดิมให้กับผู้ใช้ โดยมีความสัมพันธ์เป็นแบบหลายต่อหลาย (many to many relation) ซึ่งสามารถระบุประเภทของผู้ใช้ได้ 2 ประเภท คือ มนุษย์และโปรแกรม ส่วนบทบาทสามารถพิจารณาได้จากเซตของความชำนาญ ระดับการศึกษา และประสบการณ์ ซึ่งเมื่อระบุประเภทผู้ใช้และบทบาทชัดเจนแล้ว จึงสามารถกำหนดบทบาทให้กับผู้ใช้ได้ โดยทั้งนี้ต้องมีการกำหนดรายละเอียดของบทบาทให้ครบถ้วนเสียก่อน

3.2.2 ที่ตั้งกับบทบาท

การกำหนดบทบาทให้กับที่ตั้งมีความสัมพันธ์แบบหลายต่อหลาย (many to many relation) ในงานวิจัยฉบับนี้ได้ให้คำจำกัดความของที่ตั้งคือสถานที่ที่ในแต่ละองค์กรสร้างขึ้นไว้สำหรับดำเนินกิจการให้บรรลุวัตถุประสงค์ขององค์กร ซึ่งในแต่ละที่ตั้งจะมีกลุ่มของบทบาทที่จะร่วมทำกิจกรรมต่างๆตามนโนบายที่องค์กรได้กำหนดไว้ เช่น ที่ตั้งที่กรุงเทพฯ ต้องมีหัวหน้าไปรษณีย์เป็นหัวหน้า มีพนักงานบัญชีคอยดูแลเรื่องเงิน เป็นต้น ซึ่งที่ตั้งอื่นๆ ก็สามารถมีบทบาทแบบเดียวกับกรุงเทพได้เช่นกัน

3.2.3 บทบาทกับงาน

การกำหนดงานให้กับบทบาทมีความสัมพันธ์แบบ หลายต่อหลาย (many to many relation) โดยในแต่ละบทบาทต้องมีการกำหนดหน้าที่ความรับผิดชอบ ตัวอย่างเช่น บทบาทหัวหน้าไปรษณีย์ต้องมีหน้าที่ปิดบัญชีสิ้นวันทำการและตรวจสอบการเงินของที่ทำการ ซึ่งหน้าที่ความรับผิดชอบนี้คือ งาน (Job) โดยการกำหนดหน้าที่ความรับผิดชอบในแต่ละบทบาทนี้จะถูกกำหนดขึ้นโดยองค์กร เนื่องจากรูปแบบ RBAC เดิมได้มีการกำหนดใบอนุญาตให้กับบทบาทโดยตรง ซึ่งทำให้ไม่สามารถบอกได้ว่าบทบาทที่ผู้ใช้ได้รับนั้นมีหน้าที่ความรับผิดชอบอะไรบ้าง จึงทำให้การกำหนดใบอนุญาตให้กับบทบาทไม่ครบถ้วน

3.2.4 งานกับงานย่อย

การกำหนดงานย่อยให้กับงานมีความสัมพันธ์แบบหลายต่อหลาย (many to many relation) โดยงานย่อยหมายถึงรายละเอียดของหน้าที่ความรับผิดชอบที่นำไปสู่ใบอนุญาตที่แท้จริง ซึ่งงานย่อยเป็นเสมือนกลุ่มของใบอนุญาตนั่นเอง ตัวอย่างเช่น หน้าที่การปิดบัญชีสิ้นวัน ต้องมีงานย่อยที่ทำคือ เตรียมข้อมูลการรับฝากทั้งหมด ณ สิ้นวันทำการ สรุปยอดรับจ่ายเงิน ตรวจสอบเงินสดที่มีอยู่ในคลัง เป็นต้น

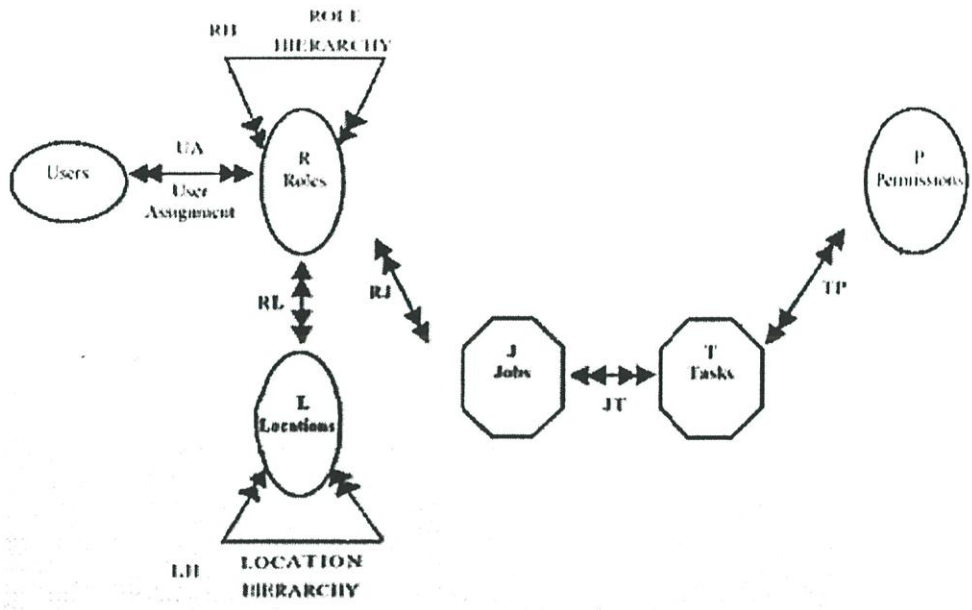
3.2.5 งานย่อยกับใบอนุญาต

ส่วนสุดท้ายเป็นการกำหนดงานย่อยให้กับใบอนุญาตมีความสัมพันธ์แบบหลายต่อหลาย (many to many relation) เมื่อกำหนดใบอนุญาตให้กับงานย่อยแล้วทำให้ทราบว่างานย่อยนั้นกระทำกิจกรรมที่แท้จริงใดๆ ต่อระบบฯ ตัวอย่างเช่น งานย่อยที่ทำคือ เตรียมข้อมูลการรับฝากทั้งหมด ณ สิ้นวันทำการ ต้องมีใบอนุญาตในการอ่านระเบียนข้อมูลการทำรายการ (Transaction) เป็นต้น

3.3 วิธีการลำดับชั้นที่ตั้ง (Location hierarchy approach)

เป็นวิธีการที่นำมาใช้ร่วมกับรูปแบบ ERBAC03 ระดับศูนย์ เพื่อพัฒนาให้เป็นรูปแบบ ERBAC03 ระดับหนึ่ง ซึ่งรูปแบบ ERBAC03 ระดับหนึ่งแตกต่างจากรูปแบบ RBAC ระดับหนึ่งคือได้เพิ่มลำดับชั้นที่ตั้ง (Location Hierarchy) และได้คงลำดับชั้นบทบาทไว้ด้วยดังรูปที่ 3.3

ความแตกต่างระหว่างลำดับชั้นที่ตั้งกับลำดับชั้นบทบาทอยู่ที่ลักษณะการถ่ายทอด เนื่องจากลำดับชั้นที่ตั้งมีการถ่ายทอดบทบาทจากบนลงล่าง แต่ลำดับชั้นบทบาทเป็นการถ่ายทอดสิทธิ์จากล่างสูบน จึงจำเป็นอย่างยิ่งต่อการแยกลำดับชั้นทั้งสองออกจากกันให้มีความชัดเจนที่สุด อีกทั้งลำดับชั้นทั้งสองยังคงมีความสัมพันธ์ที่สอดคล้องกันด้วย



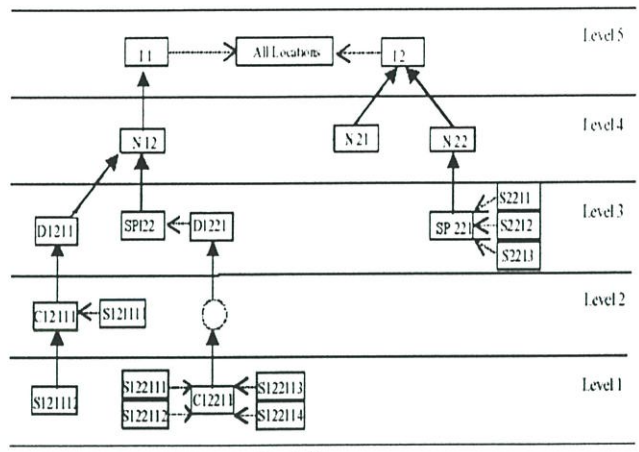
รูปที่ 3.3 รูปแบบ ERBAC03 ระดับหนึ่ง (ERBAC03$_1$)

โดยให้ที่ตั้งที่มีอำนาจมากที่สุดอยู่ด้านบนสุดของลำดับชั้น และให้ที่ตั้งที่มีอำนาจรองลงมาอยู่ตามกิ่งลดหลั่นกัน ซึ่งเสมือนเป็นการระบุระดับความปลอดภัย (Security level) ให้กับลำดับชั้นไปในตัวด้วย โดยลำดับชั้นที่ตั้งนี้จะแสดงถึงโครงสร้างขององค์กรขนาดใหญ่ที่มีอยู่หลายสาขา ซึ่งแต่ละสาขามีอำนาจหน้าที่แตกต่างกันไป ซึ่งได้แสดงตัวอย่างประเภทของที่ตั้งไว้ในตารางที่ 3.2

ตารางที่ 3.2 แสดงตัวอย่างประเภทที่ตั้งของไปรษณีย์ไทย

สถานที่	ประเภท	สถานที่	ประเภท
ไปรษณีย์ระหว่างประเทศ	I	ไปรษณีย์ท้องถิ่น	D
ไปรษณีย์ประเทศไทย	N	ไปรษณีย์เอกชน	C
ไปรษณีย์ภูมิภาค	SP	เคาน์เตอร์	S

ประเภทของที่ตั้งถูกนำไปใช้ในการสร้างลำดับชั้นที่ตั้ง ดังรูปที่ 3.4



รูปที่ 3.4 ลำดับชั้นที่ตั้ง (Location hierarchy)

นิยาม 3.3.1: สำหรับการสร้างลำดับชั้นที่ตั้ง

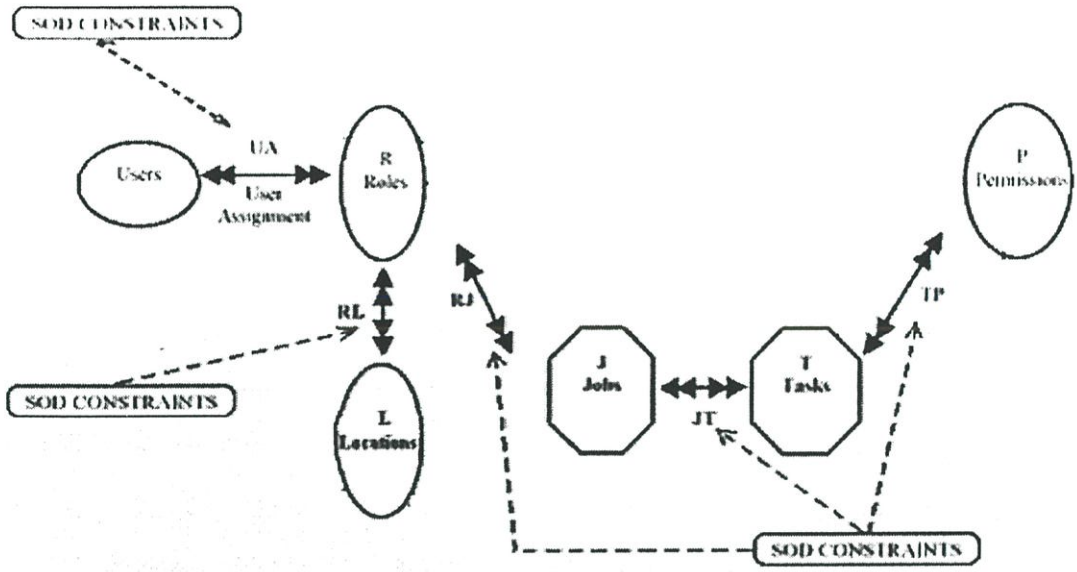
ขั้นแรก: กำหนดโดเมนหลักเริ่มแรกก่อน เช่น ทวีปเอเชีย

ขั้นสอง: กำหนดโดเมนย่อยที่สืบทอดอำนาจมาจากโดเมนหลัก

ขั้นสาม: กำหนดโดเมนย่อยอื่นๆเพิ่มเติมภายในโดเมนหลัก ถ้าไม่มีการเพิ่มโดเมนย่อยอีก ก็ให้ถือว่าลำดับชั้นของสถานที่เสร็จสมบูรณ์

3.4 การกำหนดคุณสมบัติข้อบังคับ (Constraint)

รูปแบบ ERBAC03 เมื่อมีการใช้ข้อบังคับร่วมกับรูปแบบจะทำให้รูปแบบ ERBAC03 พัฒนาจากระดับศูนย์เป็นระดับสองทันที ดังรูปที่ 3.5 มีการใช้หลักการแบ่งแยกหน้าที่แบบสแตติก (Static Separation of Duty) ซึ่งจะอยู่ในส่วนการจัดการการควบคุมการเข้าถึง โดยจำกัดให้ผู้ใช้มีสิทธิ์เท่าที่จำเป็นต่องานเท่านั้น โดยผู้ใช้หนึ่งคนจะเป็นสมาชิกของสองบทบาทที่ขัดแย้งกันไม่ได้ ซึ่งบทบาทที่ขัดแย้งกันนี้ เช่น บทบาทพนักงานบัญชีกับบทบาทผู้ตรวจสอบบัญชี ถ้าผู้ใช้เป็นสมาชิกทั้งสองบทบาท อาจทำให้เกิดการหลอกลวงของผู้ใช้ขึ้นได้ จึงจำเป็นต้องให้ผู้ใช้มีอำนาจไม่มากจนเกินไป โดยอำนาจในที่นี้คือการมีใบอนุญาต (permissions)



รูปที่ 3.5 รูปแบบ ERBAC03 ระดับสอง (ERBAC03$_2$)

ดังนั้นส่วนประกอบที่สำคัญของตัวอย่างความขัดแย้ง มีการกำหนดนิยามไว้ดังนี้

นิยาม 3.4.1: **ใบอนุญาตที่ขัดแย้งกัน** (Conflicting permissions) คือใบอนุญาตแต่ละใบเป็นสมาชิกในเซตใบอนุญาตที่ขัดแย้งก็ต่อเมื่อใบอนุญาตแต่ละใบเป็นสมาชิกในเซตใบอนุญาตที่ขัดแย้ง โดยที่ใบอนุญาตใบเดียวกันไม่เป็นสมาชิกในเซตใบอนุญาตที่ขัดแย้ง โดยนำเสนอในรูปแบบดังนี้ $CP \subseteq P \times P$ มีความสัมพันธ์แบบหลายต่อหลาย ชี้ให้เห็นถึงความขัดแย้งระหว่างใบอนุญาต พร้อมกับกำหนดรูปแบบคือ

$$(p_i, p_j) \in CP \leftrightarrow (p_j, p_i) \in CP, (p_i, p_i) \notin CP$$

ซึ่งสามารถนำเสนอตามสัจพจน์ที่นำเสนอเงื่อนไขความปลอดภัยขั้นพื้นฐาน

เงื่อนไขความปลอดภัยขั้นพื้นฐาน 1: ใบอนุญาตที่ขัดแย้งกันจะไม่ถูกกำหนดให้ผู้ใช้หรืองานย่อย โดยมีรูปแบบดังนี้

$$(perm^*(u_i) \times perm^*(u_j)) \cap CP = \varnothing \text{ และ}$$

$$(perm^*(t_i) \times perm^*(t_j)) \cap CP = \varnothing$$

เนื่องจากใบอนุญาตที่ไม่ขัดแย้งกัน ไม่มีผลกับเงื่อนไขความปลอดภัยขั้นพื้นฐาน 1 จึงได้มีการกำหนดสัจพจน์เพิ่มเติมในรูปแบบเงื่อนไขความปลอดภัยขั้นพื้นฐาน

สัจพจน์ 3.4.1: ใบอนุญาตที่ไม่ขัดแย้งกันถูกกำหนดได้ทั้งในบทบาทหรืองานย่อยที่ขัดแย้งกันและกำหนดให้บทบาทหรืองานย่อยที่ไม่ขัดแย้งกันก็ได้

นิยาม 3.4.2: **ผู้ใช้ที่ขัดแย้งกัน** (Conflicting users) คือผู้ใช้แต่ละคนเป็นสมาชิกในเซตผู้ใช้ที่ขัดแย้งกันก็ต่อเมื่อผู้ใช้แต่ละคนเป็นสมาชิกในเซตผู้ใช้ที่ขัดแย้งกัน โดยที่ผู้ใช้คนเดียวกันไม่เป็นสมาชิกในเซตผู้ใช้ที่ขัดแย้งกัน โดยนำเสนอในรูปแบบดังนี้

$CU \subseteq U \times U$ มีความสัมพันธ์แบบหลายต่อหลาย ชี้ให้เห็นถึงความขัดแย้งระหว่างผู้ใช้ พร้อมกับกำหนดรูปแบบคือ

$$(u_i, u_j) \in CU \leftrightarrow (u_j, u_i) \in CU, (u_i, u_i) \notin CU$$

สัจพจน์ 3.4.2: ผู้ใช้ที่ขัดแย้งกันจะพิจารณาคล้ายกับผู้ใช้คนเดียวกันได้

ในเงื่อนไขของผู้ใช้ที่ขัดแย้งกันอาจเป็นสมาชิกในครอบครัวหรือบุคคลที่ถูกมองว่าเป็นผู้ร่วมประสงค์ร้ายก็ได้

นิยาม 3.4.3: **บทบาทที่ขัดแย้งกัน** (Conflicting roles) คือบทบาทสองบทบาทจะเป็นสมาชิกในเซตบทบาทที่ขัดแย้งกันก็ต่อเมื่อบทบาทสองบทบาทนั้นเป็นสมาชิกในเซตบทบาทที่ขัดแย้งกัน โดยที่บทบาทเดียวกันไม่เป็นสมาชิกในเซตบทบาทที่ขัดแย้ง และ ถ้าบทบาทสองบทบาทนั้นเป็นสมาชิกในเซตบทบาทที่ขัดแย้งกันแล้วใบอนุญาตของแต่ละบทบาทนั้นเป็นสมาชิกในเซตใบอนุญาตที่ขัดแย้งกัน หรือ ถ้าบทบาทสองบทบาทนั้นเป็นสมาชิกในเซตบทบาทที่ขัดแย้งกันแล้วงานของแต่ละบทบาทนั้นเป็นสมาชิกในเซตงานที่ขัดแย้งกัน หรือถ้าบทบาทสองบทบาทนั้นเป็นสมาชิกในเซตบทบาทที่ขัดแย้งกันแล้วที่ตั้งของแต่ละบทบาทนั้นเป็นสมาชิกในเซตที่ตั้งที่ขัดแย้งกัน ซึ่งถ้านำมาใช้ร่วมกันสามารถทำให้เกิดการหลอกลวงขึ้น ตัวอย่างเช่น บทบาทสองบทบาทนั้นถูกกำหนดให้มีใบอนุญาตบางใบที่ขัดแย้งกัน โดยบทบาททั้งสองถูกนำเสนอด้วย $CR \subseteq R \times R$ มีความสัมพันธ์แบบหลายต่อหลาย ชี้ให้เห็นถึงความขัดแย้งระหว่างบทบาท โดยนำเสนอในรูปแบบดังนี้

$$(r_i, r_j) \in CR \leftrightarrow (r_j, r_i) \in CR, (r_i, r_i) \notin CR \text{ และ}$$

$$\{(r_i, r_j) \in CR \rightarrow (perm^*(r_i) \times perm^*(r_j)) \cap CP \neq \varnothing \text{ หรือ}$$

$$(r_i, r_j) \in CR \rightarrow (job^*(r_i) \times job^*(r_j)) \cap CJ \neq \varnothing \text{ หรือ}$$

$$(r_i, r_j) \in CR \rightarrow (location^*(r_i) \times location^*(r_j)) \cap CL \neq \varnothing \}$$

ซึ่งสามารถนำเสนอตามสัจพจน์ที่นำเสนอเงื่อนไขความปลอดภัยขั้นพื้นฐาน

เงื่อนไขความปลอดภัยขั้นพื้นฐาน 2: บทบาทที่ขัดแย้งกันจะไม่ถูกกำหนดให้ผู้ใช้หรือที่ตั้ง โดยมีรูปแบบดังนี้

$$(role^*(u_i) \times role^*(u_i)) \cap CR = \varnothing \text{ และ}$$

$$(role^*(l_i) \times role^*(l_j)) \cap CR = \varnothing$$

เนื่องจากบทบาทที่ไม่ขัดแย้งกัน ไม่มีผลกับเงื่อนไขความปลอดภัยขั้นพื้นฐาน 2 จึงได้มีการกำหนดสัจพจน์เพิ่มเติมในรูปแบบเงื่อนไขความปลอดภัยขั้นพื้นฐาน

สัจพจน์ 3.4.3: บทบาทที่ไม่ขัดแย้งกันจะกำหนดให้ผู้ใช้หรือที่ตั้งที่ไม่ขัดแย้งกันและกำหนดให้ผู้ใช้หรือที่ตั้งที่ขัดแย้งกันก็ได้

โดยมองเสมือนว่าบทบาทนั้นง่ายในการจัดการไม่มีความซับซ้อน เนื่องจากบทบาทที่ขัดแย้งต้องมีใบอนุญาตบางใบที่ขัดแย้งหรือบทบาทที่ขัดแย้งต้องมีงานบางงานที่ขัดแย้งกันซึ่งสามารถกล่าวได้ว่า บทบาทที่ไม่ขัดแย้งนั้นจะไม่มีใบอนุญาตที่ขัดแย้งหรือไม่มีงานที่ขัดแย้งกันด้วย

นิยาม 3.4.4: **งานที่ขัดแย้งกัน** (Conflicting jobs) เป็นงาน ซึ่งงานสองงานที่นำมาใช้ร่วมกันนั้นสามารถทำให้เกิดการหลอกลวงขึ้น โดยงานที่ขัดแย้งกันถูกนำเสนอด้วย $CJ \subseteq J \times J$ มีความสัมพันธ์แบบหลายต่อหลาย ชี้ให้เห็นถึงความขัดแย้งระหว่างงาน โดยนำเสนอในรูปแบบดังนี้

$$(j_i, j_j) \in CJ \leftrightarrow (j_j, j_i) \in CJ, (j_i, j_i) \notin CJ \text{ และ}$$

$$\{(j_i, j_j) \in CJ \rightarrow (task^{\bullet}(j_i) \times task^{\bullet}(j_j)) \cap CT \neq \varnothing \text{ หรือ}$$

$$(j_i, j_j) \in CJ \rightarrow (role^{\bullet}(j_i) \times role^{\bullet}(j_j)) \cap CR \neq \varnothing\}$$

ซึ่งสามารถนำเสนอตามสัจพจน์ที่นำเสนอเงื่อนไขความปลอดภัยขั้นพื้นฐาน

เงื่อนไขความปลอดภัยขั้นพื้นฐาน 3: งานที่ขัดแย้งกันอาจไม่ถูกกำหนดให้บทบาท โดยมีรูปแบบดังนี้

$$(job^{\bullet}(r_i) \times job^{\bullet}(r_j)) \cap CJ = \varnothing$$

เนื่องจากงานที่ไม่ขัดแย้งกัน ไม่มีผลกับเงื่อนไขความปลอดภัยขั้นพื้นฐาน 3 จึงได้มีการกำหนดสัจพจน์เพิ่มเติมในรูปแบบเงื่อนไขความปลอดภัยขั้นพื้นฐาน

สัจพจน์ 3.4.4: งานที่ไม่ขัดแย้งกันถูกกำหนดได้ทั้งในบทบาทที่ขัดแย้งกันหรือบทบาทที่ไม่ขัดแย้งกันก็ได้

นิยาม 3.4.5: **งานย่อยที่ขัดแย้งกัน** (Conflicting tasks) เป็นงานย่อยที่ต้องการใบอนุญาตที่ขัดแย้งกันอย่างสมบูรณ์ โดยนำเสนอในรูปแบบคือ $CT \subseteq T \times T$ มีความสัมพันธ์แบบหลายต่อหลาย ชี้ให้เห็นถึงความขัดแย้งระหว่างงานย่อย โดยนำเสนอในรูปแบบดังนี้

$$(t_i, t_j) \in CT \leftrightarrow (t_j, t_i) \in CT, (t_i, t_i) \notin CT \text{ และ}$$

$$(t_i, t_j) \in CT \rightarrow (perm^{\bullet}(t_i) \times perm^{\bullet}(t_j)) \cap CP \neq \varnothing$$

ซึ่งสามารถนำเสนอตามสัจพจน์ที่นำเสนอเงื่อนไขความปลอดภัยขั้นพื้นฐาน

เงื่อนไขความปลอดภัยขั้นพื้นฐาน 4: งานย่อยที่ขัดแย้งกันอาจไม่ถูกกำหนดให้งาน โดยมีรูปแบบดังนี้

$$(task^{\bullet}(j_i) \times task^{\bullet}(j_j)) \cap CT = \varnothing$$

เนื่องจากงานย่อยที่ไม่ขัดแย้งกัน ไม่มีผลกับเงื่อนไขความปลอดภัยขั้นพื้นฐาน 4 จึงได้มีการกำหนดสัจพจน์เพิ่มเติมในรูปแบบเงื่อนไขความปลอดภัยขั้นพื้นฐาน

สัจพจน์ 3.4.5: งานย่อยที่ไม่ขัดแย้งกันอาจกำหนดให้กับงานที่ขัดแย้งกันหรืองานที่ไม่ขัดแย้งกันก็ได้

นิยาม 3.4.6: **ที่ตั้งที่ขัดแย้งกัน** (Conflicting locations) เป็นที่ตั้งที่ต้องการบทบาทที่ขัดแย้งกันอย่างสมบูรณ์ โดยนำเสนอในรูปแบบคือ $CL \subseteq L \times L$ มีความสัมพันธ์แบบหลายต่อหลายชี้ให้เห็นถึงความขัดแย้งระหว่างที่ตั้ง โดยนำเสนอในรูปแบบดังนี้

$$(l_i, l_j) \in CL \leftrightarrow (l_j, l_i) \in CL, (l_i, l_i) \notin CL \text{ และ}$$

$$(l_i, l_j) \in CL \rightarrow (role^*(l_i) \times role^*(l_j)) \cap CR \neq \varnothing$$

ซึ่งสามารถนำเสนอตามสัจพจน์ที่นำเสนอเงื่อนไขความปลอดภัยขั้นพื้นฐาน

เงื่อนไขความปลอดภัยขั้นพื้นฐาน 5: ที่ตั้งที่ขัดแย้งกันอาจไม่ถูกกำหนดให้บทบาทและผู้ใช้ โดยมีรูปแบบดังนี้

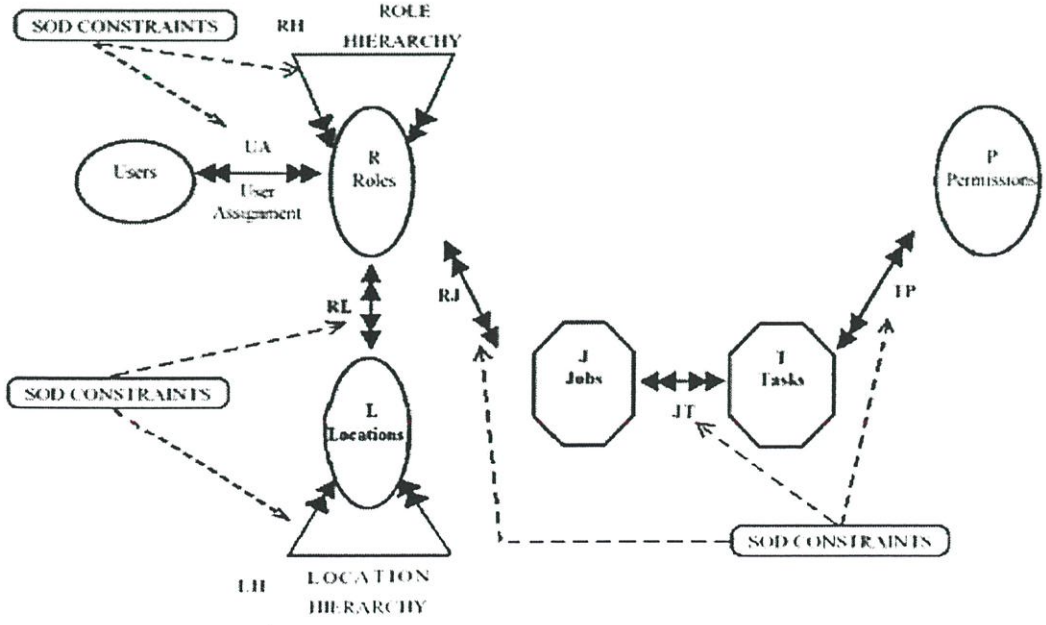
$$(location^*(r_i) \times location^*(r_j)) \cap CL = \varnothing \text{ และ}$$

$$(location^*(u_i) \times location^*(u_i)) \cap CL = \varnothing$$

เนื่องจากที่ตั้งที่ไม่ขัดแย้งกัน ไม่มีผลกับเงื่อนไขความปลอดภัยขั้นพื้นฐาน 5 จึงได้มีการกำหนดสัจพจน์เพิ่มเติมในรูปแบบเงื่อนไขความปลอดภัยขั้นพื้นฐาน

สัจพจน์ 3.4.6: ที่ตั้งที่ไม่ขัดแย้งกันอาจกำหนดให้กับบทบาทหรือผู้ใช้ที่ขัดแย้งกันหรือกำหนดให้กับบทบาทหรือผู้ใช้ที่ไม่ขัดแย้งกันก็ได้

จากหลักการการแบ่งแยกหน้าที่สามารถเขียนนิยามเพื่อปรับปรุงรูปแบบ ERBAC03 ให้มีประสิทธิภาพในด้านการกำหนดใบอนุญาตให้กับบทบาทได้อย่างเหมาะสมโดยจะพิจารณาจากที่ตั้งด้วย ซึ่งเมื่อรูปแบบ ERBAC03 มีการใช้วิธีการลำดับชั้นของที่ตั้งและข้อบังคับร่วมกัน ทำให้ได้รูปแบบ ERBAC03 ระดับสามดังรูปที่ 3.6



รูปที่ 3.6 รูปแบบ ERBAC03 ระดับสาม $ERBAC03_3$)

3.5 การกำหนดความต้องการความคงสภาพของรูปแบบ ERBAC03

ในหัวข้อนี้จะนำนิยามที่ได้กำหนด มาพิสูจน์ความถูกต้องด้านความคงสภาพของรูปแบบการควบคุมการเข้าถึงแบบโรลเบสแบบใหม่ เพื่อสนับสนุนการออกแบบเครื่องมือการบริหารจัดการความปลอดภัยให้มีประสิทธิภาพยิ่งขึ้น โดยเสนอเหตุการณ์ที่เป็นไปได้ที่สะท้อนให้เห็นถึงความต้องการทางด้านความคงสภาพของระบบที่ประยุกต์ใช้รูปแบบ ERBAC03

เหตุการณ์ที่ 3.5.1 ภายใต้เงื่อนไขความปลอดภัยขั้นพื้นฐาน 5 ผู้ใช้ที่ไม่ขัดแย้งกันจะถูกกำหนดให้กับบทบาทที่ขัดแย้งกันได้ โดยแปลงความหมายเป็นสมการทางคณิตศาสตร์เป็น ถ้าผู้ใช้สองคนมีบทบาทที่แตกต่างกันและแต่ละบทบาทอยู่ในเซตบทบาทที่ขัดแย้งกันแล้วผู้ใช้สองคนนั้นจะไม่อยู่ในเซตผู้ใช้ที่ขัดแย้งกัน

$$(u_i, r_k) \in UA \wedge (u_j, r_l) \in UA \wedge (r_k, r_l) \in CR \rightarrow (u_i, u_j) \notin CU$$

<u>การพิสูจน์</u> โดยใช้ Contradiction Proof

สมมุติให้ $(u_i, r_k) \in UA \wedge (u_j, r_l) \in UA \wedge (r_k, r_l) \in CR \wedge (u_i, u_j) \in CU$

จากนิยาม 3.1.4 จะได้

$$location^*(u_i) \supseteq location^*(r_k)$$

$$location^*(u_j) \supseteq location^*(r_l)$$

โดยที่ $(r_k, r_l) \in CR$

ได้ตามนิยาม 3.4.3

$$location^*(r_i) \times location^*(r_j) \cap CL \neq \varnothing$$

จึงทำให้ได้ผลลัพธ์คือ

$$(location^*(u_i) \times location^*(u_j)) \cap CL \neq \varnothing$$ ซ.ต.พ.

เหตุการณ์ที่ 3.5.2 ภายใต้เงื่อนไขความปลอดภัยขั้นพื้นฐาน 5 งานที่ขัดแย้งกันจะถูกกำหนดให้กับบทบาทที่ขัดแย้งกันได้ โดยแปลงความหมายเป็นสมการทางคณิตศาสตร์ ถ้างานสองงานถูกกำหนดให้กับบทบาทที่แตกต่างกันและแต่ละงานเป็นสมาชิกของเซตงานที่ขัดแย้งกันแล้วบทบาทสองบทบาทนั้นจะเป็นสมาชิกของเซตบทบาทที่ขัดแย้งกัน

$$(j_i, r_k) \in RJ \wedge (j_j, r_l) \in RJ \wedge (j_i, j_j) \in CJ \rightarrow (r_k, r_l) \in CR$$

<u>การพิสูจน์</u> โดยใช้ Contradiction Proof

สมมุติให้ $(j_i, r_k) \in RJ \wedge (j_j, r_l) \in RJ \wedge (j_i, j_j) \in CJ \wedge (r_k, r_l) \notin CR$

จากนิยาม 3.1.4 จะได้

$$location^*(r_k) \supseteq location^*(j_i)$$

$$location^*(r_l) \supseteq location^*(j_j)$$

โดยที่ $(j_i, j_j) \in CJ$

จากนิยาม 3.4.4 แสดงว่า

$$(role^{*}(j_i) \times role^{*}(j_j)) \cap CR \neq \varnothing$$

จึงทำให้ได้ผลลัพธ์คือ

$$(location^{*}(r_k) \times location^{*}(r_l)) \cap CL \neq \varnothing$$ ซ.ต.พ.

เหตุการณ์ที่ 3.5.3 ภายใต้เงื่อนไขความปลอดภัยขั้นพื้นฐาน 5 งานย่อยที่ขัดแย้งกันจะถูกกำหนดให้กับงานที่ขัดแย้งกันได้ โดยแปลงความหมายเป็นสมการทางคณิตศาสตร์ ถ้างานย่อยสองงานย่อยถูกกำหนดให้กับงานที่แตกต่างกันและแต่ละงานย่อยเป็นสมาชิกของเซตงานย่อยที่ขัดแย้งกัน แล้วงานสองงานนั้นจะเป็นสมาชิกของเซตงานที่ขัดแย้งกัน

$$(t_i, j_k) \in JT \wedge (t_j, j_l) \in JT \wedge (t_i, t_j) \in CT \rightarrow (j_k, j_l) \in CJ$$

<u>การพิสูจน์</u> โดยใช้ Contradiction Proof

สมมุติให้ งานย่อย t_i กับ t_j ที่ขัดแย้งกัน ถูกกำหนดให้กับงานที่ไม่ขัดแย้งกัน

$$(t_i, j_k) \in JT \wedge (t_j, j_l) \in JT \wedge (t_i, t_j) \in CT \wedge (j_k, j_l) \notin CJ$$

โดยสมมุติเลือกผู้ใช้ u_x ให้สัมพันธ์กับ บทบาท r_m กับ r_n โดย

$$(r_m, r_n) \notin CR$$

ดังนั้นได้ $(u_x, r_m) \in RJ \wedge (u_x, r_n) \in RJ \wedge (r_m, j_k) \in RJ \wedge (r_n, j_l) \in RJ \wedge (t_i, j_k) \in JT \wedge$

$(t_j, j_l) \in JT$

โดยงานย่อยขัดแย้งกัน $(t_i, t_j) \in CT$ จะทำให้งานขัดแย้งกันด้วย

จากนิยาม 3.4.4

$$(task^{*}(j_k) \times task^{*}(j_l)) \cap CT \neq \varnothing$$

เมื่องานขัดแย้งทำให้บทบาทขัดแย้งด้วย

จากนิยาม 3.4.3 ได้

$$(job^{*}(r_i) \times job^{*}(r_j)) \cap CJ \neq \varnothing$$

จึงทำให้ได้ผลลัพธ์คือ

$$(location^{*}(r_k) \times location^{*}(r_l)) \cap CL \neq \varnothing$$ ซ.ต.พ.

เหตุการณ์ที่ 3.5.4 ภายใต้เงื่อนไขความปลอดภัยขั้นพื้นฐาน 5 ใบอนุญาตที่ขัดแย้งกันจะถูกกำหนดให้กับงานย่อยที่ขัดแย้งกันได้ โดยแปลงความหมายเป็นสมการทางคณิตศาสตร์ ถ้าใบอนุญาตสองใบอนุญาตถูกกำหนดให้กับงานย่อยที่แตกต่างกันและแต่ละใบอนุญาตเป็นสมาชิกของเซตใบอนุญาตที่ขัดแย้งกันแล้วงานสองงานย่อยนั้นจะเป็นสมาชิกของเซตงานย่อยที่ขัดแย้งกัน

$$(p_i, t_k) \in TP \wedge (p_j, t_l) \in TP \wedge (p_i, p_j) \in CP \rightarrow (t_k, t_l) \in CT$$

<u>การพิสูจน์</u> โดยใช้ Contradiction Proof

สมมุติให้ $(p_i, t_k) \in TP \wedge (p_j, t_l) \in TP \wedge (p_i, p_j) \in CP \wedge (t_k, t_l) \notin CT$

จากนิยาม 3.1.4 จะได้

$$location^*(p_i) \subseteq location^*(t_k)$$

$$location^*(p_j) \subseteq location^*(t_l)$$

โดยที่ $(p_i, p_j) \in CP$ ทำให้ $(t_k, t_l) \in CT$ จากนิยาม 3.4.5 ได้

$$(perm^*(t_i) \times perm^*(t_j)) \cap CP \neq \varnothing$$

จากนิยาม 3.4.4 ได้

$$(task^*(j_i) \times task^*(j_j)) \cap CT \neq \varnothing$$

จากนิยาม 3.4.3 ได้

$$(job^*(r_i) \times job^*(r_j)) \cap CJ \neq \varnothing$$

จากนิยาม 3.4.6 จึงทำให้ได้ผลลัพธ์คือ

$$(role^*(l_i) \times role^*(l_j)) \cap CR \neq \varnothing$$ ซ.ต.พ.

เหตุการณ์ที่ 3.5.5 ภายใต้เงื่อนไขความปลอดภัยขั้นพื้นฐาน 5 บทบาทที่ขัดแย้งกันอาจจะถูกกำหนดให้กับที่ตั้งที่ขัดแย้งกันก็ได้ โดยแปลงความหมายเป็นสมการทางคณิตศาสตร์ ถ้าบทบาทสองบทบาทถูกกำหนดให้กับที่ตั้งที่แตกต่างกันและแต่ละบทบาทเป็นสมาชิกของเซตบทบาทที่ขัดแย้งกันแล้วที่ตั้งสองที่ตั้งนั้นจะเป็นสมาชิกของเซตที่ตั้งที่ขัดแย้งกัน

$$(r_i, l_k) \in RL \wedge (r_j, l_l) \in RL \wedge (r_i, r_j) \in CR \rightarrow (l_k, l_l) \in CL$$

<u>การพิสูจน์</u> โดยใช้ Contradiction Proof

สมมุติให้ $(r_i, l_k) \in RL \wedge (r_j, l_l) \in RL \wedge (r_i, r_j) \in CR \wedge (l_k, l_l) \notin CL$

เมื่อ $(r_i, r_j) \in CR$ จะได้ตามนิยามที่ 3.4.3 คือ

$$(location^*(r_i) \times location^*(r_j)) \cap CL \neq \varnothing$$

และจากนิยาม 3.4.6

$$(role^*(l_k) \times role^*(l_l)) \cap CR \neq \varnothing$$

ดังนั้นจึงทำให้ $(l_k, l_l) \in CL$ ซ.ต.พ.

3.6 หลักการอื่นๆ ที่เกี่ยวข้อง

เพื่อให้รูปแบบ ERBAC03 มีความสมบูรณ์มากขึ้นจึงได้มีการกำหนดถึงหลักการการให้สิทธิ์และการยกเลิกสิทธิ์ไว้ใช้เป็นแนวทางในการทำงานร่วมกับรูปแบบ ERBAC03 เป็นอย่างมีประสิทธิภาพมากขึ้น

3.6.1 การให้สิทธิ์ (Delegation) [12]

โดยการให้สิทธิ์นี้จะมุ่งประเด็นไปที่การโอนสิทธิ์ระหว่างมนุษย์กับมนุษย์เท่านั้น ซึ่งมนุษย์ที่ได้รับบทบาทสามารถโอนสิทธิ์ความเป็นสมาชิกของเขาให้กับผู้อื่นที่มีระดับความปลอดภัยเท่ากันหรือต่ำกว่าได้ภายในที่ตั้งเดียวกัน และสำหรับการให้สิทธิ์ระหว่างที่ตั้ง สามารถทำได้โดยโอนสิทธิ์ความเป็นสมาชิกของบทบาทให้กับที่ตั้งอื่นที่มีระดับความปลอดภัยของที่ตั้งเท่ากันหรือต่ำกว่า

นิยาม 3.6.1.1: เป็นการควบคุมการโอนสิทธิ์ความเป็นสมาชิกระหว่างผู้ใช้กับผู้ใช้ โดยใช้ตาราง

$$\text{can-delegate-role} \subseteq R \times R$$

นิยาม 3.6.1.2: เป็นการควบคุมการโอนสิทธิ์ความเป็นสมาชิกระหว่างที่ตั้งกับที่ตั้ง โดยใช้ตาราง

$$\text{can-delegate-location} \subseteq L \times 2^R$$

นิยามที่ 3.6.1.1 และ 3.6.1.2 ต้องมีการพิจารณาระดับความปลอดภัยอย่างระมัดระวัง

3.6.2 การยกเลิกสิทธิ์ (Revocation) [27]

นิยาม 3.6.2.1: เป็นการควบคุมการยกเลิกสิทธิ์ความเป็นสมาชิกระหว่างผู้ใช้กับผู้ใช้ โดยใช้ตาราง

$$\text{can-revoke-role} \subseteq R \times R$$

นิยาม 3.6.2.2: เป็นการควบคุมการการยกเลิกสิทธิ์ความเป็นสมาชิกระหว่างที่ตั้งกับที่ตั้ง โดยใช้ตาราง

$$\text{can-revoke-location} \subseteq L \times 2^R$$

นิยามที่ 3.6.2.1 และ 3.6.2.2 ต้องมีการพิจารณาระดับความปลอดภัยอย่างระมัดระวัง

โดยการให้สิทธิ์และยกเลิกสิทธิ์นี้เป็นวิธีการหนึ่งที่เหมาะสมกับรูปแบบ ERBAC03 เพื่อนำไปปรับใช้ได้อย่างมีประสิทธิภาพ ส่วนรายละเอียดต่างๆจะอยู่นอกเหนือจากงานวิจัยฉบับนี้

ในบทนี้ได้นำเสนอวิธีการกำหนดรูปแบบการควบคุมการเข้าถึงแบบใหม่เรียกว่า ERBAC03 ซึ่งได้มีการกำหนดนิยามและความสัมพันธ์ทั้งหมดให้ครอบคลุมถึงองค์ประกอบทุกๆ ส่วน อีกทั้งได้มีการกำหนดคุณสมบัติข้อบังคับให้กับรูปแบบการควบคุมการเข้าถึง ERBAC03 เพื่อป้องกันการกำหนดบทบาทให้กับผู้ใช้ที่ไม่ถูกต้อง ซึ่งอาจทำให้เกิดการทุจริตขึ้นได้ ในบทต่อไปได้นำนิยามและข้อบังคับทั้งหมดที่กำหนดไว้มาออกแบบอัลกอริทึมเพื่อใช้ในการทดลองต่อไป

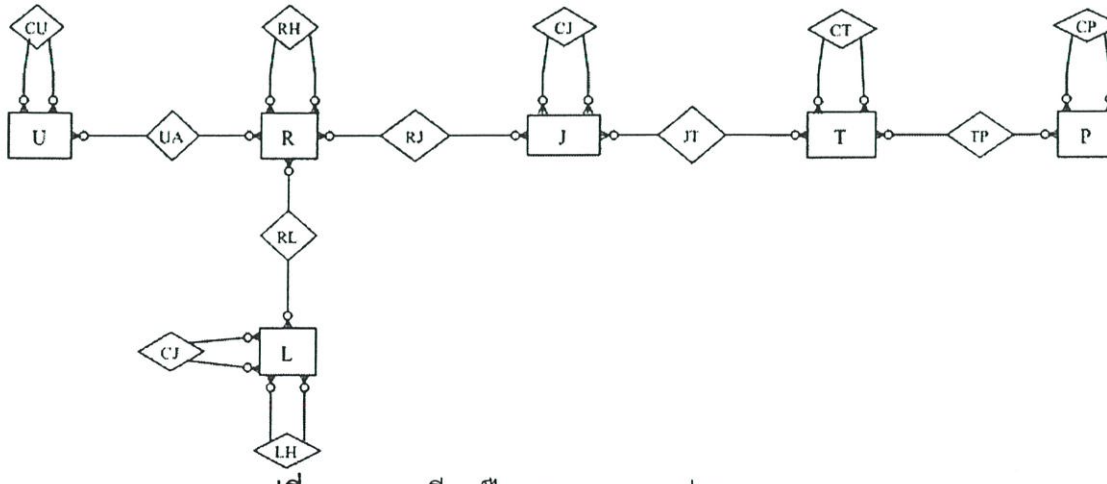
บทที่ 4
การออกแบบอัลกอริทึมสำหรับรูปแบบ ERBAC03

ในบทที่ผ่านมาได้กล่าวถึงการกำหนดนิยามขององค์ประกอบ ERBAC03 ทั้งหมด รวมถึงแบบอย่างการจัดการองค์ประกอบ ERBAC03 ที่ขัดแย้งกัน โดยแสดงออกมาในรูปของสมการทางคณิตศาสตร์ ซึ่งในบทนี้จะอธิบายถึงอัลกอริทึมทั้งหมดโดยใช้รูปแบบฐานข้อมูลแบบตาราง (Relational database system) ตัวอย่างเช่น ต้องมีการกำหนดอีอาร์ไดอะแกรมสำหรับรูปแบบ ERBAC03 ด้วย

4.1 อัลกอริทึมสำหรับรูปแบบ ERBAC03

ในบทที่ผ่านมาได้กล่าวถึงความสัมพันธ์ทางคณิตศาสตร์ระหว่างเซตของเอนทิที ซึ่งได้ระบุเป็นตารางในฐานข้อมูล ซึ่งแสดงโดยใช้อีอาร์ไดอะแกรม ดังรูปที่ 4.1

ความสัมพันธ์ทางคณิตศาสตร์นั้นเป็นวิธีการที่ง่ายในการพัฒนาเป็นอัลกอริทึม จากรูปที่ 4.1 สามารถแบ่งประเภทของตารางในฐานข้อมูลออกเป็นสามประเภทหลักๆ คือ ตารางความสัมพันธ์ ตารางความขัดแย้งและตารางเอนทิที โดยความสัมพันธ์จากรูปสามารถอธิบายได้โดยใช้ตาราง UA RL RJ JT TP RH และ LH ซึ่งมีความสัมพันธ์แบบหลายต่อหลายในแต่ละความสัมพันธ์ ความสัมพันธ์ RH และ LH นั้นค่อนข้างแตกต่างจากความสัมพันธ์อื่น ซึ่งจะใช้ในการสร้างลำดับชั้นบทบาทและที่ตั้ง



รูปที่ 4.1 แสดงอีอาร์ไดอะแกรมของรูปแบบ ERBAC03

ความขัดแย้งระหว่างเอนทิทีแสดงโดยตาราง CU CL CR CJ CT และ CP ซึ่งความขัดแย้งเป็นความสัมพันธ์แบบหลายต่อหลายระหว่างเอนทิทีของเฉพาะเซตนั้น โดยมีหลายตารางที่แสดงถึงเซตของเอนทิทีที่แสดงถึง U R L J T และ P

ภายใต้ความสัมพันธ์แต่ละประเภทนั้นมีความเป็นไปได้ที่จะมีความขัดแย้งที่แตกต่างกัน ซึ่งแต่ละความเป็นไปได้นั้นต้องการอัลกอริทึมที่เป็นต้นแบบของเครื่องมือการจัดการในการทำการทดสอบการละเมิดความคงสภาพของข้อมูล ซึ่งการกระทำใดๆ นั้นจะเป็นการละเมิดความคงสภาพข้อมูลที่ไม่อนุญาตให้ทำต่อไป โดยจะรวมการกระทำต่างๆ เช่น การเพิ่มหรือการลบเอนทิทีจากระบบฯ ที่ความคงสภาพของระบบเป็นจุดประสงค์สำคัญมาก ซึ่งเป็นไปไม่ได้เลยที่จะทำการลบเอนทิทีโดยปราศจากการตรวจสอบผลกระทบของการลบนั้น

ตารางที่ 4.1 แสดงถึงโอเปอร์เรชั่นที่แสดงถึงความสัมพันธ์ ความขัดแย้งและเอนทิที ซึ่งโอเปอร์เรชั่นเหล่านี้คือการเพิ่ม และการลบ ซึ่งรูปแบบ ERBAC03 ได้อนุญาตให้ทำการปรับปรุงหรือดัดแปลงโดยผ่านกระบวนการแรกคือการลบและการเพิ่มตามลำดับ

ตารางที่ 4.1 แสดงถึงโอเปอร์เรชั่นที่แสดงถึงความสัมพันธ์ ความขัดแย้งและเอนทิที

		โอเปอร์เรชั่น	
อัลกอริทึม	ตาราง	การเพิ่ม	การลบ
ความสัมพันธ์	UA, RL, RJ, JT, TP	หัวข้อที่ 4.2.1	หัวข้อที่ 4.2.2
	LH, RH	หัวข้อที่ 4.2.3	
ความขัดแย้ง	CU, CL, CR, CT, CT, CP	หัวข้อที่ 4.3.1	หัวข้อที่ 4.3.2
เอนทิที	Users, Locations, Roles, Jobs, Tasks, Permissions	หัวข้อที่ 4.4.1	หัวข้อที่ 4.4.2

4.2 อัลกอริทึมสำหรับความสัมพันธ์ของเอนทิที

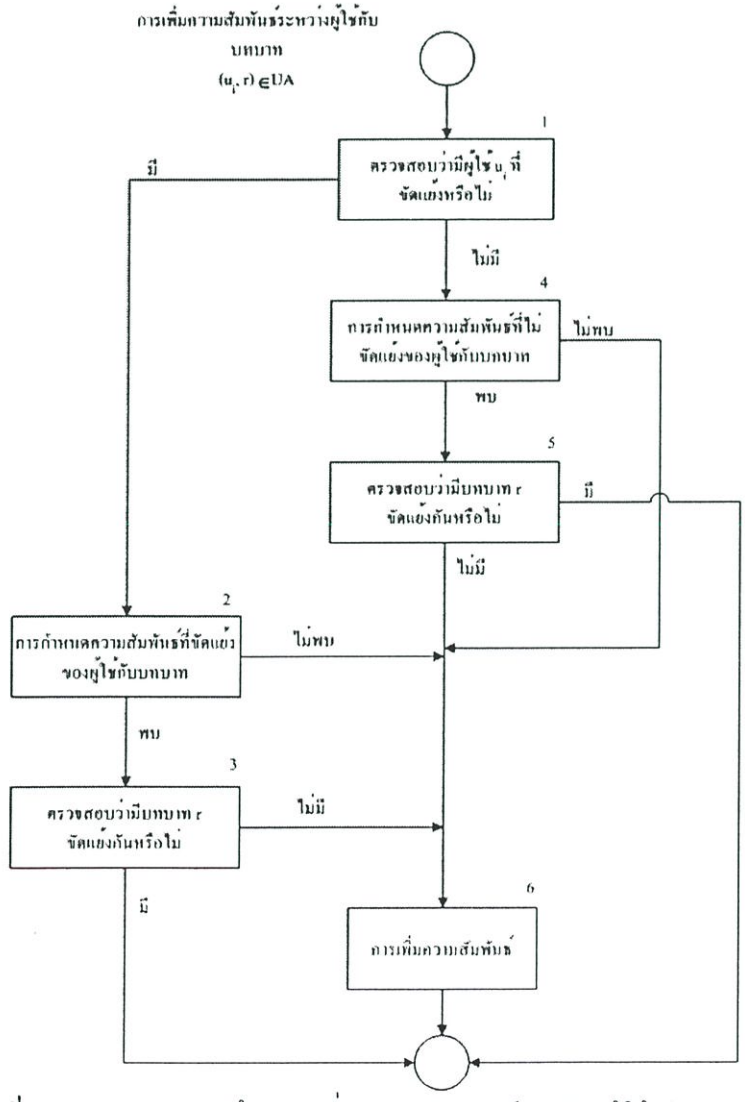
สมมุติให้เซตของเอนทิทีและความสัมพันธ์ที่ขัดแย้งกันที่มีอยู่ภายใต้สภาพแวดล้อมของการจัดการการเข้าถึงระบบ โดยอัลกอริทึมที่เกี่ยวข้องกับการสร้างและการลบของเอนทิทีและความขัดแย้งจะถูกแสดงไว้ตามหัวข้อต่างๆ ดังต่อไปนี้

4.2.1 การเพิ่มความสัมพันธ์

อัลกอริทึมแรกจะอธิบายการสร้างความสัมพันธ์ของผู้ใช้ให้กับบทบาท ตัวอย่างเช่นการมอบหมายให้ผู้ใช้ ก. มีบทบาทเป็นหัวหน้าไปรษณีย์ ซึ่งความสัมพันธ์นี้สามารถเขียนได้ดังนี้ $(u_i, r) \in UA$ ซึ่งก่อนหน้าที่จะกำหนดความสัมพันธ์จะต้องดำเนินการดังรูปที่ 4.2 ดังขั้นตอนของอัลกอริทึมต่อไปนี้

ขั้นตอนที่ 1 ตรวจสอบว่ามีผู้ใช้ u_i ที่ขัดแย้งกันผู้ใช้อื่นๆ หรือไม่ ซึ่งทำซ้ำผ่านทุกแถวของผู้ใช้ที่ขัดแย้งเมื่อผู้ใช้คนหนึ่งเป็นผู้ใช้ที่มีความสัมพันธ์ที่ขัดแย้งกัน ถ้าคำตอบคือใช่จะกระทำต่อไปในขั้นตอนที่ 2 ถ้าไม่ใช่จะทำต่อไปในขั้นตอนที่ 4

ขั้นตอนที่ 2 ทำการหาบทบาทที่มีความสัมพันธ์กับผู้ใช้ที่ขัดแย้งที่พบในขั้นตอนที่ 1 ซึ่งทำซ้ำผ่านทุกความสัมพันธ์ของผู้ใช้กับบทบาทเมื่อทุกความสัมพันธ์ของผู้ใช้ที่อยู่ในกลุ่มผู้ใช้ที่พบในขั้นตอนที่ 1 ถ้าไม่พบบทบาท จึงสามารถเพิ่มความสัมพันธ์ได้ แต่ถ้าพบว่ามีบทบาทให้ทำต่อในขั้นตอนที่ 3



รูปที่ 4.2 แสดงอัลกอริทึมการเพิ่มความสัมพันธ์ระหว่างผู้ใช้กับบทบาท

ขั้นตอนที่ 3 บทบาทใดๆ ที่พบในขั้นตอนที่ 2 ขัดแย้งกับบทบาทที่จะเพิ่มหรือไม่ ซึ่งจะบรรลุผลสำเร็จโดยการทำซ้ำผ่านแถวของบทบาทที่ขัดแย้งเมื่อบทบาทปัจจุบันและบทบาทใดๆ ที่พบในขั้นตอนที่ 2 ถ้าไม่ขัดแย้ง จะสามารถเพิ่มความสัมพันธ์ได้ ถ้าพบความสัมพันธ์ที่ขัดแย้งของบทบาทกับบทบาทใดๆ จะไม่ให้ทำการเพิ่มความสัมพันธ์

ขั้นตอนที่ 4 ทำการหาบทบาทที่มีความสัมพันธ์กับผู้ใช้ที่ไม่ขัดแย้งว่ามีหรือไม่ ซึ่งทำซ้ำผ่านทุกแถวของความสัมพันธ์ของผู้ใช้กับบทบาท โดยถ้าพบให้ทำต่อในขั้นตอนที่ 5 ถ้าไม่พบบทบาท จึงสามารถเพิ่มความสัมพันธ์ได้

ขั้นตอนที่ 5 บทบาทใดๆ ที่พบในขั้นตอนที่ 4 ขัดแย้งกับบทบาทที่จะเพิ่มหรือไม่ ซึ่งจะบรรลุผลสำเร็จโดยการทำซ้ำผ่านแถวของบทบาทที่ขัดแย้งเมื่อบทบาทปัจจุบันและบทบาทใดๆ ที่พบในขั้นตอนที่ 4 ถ้าไม่ขัดแย้ง จะสามารถเพิ่มความสัมพันธ์ได้ ถ้าพบความสัมพันธ์ที่ขัดแย้งของบทบาทกับบทบาทใดๆ จะไม่ให้ทำการเพิ่มความสัมพันธ์

ขั้นตอนที่ 6 การเพิ่มความสัมพันธ์ ซึ่งคล้ายการเพิ่มข้อมูลลงในตาราง

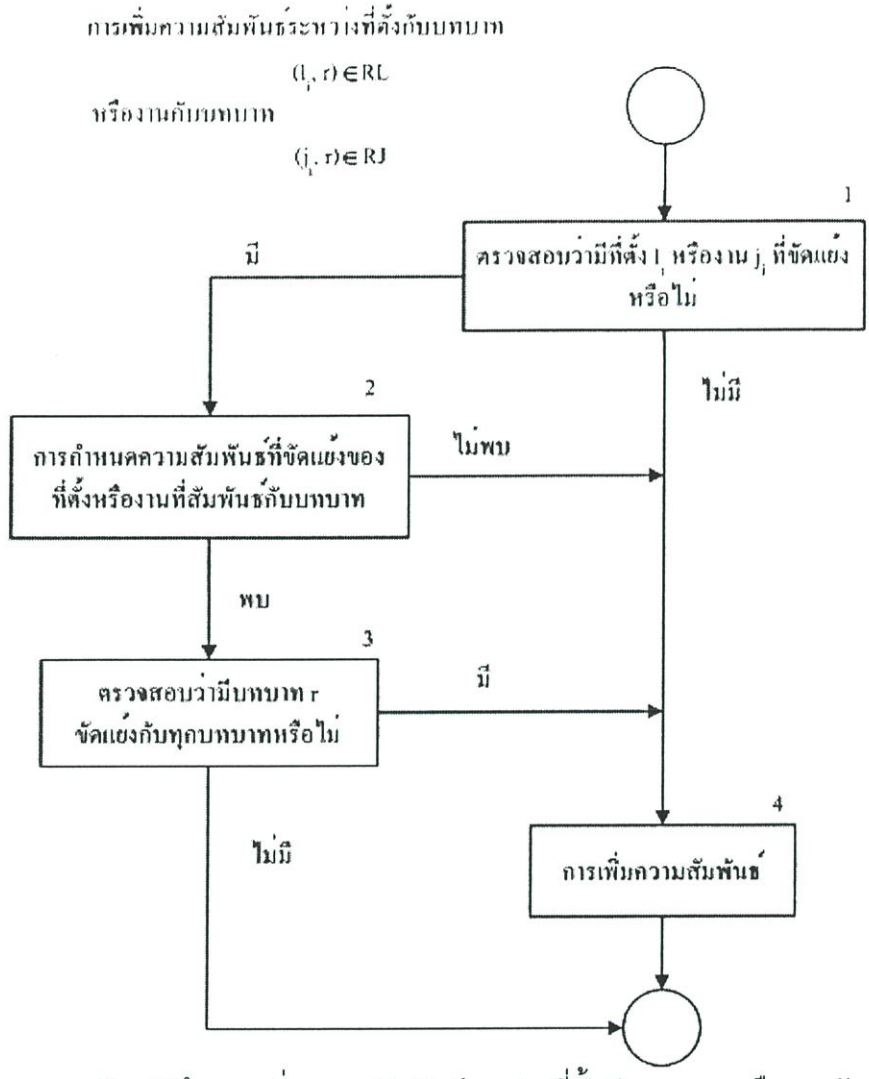
อัลกอริทึมต่อไปใช้เมื่อมีการเพิ่มความสัมพันธ์ระหว่างที่ตั้งกับบทบาท ซึ่งสามารถเขียนได้ดังนี้ $(l_i, r) \in RL$ ซึ่งความสัมพันธ์นี้สามารถเขียนแทนได้ด้วยรูปที่ 4.3 โดยมีขั้นตอนต่อไปนี้

ขั้นตอนที่ 1 ที่ตั้ง l_i มีความสัมพันธ์ที่ขัดแย้งกับที่ตั้งอื่นหรือไม่ ซึ่งจะทำซ้ำผ่านทุกความขัดแย้งของที่ตั้งเมื่อหนึ่งในที่ตั้งนั้นเป็นที่ตั้งที่กำลังทำการเพิ่มในความสัมพันธ์ ถ้าคำตอบคือมีให้ทำขั้นตอนที่ 2 ต่อไป ถ้าไม่มีให้ทำการเพิ่มความสัมพันธ์

ขั้นตอนที่ 2 หาทุกบทบาทที่มีความสัมพันธ์กับที่ตั้งที่ขัดแย้งที่พบในขั้นตอนที่ 1 ซึ่งจะทำซ้ำผ่านตารางความสัมพันธ์ระหว่างที่ตั้งกับบทบาทเมื่อความสัมพันธ์ของที่ตั้งหรืองานใดๆ ที่อยู่ในกลุ่มของที่ตั้งที่พบในขั้นตอนที่ 1 ถ้าไม่พบบทบาทนั้นสามารถเพิ่มความสัมพันธ์ได้ ถ้าพบความสัมพันธ์ระหว่างบทบาทกับที่ตั้งที่ขัดแย้งให้ทำในขั้นตอนที่ 3

ขั้นตอนที่ 3 ทุกบทบาทที่พบในขั้นตอนที่ 2 ขัดแย้งกับบทบาทที่จะเพิ่มเข้าไปหรือไม่? ซึ่งจะทำซ้ำผ่านแถวของบทบาทที่ขัดแย้งเมื่อบทบาทปัจจุบันและบทบาทใดๆ ที่พบในขั้นตอนที่ 2 ถ้าพบสามารถเพิ่มความสัมพันธ์ได้ ถ้าไม่พบบทบาทที่ขัดแย้งกับทุกบทบาทที่มีแล้วไม่ให้ทำการเพิ่มความสัมพันธ์ โดยขั้นตอนนี้ต่างจากอัลกอริทึมความสัมพันธ์ระหว่างผู้ใช้กับบทบาทก็ต่อเมื่อทุกบาทที่พบนั้นเป็นบทบาทที่ขัดแย้งเท่านั้นจึงเพิ่มความสัมพันธ์ได้

ขั้นตอนที่ 4 การเพิ่มความสัมพันธ์



รูปที่ 4.3 แสดงอัลกอริทึมการเพิ่มความสัมพันธ์ระหว่างที่ตั้งกับบทบาท หรืองานกับบทบาท

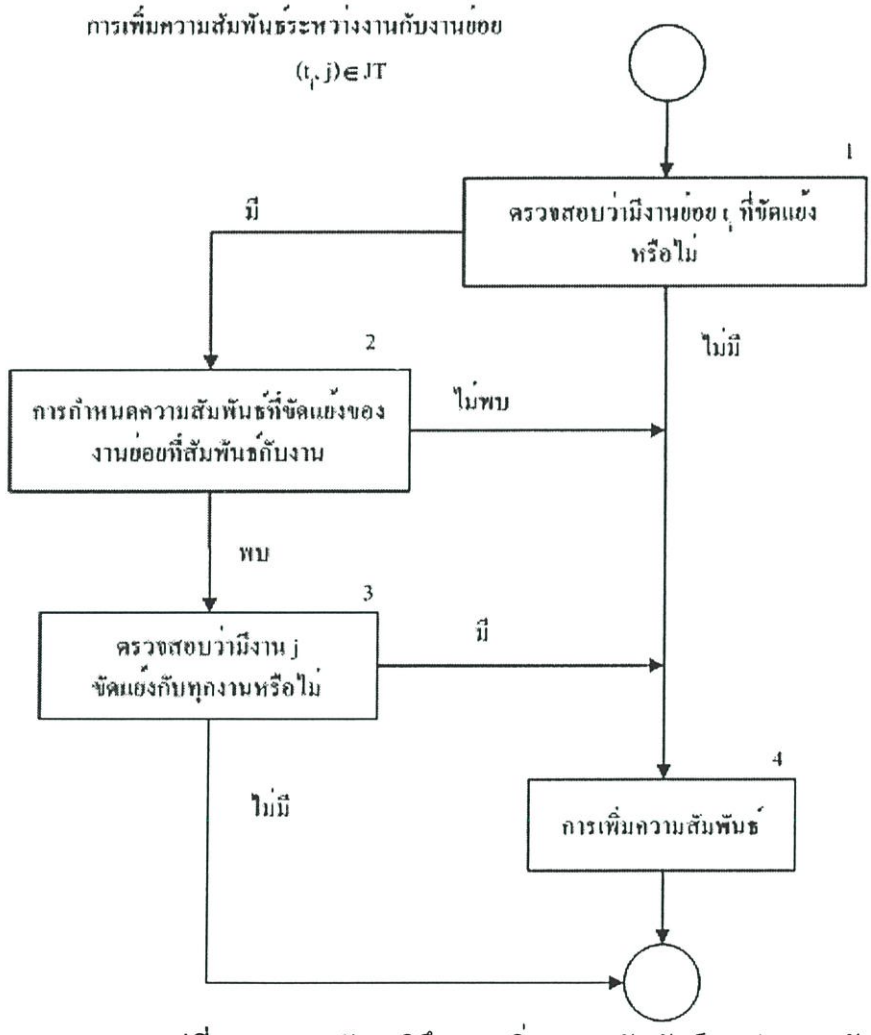
เนื่องจากอัลกอริทึมในการเพิ่มความสัมพันธ์ระหว่างงานกับบทบาทจะเหมือนกับอัลกอริทึมในการเพิ่มความสัมพันธ์ระหว่างที่ตั้งกับบทบาท ดังรูปที่ 4.3 ซึ่งสามารถเขียนความสัมพันธ์ได้ดังนี้ $(j_i, r) \in RJ$ ซึ่งความสัมพันธ์นี้สามารถเขียนแทนได้ด้วยรูปที่ 4.3 โดยมีขั้นตอนต่อไปนี้

ขั้นตอนที่ 1 ตรวจสอบว่ามีงานที่ขัดแย้งหรือไม่ ซึ่งจะทำซ้ำผ่านทุกความขัดแย้งงานเมื่อหนึ่งในงานเหล่านั้นเป็นงานที่กำลังทำการเพิ่มในความสัมพันธ์ ถ้าคำตอบคือมีให้ทำขั้นตอนที่ 2 ต่อไป ถ้าไม่พบให้ทำการเพิ่มความสัมพันธ์

ขั้นตอนที่ 2 หาทุกบทบาทที่มีความสัมพันธ์ที่ตั้งที่ขัดแย้งที่พบในขั้นตอนที่ 1 ซึ่งจะทำซ้ำผ่านตารางความสัมพันธ์ระหว่างที่ตั้งกับบทบาทเมื่อความสัมพันธ์ที่ตั้งใดๆ ที่อยู่ในกลุ่มของที่ตั้งที่พบในขั้นตอนที่ 1 ถ้าไม่พบบทบาทนั้นสามารถเพิ่มความสัมพันธ์ได้ ถ้าพบความสัมพันธ์ระหว่างบทบาทกับงานที่ขัดแย้งให้ทำในขั้นตอนที่ 3

ขั้นตอนที่ 3 ทุกบทบาทที่พบในขั้นตอนที่ 2 ขัดแย้งกับบทบาทที่จะเพิ่มเข้าไปหรือไม่ ซึ่งจะทำซ้ำผ่านแถวของบทบาทที่ขัดแย้งเมื่อบทบาทปัจจุบันและบทบาทใดๆ ที่พบในขั้นตอนที่ 2 ถ้าพบสามารถเพิ่มความสัมพันธ์ได้ ถ้าไม่พบบทบาทที่ขัดแย้งกับทุกบทบาทที่มีแล้วไม่ให้ทำการเพิ่มความสัมพันธ์ โดยขั้นตอนนี้ต่างจากอัลกอริทึมความสัมพันธ์ระหว่างผู้ใช้กับบทบาทก็ต่อเมื่อทุกบทบาทที่พบนั้นเป็นบทบาทที่ขัดแย้ง

ขั้นตอนที่ 4 การเพิ่มความสัมพันธ์



รูปที่ 4.4 แสดงอัลกอริทึมการเพิ่มความสัมพันธ์ระหว่างงานกับงานย่อย

อัลกอริทึมในการเพิ่มความสัมพันธ์ระหว่างงานกับงานย่อยจะเหมือนกับอัลกอริทึมในการเพิ่มความสัมพันธ์ระหว่างที่ตั้งกับบทบาท ดังรูปที่ 4.4 ซึ่งสามารถเขียนความสัมพันธ์ได้ดังนี้

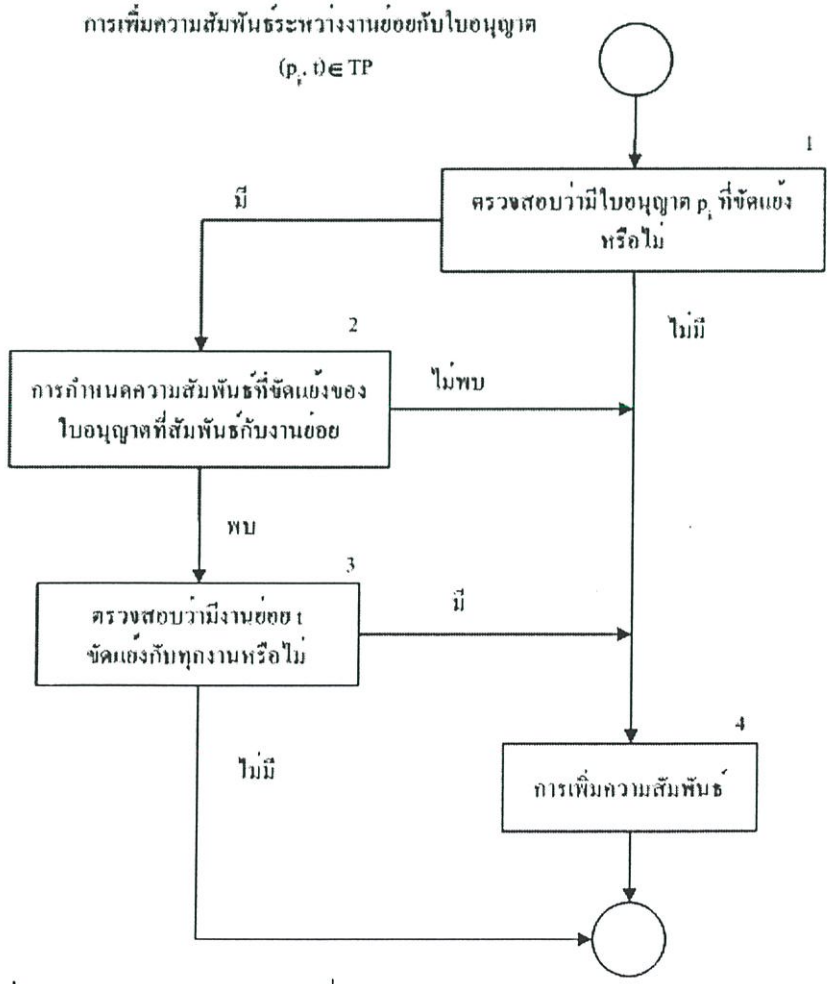
$(t_i, j) \in JT$ ซึ่งความสัมพันธ์นี้สามารถเขียนแทนได้ด้วยรูปที่ 4.4 โดยมีขั้นตอนต่อไปนี้

ขั้นตอนที่ 1 ตรวจสอบว่างานที่มีงานที่ขัดแย้งหรือไม่? ซึ่งจะทำซ้ำผ่านทุกความขัดแย้งงานเมื่อหนึ่งในงานเหล่านั้นเป็นงานที่กำลังทำการเพิ่มในความสัมพันธ์ ถ้าคำตอบคือมีให้ทำขั้นตอนที่ 2 ต่อไป ถ้าไม่พบให้ทำการเพิ่มความสัมพันธ์

ขั้นตอนที่ 2 หาทุกงานที่มีความสัมพันธ์งานย่อยที่ขัดแย้งที่พบในขั้นตอนที่ 1 ซึ่งจะทำซ้ำผ่านตารางความสัมพันธ์ระหว่างงานกับงานย่อยเมื่อความสัมพันธ์งานใดๆ ที่อยู่ในกลุ่มของงานย่อยที่พบในขั้นตอนที่ 1 ถ้าไม่พบงานนั้นสามารถเพิ่มความสัมพันธ์ได้ ถ้าพบความสัมพันธ์ระหว่างงานกับงานย่อยที่ขัดแย้งให้ทำในขั้นตอนที่ 3

ขั้นตอนที่ 3 ทุกงานที่พบในขั้นตอนที่ 2 ขัดแย้งกับงานที่จะเพิ่มเข้าไปหรือไม่? ซึ่งจะทำซ้ำผ่านแถวของงานที่ขัดแย้งเมื่องานปัจจุบันและงานใดๆ ที่พบในขั้นตอนที่ 2 ถ้าพบสามารถเพิ่มความสัมพันธ์ได้ ถ้าไม่พบงานที่ขัดแย้งกับทุกงานที่มีแล้วไม่ให้ทำการเพิ่มความสัมพันธ์ โดยขั้นตอนนี้ต่างจากอัลกอริทึมความสัมพันธ์ระหว่างผู้ใช้กับบทบาทก็ต่อเมื่อทุกบทบาทที่พบนั้นเป็นบทบาทที่ขัดแย้ง

ขั้นตอนที่ 4 การเพิ่มความสัมพันธ์



รูปที่ 4.5 แสดงอัลกอริทึมการเพิ่มความสัมพันธ์ระหว่างงานย่อยกับใบอนุญาต

อัลกอริทึมในการเพิ่มความสัมพันธ์ระหว่างงานย่อยกับใบอนุญาตจะเหมือนกับอัลกอริทึมในการเพิ่มความสัมพันธ์ระหว่างงานกับงานย่อย ดังรูปที่ 4.5 ซึ่งสามารถเขียนความสัมพันธ์ได้ดังนี้ $(p_i, t) \in TP$ ซึ่งความสัมพันธ์นี้สามารถเขียนแทนได้ด้วยรูปที่ 4.5 โดยมีขั้นตอนต่อไปนี้

ขั้นตอนที่ 1 ตรวจสอบว่ามีใบอนุญาตที่ขัดแย้งหรือไม่ ซึ่งจะทำซ้ำผ่านทุกความขัดแย้งของใบอนุญาตเมื่อหนึ่งในใบอนุญาตเหล่านั้นเป็นใบอนุญาตที่กำลังทำการเพิ่มในความสัมพันธ์ ถ้าคำตอบคือมีให้ทำขั้นตอนที่ 2 ต่อไป ถ้าไม่พบให้ทำการเพิ่มความสัมพันธ์

ขั้นตอนที่ 2 หาทุกงานย่อยที่มีความสัมพันธ์กับใบอนุญาตที่ขัดแย้งที่พบในขั้นตอนที่ 1 ซึ่งจะทำซ้ำผ่านตารางความสัมพันธ์ระหว่างงานย่อยกับใบอนุญาตเมื่อความสัมพันธ์งานย่อยใดๆ ที่อยู่ในกลุ่มของใบอนุญาตที่พบในขั้นตอนที่ 1 ถ้าไม่พบงานย่อยนั้นสามารถเพิ่มความสัมพันธ์ได้ ถ้าพบความสัมพันธ์ระหว่างงานย่อยกับใบอนุญาตที่ขัดแย้งให้ทำในขั้นตอนที่ 3

ขั้นตอนที่ 3 ทุกงานย่อยที่พบในขั้นตอนที่ 2 ขัดแย้งกับใบอนุญาตที่จะเพิ่มเข้าไปหรือไม่? ซึ่งจะทำซ้ำผ่านแถวของงานย่อยที่ขัดแย้งเมื่องานย่อยปัจจุบันและงานย่อยใดๆ ที่พบในขั้นตอนที่ 2 ถ้าพบสามารถเพิ่มความสัมพันธ์ได้ ถ้าไม่พบงานย่อยที่ขัดแย้งกับทุกงานย่อยที่มีแล้วไม่ให้ทำการเพิ่มความสัมพันธ์ โดยขั้นตอนนี้ต่างจากอัลกอริทึมความสัมพันธ์ระหว่างผู้ใช้กับบทบาทก็ต่อเมื่อทุกบทบาทที่พบนั้นเป็นบทบาทที่ขัดแย้ง

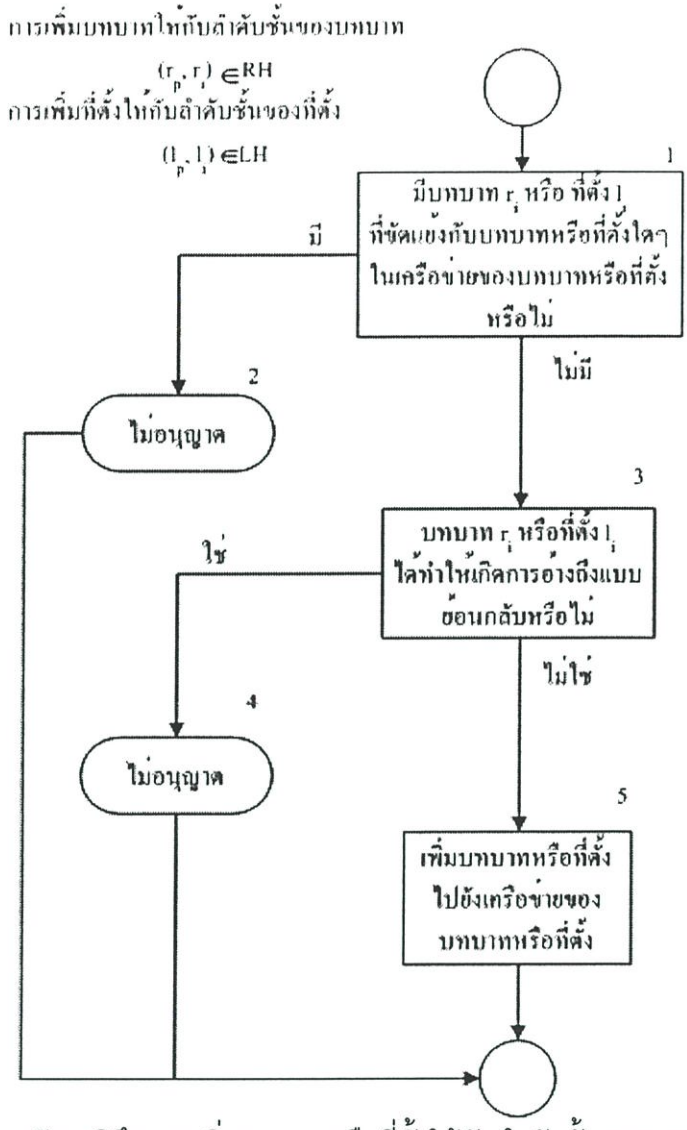
ขั้นตอนที่ 4 การเพิ่มความสัมพันธ์

4.2.2 การลบความสัมพันธ์

การลบความสัมพันธ์เหล่านี้จะไม่มีการควบคุม ซึ่งคล้ายกับว่าการลบนั้นไม่มีผลต่อความปลอดภัยแก่ระบบ แต่ก็มีความจำเป็นที่จะต้องเก็บความสัมพันธ์ที่เหลือให้ไม่กระทบต่อกระบวนการทำงานให้เป็นไปอย่างถูกต้อง

4.2.3 การบำรุงรักษาลำดับชั้นของบทบาทและที่ตั้ง

ความสัมพันธ์ประเภทสุดท้ายที่สามารถทำได้ในลำดับชั้นของบทบาทและที่ตั้ง คือแต่ละบทบาทมีบทบาทพ่อและบทบาทลูก ซึ่งไม่อนุญาตให้ความสัมพันธ์แบบย้อนกลับปรากฏภายใต้ลำดับชั้นของบทบาท ดังรูปที่ 4.6 ซึ่งแสดงอัลกอริทึมการเพิ่มบทบาทให้กับเครือข่ายของบทบาท ซึ่งสามารถเขียนได้ดังนี้ $(r_p, r_i) \in RH$ โดยบทบาท r_p เป็นบทบาทพ่อที่ถูกใช้ในการระบุว่าบทบาท r_i เป็นบทบาทที่ถูกเพิ่ม ซึ่งความสัมพันธ์นี้สามารถเขียนแทนได้ด้วยรูปที่ 4.6 โดยมีขั้นตอนต่อไปนี้



รูปที่ 4.6 แสดงอัลกอริทึมการเพิ่มบทบาทหรือที่ตั้งให้กับลำดับชั้นของบทบาทหรือที่ตั้ง

ขั้นตอนที่ 1 ตรวจสอบว่าบทบาท r_i ขัดแย้งกับบทบาทใดในลำดับชั้นของบทบาทหรือไม่ ซึ่งจะทำซ้ำผ่านทุกบทบาทที่ขัดแย้งเมื่อบทบาท r_i และบทบาทใดๆ ในลำดับชั้นของบทบาทหนึ่งในงานเหล่านั้นเป็นบทบาทที่กำลังทำการเพิ่มในความสัมพันธ์ ถ้าคำตอบคือมีให้ทำขั้นตอนที่ 2 ต่อไป ถ้าไม่มีให้ให้ทำขั้นตอนที่ 3

ขั้นตอนที่ 2 ไม่อนุญาตให้เพิ่มบทบาทในลำดับชั้นของบทบาท

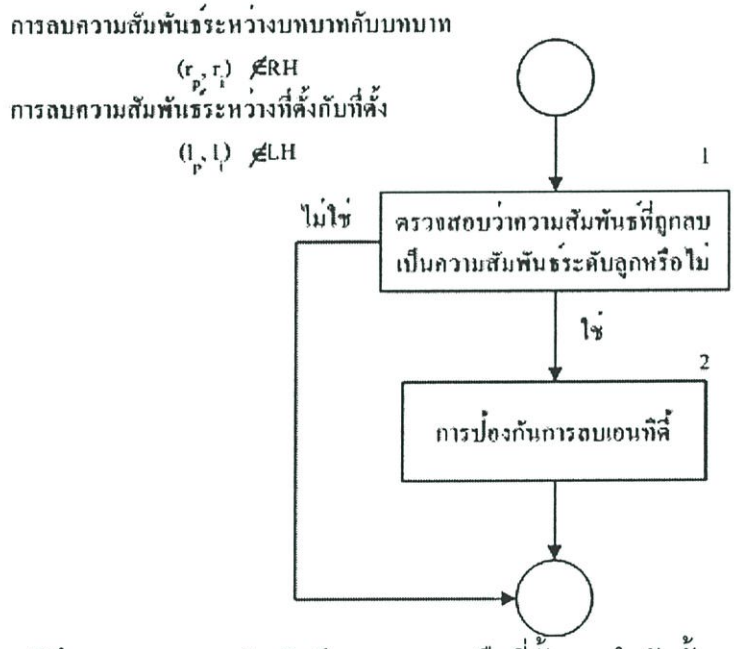
ขั้นตอนที่ 3 ตรวจสอบการเพิ่มบทบาท r_i จะสร้างแบบการอ้างถึงแบบย้อนกลับ (circular reference) ซึ่งการอ้างถึงแบบย้อนกลับนั้นเป็นบทบาทที่เคยถูกอ้างถึงไปแล้วได้ถูกอ้างถึงอีกตลอดกิ่งของลำดับชั้นของบทบาท การตรวจสอบจะสำเร็จได้โดยการตรวจสอบซ้ำทั้งกิ่งของลำดับชั้นของบทบาทที่ปรากฏในบทบาทปัจจุบัน ถ้าการอ้างถึงแบบย้อนกลับถูกตรวจพบแล้วการเพิ่ม

ความสัมพันธ์นี้จะไม่อนุญาตให้ทำให้ผ่านไปทำในขั้นตอนที่ 4 ถ้าไม่ถูกตรวจพบให้ทำการสร้างความสัมพันธ์ในขั้นตอนที่ 5

ขั้นตอนที่ 4 ไม่อนุญาตให้เพิ่มบทบาทในลำดับชั้นของบทบาท

ขั้นตอนที่ 5 สร้างความสัมพันธ์ลำดับชั้นของบทบาทโดยการเขียนข้อมูลที่ถูกต้องลงในตารางลำดับชั้นของบทบาท

เมื่อบทบาทถูกลบจากลำดับชั้นของบทบาท ต้องทำให้แน่ใจได้ว่าบทบาทพ่อและบทบาทลูกของบทบาทที่ถูกลบไม่มีความสัมพันธ์กับส่วนอื่นๆ โดยอัลกอริทึมนี้ได้อธิบายเหตุผลด้วยรูปที่ 4.7



รูปที่ 4.7 แสดงอัลกอริทึมการลบความสัมพันธ์ของบทบาทหรือที่ตั้งจากลำดับชั้นของบทบาทหรือที่ตั้ง

ขั้นตอนที่ 1 ตรวจสอบว่าความสัมพันธ์ที่จะถูกลบมีความสัมพันธ์ระดับลูกที่มีอยู่แล้วหรือไม่ ทำโดยการเลือกทุกแถวเมื่อบทบาทระดับพ่อหรือที่ตั้งหลักเท่ากับบทบาทลูกหรือที่ตั้งย่อยที่ถูกลบไปภายใต้ลำดับชั้นของบทบาทหรือที่ตั้งเดียวกัน ถ้าความสัมพันธ์กับบทบาทลูกหรือที่ตั้งย่อยมีอยู่แล้วให้ทำขั้นตอนที่ 2

ขั้นตอนที่ 2 แสดงข้อผิดพลาดและป้องกันการลบเอนทิที

อัลกอริทึมการเพิ่มและลบที่ตั้งคล้ายกับอัลกอริทึมที่แสดงดังรูปที่ 4.6 และ 4.7 ตามลำดับ ในส่วนของลำดับชั้นของบทบาทเป็นส่วนที่มีความซับซ้อนมากและยังมีอีกหลายประเด็นที่ไม่ได้กล่าวถึงในรูปแบบ ERBAC03 เช่น การสืบทอดใบอนุญาต เป็นต้น

4.3 อัลกอริทึมสำหรับเอนทิทีที่ขัดแย้ง

เป็นอัลกอริทึมที่มีความสำคัญอย่างมากต่อการจัดการความถูกต้องของระบบฯ โดยงานวิจัยฉบับนี้ยังคงกำหนดให้มีความขัดแย้งเกิดขึ้นในสภาวะแวดล้อมของการจัดการ

4.3.1 การเพิ่มเอนทิทีที่ขัดแย้ง

การกำหนดความขัดแย้งให้กับทุกเอนทิทีจะมีวิธีการกำหนดเหมือนกันในทุกๆ เอนทิที โดยมีอัลกอริทึมที่ใช้ควบคุมการกำหนดความขัดแย้ง ซึ่งแสดงออกมาในรูปแบบคณิตศาสตร์ดังนี้ $(u_i, u_j) \in CU$ สำหรับความขัดแย้งของผู้ใช้ $(l_i, l_j) \in CL$ สำหรับความขัดแย้งของที่ตั้ง $(r_i, r_j) \in CR$ สำหรับความขัดแย้งของบทบาท $(j_i, j_j) \in CJ$ สำหรับความขัดแย้งของงาน $(t_i, t_j) \in CT$ สำหรับความขัดแย้งของงานย่อย $(p_i, p_j) \in CP$ สำหรับความขัดแย้งของใบอนุญาต อัลกอริทึมทั้งหมดได้แสดงดังรูปที่ 4.8

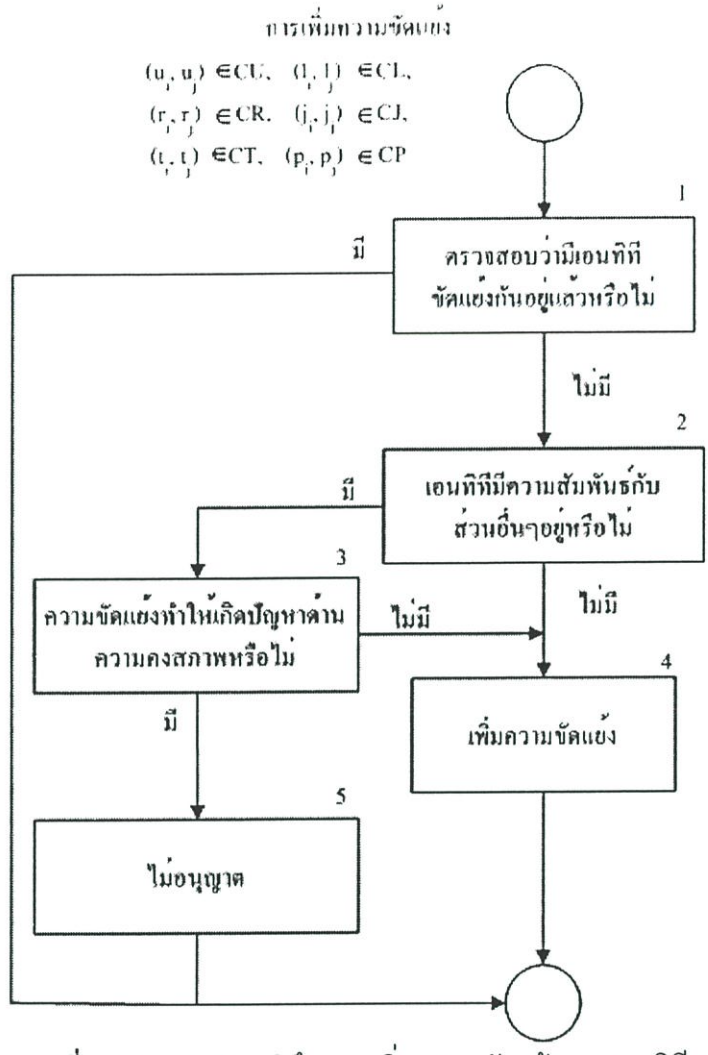
ขั้นตอนที่ 1 ตรวจสอบว่ามีเอนทิทีขัดแย้งกันอยู่แล้วหรือไม่ โดยตรวจสอบทุกแถวในตารางของฐานข้อมูลที่มีค่าข้อมูลตรงกับเอนทิทีปัจจุบัน ถ้าพบ แสดงว่าการกำหนดความขัดแย้งนี้ได้กำหนดไว้แล้วไม่ต้องทำซ้ำอีก แต่ถ้าไม่พบ ให้ทำในขั้นตอนที่ 2

ขั้นตอนที่ 2 ตรวจสอบว่าเอนทิทีมีความสัมพันธ์กับส่วนอื่นๆ อยู่หรือไม่ โดยดูจากตารางความสัมพันธ์ในฐานข้อมูล ถ้าพบให้ทำขั้นตอนที่ 3 ถ้าไม่พบให้เพิ่มความขัดแย้งของเอนทิทีได้

ขั้นตอนที่ 3 ถ้ามีการกำหนดความขัดแย้งต่อไปจะทำให้ความสัมพันธ์ที่มีอยู่เดิมไม่ถูกต้อง จึงจำเป็นต้องไม่ให้เพิ่มความขัดแย้งกันระหว่างเอนทิทีนั้นๆ ที่จะทำให้เกิดปัญหาความคงสภาพของระบบฯ ซึ่งขั้นตอนนี้ได้ขยายให้ชัดเจนในรูปที่ 4.9 4.10 4.11 4.12 4.13 และ 4.14 ถ้าการเพิ่มความขัดแย้งของเอนทิทีไม่ทำให้เกิดปัญหาด้านความคงสภาพของระบบฯ ก็ทำการเพิ่มความขัดแย้งของเอนทิทีได้ แต่ถ้าทำให้เกิดปัญหาขึ้น จะไม่อนุญาตให้เพิ่มความขัดแย้งของเอนทิทีเลย

ขั้นตอนที่ 4 ไม่อนุญาตให้เพิ่มความขัดแย้งของเอนทิที

ขั้นตอนที่ 5 เพิ่มความขัดแย้งของเอนทิทีได้ โดยการบันทึกลงฐานข้อมูล



รูปที่ 4.8 แสดงอัลกอริทึมการเพิ่มความขัดแย้งของเอนทิที

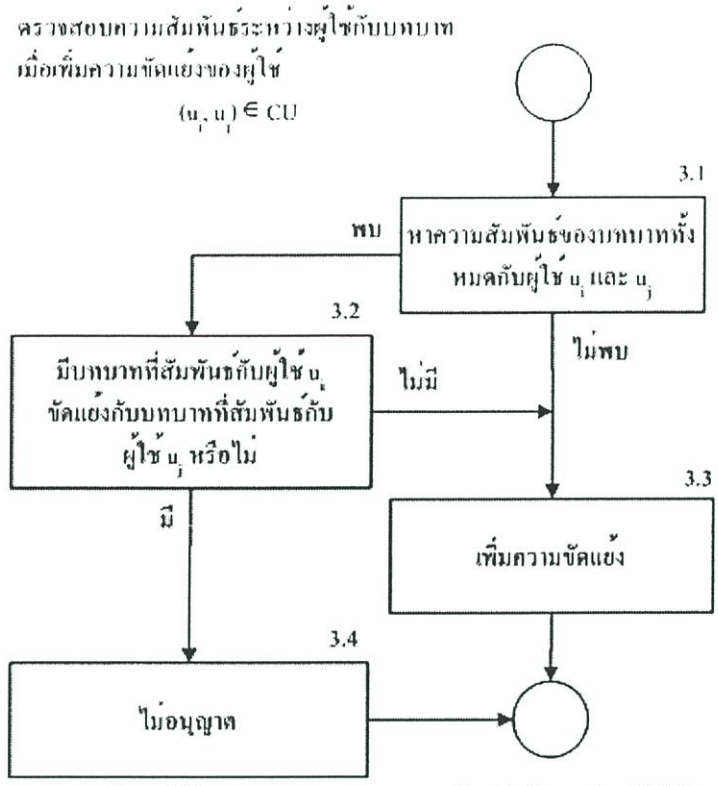
ถ้ามีการกำหนดความขัดแย้งระหว่างผู้ใช้สองคน ให้พิจารณาอัลกอริทึมดังรูปที่ 4.9 ซึ่งมีขั้นตอนการทำงานดังนี้

ขั้นตอนที่ 3.1 สมมุติให้ผู้ใช้ u_i และ u_j มีความขัดแย้งกัน และมีการหาความสัมพันธ์ที่เกี่ยวข้องกับบทบาทของผู้ใช้ทั้งสองคน ถ้าพบความสัมพันธ์กับบทบาทให้ทำในขั้นตอนที่ 3.2 ถ้าไม่พบ ให้เพิ่มความขัดแย้งของผู้ใช้ได้

ขั้นตอนที่ 3.2 ตรวจสอบบทบาททั้งหมดที่มีความสัมพันธ์กับผู้ใช้ u_i กับที่สัมพันธ์กับผู้ใช้ u_j โดยตรวจสอบในตารางความขัดแย้งของบทบาทในฐานข้อมูล ถ้าไม่พบบทบาทที่ขัดแย้งกัน ให้เพิ่มความขัดแย้งของเอนทิทีได้

ขั้นตอนที่ 3.3 ไม่ให้เพิ่มความขัดแย้งของเอนทิที ที่เป็นสาเหตุให้เกิดความสัมพันธ์ที่ไม่ถูกต้องขึ้น

ขั้นตอนที่ 3.4 เพิ่มความขัดแย้งโดยบันทึกลงในตารางในฐานข้อมูล



รูปที่ 4.9 แสดงอัลกอริทึมการตรวจสอบความสัมพันธ์ระหว่างผู้ใช้กับบทบาท

ถ้ามีการกำหนดความขัดแย้งระหว่างสองงาน (j_i, j_j) ∈ CJ) ให้ดำเนินตามอัลกอริทึมดังรูปที่ 4.10 ดังนี้

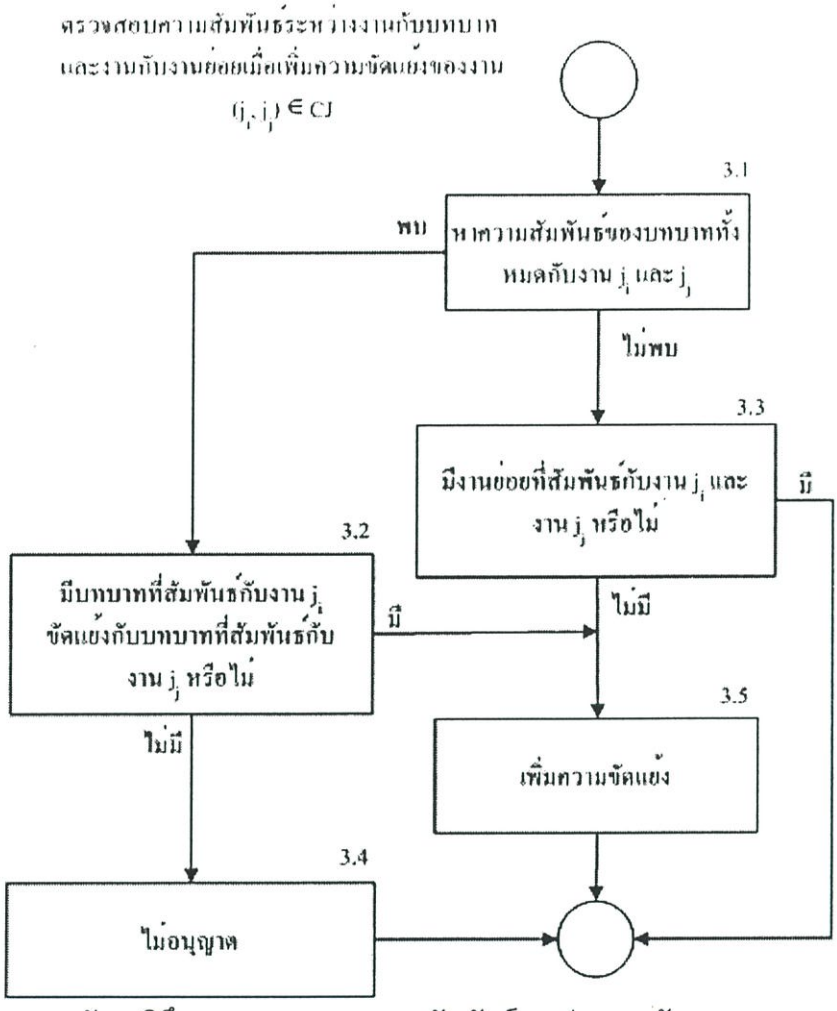
ขั้นตอนที่ 3.1 สมมุติให้เอนทิที j_i กับ j_j มีความขัดแย้งกัน โดยต้องหาความสัมพันธ์ทั้งหมดระหว่างบทบาทกับเอนทิทีทั้งสอง ถ้าพบให้ทำขั้นตอนที่3.2 แต่ถ้าไม่พบ ให้ทำขั้นตอนที่ 3.3

ขั้นตอนที่ 3.2 ตรวจสอบความสัมพันธ์ระหว่างบทบาทกับเอนทิทีแรกกับความสัมพันธ์ระหว่างบทบาทกับเอนทิทีที่สองว่าขัดแย้งกันหรือไม่ ซึ่งตรวจสอบโดยค้นจากคู่บทบาทที่อยู่ในตาราง CR ถ้าพบว่าบทบาทมีความขัดแย้งกัน จะอนุญาตให้เพิ่มความขัดแย้งระหว่างเอนทิทีทั้งสองได้

ขั้นตอนที่ 3.3 ตรวจสอบความสัมพันธ์ระหว่างงานย่อยกับงานว่ามีหรือไม่ ซึ่งตรวจสอบโดยค้นจากตารางความสัมพันธ์ JT ถ้าพบ จะไม่อนุญาตให้เพิ่มความขัดแย้งระหว่างเอนทิทีทั้งสองได้ แต่ถ้าไม่พบ จะอนุญาตให้เพิ่มความขัดแย้งระหว่างเอนทิทีทั้งสองได้

ขั้นตอนที่ 3.4 ไม่ให้เพิ่มความขัดแย้งของเอนทิที ที่เป็นสาเหตุให้เกิดความสัมพันธ์ที่ไม่ถูกต้องขึ้น

ขั้นตอนที่ 3.5 เพิ่มความขัดแย้งโดยบันทึกลงในตารางในฐานข้อมูล



รูปที่ 4.10 แสดงอัลกอริทึมการตรวจสอบความสัมพันธ์ระหว่างงานกับบทบาทและงานกับงานย่อย

ถ้ามีการกำหนดความขัดแย้งระหว่างสองงานย่อย ($(t_i, t_j) \in CT$) ให้ดำเนินตามอัลกอริทึมดังรูปที่ 4.11 ดังนี้

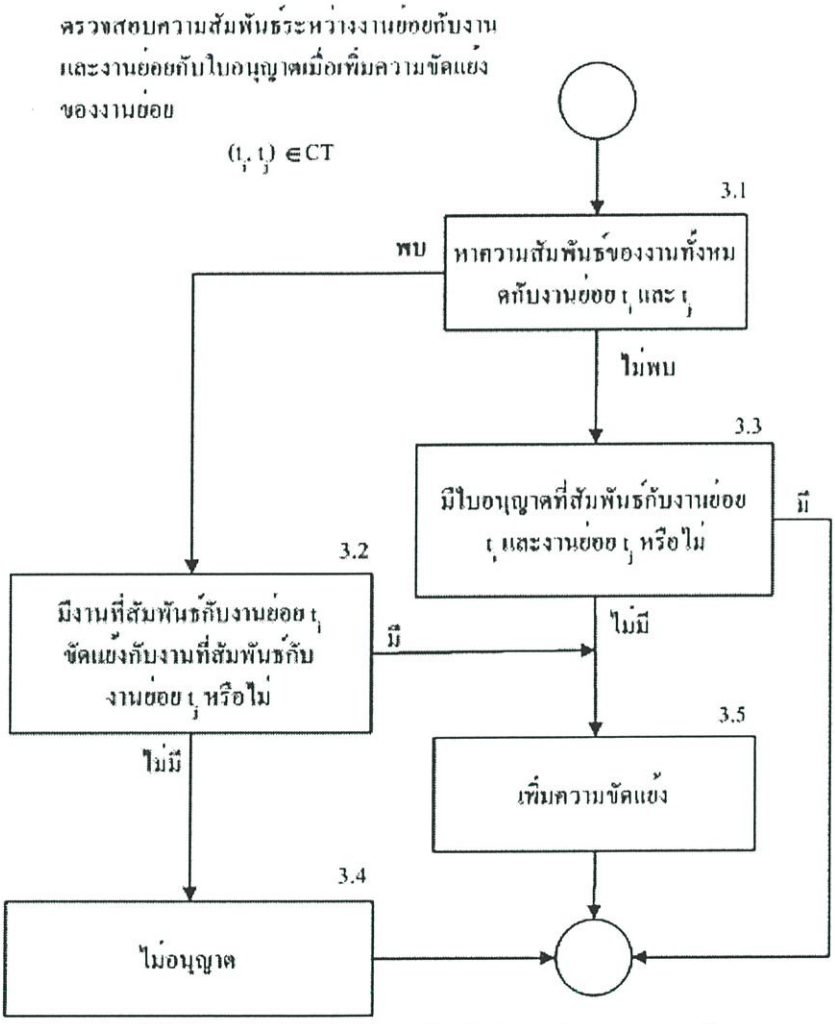
ขั้นตอนที่ 3.1 สมมุติให้เอนทิที t_i กับ t_j มีความขัดแย้งกัน โดยต้องหาความสัมพันธ์ทั้งหมดระหว่างงานกับเอนทิทีทั้งสอง ถ้าพบให้ทำขั้นตอนที่3.2 แต่ถ้าไม่พบ ให้ทำขั้นตอนที่ 3.3

ขั้นตอนที่ 3.2 ตรวจสอบความสัมพันธ์ระหว่างงานกับเอนทิทีแรกกับความสัมพันธ์ระหว่างงานกับเอนทิทีที่สองว่าขัดแย้งกันหรือไม่ ซึ่งตรวจสอบโดยค้นจากคู่งานที่อยู่ในตาราง CJ ถ้าพบว่างานมีความขัดแย้งกัน จะอนุญาตให้เพิ่มความขัดแย้งระหว่างเอนทิทีทั้งสองได้

ขั้นตอนที่ 3.3 ตรวจสอบความสัมพันธ์ระหว่างใบอนุญาตกับงานย่อยว่ามีหรือไม่ ซึ่งตรวจสอบโดยค้นจากตารางความสัมพันธ์ TP ถ้าพบ จะไม่อนุญาตให้เพิ่มความขัดแย้งระหว่างเอนทิทีทั้งสองได้ แต่ถ้าไม่พบ จะอนุญาตให้เพิ่มความขัดแย้งระหว่างเอนทิทีทั้งสองได้

ขั้นตอนที่ 3.4 ไม่ให้เพิ่มความขัดแย้งของเอนทิที ที่เป็นสาเหตุให้เกิดความสัมพันธ์ที่ไม่ถูกต้องขึ้น

ขั้นตอนที่ 3.5 เพิ่มความขัดแย้งโดยบันทึกลงในตารางในฐานข้อมูล



รูปที่ 4.11 แสดงอัลกอริทึมการตรวจสอบความสัมพันธ์ระหว่างงานย่อยกับงานและงานย่อยกับใบอนุญาต

ถ้ามีการกำหนดความขัดแย้งระหว่างสองใบอนุญาต ($(p_i, p_j) \in CP$) ให้ดำเนินตามอัลกอริทึมดังรูปที่ 4.12 ดังนี้

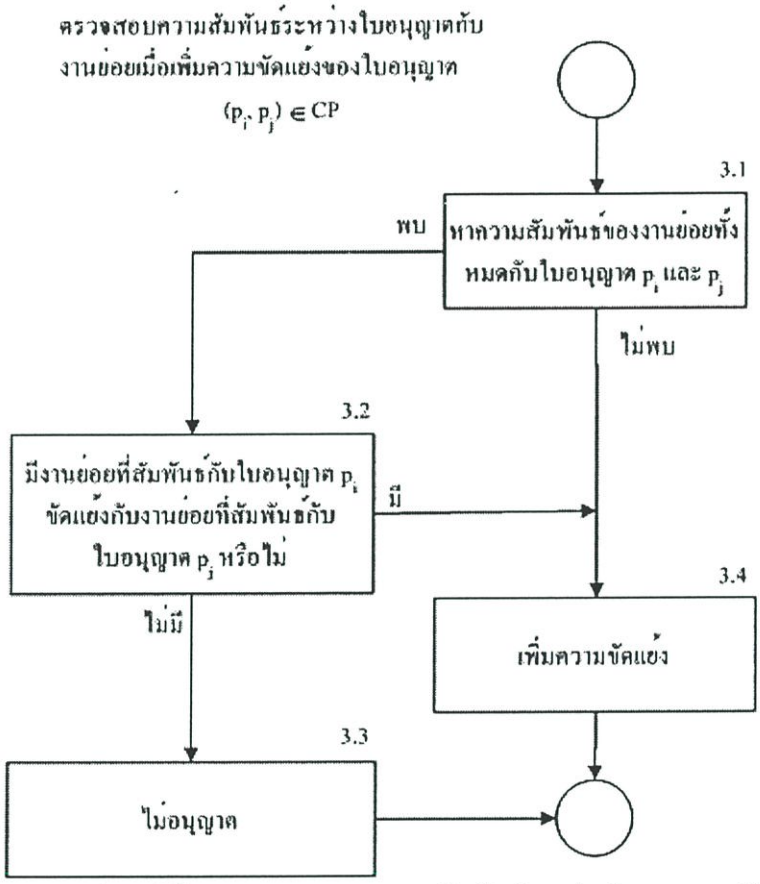
ขั้นตอนที่ 3.1 สมมุติให้เอนทิที p_i กับ p_j มีความขัดแย้งกัน โดยต้องหาความสัมพันธ์ทั้งหมดระหว่างงานย่อยกับเอนทิทีทั้งสอง ถ้าพบให้ทำขั้นตอนที่3.2 แต่ถ้าไม่พบ ให้ทำขั้นตอนที่ 3.4

ขั้นตอนที่ 3.2 ตรวจสอบความสัมพันธ์ระหว่างงานย่อยกับเอนทิทีแรกกับความสัมพันธ์ระหว่างงานย่อยกับเอนทิทีที่สองว่าขัดแย้งกันหรือไม่ ซึ่งตรวจสอบโดยค้นจากคู่งานย่อยที่อยู่ใน

ตาราง CT ถ้าพบว่างานย่อยมีความขัดแย้งกัน จะอนุญาตให้เพิ่มความขัดแย้งระหว่างเอนทิทีทั้งสองได้

ขั้นตอนที่ 3.3 ไม่ให้เพิ่มความขัดแย้งของเอนทิที ที่เป็นสาเหตุให้เกิดความสัมพันธ์ที่ไม่ถูกต้องขึ้น

ขั้นตอนที่ 3.4 เพิ่มความขัดแย้งโดยบันทึกลงในตารางในฐานข้อมูล



รูปที่ 4.12 แสดงอัลกอริทึมการตรวจสอบความสัมพันธ์ระหว่างใบอนุญาตกับงานย่อย

ถ้ามีการกำหนดความขัดแย้งระหว่างสองบทบาท ($(r_i, r_j) \in CR$) ให้ดำเนินตามอัลกอริทึมดังรูปที่ 4.13 ดังนี้

ขั้นตอนที่ 3.1 ตรวจสอบว่าบทบาททั้งสองมีการกำหนดลำดับชั้นของบทบาทไว้แล้วหรือไม่ ถ้ามี ให้ทำขั้นตอนที่ 3.6 ถ้าไม่มี ให้ทำขั้นตอนที่ 3.2

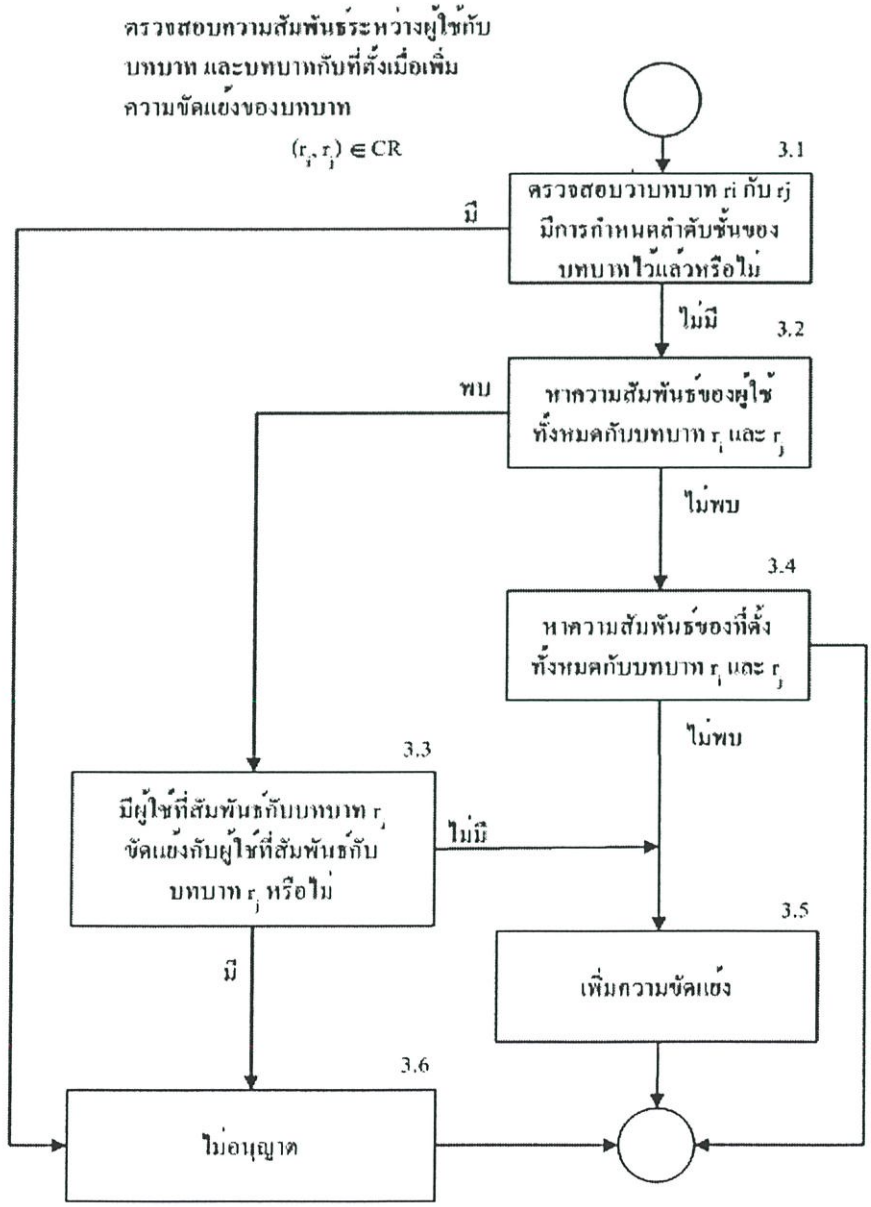
ขั้นตอนที่ 3.2 สมมุติให้บทบาท r_i กับบทบาท r_j ขัดแย้งกัน โดยต้องหาความสัมพันธ์ระหว่างผู้ใช้กับบทบาททั้งสอง ถ้าพบ ให้ทำขั้นตอนที่ 3.3 แต่ถ้าไม่พบ ให้ทำขั้นตอนที่ 3.4

ขั้นตอนที่ 3.3 ตรวจสอบความขัดแย้งระหว่างผู้ใช้ที่กำหนดให้กับบทบาท r_i กับบทบาท r_j ว่ามีหรือไม่ ซึ่งตรวจสอบโดยค้นจากคู่ผู้ใช้ที่อยู่ในตาราง CU ถ้าพบว่าผู้ใช้มีความขัดแย้งกัน จะไม่อนุญาตให้เพิ่มความขัดแย้งระหว่างเอนทิทีทั้งสองได้

ขั้นตอนที่ 3.4 ตรวจสอบความสัมพันธ์ระหว่างที่ตั้งกับบทบาททั้งสอง ซึ่งตรวจสอบได้จากตารางความสัมพันธ์ RL ในฐานข้อมูล ถ้าพบ จะไม่อนุญาตให้เพิ่มความขัดแย้งระหว่างเอนทิทีทั้งสองได้ แต่ถ้าไม่พบ ให้ทำขั้นตอนที่ 3.5

ขั้นตอนที่ 3.5 เพิ่มความขัดแย้งโดยบันทึกลงในตารางในฐานข้อมูล

ขั้นตอนที่ 3.6 ไม่ให้เพิ่มความขัดแย้งของเอนทิที ที่เป็นสาเหตุให้เกิดความสัมพันธ์ที่ไม่ถูกต้องขึ้น



รูปที่ 4.13 แสดงอัลกอริทึมการตรวจสอบความสัมพันธ์ระหว่างผู้ใช้กับบทบาท และบทบาทกับที่ตั้ง

ถ้ามีการกำหนดความขัดแย้งระหว่างสองที่ตั้ง ($(l_i, l_j) \in CL$) ให้ดำเนินตามอัลกอริทึมดังรูปที่ 4.14 ดังนี้

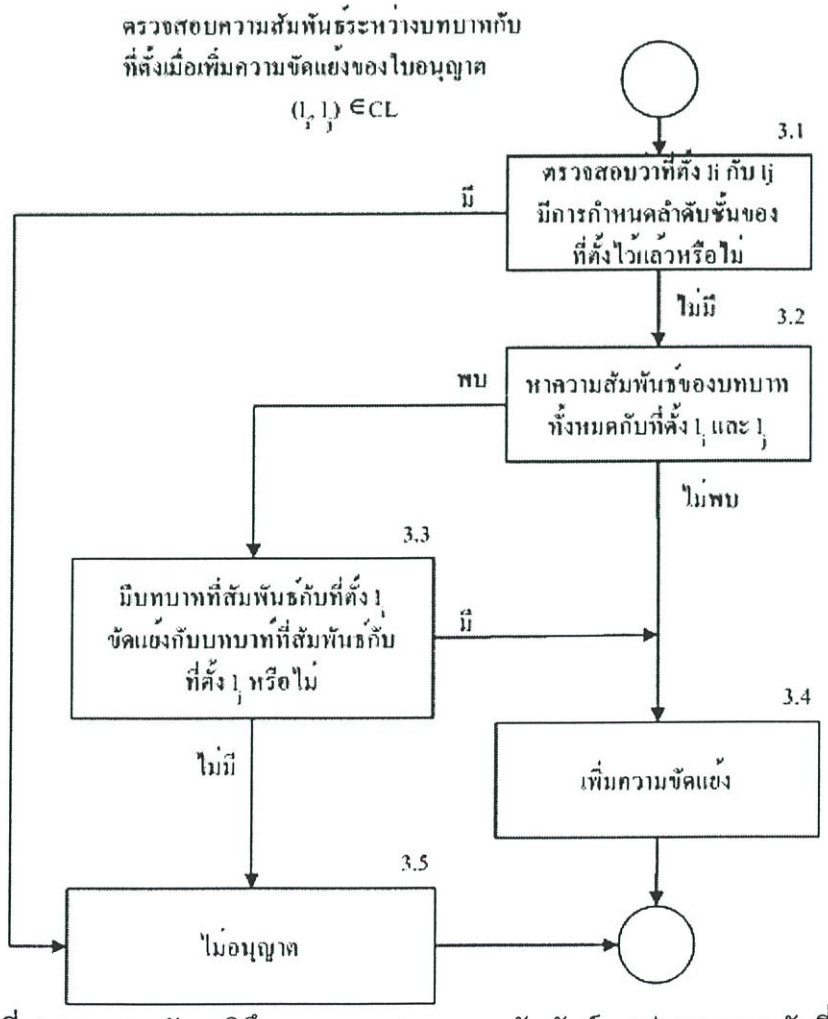
ขั้นตอนที่ 3.1 ตรวจสอบว่าที่ตั้งทั้งสองมีการกำหนดลำดับชั้นของที่ตั้งไว้แล้วหรือไม่ ถ้ามี ให้ทำขั้นตอนที่ 3.5 ถ้าไม่มี ให้ทำขั้นตอนที่ 3.2

ขั้นตอนที่ 3.2 สมมุติให้ที่ตั้ง l_i กับที่ตั้ง l_j ขัดแย้งกัน โดยต้องหาความสัมพันธ์ระหว่างบทบาทกับที่ตั้งทั้งสอง ถ้าพบ ให้ทำขั้นตอนที่ 3.3 แต่ถ้าไม่พบ ให้ทำขั้นตอนที่ 3.4

ขั้นตอนที่ 3.3 ตรวจสอบความขัดแย้งระหว่างบทบาทที่กำหนดให้กับที่ตั้ง l_i กับที่ตั้ง l_j ว่ามีหรือไม่ ซึ่งตรวจสอบโดยค้นจากคู่บทบาทที่อยู่ในตาราง CR ถ้าพบว่าบทบาทมีความขัดแย้งกัน จะอนุญาตให้เพิ่มความขัดแย้งระหว่างเอนทิทีทั้งสอง

ขั้นตอนที่ 3.4 เพิ่มความขัดแย้งโดยบันทึกลงในตารางในฐานข้อมูล

ขั้นตอนที่ 3.5 ไม่ให้เพิ่มความขัดแย้งของเอนทิที ที่เป็นสาเหตุให้เกิดความสัมพันธ์ที่ไม่ถูกต้องขึ้น

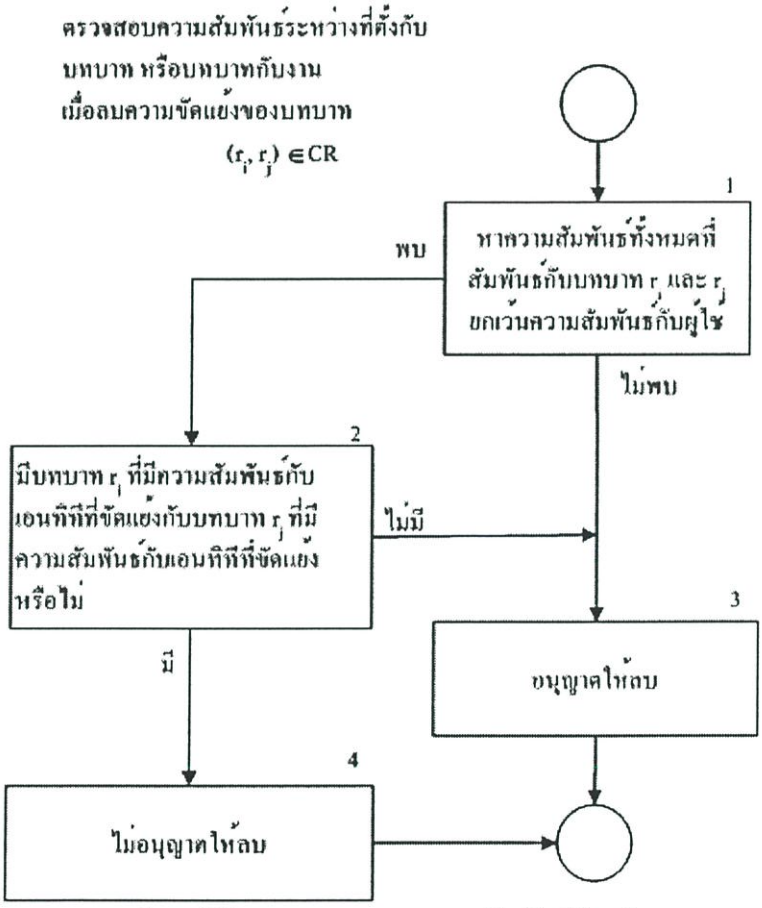


รูปที่ 4.14 แสดงอัลกอริทึมการตรวจสอบความสัมพันธ์ระหว่างบทบาทกับที่ตั้ง

4.3.2 **การลบเอนทิทีที่ขัดแย้ง**

การลบความขัดแย้งเป็นสิ่งที่สำคัญมากต่อความคงสภาพของระบบฯ โดยต้องมีการตรวจสอบที่ถูกต้องก่อนการลบทุกครั้ง ซึ่งการลบความขัดแย้งของผู้ใช้ ที่ตั้ง และใบอนุญาตสามารถลบออกได้ทันทีโดยไม่กระทบต่อความคงสภาพของระบบฯ แต่การลบที่ต้องเข้มงวดเป็นพิเศษจะเกี่ยวข้องกับบทบาท งาน และงานย่อย

การลบบทบาทต้องมีการตรวจสอบให้แน่ใจว่าความสัมพันธ์ที่เหลืออยู่ยังคงมีความถูกต้อง ซึ่งอัลกอริทึมได้แสดงดังรูปที่ 4.15



รูปที่ 4.15 แสดงอัลกอริทึมการตรวจสอบความสัมพันธ์สำหรับการลบบทบาท

ขั้นตอนที่ 1 หาความสัมพันธ์ทั้งหมดระหว่างที่ตั้งและงานกับบทบาท r_i และ r_j ถ้าพบความสัมพันธ์ ให้ทำขั้นตอนที่ 2 ถ้าไม่พบ ให้ลบบทบาทที่ขัดแย้งได้

ขั้นตอนที่ 2 ตรวจสอบที่ตั้งทั้งหมดที่สัมพันธ์กับบทบาท r_i ที่ขัดแย้งกับที่ตั้งที่สัมพันธ์กับบทบาท r_j โดยในขณะเดียวกันก็ตรวจสอบงานทั้งหมดที่สัมพันธ์กับบทบาท r_i ที่ขัดแย้งกับงานที่สัมพันธ์กับบทบาท r_j ซึ่งสามารถตรวจสอบได้โดยค้นหาจากตารางความขัดแย้งในฐานข้อมูล ถ้ามีที่ตั้งหรืองานใดๆ ที่ขัดแย้งกัน จะไม่อนุญาตให้ลบบทบาทที่ขัดแย้งกัน

ขั้นตอนที่ 3 อนุญาตให้ลบแถวที่ขัดแย้งกันออกจากตารางในฐานข้อมูล

ขั้นตอนที่ 4 ไม่อนุญาตให้ลบความขัดแย้งของบทบาทที่เป็นสาเหตุให้ความสัมพันธ์ที่มีอยู่ไม่ถูกต้อง

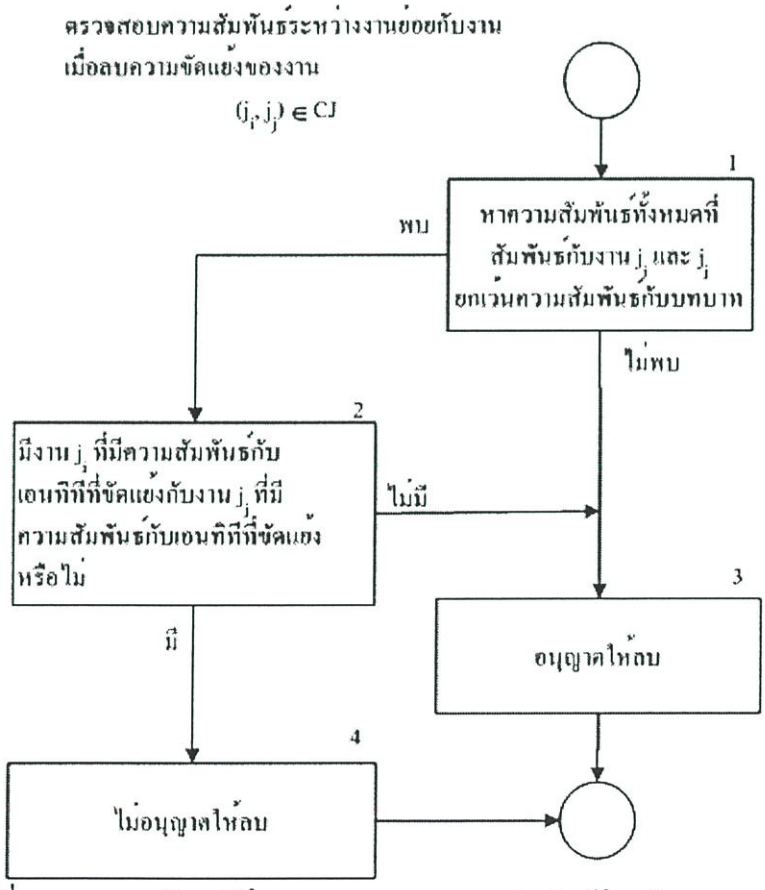
อัลกอริทึมการลบงานที่ขัดแย้งสามารถอธิบายได้ดังรูปที่ 4.16 ดังนี้

ขั้นตอนที่ 1 หาความสัมพันธ์ทั้งหมดระหว่างงานย่อยกับงาน j_i และ j_j ถ้าพบความสัมพันธ์ ให้ทำขั้นตอนที่ 2 ถ้าไม่พบ ให้ลบงานที่ขัดแย้งได้

ขั้นตอนที่ 2 ตรวจสอบงานย่อยที่สัมพันธ์กับงาน j_i ที่ขัดแย้งกับงานย่อยที่สัมพันธ์กับงาน j_j ซึ่งสามารถตรวจสอบได้โดยค้นหาจากตารางความขัดแย้งในฐานข้อมูล ถ้ามีงานย่อยใดๆ ที่ขัดแย้งกัน จะไม่อนุญาตให้ลบงานที่ขัดแย้งกัน

ขั้นตอนที่ 3 อนุญาตให้ลบแถวที่ขัดแย้งกันออกจากตารางในฐานข้อมูล

ขั้นตอนที่ 4 ไม่อนุญาตให้ลบความขัดแย้งของงานที่เป็นสาเหตุให้ความสัมพันธ์ที่มีอยู่ไม่ถูกต้อง



รูปที่ 4.16 แสดงอัลกอริทึมการตรวจสอบความสัมพันธ์สำหรับการลบงาน

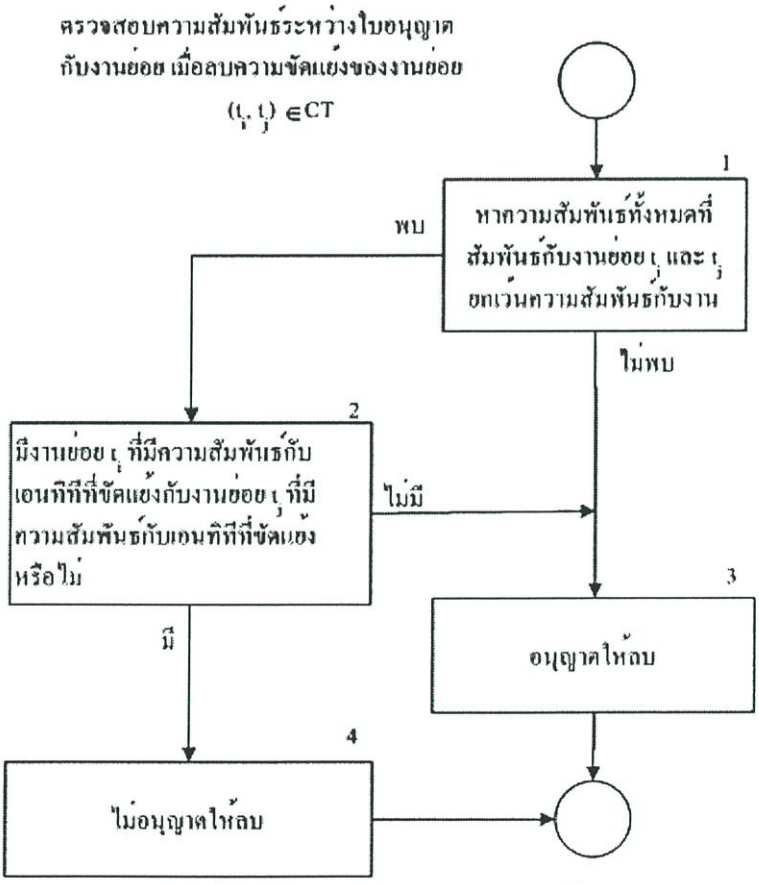
อัลกอริทึมการลบงานย่อยที่ขัดแย้งสามารถอธิบายได้ดังรูปที่ 4.17 ดังนี้

ขั้นตอนที่ 1 หาความสัมพันธ์ทั้งหมดระหว่างใบอนุญาตกับงานย่อย t_i และ t_j ถ้าพบความสัมพันธ์ ให้ทำขั้นตอนที่ 2 ถ้าไม่พบ ให้ลบงานย่อยที่ขัดแย้งได้

ขั้นตอนที่ 2 ตรวจสอบใบอนุญาตที่สัมพันธ์กับงานย่อย t_i ที่ขัดแย้งกับใบอนุญาตที่สัมพันธ์กับงานย่อย t_j ซึ่งสามารถตรวจสอบได้โดยค้นหาจากตารางความขัดแย้งในฐานข้อมูล ถ้ามีใบอนุญาตใดๆ ที่ขัดแย้งกัน จะไม่อนุญาตให้ลบงานย่อยที่ขัดแย้งกัน

ขั้นตอนที่ 3 อนุญาตให้ลบแถวที่ขัดแย้งกันออกจากตารางในฐานข้อมูล

ขั้นตอนที่ 4 ไม่อนุญาตให้ลบความขัดแย้งของงานย่อยที่เป็นสาเหตุให้ความสัมพันธ์ที่มีอยู่ไม่ถูกต้อง



รูปที่ 4.17 แสดงอัลกอริทึมการตรวจสอบความสัมพันธ์สำหรับการลบงานย่อย

ในหัวข้อต่อไปกล่าวถึงการเพิ่มและการลบเอนทิทีทั้งหมดในรูปแบบ ERBAC03

4.4 อัลกอริทึมสำหรับจัดการเอนทิที

4.4.1 การเพิ่มเอนทิที

การเพิ่มเอนทิทีในระบบจะกระทำเมื่อใดก็ได้ โดยเอนทิทีใหม่ที่เพิ่มขึ้นจะไม่มีผลกระทบต่อความคงสภาพของข้อมูลในระบบฯ แต่การลบเอนทิทีต้องระมัดระวังเป็นอย่างมาก ซึ่งต้องแน่ใจว่าเอนทิทีที่ลบนั้นทำให้ความคงสภาพของระบบยังคงอยู่

4.4.2 การลบเอนทิที

เพื่อให้การลบเอนทิทีมีความปลอดภัยที่สุด ต้องตรวจสอบก่อนว่าเอนทิทีที่ลบนั้นมีความสัมพันธ์กับเอนทิทีอื่นๆ หรือไม่ ซึ่งในรูปแบบ ERBAC03 จะไม่ให้ลบเอนทิทีที่มีความสัมพันธ์กับเอนทิทีอื่นๆ โดยเด็ดขาด

อัลกอริทึมการลบเอนทิทีได้แสดงในรูปที่ 4.18 ซึ่งการลบเอนทิทีทั้งหมดแสดงด้วยรูปแบบทางคณิตศาสตร์ดังนี้ ผู้ใช้ ($u_i \notin U$) ที่ตั้ง ($l_i \notin L$) บทบาท ($r_i \notin R$) งาน ($j_i \notin J$) งานย่อย ($t_i \notin T$) และใบอนุญาต ($p_i \notin P$)

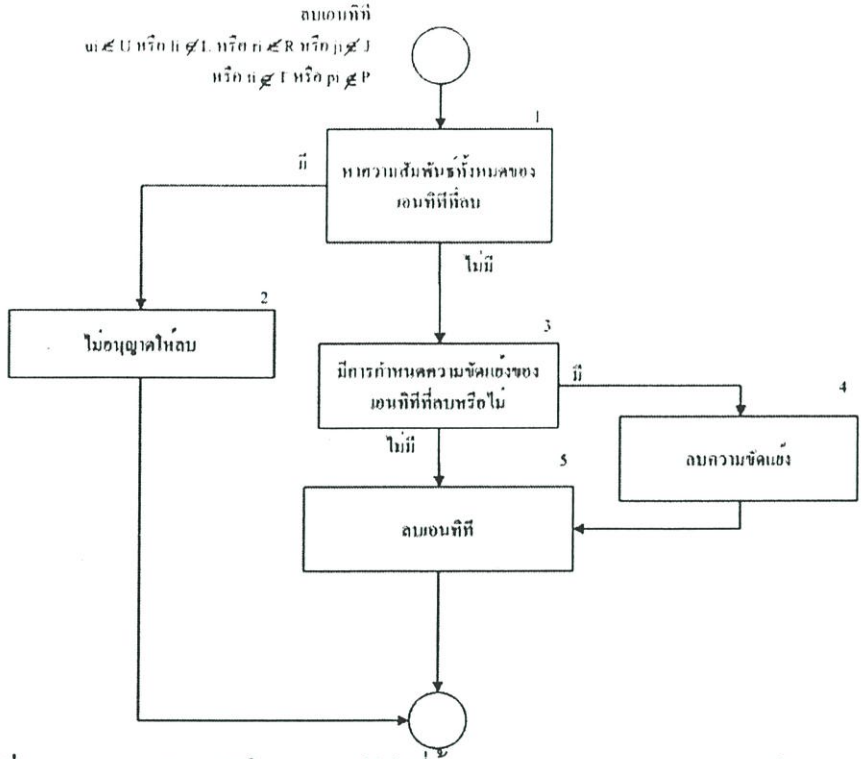
ขั้นตอนที่ 1 ตรวจสอบเอนทิทีที่มีความสัมพันธ์กันทั้งหมด ซึ่งสามารถตรวจสอบจากทุกแถวในตารางความสัมพันธ์ที่ตรงกับเอนทิทีนั้น ถ้าพบแถว แสดงว่าไม่สามารถลบเอนทิทีได้ แต่ถ้าไม่พบ ให้ทำขั้นตอนที่ 3

ขั้นตอนที่ 2 ไม่อนุญาตให้ลบเอนทิที โดยหน้าจอส่วนการจัดการอาจจะแสดงข้อความที่เกิดความผิดพลาดในการลบขึ้น

ขั้นตอนที่ 3 ตรวจสอบความสัมพันธ์ที่ขัดแย้งทั้งหมดในแต่ละเอนทิที ซึ่งสามารถตรวจสอบได้โดยค้นหาจากตารางความขัดแย้งในฐานข้อมูล ถ้าพบความสัมพันธ์ ต้องลบแถวที่ขัดแย้งก่อนจึงสามารถลบเอนทิทีได้ ถ้าไม่พบความสัมพันธ์ ให้ลบเอนทิทีได้ทันที

ขั้นตอนที่ 4 ลบความขัดแย้งทั้งหมดระหว่างเอนทิทีที่ลบกับเอนทิทีอื่นๆ

ขั้นตอนที่ 5 ลบเอนทิทีออกจากตารางในฐานข้อมูล



รูปที่ 4.18 แสดงอัลกอริทึมการลบผู้ใช้ ที่ตั้ง บทบาท งาน งานย่อย และใบอนุญาต

ในบทนี้ได้อธิบายอัลกอริทึมทั้งหมดที่ได้มาจากการกำหนดนิยามและสัจพจน์ในบทที่ 3 ที่ใช้สำหรับรูปแบบ ERBAC03 ส่วนในบทต่อไปได้อธิบายถึงการนำอัลกอริทึมทั้งหมดไปปฏิบัติจริง และยังได้มีการเพิ่มเครื่องมือที่ช่วยให้ผู้บริหารจัดการระบบใช้งานได้ง่ายยิ่งขึ้น

บทที่ 5
การพัฒนาโปรแกรมสำหรับรูปแบบ ERBAC03

ในบทที่ 3 ได้กล่าวถึงนิยามของข้อบังคับด้านความคงสภาพ และในบทที่ 4 ได้อธิบายถึงอัลกอริทึมทั้งหมดสำหรับรูปแบบ ERBAC03 ส่วนในบทนี้ ได้นำอัลกอริทึมที่ออกแบบมาพัฒนาโปรแกรมสำหรับการทดลองในงานวิจัยเล่มนี้ โดยวัตถุประสงค์หลักของการพัฒนาโปรแกรมนี้คือเพื่อใช้ป้องกันการกำหนดเอนทิที ความสัมพันธ์ และความขัดแย้งต่างๆ ของรูปแบบ ERBAC03 ไม่ให้เกิดความผิดพลาดขึ้น

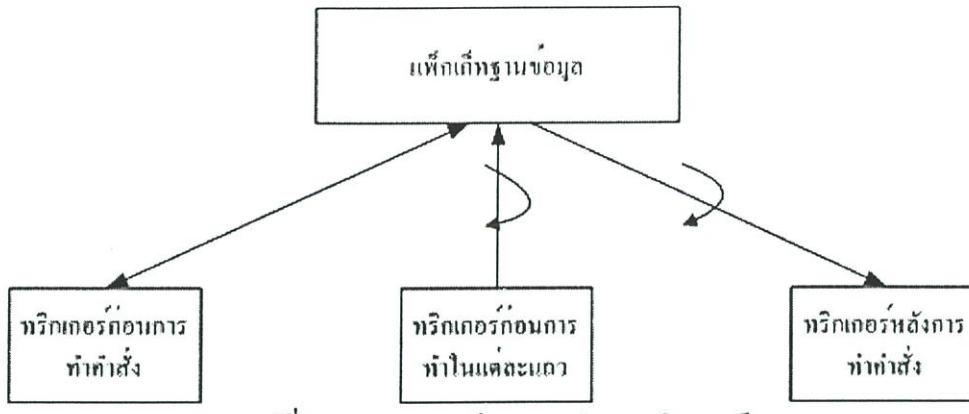
การพัฒนาโปรแกรมนี้ได้แบ่งออกเป็น 2 ส่วนหลักๆ ดังต่อไปนี้

5.1 การพัฒนาโปรแกรมแบบทริกเกอร์ในฐานข้อมูล (Database trigger)

การพัฒนาโปรแกรมประเภทนี้จะกระทำบนฐานข้อมูลที่รองรับคุณสมบัติแบบทริกเกอร์ (Trigger) และใช้ภาษา SQL ในการกระทำคำสั่งทุกคำสั่ง โดยทริกเกอร์หมายถึงโปรแกรมที่ถูกประมวลผลเมื่อมีเหตุการณ์ณ์ใดๆ ตามเงื่อนที่ทริกเกอร์กำหนดให้ทำ และฐานข้อมูลใดที่รองรับข้อบังคับแบบสแตติกหรือแบบไดนามิกได้ [45] ซึ่งงานวิจัยฉบับนี้ได้เลือกที่จะพัฒนาทริกเกอร์บนฐานข้อมูลออราเคิล โดยฐานข้อมูลออราเคิลจะรองรับการทำงานแบบทริกเกอร์ และยังไม่ได้มีการกำหนดข้อบังคับแบบสแตติกไว้เลย

การพัฒนาโปรแกรมแบบทริเกอร์นี้มีข้อดีที่ Perelson [39] ได้กล่าวไว้คือ มีการจัดการคุณสมบัติข้อบังคับที่ฝั่งฐานข้อมูลไม่ใช่ที่ฝั่งไคเอนท์ (Client) ทำให้ลดความคับคั่งของเครือข่าย โดยฝั่งไคเอนท์ไม่ต้องมีข้อมูลการตรวจสอบข้อบังคับทั้งหมดที่มีอยู่เลยทำให้ลดข้อมูลที่ส่งผ่านเครือข่ายได้ อีกทั้งฝั่งไคเอนท์มีการตรวจสอบความถูกต้องในการกำหนดความสัมพันธ์ไม่แม่นยำเหมือนกับฐานข้อมูลที่สามารถควบคุมการเพิ่มหรือลบความสัมพันธ์ได้ และการทำงานที่ฝั่งฐานข้อมูลเมื่อผิดพลาดก็สามารถแก้ไขที่ฝั่งฐานข้อมูลเพียงที่เดียว ซึ่งไม่ต้องไปแก้โปรแกรมที่ฝั่งไคเอนท์เลย ซึ่งถ้ามีหลายไคเอนท์ที่ใช้งานโปรแกรมนั้นอยู่ก็ต้องแก้ให้ครบทุกไคเอนท์

5.1.1 วิธีการออกแบบทริกเกอร์



รูปที่ 5.1 แสดงการกำหนดรูปแบบทริกเกอร์

ทริกเกอร์แรกคือทริกเกอร์ที่กระทำก่อนทำคำสั่งใดๆ โดยเป็นทริกเกอร์ที่กำหนดค่าเริ่มต้นก่อนที่จะมีการเพิ่มหรือลบข้อมูลในตารางในฐานข้อมูล

ทริกเกอร์ที่สองคือทริกเกอร์ที่รับค่าข้อมูลจากทุกๆ แถวในตารางที่มีผลกระทบการค่าที่เพิ่มหรือลบ โดยทริกเกอร์นี้จะกระทำซ้ำแบบวนลูปจนครบจำนวนฟิลด์ในตาราง

ทริกเกอร์ที่สามคือทริกเกอร์ที่กระทำหลักจากได้มีการปฏิบัติคำสั่งไปแล้ว โดยทริกเกอร์นี้จะกระทำซ้ำในทุกๆ แถวที่เพื่อทำการตรวจสอบข้อบังคับที่ได้กำหนดไว้ว่าเมื่อมีการเพิ่มหรือลบข้อมูลแล้วทำให้ข้อมูลในตารางยังคงมีความคงสภาพอยู่หรือไม่

จากการออกแบบทริกเกอร์ทั้งสามนี้ได้แนวคิดมาจาก Perelson [39] มาปรับใช้ให้สอดคล้องกับรูปแบบ ERBAC03 โดยฝั่งไคเอนท์ไม่จำเป็นต้องมีวิธีการตรวจสอบข้อบังคับนี้เลย แต่มีเพียงการขึ้นข้อความแสดงความผิดพลาดที่ได้รับมาจากฝั่งฐานข้อมูลเท่านั้น

5.1.2 หลักการทำงาน

ทริกเกอร์ทั้งหมดนี้ได้ออกแบบตามอัลกอริทึมทั้งหมดในบทที่ 4 โดยผู้บริหารจัดการด้านความปลอดภัยจะเป็นคนเพิ่มความสัมพันธ์และความขัดแย้งทั้งหมดตามนโยบายขององค์กร ซึ่งได้ทดลองโดยใช้ข้อมูลตัวอย่างของบริษัทไปรษณีย์ไทยจำกัดดังต่อไปนี้

ตารางที่ 5.1 แสดงตัวอย่างตารางผู้ใช้

USERID	USER NAME
1	Burin
2	Aradee
3	Bom
4	Pla

ตารางที่ 5.2 แสดงตัวอย่างตารางที่ตั้ง

LOCATIONID	LOCATION NAME
1	Bangsau
2	Nakornpatom
3	Cat
4	Pompap

ตารางที่ 5.3 แสดงตัวอย่างตารางบทบาท

ROLEID	ROLE NAME
1	Chief Post Office
2	Accountant
3	Counter Issuer
4	Office Boy
5	Mail Man

ตารางที่ 5.4 แสดงตัวอย่างตารางงาน

JOBID	JOB NAME
1	Closing End of Day Account
2	Summarize Financial
3	Issuer the Mail
4	Check Stock
5	Send the Mail

ตารางที่ 5.5 แสดงตัวอย่างตารางงานย่อย

TASKID	TASK NAME
1	Counting the Money
2	Check the Old Account
3	Record the Revenue and Expenditure
4	Calculate Summary Account
5	Checking the Mail Address

ตารางที่ 5.6 แสดงตัวอย่างตารางใบอนุญาต

PERMID	PERMISSION NAME
1	Read Account Record
2	Write Account Record
3	Read the Employee Record
4	Read the Transaction Record

มีการสรุปเหตุการณ์ทั้งหมดโดยใช้ข้อมูลตัวอย่างในการสาธิตในตารางที่ 5.7 ดังต่อไปนี้

ตารางที่ 5.7 แสดงการสรุปเหตุการณ์ที่มีในรูปแบบ ERBAC03

	โอเปอร์เรชั่น	หัวข้อ
ความขัดแย้ง	การเพิ่มความขัดแย้งของผู้ใช้	5.1.2.1
	การเพิ่มความขัดแย้งของที่ตั้ง	5.1.2.2
	การเพิ่มความขัดแย้งของบทบาท	5.1.2.3
	การเพิ่มความขัดแย้งของงาน	5.1.2.4
	การเพิ่มความขัดแย้งของงานย่อย	5.1.2.5
	การเพิ่มความขัดแย้งของใบอนุญาต	5.1.2.6
	การลบความขัดแย้งของเอนทิที	5.1.2.7
ลำดับชั้นของที่ตั้งและบทบาท	การเพิ่มลำดับชั้นของที่ตั้ง	5.1.2.8
	การลบลำดับชั้นของที่ตั้ง	5.1.2.9
	การเพิ่มลำดับชั้นของบทบาท	5.1.2.10
	การลบลำดับชั้นของบทบาท	5.1.2.11

ตารางที่ 5.7 แสดงการสรุปเหตุการณ์ที่มีในรูปแบบ ERBAC03 (ต่อ)

	โอเปอร์เรชั่น	หัวข้อ
ความสัมพันธ์	การเพิ่มความสัมพันธ์ระหว่างผู้ใช้กับบทบาท	5.1.2.12
	การเพิ่มความสัมพันธ์ระหว่างที่ตั้งกับบทบาท	5.1.2.13
	การเพิ่มความสัมพันธ์ระหว่างงานกับบทบาท	5.1.2.14
	การเพิ่มความสัมพันธ์ระหว่างงานกับงานย่อย	5.1.2.15
	การเพิ่มความสัมพันธ์ระหว่างงานย่อยกับใบอนุญาต	5.1.2.16
	การลบความสัมพันธ์	5.1.2.17

5.1.2.1 การเพิ่มความขัดแย้งของผู้ใช้

กำหนดให้ผู้ใช้สองคนมีความขัดแย้งกัน โดยให้ผู้ใช้ชื่อ Burin กับ Aradee ซึ่งมีรูปแบบทางคณิตศาสตร์ดังนี้ $(u_1, u_2) \in CU$ ส่วนคำสั่งในการเพิ่มข้อมูลในตาราง CU คือ

```
INSERT INTO CU VALUES(1, 2);
```

ทริกเกอร์ "custatafter_trig" จะตรวจสอบความขัดแย้งว่ามีอยู่แล้วหรือไม่ ซึ่งทริกเกอร์ทั้งหมดจะออกแบบตามอัลกอริทึมที่มีในบทที่ 4 ทุกประการ ถ้ามีความขัดแย้งอยู่แล้วจะแสดงข้อความผิดพลาดคือ

```
ORA-20212: Cannot insert user conflict - the user conflict already exists.
ORA-06512: at "ERBAC03.CUSTATAFTER_TRIG", line 49
ORA-04088: error during execution of trigger 'ERBAC03.CUSTATAFTER_TRIG'
```

ถ้าทริกเกอร์ตรวจสอบว่ามีผู้ใช้ที่มีความสัมพันธ์กับบทบาทที่ขัดแย้งกันอยู่ก่อน และเมื่อพยายามทำให้ผู้ใช้ทั้งสองขัดแย้งกัน ทริกเกอร์จะแสดงข้อความผิดพลาดคือ

```
ORA-20211: Cannot insert user conflict - both users are already associated to conflicting
roles.
ORA-06512: at "ERBAC03.CUSTATAFTER_TRIG", line 43
ORA-04088: error during execution of trigger 'ERBAC03.CUSTATAFTER_TRIG'
```

นอกเหนือจากเหตุการณ์ทั้งสองนี้ จะสามารถเพิ่มความขัดแย้งระหว่างผู้ใช้ได้

5.1.2.2 การเพิ่มความขัดแย้งของที่ตั้ง

กำหนดให้ที่ตั้งสองที่มีความขัดแย้งกัน โดยให้ที่ตั้งชื่อ Nakornpatom กับ Cat ซึ่งมีรูปแบบทางคณิตศาสตร์ดังนี้ $(l_2, l_3) \in CL$ ส่วนคำสั่งในการเพิ่มข้อมูลในตาราง CL คือ

```
INSERT INTO CL VALUES(2, 3);
```

ทริกเกอร์ "ctstatafter_trig" จะตรวจสอบความขัดแย้งว่ามีอยู่แล้วหรือไม่ ถ้ามีความขัดแย้งอยู่แล้วจะแสดงข้อความผิดพลาดคือ

ORA-20226: Cannot insert location conflict - the location conflict already exists.

ORA-06512: at "ERBAC03.CLSTATAFTER_TRIG", line 60

ORA-04088: error during execution of trigger 'ERBAC03.CLSTATAFTER_TRIG'

ถ้าทริกเกอร์ตรวจสอบว่ามีที่ตั้งที่มีความสัมพันธ์กับบทบาทที่ขัดแย้งกันอยู่ก่อน และเมื่อพยายามทำให้ที่ตั้งทั้งสองขัดแย้งกัน ทริกเกอร์จะแสดงข้อความผิดพลาดคือ

ORA-20225: Cannot insert location conflict - both locations are already associated to conflicting roles.

ORA-06512: at "ERBAC03.CLSTATAFTER_TRIG", line 54

ORA-04088: error during execution of trigger 'ERBAC03.CLSTATAFTER_TRIG'

นอกเหนือจากเหตุการณ์ทั้งสองนี้ จะสามารถเพิ่มความขัดแย้งระหว่างที่ตั้งได้

5.1.2.3 การเพิ่มความขัดแย้งของบทบาท

กำหนดให้บทบาทสองที่มีความขัดแย้งกัน โดยให้บทบาทชื่อ Chief Post Office กับ Accountant ซึ่งมีรูปแบบทางคณิตศาสตร์ดังนี้ $(r_1, r_2) \in CR$ ส่วนคำสั่งในการเพิ่มข้อมูลในตาราง CR คือ

INSERT INTO CR VALUES(1, 2);

ทริกเกอร์ "crstatafter_trig" จะตรวจสอบความขัดแย้งว่ามีอยู่แล้วหรือไม่ ถ้ามีความขัดแย้งอยู่แล้วจะแสดงข้อความผิดพลาดคือ

ORA-20210: Cannot insert role conflict - the role conflict already exists.

ORA-06512: at "ERBAC03.CRSTATAFTER_TRIG", line 60

ORA-04088: error during execution of trigger 'ERBAC03.CRSTATAFTER_TRIG'

ต่อมาทริกเกอร์จะตรวจสอบว่าสองบทบาทนั้นเป็นส่วนหนึ่งในลำดับชั้นของบทบาทเดียวกันหรือไม่ ถ้ามี ทริกเกอร์จะแสดงข้อความผิดพลาดคือ

ORA-20208: Cannot insert role conflict - both roles are already part of a role network.

ORA-06512: at "ERBAC03.CRSTATAFTER_TRIG", line 24

ORA-04088: error during execution of trigger 'ERBAC03.CRSTATAFTER_TRIG'

ถ้าทริกเกอร์ตรวจสอบว่ามีบทบาทที่มีความสัมพันธ์กับผู้ใช้ที่ขัดแย้งกันอยู่ก่อน และเมื่อพยายามทำให้ที่ตั้งทั้งสองขัดแย้งกัน ทริกเกอร์จะแสดงข้อความผิดพลาดคือ

ORA-20209: Cannot insert role conflict - both roles are already associated to conflicting users.

ORA-06512: at "ERBAC03.CRSTATAFTER_TRIG", line 54

ORA-04088: error during execution of trigger 'ERBAC03.CRSTATAFTER_TRIG'

นอกเหนือจากเหตุการณ์ทั้งสองนี้ จะสามารถเพิ่มความขัดแย้งระหว่างบทบาทได้

5.1.2.4 การเพิ่มความขัดแย้งของงาน

กำหนดตัวอย่างงานที่ขัดแย้งกันคือ ให้งาน Closing End of Day Account ขัดแย้งกับงาน Summarize Financial และงาน Check Stock โดยมีรูปแบบทางคณิตศาสตร์ดังนี้คือ $(j_1, j_2) \in$ CJ กับ $(j_1, j_4) \in$ CJ คำสั่งที่ใช้ในการเพิ่มสองแถวนี้คือ

```
INSERT INTO CJ VALUES(1, 2);
INSERT INTO CJ VALUES(1, 4);
```

ขั้นแรกทริกเกอร์ "cjstatafter_trig" จะตรวจสอบความขัดแย้งว่ามีอยู่แล้วหรือไม่ ต่อมาจะตรวจสอบความสัมพันธ์ระหว่างงานกับบทบาทใดๆ ว่ามีหรือไม่ ถ้ามี ทริกเกอร์จะแสดงข้อความผิดพลาดคือ

ORA-20221: Cannot insert job conflict - one or more of the associated roles of the two jobs are not conflicting.

ORA-06512: at "ERBAC03.CJSTATAFTER_TRIG", line 47

ORA-04088: error during execution of trigger 'ERBAC03.CJSTATAFTER_TRIG'

ถ้าไม่มีความสัมพันธ์ระหว่างบทบาทเลยให้ตรวจสอบความสัมพันธ์ระหว่างงานกับงานย่อยว่ามีหรือไม่ ถ้ามี ทริกเกอร์จะแสดงข้อความผิดพลาดคือ

ORA-20222: Cannot insert job conflict - one or more of the associated tasks of the two tasks are not conflicting.

ORA-06512: at "ERBAC03.CJSTATAFTER_TRIG", line 73

ORA-04088: error during execution of trigger 'ERBAC03.CJSTATAFTER_TRIG'

นอกเหนือจากเหตุการณ์ทั้งสองนี้ จะสามารถเพิ่มความขัดแย้งระหว่างงานได้

5.1.2.5 การเพิ่มความขัดแย้งของงานย่อย

กำหนดตัวอย่างงานย่อยที่ขัดแย้งกันคือ ให้งานย่อย Counting the Money ขัดแย้งกับงานย่อย Record the Revenue and Expenditure โดยมีรูปแบบทางคณิตศาสตร์ดังนี้คือ $(t_1, t_3) \in$ CT คำสั่งที่ใช้ในการเพิ่มสองแถวนี้คือ

```
INSERT INTO CT VALUES(1, 3);
```

ขั้นแรกทริกเกอร์ "ctstatafter_trig" จะตรวจสอบความขัดแย้งว่ามีอยู่แล้วหรือไม่ ต่อมาจะตรวจสอบความสัมพันธ์ระหว่างงานย่อยกับงานใดๆ ว่ามีหรือไม่ ถ้ามี ทริกเกอร์จะแสดงข้อความผิดพลาดคือ

ORA-20217: Cannot insert task conflict - one or more of the associated jobs of the two tasks are not conflicting.

ORA-06512: at "ERBAC03.CTSTATAFTER_TRIG", line 47

ORA-04088: error during execution of trigger 'ERBAC03.CTSTATAFTER_TRIG'

ถ้าไม่มีความสัมพันธ์ระหว่างงานเลยให้ตรวจสอบความสัมพันธ์ระหว่างงานย่อยกับใบอนุญาตว่ามีหรือไม่ ถ้ามี ทริกเกอร์จะแสดงข้อความผิดพลาดคือ

ORA-20218: Cannot insert task conflict - one or more of the associated perms of the two tasks are not conflicting.

ORA-06512: at "ERBAC03.CTSTATAFTER_TRIG", line 73

ORA-04088: error during execution of trigger 'ERBAC03.CTSTATAFTER_TRIG'

นอกเหนือจากเหตุการณ์ทั้งสองนี้ จะสามารถเพิ่มความขัดแย้งระหว่างงานย่อยได้

5.1.2.6 การเพิ่มความขัดแย้งของใบอนุญาต

กำหนดตัวอย่างใบอนุญาตที่ขัดแย้งกันคือ ให้ใบอนุญาต Read Account Record ขัดแย้งกับใบอนุญาต Write Account Record โดยมีรูปแบบทางคณิตศาสตร์ดังนี้คือ $(p_1, p_2) \in CP$ คำสั่งที่ใช้ในการเพิ่มสองแถวนี้คือ

```
INSERT INTO CP VALUES(1, 2);
```

ขั้นแรกทริกเกอร์ "cpstatafter_trig" จะตรวจสอบความขัดแย้งว่ามีอยู่แล้วหรือไม่ ต่อมาจะตรวจสอบความสัมพันธ์ระหว่างใบอนุญาตกับงานย่อยใดๆ ว่ามีหรือไม่ ถ้ามีทริกเกอร์จะแสดงข้อความผิดพลาดคือ

ORA-20214: Cannot insert permission conflict - one or more of the associated tasks of the two permissions are not conflicting.

ORA-06512: at "ERBAC03.CPSTATAFTER_TRIG", line 47

ORA-04088: error during execution of trigger 'ERBAC03.CPSTATAFTER_TRIG'

นอกเหนือจากเหตุการณ์ทั้งสองนี้ จะสามารถเพิ่มความขัดแย้งระหว่างงานย่อยได้

5.1.2.7 การลบความขัดแย้งของเอนทิที

การลบความขัดแย้งที่มีผลต่อความคงสภาพของระบบฯ คือการลบความขัดแย้งของบทบาทงานและงานย่อยเท่านั้น นอกเหนือจากเอนทิทีเหล่านี้สามารถลบความขัดแย้งได้ทันที การลบ

ความขัดแย้งของบทบาทจะไม่สามารถลบได้ถ้าบทบาทแรกที่มีความสัมพันธ์กับที่ตั้งหรืองานที่ขัดแย้งกันกับบทบาทที่สองที่มีความสัมพันธ์กับที่ตั้งหรืองาน หมายความว่าถ้าความขัดแย้งของที่ตั้งหรืองานสัมพันธ์กับความขัดแย้งของบทบาท จะไม่สามารถลบความขัดแย้งของบทบาทได้ ถ้าผู้บริหารจัดการระบบความปลอดภัยพยายามที่จะลบความขัดแย้งของบทบาท ทริกเกอร์จะแสดงข้อความผิดพลาดคือ

ORA-20227: Cannot delete role conflict - the roles form part of associations to locations or jobs that will become invalid if the role conflict is deleted.

ORA-06512: at "ERBAC03.DCRSTATAFTER_TRIG", line 90

ORA-04088: error during execution of trigger 'ERBAC03.DCRSTATAFTER_TRIG'

ส่วนการลบความขัดแย้งของงานนั้น จะลบได้ก็ต่อเมื่อไม่มีความขัดแย้งของงานสัมพันธ์กับความขัดแย้งของงานย่อย แต่ถ้ามีความสัมพันธ์กัน ทริกเกอร์จะแสดงข้อความผิดพลาดคือ

ORA-20228: Cannot delete job conflict - the jobs form part of associations to tasks that will become invalid if the job conflict is deleted.

ORA-06512: at "ERBAC03.DCJSTATAFTER_TRIG", line 40

ORA-04088: error during execution of trigger 'ERBAC03.DCJSTATAFTER_TRIG'

สุดท้ายเป็นการลบความขัดแย้งของงานย่อย จะลบได้ก็ต่อเมื่อไม่มีความขัดแย้งของงานย่อยสัมพันธ์กับความขัดแย้งของงาน แต่ถ้ามีความสัมพันธ์กัน ทริกเกอร์จะแสดงข้อความผิดพลาดคือ

ORA-20229: Cannot delete task conflict - the tasks form part of associations to permissions that will become invalid if the task conflict is deleted.

ORA-06512: at "ERBAC03.DCTSTATAFTER_TRIG", line 40

ORA-04088: error during execution of trigger 'ERBAC03.DCTSTATAFTER_TRIG'

5.1.2.8 การเพิ่มลำดับชั้นของที่ตั้ง

ขั้นแรกในการเพิ่มลำดับชั้นของที่ตั้งต้องกำหนดชื่อให้กับลำดับชั้นของที่ตั้งก่อน โดยเพิ่มในตารางที่ชื่อ "Locationnet" ซึ่งในตัวอย่างนี้กำหนดชื่อลำดับชั้นของที่ตั้งว่า "ThailandPost" และจะมีการกำหนดรหัสให้กับชื่อนี้ด้วย ซึ่งรหัสนี้จะใช้ในการอ้างอิงสำหรับการเพิ่มที่ตั้งเข้าไปในลำดับชั้นของที่ตั้งทุกครั้ง โดยมีคำสั่งในการบันทึกข้อมูลลงในตารางดังนี้

INSERT INTO LOCATIONNET VALUES(1, 'ThailandPost');

ตารางที่ใช้เก็บข้อมูลลำดับชั้นของที่ตั้งมีชื่อว่า "LH" โดยที่ตั้งแรกที่กำหนดให้กับลำดับชั้นของที่ตั้งต้องเป็นที่ตั้งเริ่มแรก (Root location) และวิธีการกำหนดจะกระทำโดยใส่ค่าว่างให้เป็นที่ตั้งหลัก ส่วนที่ตั้งเริ่มแรกให้เป็นที่ตั้งรอง ในตัวอย่างนี้ กำหนดให้ที่ตั้ง "Cat" เป็นที่ตั้งเริ่มแรก ซึ่งมีรูปแบบคำสั่ง SQL ดังนี้

INSERT INTO LH VALUES(1, null, 3);

การเพิ่มข้อมูลนี้จะกระทำไม่ได้ถ้ายังไม่ได้มีการกำหนดชื่อของลำดับชั้นของที่ตั้งก่อน ซึ่งมีข้อบังคับความคงสภาพถูกควบคุมโดยคีย์นอก และทริกเกอร์ "lhstatafter_trig" จะตรวจสอบข้อมูลที่เพิ่มทุกครั้งว่าละเมิดความคงสภาพหรือไม่ โดยเริ่มจากตรวจสอบที่ตั้งแรกเริ่มว่ามีอยู่จริงหรือไม่ ถ้ามีอยู่แล้ว ทริกเกอร์จะแสดงข้อความผิดพลาดดังนี้

ORA-20236: Cannot insert location into location network - root location already exists for the particular location network.

ORA-06512: at "ERBAC03.LHSTATAFTER_TRIG", line 42

ORA-04088: error during execution of trigger 'ERBAC03.LHSTATAFTER_TRIG'

ที่ตั้งที่จะเพิ่มต่อจากที่ตั้งแรกเริ่มนี้ กำหนดให้เป็นที่ "Nakornpatom" ซึ่งกระทำโดยมีคำสั่งดังนี้

INSERT INTO LH VALUES(1, 3, 2);

ทริกเกอร์ "lhstatafter_trig" จะตรวจสอบที่ตั้งแรกเริ่มก่อนเป็นอันดับแรกว่ามีอยู่แล้วหรือไม่ ซึ่งได้กระทำตามอัลกอริทึมทั้งหมดที่เกี่ยวข้องกับลำดับชั้นของที่ตั้งที่ออกแบบไว้ในบทที่ 4 และเมื่อพบที่ตั้งเริ่มแรกแล้ว ต่อมาให้ตรวจสอบที่ตั้งหลักที่จะมีการเพิ่มข้อมูลว่าเป็นส่วนหนึ่งของลำดับชั้นของที่ตั้งหรือไม่ และที่ตั้งหลักนั้นเคยเป็นที่ตั้งรองมาก่อนหรือไม่ด้วย ในกรณีนี้ที่ตั้งหลักคือ "Cat" ซึ่งเป็นที่ตั้งเริ่มแรกด้วย เนื่องจากมีการกำหนดค่าว่างให้กับที่ตั้งนี้ ถ้าที่ตั้งหลักยังไม่ได้ถูกกำหนดไว้ในลำดับชั้นของที่ตั้ง เช่น การทำคำสั่ง SQL ดังนี้

INSERT INTO LH VALUES(1, 1, 2);

ทริกเกอร์จะแสดงข้อความผิดพลาดขึ้นทันทีดังนี้

ORA-20238: Cannot insert location into location network - the parent location is not an existing child location.

ORA-06512: at "ERBAC03.LHSTATAFTER_TRIG", line 87

ORA-04088: error during execution of trigger 'ERBAC03.LHSTATAFTER_TRIG'

ต่อจากนั้นทริกเกอร์การอ้างถึงแบบย้อนกลับ (Circular reference) ว่ามีหรือไม่ ถ้ามีจะแสดงข้อความผิดพลาดดังนี้

ORA-20239: Cannot insert location into location network - it will cause a circular reference to occur.

ORA-06512: at "ERBAC03.LHSTATAFTER_TRIG", line 94

ORA-04088: error during execution of trigger 'ERBAC03.LHSTATAFTER_TRIG'

สุดท้ายจะเป็นการตรวจสอบว่าที่ตั้งที่เพิ่มมีความขัดแย้งกับที่ตั้งที่มีอยู่เดิมในลำดับชั้นของที่ตั้งหรือไม่ ถ้ามีความขัดแย้งกัน จะแสดงข้อความผิดพลาดคือ

ORA-20237: Cannot insert location into location network – conflicting location(s) are already present in the location network.

ORA-06512: at "ERBAC03.LHSTATAFTER_TRIG", line 81

ORA-04088: error during execution of trigger 'ERBAC03.LHSTATAFTER_TRIG'

การตรวจสอบนี้เพื่อให้เหมือนกับทฤษฎีลำดับชั้นของบทบาทที่ได้กำหนดไว้ในบทที่ 2 ที่กล่าวว่าบทบาทพ่อสืบทอดใบอนุญาตมาจากบทบาทลูก ทำให้ทั้งสองบทบาทจะขัดแย้งกันไม่ได้เพื่อป้องกันการทุจริตของผู้ใช้ จากทฤษฎีลำดับชั้นของบทบาทนี้เองจึงนำมาประยุกต์ใช้กับการกำหนดลำดับชั้นของที่ตั้งได้อย่างเหมาะสม ส่วนการกำหนดระดับความปลอดภัยให้กับที่ตั้งนั้น สามารถนำมาประยุกต์ใช้ในการกำหนดลำดับชั้นของที่ตั้งได้โดยพิจารณาจากการกำหนดที่ตั้งหลักกับที่ตั้งรอง ที่ตั้งหลักก็จะมีการสืบทอดบทบาทที่มีทั้งหมดจากที่ตั้งรอง และที่ตั้งที่สืบทอดบทบาทกันนั้นต้องไม่ขัดแย้งกันด้วย

5.1.2.9 การลบลำดับชั้นของที่ตั้ง

ถ้าต้องการลบที่ตั้งออกจากลำดับชั้นของที่ตั้ง ทริกเกอร์ "dlhstatafter_trig" จะตรวจสอบว่าที่ตั้งที่จะลบนั้นไม่มีที่ตั้งรองกำหนดให้กับตัวมันเองในลำดับชั้นของที่ตั้ง ซึ่งเมื่อกระทำคำสั่งดังต่อไปนี้

DELETE FROM LH WHERE LPARENTID IS NULL AND LOCATIONNETID = 1 AND LCHILDID = 3;

จะแสดงข้อความผิดพลาดดังนี้

ORA-20241: Cannot delete location association - children associations already exist for the location been deleted.

ORA-06512: at "ERBAC03.DLHSTATAFTER_TRIG", line 17

ORA-04088: error during execution of trigger 'ERBAC03.DLHSTATAFTER_TRIG'

ถ้าไม่มีที่ตั้งกำหนดอยู่ในลำดับชั้นของที่ตั้งเลย ก็สามารถลบทั้งลำดับชั้นของที่ตั้งนั้นได้ทันที

5.1.2.10 การเพิ่มลำดับชั้นของบทบาท

วิธีการทำงานของอัลกอริทึมนี้จะเหมือนกับการเพิ่มลำดับชั้นของที่ตั้งทุกประการ แต่ใช้ทริกเกอร์ "rhstatafter_trig" ในการควบคุมความถูกต้องของข้อมูลที่เพิ่ม โดยเริ่มจากตรวจสอบบทบาทแรกเริ่มว่ามีอยู่จริงหรือไม่ ถ้ามีอยู่แล้ว ทริกเกอร์จะแสดงข้อความผิดพลาดดังนี้

ORA-20217: Cannot insert role into role network - root role already exists for the particular role network.

ORA-06512: at "ERBAC03.RHSTATAFTER_TRIG", line 42

ORA-04088: error during execution of trigger 'ERBAC03.RHSTATAFTER_TRIG'

ต่อมาให้ตรวจสอบบทบาทพ่อที่จะมีการเพิ่มข้อมูลว่าเป็นส่วนหนึ่งของลำดับชั้นของบทบาทหรือไม่ และบทบาทพ่อนั้นเคยเป็นบทบาทลูกมาก่อนหรือไม่ด้วย ถ้าบทบาทพ่อยังไม่ได้ถูกกำหนดไว้ในลำดับชั้นของบทบาท ทริกเกอร์จะแสดงข้อความผิดพลาดขึ้นทันทีดังนี้

ORA-20233: Cannot insert role into role network - the parent role is not an existing child role.

ORA-06512: at "ERBAC03.RHSTATAFTER_TRIG", line 87

ORA-04088: error during execution of trigger 'ERBAC03.RHSTATAFTER_TRIG'

ต่อจากนั้นทริกเกอร์การอ้างถึงแบบย้อนกลับ (Circular reference) ว่ามีหรือไม่ ถ้ามีจะแสดงข้อความผิดพลาดดังนี้

ORA-20234: Cannot insert role into role network - it will cause a circular reference to occur.

ORA-06512: at "ERBAC03.RHSTATAFTER_TRIG", line 94

ORA-04088: error during execution of trigger 'ERBAC03.RHSTATAFTER_TRIG'

สุดท้ายจะเป็นการตรวจสอบว่าบทบาทที่เพิ่มมีความขัดแย้งกับบทบาทที่มีอยู่เดิมในลำดับชั้นของบทบาทหรือไม่ ถ้ามีความขัดแย้งกัน จะแสดงข้อความผิดพลาดคือ

ORA-20232: Cannot insert role into role network - conflicting role(s) are already present in the role network.

ORA-06512: at "ERBAC03.RHSTATAFTER_TRIG", line 81

ORA-04088: error during execution of trigger 'ERBAC03.RHSTATAFTER_TRIG'

5.1.2.11 การลบลำดับชั้นของบทบาท

ถ้าต้องการลบบทบาทออกจากลำดับชั้นของบทบาท ทริกเกอร์ “drhstatafter_trig” จะตรวจสอบว่าบทบาทที่จะลบนั้นไม่มีบทบาทลูกกำหนดให้กับตัวมันเองในลำดับชั้นของบทบาท ซึ่งเมื่อกระทำคำสั่งดังต่อไปนี้

```
DELETE FROM RH WHERE PARENTID IS NULL AND ROLENETID = 1 AND
CHILDID = 3;
```

จะแสดงข้อความผิดพลาดดังนี้

ORA-20240: Cannot delete role association - children associations already exist for the role been deleted.

ORA-06512: at "ERBAC03.DRHSTATAFTER_TRIG", line 17

ORA-04088: error during execution of trigger 'ERBAC03.DRHSTATAFTER_TRIG'

ถ้าไม่มีบทบาทกำหนดอยู่ในลำดับชั้นของบทบาทเลย ก็สามารถลบทั้งลำดับชั้นของบทบาทนั้นได้ทันที

5.1.2.12 การเพิ่มความสัมพันธ์ระหว่างผู้ใช้กับบทบาท

กำหนดตัวอย่างดังตารางที่ 5.8 ดังต่อไปนี้

ตารางที่ 5.8 แสดงตัวอย่างการกำหนดความสัมพันธ์ระหว่างผู้ใช้กับบทบาท

ผู้ใช้	บทบาท
Burin	Chief Post Office
Aradee	Accountant

ผู้ใช้สัมพันธ์กับบทบาทถูกแทนด้วยรูปแบบทางคณิตศาสตร์คือ $(u_i, r_i) \in UA$ โดยมีคำสั่งในการเพิ่มข้อมูลดังนี้

คำสั่งการกำหนดให้ผู้ใช้ "Burin" ได้รับบทบาท "Chief Post Office"

```
INSERT INTO UA VALUES(1, 1);
```

คำสั่งการกำหนดให้ผู้ใช้ "Aradee" ได้รับบทบาท "Accountant"

```
INSERT INTO UA VALUES(2, 2);
```

การเพิ่มข้อมูลนี้จะมีทริกเกอร์คอยตรวจสอบความถูกต้องชื่อว่า "uastatafter_trig" ซึ่งอัลกอริทึมทั้งหมดที่เกี่ยวข้องกับการเพิ่มผู้ใช้ให้กับบทบาทนี้ได้แสดงอยู่ในบทที่ 4 โดยทริกเกอร์นี้จะแสดงข้อความผิดพลาดขึ้นถ้าเกิดเหตุการณ์ที่ผู้ใช้ที่ขัดแย้งถูกกำหนดให้กับบทบาทที่ขัดแย้งกันซึ่งไม่สามารถกระทำได้ เนื่องจากถ้ากำหนดให้ผู้ใช้ "Burin" ขัดแย้งกับผู้ใช้ "Aradee" โดยมีบทบาทที่ขัดแย้งกันคือ "Chief Post Office" กับ "Accountant" ถ้าผู้ใช้ "Burin" ถูกกำหนดให้ได้รับบทบาท "Chief Post Office" แล้วผู้ใช้ "Aradee" จะถูกกำหนดให้กับบทบาท "Accountant" ไม่ได้ เนื่องจากว่าได้มีการกำหนดให้ผู้ใช้ทั้งสองขัดแย้งกันก่อนหน้านี้ ซึ่งการที่ผู้ใช้ขัดแย้งกันหมายถึงว่ามีความเป็นไปได้ที่ผู้ใช้ทั้งสองจะสมรู้ร่วมคิดกันทุจริต เพราะฉะนั้นถ้าทั้งสองคนได้รับบทบาทที่ขัดแย้งกันแล้วจะทำให้เกิดการทุจริตขึ้นได้ โดยข้อความผิดพลาดที่แสดงคือ

ORA-20202: Cannot insert user to role association - a conflicting user is already associated to a conflicting role.

ORA-06512: at "ERBAC03.UASTATAFTER_TRIG", line 72

ORA-04088: error during execution of trigger 'ERBAC03.UASTATAFTER_TRIG'

อีกกรณีหนึ่งถ้าผู้ใช้ได้รับบทบาทแล้ว และมีการกำหนดบทบาทที่ขัดแย้งกับบทบาทที่ผู้ใช้นั้นได้รับอีก ทริกเกอร์จะแสดงข้อความผิดพลาดดังนี้

ORA-20201: Cannot insert user to role association because user is being conflict

ORA-06512: at "ERBAC03.UASTATAFTER_TRIG", line 55

ORA-04088: error during execution of trigger 'ERBAC03.UASTATAFTER_TRIG'

5.1.2.13 การเพิ่มความสัมพันธ์ระหว่างที่ตั้งกับบทบาท

กำหนดตัวอย่างดังตารางที่ 5.9 ดังต่อไปนี้

ตารางที่ 5.9 แสดงตัวอย่างการกำหนดความสัมพันธ์ระหว่างที่ตั้งกับบทบาท

ที่ตั้ง	บทบาท
Cat	Chief Post Office
Nakornpatom	Accountant

ที่ตั้งสัมพันธ์กับบทบาทถูกแทนด้วยรูปแบบทางคณิตศาสตร์คือ $(l_i, r_i) \in RL$ โดยมีคำสั่งในการเพิ่มข้อมูลดังนี้

คำสั่งการกำหนดให้บทบาท “Chief Post Office” สามารถปฏิบัติงานที่ที่ตั้ง Cat ได้

```
INSERT INTO RL VALUES(3, 1);
```

คำสั่งการกำหนดให้บทบาท “Accountant”สามารถปฏิบัติงานที่ที่ตั้ง Nakornpatom ได้

```
INSERT INTO RL VALUES(2, 2);
```

การเพิ่มข้อมูลนี้จะมีทริกเกอร์คอยตรวจสอบความถูกต้องชื่อว่า “rlstatafter_trig”โดยทริกเกอร์นี้จะแสดงข้อความผิดพลาดขึ้นถ้าเกิดเหตุการณ์ที่ที่ตั้งที่ขัดแย้งถูกกำหนดให้กับบทบาทที่ไม่ขัดแย้งกัน เช่น กำหนดให้ที่ตั้ง “Cat” กับ “Nakornpatom” ขัดแย้งกัน โดยในแต่ละที่ตั้งต่างกำหนดให้กับบทบาททั้งสองที่ไม่ขัดแย้งกันเลย ซึ่งไม่สามารถกระทำได้ โดยข้อความผิดพลาดที่แสดงคือ

ORA-20203: Cannot insert location to role association - a conflicting location must be assigned to a conflicting role.

ORA-06512: at "ERBAC03.RLSTATAFTER_TRIG", line 49

ORA-04088: error during execution of trigger 'ERBAC03.RLSTATAFTER_TRIG'

5.1.2.14 การเพิ่มความสัมพันธ์ระหว่างงานกับบทบาท

กำหนดตัวอย่างดังตารางที่ 5.10 ดังต่อไปนี้

ตารางที่ 5.10 แสดงตัวอย่างการกำหนดความสัมพันธ์ระหว่างงานกับบทบาท

งาน	บทบาท
Closing End of Day Account	Chief Post Office
Summarize Financial	Accountant
Check Stock	

งานสัมพันธ์กับบทบาทถูกแทนด้วยรูปแบบทางคณิตศาสตร์คือ $(j_i, r_i) \in RJ$ โดยมีคำสั่งในการเพิ่มข้อมูลดังนี้

คำสั่งการกำหนดให้บทบาท "Chief Post Office" มีงานที่รับผิดชอบคือ "Closing End of Day Account"

```
INSERT INTO RL VALUES(1, 1);
```

คำสั่งการกำหนดให้บทบาท "Accountant" มีงานที่รับผิดชอบคือ "Summarize Financial" กับ "Check Stock"

```
INSERT INTO RL VALUES(2, 2);
INSERT INTO RL VALUES(4, 2);
```

การเพิ่มข้อมูลนี้จะมีทริกเกอร์คอยตรวจสอบความถูกต้องชื่อว่า "rjstatafter_trig"โดยทริกเกอร์นี้จะแสดงข้อความผิดพลาดขึ้นถ้าเกิดเหตุการณ์ที่งานที่ขัดแย้งถูกกำหนดให้กับบทบาทที่ไม่ขัดแย้งกัน ซึ่งไม่สามารถกระทำได้ โดยข้อความผิดพลาดที่แสดงคือ

ORA-20204: Cannot insert job to role association - a conflicting job must be assigned to a conflicting role.

ORA-06512: at "ERBAC03.RJSTATAFTER_TRIG", line 48

ORA-04088: error during execution of trigger 'ERBAC03.RJSTATAFTER_TRIG'

5.1.2.15 การเพิ่มความสัมพันธ์ระหว่างงานกับงานย่อย

กำหนดตัวอย่างดังตารางที่ 5.11 ดังต่อไปนี้

ตารางที่ 5.11 แสดงตัวอย่างการกำหนดความสัมพันธ์ระหว่างงานกับงานย่อย

งานย่อย	งาน
Counting the Money	Closing End of Day Account
Check the Old Account	Summarize Financial
Record the Revenue and Expenditure	
Calculate Summary Account	

งานย่อยสัมพันธ์กับงานถูกแทนด้วยรูปแบบทางคณิตศาสตร์คือ $(t, j_i) \in JT$ โดยมีคำสั่งในการเพิ่มข้อมูลดังนี้

คำสั่งการกำหนดให้งาน "Closing End of Day Account" มีรายละเอียดการทำงานคือ "Counting the Money"

```
INSERT INTO JT VALUES(1, 1);
```

คำสั่งการกำหนดให้งาน "Summarize Financial" มีรายละเอียดการทำงานคือ "Check the Old Account" กับ "Record the Revenue and Expenditure" และ "Calculate Summary Account"

```
INSERT INTO JT VALUES(2, 2);
INSERT INTO JT VALUES(3, 2);
INSERT INTO JT VALUES(4, 2);
```

การเพิ่มข้อมูลนี้จะมีทริกเกอร์คอยตรวจสอบความถูกต้องชื่อว่า "jtstatafter_trig"โดยทริกเกอร์นี้จะแสดงข้อความผิดพลาดขึ้นถ้าเกิดเหตุการณ์ที่งานย่อยที่ขัดแย้งถูกกำหนดให้กับงานที่ไม่ขัดแย้งกัน ซึ่งไม่สามารถกระทำได้ โดยข้อความผิดพลาดที่แสดงคือ

ORA-20205: Cannot insert task to job association - a conflicting task must be assigned to a conflicting job.

ORA-06512: at "ERBAC03.JTSTATAFTER_TRIG", line 48

ORA-04088: error during execution of trigger 'ERBAC03.JTSTATAFTER_TRIG'

5.1.2.16 การเพิ่มความสัมพันธ์ระหว่างงานย่อยกับใบอนุญาต

กำหนดตัวอย่างดังตารางที่ 5.12 ดังต่อไปนี้

ตารางที่ 5.12 แสดงตัวอย่างการกำหนดความสัมพันธ์ระหว่างงานย่อยกับใบอนุญาต

ใบอนุญาต	งานย่อย
Read the Transaction Record	Counting the Money
Read Account Record	Record the Revenue and Expenditure
Write Account Record	

ใบอนุญาตสัมพันธ์กับงานย่อยถูกแทนด้วยรูปแบบทางคณิตศาสตร์คือ $(p, t_j) \in TP$ โดยมีคำสั่งในการเพิ่มข้อมูลดังนี้

คำสั่งการกำหนดให้งานย่อย "Counting the Money" มีใบอนุญาตคือ "Read the Transaction Record"

INSERT INTO TP VALUES(4, 1);

คำสั่งการกำหนดให้งานย่อย "Record the Revenue and Expenditure" มีใบอนุญาตคือ "Read Account Record" กับ "Write Account Record"

INSERT INTO TP VALUES(1, 3);

INSERT INTO TP VALUES(2, 3);

การเพิ่มข้อมูลนี้จะมีทริกเกอร์คอยตรวจสอบความถูกต้องชื่อว่า "tpstatafter_trig"โดยทริกเกอร์นี้จะแสดงข้อความผิดพลาดขึ้นถ้าเกิดเหตุการณ์ที่ใบอนุญาตที่ขัดแย้งถูกกำหนดให้กับงานย่อยที่ไม่ขัดแย้งกัน ซึ่งไม่สามารถกระทำได้ โดยข้อความผิดพลาดที่แสดงคือ

ORA-20206: Cannot insert permission to task association - a conflicting permission must be assigned to a conflicting task.

ORA-06512: at "ERBAC03.TPSTATAFTER_TRIG", line 48

ORA-04088: error during execution of trigger 'ERBAC03.TPSTATAFTER_TRIG'

5.1.2.17 การลบความสัมพันธ์

การลบความสัมพันธ์สามารถกระทำได้โดยไม่มีผลกระทบต่อความคงสภาพของข้อมูล สำหรับรูปแบบ ERBAC03 ได้สาธิตการใช้งานโปรแกรมแบบทริกเกอร์ในฐานข้อมูลที่มีประสิทธิภาพในการควบคุมการเข้าถึงระบบฯ ให้เป็นไปอย่างถูกต้องเหมาะสม

เนื่องจากการกำหนดข้อบังคับแบบใช้คำสั่ง SQL ไม่สะดวกต่อผู้บริหารจัดการระบบมากนัก งานวิจัยฉบับนี้จึงได้ออกแบบโปรแกรมที่แสดงผลแบบกราฟิก เพื่อให้ง่ายต่อการกำหนดข้อบังคับ โดยจะกล่าวในหัวข้อถัดไป

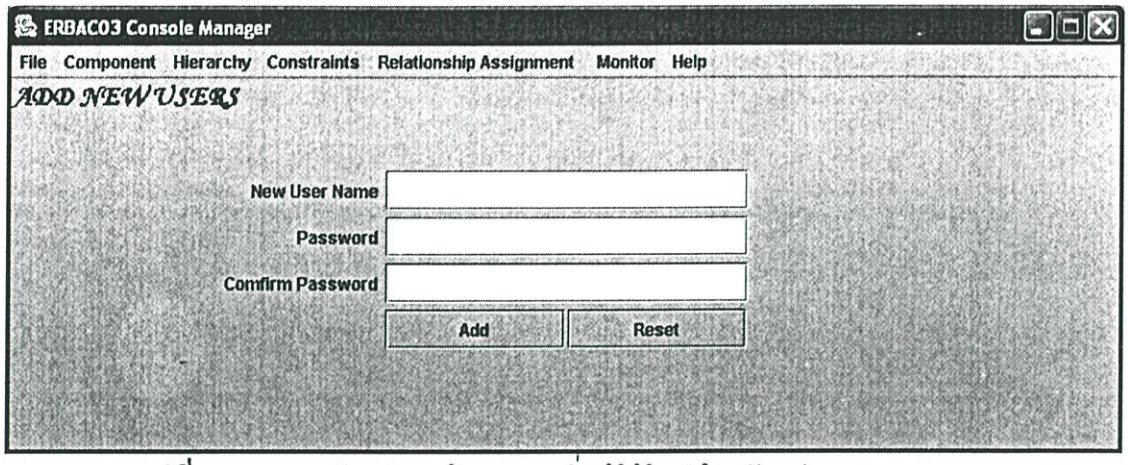
5.2 การพัฒนาโปรแกรมแบบกราฟิกสำหรับการบริหารจัดการระบบฯ

การพัฒนาโปรแกรมประเภทนี้ใช้ภาษาจาวาในการพัฒนา ซึ่งจาวามีจุดเด่นคือเขียนเพียงครั้งเดียวแต่สามารถนำไปใช้งานได้ทุกแพลทฟอร์ม (Write Once, Run Anywhere) ซึ่งเครื่องของผู้ใช้ต้องมีโปรแกรมสำหรับชุดคำสั่งที่ใช้สร้างแอพพลิเคชั่น (Java virtual machine) โดยการเขียนภาษาจาวานี้มีการออกแบบส่วนติดต่อผู้ใช้ (User interface) ซึ่งใช้ชุดอินเตอร์เฟสที่เรียกว่า จาวาสวิงซ์ (Java swing) เป็นชุดอินเตอร์เฟสที่มาช่วยเสริมการสร้างส่วนติดต่อผู้ใช้ให้มีประสิทธิภาพยิ่งขึ้น โดยลักษณะของสวิงซ์จะแตกต่างจากอินเตอร์เฟสเดิมที่ใช้คือ AWT ที่มีข้อจำกัดที่ให้ลักษณะแตกต่างกันเมื่อนำไปใช้งานบนระบบปฏิบัติการที่ต่างกัน แต่สวิงซ์มาลดจุดบงพร่องเหล่านี้ให้มีหน้าตาอินเตอร์เฟสที่เหมือนกัน ไม่ว่าจะนำโปรแกรมไปใช้ในระบบปฏิบัติการใดๆ ก็ตาม

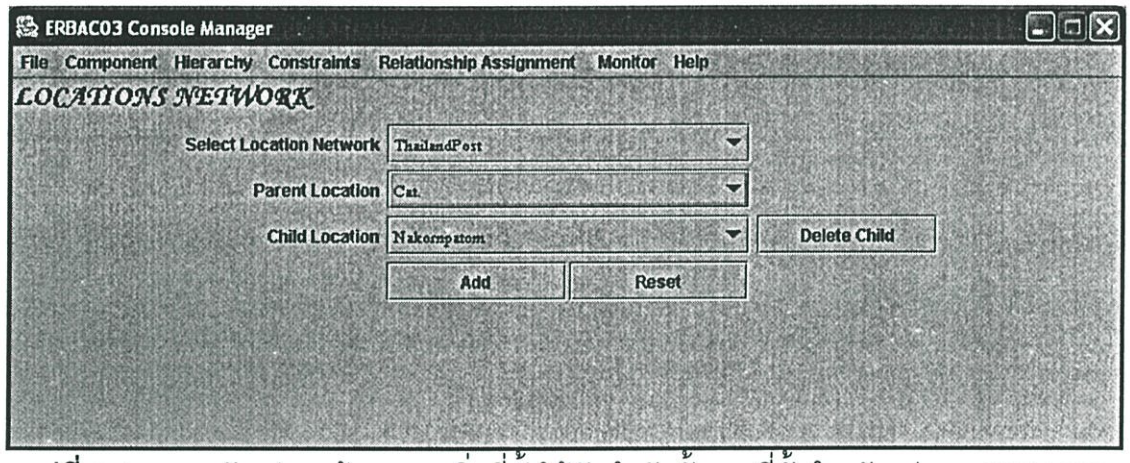
ส่วนจุดเด่นในการออกแบบอีกประการคือ การติดต่อฐานข้อมูลออราเคิลผ่าน JDBC ซึ่งสามารถใช้งานร่วมกับทริกเกอร์ที่สร้างขึ้นก่อนหน้านี้แล้วได้ทันที ทำให้โปรแกรมแบบกราฟิกนี้เปรียบเสมือนเพียงแค่หน้ากากเท่านั้น โดยจะมีทริกเกอร์ในฐานข้อมูลคอยตรวจสอบความถูกต้องในการบริหารจัดการระบบฯของเจ้าหน้าที่ ซึ่งเมื่อทริกเกอร์ตรวจจับความผิดพลาดได้ก็จะส่งข้อความแจ้งขึ้นมายังหน้าจอของผู้ใช้ทันที ซึ่งโปรแกรมไม่ต้องมีการกำหนดคุณสมบัติของข้อบังคับใดๆ เลย

รูปที่ 5.2 แสดงหน้าจอหลักของโปรแกรมบริหารจัดการสำหรับรูปแบบ ERBAC03

จากรูป 5.2 แสดงให้เห็นถึงฟังก์ชั่นการทำงานของโปรแกรมทั้งหมด ซึ่งมีครบทุกอัลกอริทึมที่ออกแบบในบทที่ 4 โดยการทำงานจะเริ่มตั้งแต่การกำหนดองค์ประกอบของรูปแบบ ERBAC03 ทั้งหมดดังรูปที่ 5.3

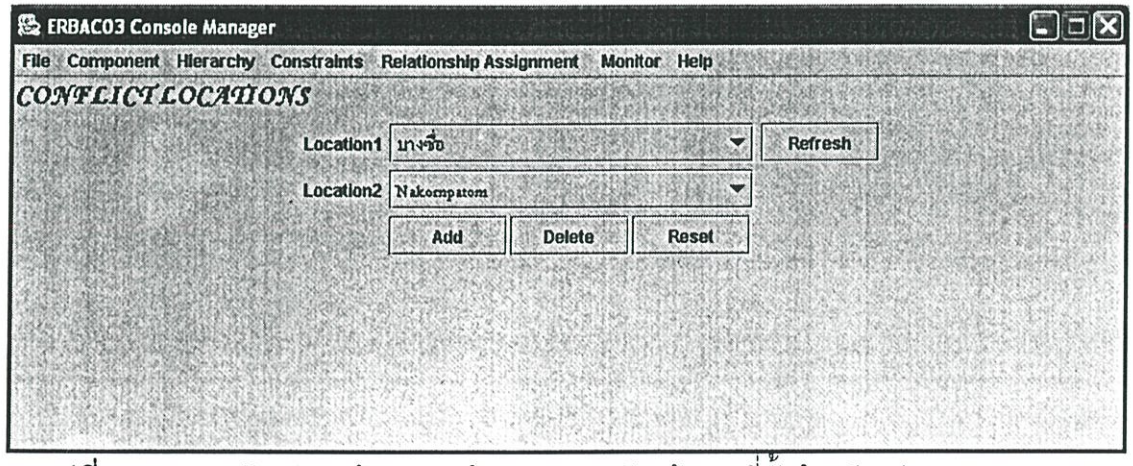


รูปที่ 5.3 แสดงตัวอย่างหน้าจอการเพิ่มผู้ใช้ใหม่สำหรับรูปแบบ ERBAC03

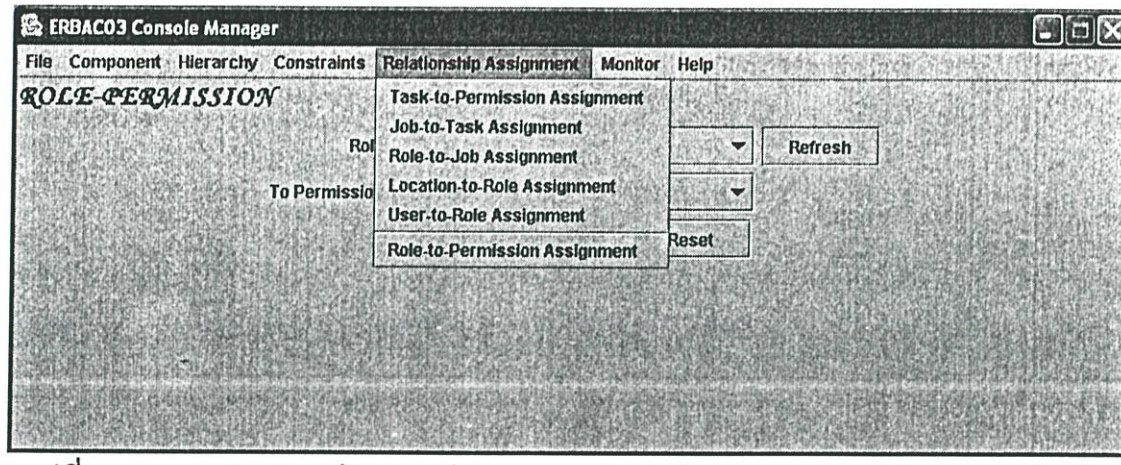


รูปที่ 5.4 แสดงตัวอย่างหน้าจอการเพิ่มที่ตั้งให้กับลำดับชั้นของที่ตั้งสำหรับรูปแบบ ERBAC03

จากรูปที่ 5.4 สามารถเพิ่มและลบที่ตั้งที่เป็นส่วนหนึ่งของลำดับชั้นของที่ตั้งได้ทันที ซึ่งถ้ามีข้อความผิดพลาดก็จะแจ้งเตือนให้ผู้ใช้ได้รับทราบทันที ส่วนตัวอย่างการกำหนดความขัดแย้งและการกำหนดความสัมพันธ์ ได้แสดงดังรูปที่ 5.5 และ 5.6 ตามลำดับ

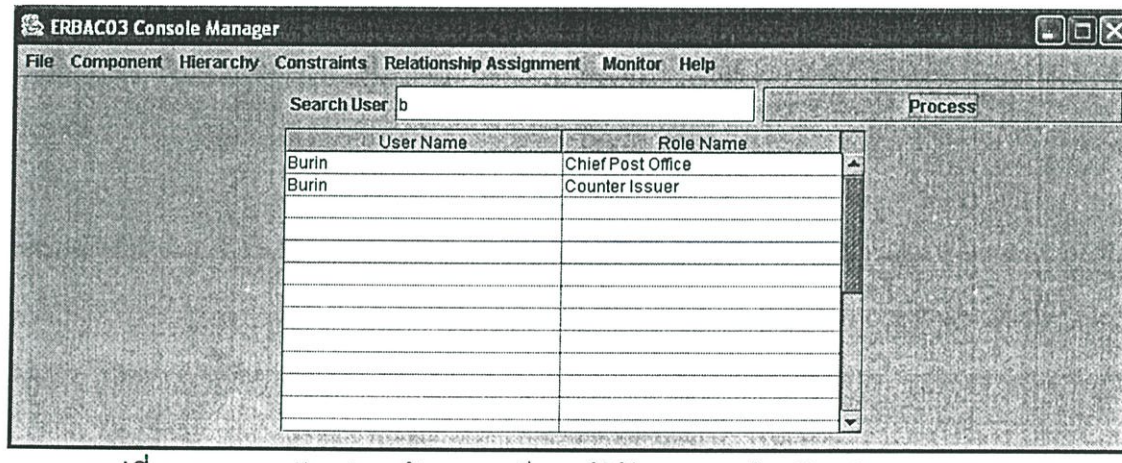


รูปที่ 5.5 แสดงตัวอย่างหน้าจอการกำหนดความขัดแย้งของที่ตั้งสำหรับรูปแบบ ERBAC03



รูปที่ 5.6 แสดงตัวอย่างหน้าจอการกำหนดความสัมพันธ์ระหว่างบทบาทกับใบอนุญาตสำหรับรูปแบบ RBAC

ส่วนสุดท้ายเป็นการแสดงผู้ใช้ที่มีอยู่ในระบบทั้งหมด โดยผู้บริหารจัดการระบบฯสามารถรู้ได้ว่ามีผู้ใช้คนใดบ้างที่อยู่ในระบบ และแต่ละคนได้รับบทบาทอะไรบ้าง ซึ่งแสดงดังรูปที่ 5.7



รูปที่ 5.7 แสดงตัวอย่างหน้าจอการเรียกดูผู้ใช้ในระบบฯสำหรับรูปแบบ ERBAC03

การออกแบบทริกเกอร์และโปรแกรมบริหารจัดการระบบฯแบบกราฟิกนี้ได้พัฒนาตามนิยาม ทฤษฎีและสัจพจน์ทั้งหมดที่ได้กำหนดไว้ก่อนหน้านี้ในบทที่ 3 ซึ่งโปรแกรมทั้งสองประเภทนี้นำมาใช้ร่วมกันได้อย่างมีประสิทธิภาพในด้านของการกำหนดความขัดแย้งให้กับเอนทิที ทำให้ระบบฯ รักษาความคงสภาพของข้อมูลอยู่ได้ เนื่องจากโปรแกรมจะมีการกรองข้อมูลก่อนที่จะมีการกำหนดบทบาทให้กับผู้ใช้ทุกครั้ง ข้อมูลจึงมีความถูกต้องเสมอ

บทที่ 6
การทดลองและผลการทดลอง

จากที่ได้อธิบายถึงนิยาม ทฤษฎีและอัลกอริทึมทั้งหมดในบทก่อนหน้านี้ โดยนำมาพัฒนาเป็นโปรแกรมเพื่อทดลองใช้งานตามสภาพแวดล้อมที่ได้กำหนดไว้ และนำผลการทดลองที่ได้มาวิเคราะห์ว่าเมื่อนำรูปแบบ ERBAC03 มาใช้งานแล้วทำให้เกิดผลดีผลเสียอย่างไรบ้าง

6.1 การออกแบบการทดลอง

ในการทดลอง ERBAC03 นั้นได้ทำการกำหนดสภาพแวดล้อมของระบบฯ คือ ได้ทดลองกับระบบควบคุมส่วนกลาง (Event Management Server) ซึ่งเป็นระบบฐานข้อมูลของบริษัท ไปรษณีย์ไทย จำกัด โดยระบบควบคุมส่วนกลางนี้ถูกออกแบบมาเพื่อรองรับข้อมูลที่ทำการไปรษณีย์ทั่วประเทศที่มีการทำรายการทรานแซคชั่นตลอดวัน ซึ่งระบบจะถูกเข้าถึงโดยผู้ใช้ที่อยู่ส่วนกลางที่สังกัด

- ส่วนการปฏิบัติการคอมพิวเตอร์และเครือข่าย
- ส่วนประมวลผลข้อมูล
- ส่วนสถิติและข้อมูลสารสนเทศ

ส่วนการปฏิบัติการคอมพิวเตอร์และเครือข่ายมีบทบาทที่ใช้สำหรับติดต่อระบบควบคุมส่วนกลางคือ

- "ROAPRD" เป็นบทบาทประเภทผู้บริหารจัดการระบบฯ ที่แผนกฐานข้อมูลมีสิทธิ์แต่เพียงผู้เดียวในการใช้ปฏิบัติงาน
- "ROSSRPT" เป็นบทบาทประเภทผู้ใช้ทั่วไป ที่แผนกเครือข่ายมีสิทธิ์ใช้เท่านั้น โดยบทบาทนี้มีหน้าที่หลักเพียงแค่เรียกดูการปิดบัญชีสิ้นวันของแต่ละที่ทำการไปรษณีย์เท่านั้น

ส่วนประมวลผลข้อมูลมีบทบาทที่ใช้สำหรับติดต่อระบบควบคุมส่วนกลางคือ

- "MOUSER" เป็นบทบาทประเภทผู้ใช้ทั่วไป ที่แผนกตัวแทนรับชำระและธนาณัติมีสิทธิ์ใช้เท่านั้น โดยบทบาทนี้มีหน้าที่หลักเพียงแค่เรียกดูการรับจ่ายเงินและธนาณัติของแต่ละที่ทำการไปรษณีย์เท่านั้น

ส่วนสถิติและข้อมูลสารสนเทศมีบทบาทที่ใช้สำหรับติดต่อระบบควบคุมส่วนกลางคือ

- "MISUSER" เป็นบทบาทประเภทผู้ใช้ทั่วไป ที่แผนกสถิติมีสิทธิ์ใช้เท่านั้น โดยบทบาทนี้มีหน้าที่หลักเพียงแค่เรียกดูข้อมูลทางการเงินและข้อมูลคลังของแต่ละที่ทำการไปรษณีย์เท่านั้น

- "GLINT" เป็นบทบาทประเภทผู้ใช้ทั่วไป ที่แผนกบัญชีมีสิทธิ์ใช้เท่านั้น โดยบทบาทนี้มีหน้าที่หลักเพียงแค่เรียกดูข้อมูลทางการเงินทั้งหมดของแต่ละที่ทำการไปรษณีย์เท่านั้น

สภาพแวดล้อมทั่วไปที่ใช้ในการทดลองทั้ง 3 การทดลองประกอบด้วย

- เครื่องคอมพิวเตอร์แม่ข่าย 1 เครื่อง มี 4 CPU ความเร็วตัวละ 500 MHz. RAM 4 GB. Hard Disk 200GB. รองรับการเชื่อมต่อ 50 ครั้ง
- ระบบควบคุมส่วนกลาง ไม่มีไฟล์วอลล์ป้องกันการเข้าถึงในแต่ละส่วนงาน
- ระบบเครือข่ายมีความเร็ว 10 Mbps.
- ฐานข้อมูลที่ใช้คือ ออราเคิล ซึ่งใช้รูปแบบการควบคุมการเข้าถึงแบบโรลเบสอยู่แล้ว

บทบาทที่สำคัญที่สุดของระบบควบคุมส่วนกลางนี้คือ "ROAPRD" ส่วนฐานข้อมูลที่ใช้ในการทดลองครั้งนี้คือออราเคิล ซึ่งเป็นฐานข้อมูลที่ใช้วิธีการควบคุมการเข้าถึงแบบโรลเบส อีกทั้งยังสนับสนุนการเขียนโปรแกรมประเภททริกเกอร์ด้วย

จากการทดลองครั้งนี้ได้มีการตั้งสมมุติฐาน ตัววัด และปัจจัยสำคัญในแต่ละเหตุการณ์ดังนี้

ตารางที่ 6.1 แสดงข้อสมมุติฐานทั้งหมดที่จะนำมาทดลอง

การตั้งสมมติฐาน	**ตัววัด** (Metric)	**ปัจจัย** (Factor)
ERBAC03 ลดความถี่ในการบุกรุกและเพิ่มความปลอดภัยแก่ระบบ	ขจัดการบุกรุกเพิ่มความปลอดภัยให้แก่ระบบ	เปอร์เซ็นต์การบุกรุกเข้ามาในระบบสามารถถูกขจัดโดย ERBAC03
ERBAC03 เพิ่มความถูกต้องของข้อมูลโดยใช้ข้อบังคับ	ข้อมูลในระบบฯ ถูกต้อง	เปอร์เซ็นต์ความผิดพลาดของข้อมูลถูกขจัดโดย ERBAC03

การกำหนดสมมุติฐานที่สองเป็นประเด็นหลักของงานวิจัยฉบับนี้ว่ามีการกำหนดข้อมูลให้ถูกต้องเพียงใด เพื่อวัตถุประสงค์ทางด้านความปลอดภัยของระบบฯ ซึ่งระบบฯที่ถูกละเมิดความปลอดภัยทำให้องค์กรมีความสูญเสียทางเศรษฐกิจมหาศาล โดยจากเอกสารการวิจัย [46] พบว่าการละเมิดความปลอดภัยส่วนใหญ่เกิดจากความสะเพร่าของพนักงานในองค์กรเอง ซึ่งมีถึง 58% ตามด้วย บุคคลภายนอกที่ไม่มีสิทธิ์ 42% และอีก 14% จากพนักงานที่ออกจากองค์กรไปแล้วแต่ยังมีสิทธิ์ใช้งานระบบอยู่ โดยที่การละเมิดส่วนใหญ่จะเกี่ยวข้องกับการทุจริตทางด้านการเงินเป็นอันดับแรก ตามด้วยการไม่มีสิทธิ์เข้าถึงข้อมูลจากภายในองค์กรเอง

งานวิจัยฉบับนี้เปรียบเทียบรูปแบบ RBAC และ ERBAC ในระดับ 0 เท่านั้นสำหรับสมมุติฐานแรก ส่วนสมมุติฐานที่สองจะเปรียบเทียบรูปแบบทั้งสองในระดับที่ 1 โดยเน้นถึงการป้องกันการกำหนดสิทธิ์อย่างไม่เหมาะสมของผู้บริหารจัดการระบบฯ ด้านความปลอดภัยเป็นหลัก ซึ่งได้มีการ

กำหนดหลักการและวิธีการทั้งหมดจากบทที่ 3 บทที่4 และ บทที่5 ตามลำดับ โดยได้นำแนวคิดทฤษฎีทั้งหมดที่ออกแบบมาทดสอบใช้ดังวิธีการต่อไปนี้

6.1.1 วิธีการทดลองสมมุติฐานการลดความถี่ในการบุกรุก

สมมุติฐานแรกเกี่ยวข้องกับความถูกต้องของข้อมูล (Data accuracy) ในการกำหนดบทบาทให้กับผู้ใช้อย่างถูกต้อง จึงต้องมีการเปรียบเทียบค่าความถูกต้องของข้อมูลระหว่างระบบที่ใช้รูปแบบ RBAC กับ ERBAC03 ซึ่งการทดสอบนี้จะกระทำในสภาพแวดล้อมที่กำหนดดังต่อไปนี้

- เครื่องคอมพิวเตอร์ลูกข่าย 65 เครื่อง แต่ละเครื่องประกอบด้วย 1 CPU ความเร็ว 450 MHz. RAM 128 MB. Hard Disk 8 GB.
- เครื่องคอมพิวเตอร์ลูกข่ายที่มีสิทธิ์เข้าถึงฐานข้อมูลส่วนกลางในฐานะบทบาท "ROAPRD" มี 5 เครื่อง แต่ละเครื่องประกอบด้วย 1 CPU ความเร็ว 450 MHz. RAM 128 MB. Hard Disk 8 GB. ชื่อเครื่องห้าเครื่องที่ประกอบไปด้วย WRKDBA_01 WRKDBA_02 WRKDBA_03 WRKDBA_04 และ WRKDBA_05
- บุคคลที่มีสิทธิ์ใช้บทบาท "ROAPRD" ประกอบด้วย 5 คน คือ burin, administrator, anan, sys และ system

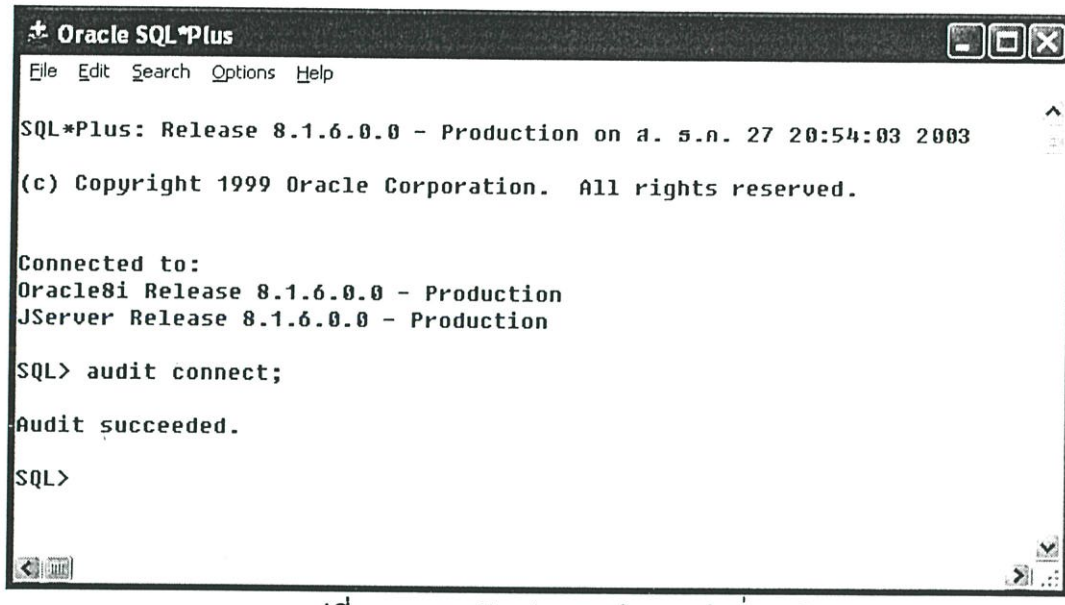
โดยเงื่อนไขการทดลองมีดังนี้

- ผู้ใช้สามารถเชื่อมต่อกับฐานข้อมูลระบบฯ ได้หลายการเชื่อมต่ออย่างไม่จำกัด
- ทั้งสามส่วนงานมีสิทธิ์ใช้ฐานข้อมูลระบบฯ ได้
- เวลาที่ใช้ในการเก็บข้อมูลต่อวัน เริ่มตั้งแต่เวลา 8.30-16.30 น.
- การทดลองการป้องกันการบุกรุกจะกระทำตั้งแต่เวลา 8.30-16.30 น.
- บทบาทที่ใช้ในการทดลองคือ "ROAPRD"

ซึ่งสามารถเก็บข้อมูลตัวอย่างเพื่อใช้วัดความถูกต้องของทั้งสองรูปแบบได้โดยการใช้ฟังก์ชั่นของฐานข้อมูลออราเคิลนั้นคือ การออดิท ซึ่งเป็นวิธีการที่ใช้สำหรับตรวจสอบบัญชีรายชื่อของผู้ใช้ฐานข้อมูลออราเคิล โดยมีขั้นตอนการเซตค่าดังนี้

ขั้นตอนที่ 1 แก้ไขพารามิเตอร์ไฟล์ของฐานข้อมูลระบบควบคุมส่วนกลาง โดยกำหนด audit_trail = true เพื่อเป็นการบอกให้ฐานข้อมูลเก็บสถิติการใช้งานของผู้ใช้ได้

ขั้นตอนที่ 2 กำหนดให้มีการเก็บสถิติการล็อกอินและล็อกเอาน์ของผู้ใช้ให้กับทั้งสองรูปแบบโดยใช้คำสั่งในโปรแกรมเอสคิวแอลพลัสดังรูปที่ 6.1



รูปที่ 6.1 แสดงตัวอย่างการกำหนดคำสั่งออดิท

ขั้นตอนที่ 3 ตรวจสอบข้อมูลที่เก็บได้โดยดูจากตาราง AUD$ ที่มีผู้ใช้ "SYS" เป็นเจ้าของอยู่ ซึ่งได้ตัวอย่างข้อมูลดังตารางที่ 6.2

ตารางที่ 6.2 แสดงตัวอย่างข้อมูลการล็อกอินและล็อกเอาน์ของผู้ใช้ในระบบควบคุมส่วนกลางโดยใช้รูปแบบ RBAC

TIMESTAMP#	USERID	ROLEID	TERMINAL	LOGOFF$TIME
19/3/2004 10:26	burin	ROAPRD	WRKDBA_01	19/3/2004 10:26
19/3/2004 10:31	SYSTEM	SYS	SVRCAGC_02	19/3/2004 10:31
19/3/2004 10:33	dtpadmin	ROAPRD	SVRCCPS_03	19/3/2004 10:33
19/3/2004 10:33	dtpadmin	ROAPRD	SVRCCPS_01	19/3/2004 10:33
19/3/2004 10:35	Administrator	MOUSR	WRKCDES_03	23/3/2004 8:58
19/3/2004 10:35	Administrator	MOUSR	WRKCDES_03	23/3/2004 8:58
19/3/2004 10:35	SYSTEM	ROSSRPT	WRKCSMS_02	19/3/2004 10:35
19/3/2004 10:35	Zintoo	ROAPRD	ZINTOOXP	19/3/2004 10:36
19/3/2004 10:38	anan	ROAPRD	WRKDBA_02	19/3/2004 10:39

จากการทดลองในครั้งนี้ข้อมูลที่พิจารณาเป็นพิเศษคือ ข้อมูลที่เกี่ยวข้องกับบทบาท "ROAPRD" ทั้งหมด โดยมีการกำหนดให้ผู้ใช้ที่ปฏิบัติงานอยู่ในส่วนการปฏิบัติการคอมพิวเตอร์และเครือข่ายใช้ได้เท่านั้น

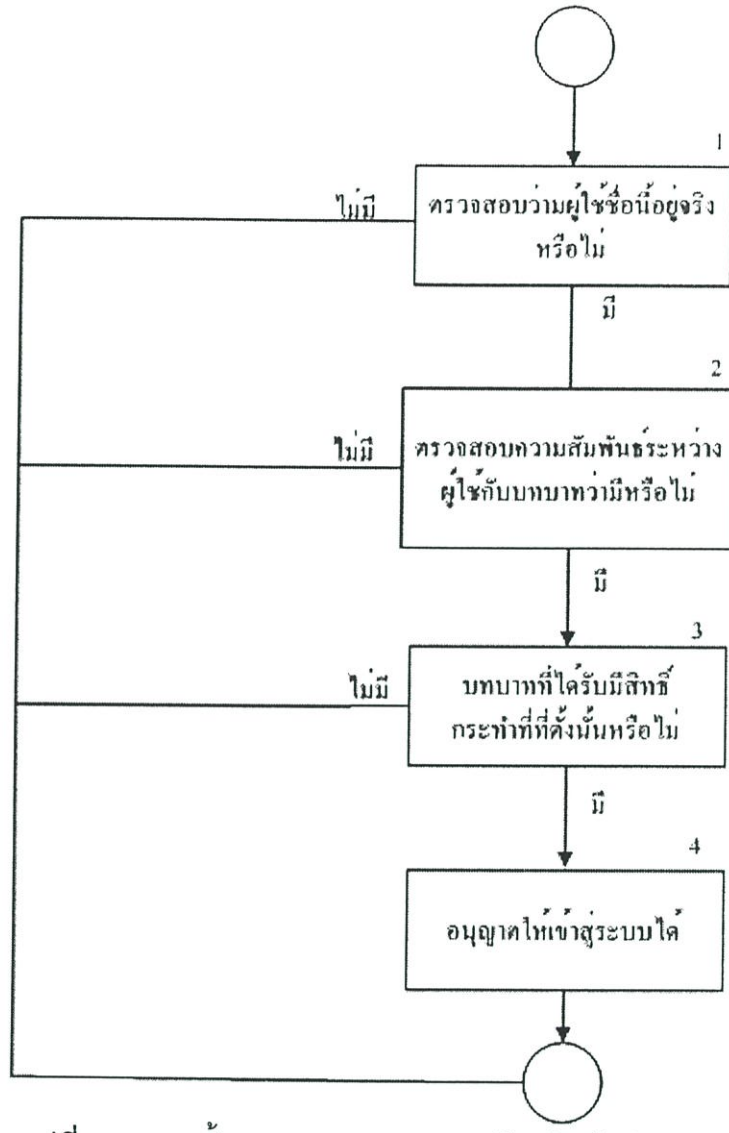
ซึ่งจากตัวอย่างข้อมูลนี้จะสังเกตได้ว่ามีการใช้บทบาทในสถานที่ที่ไม่เหมาะสม ซึ่งสามารถหาอัตราส่วนความถูกต้องของข้อมูลได้จากสมการ 1

$$\text{ร้อยละความถูกต้องของข้อมูล} = \frac{\text{จำนวนครั้งของข้อมูลที่ถูกต้อง}}{\text{จำนวนครั้งของข้อมูลทั้งหมด}} \times 100 \qquad (1)$$

เนื่องจากฐานข้อมูลออราเคิลได้มีการใช้รูปแบบ RBAC อยู่แล้ว ข้อมูลที่เก็บมาทั้งหมดจึงสามารถนำมาเปรียบเทียบกับรูปแบบ ERBAC03 ได้ทันที จากนั้นให้ทดลองด้วยวิธีการของรูปแบบ ERBAC03 โดยสมมุติว่าได้เพิ่มข้อมูลชื่อเครื่องคอมพิวเตอร์ลงในตาราง Locations เรียบร้อยแล้วตั้งแต่รหัสหนึ่งถึงห้า และกำหนดให้พนักงานแผนกฐานข้อมูลมีสิทธิ์ใช้บทบาท "ROAPRD" ด้วย สุดท้ายให้ใช้คำสั่งเอสคิวแอลเพื่อกำหนดว่าบทบาท "ROAPRD" มีสิทธิ์ใช้ที่ใดบ้าง ดังนี้

```
INSERT INTO RL VALUES(1, 1);
INSERT INTO RL VALUES(2, 1);
INSERT INTO RL VALUES(3, 1);
INSERT INTO RL VALUES(4, 1);
INSERT INTO RL VALUES(5, 1);
```

เมื่อได้กำหนดข้อมูลครบแล้วให้ตรวจสอบข้อมูลหลังจากการล็อกอินของผู้ใช้ว่าตรงกับข้อมูลที่กำหนดในตารางของรูปแบบ ERBAC03 หรือไม่ ดังรูปที่ 6.2



รูปที่ 6.2 แสดงขั้นตอนการตรวจสอบการล็อกอินเข้าสู่ระบบฯ

6.1.2 วิธีการทดลองสมมุติฐานการเพิ่มความถูกต้องของข้อมูล

เนื่องจากรูปแบบ RBAC ได้มีการกำหนดทฤษฎีข้อบังคับไว้แต่ไม่ได้ระบุถึงวิธีการนำไปปฏิบัติ จึงมีนักวิจัยบางท่าน [39] ได้นำไปปฏิบัติ และสามารถนำรูปแบบ ERBAC03 มาเปรียบเทียบกับงานวิจัยชิ้นนั้นได้ ซึ่งมีสภาพแวดล้อมในการทดลองดังนี้

- เครื่องคอมพิวเตอร์ 1 เครื่องประกอบด้วย 1 CPU ความเร็ว 450 MHz. RAM 128 MB. Hard Disk 8 GB. ชื่อเครื่อง WRKDBA_01
- ผู้ใช้ที่ทำการทดลองมีเพียง 1 คน

อีกทั้งได้มีการกำหนดเงื่อนไขในการทดลองคือ

- เวลาในการทดลองทำเมื่อใดก็ได้

- ใช้วิธีการของ Perelson มาเปรียบเทียบกับรูปแบบ ERBAC03 เพราะ Perelson ได้ทดลองรูปแบบ RBAC ไว้อยู่แล้ว

จากนั้นให้ทำการทำการทดลองกับทั้งสองรูปแบบดังต่อไปนี้

สมมุติให้บทบาท "ROAPRD" กับบทบาท "GLINT" ขัดแย้งกัน โดยพิจารณาขั้นตอนดังนี้

ขั้นตอนที่ 1 ถ้ายังไม่มีบทบาททั้งสองอยู่ในระบบฯ ให้พิมพ์คำสั่งเอสคิวแอลดังนี้

```
INSERT INTO ROLES VALUES(1, 'ROAPRD');
INSERT INTO ROLES VALUES(2, 'GLINT');
```

ขั้นตอนที่ 2 กำหนดบทบาทให้ขัดแย้งกันโดยพิมพ์คำสั่งเอสคิวแอลดังนี้

```
INSERT INTO CR VALUES(1, 2);
```

ขั้นตอนที่ 3 กำหนดบทบาททั้งสองให้กับผู้ใช้ "Burin" โดยพิมพ์คำสั่งเอสคิวแอลดังนี้

```
INSERT INTO UA VALUES(1, 1);
INSERT INTO UA VALUES(2, 1);
```

6.1.3 วิธีการทดลองการหาเวลาการทำงาน

การทดลองนี้มีวัตถุประสงค์เพื่อต้องการทราบเวลาที่สูญเสียไปจากการกำหนดรายละเอียดที่เพิ่มมากขึ้นกว่ารูปแบบ RBAC ว่าการปฏิบัติในแต่ละกิจกรรมมีความแตกต่างกันมากเพียงใด และมีความเหมาะสมหรือไม่ ถ้าแลกกับการเก็บรายละเอียดข้อมูลในการกำหนดหน้าที่ความรับผิดชอบทั้งหมดเพื่อใช้ตรวจสอบความถูกต้องในของการกำหนดนั้นๆ อีกทั้งยังได้บ่งบอกถึงความชัดเจนของอำนาจหน้าที่ในแต่ละบทบาทด้วย

เนื่องจากวิธีการทดลองนี้เกี่ยวข้องกับเวลา เพราะฉะนั้นการทดลองนี้จะกระทำโดยการจับเวลาการปฏิบัติงานของทั้งสี่กิจกรรมดังนี้

- การจัดการการมอบหมายสิทธิ์ที่มีอยู่เดิมให้แก่ผู้ใช้ใหม่
- การเปลี่ยนสิทธิ์ของผู้ใช้ที่มีอยู่เดิม
- การสร้างสิทธิ์ใหม่แก่ผู้ใช้ที่มีอยู่เดิม
- การยกเลิกสิทธิ์

สภาพแวดล้อมในการทดลองมีดังนี้

- เครื่องคอมพิวเตอร์ 1 เครื่องประกอบด้วย 1 CPU ความเร็ว 450 MHz. RAM 128 MB. Hard Disk 8 GB.
- ผู้บริหารจัดการระบบฯ 1 คน
- ข้อมูลผู้ใช้ที่นำมาทำการกำหนดสิทธิ์มีจำนวนทั้งสิ้น 10 คน

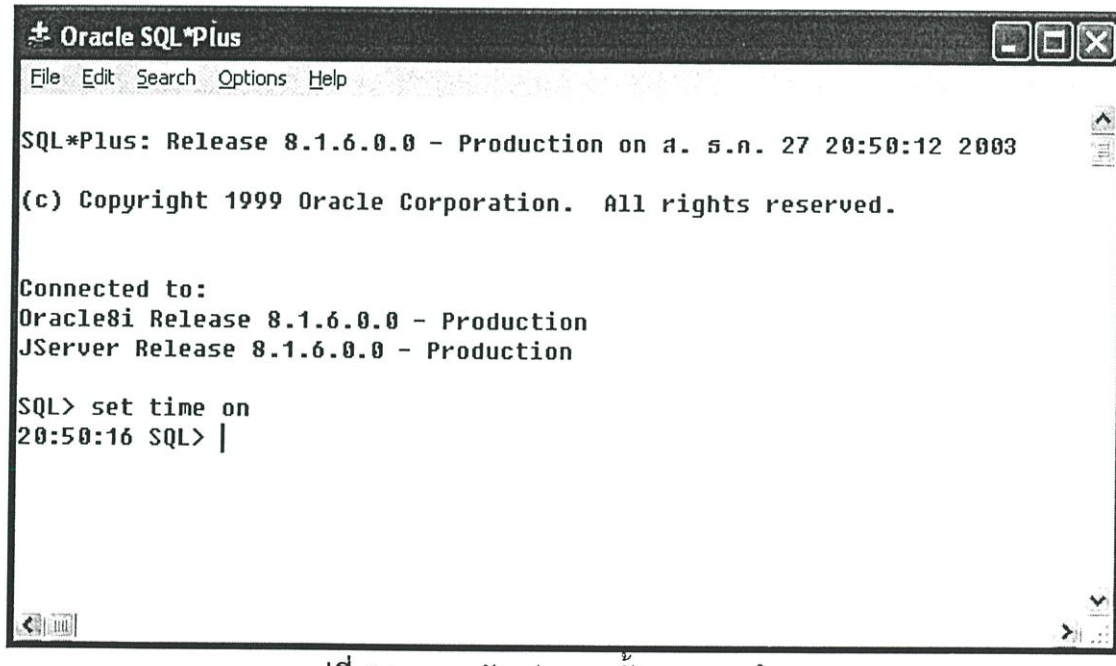
เงื่อนไขในการทดลองประกอบด้วย

- การจับเวลาจะกระทำเฉพาะการพิมพ์คำสั่งเอสคิวแอลเท่านั้น
- ผู้บริหารจัดการระบบฯ ทำการพิมพ์คำสั่งเอสคิวแอลเป็นลำดับจนเสร็จต่อหนึ่งคนซ้ำไปเรื่อยๆ จนครบทั้ง 10 คน
- เริ่มจับเวลาตั้งแต่พิมพ์คำสั่งเอสคิวแอลคำสั่งแรกและสิ้นสุดการจับเวลาเมื่อเคาะปุ่ม enter ในคำสั่งสุดท้าย โดยการพิมพ์คำสั่งในที่นี้ขึ้นอยู่กับการทดลองตามกิจกรรมทั้งสี่ เช่น กิจกรรมแรก เริ่มจับเวลาตั้งแต่การพิมพ์คำสั่งเพิ่มผู้ใช้ใหม่และสิ้นสุดการจับเวลาเมื่อมีการกำหนดบทบาทให้กับผู้ใช้เรียบร้อยแล้ว เป็นต้น

ในฐานข้อมูลออราเคิลสามารถใช้คุณสมบัติที่เกี่ยวกับเวลาได้โดยการใช้คำสั่งในการกำหนดดังนี้

SET TIME ON

เมื่อกระทำคำสั่งดังกล่าวแล้วโปรแกรมฐานข้อมูลจะแสดงเวลาขึ้นมาบนหน้าจอดังรูป 6.3



รูปที่ 6.3 แสดงตัวอย่างการตั้งเวลาการทำงาน

หลังจากนั้นให้เริ่มทำการทดลองตามกิจกรรมทั้งสี่ตามลำดับ โดยการทดลองในครั้งนี้จะแสดงตัวอย่างข้อมูลที่ใช้ในการทดลองของบริษัทไปรษณีย์ไทย จำกัด ซึ่งมีข้อมูลดังนี้

บทบาท "ROAPRD" มีหน้าที่ความรับผิดชอบดังนี้

- กำหนดสิทธิ์ให้กับพนักงานทุกคน มีรายละเอียดการทำงานคือ
    - ตรวจดูรายชื่อของพนักงานในองค์กร มีสิทธิ์การเข้าถึงข้อมูลดังนี้

* อ่านข้อมูลจากตารางพนักงาน

● ตรวจดูรายชื่อของบทบาทที่มีอยู่ในองค์กร

* อ่านข้อมูลจากตารางบทบาท

● กำหนดสิทธิ์ให้กับพนักงาน

* เขียนข้อมูลลงตารางที่เกี่ยวข้องกับรูปแบบ ERBAC03 ทั้งหมด

- โอนย้ายข้อมูลทั้งหมดที่ส่งมาจากที่ทำการไปรษณีย์ทั่วประเทศลงฐานข้อมูลระบบควบคุมส่วนกลาง มีรายละเอียดการทำงานคือ

● ตรวจสอบความถูกต้องของข้อมูลที่ได้รับมาจากที่ทำการไปรษณีย์

* อ่านข้อมูลจากตาราง EOD_HSTRY

* อ่านข้อมูลจากตาราง EOP_HSTRY

* อ่านข้อมูลจากตารางทรานแซคชั่น

● ตรวจสอบรายชื่อที่ทำการไปรษณีย์ทั้งหมด

* อ่านข้อมูลจากตาราง LOCATIONS

* อ่านข้อมูลจากตาราง LH

● บันทึกข้อมูลทั้งลงฐานข้อมูลโดยแยกตามตาราง

* เขียนข้อมูลได้ทุกตารางของระบบควบคุมส่วนกลาง

- เปลี่ยนแปลงโครงสร้างของฐานข้อมูลระบบควบคุมส่วนกลาง

● ตรวจสอบรายละเอียดการทำงานที่เกิดขึ้นของระบบควบคุมส่วนกลาง

* อ่านข้อมูลจากวิว V$LOG_HISTORY

● แก้ไขค่าพารามิเตอร์ของระบบฐานข้อมูล

* เขียนแฟ้มข้อมูลลงฮาร์ดดิสค์

- สำรองและกู้คืนข้อมูลของระบบควบคุมส่วนกลาง

● ตรวจสอบเนื้อที่จัดเก็บข้อมูล

* อ่านข้อมูลจากจากฮาร์ดดิสค์

● ตรวจสอบปัญหาที่เกิดขึ้นของระบบควบคุมส่วนกลาง

* อ่านข้อมูลจากวิว V$RECOVERY_LOG

* อ่านข้อมูลจากวิว V$INSTANCE_RECOVERY

● สำรองข้อมูลระบบทั้งหมด

* ประมวลผลโปรแกรมสำรองข้อมูล

● จัดเก็บข้อมูลที่สำรองไว้แล้วลงสื่อ

* เขียนแฟ้มข้อมูลลงอุปกรณ์ต่อพ่วง

จากรายละเอียดทั้งหมดที่กำหนดโดยนโยบายขององค์กร นำมาทดลองการสร้างสิทธิ์ใหม่ให้กับผู้ใช้ที่มีอยู่เดิม สามารถนำมาแปลงลงฐานข้อมูลของรูปแบบ RBAC และ ERBAC03 ได้ตามคำสั่งเอสคิวแอลดังขั้นตอนดังนี้

ขั้นตอนที่ 1 สำหรับทั้งสองรูปแบบให้พิจารณาข้อมูลของบทบาทที่เพิ่มขึ้นว่ามีอยู่แล้วหรือไม่ ถ้ายังไม่มี ให้พิมพ์คำสั่ง

INSERT INTO ROLES VALUES(1, "ROAPRD");

ขั้นตอนที่ 2 เฉพาะรูปแบบ ERBAC03 ให้พิจารณารายละเอียดงานที่ทำทั้งหมด ซึ่งถ้ามีการกำหนดใหม่ทั้งหมด ให้พิมพ์คำสั่ง

INSERT INTO JOBS VALUES(1, "Define privilege for employee");

INSERT INTO JOBS VALUES(2, "Transfer data from post office to EMS");

INSERT INTO JOBS VALUES(3, "Change database structure");

INSERT INTO JOBS VALUES(4, "Backup and recovery EMS database");

ขั้นตอนที่ 3 เฉพาะรูปแบบ ERBAC03 ให้พิจารณางานย่อยทั้งหมด ซึ่งถ้ามีการกำหนดใหม่ทั้งหมด ให้พิมพ์คำสั่ง

INSERT INTO TASKS VALUES(1, "Check the employee's name");

INSERT INTO TASKS VALUES(2, "Check the role's name");

INSERT INTO TASKS VALUES(3, "Add privilege to employee");

INSERT INTO TASKS VALUES(4, "Check data accuracy from post office");

INSERT INTO TASKS VALUES(5, "Check the post office's name");

INSERT INTO TASKS VALUES(6, "Record data to database separate by EMS table");

INSERT INTO TASKS VALUES(7, "Check database log");

INSERT INTO TASKS VALUES(8, "Edit system parameter");

INSERT INTO TASKS VALUES(9, "Check hard disk's free space");

INSERT INTO TASKS VALUES(10, "Check EMS problem");

INSERT INTO TASKS VALUES(11, "Backup EMS database");

INSERT INTO TASKS VALUES(12, "Record the backup file to media");

ขั้นตอนที่ 4 สำหรับทั้งสองรูปแบบพิจารณาใบอนุญาตทั้งหมด ซึ่งถ้ามีการกำหนดใหม่ทั้งหมด ให้พิมพ์คำสั่ง

INSERT INTO PERMS VALUES(1, "Read employee table");

INSERT INTO PERMS VALUES(2, "Read roles table");

```
INSERT INTO PERMS VALUES(3, "Write any ERBAC03 table");
INSERT INTO PERMS VALUES(4, "Read EOD_HSTRY table");
INSERT INTO PERMS VALUES(5, "Read EOP_HSTRY table");
INSERT INTO PERMS VALUES(6, "Read transaction table");
INSERT INTO PERMS VALUES(7, "Read locations table");
INSERT INTO PERMS VALUES(8, "read LH table");
INSERT INTO PERMS VALUES(9, "Write any EMS table");
INSERT INTO PERMS VALUES(10, "Read view V$LOG_HISTORY");
INSERT INTO PERMS VALUES(11, "Write file");
INSERT INTO PERMS VALUES(12, "Read file");
INSERT INTO PERMS VALUES(13, "Read view V$RECOVERY_LOG");
INSERT INTO PERMS VALUES(14, "Read view V$INSTANCE_RECOVERY");
INSERT INTO PERMS VALUES(15, "Execute backup program");
INSERT INTO PERMS VALUES(16, "Write file to media");
```

ขั้นตอนที่ 5 เฉพาะรูปแบบ ERBAC03 ให้กำหนดความสัมพันธ์ระหว่างใบอนุญาตกับงานย่อย ให้พิมพ์คำสั่ง

```
INSERT INTO TP VALUES(1, 1);
INSERT INTO TP VALUES(2, 2);
INSERT INTO TP VALUES(3, 3);
INSERT INTO TP VALUES(4, 4);
INSERT INTO TP VALUES(5, 4);
INSERT INTO TP VALUES(6, 4);
INSERT INTO TP VALUES(7, 5);
INSERT INTO TP VALUES(8, 5);
INSERT INTO TP VALUES(9, 6);
INSERT INTO TP VALUES(10, 7);
INSERT INTO TP VALUES(11, 8);
INSERT INTO TP VALUES(12, 9);
INSERT INTO TP VALUES(13, 10);
INSERT INTO TP VALUES(14, 10);
INSERT INTO TP VALUES(15, 11);
```

INSERT INTO TP VALUES(16, 12);

ขั้นตอนที่ 6 เฉพาะรูปแบบ ERBAC03 ให้กำหนดความสัมพันธ์ระหว่างงานย่อยกับงาน ให้พิมพ์คำสั่ง

INSERT INTO JT VALUES(1, 1);

INSERT INTO JT VALUES(2, 1);

INSERT INTO JT VALUES(3, 1);

INSERT INTO JT VALUES(4, 2);

INSERT INTO JT VALUES(5, 2);

INSERT INTO JT VALUES(6, 2);

INSERT INTO JT VALUES(7, 3);

INSERT INTO JT VALUES(8, 3);

INSERT INTO JT VALUES(9, 4);

INSERT INTO JT VALUES(10, 4);

INSERT INTO JT VALUES(11, 4);

INSERT INTO JT VALUES(12, 4);

ขั้นตอนที่ 7 เฉพาะรูปแบบ ERBAC03 ให้กำหนดความสัมพันธ์ระหว่างงานกับบทบาท ให้พิมพ์คำสั่ง

INSERT INTO RJ VALUES(1, 1);

INSERT INTO RJ VALUES(2, 1);

INSERT INTO RJ VALUES(3, 1);

INSERT INTO RJ VALUES(4, 1);

ขั้นตอนที่ 8 เฉพาะรูปแบบ RBAC ให้กำหนดความสัมพันธ์ระหว่างใบอนุญาตกับบทบาท ให้พิมพ์คำสั่ง

INSERT INTO PA VALUES(1, 1);

INSERT INTO PA VALUES(2, 1);

INSERT INTO PA VALUES(3, 1);

INSERT INTO PA VALUES(4, 1);

INSERT INTO PA VALUES(5, 1);

INSERT INTO PA VALUES(6, 1);

INSERT INTO PA VALUES(7, 1);

INSERT INTO PA VALUES(8, 1);

```
INSERT INTO PA VALUES(9, 1);
INSERT INTO PA VALUES(10, 1);
INSERT INTO PA VALUES(11, 1);
INSERT INTO PA VALUES(12, 1);
INSERT INTO PA VALUES(13, 1);
INSERT INTO PA VALUES(14, 1);
INSERT INTO PA VALUES(15, 1);
INSERT INTO PA VALUES(16, 1);
```

ขั้นตอนที่ 9 สำหรับทั้งสองรูปแบบให้กำหนดความสัมพันธ์ระหว่างผู้ใช้กับบทบาท โดยกำหนดบทบาทให้กับผู้ใช้ชื่อ "Burin" ซึ่งกำหนด ID เท่ากับ 1 ให้พิมพ์คำสั่ง

```
INSERT INTO UA VALUES(1, 1);
```

จาก 9 ขั้นตอนทั้งหมดนี้ รูปแบบ RBAC จะกระทำเฉพาะขั้นตอนที่ 1, 4, 8 และ 9 เท่านั้น ส่วนรูปแบบ ERBAC03 ทำเฉพาะขั้นตอน 1-7 และ 9 เมื่อมีการกำหนดครบสมบูรณ์แล้ว จะทราบถึงเวลาที่ใช้ทั้งหมดในการกำหนดบทบาทใหม่ให้กับผู้ใช้ที่มีอยู่เดิมของทั้งสองรูปแบบ

จากตัวอย่างข้างต้นเป็นการทดลองเพียงกิจกรรมเดียวเท่านั้น ซึ่งเป็นกิจกรรมที่สาม ส่วนกิจกรรมอื่นๆ ที่เหลือจะมีวิธีการทดลองคือ

- กิจกรรมที่ 1 จะเริ่มจับเวลาตั้งแต่การเพิ่มผู้ใช้จนถึงการกำหนดบทบาทให้กับผู้ใช้จนเสร็จสิ้น
- กิจกรรมที่ 2 จะเริ่มจับเวลาตั้งแต่การเพิ่มหรือลบ งาน งานย่อย หรือใบอนุญาต พร้อมทั้งกำหนดความสัมพันธ์ใหม่ทั้งหมดให้มีความถูกต้อง และสิ้นสุดการจับเวลาจนถึงการกำหนดความสัมพันธ์เสร็จสิ้นแล้ว
- กิจกรรมที่ 4 จะเริ่มจับเวลาตั้งแต่การลบข้อมูลผู้ใช้ออกจากตาราง ซึ่งมีเพียงคำสั่งเดียว

การจับเวลาทั้งหมดนี้จะไม่รวมถึงเวลาคิดในการพิมพ์คำสั่งของผู้บริหารจัดการระบบฯ แต่จะเป็นเวลาที่ได้จากการคีย์คำสั่งเอสคิวแอลที่จัดเตรียมไว้แล้วลงสู่ฐานข้อมูลเท่านั้น ช่วงเวลาที่ได้จากการปฏิบัติกิจกรรมทั้งหมด แต่ละกิจกรรมให้นำมาหาค่าเฉลี่ยอย่างง่ายในการกำหนดให้กับผู้ใช้ต่อคน เพื่อให้ได้ผลลัพธ์ของการทดลองนี้

6.2 ผลการทดลอง

6.2.1 ผลการทดลองสมมุติฐานการลดความถี่ในการบุกรุก

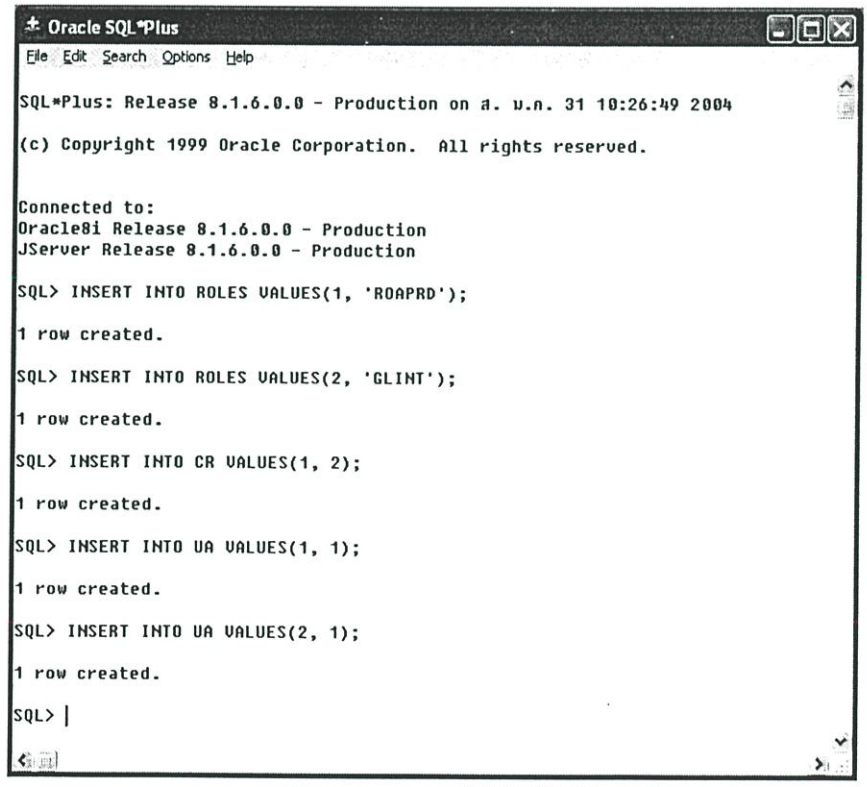
จากตัวอย่างข้อมูลหนึ่งสัปดาห์ เก็บข้อมูลการล็อกอินผู้ใช้จำนวนทั้งสิ้น 4,244 ครั้ง จาก 26 ที่ตั้งที่แตกต่างกันและข้อมูลที่ถูกต้องมีเพียง 270 ครั้ง นำไปคำนวณในสมการที่ 1 ได้ผลลัพธ์คือ

$$\text{ร้อยละความถูกต้องของข้อมูล} = \frac{270}{4,244} \times 100 = 6.36\ \%$$

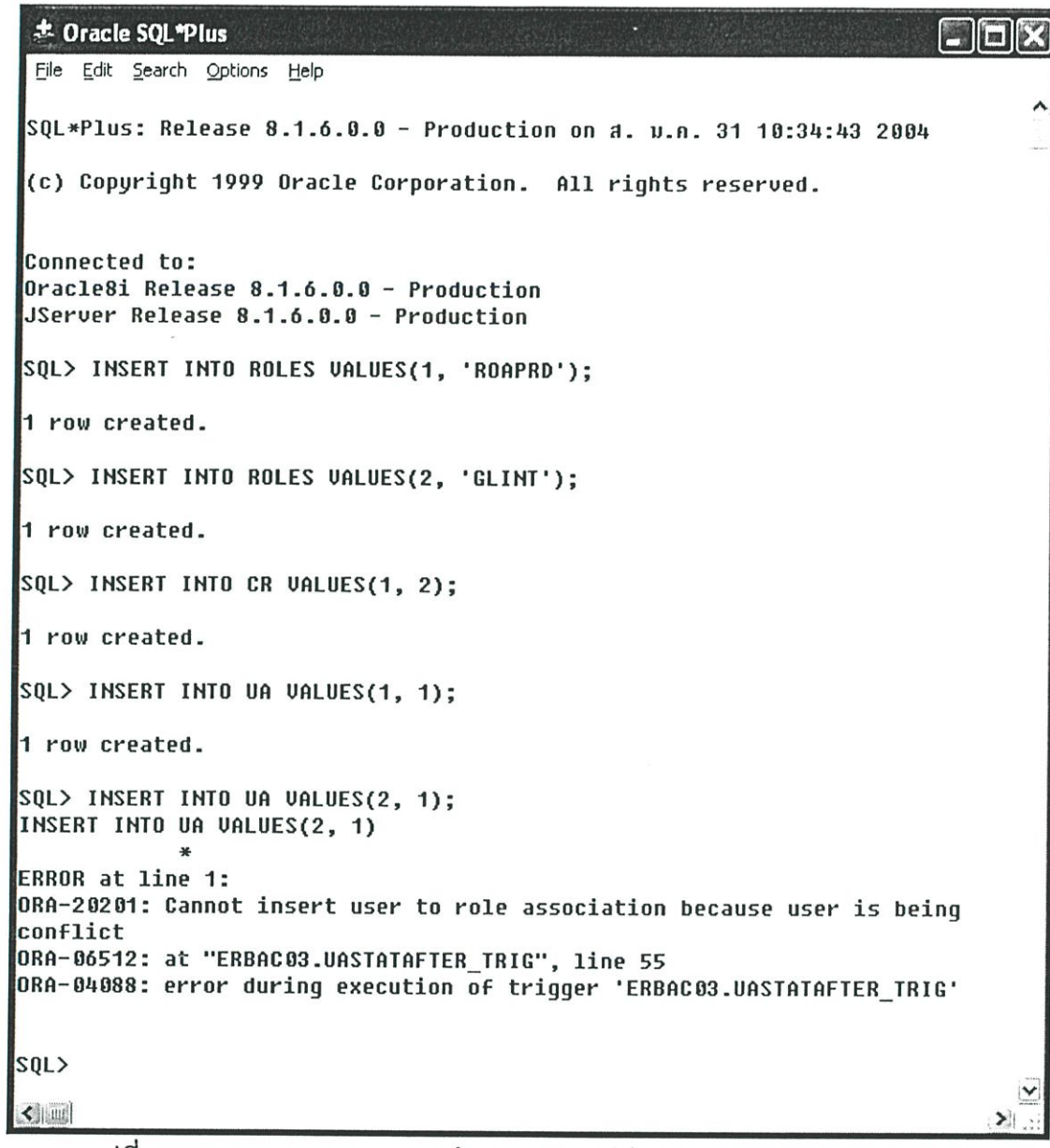
จากผลลัพธ์สรุปได้ว่ามีผู้ใช้ที่บุกรุกระบบควบคุมส่วนกลางสูงถึง 93.64 % ซึ่งการกระทำเช่นนี้เป็นเหตุให้ระบบควบคุมส่วนกลางล่ม 4 ครั้ง ซึ่งสามารถตรวจสอบได้จากแฟ้มเหตุการณ์ในฐานข้อมูลระบบฯ สาเหตุที่ระบบฯ หยุดทำงานเนื่องจากมีบุคคลที่ไม่มีส่วนเกี่ยวพยายามติดต่อใช้งานระบบควบคุมส่วนกลางเป็นจำนวนมากทำให้ระบบแบกรับภาระงานไม่ได้ (Denial of service) และเจ้าหน้าที่แผนกฐานข้อมูลไม่สามารถควบคุมการเข้าถึงจากสถานที่อื่นได้ จึงได้มีการทดลองด้วยรูปแบบ ERBAC03 ตามวิธีการและอัลกอริทึมที่ออกแบบข้างต้น ทำให้ขจัดการบุกรุกที่ไม่เหมาะสมนี้ได้ 100 % โดยใน 1 สัปดาห์มีการเข้าถึงข้อมูลด้วยบทบาท "ROAPRD" ทั้งสิ้น 353 ครั้ง และมีการเข้าถึงข้อมูลในสถานที่ที่กำหนดไว้ทั้งหมด 353 ครั้งเช่นกัน อีกทั้งระบบควบคุมส่วนกลางมีการหยุดการทำงานแค่ 1 ครั้งเท่านั้น เพื่อทำการสำรองข้อมูลระบบทั้งหมดประจำสัปดาห์

6.2.2 ผลการทดลองสมมุติฐานการเพิ่มความถูกต้องของข้อมูล

เมื่อทำการทดลองแล้วได้ผลการทดลองดังรูปที่ 6.4 และ 6.5 ตามลำดับ



รูปที่ 6.4 แสดงผลการทดลองการกำหนดบทบาทให้กับผู้ใช้ด้วยรูปแบบ RBAC ของ Perelson [39]



รูปที่ 6.5 แสดงผลการทดลองการกำหนดบทบาทให้กับผู้ใช้ด้วยรูปแบบ ERBAC03

จากรูป 6.4 แสดงถึงการกำหนดบทบาทที่ขัดแย้งให้กับผู้ใช้ได้ ซึ่งตามทฤษฎีข้อบังคับแบบสแตติกแล้วนั้น จะไม่สามารถกระทำได้ โดยข้อมูลในระบบฯจะไม่ถูกต้อง ส่วนรูป 6.5 เป็นการกำหนดบทบาทให้กับผู้ใช้โดยใช้รูปแบบ ERBAC03 ซึ่งแสดงการป้องกันการกำหนดบทบาทที่ขัดแย้งกันให้กับผู้ใช้ที่เคยได้รับบทบาทใดบทบาทหนึ่งไปแล้ว และข้อมูลการกำหนดบทบาท "GLINT" ให้กับผู้จะไม่ถูกบันทึกลงในระบบฯโดยเด็ดขาด เนื่องจากมีการกำหนดข้อบังคับป้องกันไว้

6.2.3 ผลการทดลองการหาเวลาการทำงาน

จากการทดลองทั้งสี่กิจกรรมได้ผลการทดลองซึ่งแสดงเป็นเวลาเฉลี่ยดังนี้

ตารางที่ 6.3 ผลการทดลองการหาเวลาการทำงานเฉลี่ยหนึ่งชุดคำสั่งต่อหนึ่งผู้ใช้

วิธีการ	การกำหนดสิทธิ์ที่มีอยู่ให้กับผู้ใช้คนใหม่ (นาที)	การเปลี่ยนแปลงสิทธิ์ให้แก่ผู้ใช้เดิม (นาที)	การสร้างสิทธิ์ใหม่ให้แก่ผู้ใช้ที่มีอยู่เดิม (นาที)	การยกเลิกสิทธิ์ (นาที)
$RBAC_0$	0.5	0.65	1.3	0.2
$ERBAC03_0$	0.5	0.65	2.2	0.2
ผลต่าง (%)	**0.0**	**0.0**	**40.9**	**0.0**

จากการทดลองทั้งสี่กิจกรรมที่กระทำซ้ำตลอดเดือนดังตารางที่ 6.3 ปรากฏว่ารูปแบบ ERBAC03 มีเวลาการกำหนดสิทธิ์ที่มีอยู่เดิมให้กับผู้ใช้เท่ากับรูปแบบ RBAC และเวลาการเปลี่ยนแปลงสิทธิ์ให้แก่ผู้ใช้เดิมก็เท่ากับรูปแบบ RBAC เช่นกัน ส่วนเวลาการสร้างสิทธิ์ใหม่ของรูปแบบ ERBAC03 จะมีค่าความแตกต่างมากที่สุดจากทั้งสี่กิจกรรม เนื่องจากปัจจัยที่มีผลกระทบต่อกิจกรรมที่สามนี้คือ จำนวนหน้าที่ในงานใหม่ ซึ่งแปรผันตรงต่อเวลาในการสร้างสิทธิ์ใหม่นี้ ซึ่งถ้ามีรายละเอียดของงานมากขึ้น ก็จะทำให้เวลาในการสร้างสิทธิ์ทั้งหมดก็มากตามไปด้วย ส่วนกิจกรรมสุดท้ายมีเวลาไม่แตกต่างจากรูปแบบของ RBAC เลย ทั้งนี้เนื่องจากการยกเลิกสิทธิ์จะทำการลบความสัมพันธ์ระหว่างผู้ใช้กับบทบาทออกเท่านั้น

จากการทดลองการหาเวลาการทำงานสรุปได้ว่าการกระทำกิจกรรมทั้งสี่โดยใช้รูปแบบ ERBAC03 สูญเสียเวลาที่ไม่แตกต่างจากวิธีของ RBAC มากนัก เพื่อแลกกับประโยชน์ที่ได้คือข้อมูลรายละเอียดของหน้าที่ความรับผิดชอบ ซึ่งมีไว้ตรวจสอบถึงความชัดเจนในการกำหนดของผู้บริหารจัดการระบบฯ เองว่ามีการกำหนดครบถ้วนหรือผิดพลาดหรือไม่ อีกทั้งผู้บริหารจัดการระบบยังสามารถบอกได้ว่าสิทธิ์ที่ผู้ใช้แต่ละคนได้รับนั้น นำไปใช้ปฏิบัติงานใดได้อีกด้วย

บทที่ 7
สรุปผลการวิจัยและข้อเสนอแนะ

องค์กรขนาดใหญ่ให้ความสำคัญต่อการรักษาความปลอดภัยในเรื่องของการควบคุมการเข้าถึงเป็นอย่างมาก ซึ่งการควบคุมการเข้าถึงแบบโรลเบสได้รับความสนใจในปัจจุบัน ถึงแม้ว่าการควบคุมการเข้าถึงแบบโรลเบสเป็นรูปแบบที่ดี แต่การบริหารจัดการ รวมถึงการสร้างและดูแลรักษาข้อมูลการควบคุมการเข้าถึงยังคงเป็นปัญหาใหญ่สำหรับองค์กรขนาดใหญ่ จึงได้มีการวิจัยเพื่อแก้ปัญหาที่เกิดขึ้น และสำหรับวิทยานิพนธ์ฉบับนี้ได้นำเสนอถึงวิธีการที่เหมาะสมสำหรับการนำรูปแบบการควบคุมการเข้าถึงแบบโรลเบสไปประยุกต์ใช้ร่วมกับองค์กรที่มีลักษณะการดำเนินการของระบบคอมพิวเตอร์ในสภาวะแวดล้อมแบบกระจาย ซึ่งได้มีการพัฒนารูปแบบการควบคุมการเข้าถึงใหม่ที่เรียกสั้นๆ ว่า "ERBAC03" โดยรูปแบบใหม่นี้มีการกำหนดองค์ประกอบขึ้นมาใหม่จากที่มีอยู่เดิมคือ ที่ตั้ง (location) งาน (job) และงานย่อย (task) ซึ่งที่ตั้งกำหนดเพื่อใช้รองรับการดำเนินงานขององค์กรที่มีลักษณะที่ตั้งอยู่หลายสาขา อีกทั้งในแต่ละสาขายังมีบทบาทที่รับผิดชอบเหมือนกันด้วย และรูปแบบ ERBAC03 ยังมีการนำวิธีการลำดับชั้นของที่ตั้งมาปรับใช้ร่วมกันเพื่อประสิทธิภาพที่ดีขึ้นด้วย โดยสามารถใช้ข้อมูลที่กำหนดระดับความปลอดภัยตามลำดับขั้นของที่ตั้งสำหรับควบคุมการให้สิทธิ์และยกเลิกสิทธิ์ของผู้ใช้ได้ด้วย ส่วนงานและงานย่อยออกแบบมาเพื่อบ่งบอกถึงความชัดเจนในการกำหนดหน้าที่ความรับผิดชอบในแต่ละบทบาทว่าปฏิบัติงานอะไรบ้าง ซึ่งถือว่าการออกแบบเช่นนี้มีประโยชน์มากต่อผู้บริหารจัดการระบบฯ เพื่อให้การกำหนดหน้าที่ความรับผิดชอบให้กับบทบาทไม่มีความผิดพลาดเกิดขึ้น เนื่องจากต้องกำหนดรายละเอียดของงานที่บทบาทนั้นต้องทำให้ครบถ้วน ถึงจะกำหนดใบอนุญาตให้กับบทบาทนั้นได้ทำให้ไม่พลาดงานบางงานที่ต้องปฏิบัติได้ และข้อดีอีกประการหนึ่งคือผู้บริหารจัดการระบบฯ สามารถตรวจสอบได้โดยดูจากใบอนุญาต โดยสามารถบอกได้ว่าใบอนุญาตนี้มีไว้เพื่องานใดที่บทบาทนั้นต้องปฏิบัติ

การพัฒนารูปแบบ ERBAC03 ระดับที่ 1 ที่ประกอบไปด้วยองค์ประกอบทั้งสามนี้ จะสามารถป้องกันการถูกบุกรุกจากผู้ใช้ที่ไม่มีสิทธิ์ปฏิบัติหน้าที่ในที่ตั้งนั้นๆ ได้ เช่น ผู้ใช้ที่เป็นหัวหน้าไปรษณีย์ที่จังหวัดกรุงเทพฯ ก็ไม่สามารถไปปฏิบัติงานที่จังหวัดชลบุรีได้ เป็นต้น แต่ในรูปแบบเดิมไม่สามารถป้องกันตรงจุดนี้ได้ ดังตัวอย่างการทดลองแสดงให้เห็นได้ว่าระบบฯที่มีการใช้รูปแบบ RBAC ของบริษัท ไปรษณีย์ไทย จำกัด ถูกผู้ใช้ที่มีสิทธิ์เข้าถึงระบบฯ เข้าถึงจากสถานที่ที่แผนกฐานข้อมูลไม่สามารถควบคุมได้ ทำให้ระบบหยุดทำงานในที่สุด ซึ่งจากการทดลองสมมุติฐานแรกคือลดความถี่ในการบุกรุก สามารถบอกได้ว่ารูปแบบ ERBAC03 มีการป้องกันผู้ใช้จากที่ตั้งไม่

เหมาะสมได้ถึง 100% ส่วนรูปแบบ RBAC ไม่มีคุณลักษณะการป้องกันจากที่ตั้งที่ไม่เหมาะสมได้ จึงทำให้มีการบุกรุกสูงถึง 93.64% ซึ่งถ้ารูปแบบ RBAC ได้มีการกำหนดคุณลักษณะที่ตั้งโดยการเพิ่มฟิลด์ที่ตั้ง ก็สามารถขจัดการบุกรุกจากที่ตั้งที่ไม่เหมาะสมได้เช่นกัน ซึ่งทำให้เกิดปัญหาในเรื่องการจัดการข้อมูลของผู้บริหารจัดการระบบฯที่ไม่เหมาะสม แต่การกำหนดรายละเอียดที่มากขึ้นนี้ก็ต้องแลกกับเวลาที่สูญเสียไปมากขึ้นกว่ารูปแบบ RBAC ซึ่งจากการทดลองทำให้ทราบว่าเวลาเพิ่มขึ้นมีเฉพาะในกิจกรรมที่สามคือการสร้างสิทธิ์ใหม่ให้กับผู้ใช้ที่มีอยู่เดิม ซึ่งเพิ่มขึ้นถึง 40.9% ส่วนกิจกรรมที่เหลือไม่มีความแตกต่างในเรื่องของเวลาเลย

เนื่องจากรูปแบบ ERBAC03 ระดับ 1 ยังคงไม่สามารถป้องกันผู้ใช้ที่มีสิทธิ์เข้าถึงเข้ามูลเพื่อประโยชน์ส่วนตนได้ จึงได้มีการยกระดับรูปแบบ ERBAC03 เป็นระดับ 3 โดยมีการเพิ่มคุณสมบัติที่เรียกว่าข้อจำกัด (constraint) และชนิดของข้อจำกัดที่นำมาปรับใช้นี้คือ หลักการแบ่งแยกหน้าที่แบบสแตติก (static separation of duty) เป็นหลักการที่สร้างขึ้นเพื่อความคงสภาพความปลอดภัยของระบบฯ ซึ่งหลักการแบบสแตติกนี้จะใช้วิธีการจำกัดสิทธิ์ของผู้ใช้ให้ไม่สามารถทำการทุจริตได้ โดยมีการกำหนดสิทธิ์ที่ขัดแย้งกันด้วย ถ้าสิทธิ์ใดๆ กำหนดให้ขัดแย้งกัน ก็แสดงว่าจะไม่สามารถกำหนดสิทธิ์ทั้งสองให้กับบทบาทเดียวกันได้ ซึ่งจากการทดลองแสดงให้ทราบว่าการกำหนดบทบาทที่ขัดแย้งให้กับผู้ใช้ในรูปแบบ ERBAC03 จะไม่สามารถกระทำได้ แต่รูปแบบ RBAC สามารถบันทึกข้อมูลที่ผิดพลาดลงไปในฐานข้อมูลได้

จากการทดลองทั้งหมดกระทำโดยใช้วิธีการพิมพ์คำสั่งเอสคิวแอล (Structure Query Language: SQL) ซึ่งในความเป็นจริงแล้วการที่จะให้ผู้บริหารจัดการระบบฯใช้งานวิธีนี้จึงไม่สะดวกเท่าใด งานวิจัยฉบับนี้จึงได้มีการออกแบบโปรแกรมส่วนการจัดการระบบฯ ตามรูปแบบ ERBAC03 เพื่อให้เจ้าหน้าที่ที่มีส่วนเกี่ยวข้องใช้งานได้ง่ายขึ้น โดยโปรแกรมนี้พัฒนาด้วยภาษาจาวาที่มีคุณสมบัติเด่นมากมาย และโปรแกรมยังนำฐานข้อมูลออราเคิลมาใช้งานร่วมกันด้วย ซึ่งมีข้อดีอีกอย่างหนึ่งคือการตรวจสอบเงื่อนไขทั้งหมดจะถูกกระทำที่ฝั่งฐานข้อมูล ส่วนฝั่งไคเอนท์ไม่ต้องมีการตรวจสอบเงื่อนไขใดๆ ทั้งสิ้นทำให้การกำหนดหรือเปลี่ยนแปลงข้อจำกัดจะถูกกระทำเพียงแห่งเดียว

งานวิจัยฉบับนี้ยังคงมีข้อจำกัดบางประการคือการขาดความยืดหยุ่น ซึ่งหมายถึงการนำงานวิจัยนี้ไปปรับใช้ร่วมกับองค์กรเพื่อให้เกิดประสิทธิภาพสูงสุดนั้น ต้องมีการเปลี่ยนแปลงระบบการทำงานขององค์กรค่อนข้างมาก เนื่องจากการเข้มงวดในการกำหนดบทบาทให้กับผู้ใช้ โดยผู้ใช้สามารถได้รับบทบาทที่ไม่ขัดแย้งกันเท่านั้น ซึ่งในความเป็นจริงนั้น ผู้ใช้อาจจะได้รับบทบาทที่ขัดแย้งกันได้ แต่ต้องมีข้อจำกัดให้ผู้ใช้ไม่สามารถปฏิบัติบทบาทได้ในเวลาเดียวกัน หลักการนี้เรียกว่า หลักการแบ่งแยกหน้าที่แบบไดนามิก (dynamic separation of duty) โดยรูปแบบ ERBAC03 ในอนาคตจะพัฒนาให้รองรับข้อจำกัดประเภทนี้ได้ อีกทั้งอาจจะมีการเพิ่ม

องค์ประกอบที่เกี่ยวกับเวลาเข้ามาใช้เพื่อกำหนดช่วงเวลาการเข้าถึงระบบฯ ที่รองรับงานประเภทที่มีลักษณะการทำงานเป็นเวลาเสมือนการเข้ากะ

สุดท้ายนี้การออกแบบโปรแกรมส่วนจัดการแบบกราฟฟิคสามารถออกแบบให้มีประสิทธิภาพมากขึ้นโดยการแปลงข้อมูลที่มีอยู่ในตารางทั้งหมดเป็นสิทธิ์ที่ถูกกำหนดในฐานข้อมูลจริงเสมือนผู้ใช้ติดต่อกับฐานข้อมูลโดยตรง

เอกสารอ้างอิง

[1] Lampson B.W. "Protection." In 5th Princeton Symposium on Information Science and Systems, 1971. Pp. 437-443.

[2] Graham G.S. and Denning P.J. "Protection – Principles and Practice." In AFIPS Spring Joint Computer Conference, vol. 40, 1972 Pp. 417-429.

[3] Harrison M.H., Ruzzo W.L. and Ullman J.D. "Protection in Operating Systems." Communications of the ACM, vol. 19, no. 8, 1976. Pp. 461-471.

[4] Lipton R.J. and Snyder L. "A Linear Time Algorithm for Deciding Subject Security." Journal of the ACM, vol. 24, no. 3, 1977. Pp. 455-464.

[5] Sandhu R. "The Schematic Protection Model: Its Definition and Analysis for Acylic Attenuating Schemes." Journal of the ACM, vol. 35, no. 2, April 1988. Pp.404-432.

[6] Ammann P.E., Lipton R.J. and Sandhu R. "The Expressive Power of Multi-Parent Creation in Monotonic Access Control Models." In Proceedings of IEEE Computer Security Foundations Workshop, June 1992. Pp. 148-156.

[7] Sandhu R. "Expressive Power of the Schematic Protection Model." The Journal of Computer Security, vol. 1, no. 1, 1992. Pp. 59-98.

[8] Ammann P.E. and Sandhu R. "Implementing Transaction Control Expressions by Checking for Absence of Access Rights." In Proceedings of 8th Annual Computer Security Application Conference, December 1992. Pp. 131-140.

[9] Sandhu R. and Ganta S. "On Testing for Absence of Rights in Access Control Models." In Proceedings of IEEE Computer Security Foundations Workshop, June 1993. Pp. 109-118.

[10] Sandhu R. and Suri G. "Non-Monotonic Transformations of Access Rights." In Proceedings of IEEE Symposium on Research in Security and Privacy, May 1992. Pp. 148-161.

[11] Sandhu R. and Samarati P. "Access Control: Principles and Practice." IEEE Communications, vol. 32, no. 9, 1994. Pp. 40-48.

[12] Sandhu R., Edward J.C., Feinstein H.L. and Charles E.Y. "Role-Based Access Control Models." **IEEE Computer**, vol. 29, no. 2, Febuary 1996. Pp. 38-47.

[13] Sandhu R., Editor. **Role-Based Access Control.** Zelkowitz : Academic Press. 1998.

[14] Laboratory for Information Security Technology. "Role-Based Access Control." [Online]. Availble : http://www.isse.gmu.edu/faculty/sandhu/index.htm. 1997.

[15] Mavridis I., Pangalos G., Khair M. and Bozios L. "Defining Access Control Mechanisms for Privacy Protection in Distributed Medical Databases." **In Proceedings of IFIP Working Conference on User Identification and Privacy Protection**, Stockholm, Sweden, June 1999.

[16] Epstien P. and Sandhu R. "Engineering of Role/Permission Assignments." 17th Annual Computer Security Applications Conference, December 2001. Pp. 127-136.

[17] Kern A. "Advanced Features for Enterprise-Wide Role-Based Access Control." **18th Annual Computer Security Applications Conference**, December 2002.

[18] Moyer M.J. and Abamad M. "Generalized role-based access control." Proceedings 21st International Conference on Distributed Computing Systems, 2001. Pp. 391-398.

[19] Thomas E. and Biddle B., Editor. **The Nature and History of Role Theory, in Role Theory: Concepts and Research.** Kreiger Publishing. 1979.

[20] Ting T.C. and Landwehr C.E., Editor. **A User-Role Based Data Security Approach, in Database Security: Status and Prospects.** Elsevier. 1988.

[21] Coyne E. and Youman C. "Workshop Discussion." **In Proceedings of First ACM Workshop on Role-Based Access Control**, Gaithersburg, MD, November 30-December 1, 1995.

[22] Coyne E. "Role Engineering." **In Proceedings of First ACM Workshop on Role-Based Access Control**, Gaithersburg, MD, November 30-December 1, 1995.

[23] Nyanchama M. and Osborn., Spooner D.L., Demurjian S.A. and Dobson J.E., Editor. "Modeling Mandatory Access Control in Role-Based Security Systems." **Proceedings of the IFIP WG 11.3 Ninth Annual Working Conference on Database Security**. Chapman & Hall. 1995.

[24] Osborn S. "Mandatory Access Control and Role-Based Access Control Revisited." **In Proceedings of Second ACM Workshop on Role-Based Access Control**, November 1997.

[25] Sandhu R. "Role Hierarchies and Constraints for Lattice-Based Access Controls." **In Computer Security – ESORICS 96**, Springer Verlag, 1996.

[26] Sandhu R. and Munawer Q. "How to do Discretionary Access Control using Roles." **In Proceedings of the 3rd ACM Workshop on Role-Based Access Control**, Fairfax, Virginia, September 1998.

[27] Osborn S. and Sandhu R. "Configuring Role-based Access Control to Enforce Mandatory and Discretionary Access Control Policies." **ACM Transaction on Information and System Security**, vol. 3, no. 2, May 2000. Pp. 85-106.

[28] Gasser M. and Mcdermott E. "An Architecture for Practical Delegation in a Distributed System." **1990 IEEE Computer Society Symposium on Research in Security and Privacy**, Oakland, CA, May 1990.

[29] Saltzer J.H. and Schroeder M.D. "The Protection of Information in Computer Systems." **In Proceedings of IEEE**, vol. 63, 1975. Pp. 1278-1308.

[30] Clark D.D. and Wilson D.R. "A Comparison of Commercial and Military Computer Security Policies." **In Proceedings of IEEE Symposium on Security and Privacy**, April 1987. Pp. 184-194.

[31] Baldwin R.W. "Naming and Grouping Privileges to Simplify Security Management in Large Database." **In Proceedings 1990 IEEE Symposium on Security and Privacy**, May 1990. Pp. 116-132.

[32] Gligor V.D., Gavrila S.I. and Ferraiolo D. "On the Formal Definition of Separation of Duty Policies and their Composition." **In Proceedings of IEEE Symposium on Security and Privacy**, Oakland, California, May 1998.

[33] Nash M.J. and Poland K.R. "Some Conundrums Concerning Separation of Duty." In Proceedings of the IEEE Symposium on Security and Privacy, May1990. Pp. 201-207.

[34] Sandhu R. "Transaction Control Expressions for Separation of Duties." In Proceedings of 4th Aerospace Computer Security Conference, December 1988. Pp. 282-286.

[35] Sandhu R. "Separation of Duties in Computerized Information Systems." In Proceedings of IFIP WG11.3 Workshop on Database Security, September 1990.

[36] Simon R. and Zurko M.E. "Separation of Duty in Role-Based Environments." In Proceedings of 10th Computer Security Foundation Workshop, Rockport, Massachusetts, June 1997.

[37] Kuhn D.R. "Mutual Exclusion of Roles as a Means of Implementing Separation of Duty in Role-Based Access Control Systems." In Proceedings of the 2nd ACM Workshop on Role-Based Access Control, Fairfax, VA, October 1997.

[38] Botha R.A. and Eloff J.H.P. "Separation of Duties for Access Control Enforcement in Workflow Environments." IBM Systems Journal, vol. 40, no. 3, 2001.

[39] Perelson S. and Botha R.A. "Conflict Analysis as a Means of Enforcing Static Separation of Duty Requirements in Workflow Environments." South African Computer Journal, vol. 26, November 2000. Pp. 212-216.

[40] Nyanchama M. and Osborn S. "The Role Graph Model and Conflict of Interest." ACM Transactions on Information and System Security, vol. 2., no. 1, February 1999. Pp. 3-33.

[41] Ahn G.J. and Sandhu R. "The RSL99 Language for Role-Based Separation of Duty Constraints." In Proceedings of the 4th ACM Workshop on Role-Based Access Control, Fairfax, Virginia, October 1999. Pp. 28-29.

[42] Freudenthal E. and Pesin T. "dRBAC: Distributed Role-Based Access Control for Dynamic Coalition Environments" In Proceedings of the 22nd International Conference on Distributed Computing Systems, 2002.

[43] Date C.J. **An Introduction to Database Systems.** 7th Ed. Reading: Addison-Wesley publishing Company, Inc. 2000.

[44] Hollingsworth D. "Workflow Management Coalition: The workflow reference model." [Online]. Available: URL http://www.wfmc.org. 1998.

[45] Chandramouli R. and Sandhu R. "Role-Based Access Control Features in Commercial Database Management Systems" **21st National Information Systems Security Conference**, Crystal City, Virginia, October 1998.

[46] Gregory T. "The Economic Impact of Role-Based Access Control." NIST Planning Report 02-1, March 2002.

ภาคผนวก ก.

ผลงานวิจัยที่ได้รับการตีพิมพ์

1. C. Sathitwiriyawong and B. Yenmunkong, "An Enhanced Role-Based Access Control for Large Enterprises," Proceeding of the 3rd International Symposium on Communications and Information Technologies, September 2003, pp. 279-283.
2. B. Yenmunkong and C. Sathitwiriyawong, "An Enhanced Role-Based Access Control Model using Constraint Features," Proceeding of the 7th National Computer Science and Engineering Conference, October 2003, pp. 103-108.

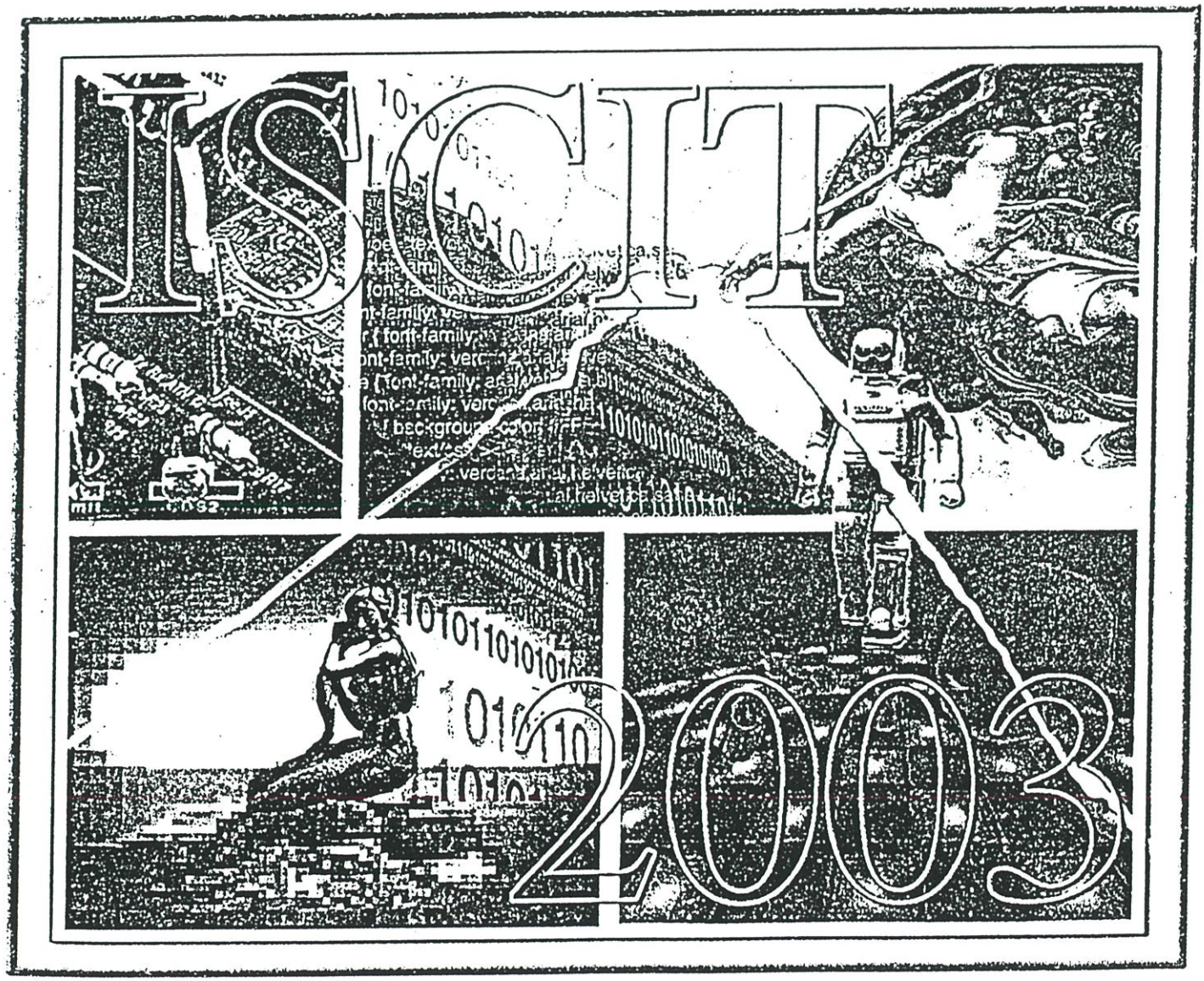
Volume I

Proceedings

The Third International Symposium
n Communications and Information Technologies

September 3-5, 2003

BP Samila Beach Hotel and Resort, Songkhla, Thailand



IT
C 2003
IS

CoE PSU
NECTEC
LOXINFO
IEEE
THAILAND SECTION
IEEE COMMUNICATIONS SOCIETY
THAILAND CHAPTER
TOT

An Enhanced Role-Based Access Control Model for Large Enterprises

C. Sathitwiriyawong and B. Yenmunkong

Faculty of Information Technology, and Research Center for Communication and Information Technology, King Mongkut's Institute of Technology Ladkrabang, Ladkrabang, Bangkok 10520, Thailand
Email: chanboon@it.kmitl.ac.th

Faculty of Information Technology, King Mongkut's Institute of Technology Ladkrabang, Ladkrabang, Bangkok 10520, Thailand
Email: bom_burin@hotmail.com

Abstract

Access control is one of the most important security issues for large organizations. Role-based access control (RBAC) has recently received considerable attention as a promising alternative to traditional discretionary and mandatory access control. Though RBAC is a good model, the administration of RBAC including building and maintaining access control information remains a difficult problem in large enterprises. RBAC model itself dose not tell the solution. Some researches were done on practical ways to find the information that fills RBAC components.

In this paper we suggest an extended model of RBAC for large enterprises. A new model consists of some mandatory and discretionary features called "Enhanced Role-Based Access Control (ERBAC03)". It also includes define the assignment of permissions in order to appropriate roles. Moreover, location hierarchies approach was applied to the new model for distributed environment. It is easy to manage access control information and makes benefit for large enterprises in the future.

1. Introduction

Role-based access control has recently received considerable attention as a promising alternative to traditional discretionary and mandatory access control. In RBAC96 model [6], permissions associated with roles and users are members of appropriate. This simplifies management of permission.

In large enterprises, the number of roles is created for the various job functions and able to double to hundreds or thousands. To manage these roles, permissions, users and interrelationships is a formidable task. It is often centralized in a small team of security administrator. Each enterprise has many branches, so we must improve efficiency of RBAC96 model for enterprises in the real world. The RBAC96 model was viewed as a distributed model, not a centralized model.

A core aspect of RBAC is the Role/Permission Relation. In addition, application developers have accepted the definition of RBAC permissions. However, the research has not given details on a systematic model for determining the assignment of permission to roles in a distributed environment. We emphasize that the focus of this paper is on a model for role to permission assignment based on location authorization control. The central contribution of this paper is to identify some approach possible for RBAC in distributed systems and to present a location hierarchy approach for user-role assignment and role-permission assignment. We classify location hierarchies based on organization policies and role classification. The objectives of this research are to improve RBAC model for efficiency of data security, to work in enterprises and to manage easily for RBAC administrator due to less complicated than the traditional model.

The rest of the paper is organized as follows. In Section 2, we outline related work for RBAC96 model and distributed access control. In Section 3, the definition of RBAC96 model is explained. In Section 4, we formally define a new model extended from traditional RBAC96 model for enhanced efficiency. Finally, we conclude this paper in Section 5.

2. Related works

Mavridis [1] defined access control mechanisms for privacy protection in distributed medical database that is defined on the basis of RBAC components and supports both mandatory and discretionary features.

Epstein [2] constructed a new model that extends the permission assignment of RBAC96 model between the roles and permissions. His goal was to define a layered model that served as a basis for detailing as effective methodology to assign permission to roles.

Kern [3] defined enterprise roles as "Enterprise Role-based Access Control (ERBAC)" capable of spanning all IT systems in an organization. This model supports the IT environments of a large enterprise consisting of a variety of platforms and applications. These include a number of operating systems (i.e., Windows NT/2000 and Unix).

Moyer [4] designed a security system based on a paradigm call "Gerenalized Role-based Access Control (GRBAC)". GRBAC is an extension of traditional role-based access control. It enhances traditional RBAC by incorporating the notion of object roles and environment roles. These new types of roles allow one to define rich and easy to understand security policies without having significant technical.

Gasser and McDemott [5] defined user to machine delegation as "the process whereby a user to machine environment authorizes a system to access remote resources on his behalf". The user's authorization of his process to act on his behalf is a form of delegation of rights from the user to the process. In some cases the user may delegate the right to one of several permissible roles or identities (i.e., by logging in using different names and/or passwords). Limited delegation also occurs routinely in multilevel secure systems where the user selects a single classification of his process that is a subset of the access class for which the user is authorized.

3. Role-based access control (RBAC) model

The model developed in this paper is constructed by extending the RBAC96 model [6]. The RBAC96 model comprises four models: $RBAC_0$, $RBAC_1$, $RBAC_2$ and $RBAC_3$. $RBAC_0$ is the base model. $RBAC_1$ and $RBAC_2$ add role hierarchies and constraints, respectively. $RBAC_3$ is a consolidated model. RBAC96 also makes a distinction between user and administrative roles.

There are three components of RBAC96 that we are inherited in using for the extension of the model: user(U), roles(R) and permissions(P).

A user is defined as a human being. Although the concept of a user can be extended to include machines, networks, or intelligent autonomous agents, for simplicity reasons we limit a user to person in this article. A role is a job function within the context of an organization with some associated semantics regarding the authority and responsibility conferred on the user assigned to the role. A role may be responsible to perform more than one type of work. Permission is an approval to perform an operation on one or more objects.

The model, as depicted in figure 1, defined the components of role/permission assignment by PA. It also defined the role hierarchy (RH) and the component of user/role assignment by UA.

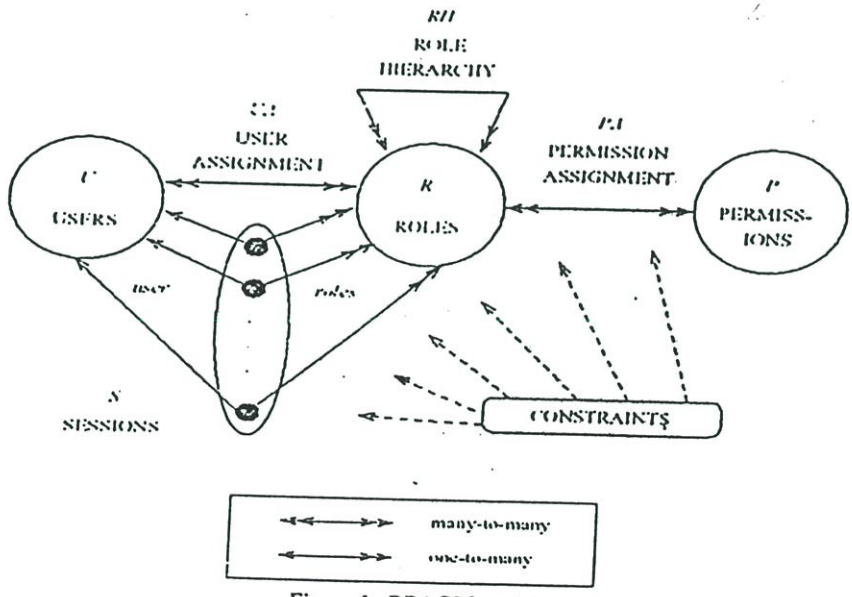


Figure 1. RBAC96 model

Consider a short formal summary of the relevant components in the RBAC96 model.

Definition 3.1: For RBAC96 entities
U = a set of users $\{u_1, u_2, \ldots, u_n\}$
R = a set of roles $\{r_1, r_2, \ldots, r_n\}$
P = a set of permissions $\{p_1, p_2, \ldots, p_n\}$

Definition 3.2: For RBAC associations
$UA \subseteq U \times R$, a many-to-many user-to-role assignment relation.

$PA \subseteq P \times R$, a many-to-many permission-to-role assignment relation.
$RH \subseteq R \times R$, a partial order on R called the role hierarchy.

Definition 3.3: For roles function
roles: $U \cup P \rightarrow 2^R$, a function mapping the sets U and P to a set of roles.
roles*: $U \cup P \rightarrow 2^R$, extends roles in the presence of a role hierarchy.
$roles(u_i) = \{r \in R \mid (u_i, r) \in UA\}$
$roles(p_i) = \{r \in R \mid (p_i, r) \in PA\}$
$roles(u_i)^* = \{r \in R \mid (\exists r \leq r')[(u_i, r') \in UA]\}$
$roles(p_i)^* = \{r \in R \mid (\exists r \leq r')[(p_i, r') \in PA]\}$

Note that the definition of roles* is carefully formulated to reflect the role inheritance with respect to users going downwards and with respect to permissions going upwards.

Definition 3.4: For permissions function
perm: $U \cup R \rightarrow 2^P$, a function mapping users and roles to a set of permissions.
perm*: $U \cup R \rightarrow 2^P$, extends permissions in the presence of a role hierarchy.
$perm(r_i) = \{p \in P \mid (p, r_i) \in PA\}$
$perm(u_i) = \{p \in P \mid (\exists r \in roles(u_i))\ [(p, r) \in PA]\}$
$perm(r_i)^* = \{p \in P \mid (\exists r' \leq r)[(p, r') \in PA]\}$
$perm(u_i)^* = \{p \in P \mid (\exists r \in roles^*(u_i))\ [(p, r) \in PA]\}$

Subsequently we introduce the RBAC extension.

4. Enhanced role-based access control (ERBAC03) Model

We extend the RBAC96 model by including three additional entities (see figure 2).

We consider that the locations consist of many roles. Each location is related to roles and reflects large enterprises which have various branches. Each branch consists of many roles that reflect the model management in distributed aspect. A role may perform more than one type of work. A role is responsible for all the activities that are required to perform the work. We define each type of work as a job. The jobs don't need any sequence and we show that the tasks requiring access to application will be mapped to the permissions granted the desired access.

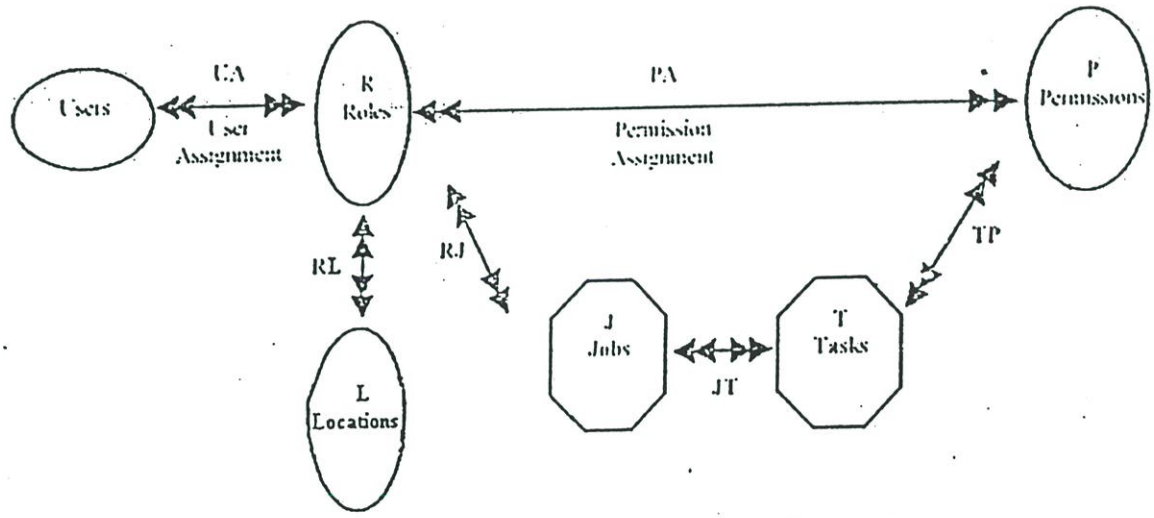


Figure 2. Enhanced role-based access control (ERBAC03) model

Definition 4.1: For additional entities
L = set of locations $\{l_1, l_2, \ldots, l_n\}$
J = set of jobs $\{j_1, j_2, \ldots, j_n\}$
T = set of tasks $\{t_1, t_2, \ldots, t_n\}$

Definition 4.2: For RBAC associations
$RL \subseteq L \times R$, a many-to-many location-to-role assignment relation.
$RJ \subseteq J \times R$, a many-to-many role-to-job assignment relation.
$JT \subseteq T \times J$, a many-to-many job-to-task assignment relation.
$TP \subseteq P \times T$, a many-to-many task-to-permission assignment relation.
$LH \subseteq L \times L$, a partial order on L called the location hierarchy.

Definition 4.3: For extend roles function
roles: $R \rightarrow 2^L$, a function mapping the sets R to a set of locations.
$roles(l_i) = \{r \in R \mid (l_i, r) \in RL\}$
$roles(j_i) = \{r \in R \mid (j_i, r) \in RJ\}$
$roles(l_i)^* = \{r \in R \mid (\exists r \leq r')[(l_i, r') \in RL]\}$
$roles(j_i)^* = \{r \in R \mid (\exists r \leq r')[(j_i, r') \in RJ]\}$

Definition 4.4: For locations function
location(r_i) = {l ∈ L | (l, r_i) ∈ RL}
location(u_i) = {l ∈ L | (∃r ∈ roles(u_i)) [(l, r) ∈ RL]}
location(j_i) = {l ∈ L | (∃r ∈ roles(j_i)) [(l, r) ∈ RL]}
location(p_i) = {l ∈ L | (∃r ∈ roles(p_i)) [(l, r) ∈ RL]}
location(r_i)* = {l ∈ L | (∃r' ≤ r)[(l, r') ∈ RL]}
location(u_i)* = {l ∈ L | (∃r ∈ roles*(u_i)) [(l, r) ∈ RL]}
location(j_i)* = {l ∈ L | (∃r ∈ roles*(j_i)) [(l, r) ∈ RL]}
location(p_i)* = {l ∈ L | (∃r ∈ roles*(p_i)) [(l, r) ∈ RL]}
locations: $U \cup R \cup P \rightarrow 2^L$, a function mapping users, roles and permissions to a set of locations.
locations*: $U \cup R \cup P \rightarrow 2^L$, extends locations in the presence of a location hierarchy.

Definition 4.5: For jobs function
jobs: $J \rightarrow 2^T$, a function mapping the sets J to a set of tasks.
jobs*: $J \rightarrow 2^T$, extends jobs in the presence of a role hierarchy or location hierarchy.
jobs(t_i) = {j ∈ J | (t_i, j) ∈ JT}

Definition 4.6: For tasks function
tasks: $T \rightarrow 2^P$, a function mapping the sets T to a set of permissions.
tasks*: $T \rightarrow 2^P$, extends tasks in the presence of a role hierarchy or location hierarchy.
tasks(p_i) = {t ∈ T | (p_i, t) ∈ TP}

Definition 4.7: For extend permissions function
perm(l_i) = {p ∈ P | (∃r ∈ roles(l_i)) [(p, r) ∈ PA]}
perm(j_i) = {p ∈ P | (∃t ∈ tasks(j_i)) [(p, t) ∈ TP]}
perm(l_i)* = {p ∈ P | (∃r ∈ roles*(l_i)) [(p, r) ∈ PA]}
perm(j_i)* = {p ∈ P | (∃t ∈ tasks*(j_i)) [(p, r) ∈ TP]}
perm: $U \cup R \cup L \cup J \cup T \rightarrow 2^P$, a function mapping users, roles, locations, jobs and tasks to a set of permissions.
perm*: $U \cup R \cup L \cup J \cup T \rightarrow 2^P$, extends permissions in the presence of a role hierarchy or location hierarchy.

4.1 The location hierarchy approach

The location hierarchy (LH) means for representing the organizational structure of the postal establishments (PE) involved in the application. A number of possible types of locations are shown in table 1.

Table 1. Types of locations

Locations	Type	Locations	Type
International Postal	I	District Postal	D
National Postal	N	Commercial Postal	C
Sector Postal	SP	Site	S

These location types have been used for the construction of LH (see figure 3).

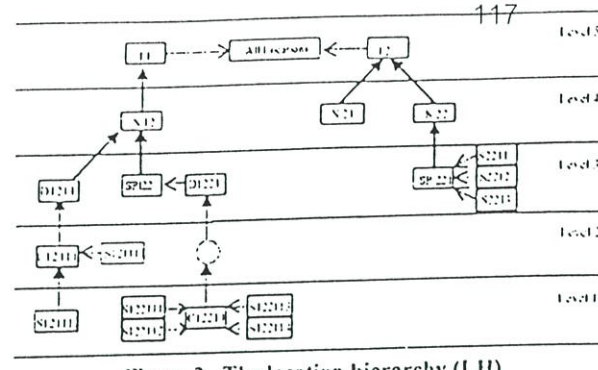


Figure 3. The location hierarchy (LH)

A location hierarchy results in the possession of international administrative domain at higher levels (e.g. level 5) and site as leaves. We initialise at level 5 (highest level) which is the sensitivity level and move administrative domain to the top of the hierarchy. A location hierarchy includes national or international administrative domains, as well as a number or all the possible workstations from where a given user can log in the distributed postal database system.

Definition 4.1.1:
Step 1: We begin by defining the initial administrative domain (i.e. Asia Pacific) and all its sub-domains.
Step 2: The LH previously defined in the ancestor domain is inherited by the sub-domain and the administrative sub-domains is contained in it.
Step 3: Any further administrative sub-domains that may be contained in it. If no further refinement is needed, this will be the last administrative domain level.

The concept of location hierarchy can apply to the role hierarchy in the same way.

4.2 Delegation

The scope of our model is to address the human to human delegation, whereby a user in a role is able to delegate his role membership to another user on the equal or lower security level in the same location. For the location delegation, each location can delegate some roles to another location on the the equal or lower security level.

Definition 4.2.1: The model controls user to user delegation by using the relation

$$\text{can-delegate-role} \subseteq R \times R$$

Definition 4.2.2: The model controls location to location delegation by using the relation

$$\text{can-delegate-location} \subseteq L \times 2^R$$

Definition 4.2.3: Both definitions must be considered on the security level carefully.

4.3 Revocation

We have described how users in a delegating role can delegate their permissions to others users in another roles. We use grant dependent revocation approach [7] to apply the ERBAC03 model.

Definition 4.3.1: The model controls user to user revocation by using the relation

$$\text{can-revoke-role} \subseteq R \times R$$

Definition 4.3.2: The model controls location to location revocation by using the relation

$$\text{can-revoke-location} \subseteq L \times 2^R$$

Definition 4.3.3: Only user delegated their permissions to other users can revoke their permissions.

4.4 Model example

We consider a design example (see table 2), for example, John has the role as chief of post office located in bangkok. He must be able to perform the jobs: 1) closing the end of day account, 2) counting the money, 3) checking the reports of the end of day account and 4) receiving the mails. To perform the first jobs of closing the end of day account, he needs to generate data into postal records and account records.

John is a user in set U, a chief of post office is a role in set R and bangkok is a location in set L. The chief of post office can perform four jobs: Job J_1 – closing the end of day account, Job J_2 – counting the money, Job J_3 – checking the reports of the end of day account, Job J_4 – receiving the mails.

Job J_1 consist of many tasks: Task T_1 is to generate data into postal records. Task T_2 is to generate data into account records.

Tasks T_1 requires a permission to generate data into postal database (P_1). Task T_2 requires a permission to generate data into account record (P_2).

The set of Roles = {R_i (chief of post office)}
The set of Locations = {L_1 (bangkok)}
There is a set of Jobs = {J_1, J_2, J_3, J_4}
There is a set of Tasks = {T_1, T_2}
There is a set of Permissions = {P_1, P_2}

Table 2. Chief of post office example

User	Role	Location	Job	Task	Permission
U_1	R_i	L_1	J_1	T_1, T_2	P_1, P_2

5. Conclusions

Access control mechanism in the enterprise environment has deep relationship in the real world. In this paper, we have introduced an enhanced model for large enterprises using location hierarchy approach and call ERBAC03 model, which is simple and easy to understand. The ERBAC03 model requires the modification of the end-points (i.e., roles and permissions). We have introduced three additional entities consisting of locations, jobs and tasks. We concentrate on the creation of the elements between the end-points such as jobs and tasks in order to make the permissions to roles more clear and able to apply to large enterprises effectively. The definition of entity location is related to roles and reflects large enterprises that have various branches. Each branch consists of many roles that reflect the model management in distributed aspect, not centralized aspect. It reduces workload for administrator and can use location hierarchy approach in order to define the security level for the location of enterprises to increase the efficiency of data security and data integrity. Finally, the definition of delegation and revocation is provided to improve the ERBAC03 model.

6. References

[1] I. Mavridis, G. Pangalos, M. Khair and L. Bozios, "Defining Access Control Mechanisms for Privacy Protection in Distributed Medical Databases," Proceedings of IFIP Working Conference on User Identification and Privacy Protection, Stockholm, Sweden, June 1999.

[2] P. Epstien and R. Sandhu, "Engineering of Role/Permission Assignments," 17th Annual Computer Security Applications Conference, December 2001.

[3] A. Kern, "Advanced Features for Enterprise-Wide Role-Based Access Control," 18th Annual Computer Security Applications Conference, December 2002.

[4] M.J. Moyer and M. Ahamad, "Generalized role-based access control," Proceedings 21st International Conference on Distributed Computing Systems, 2001, pp. 391-398.

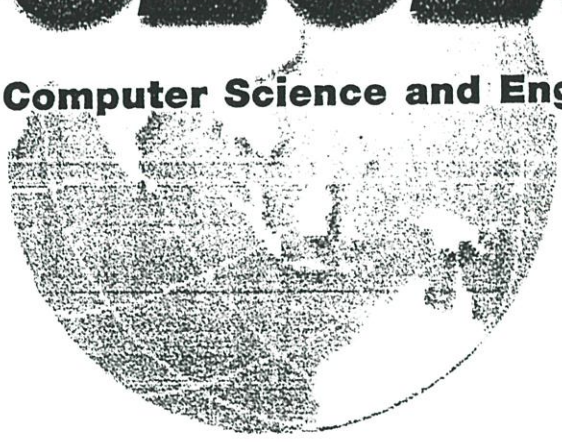
[5] M. Gasser and E. Mcdermott, "An Architecture for Practical Delegation in a Distributed System," 1990 IEEE Computer Society Symposium on Research in Security and Privacy, Oakland, CA, May 1990.

[6] R. Sandhu, E.J. Coyne, H.L. Feinstein and C.E. Youman, "Role Based Access Control Models," IEEE Computer, Vol. 2, February 1996, pp. 38-47.

[7] S. Osborn and R. Sandhu, "Configuring Role-based Access Control to Enforce Mandatory and Discretionary Access Control Policies," ACM Transaction on Information and System Security, Vol. 3, No. 2, May 2000, pp. 85-106.

NCSEC2003

The 7th National Computer Science and Engineering Conference



October 28-30, 2003

Chonburi, THAILAND

Organized by :

Department of Computer Science,

Faculty of Science,

Burapha University

ISBN : 974-382-604-1

An Enhanced Role-Based Access Control Model using Constraint Features

Burin Yenmunkong
Faculty of Information Technology,
King Mongkut's Institute of Technology Ladkrabang,
Ladkrabang, Bangkok 10520, Thailand
Email: bom_burin@hotmail.com

Chanboon Sathitwiriyawong
Faculty of Information Technology, and Research Center
for Communication and Information Technology,
King Mongkut's Institute of Technology Ladkrabang,
Ladkrabang, Bangkok 10520, Thailand
Email: chanboon@it.kmitl.ac.th

Abstract

In this paper we suggest an extended model of RBAC for large enterprises called "Enhanced Role-Based Access Control (ERBAC03)", including the assignment of permissions in order to appropriate roles into appropriate locations. The ERBAC03 model is improved for integrity requirement by using constraint features. This model deals with specified static separation of duty constraint and proposes as a mathematical model based on the concept of "Conflicting entities" to express static separation of duty requirements. It provides a detailed explanation of the integrity checking taken at administration time to ensure that specified separation of duty requirements are practical for efficiency of data integrity.

Key-Words: Role-based access control (RBAC), constraint, distributed environment, static separation of duty, data integrity, conflict of interest

1. Introduction

Role-based access control has recently received considerable attention as a promising alternative to traditional discretionary and mandatory access controls [1]. In RBAC96 model [2], permissions associated with roles and users are members of appropriate roles thereby acquiring the roles permissions. This simplifies management of permission.

In large enterprises, the number of roles is created for the various job functions and is able to double to hundreds or thousands. Managing these roles, permissions, users and their interrelationships is a formidable task that is often centralized in a small team of security administrator. Each enterprise has many branches, so we must improve efficiency of RBAC96 model for enterprises in the real world. The RBAC96 model was viewed as a distributed model, not a centralized model.

A core aspect of RBAC is the Role/Permission Relation. In addition, application developers have accepted the definition of RBAC permissions. However, other researches have not given details on a systematic model for determining the assignment of permission to roles in a distributed environment [3]. This paper focuses on a model for role to permission assignment based on location authorization control [4][5]. The key concept is to identify some approaches possible for RBAC96 in distributed systems and to present the constraint features for the new model referring as static separation of duty (SSoD) [6][7]. The user does not have fully authorization to perform activity. The objectives of this research are to improve RBAC96 model for efficiency of data integrity, to prevent fraud, to increase efficiency of work in enterprises and to manage easily for RBAC administrator.

The rest of the paper is organized as follows. In Section 2, the definition of RBAC96 model is explained. In Section 3, we formally define a new model extended from traditional RBAC96 model for enhanced efficiency. In Section 4, constraint features are also added for ERBAC03. In Section 5, we present a number of possible theorem reflecting

integrity requirement and the conclusions are shown in Section 6.

2. Role-Based Access Control Model

The developed model in this paper is constructed by extending the RBAC96 model [2]. The RBAC96 model comprises of four models: $RBAC_0$, $RBAC_1$, $RBAC_2$ and $RBAC_3$. $RBAC_0$ is the base model. $RBAC_1$ and $RBAC_2$ add role hierarchies and constraints, respectively. $RBAC_3$ is a consolidated model. RBAC96 also makes a distinction between user and administrative roles.

There are three components of RBAC96 that we are inherited in using for the extension of the model: users (U), roles (R) and permissions (P).

The model, as depicted in figure 1, defined the components of role/permission assignment by PA. It also defined the role hierarchy (RH) and the component of user/role assignment by UA.

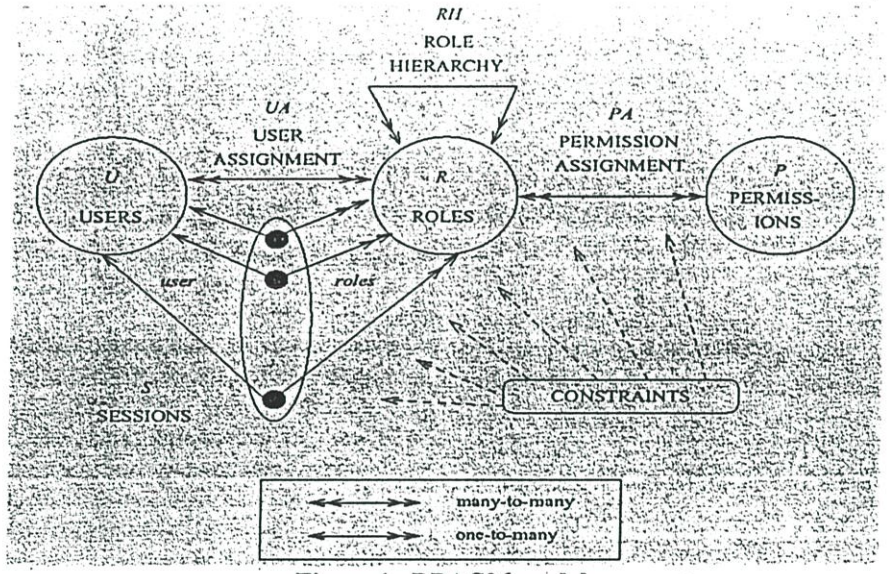


Figure 1. RBAC96 model

Consider a short formal summary of the relevant components in the RBAC96 model.

Definition 2.1: For RBAC96 entities
U = a set of users $\{u_1, u_2, \ldots, u_n\}$
R = a set of roles $\{r_1, r_2, \ldots, r_n\}$
P = a set of permissions $\{p_1, p_2, \ldots, p_n\}$

Definition 2.2: For RBAC associations
$UA \subseteq U \times R$, a many-to-many user-to-role assignment relation.
$PA \subseteq P \times R$, a many-to-many permission-to-role assignment relation.
$RH \subseteq R \times R$, a partial order on R called the role hierarchy.

Definition 2.3: For roles function
roles: $U \cup P \rightarrow 2^R$, a function mapping the sets U and P to a set of roles.
roles*: $U \cup P \rightarrow 2^R$, extends roles in the presence of a role hierarchy.
$roles(u_i) = \{r \in R \mid (u_i, r) \in UA\}$
$roles(p_i) = \{r \in R \mid (p_i, r) \in PA\}$
$roles(u_i)^* = \{r \in R \mid (\exists r \leq r')[(u_i, r') \in UA]\}$
$roles(p_i)^* = \{r \in R \mid (\exists r \leq r')[(p_i, r') \in PA]\}$

Note that the definition of roles* is carefully formulated to reflect the role inheritance with respect to users going downwards and with respect to permissions going upwards.

Definition 2.4: For permissions function
perm: $U \cup R \rightarrow 2^P$, a function mapping users and roles to a set of permissions.
perm*: $U \cup R \rightarrow 2^P$, extends permissions in the presence of a role hierarchy.
$perm(r_i) = \{p \in P \mid (p, r_i) \in PA\}$
$perm(u_i) = \{p \in P \mid (\exists r \in roles(u_i))\ [(p, r) \in PA]\}$
$perm(r_i)^* = \{p \in P \mid (\exists r' \leq r)[(p, r') \in PA]\}$
$perm(u_i)^* = \{p \in P \mid (\exists r \in roles^*(u_i))\ [(p, r) \in PA]\}$

Subsequently we introduce the RBAC extension.

3. Enhanced Role-Based Access Control (ERBAC03) Model

In the previous research [4], we extended the RBAC96 model by defining three additional entities (see figure 2). The concept of new model was influenced by the ideas in [8].

The ERBAC03 model requires the modification of the end-points (i.e., roles and permissions) by

introducing three additional entities which consist of locations, jobs and tasks. We concentrate on the creation of the elements between the end-points such as jobs and tasks in order to make the permissions to roles more clear, also to apply to large enterprises effectively.

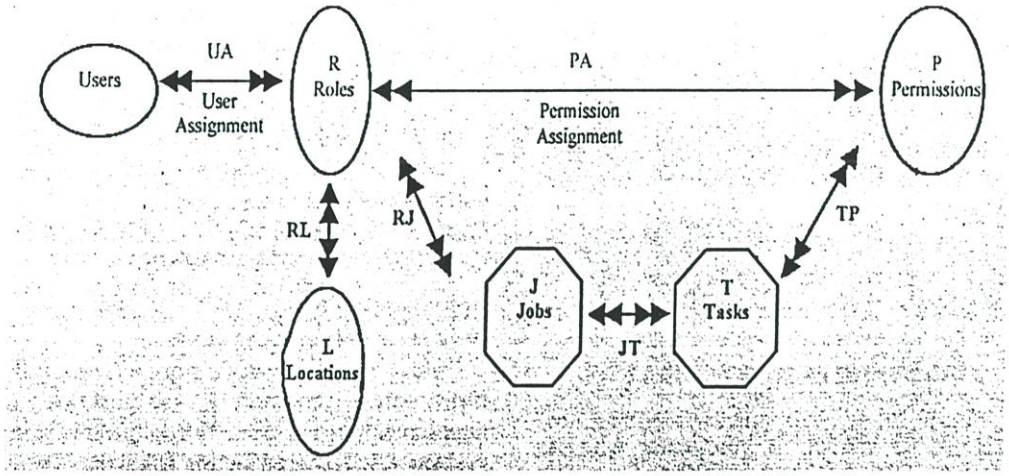


Figure 2. Enhanced role-based access control (ERBAC03) model

We consider that the locations consist of many roles. Each location is related to roles and reflects large enterprises which have various branches. Each branch consists of many roles that reflect the model management in distributed aspect. A role may perform more than one type of work. A role is responsible for all the activities that are required to perform the work. We define each type of work as a job. The jobs need not be in any sequence and we show that the tasks requiring access to application will be mapped to the permissions granted the desired access.

Consider a short formal summary of the relevant components in the extension of RBAC96 model.

Definition 3.1: For additional entities
L = set of locations $\{l_1, l_2, \ldots, l_n\}$
J = set of jobs $\{j_1, j_2, \ldots, j_n\}$
T = set of tasks $\{t_1, t_2, \ldots, t_n\}$

Definition 3.2: For RBAC associations
$RL \subseteq L \times R$, a many-to-many location-to-role assignment relation.
$RJ \subseteq J \times R$, a many-to-many role-to-job assignment relation.
$JT \subseteq T \times J$, a many-to-many job-to-task assignment relation.
$TP \subseteq P \times T$, a many-to-many task-to-permission assignment relation.
$LH \subseteq L \times L$, a partial order on L called the location hierarchy.

Definition 3.3: For extend roles function
roles: $R \rightarrow 2^L$, a function mapping the sets R to a set of locations.

$roles(l_i) = \{r \in R \mid (l_i, r) \in RL\}$
$roles(j_i) = \{r \in R \mid (j_i, r) \in RJ\}$
$roles(l_i)^* = \{r \in R \mid (\exists r \leq r')[(l_i, r') \in RL]\}$
$roles(j_i)^* = \{r \in R \mid (\exists r \leq r')[(j_i, r') \in RJ]\}$

Definition 3.4: For locations function
$location(r_i) = \{l \in L \mid (l, r_i) \in RL\}$
$location(u_i) = \{l \in L \mid (\exists r \in roles(u_i))\ [(l, r) \in RL]\}$
$location(j_i) = \{l \in L \mid (\exists r \in roles(j_i))\ [(l, r) \in RL]\}$
$location(p_i) = \{l \in L \mid (\exists r \in roles(p_i))\ [(l, r) \in RL]\}$
$location(r_i)^* = \{l \in L \mid (\exists r' \leq r)[(l, r') \in RL]\}$
$location(u_i)^* = \{l \in L \mid (\exists r \in roles^*(u_i))\ [(l, r) \in RL]\}$
$location(j_i)^* = \{l \in L \mid (\exists r \in roles^*(j_i))\ [(l, r) \in RL]\}$
$location(p_i)^* = \{l \in L \mid (\exists r \in roles^*(p_i))\ [(l, r) \in RL]\}$
locations: $U \cup R \cup P \rightarrow 2^L$, a function mapping users, roles and permissions to a set of locations.
locations*: $U \cup R \cup P \rightarrow 2^L$, extends locations in the presence of a location hierarchy.

Definition 3.5: For jobs function
jobs: $J \rightarrow 2^T$, a function mapping the sets J to a set of tasks.
jobs*: $J \rightarrow 2^T$, extends jobs in the presence of a role hierarchy or location hierarchy.
$jobs(t_i) = \{j \in J \mid (t_i, j) \in JT\}$

Definition 3.6: For tasks function
tasks: $T \rightarrow 2^P$, a function mapping the sets T to a set of permissions.

tasks*: $T \rightarrow 2^P$, extends tasks in the presence of a role hierarchy or location hierarchy.
$tasks(p_i) = \{t \in T \mid (p_i, t) \in TP\}$

Definition 3.7: For extend permissions function
$perm(l_i) = \{p \in P \mid (\exists r \in roles(l_i))\ [(p, r) \in PA]\}$
$perm(j_i) = \{p \in P \mid (\exists t \in tasks(j_i))\ [(p, t) \in TP]\}$
$perm(l_i)^* = \{p \in P \mid (\exists r \in roles^*(l_i))\ [(p, r) \in PA]\}$
$perm(j_i)^* = \{p \in P \mid (\exists t \in tasks^*(j_i))\ [(p, r) \in TP]\}$
perm: $U \cup R \cup L \cup J \cup T \rightarrow 2^P$, a function mapping users, roles, locations, jobs and tasks to a set of permissions.
perm*: $U \cup R \cup L \cup J \cup T \rightarrow 2^P$, extends permissions in the presence of a role hierarchy or location hierarchy.

4. Constraint Features with ERBAC03

Static separation of duty (SSoD) constraint is used in ERBAC03 model. It concerned with the prevention of fraud to ensure that a single user does not have too much authority. The authority is set as a permission based on location, therefore the essence of our paradigm depends on the conflicting permissions and locations.

Definition 4.1 Conflicting permissions are permissions that can result in unnessary power if bestowed on the same person. Formally it is represented by $CP \subseteq P \times P$, a many-to-many relation indicating conflict between permissions with

$(p_i, p_j) \in CP \leftrightarrow (p_j, p_i) \in CP, (p_i, p_i) \notin CP$

We can now present the following axiom which will represent our basic safety condition.

Basic Safety Condition 1: Conflicting permissions may not be assigned to a user or jobs.

Formally, $(perm^*(u_i) \times perm^*(u_i)) \cap CP = \varnothing$ and
$(perm^*(t_i) \times perm^*(t_j)) \cap CP = \varnothing$

Since non-conflicting permissions cannot influence the basic safety condition 1, the following axiom to supplement the basic safety condition is formulated.

Axiom 4.1: Non-conflicting permissions may be assigned to both conflicting or non-conflicting roles or both conflicting or non-conflicting jobs.

Definition 4.2 Conflicting users are users who are likely to conspire. Formally they are represented by

$CU \subseteq U \times U$, a many-to-many relation indicating conflict between users with

$(u_i, u_j) \in CU \leftrightarrow (u_j, u_i) \in CU, (u_i, u_i) \notin CU$

Axiom 4.2: Conflicting users are considered as a single user.

In practical terms conflicting users may be family members or people who are known to have conspired.

Definition 4.3 Conflicting roles are roles that together have the ability to conspire, i.e. they are assigned some (but not all) conflicting permissions. They are represented by $CR \subseteq R \times R$, a many-to-many relation indicating conflict between roles with

$(r_i, r_j) \in CR \leftrightarrow (r_j, r_i) \in CR, (r_i, r_i) \notin CR$ and
$\{(r_i, r_j) \in CR \rightarrow (perm^*(r_i) \times perm^*(r_j)) \cap CP \neq \varnothing$ or
$(r_i, r_j) \in CR \rightarrow (job^*(r_i) \times job^*(r_j)) \cap CJ \neq \varnothing$ or (r_i, r_j)
$\in CR \rightarrow (location^*(r_i) \times location^*(r_j)) \cap CL \neq \varnothing\}$

We can now present the following axiom which will represent our basic safety condition.

Basic Safety Condition 2: Conflicting roles may not be assigned to a user or locations.

$(role^*(u_i) \times role^*(u_i)) \cap CR = \varnothing$ and
$(role^*(l_i) \times role^*(l_j)) \cap CR = \varnothing$

Since non-conflicting roles cannot influence the basic safety condition 2, the following axiom to supplement the basic safety condition is formulated.

Axiom 4.3: Non-conflicting roles may be assigned to both conflicting or non-conflicting users or both conflicting or non-conflicting locations.

Since conflicting roles must have some conflicting permissions or jobs, we can state that non-conflicting roles do not have conflicting permissions or jobs.

Definition 4.4 Conflicting jobs are jobs that together have the ability to fraud. They are represented by

$CJ \subseteq J \times J$, a many-to-many relation indicating conflict between jobs with

$(j_i, j_j) \in CJ \leftrightarrow (j_j, j_i) \in CJ, (j_i, j_i) \notin CJ$ and
$\{(j_i, j_j) \in CJ \rightarrow (task^*(j_i) \times task^*(j_j)) \cap CT \neq \varnothing$ or
$(j_i, j_j) \in CJ \rightarrow (role^*(j_i) \times role^*(j_j)) \cap CR \neq \varnothing\}$

We can now present the following axiom which will represent our basic safety condition.

Basic Safety Condition 3: Conflicting jobs may not be assigned to roles.

$(job^*(r_i) \times job^*(r_j)) \cap CJ = \varnothing$

Since non-conflicting jobs cannot influence the basic safety condition 3, the following axiom to supplement the basic safety condition is formulated.

Axiom 4.4: Non-conflicting jobs may be assigned to both conflicting or non-conflicting roles.

Definition 4.5 Conflicting tasks are tasks requiring conflicting permissions to complete. Formally they are represented by $CT \subseteq T \times T$, a many-to-many relation indicating conflict between tasks with

$(t_i, t_j) \in CT \leftrightarrow (t_j, t_i) \in CT, (t_i, t_i) \notin CT$ and
$(t_i, t_j) \in CT \rightarrow (perm^*(t_i) \times perm^*(t_j)) \cap CP \neq \varnothing$

We can now present the following axiom which will represent our basic safety condition.

Basic Safety Condition 4: Conflicting tasks may not be assigned to jobs.

$(task^*(j_i) \times task^*(j_j)) \cap CT = \varnothing$

Since non-conflicting tasks cannot influence the basic safety condition 4, the following axiom to supplement the basic safety condition is formulated.

Axiom 4.5: Non-conflicting tasks may be assigned to both conflicting or non-conflicting jobs.

Definition 4.6 Conflicting locations are locations requiring conflicting permissions to complete. Formally they are represented by $CL \subseteq L \times L$, a many-to-many relation indicating conflict between locations with

$(l_i, l_j) \in CL \leftrightarrow (l_j, l_i) \in CL, (l_i, l_i) \notin CL$ and
$(l_i, l_j) \in CL \rightarrow (role^*(l_i) \times role^*(l_j)) \cap CR \neq \varnothing$

We can now present the following axiom which will represent our basic safety condition.

Basic Safety Condition 5: Conflicting locations may not be assigned to roles and a user. The location only have non-conflicting roles assigned to them.

$(location^*(r_i) \times location^*(r_j)) \cap CL = \varnothing$ and
$(location^*(u_i) \times location^*(u_i)) \cap CL = \varnothing$

Since non-conflicting locations cannot influence the basic safety condition 5, the following axiom to supplement the basic safety condition is formulated.

Axiom 4.6: Non-conflicting locations may be assigned to both conflicting or non-conflicting roles or both conflicting or non-conflicting user.

These principles and definitions are essentially focused on the permissions exercised by the users including the location of work.

The next topic will therefore show the integrity requirements pertaining to the associations allowed in the ERBAC03 model.

5. Integrity Requirements

In this section, we present a number of possible theorem reflecting integrity requirement that will have to be upheld in a security administration tool.

Theorem 5.1: Under the basic safety condition 5, conflicting roles may only have non-conflicting users assigned to them, i.e.

$(u_i, r_k) \in UA \wedge (u_j, r_l) \in UA \wedge (r_k, r_l) \in CR \rightarrow$
$(u_i, u_j) \notin CU$

Proof:

Assume that $(u_i, r_k) \in UA \wedge (u_j, r_l) \in UA \wedge$
$(r_k, r_l) \in CR \wedge (u_i, u_j) \in CU$

$location^*(u_i) \supseteq location^*(r_k)$ (Def. 3.4)
$location^*(u_j) \supseteq location^*(r_l)$ (Def. 3.4)

$(r_k, r_l) \in CR$

$\Rightarrow (location^*(r_i) \times location^*(r_j)) \cap CL \neq \varnothing$ (Def. 4.3)
$\Rightarrow (location^*(u_i) \times location^*(u_j)) \cap CL \neq \varnothing$

which contradicts the Basic Safety Condition. *Q.E.D.*

Theorem 5.2: Under the basic safety condition 5, conflicting roles may only have conflicting jobs assigned to them, i.e.

$(j_i, r_k) \in RJ \wedge (j_j, r_l) \in RJ \wedge (r_k, r_l) \in CR \rightarrow$
$(j_i, j_j) \in CJ$

Proof:

Assume that $(j_i, r_k) \in RJ \wedge (j_j, r_l) \in RJ \wedge$
$(r_k, r_l) \in CR \wedge (j_i, j_j) \notin CJ$

$location^*(r_k) \supseteq location^*(j_i)$ (Def. 3.4)
$location^*(r_l) \supseteq location^*(j_j)$ (Def. 3.4)

$(r_k, r_l) \in CR$

$\Rightarrow (job^*(r_i) \times job^*(r_j)) \cap CJ \neq \varnothing$ (Def. 4.3)
$\Rightarrow (location^*(r_k) \times location^*(r_l)) \cap CL \neq \varnothing$

which contradicts the Basic Safety Condition. *Q.E.D.*

Theorem 5.3: Under the basic safety condition 5, conflicting jobs may only have conflicting tasks assigned to them, i.e.

$(t_i, j_k) \in JT \wedge (t_j, j_l) \in JT \wedge (t_i, t_j) \in CT \rightarrow (j_k, j_l) \in CJ$

Proof:

Assume that two conflicting tasks t_i and t_j are assigned to non-conflicting jobs.

$(t_i, j_k) \in JT \wedge (t_j, j_l) \in JT \wedge (t_i, t_j) \in CT \wedge (j_k, j_l) \notin CJ$

Choose a role r_x and associate it with jobs j_k and j_l.

Since $(j_k, j_l) \notin CJ$

Then $(r_x, j_k) \in RJ \wedge (r_x, j_l) \in RJ \wedge (t_i, j_k) \in JT \wedge$
$(t_j, j_l) \in JT$

Since $(t_i, t_j) \in CT$, the jobs are also conflicted.

$\Rightarrow (task^*(j_k) \times task^*(j_l)) \cap CT \neq \varnothing$ (Def. 4.4)
$\Rightarrow \{j_k, j_l\} \subseteq location^*(r_x)$ (Def. 3.4)

But $(j_k, j_l) \in CJ$, which contradicts the Basic Safety Condition. *Q.E.D.*

Theorem 5.4: Under the basic safety condition 5, conflicting permissions may only to be assigned to conflicting roles, i.e.

$(p_i, r_k) \in PA \wedge (p_j, r_l) \in PA \wedge (p_i, p_j) \in CP \rightarrow$
$(r_k, r_l) \in CR$

Proof:

Assume that $(p_i, r_k) \in PA \wedge (p_j, r_l) \in PA \wedge$
$(p_i, p_j) \in CP \wedge (r_k, r_l) \notin CR$

$location^*(p_i) \subseteq location^*(r_k)$ (Def. 3.4)
$location^*(p_j) \subseteq location^*(r_l)$ (Def. 3.4)

But $(p_i, p_j) \in CP$, which contradicts the Basic Safety Condition. *Q.E.D.*

Theorem 5.5: Under the basic safety condition 5, conflicting roles may only to be assigned to conflicting locations, i.e.

$(r_i, l_k) \in RL \wedge (r_j, l_l) \in RL \wedge (r_i, r_j) \in CR \rightarrow$
$(l_k, l_l) \in CL$

Proof:

Assume that $(r_i, l_k) \in RL \wedge (r_j, l_l) \in RL \wedge (r_i, r_j) \in CR \wedge (l_k, l_l) \notin CL$

When $(r_i, r_j) \in CR$

$\Rightarrow (location^*(r_i) \times location^*(r_j)) \cap CL \neq \varnothing$ (Def. 4.3)

$\Rightarrow (role^*(l_i) \times role^*(l_j)) \cap CR \neq \varnothing$ (Def. 4.6)

therefore $(l_k, l_l) \in CL$

which contradicts the Basic Safety Condition. *Q.E.D.*

6. Conclusions

In this paper, we have improved an enhanced model for large enterprises called ERBAC03 model. The ERBAC03 model requires the modification of the end-points by introducing three additional entities which consist of locations, jobs and tasks and are able to make the permissions to roles more clear, also to apply to large enterprises effectively. The definition of entity location is related to roles and reflects large enterprises which have various branches. Each branch consists of many roles that reflect the model management in distributed aspect, not centralized aspect. The constraint features used in this model are referred as static separation of duty, and able to deal with the ERBAC03 components. In particular, it demonstrated how static separation of duty requirements specified through the use of conflicting permissions, users, roles, jobs, tasks and locations. The definition of conflicting entities has been already proven for integrity requirements.

Table 1. The integrity of the associations allowed between components

May be associated with		Roles	
		Conflicting	Non-conflicting
Users	Conflicting	×	✓
		Th 5.1	Ax 4.3
	Non-conflicting	✓	✓
		Th 5.1	Ax 4.3
Permis-sions	Conflicting	✓	×
		Theorem 5.4	
	Non-conflicting	✓	✓
		Axiom 4.1	
Jobs	Conflicting	✓	×
		Theorem 5.2	
	Non-conflicting	✓	✓
		Axioms 4.4	
Tasks	Conflicting	×	×
		No Theorem	
	Non-conflicting	×	×
		No Axioms	

Table 1. The integrity of the associations allowed between components (continued)

May be associated with		Roles	
		Conflicting	Non-conflicting
Locations	Conflicting	✓	×
		Theorem 5.5	
	Non-conflicting	✓	✓
		Axioms 4.6	

A symbol "✓" in the table 1 indicates that an association is allowed while a symbol "×" shows that an association is prohibited.

It results in the reduction of workload for administrator, the higher efficiency of data integrity, prevention fraud, the higher efficiency of work in enterprises and easier management for RBAC administrator.

7. References

[1] S. Osborn and R. Sandhu, "Configuring Role-based Access Control to Enforce Mandatory and Discretionary Access Control Policies," ACM Transaction on Information and System Security, Vol. 3, No. 2, May 2000, pp. 85-106.

[2] R. Sandhu, E.J. Coyne, H.L. Feinstein and C.E. Youman, "Role Based Access Control Models," IEEE Computer, Vol. 2, February 1996, pp. 38-47.

[3] R.J. Hayton, J.M. Bacon and K. Moody, "OASIS: Access Control in an Open Distributed Environment," Proceedings of the IEEE Symposium on Security and Privacy, Oakland, CA, May 1998, pp. 3-14.

[4] C. Sathitwiriyawong and B. Yenmunkong, "An Enhanced Role-Based Access Control for Large Enterprises," Proceeding of the 3rd International Symposium on Communications and Information Technologies, September 2003, pp. 279-283.

[5] I. Mavridis, G. Pangalos, M. Khair and L. Bozios, "Defining Access Control Mechanisms for Privacy Protection in Distributed Medical Databases," Proceedings of IFIP Working Conference on User Identification and Privacy Protection, Stockholm, Sweden, June 1999.

[6] R. A. Botha and J. H. P. Eloff, "Separation of Duties for Access Control Enforcement in Workflow Environments," IBM Systems Journal, Vol. 40, No. 3, 2001.

[7] S. Perelson and R. A. Botha, "Conflict Analysis as a Means of Enforcing Static Separation of Duty Requirements in Workflow Environments," South African Computer Journal, Vol. 26, November 2000, pp. 212-216.

[8] P. Epstien and R. Sandhu, "Engineering of Role/Permission Assignments," 17th Annual Computer Security Applications Conference, December 2001.

ภาคผนวก ข.

ตัวอย่างโปรแกรมทริกเกอร์

เป็นโปรแกรมทั้งหมดที่ใช้ในการทดลองรูปแบบ ERBAC03 ตามอัลกอริทึมที่ออกแบบไว้

```
-------------------------------------------------------------------------------
-- ERBAC03 integrity constraint rules
-- By Burin Yenmunkong
-- Copyright  2004, Burin Yenmunkong
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
--
-- ERBAC03 Prototype Database Tables
--
-------------------------------------------------------------------------------
--Drop all constraints
-- FK's for ua
ALTER TABLE ua
DROP CONSTRAINT fk_userroleid;
ALTER TABLE ua
DROP CONSTRAINT fk_useruserid;
-- FK's for pa
ALTER TABLE pa
DROP CONSTRAINT fk_permroleid;
ALTER TABLE pa
DROP CONSTRAINT fk_permpermid;
-- FK's for tp
ALTER TABLE tp
DROP CONSTRAINT fk_taskpermid;
ALTER TABLE tp
DROP CONSTRAINT fk_tasktaskid;
-- FK's for cu
ALTER TABLE cu
DROP CONSTRAINT fk_user1userid;
ALTER TABLE cu
DROP CONSTRAINT fk_user2userid;
-- FK's for cr
ALTER TABLE cr
DROP CONSTRAINT fk_role1roleid;
ALTER TABLE cr
DROP CONSTRAINT fk_role2roleid;
-- FK's for cp
ALTER TABLE cp
DROP CONSTRAINT fk_perm1permid;
ALTER TABLE cp
DROP CONSTRAINT fk_perm2permid;
-- FK's for ct
ALTER TABLE ct
DROP CONSTRAINT fk_task1taskid;
ALTER TABLE ct
DROP CONSTRAINT fk_task2taskid;
-- FK's for rh
ALTER TABLE rh
DROP CONSTRAINT fk_rolenetrolenetid;
ALTER TABLE rh
DROP CONSTRAINT fk_parentroleid;
ALTER TABLE rh
DROP CONSTRAINT fk_childroleid;

Drop Table tasks;
Drop Table roles;
Drop Table users;
Drop Table perms;
Drop Table rolenet;
Drop Table locationnet;
Drop Table rh;
Drop Table lh;
Drop Table ct;
Drop Table cu;
Drop Table cr;
Drop Table cp;
Drop Table ct;
Drop Table cj;
Drop Table cl;
Drop Table ua;
Drop Table pa;
Drop Table tp;
Drop Table jt;
Drop Table rj;
Drop Table rl;
--End of Dropping ERBAC03Tables

CREATE TABLE tp
( permid Number(38),
taskid Number(38),
CONSTRAINT pk_tp PRIMARY KEY (permid, taskid)
);
CREATE TABLE pa
( roleid Number(38),
permid Number(38),
CONSTRAINT pk_pa PRIMARY KEY (roleid, permid)
);
```

```
CREATE TABLE ua
( roleid Number(38),
userid Number(38),
CONSTRAINT pk_ua PRIMARY KEY (roleid, userid)
);
CREATE TABLE rl
( locationid Number(38),
roleid Number(38),
CONSTRAINT pk_rl PRIMARY KEY (locationid, roleid)
);
CREATE TABLE rj
( jobid Number(38),
roleid Number(38),
CONSTRAINT pk_rj PRIMARY KEY (jobid, roleid)
);
CREATE TABLE jt
( taskid Number(38),
jobid Number(38),
CONSTRAINT pk_jt PRIMARY KEY (taskid, jobid)
);
CREATE TABLE ct
( taskid1 Number(38),
taskid2 Number(38),
CONSTRAINT pk_ct PRIMARY KEY (taskid1, taskid2)
);
CREATE TABLE cp
( permid1 Number(38),
permid2 Number(38),
CONSTRAINT pk_cp PRIMARY KEY (permid1, permid2)
);
CREATE TABLE cu
( userid1 Number(38),
userid2 Number(38),
CONSTRAINT pk_cu PRIMARY KEY (userid1, userid2)
);
CREATE TABLE cr
( roleid1 Number(38),
roleid2 Number(38),
CONSTRAINT pk_cr PRIMARY KEY (roleid1, roleid2)
);
CREATE TABLE cj
( jobid1 Number(38),
jobid2 Number(38),
CONSTRAINT pk_cj PRIMARY KEY (jobid1, jobid2)
);
CREATE TABLE cl
( locationid1 Number(38),
locationid2 Number(38),
CONSTRAINT pk_cl PRIMARY KEY (locationid1, locationid2)
);
CREATE TABLE rh
( rolenetid Number(38) NOT NULL,
parentid Number(38),
childid Number(38) NOT NULL,
CONSTRAINT pk_rh UNIQUE (rolenetid, parentid, childid)
);
CREATE TABLE rolenet
( rolenetid Number(38),
description Varchar2(128),
CONSTRAINT pk_rolenet PRIMARY KEY (rolenetid)
);
CREATE TABLE lh
( locationnetid Number(38) NOT NULL,
lparentid Number(38),
lchildid Number(38) NOT NULL,
CONSTRAINT pk_lh UNIQUE (locationnetid, lparentid, lchildid)
);
CREATE TABLE locationnet
( locationnetid Number(38),
description Varchar2(128),
CONSTRAINT pk_locationnet PRIMARY KEY (locationnetid)
);
CREATE TABLE perms
( permid Number(38),
pname Varchar2(128),
CONSTRAINT pk_perms PRIMARY KEY (permid)
);
CREATE TABLE users
( userid Number(38),
uname Varchar2(128),
CONSTRAINT pk_users PRIMARY KEY (userid)
);
--alter table users add(upassword varchar2(15));
CREATE TABLE roles
( roleid Number(38),
rname Varchar2(128),
CONSTRAINT pk_roles PRIMARY KEY (roleid)
```

```
);
CREATE TABLE locations
( locationid Number(38),
lname Varchar2(128),
CONSTRAINT pk_locations PRIMARY KEY (locationid)
);
CREATE TABLE jobs
( jobid Number(38),
jname Varchar2(128),
CONSTRAINT pk_jobs PRIMARY KEY (jobid)
);
CREATE TABLE tasks
( taskid Number(38),
tname Varchar2(128),
CONSTRAINT pk_tasks PRIMARY KEY (taskid)
);
-- End of create tables
--Beginning of Foreign Key constraints
-- FK's for ua
ALTER TABLE ua
ADD CONSTRAINT fk_userroleid FOREIGN KEY (roleid)
REFERENCES roles(roleid);
ALTER TABLE ua
ADD CONSTRAINT fk_useruserid FOREIGN KEY (userid)
REFERENCES users(userid);
-- FK's for pa
ALTER TABLE pa
ADD CONSTRAINT fk_permroleid FOREIGN KEY (roleid)
REFERENCES roles(roleid);
ALTER TABLE pa
ADD CONSTRAINT fk_permpermid FOREIGN KEY (permid)
REFERENCES perms(permid);
-- FK's for rl
ALTER TABLE rl
ADD CONSTRAINT fk_locationroleid FOREIGN KEY (roleid)
REFERENCES roles(roleid);
ALTER TABLE rl
ADD CONSTRAINT fk_locationlocationid FOREIGN KEY (locationid)
REFERENCES locations(locationid);
-- FK's for rj
ALTER TABLE rj
ADD CONSTRAINT fk_rolejobid FOREIGN KEY (roleid)
REFERENCES roles(roleid);
ALTER TABLE rj
ADD CONSTRAINT fk_jobjobid FOREIGN KEY (jobid)
REFERENCES jobs(jobid);
-- FK's for jt
ALTER TABLE jt
ADD CONSTRAINT fk_jobtaskid FOREIGN KEY (jobid)
REFERENCES roles(roleid);
ALTER TABLE jt
ADD CONSTRAINT fk_tasktaskid FOREIGN KEY (taskid)
REFERENCES tasks(taskid);
-- FK's for tp
ALTER TABLE tp
ADD CONSTRAINT fk_taskpermid FOREIGN KEY (taskid)
REFERENCES perms(permid);
ALTER TABLE tp
ADD CONSTRAINT fk_permtaskid FOREIGN KEY (permid)
REFERENCES tasks(taskid);
-- FK's for cu
ALTER TABLE cu
ADD CONSTRAINT fk_user1userid FOREIGN KEY (userid1)
REFERENCES users(userid)
ON DELETE CASCADE;
ALTER TABLE cu
ADD CONSTRAINT fk_user2userid FOREIGN KEY (userid2)
REFERENCES users(userid)
ON DELETE CASCADE;
-- FK's for cj
ALTER TABLE cj
ADD CONSTRAINT fk_job1jobid FOREIGN KEY (jobid1)
REFERENCES jobs(jobid)
ON DELETE CASCADE;
ALTER TABLE cj
ADD CONSTRAINT fk_job2jobid FOREIGN KEY (jobid2)
REFERENCES jobs(jobid)
ON DELETE CASCADE;
-- FK's for cl
ALTER TABLE cl
ADD CONSTRAINT fk_location1locationid FOREIGN KEY (locationid1)
REFERENCES locations(locationid)
ON DELETE CASCADE;
ALTER TABLE cl
ADD CONSTRAINT fk_location2locationid FOREIGN KEY (locationid2)
REFERENCES locations(locationid)
ON DELETE CASCADE;
```

```
-- FK's for cr
ALTER TABLE cr
ADD CONSTRAINT fk_role1roleid FOREIGN KEY (roleid1)
REFERENCES roles(roleid)
ON DELETE CASCADE;
ALTER TABLE cr
ADD CONSTRAINT fk_role2roleid FOREIGN KEY (roleid2)
REFERENCES roles(roleid)
ON DELETE CASCADE;
-- FK's for cp
ALTER TABLE cp
ADD CONSTRAINT fk_perm1permid FOREIGN KEY (permid1)
REFERENCES perms(permid)
ON DELETE CASCADE;
ALTER TABLE cp
ADD CONSTRAINT fk_perm2permid FOREIGN KEY (permid2)
REFERENCES perms(permid)
ON DELETE CASCADE;
-- FK's for ct
ALTER TABLE ct
ADD CONSTRAINT fk_task1taskid FOREIGN KEY (taskid1)
REFERENCES tasks(taskid)
ON DELETE CASCADE;
ALTER TABLE ct
ADD CONSTRAINT fk_task2taskid FOREIGN KEY (taskid2)
REFERENCES tasks(taskid)
ON DELETE CASCADE;
-- FK's for rh
ALTER TABLE rh
ADD CONSTRAINT fk_rolenetrolenetid FOREIGN KEY (rolenetid)
REFERENCES rolenet(rolenetid);
ALTER TABLE rh
ADD CONSTRAINT fk_parentroleid FOREIGN KEY (parentid)
REFERENCES roles(roleid);
ALTER TABLE rh
ADD CONSTRAINT fk_childroleid FOREIGN KEY (childid)
REFERENCES roles(roleid);
-- End of Foreign Key constraints
-- Beginning of Sequence drop and create
Drop Sequence seq_rolenet;
Drop Sequence seq_perms;
Drop Sequence seq_users;
Drop Sequence seq_roles;
Drop Sequence seq_tasks;
Drop Sequence seq_jobs;
Drop Sequence seq_locations;
Drop Sequence seq_locationnet;
CREATE SEQUENCE seq_rolenet;
CREATE SEQUENCE seq_perms;
CREATE SEQUENCE seq_users;
CREATE SEQUENCE seq_roles;
CREATE SEQUENCE seq_tasks;
CREATE SEQUENCE seq_jobs;
CREATE SEQUENCE seq_locations;
CREATE SEQUENCE seq_locationnet;
commit; -- End of Create Sequences

-- When inserting
-- *associations into the ua, pa, rl, rj, jt and tp tables
-- *conflicts into the cu, cl, cr, cj, ct, and cp tables
-- *roles into a role network (rh table)
-- *locations into a location network (lh table)
-- and when deleting
-- *role conflicts from the cr table
-- *role associations from the rh table
-- it becomes necessary to restrict them from occurring based upon
-- the ERBAC integrity constraints.
-- First create the package with the variables for the triggers. This
-- is to prevent mutating table errors. This package contains all the
-- variables for every ERBAC trigger.
CREATE OR REPLACE PACKAGE erbac03_pkg
IS
-- variables for the ua table (user to role association)
TYPE arrua IS TABLE OF ua%ROWTYPE
INDEX BY BINARY_INTEGER;
uavalues arrua;
uaempty arrua;
-- variables for the pa table (permission to role association)
TYPE arrpa IS TABLE OF pa%ROWTYPE
INDEX BY BINARY_INTEGER;
pavalues arrpa;
paempty arrpa;
-- variables for the tp table (permission to task association)
TYPE arrtp IS TABLE OF tp%ROWTYPE
INDEX BY BINARY_INTEGER;
tpvalues arrtp;
tpempty arrtp;
```

```
    -- variables for the rl table (location to role association)
    TYPE arrrl IS TABLE OF rl%ROWTYPE
    INDEX BY BINARY_INTEGER;
    rlvalues arrrl;
    rlempty arrrl;
    -- variables for the rj table (job to role association)
    TYPE arrrj IS TABLE OF rj%ROWTYPE
    INDEX BY BINARY_INTEGER;
    rjvalues arrrj;
    rjempty arrrj;
    -- variables for the jt table (task to job association)
    TYPE arrjt IS TABLE OF jt%ROWTYPE
    INDEX BY BINARY_INTEGER;
    jtvalues arrjt;
    jtempty arrjt;
    -- variables for the cr table (role conflict)
    TYPE arrcr IS TABLE OF cr%ROWTYPE
    INDEX BY BINARY_INTEGER;
    crvalues arrcr;
    crempty arrcr;
    -- variables for the cu table (user conflict)
    TYPE arrcu IS TABLE OF cu%ROWTYPE
    INDEX BY BINARY_INTEGER;
    cuvalues arrcu;
    cuempty arrcu;
    -- variables for the cp table (permission conflict)
    TYPE arrcp IS TABLE OF cp%ROWTYPE
    INDEX BY BINARY_INTEGER;
    cpvalues arrcp;
    cpempty arrcp;
    -- variables for the ct table (task conflict)
    TYPE arrct IS TABLE OF ct%ROWTYPE
    INDEX BY BINARY_INTEGER;
    ctvalues arrct;
    ctempty arrct;
    -- variables for the cj table (job conflict)
    TYPE arrcj IS TABLE OF cj%ROWTYPE
    INDEX BY BINARY_INTEGER;
    cjvalues arrcj;
    cjempty arrcj;
    -- variables for the cl table (location conflict)
    TYPE arrcl IS TABLE OF cl%ROWTYPE
    INDEX BY BINARY_INTEGER;
    clvalues arrcl;
    clempty arrcl;
    -- variables for the cr table (delete role conflict)
    TYPE arrdcr IS TABLE OF cr%ROWTYPE
    INDEX BY BINARY_INTEGER;
    dcrvalues arrdcr;
    dcrempty arrdcr;
    -- variables for the cu table (delete user conflict)
    TYPE arrdcu IS TABLE OF cu%ROWTYPE
    INDEX BY BINARY_INTEGER;
    dcuvalues arrdcu;
    dcuempty arrdcu;
    -- variables for the cl table (delete location conflict)
    TYPE arrdcl IS TABLE OF cl%ROWTYPE
    INDEX BY BINARY_INTEGER;
    dclvalues arrdcl;
    dclempty arrdcl;
    -- variables for the cj table (delete job conflict)
    TYPE arrdcj IS TABLE OF cj%ROWTYPE
    INDEX BY BINARY_INTEGER;
    dcjvalues arrdcj;
    dcjempty arrdcj;
    -- variables for the ct table (delete task conflict)
    TYPE arrdct IS TABLE OF ct%ROWTYPE
    INDEX BY BINARY_INTEGER;
    dctvalues arrdct;
    dctempty arrdct;
    -- variables for the cp table (delete permission conflict)
    TYPE arrdcp IS TABLE OF cp%ROWTYPE
    INDEX BY BINARY_INTEGER;
    dcpvalues arrdcp;
    dcpempty arrdcp;
    -- variables for the rh table (role network)
    TYPE arrrh IS TABLE OF rh%ROWTYPE
    INDEX BY BINARY_INTEGER;
    rhvalues arrrh;
    rhempty arrrh;
    -- variables for the rh table (delete role network)
    TYPE arrdrh IS TABLE OF rh%ROWTYPE
    INDEX BY BINARY_INTEGER;
    drhvalues arrdrh;
    drhempty arrdrh;
    -- variables for the lh table (location network)
    TYPE arrlh IS TABLE OF lh%ROWTYPE
```

```
INDEX BY BINARY_INTEGER;
lhvalues arrlh;
lhempty arrlh;
-- variables for the lh table (delete location network)
TYPE arrdlh IS TABLE OF lh%ROWTYPE
INDEX BY BINARY_INTEGER;
dlhvalues arrdlh;
dlhempty arrdlh;
END;
/
--------------------------------------------------------------------------------
--
-- Triggers for the constraint of the user to role associations.
--
--------------------------------------------------------------------------------
CREATE OR REPLACE TRIGGER uastatbef_trig
BEFORE INSERT ON ua
BEGIN
-- Empty the main PL/SQL table buffer.
erbac03_pkg.uavalues := erbac03_pkg.uaempty;
END;
/
CREATE OR REPLACE TRIGGER uarowbef_trig
BEFORE INSERT ON ua
FOR EACH ROW
DECLARE
i NUMBER := erbac03_pkg.uavalues.COUNT + 1;
BEGIN
-- Copy the row's values across to the PL/SQL table.
erbac03_pkg.uavalues (i).roleid := :new.roleid;
erbac03_pkg.uavalues (i).userid := :new.userid;
END;
/
CREATE OR REPLACE TRIGGER uastatafter_trig
AFTER INSERT ON ua
DECLARE
-- Cursor to select all conflicting users for a particular user.
CURSOR conusercur (v_userid NUMBER) IS
SELECT userid FROM users
WHERE userid IN
(SELECT userid2 FROM cu WHERE userid1 = v_userid
UNION
SELECT userid1 FROM cu WHERE userid2 = v_userid);
-- Get the conflicting roles for the roles associated with the
-- conflicting users.
CURSOR assrolecur (v_userid NUMBER, v_roleid NUMBER) IS
SELECT roleid FROM ua, cr WHERE ua.userid = v_userid
AND ((roleid = cr.roleid1) OR (roleid = cr.roleid2))
AND ((cr.roleid1 = v_roleid) OR (cr.roleid2 = v_roleid));
TYPE arrcu IS TABLE OF users.userid%TYPE
INDEX BY BINARY_INTEGER;
conuser arrcu;
TYPE arrassroles IS TABLE OF ua.roleid%TYPE
INDEX BY BINARY_INTEGER;
assroles arrassroles;
vuserid ua.userid%TYPE;
vroleid ua.roleid%TYPE;
v_count NUMBER(38);
v_stop BOOLEAN := false;
v_cannot NUMBER(1) := 0;
v_can NUMBER(38) := 0;
BEGIN
FOR i IN 1..erbac03_pkg.uavalues.COUNT
LOOP
-- Do the checks required here for each record inserted.
vuserid := erbac03_pkg.uavalues(i).userid;
vroleid := erbac03_pkg.uavalues(i).roleid;
FOR conuserrec IN conusercur(vuserid) LOOP
v_count := conuser.COUNT + 1;
conuser (v_count) := conuserrec.userid;
END LOOP;
-- Count the conflicting roles.
SELECT count(*) INTO v_can
FROM roles
WHERE roleid IN (SELECT roleid2 FROM cr WHERE roleid1 = vroleid
UNION
SELECT roleid1 FROM cr WHERE roleid2 = vroleid);
IF (v_can <> 0) THEN
-- Select all the users associated to the conflicting roles.
-- If they are not all conflicting with the inserted user then it
-- should raise an error.
SELECT count(1) INTO v_cannot
FROM DUAL
WHERE NOT EXISTS
(SELECT distinct(userid) FROM ua WHERE roleid in (SELECT roleid2 FROM cr WHERE roleid1
= vroleid UNION
SELECT roleid1 FROM cr WHERE roleid2 = vroleid) and userid = vuserid);
```

```
IF v_cannot = 0 then
-- There are conflicting users already assigned to conflicting
-- roles. Therefore raise an error.
raise_application_error(-20201, 'Cannot insert user to ' ||
'role association because user is being conflict');
END IF;
END IF;
-- Move to the next step: check for conflicting roles.
IF conuser.COUNT > 0 then
FOR j IN 1..conuser.COUNT LOOP
FOR assrolerec IN assrolecur(conuser(j), vroleid) LOOP
-- This could have been done with a simple SELECT (count 1)
-- SQL statement with an IF statement to check it, but this works.
v_stop := true;
END LOOP;
END LOOP;
-- Move to the next step.
if v_stop = true then -- assroles.COUNT > 0 then
-- There are conflicting roles already assigned to conflicting users.
-- Therefore raise an error.
raise_application_error(-20202,'Cannot insert user to role ' ||
'association - a conflicting user ' ||
'is already associated to a ' ||
'conflicting role.');
END IF;
END IF;
-- Insert the record (or in this case don't do anything because it is
-- already inserted).
END LOOP;
END;
/
--------------------------------------------------------------------------------
--
-- Triggers for the constraint of the job to role associations.
--
--------------------------------------------------------------------------------
CREATE OR REPLACE TRIGGER rjstatbef_trig
BEFORE INSERT ON rj
BEGIN
-- Empty the main PL/SQL table buffer.
erbac03_pkg.rjvalues := erbac03_pkg.rjempty;
END;
/
CREATE OR REPLACE TRIGGER rjrowbef_trig
BEFORE INSERT ON rj
FOR EACH ROW
DECLARE
i NUMBER := erbac03_pkg.rjvalues.COUNT + 1;
BEGIN
-- Copy the row's values across to the PL/SQL table.
erbac03_pkg.rjvalues (i).roleid := :new.roleid;
erbac03_pkg.rjvalues (i).jobid := :new.jobid;
END;
/
CREATE OR REPLACE TRIGGER rjstatafter_trig
AFTER INSERT ON rj
DECLARE
vjobid rj.jobid%TYPE;
vroleid rj.roleid%TYPE;
v_cannot NUMBER(1) := 0;
v_can NUMBER(38) := 0;
v_none NUMBER(1) := 0;
v_tally NUMBER(38) := 0;
BEGIN
FOR i IN 1..erbac03_pkg.rjvalues.COUNT
LOOP
-- Do the checks required here for each record inserted.
v_tally := erbac03_pkg.rjvalues.COUNT;
SELECT count(*) - v_tally INTO v_none FROM rj;
IF (v_none <> 0) THEN
vjobid := erbac03_pkg.rjvalues(i).jobid;
vroleid := erbac03_pkg.rjvalues(i).roleid;
-- Count the conflicting jobs.
SELECT count(*) INTO v_can
FROM jobs
WHERE jobid IN (SELECT jobid2 FROM cj WHERE jobid1 = vjobid
UNION
SELECT jobid1 FROM cj WHERE jobid2 = vjobid);
IF (v_can <> 0) THEN
-- Select all the roles associated to the conflicting jobs.
-- If they are not all conflicting with the inserted role then it
-- should raise an error.
SELECT count(1) INTO v_cannot
FROM DUAL
WHERE NOT EXISTS
(SELECT distinct(roleid) FROM roles
WHERE roleid in
```

```
(SELECT distinct(roleid) FROM rj
WHERE jobid in (SELECT jobid2 FROM cj WHERE jobid1 = vjobid
UNION
SELECT jobid1 FROM cj WHERE jobid2 = vjobid))
MINUS
(SELECT roleid2 FROM cr WHERE roleid1 = vroleid
UNION
SELECT roleid1 FROM cr WHERE roleid2 = vroleid))
AND EXISTS
(SELECT distinct(roleid) FROM rj
WHERE jobid in (SELECT jobid2 FROM cj WHERE jobid1 = vjobid
UNION
SELECT jobid1 FROM cj WHERE jobid2 = vjobid));
IF v_cannot = 0 then
-- There are conflicting roles already assigned to conflicting
-- jobs. Therefore raise an error.
raise_application_error(-20204, 'Cannot insert job to ' ||
'role association - a conflicting ' ||
'job must be assigned to a ' ||
'conflicting role.');
END IF;
END IF;
END IF;
-- Insert the record (or in this case don't do anything because it is
-- already inserted).
END LOOP;
END;
/
--------------------------------------------------------------------------------
--
-- Triggers for the constraint of the task to job associations.
--
--------------------------------------------------------------------------------
CREATE OR REPLACE TRIGGER jtstatbef_trig
BEFORE INSERT ON jt
BEGIN
-- Empty the main PL/SQL table buffer.
erbac03_pkg.jtvalues := erbac03_pkg.jtempty;
END;
/
CREATE OR REPLACE TRIGGER jtrowbef_trig
BEFORE INSERT ON jt
FOR EACH ROW
DECLARE
i NUMBER := erbac03_pkg.jtvalues.COUNT + 1;
BEGIN
-- Copy the row's values across to the PL/SQL table.
erbac03_pkg.jtvalues (i).jobid := :new.jobid;
erbac03_pkg.jtvalues (i).taskid := :new.taskid;
END;
/
CREATE OR REPLACE TRIGGER jtstatafter_trig
AFTER INSERT ON jt
DECLARE
vtaskid jt.taskid%TYPE;
vjobid jt.jobid%TYPE;
v_cannot NUMBER(1) := 0;
v_can NUMBER(38) := 0;
v_none NUMBER(1) := 0;
v_tally NUMBER(38) := 0;
BEGIN
FOR i IN 1..erbac03_pkg.jtvalues.COUNT
LOOP
-- Do the checks required here for each record inserted.
v_tally := erbac03_pkg.jtvalues.COUNT;
SELECT count(*) - v_tally INTO v_none FROM jt;
IF (v_none <> 0) THEN
vtaskid := erbac03_pkg.jtvalues(i).taskid;
vjobid := erbac03_pkg.jtvalues(i).jobid;
-- Count the conflicting tasks.
SELECT count(*) INTO v_can
FROM tasks
WHERE taskid IN (SELECT taskid2 FROM ct WHERE taskid1 = vtaskid
UNION
SELECT taskid1 FROM ct WHERE taskid2 = vtaskid);
IF (v_can <> 0) THEN
-- Select all the jobs associated to the conflicting tasks.
-- If they are not all conflicting with the inserted job then it
-- should raise an error.
SELECT count(1) INTO v_cannot
FROM DUAL
WHERE NOT EXISTS
(SELECT distinct(jobid) FROM jobs
WHERE jobid in
(SELECT distinct(jobid) FROM jt
WHERE taskid in (SELECT taskid2 FROM ct WHERE taskid1 = vtaskid
UNION
```

```
SELECT taskid1 FROM ct WHERE taskid2 = vtaskid))
MINUS
(SELECT jobid2 FROM cj WHERE jobid1 = vjobid
UNION
SELECT jobid1 FROM cj WHERE jobid2 = vjobid))
AND EXISTS
(SELECT distinct(jobid) FROM jt
WHERE taskid in (SELECT taskid2 FROM ct WHERE taskid1 = vtaskid
UNION
SELECT taskid1 FROM ct WHERE taskid2 = vtaskid));
IF v_cannot = 0 then
-- There are conflicting jobs already assigned to conflicting
-- tasks. Therefore raise an error.
raise_application_error(-20205, 'Cannot insert task to ' ||
'job association - a conflicting ' ||
'task must be assigned to a ' ||
'conflicting job.');
END IF;
END IF;
END IF;
-- Insert the record (or in this case don't do anything because it is
-- already inserted).
END LOOP;
END;
/
--------------------------------------------------------------------------------
--
-- Triggers for the constraint of the permission to task associations.
--
--------------------------------------------------------------------------------
CREATE OR REPLACE TRIGGER tpstatbef_trig
BEFORE INSERT ON tp
BEGIN
-- Empty the main PL/SQL table buffer.
erbac03_pkg.tpvalues := erbac03_pkg.tpempty;
END;
/
CREATE OR REPLACE TRIGGER tprowbef_trig
BEFORE INSERT ON tp
FOR EACH ROW
DECLARE
i NUMBER := erbac03_pkg.tpvalues.COUNT + 1;
BEGIN
-- Copy the row's values across to the PL/SQL table.
erbac03_pkg.tpvalues (i).taskid := :new.taskid;
erbac03_pkg.tpvalues (i).permid := :new.permid;
END;
/
CREATE OR REPLACE TRIGGER tpstatafter_trig
AFTER INSERT ON tp
DECLARE
vpermid tp.permid%TYPE;
vtaskid tp.taskid%TYPE;
v_cannot NUMBER(1) := 0;
v_can NUMBER(38) := 0;
v_none NUMBER(1) := 0;
v_tally NUMBER(38) := 0;
BEGIN
FOR i IN 1..erbac03_pkg.tpvalues.COUNT
LOOP
-- Do the checks required here for each record inserted.
v_tally := erbac03_pkg.tpvalues.COUNT;
SELECT count(*) - v_tally INTO v_none FROM tp;
IF (v_none <> 0) THEN
vpermid := erbac03_pkg.tpvalues(i).permid;
vtaskid := erbac03_pkg.tpvalues(i).taskid;
-- Count the conflicting permissions.
SELECT count(*) INTO v_can
FROM perms
WHERE permid IN (SELECT permid2 FROM cp WHERE permid1 = vpermid
UNION
SELECT permid1 FROM cp WHERE permid2 = vpermid);
IF (v_can <> 0) THEN
-- Select all the tasks associated to the conflicting permissions.
-- If they are not all conflicting with the inserted task then it
-- should raise an error.
SELECT count(1) INTO v_cannot
FROM DUAL
WHERE NOT EXISTS
(SELECT distinct(taskid) FROM tasks
WHERE taskid in
(SELECT distinct(taskid) FROM tp
WHERE permid in (SELECT permid2 FROM cp WHERE permid1 = vpermid
UNION
SELECT permid1 FROM cp WHERE permid2 = vpermid))
MINUS
(SELECT taskid2 FROM ct WHERE taskid1 = vtaskid
```

```
UNION
SELECT taskid1 FROM ct WHERE taskid2 = vtaskid))
AND EXISTS
(SELECT distinct(taskid) FROM tp
WHERE permid in (SELECT permid2 FROM cp WHERE permid1 = vpermid
UNION
SELECT permid1 FROM cp WHERE permid2 = vpermid));
IF v_cannot = 0 then
-- There are conflicting tasks already assigned to conflicting
-- permissions. Therefore raise an error.
raise_application_error(-20206, 'Cannot insert permission to ' ||
'task association - a conflicting ' ||
'permission must be assigned to a ' ||
'conflicting task.');
END IF;
END IF;
END IF;
-- Insert the record (or in this case don't do anything because it is
-- already inserted).
END LOOP;
END;
/
--------------------------------------------------------------------------------
--
-- Triggers for the constraint of the role conflict assignment.
--
--------------------------------------------------------------------------------
CREATE OR REPLACE TRIGGER crstatbef_trig
BEFORE INSERT ON cr
BEGIN
-- Empty the main PL/SQL table buffer.
erbac03_pkg.crvalues := erbac03_pkg.crempty;
END;
/
CREATE OR REPLACE TRIGGER crrowbef_trig
BEFORE INSERT ON cr
FOR EACH ROW
DECLARE
i NUMBER := erbac03_pkg.crvalues.COUNT + 1;
BEGIN
-- Copy the row's values across to the PL/SQL table.
erbac03_pkg.crvalues (i).roleid1 := :new.roleid1;
erbac03_pkg.crvalues (i).roleid2 := :new.roleid2;
END;
/
CREATE OR REPLACE TRIGGER crstatafter_trig
AFTER INSERT ON cr
DECLARE
vroleid1 cr.roleid1%TYPE;
vroleid2 cr.roleid2%TYPE;
v_exists NUMBER(38) := 0;
v_asso NUMBER(38) := 0;
v_conf NUMBER(38) := 0;
v_rolenet NUMBER(38) := 0;
BEGIN
FOR i IN 1..erbac03_pkg.crvalues.COUNT
LOOP
vroleid1 := erbac03_pkg.crvalues (i).roleid1;
vroleid2 := erbac03_pkg.crvalues (i).roleid2;
-- Check whether the roles are the same.
IF vroleid1 = vroleid2 THEN
raise_application_error(-20207,'Cannot insert role conflict - ' ||
'both roles are identical.');
END IF;
-- Do they form part of a role network?
SELECT COUNT(1) INTO v_rolenet FROM DUAL
WHERE 2 <= ANY (SELECT COUNT(1) FROM rh
WHERE childid = vroleid1 OR childid = vroleid2
GROUP BY rolenetid);
IF v_rolenet > 0 THEN -- Yes they do so raise an error
raise_application_error(-20208,'Cannot insert role conflict - ' ||
'both roles are already part of a ' ||
'role network.');
END IF;
-- Are they already conflicting? We don't worry about checking whether
-- id1 and id2 are in their respective places as the primary key
-- constraints will prevent that from occuring.
-- This also prevents the same role from conflicting with itself.
SELECT count(1) INTO v_exists FROM cr
WHERE roleid1 = vroleid2 AND roleid2 = vroleid1;
IF (v_exists = 0) THEN
-- Doesn't exist yet so we can continue checking.
-- Check for user-role associations.
SELECT count(1) INTO v_asso FROM ua
WHERE roleid = vroleid1 OR roleid = vroleid2;
IF (v_asso <> 0) THEN
-- Found some associations so go to step 3.
```

```
SELECT count(1) INTO v_conf FROM DUAL
WHERE EXISTS ((SELECT userid FROM users
WHERE userid IN (SELECT userid2 FROM cu
WHERE userid1 IN (SELECT userid FROM ua
WHERE roleid=vroleid1)
UNION
SELECT userid1 FROM cu
WHERE userid2 IN (SELECT userid FROM ua
WHERE roleid=vroleid1)))
INTERSECT
(SELECT userid FROM ua
WHERE roleid = vroleid2));
IF (v_conf <> 0) THEN
raise_application_error(-20209,'Cannot insert role conflict - ' ||
'both roles are already associated ' ||
'to conflicting users.');
END IF;
END IF;
ELSE
raise_application_error(-20210,'Cannot insert role conflict - ' ||
'the role conflict already exists.');
END IF;
END LOOP;
END;
/
--------------------------------------------------------------------------------
--
-- Triggers for the constraint of the user conflict assignment.
--
--------------------------------------------------------------------------------
CREATE OR REPLACE TRIGGER custatbef_trig
BEFORE INSERT ON cu
BEGIN
-- Empty the main PL/SQL table buffer.
erbac03_pkg.cuvalues := erbac03_pkg.cuempty;
END;
/
CREATE OR REPLACE TRIGGER curowbef_trig
BEFORE INSERT ON cu
FOR EACH ROW
DECLARE
i NUMBER := erbac03_pkg.cuvalues.COUNT + 1;
BEGIN
-- Copy the row's values across to the PL/SQL table.
erbac03_pkg.cuvalues (i).userid1 := :new.userid1;
erbac03_pkg.cuvalues (i).userid2 := :new.userid2;
END;
/
CREATE OR REPLACE TRIGGER custatafter_trig
AFTER INSERT ON cu
DECLARE
vuserid1 cu.userid1%TYPE;
vuserid2 cu.userid2%TYPE;
v_exists NUMBER(38) := 0;
v_asso NUMBER(38) := 0;
v_conf NUMBER(38) := 0;
BEGIN
FOR i IN 1..erbac03_pkg.cuvalues.COUNT
LOOP
vuserid1 := erbac03_pkg.cuvalues (i).userid1;
vuserid2 := erbac03_pkg.cuvalues (i).userid2;
-- Check whether the users are the same.
IF vuserid1 = vuserid2 THEN
raise_application_error(-20219,'Cannot insert user conflict - ' ||
'both users are identical.');
END IF;
-- Are they already conflicting? We don't worry about checking whether
-- id1 and id2 are in their respective places as the primary key
-- constraints will prevent that from occuring.
-- This also prevents the same user from conflicting with itself.
SELECT count(1) INTO v_exists FROM cu
WHERE userid1 = vuserid2 AND userid2 = vuserid1;
IF (v_exists = 0) THEN
-- Doesn't exist yet so we can continue checking.
-- Check for user-role associations.
SELECT count(1) INTO v_asso FROM ua
WHERE userid = vuserid1 OR userid = vuserid2;
IF (v_asso <> 0) THEN
-- Found some associations so go to step 3.
SELECT count(1) INTO v_conf FROM DUAL
WHERE EXISTS ((SELECT roleid FROM roles
WHERE roleid IN (SELECT roleid2 FROM cr
WHERE roleid1 IN (SELECT roleid FROM ua
WHERE userid=vuserid1)
UNION
SELECT roleid1 FROM cr
WHERE roleid2 IN (SELECT roleid FROM ua
```

```
WHERE userid=vuserid1)))
INTERSECT
(SELECT roleid FROM ua
WHERE userid = vuserid2));
IF (v_conf <> 0) THEN
raise_application_error(-20211,'Cannot insert user conflict - ' ||
'both users are already associated ' ||
'to conflicting roles.');
END IF;
END IF;
ELSE
raise_application_error(-20212,'Cannot insert user conflict - ' ||
'the user conflict already exists.');
END IF;
END LOOP;
END;
/
--------------------------------------------------------------------------------
--
-- Triggers for the constraint of the permission conflict assignment.
--
--------------------------------------------------------------------------------
CREATE OR REPLACE TRIGGER cpstatbef_trig
BEFORE INSERT ON cp
BEGIN
-- Empty the main PL/SQL table buffer.
erbac03_pkg.cpvalues := erbac03_pkg.cpempty;
END;
/
CREATE OR REPLACE TRIGGER cprowbef_trig
BEFORE INSERT ON cp
FOR EACH ROW
DECLARE
i NUMBER := erbac03_pkg.cpvalues.COUNT + 1;
BEGIN
-- Copy the row's values across to the PL/SQL table.
erbac03_pkg.cpvalues (i).permid1 := :new.permid1;
erbac03_pkg.cpvalues (i).permid2 := :new.permid2;
END;
/
CREATE OR REPLACE TRIGGER cpstatafter_trig
AFTER INSERT ON cp
DECLARE
vpermid1 cp.permid1%TYPE;
vpermid2 cp.permid2%TYPE;
v_exists NUMBER(38) := 0;
v_asso NUMBER(38) := 0;
v_conf NUMBER(38) := 0;
BEGIN
FOR i IN 1..erbac03_pkg.cpvalues.COUNT
LOOP
-- We need to check for an empty table?
vpermid1 := erbac03_pkg.cpvalues (i).permid1;
vpermid2 := erbac03_pkg.cpvalues (i).permid2;
-- Check whether the permissions are the same.
IF vpermid1 = vpermid2 THEN
raise_application_error(-20213,'Cannot insert permission conflict - ' ||
'both permissions are identical.');
END IF;
-- Are they already conflicting? We don't worry about checking whether
-- id1 and id2 are in their respective places as the primary key
-- constraints will prevent that from occuring.
-- This also prevents the same permission from conflicting with itself.
SELECT count(1) INTO v_exists FROM cp
WHERE permid1 = vpermid2 AND permid2 = vpermid1;
IF (v_exists = 0) THEN
-- Doesn't exist yet so we can continue checking.
-- Check for task-permission associations for both permissions only.
-- If one of the permissions has no association yet then allow add.
SELECT COUNT(1) INTO v_asso FROM tp p1, tp p2
WHERE p1.permid = vpermid1 AND p2.permid = vpermid2;
IF (v_asso <> 0) THEN
-- Found some associations so go to step 3.
SELECT count(1) INTO v_conf FROM DUAL
WHERE EXISTS ((SELECT taskid FROM tp
WHERE permid = vpermid2)
MINUS
(SELECT taskid FROM tasks
WHERE taskid IN (SELECT taskid2 FROM ct
WHERE taskid1 IN
(SELECT taskid FROM tp
WHERE permid=vpermid1)
UNION
SELECT taskid1 FROM ct
WHERE taskid2 IN
(SELECT taskid FROM tp
WHERE permid=vpermid1))));
```

```
IF (v_conf <> 0) THEN
raise_application_error(-20214,'Cannot insert permission ' ||
'conflict - one or more of the ' ||
'associated tasks of the two ' ||
'permissions are not conflicting.');
END IF;
END IF;
ELSE
raise_application_error(-20215,'Cannot insert permission conflict ' ||
'- permission conflict already exists.');
END IF;
END LOOP;
END;
/
--------------------------------------------------------------------------------
--
-- Triggers for the constraint of the task conflict assignment.
--
--------------------------------------------------------------------------------
CREATE OR REPLACE TRIGGER ctstatbef_trig
BEFORE INSERT ON ct
BEGIN
-- Empty the main PL/SQL table buffer.
erbac03_pkg.ctvalues := erbac03_pkg.ctempty;
END;
/
CREATE OR REPLACE TRIGGER ctrowbef_trig
BEFORE INSERT ON ct
FOR EACH ROW
DECLARE
i NUMBER := erbac03_pkg.ctvalues.COUNT + 1;
BEGIN
-- Copy the row's values across to the PL/SQL table.
erbac03_pkg.ctvalues (i).taskid1 := :new.taskid1;
erbac03_pkg.ctvalues (i).taskid2 := :new.taskid2;
END;
/
CREATE OR REPLACE TRIGGER ctstatafter_trig
AFTER INSERT ON ct
DECLARE
vtaskid1 ct.taskid1%TYPE;
vtaskid2 ct.taskid2%TYPE;
v_exists NUMBER(38) := 0;
v_asso NUMBER(38) := 0;
v_conf NUMBER(38) := 0;
BEGIN
FOR i IN 1..erbac03_pkg.ctvalues.COUNT
LOOP
-- We need to check for an empty table?
vtaskid1 := erbac03_pkg.ctvalues (i).taskid1;
vtaskid2 := erbac03_pkg.ctvalues (i).taskid2;
-- Check whether the tasks are the same.
IF vtaskid1 = vtaskid2 THEN
raise_application_error(-20216,'Cannot insert task conflict - ' ||
'both tasks are identical.');
END IF;
-- Are they already conflicting? We don't worry about checking whether
-- id1 and id2 are in their respective places as the primary key
-- constraints will prevent that from occuring.
-- This also prevents the same task from conflicting with itself.
SELECT count(1) INTO v_exists FROM ct
WHERE taskid1 = vtaskid2 AND taskid2 = vtaskid1;
IF (v_exists = 0) THEN
-- Doesn't exist yet so we can continue checking.
-- Check for task-job associations for both tasks only.
-- If one of the tasks has no association yet then allow add.
SELECT COUNT(1) INTO v_asso FROM jt t1, jt t2
WHERE t1.taskid = vtaskid1 AND t2.taskid = vtaskid2;
IF (v_asso <> 0) THEN
-- Found some associations so go to step 3.
SELECT count(1) INTO v_conf FROM DUAL
WHERE EXISTS ((SELECT jobid FROM jt
WHERE taskid = vtaskid2)
MINUS
(SELECT jobid FROM jobs
WHERE jobid IN (SELECT jobid2 FROM cj
WHERE jobid1 IN
(SELECT jobid FROM jt
WHERE taskid=vtaskid1)
UNION
SELECT jobid1 FROM cj
WHERE jobid2 IN
(SELECT jobid FROM jt
WHERE taskid=vtaskid1))));
IF (v_conf <> 0) THEN
raise_application_error(-20217,'Cannot insert task conflict - ' ||
'one or more of the associated ' ||
```

```
    'jobs of the two tasks are not ' ||
    'conflicting.');
    END IF;
    END IF;
    -- Check for task-permission associations for both tasks only.
    -- If one of the tasks has no association yet then allow add.
    SELECT COUNT(1) INTO v_asso FROM tp t1, tp t2
    WHERE t1.taskid = vtaskid1 AND t2.taskid = vtaskid2;
    IF (v_asso <> 0) THEN
    SELECT count(1) INTO v_conf FROM DUAL
    WHERE EXISTS ((SELECT permid FROM tp
    WHERE taskid = vtaskid2)
    MINUS
    (SELECT permid FROM perms
    WHERE permid IN (SELECT permid2 FROM cp
    WHERE permid1 IN
    (SELECT permid FROM tp
    WHERE taskid=vtaskid1)
    UNION
    SELECT permid1 FROM cp
    WHERE permid2 IN
    (SELECT permid FROM tp
    WHERE taskid=vtaskid1))));
    IF (v_conf <> 0) THEN
    raise_application_error(-20218,'Cannot insert task conflict - ' ||
    'one or more of the associated ' ||
    'perms of the two tasks are not ' ||
    'conflicting.');
    END IF;
    END IF;
    ELSE
    raise_application_error(-20219,'Cannot insert task conflict - ' ||
    'task conflict already exists.');
    END IF;
    END LOOP;
    END;
    /
    --------------------------------------------------------------------------------
    --
    -- Triggers for the constraint of the job conflict assignment.
    --
    --------------------------------------------------------------------------------
    CREATE OR REPLACE TRIGGER cjstatbef_trig
    BEFORE INSERT ON cj
    BEGIN
    -- Empty the main PL/SQL table buffer.
    erbac03_pkg.cjvalues := erbac03_pkg.cjempty;
    END;
    /
    CREATE OR REPLACE TRIGGER cjrowbef_trig
    BEFORE INSERT ON cj
    FOR EACH ROW
    DECLARE
    i NUMBER := erbac03_pkg.cjvalues.COUNT + 1;
    BEGIN
    -- Copy the row's values across to the PL/SQL table.
    erbac03_pkg.cjvalues (i).jobid1 := :new.jobid1;
    erbac03_pkg.cjvalues (i).jobid2 := :new.jobid2;
    END;
    /
    CREATE OR REPLACE TRIGGER cjstatafter_trig
    AFTER INSERT ON cj
    DECLARE
    vjobid1 cj.jobid1%TYPE;
    vjobid2 cj.jobid2%TYPE;
    v_exists NUMBER(38) := 0;
    v_asso NUMBER(38) := 0;
    v_conf NUMBER(38) := 0;
    BEGIN
    FOR i IN 1..erbac03_pkg.cjvalues.COUNT
    LOOP
    -- We need to check for an empty table?
    vjobid1 := erbac03_pkg.cjvalues (i).jobid1;
    vjobid2 := erbac03_pkg.cjvalues (i).jobid2;
    -- Check whether the jobs are the same.
    IF vjobid1 = vjobid2 THEN
    raise_application_error(-20220,'Cannot insert job conflict - ' ||
    'both jobs are identical.');
    END IF;
    -- Are they already conflicting? We don't worry about checking whether
    -- id1 and id2 are in their respective places as the primary key
    -- constraints will prevent that from occuring.
    -- This also prevents the same job from conflicting with itself.
    SELECT count(1) INTO v_exists FROM cj
    WHERE jobid1 = vjobid2 AND jobid2 = vjobid1;
    IF (v_exists = 0) THEN
    -- Doesn't exist yet so we can continue checking.
```

```
   -- Check for job-role associations for both jobs only.
   -- If one of the jobs has no association yet then allow add.
   SELECT COUNT(1) INTO v_asso FROM rj t1, rj t2
   WHERE t1.jobid = vjobid1 AND t2.jobid = vjobid2;
   IF (v_asso <> 0) THEN
   -- Found some associations so go to step 3.
   SELECT count(1) INTO v_conf FROM DUAL
   WHERE EXISTS ((SELECT roleid FROM rj
   WHERE jobid = vjobid2)
   MINUS
   (SELECT roleid FROM roles
   WHERE roleid IN (SELECT roleid2 FROM cr
   WHERE roleid1 IN
   (SELECT roleid FROM rj
   WHERE jobid=vjobid1)
   UNION
   SELECT roleid1 FROM cr
   WHERE roleid2 IN
   (SELECT roleid FROM rj
   WHERE jobid=vjobid1))));
   IF (v_conf <> 0) THEN
   raise_application_error(-20221,'Cannot insert job conflict - ' ||
   'one or more of the associated ' ||
   'roles of the two jobs are not ' ||
   'conflicting.');
   END IF;
   END IF;
   -- Check for job-task associations for both jobs only.
   -- If one of the jobs has no association yet then allow add.
   SELECT COUNT(1) INTO v_asso FROM jt t1, jt t2
   WHERE t1.jobid = vjobid1 AND t2.jobid = vjobid2;
   IF (v_asso <> 0) THEN
   SELECT count(1) INTO v_conf FROM DUAL
   WHERE EXISTS ((SELECT taskid FROM jt
   WHERE jobid = vjobid2)
   MINUS
   (SELECT taskid FROM tasks
   WHERE taskid IN (SELECT taskid2 FROM ct
   WHERE taskid1 IN
   (SELECT taskid FROM jt
   WHERE jobid=vjobid1)
   UNION
   SELECT taskid1 FROM ct
   WHERE taskid2 IN
   (SELECT taskid FROM jt
   WHERE jobid=vjobid1))));
   IF (v_conf <> 0) THEN
   raise_application_error(-20222,'Cannot insert job conflict - ' ||
   'one or more of the associated ' ||
   'tasks of the two tasks are not ' ||
   'conflicting.');
   END IF;
   END IF;
   ELSE
   raise_application_error(-20223,'Cannot insert job conflict - ' ||
   'job conflict already exists.');
   END IF;
   END LOOP;
   END;
   /
   ------------------------------------------------------------------------------
   --
   -- Triggers for the constraint of the location conflict assignment.
   --
   ------------------------------------------------------------------------------
   CREATE OR REPLACE TRIGGER clstatbef_trig
   BEFORE INSERT ON cl
   BEGIN
   -- Empty the main PL/SQL table buffer.
   erbac03_pkg.clvalues := erbac03_pkg.clempty;
   END;
   /
   CREATE OR REPLACE TRIGGER clrowbef_trig
   BEFORE INSERT ON cl
   FOR EACH ROW
   DECLARE
   i NUMBER := erbac03_pkg.clvalues.COUNT + 1;
   BEGIN
   -- Copy the row's values across to the PL/SQL table.
   erbac03_pkg.clvalues (i).locationid1 := :new.locationid1;
   erbac03_pkg.clvalues (i).locationid2 := :new.locationid2;
   END;
   /
   CREATE OR REPLACE TRIGGER clstatafter_trig
   AFTER INSERT ON cl
   DECLARE
   vlocationid1 cl.locationid1%TYPE;
```

```
vlocationid2 cl.locationid2%TYPE;
v_exists NUMBER(38) := 0;
v_asso NUMBER(38) := 0;
v_conf NUMBER(38) := 0;
v_locationnet NUMBER(38) := 0;
BEGIN
FOR i IN 1..erbac03_pkg.clvalues.COUNT
LOOP
vlocationid1 := erbac03_pkg.clvalues (i).locationid1;
vlocationid2 := erbac03_pkg.clvalues (i).locationid2;
-- Check whether the locations are the same.
IF vlocationid1 = vlocationid2 THEN
raise_application_error(-20218,'Cannot insert location conflict - ' ||
'both locations are identical.');
END IF;
-- Do they form part of a location network?
SELECT COUNT(1) INTO v_locationnet FROM DUAL
WHERE 2 <= ANY (SELECT COUNT(1) FROM lh
WHERE lchildid = vlocationid1 OR lchildid = vlocationid2
GROUP BY locationnetid);
IF v_locationnet > 0 THEN -- Yes they do so raise an error
raise_application_error(-20224,'Cannot insert location conflict - ' ||
'both locations are already part of a ' ||
'location network.');
END IF;
-- Are they already conflicting? We don't worry about checking whether
-- id1 and id2 are in their respective places as the primary key
-- constraints will prevent that from occuring.
-- This also prevents the same location from conflicting with itself.
SELECT count(1) INTO v_exists FROM cl
WHERE locationid1 = vlocationid2 AND locationid2 = vlocationid1;
IF (v_exists = 0) THEN
-- Doesn't exist yet so we can continue checking.
-- Check for location-role associations.
SELECT count(1) INTO v_asso FROM rl
WHERE locationid = vlocationid1 OR locationid = vlocationid2;
IF (v_asso <> 0) THEN
-- Found some associations so go to step 3.
SELECT count(1) INTO v_conf FROM DUAL
WHERE EXISTS ((SELECT roleid FROM roles
WHERE roleid IN (SELECT roleid2 FROM cr
WHERE roleid1 IN (SELECT roleid FROM rl
WHERE locationid=vlocationid1)
UNION
SELECT roleid1 FROM cr
WHERE roleid2 IN (SELECT roleid FROM rl
WHERE locationid=vlocationid1)))
INTERSECT
(SELECT roleid FROM rl
WHERE locationid = vlocationid2));
IF (v_conf <> 0) THEN
raise_application_error(-20225,'Cannot insert location conflict - ' ||
'both locations are already associated ' ||
'to conflicting roles.');
END IF;
END IF;
ELSE
raise_application_error(-20226,'Cannot insert location conflict - ' ||
'the location conflict already exists.');
END IF;
END LOOP;
END;
/
--------------------------------------------------------------------------------
--
-- Triggers for ensuring integrity when deleting role conflict assignments.
--
--------------------------------------------------------------------------------
CREATE OR REPLACE TRIGGER dcrstatbef_trig
BEFORE DELETE ON cr
BEGIN
-- Empty the main PL/SQL table buffer.
erbac03_pkg.dcrvalues := erbac03_pkg.dcrempty;
END;
/
CREATE OR REPLACE TRIGGER dcrrowbef_trig
BEFORE DELETE ON cr
FOR EACH ROW
DECLARE
i NUMBER := erbac03_pkg.dcrvalues.COUNT + 1;
BEGIN
-- Copy the row's values across to the PL/SQL table.
erbac03_pkg.dcrvalues (i).roleid1 := :old.roleid1;
erbac03_pkg.dcrvalues (i).roleid2 := :old.roleid2;
END;
/
CREATE OR REPLACE TRIGGER dcrstatafter_trig
```

```
AFTER DELETE ON cr
DECLARE
vroleid1 cr.roleid1%TYPE;
vroleid2 cr.roleid2%TYPE;
v_stop BOOLEAN := false;
v_exists NUMBER(38) := 0;
v_passo1 NUMBER(38) := 0;
v_passo2 NUMBER(38) := 0;
v_rlsso1 NUMBER(38) := 0;
v_rlsso2 NUMBER(38) := 0;
v_rjsso1 NUMBER(38) := 0;
v_rjsso2 NUMBER(38) := 0;
v_conf NUMBER(38) := 0;
BEGIN
FOR i IN 1..erbac03_pkg.dcrvalues.COUNT
LOOP
vroleid1 := erbac03_pkg.dcrvalues (i).roleid1;
vroleid2 := erbac03_pkg.dcrvalues (i).roleid2;
-- Doesn't exist yet so we can continue checking.
SELECT count(1) INTO v_passo1 FROM pa
WHERE roleid = vroleid1;
SELECT count(1) INTO v_passo2 FROM pa
WHERE roleid = vroleid2;
SELECT count(1) INTO v_rlsso1 FROM rl
WHERE roleid = vroleid1;
SELECT count(1) INTO v_rlsso2 FROM rl
WHERE roleid = vroleid2;
SELECT count(1) INTO v_rjsso1 FROM rj
WHERE roleid = vroleid1;
SELECT count(1) INTO v_rjsso2 FROM rj
WHERE roleid = vroleid2;
IF ((v_passo1 <> 0) AND (v_passo2 <> 0))
OR ((v_rlsso1 <> 0) AND (v_rlsso2 <> 0))
OR ((v_rjsso1 <> 0) AND (v_rjsso2 <> 0)) THEN
-- Found some associations so go to step 3.
IF ((v_passo1 <> 0) AND (v_passo2 <> 0)) THEN
-- Check the permissions first.
SELECT count(1) INTO v_conf FROM DUAL
WHERE EXISTS ((SELECT permid FROM perms
WHERE permid IN (SELECT permid2 FROM cp
WHERE permid1 IN (SELECT permid FROM pa
WHERE roleid=vroleid1)
UNION
SELECT permid1 FROM cp
WHERE permid2 IN (SELECT permid FROM pa
WHERE roleid=vroleid1)))
INTERSECT
(SELECT permid FROM pa
WHERE roleid = vroleid2));
IF (v_conf <> 0) THEN
v_stop := true;
END IF;
END IF;
v_conf := 0;
IF ((v_rlsso1 <> 0) AND (v_rlsso2 <> 0)) THEN -- check the locations next.
SELECT count(1) INTO v_conf FROM DUAL
WHERE EXISTS ((SELECT locationid FROM locations
WHERE locationid IN (SELECT locationid2 FROM cl
WHERE locationid1 IN (SELECT locationid FROM rl
WHERE roleid=vroleid1)
UNION
SELECT locationid1 FROM cl
WHERE locationid2 IN (SELECT locationid FROM rl
WHERE roleid=vroleid1)))
INTERSECT
(SELECT locationid FROM rl
WHERE roleid = vroleid2));
IF (v_conf <> 0) THEN
v_stop := true;
END IF;
END IF;
v_conf := 0;
IF ((v_rjsso1 <> 0) AND (v_rjsso2 <> 0)) THEN -- check the jobs next.
SELECT count(1) INTO v_conf FROM DUAL
WHERE EXISTS ((SELECT jobid FROM jobs
WHERE jobid IN (SELECT jobid2 FROM cj
WHERE jobid1 IN (SELECT jobid FROM rj
WHERE roleid=vroleid1)
UNION
SELECT jobid1 FROM cj
WHERE jobid2 IN (SELECT jobid FROM rj
WHERE roleid=vroleid1)))
INTERSECT
(SELECT jobid FROM rj
WHERE roleid = vroleid2));
IF (v_conf <> 0) THEN
v_stop := true;
```

```
END IF;
END IF;
IF (v_stop = true) THEN
raise_application_error(-20227,'Cannot delete role conflict - ' ||
'the roles form part of associations ' ||
'to locations or jobs that will ' ||
'become invalid if the role conflict ' ||
'is deleted.');
END IF;
END IF;
END LOOP;
END;
/
--------------------------------------------------------------------------------
--
-- Triggers for ensuring integrity when deleting job conflict assignments.
--
--------------------------------------------------------------------------------
CREATE OR REPLACE TRIGGER dcjstatbef_trig
BEFORE DELETE ON cj
BEGIN
-- Empty the main PL/SQL table buffer.
erbac03_pkg.dcjvalues := erbac03_pkg.dcjempty;
END;
/
CREATE OR REPLACE TRIGGER dcjrowbef_trig
BEFORE DELETE ON cj
FOR EACH ROW
DECLARE
i NUMBER := erbac03_pkg.dcjvalues.COUNT + 1;
BEGIN
-- Copy the row's values across to the PL/SQL table.
erbac03_pkg.dcjvalues (i).jobid1 := :old.jobid1;
erbac03_pkg.dcjvalues (i).jobid2 := :old.jobid2;
END;
/
CREATE OR REPLACE TRIGGER dcjstatafter_trig
AFTER DELETE ON cj
DECLARE
vjobid1 cj.jobid1%TYPE;
vjobid2 cj.jobid2%TYPE;
v_stop BOOLEAN := false;
v_exists NUMBER(38) := 0;
v_jtsso1 NUMBER(38) := 0;
v_jtsso2 NUMBER(38) := 0;
v_conf NUMBER(38) := 0;
BEGIN
FOR i IN 1..erbac03_pkg.dcjvalues.COUNT
LOOP
vjobid1 := erbac03_pkg.dcjvalues (i).jobid1;
vjobid2 := erbac03_pkg.dcjvalues (i).jobid2;
-- Doesn't exist yet so we can continue checking.
SELECT count(1) INTO v_jtsso1 FROM jt
WHERE jobid = vjobid1;
SELECT count(1) INTO v_jtsso2 FROM jt
WHERE jobid = vjobid2;
IF ((v_jtsso1 <> 0) AND (v_jtsso2 <> 0)) THEN
-- Found some associations so go to step 3.
IF ((v_jtsso1 <> 0) AND (v_jtsso2 <> 0)) THEN
-- Check the tasks first.
SELECT count(1) INTO v_conf FROM DUAL
WHERE EXISTS ((SELECT taskid FROM tasks
WHERE taskid IN (SELECT taskid2 FROM ct
WHERE taskid1 IN (SELECT taskid FROM jt
WHERE jobid=vjobid1)
UNION
SELECT taskid1 FROM ct
WHERE taskid2 IN (SELECT taskid FROM jt
WHERE jobid=vjobid1)))
INTERSECT
(SELECT taskid FROM jt
WHERE jobid = vjobid2));
IF (v_conf <> 0) THEN
v_stop := true;
END IF;
END IF;
IF (v_stop = true) THEN
raise_application_error(-20228,'Cannot delete job conflict - ' ||
'the jobs form part of associations ' ||
'to tasks that will ' ||
'become invalid if the job conflict ' ||
'is deleted.');
END IF;
END IF;
END LOOP;
END;
/
```

```
--------------------------------------------------------------------------------
--
-- Triggers for ensuring integrity when deleting task conflict assignments.
--
--------------------------------------------------------------------------------
CREATE OR REPLACE TRIGGER dctstatbef_trig
BEFORE DELETE ON ct
BEGIN
-- Empty the main PL/SQL table buffer.
erbac03_pkg.dctvalues := erbac03_pkg.dctempty;
END;
/
CREATE OR REPLACE TRIGGER dctrowbef_trig
BEFORE DELETE ON ct
FOR EACH ROW
DECLARE
i NUMBER := erbac03_pkg.dctvalues.COUNT + 1;
BEGIN
-- Copy the row's values across to the PL/SQL table.
erbac03_pkg.dctvalues (i).taskid1 := :old.taskid1;
erbac03_pkg.dctvalues (i).taskid2 := :old.taskid2;
END;
/
CREATE OR REPLACE TRIGGER dctstatafter_trig
AFTER DELETE ON ct
DECLARE
vtaskid1 ct.taskid1%TYPE;
vtaskid2 ct.taskid2%TYPE;
v_stop BOOLEAN := false;
v_exists NUMBER(38) := 0;
v_tpsso1 NUMBER(38) := 0;
v_tpsso2 NUMBER(38) := 0;
v_conf NUMBER(38) := 0;
BEGIN
FOR i IN 1..erbac03_pkg.dctvalues.COUNT
LOOP
vtaskid1 := erbac03_pkg.dctvalues (i).taskid1;
vtaskid2 := erbac03_pkg.dctvalues (i).taskid2;
-- Doesn't exist yet so we can continue checking.
SELECT count(1) INTO v_tpsso1 FROM tp
WHERE taskid = vtaskid1;
SELECT count(1) INTO v_tpsso2 FROM tp
WHERE taskid = vtaskid2;
IF ((v_tpsso1 <> 0) AND (v_tpsso2 <> 0)) THEN
-- Found some associations so go to step 3.
IF ((v_tpsso1 <> 0) AND (v_tpsso2 <> 0)) THEN
-- Check the permissions first.
SELECT count(1) INTO v_conf FROM DUAL
WHERE EXISTS ((SELECT permid FROM perms
WHERE permid IN (SELECT permid2 FROM cp
WHERE permid1 IN (SELECT permid FROM tp
WHERE taskid=vtaskid1)
UNION
SELECT permid1 FROM cp
WHERE permid2 IN (SELECT permid FROM tp
WHERE taskid=vtaskid1)))
INTERSECT
(SELECT permid FROM tp
WHERE taskid = vtaskid2));
IF (v_conf <> 0) THEN
v_stop := true;
END IF;
END IF;
IF (v_stop = true) THEN
raise_application_error(-20229,'Cannot delete task conflict - ' ||
'the tasks form part of associations ' ||
'to permissions that will ' ||
'become invalid if the task conflict ' ||
'is deleted.');
END IF;
END IF;
END LOOP;
END;
/
--------------------------------------------------------------------------------
--
-- Triggers for the constraint of the role network association.
--
--------------------------------------------------------------------------------
CREATE OR REPLACE TRIGGER rhstatbef_trig
BEFORE INSERT ON rh
BEGIN
-- Empty the main PL/SQL table buffer.
erbac03_pkg.rhvalues := erbac03_pkg.rhempty;
END;
/
CREATE OR REPLACE TRIGGER rhrowbef_trig
```

```
BEFORE INSERT ON rh
FOR EACH ROW
DECLARE
i NUMBER := erbac03_pkg.rhvalues.COUNT + 1;
BEGIN
-- Copy the row's values across to the PL/SQL table.
erbac03_pkg.rhvalues (i).rolenetid := :new.rolenetid;
erbac03_pkg.rhvalues (i).parentid := :new.parentid;
erbac03_pkg.rhvalues (i).childid := :new.childid;
END;
/
CREATE OR REPLACE TRIGGER rhstatafter_trig
AFTER INSERT ON rh
DECLARE
e_rolenet_loop EXCEPTION;
PRAGMA EXCEPTION_INIT (
e_rolenet_loop, -1436);
vrolenetid rh.rolenetid%TYPE;
vparentid rh.parentid%TYPE;
vchildid rh.childid%TYPE;
v_root1 NUMBER(38) := 0;
v_root2 NUMBER(38) := 0;
v_exists NUMBER(38) := 0;
v_circ NUMBER(38) := 0;
v_conf NUMBER(38) := 0;
v_count NUMBER(38) := erbac03_pkg.rhvalues.COUNT;
BEGIN
FOR i IN 1..erbac03_pkg.rhvalues.COUNT
LOOP
vrolenetid := erbac03_pkg.rhvalues (i).rolenetid;
vparentid := erbac03_pkg.rhvalues (i).parentid;
vchildid := erbac03_pkg.rhvalues (i).childid;
-- If there is no root record for the current rolenet
-- (count - current records) then we cannot add the record
-- if the parentid is not null.
IF (vparentid IS NOT NULL) THEN
SELECT count(1) INTO v_root1 FROM rh
WHERE rolenetid = vrolenetid AND parentid IS NULL;
IF (v_root1 = 0) THEN
raise_application_error(-20231,'Cannot insert role into role ' ||
'network - root role is missing ' ||
'for the particular role network');
END IF;
END IF;
-- Check for an existing root role. If one is found for the particular
-- role network then raise an error.
IF (vparentid IS NULL) THEN
-- Must subtract the count because the record is already in the table.
-- This of course assumes that there will only ever be one record
-- inserted at a time. Disable all triggers and constraints when doing
-- batch transfers.
SELECT count(1) - v_count INTO v_root2 FROM rh
WHERE rolenetid = vrolenetid AND parentid IS NULL;
IF (v_root2 > 0) THEN
raise_application_error(-20217,'Cannot insert role into role ' ||
'network - root role already ' ||
'exists for the particular role network.');
END IF;
END IF;
-- Is parent role in the role network as a child?
-- If not then raise an error.
SELECT count(1) INTO v_exists FROM rh
WHERE rolenetid = rolenetid AND childid = vparentid;
-- Added extra checks to make the error message that
-- occurs better because the record is already in the
-- table. This screws up this check in cases when the
-- parentid = childid. Also check if it is root level.
IF ((v_exists <> 0) AND (vparentid <> vchildid)) OR (vparentid IS NULL) THEN
-- Will it cause a circular reference? Rely on an
-- exception to be raised when the sql statement
-- fails. If it doesn't fail then it will work just
-- fine. This works because the record is already in
-- the table when this trigger fires.
SELECT count(childid) INTO v_circ
FROM rh
WHERE childid <> vparentid
START WITH childid = vparentid AND rolenetid = vrolenetid
CONNECT BY PRIOR parentid = childid AND rolenetid = vrolenetid;
-- Don't worry about checking the result of the previous
-- SQL statement as it will not get to the next statement
-- if it failed.
-- Check whether any of the records in the role network
-- are conflicting with the child we wish to add. We only
-- need to check the children in the role network.
SELECT count(1) INTO v_conf FROM DUAL
WHERE vchildid IN
(SELECT roleid FROM roles WHERE roleid IN
```

```
(SELECT roleid2 FROM cr WHERE roleid1 IN
(SELECT childid FROM rh WHERE rolenetid = vrolenetid)
UNION
SELECT roleid1 FROM cr WHERE roleid2 IN
(SELECT childid FROM rh WHERE rolenetid = vrolenetid)));
IF (v_conf > 0) THEN
raise_application_error(-20232,'Cannot insert role into role ' ||
'network - conflicting role(s) ' ||
'are already present in the role ' ||
'network.');
END IF;
ELSE
raise_application_error(-20233,'Cannot insert role into role ' ||
'network - the parent role is not an ' ||
'existing child role.');
END IF;
END LOOP;
EXCEPTION
WHEN e_rolenet_loop THEN
raise_application_error(-20234,'Cannot insert role into role ' ||
'network - it will cause a circular ' ||
'reference to occur.');
END;
/
--------------------------------------------------------------------------------
--
-- Triggers for the constraint of the location network association.
--
--------------------------------------------------------------------------------
CREATE OR REPLACE TRIGGER lhstatbef_trig
BEFORE INSERT ON lh
BEGIN
-- Empty the main PL/SQL table buffer.
erbac03_pkg.lhvalues := erbac03_pkg.lhempty;
END;
/
CREATE OR REPLACE TRIGGER lhrowbef_trig
BEFORE INSERT ON lh
FOR EACH ROW
DECLARE
i NUMBER := erbac03_pkg.lhvalues.COUNT + 1;
BEGIN
-- Copy the row's values across to the PL/SQL table.
erbac03_pkg.lhvalues (i).locationnetid := :new.locationnetid;
erbac03_pkg.lhvalues (i).lparentid := :new.lparentid;
erbac03_pkg.lhvalues (i).lchildid := :new.lchildid;
END;
/
CREATE OR REPLACE TRIGGER lhstatafter_trig
AFTER INSERT ON lh
DECLARE
e_locationnet_loop EXCEPTION;
PRAGMA EXCEPTION_INIT (
e_locationnet_loop, -1436);
vlocationnetid lh.locationnetid%TYPE;
vlparentid lh.lparentid%TYPE;
vlchildid lh.lchildid%TYPE;
v_root1 NUMBER(38) := 0;
v_root2 NUMBER(38) := 0;
v_exists NUMBER(38) := 0;
v_circ NUMBER(38) := 0;
v_conf NUMBER(38) := 0;
v_count NUMBER(38) := erbac03_pkg.lhvalues.COUNT;
BEGIN
FOR i IN 1..erbac03_pkg.lhvalues.COUNT
LOOP
vlocationnetid := erbac03_pkg.lhvalues (i).locationnetid;
vlparentid := erbac03_pkg.lhvalues (i).lparentid;
vlchildid := erbac03_pkg.lhvalues (i).lchildid;
-- If there is no root record for the current locationnet
-- (count - current records) then we cannot add the record
-- if the lparentid is not null.
IF (vlparentid IS NOT NULL) THEN
SELECT count(1) INTO v_root1 FROM lh
WHERE locationnetid = vlocationnetid AND lparentid IS NULL;
IF (v_root1 = 0) THEN
raise_application_error(-20234,'Cannot insert location into location ' ||
'network - root location is missing ' ||
'for the particular location network');
END IF;
END IF;
-- Check for an existing root location. If one is found for the particular
-- location network then raise an error.
IF (vlparentid IS NULL) THEN
-- Must subtract the count because the record is already in the table.
-- This of course assumes that there will only ever be one record
-- inserted at a time. Disable all triggers and constraints when doing
```

```
-- batch transfers.
SELECT count(1) - v_count INTO v_root2 FROM lh
WHERE locationnetid = vlocationnetid AND lparentid IS NULL;
IF (v_root2 > 0) THEN
raise_application_error(-20236,'Cannot insert location into location ' ||
'network - root location already ' ||
'exists for the particular location network.');
END IF;
END IF;
-- Is parent location in the location network as a child?
-- If not then raise an error.
SELECT count(1) INTO v_exists FROM lh
WHERE locationnetid = locationnetid AND lchildid = vlparentid;
-- Added extra checks to make the error message that
-- occurs better because the record is already in the
-- table. This screws up this check in cases when the
-- lparentid = lchildid. Also check if it is root level.
IF ((v_exists <> 0) AND (vlparentid <> vlchildid)) OR (vlparentid IS NULL) THEN
-- Will it cause a circular reference? Rely on an
-- exception to be raised when the sql statement
-- fails. If it doesn't fail then it will work just
-- fine. This works because the record is already in
-- the table when this trigger fires.
SELECT count(lchildid) INTO v_circ
FROM lh
WHERE lchildid <> vlparentid
START WITH lchildid = vlparentid AND locationnetid = vlocationnetid
CONNECT BY PRIOR lparentid = lchildid AND locationnetid = vlocationnetid;
-- Don't worry about checking the result of the previous
-- SQL statement as it will not get to the next statement
-- if it failed.
-- Check whether any of the records in the location network
-- are conflicting with the child we wish to add. We only
-- need to check the children in the location network.
SELECT count(1) INTO v_conf FROM DUAL
WHERE vlchildid IN
(SELECT locationid FROM locations WHERE locationid IN
(SELECT locationid2 FROM cl WHERE locationid1 IN
(SELECT lchildid FROM lh WHERE locationnetid = vlocationnetid)
UNION
SELECT locationid1 FROM cl WHERE locationid2 IN
(SELECT lchildid FROM lh WHERE locationnetid = vlocationnetid)));
IF (v_conf > 0) THEN
raise_application_error(-20237,'Cannot insert location into location ' ||
'network - conflicting location(s) ' ||
'are already present in the location ' ||
'network.');
END IF;
ELSE
raise_application_error(-20238,'Cannot insert location into location ' ||
'network - the parent location is not an ' ||
'existing child location.');
END IF;
END LOOP;
EXCEPTION
WHEN e_locationnet_loop THEN
raise_application_error(-20239,'Cannot insert location into location ' ||
'network - it will cause a circular ' ||
'reference to occur.');
END;
/
--------------------------------------------------------------------------------
--
-- Triggers for ensuring the integrity when deleting role network associations.
--
--------------------------------------------------------------------------------
CREATE OR REPLACE TRIGGER drhstatbef_trig
BEFORE DELETE ON rh
BEGIN
-- Empty the main PL/SQL table buffer.
erbac03_pkg.drhvalues := erbac03_pkg.drhempty;
END;
/
CREATE OR REPLACE TRIGGER drhrowbef_trig
BEFORE DELETE ON rh
FOR EACH ROW
DECLARE
i NUMBER := erbac03_pkg.drhvalues.COUNT + 1;
BEGIN
-- Copy the row's values across to the PL/SQL table.
erbac03_pkg.drhvalues (i).rolenetid := :old.rolenetid;
erbac03_pkg.drhvalues (i).parentid := :old.parentid;
erbac03_pkg.drhvalues (i).childid := :old.childid;
END;
/
CREATE OR REPLACE TRIGGER drhstatafter_trig
AFTER DELETE ON rh
```

```
DECLARE
vrolenetid rh.rolenetid%TYPE;
vparentid rh.parentid%TYPE;
vchildid rh.childid%TYPE;
v_child NUMBER(38) := 0;
BEGIN
FOR i IN 1..erbac03_pkg.drhvalues.COUNT
LOOP
vrolenetid := erbac03_pkg.drhvalues (i).rolenetid;
vparentid := erbac03_pkg.drhvalues (i).parentid;
vchildid := erbac03_pkg.drhvalues (i).childid;
-- Does it have children?
SELECT COUNT(parentid) INTO v_child
FROM rh
WHERE parentid = vchildid AND rolenetid = vrolenetid;
IF (v_child > 0) THEN --It has children so prevent it from been deleted.
raise_application_error(-20240,'Cannot delete role association - ' ||
'children associations already ' ||
'exist for the role been deleted.');
END IF;
END LOOP;
END;
/

--------------------------------------------------------------------------------
--
-- Triggers for ensuring the integrity when deleting location network associations.
--
--------------------------------------------------------------------------------
CREATE OR REPLACE TRIGGER dlhstatbef_trig
BEFORE DELETE ON lh
BEGIN
-- Empty the main PL/SQL table buffer.
erbac03_pkg.dlhvalues := erbac03_pkg.dlhempty;
END;
/
CREATE OR REPLACE TRIGGER dlhrowbef_trig
BEFORE DELETE ON lh
FOR EACH ROW
DECLARE
i NUMBER := erbac03_pkg.dlhvalues.COUNT + 1;
BEGIN
-- Copy the row's values across to the PL/SQL table.
erbac03_pkg.dlhvalues (i).locationnetid := :old.locationnetid;
erbac03_pkg.dlhvalues (i).lparentid := :old.lparentid;
erbac03_pkg.dlhvalues (i).lchildid := :old.lchildid;
END;
/
CREATE OR REPLACE TRIGGER dlhstatafter_trig
AFTER DELETE ON lh
DECLARE
vlocationnetid lh.locationnetid%TYPE;
vlparentid lh.lparentid%TYPE;
vlchildid lh.lchildid%TYPE;
v_child NUMBER(38) := 0;
BEGIN
FOR i IN 1..erbac03_pkg.dlhvalues.COUNT
LOOP
vlocationnetid := erbac03_pkg.dlhvalues (i).locationnetid;
vlparentid := erbac03_pkg.dlhvalues (i).lparentid;
vlchildid := erbac03_pkg.dlhvalues (i).lchildid;
-- Does it have children?
SELECT COUNT(lparentid) INTO v_child
FROM lh
WHERE lparentid = vlchildid AND locationnetid = vlocationnetid;
IF (v_child > 0) THEN --It has children so prevent it from been deleted.
raise_application_error(-20241,'Cannot delete location association - ' ||
'children associations already ' ||
'exist for the location been deleted.');
END IF;
END LOOP;
END;
/
--------------------------------------------------------------------------------
--
-- End of ERBAC03 integrity constraint rules.
--
--------------------------------------------------------------------------------
```

ประวัติผู้เขียน

ชื่อ-นามสกุล นายบุรินทร์ เย็นมั่นคง

วัน เดือน ปีเกิด 10 พฤศจิกายน 2519 ที่กรุงเทพฯ

ที่อยู่ 67 ซ.รามคำแหง 46 ถนนรามคำแหง แขวงหัวหมาก
เขตบางกะปิ กรุงเทพฯ 10240 โทร. 0-2732-2585

ประวัติการศึกษา 2541 วิทยาศาสตรบัณฑิต สาขาวิชาวิทยาการคอมพิวเตอร์ มหาวิทยาลัยรังสิต
2543 นิเทศศาสตรบัณฑิต สาขาวิชาการโฆษณา
มหาวิทยาลัยสุโขทัยธรรมาธิราช

ความชำนาญเฉพาะด้าน 1.) การวิเคราะห์และออกแบบฐานข้อมูล
2.) เขียนโปรแกรมจัดการฐานข้อมูล
3.) ระบบคอมพิวเตอร์แบบกระจาย

ประสบการณ์การทำงานและผลงานวิจัย

ปัจจุบัน ตำแหน่งพนักงานโปรแกรมคอมพิวเตอร์ระดับ 5 บริษัทไปรษณีย์ไทย จำกัด