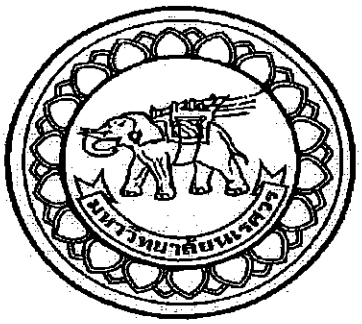


โปรแกรมประเมิน Design Pattern จาก source code

Design Pattern Evaluation Program

นายธีรภัทร์ ยังดำรง รหัส 46361937

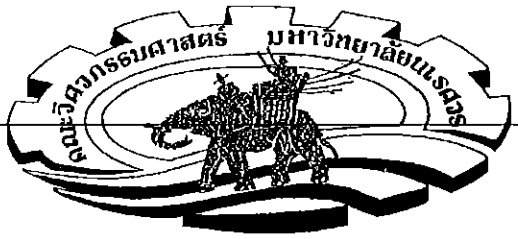
นางสาววณิชยา ราชบัณฑิต รหัส 46362067

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมไฟฟ้าและคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ มหาวิทยาลัยนเรศวร

ปีการศึกษา 2549



ใบรับรองโครงงานวิศวกรรม

หัวข้อโครงงาน โปรแกรมประเมิน Design Pattern จาก source code

ผู้ดำเนินโครงงาน นายธีรภัทร์ ยังดำรง รหัส 46361937

นางสาววณิชยา ราชบัณฑิต รหัส 46362067

อาจารย์ที่ปรึกษา ดร. สุรเดช จิตประไพกุลศาล

สาขาวิชา วิศวกรรมคอมพิวเตอร์

ภาควิชา วิศวกรรมไฟฟ้าและคอมพิวเตอร์

ปีการศึกษา 2549

.....

คณะวิศวกรรมศาสตร์ มหาวิทยาลัยนเรศวร อนุมัติให้โครงงานฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิศวกรรมศาสตรบัณฑิต สาขาวิศวกรรมคอมพิวเตอร์

คณะกรรมการสอบโครงงานวิศวกรรม

.....ประธานกรรมการ

(ดร.สุรเดช จิตประไพกุลศาล)

.....กรรมการ

(ดร.พนมขวัญ ริยะมงคล)

.....กรรมการ

(อาจารย์จิราพร พุกสุข)

หัวข้อโครงงาน	โปรแกรมประเมิน Design Pattern จาก source code		
ผู้ดำเนินโครงงาน	นายธีรภัทร์	ยังดำรง	รหัส 46361937
	นางสาววณิชยา	ราชบัณฑิต	รหัส 46362067
อาจารย์ที่ปรึกษา	ดร. สุรเดช	จิตประไพกุลศาล	
สาขาวิชา	วิศวกรรมคอมพิวเตอร์		
ภาควิชา	วิศวกรรมไฟฟ้าและคอมพิวเตอร์		
ปีการศึกษา	2549		

บทคัดย่อ

ในโครงงานนี้เราได้พัฒนาโปรแกรมที่สำหรับเปรียบเทียบ (Matching) โครงสร้างของ Design Pattern ใน format ของ RSF กับ Design Pattern ใน format ของ RML ที่จัดเก็บในฐานข้อมูล โปรแกรมนี้จะใช้หลักการของ Adjacency Matrix, Graph Isomorphism และ Genetic Algorithm ช่วยในการเปรียบเทียบโครงสร้าง จากผลการทดลองพบว่าโปรแกรมสามารถตรวจเปรียบเทียบโครงสร้างของ Design Pattern ใน format ของ RSF ว่ามีลักษณะเหมือนกับ Design Pattern ชนิดใดที่จัดเก็บฐานข้อมูลได้

Project title	Design Pattern Evaluation Program		
Name	Mr. Teerapat	Yangdamrong	ID. 46361937
	Miss Wanichaya	Rachabundid	ID. 46362067
Project advisor	Dr. Suradet	Jitprapaikulsarn	
Major	Computer Engineering		
Department	Electrical and Computer Engineering		
Academic year	2006		

Abstract

This project develops a program for matching structure of a Design Pattern in RSF format to a Design Pattern in RML format from database. This program employs the principle of Adjacency Matrix, Graph Isomorphism and Genetic Algorithm to carry out the structure. Our experiments show that the Design Pattern Evaluation program can match structures of patterns to Design Patterns in database and show the matching Design Patterns.

กิตติกรรมประกาศ

โครงงานวิศวกรรมคอมพิวเตอร์สำเร็จได้ด้วยดี ก็เนื่องจากความอนุเคราะห์จากท่านอาจารย์ที่ปรึกษา อาจารย์สุรเดช จิตประไพกุลศาล ที่กรุณาให้คำปรึกษา แนะนำวิธีการในการทำงาน ตลอดถึงการตรวจสอบการทำงานพร้อมทั้งเสนอแนวทางการแก้ไขตลอดระยะเวลาการทำโครงงาน สุดท้ายต้องขอขอบพระคุณอาจารย์ทุกท่านและเพื่อนๆ ทุกคนที่ยังไม่ได้เอ่ยนามที่คอยสนับสนุนในการทำโครงงานครั้งนี้

สารบัญ

หน้า

สารบัญ

หน้า

สารบัญตาราง

ตารางที่ หน้า

สารบัญรูป

รูปที่ หน้า

สารบัญรูป(ต่อ)

รูปที่ หน้า

สารบัญรูป(ต่อ)

รูปที่ หน้า

สารบัญรูป(ต่อ)

รูปที่ หน้า

บทที่ 1
บทนำ

1.1 ที่มาและความสำคัญของโครงงาน

ปัจจุบันเทคโนโลยีสารสนเทศและการสื่อสารได้เข้ามามีบทบาทสำคัญอย่างยิ่งในชีวิตประจำวันของคนทั่วไป ทั้งทางตรงและทางอ้อม ซึ่งเราได้นำเทคโนโลยีเหล่านี้มาช่วยในการดำเนินงานต่างๆ เพื่อช่วยให้ประสิทธิภาพในการทำงานดีขึ้น ส่งผลให้ตลาดของอุตสาหกรรมซอฟต์แวร์มีการเจริญเติบโตอย่างรวดเร็ว จึงทำให้มีการแข่งขันทางด้านอุตสาหกรรมการพัฒนาซอฟต์แวร์สูงขึ้นตามไปด้วย

สิ่งสำคัญประการหนึ่งในการพัฒนาซอฟต์แวร์ให้เป็นที่ต้องการของผู้ใช้ คือ คุณภาพของซอฟต์แวร์ หากซอฟต์แวร์ที่พัฒนาขึ้นมานั้นมีคุณภาพดี ใช้ย่อมเกิดความมั่นใจในตัวของซอฟต์แวร์และผู้พัฒนาเอง แต่ถ้าซอฟต์แวร์ขาดคุณภาพและเกิดความผิดพลาดก็จะส่งผลกระทบต่อผู้ใช้ และยังส่งผลให้ผู้พัฒนาได้รับผลกระทบจากการขาดความเชื่อถือด้วย ดังนั้นการพัฒนาซอฟต์แวร์จึงจำเป็นอย่างยิ่งที่จะต้องควบคุมคุณภาพของซอฟต์แวร์ให้มีคุณภาพที่ดี

สิ่งที่จะช่วยให้การออกแบบซอฟต์แวร์มีคุณภาพที่ดีนั้น คือ การออกแบบซอฟต์แวร์โดยใช้ Design Pattern เนื่องจาก Design Pattern เป็นรูปแบบของการออกแบบที่ได้ผ่านการทดสอบใช้งานจริง อีกทั้งยังผ่านการตรวจทาน (Review) โดยผู้เชี่ยวชาญ ซอฟต์แวร์ที่ใช้ Design Pattern ในการออกแบบมักจะมีความยืดหยุ่นสูงและง่ายต่อการพัฒนา, เปลี่ยนแปลง, แก้ไขและทดสอบ ซึ่งทำให้ซอฟต์แวร์มีคุณภาพดีเป็นที่ยอมรับและเชื่อถือของผู้ใช้

ดังนั้น การประเมินคุณภาพการออกแบบซอฟต์แวร์ (Software Design) โดยใช้ Design Pattern จะเริ่มจากการทำวิศวกรรมย้อนกลับ (reverse engineering) ของ source code เพื่อวิเคราะห์โครงสร้างและจัดทำคำอธิบายโครงสร้างใน Pattern Language ออกมาซึ่งเราจะนำไปเปรียบเทียบและตรวจสอบกับตัว Design Pattern ในฐานข้อมูล โดยวิธีการประเมินคุณภาพของ Software Design โดยใช้ Design Pattern นั้นจำเป็นที่จะต้องคำนึงถึงว่าในแต่ละ Design นั้นอาจใช้ Design Pattern ได้หลายชนิด และอาจใช้หลาย Design Pattern ผสมกันอยู่ ซึ่ง การใช้ Design Pattern บางชนิดจำเป็นที่จะต้องประกอบกันอยู่กับ Design Pattern อีกชนิด ซึ่งจะทำให้เราสามารถทราบได้ว่า source code นั้นมีการใช้ Design Pattern ที่เหมาะสมหรือไม่

1.2 วัตถุประสงค์โครงงาน

1. ศึกษาเกี่ยวกับทฤษฎีและหลักการเกี่ยวกับ Design Pattern เพื่อนำไปตรวจเปรียบเทียบการใช้ Design Pattern ใน source code ได้
2. เขียนโปรแกรมที่ใช้ในการตรวจสอบโครงสร้างระหว่าง Pattern ในรูปแบบของ RSF กับ Design Pattern ในรูปแบบ RML ที่จัดเก็บในฐานข้อมูล

1.3 ขอบข่ายการทำงาน

1. ศึกษาหลักการและโครงสร้างของ Design Pattern
2. สร้างโปรแกรมในการตรวจสอบและประเมิน Pattern ระหว่างใน source code กับ Design Pattern ในฐานข้อมูล
3. ศึกษาการติดต่อกับฐานข้อมูลและการเก็บข้อมูลในฐานข้อมูลในรูป RML

1.4 ขั้นตอนดำเนินงาน

1. ศึกษาเกี่ยวกับทฤษฎีและหลักการในสิ่งต่างๆ เหล่านี้
   - หลักการและวิธีการใช้ Design Pattern
   - หลักการทำงานของ RML และ RSF
   - การเขียนโปรแกรมเพื่อตรวจสอบและประเมิน Pattern Language
   - การเขียนโปรแกรมเพื่อติดต่อกับฐานข้อมูล
2. ออกแบบและพัฒนาโปรแกรม
3. ทดสอบโปรแกรม
4. ทำการปรับปรุงและแก้ไขโปรแกรม
5. วิเคราะห์การทดสอบพร้อมทั้งสรุปผล
6. จัดทำเป็นรูปเล่ม

1.5 แผนการดำเนินงาน

กิจกรรม	ปี 2548		ปี 2549									
	พ.ย.	ธ.ค.	ม.ค.	ก.พ.	มี.ค.	เม.ย.	พ.ค.	มิ.ย.	ก.ค.	ส.ค.	ก.ย.	ต.ค.
1. ศึกษาหลักการและวิธีการใช้ Design Pattern, การเขียนโปรแกรมเพื่อประเมิน, การติดต่อกับฐานข้อมูล	←		—	→								
2. ออกแบบและพัฒนาโปรแกรม					←	—	→					
3. ทดสอบโปรแกรม						←	—	—	—	→		
4. ทำการปรับปรุงและแก้ไขโปรแกรม							←	—	—	→		
5. วิเคราะห์การทดสอบพร้อมทั้งสรุปผล								←	—	→		
6. จัดทำรูปเล่มโครงงาน										←	—	→

1.6 ผลที่คาดว่าจะได้รับ

1. เข้าใจหลักการและโครงสร้าง Design Pattern
2. เข้าใจหลักการเขียนโปรแกรมเพื่อเปรียบเทียบโครงสร้าง Design Pattern
3. เข้าใจหลักการทำงาน RML และ RSF
4. สามารถสร้างและพัฒนาโปรแกรมเพื่อเปรียบเทียบโครงสร้าง Pattern กับ Design Pattern ในฐานข้อมูลได้
5. โปรแกรมที่พัฒนาสามารถติดต่อกับฐานข้อมูลที่เก็บ Design Pattern ไว้ได้

1.7 งบประมาณ

1.	ค่าวัสดุสำนักงาน	เป็นเงิน	500	บาท
2.	ค่าวัสดุคอมพิวเตอร์	เป็นเงิน	500	บาท
3.	ค่าถ่ายเอกสาร	เป็นเงิน	800	บาท
4.	ค่าวัสดุอื่น ๆ	เป็นเงิน	200	บาท
		รวมเป็นเงินทั้งสิ้น	2,000	บาท (สองพันบาทถ้วน)

บทที่ 2
หลักการและทฤษฎี

2.1 ทฤษฎีพื้นฐานของกราฟ

กราฟเป็นโครงสร้างข้อมูลที่ใช้แสดงการเชื่อมต่อระหว่างสิ่งต่างๆ โดยประกอบไปด้วยจุด (Vertex) และเส้นเชื่อม (Edge) ที่ต่อจุดต่างๆ เข้าด้วยกัน ซึ่งจากแนวความคิดในเรื่องของกราฟได้นำมาแก้ปัญหาในการทำงานหลายๆ ด้าน เช่น การหาระยะทางที่สั้นที่สุด การสร้างแผนที่ หรือใช้เขียนแทนการเชื่อมต่อระหว่างเครื่องคอมพิวเตอร์

กราฟ (Graph) คือ คู่อันดับ (V, E) โดยที่ V คือ เซตของจุดยอด (Vertex set) E คือ เซตของด้านหรือเส้นเชื่อม (Edge set) โดยแต่ละด้านนั้นจะเกิดจากจุด 2 จุดในเซต V

2.2 ประเภทของกราฟ

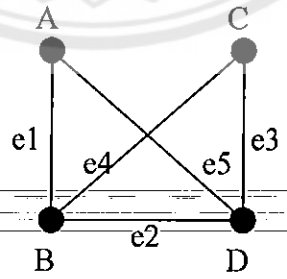
2.2.1 กราฟไม่มีทิศทาง(Undirected Graph)

กราฟไม่มีทิศทาง คือ กราฟที่แสดงเส้นเชื่อมต่อระหว่าง vertex แต่ไม่แสดงทิศทาง สัญลักษณ์ที่ใช้ในการแทนกราฟ คือ

$$G = (V, E)$$

โดยที่ เซต V เรียกสมาชิกของ V ว่าจุดยอด (Vertex)

เซต E เป็นเซตคู่ไม่อันดับของจุดยอด (Edge) โดยที่ไม่มี edge จากจุดยอดใดไปหาตัวมันเอง และมีความสัมพันธ์แบบสมมาตร คือ คู่อันดับ (u, v)



รูปที่ 2.1 กราฟไม่มีทิศทาง (Undirected Graph)

จากรูปที่ 2.1

เซต V = {A, B, C, D}

เซต E = {e1, e2, e3, e4} โดยที่ e1 = {A, B}, e2 = {B, D}, e3 = {C, D},
e4 = {B, C} และ e5 = {A, D}

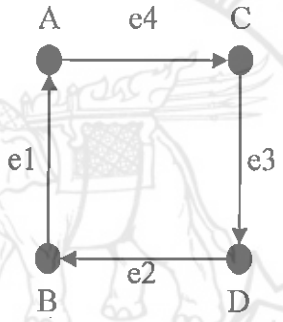
2.2.2 กราฟแสดงทิศทาง

กราฟแสดงทิศทาง คือ กราฟที่ประกอบไปด้วยเซตของ vertex และ edge โดยแต่ละเส้นของ edge จะเชื่อมโยงจุดสองจุด โดยมีลำดับที่แน่นอน สัญลักษณ์ที่ใช้ในการแทนกราฟ คือ G = (V, E)

โดยที่

เซต V เรียกสมาชิกของ V ว่าจุดยอด (Vertex)

เซต E โดย e แต่ละด้านใน E ตรงกับคู่อันดับที่เป็นจุดยอดใน V



รูปที่ 2.2 กราฟไม่มีทิศทาง (Directed Graph)

จากรูปที่ 2.2

เซต V = {A, B, C, D}

เซต E = {e1, e2, e3, e4} โดยที่

e1 ตรงกับคู่อันดับ (B, A) จึงเขียนลูกศรชี้ไปจุด A

e2 ตรงกับคู่อันดับ (D, B) จึงเขียนลูกศรชี้ไปจุด B

e3 ตรงกับคู่อันดับ (C, D) จึงเขียนลูกศรชี้ไปจุด D

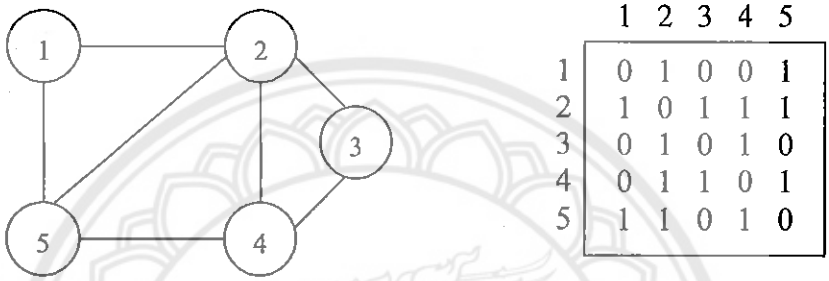
e4 ตรงกับคู่อันดับ (A, C) จึงเขียนลูกศรชี้ไปจุด C

2.3 การ Represent graph

เราสามารถแทนกราฟได้ 2 วิธี คือการใช้ Adjacency Matrix และ Adjacency List

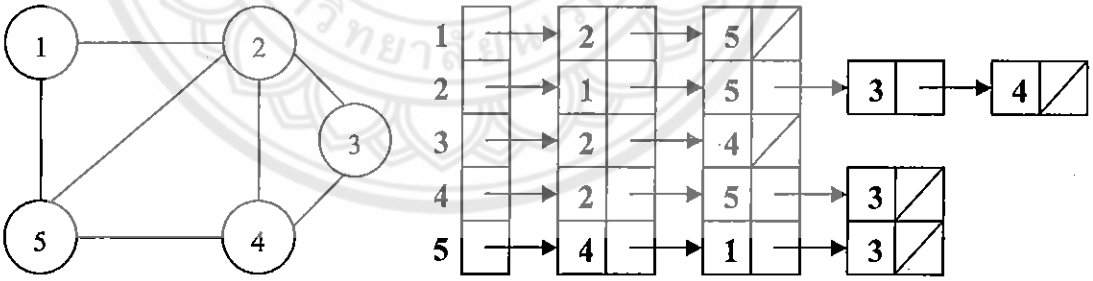
2.3.1 เมตริกซ์ประชิด (Adjacency Matrix)

การใช้เมตริกซ์ประชิด โดยการเลือกจุดยอดในอันดับใดๆ ก็ได้ แล้วเขียนแถวและคอลัมน์ของเมตริกซ์ชุดหนึ่งด้วยจุดยอดที่เรียงอันดับแล้ว สมาชิกในเมตริกซ์จะเป็น 1 ถ้ามีด้านเชื่อมโยงจุดยอดของแถวและคอลัมน์ในกราฟ และเป็น 0 ถ้าไม่มีด้านเชื่อมโยงจุดยอดของแถวและคอลัมน์ในกราฟ เมตริกซ์ประชิดนิยมใช้แทนกราฟที่ลักษณะหนาแน่น (Densed Matrix)



รูปที่ 2.3 แสดงการแทนกราฟด้วยเมตริกซ์ประชิด

2.3.2 Adjacency List



รูปที่ 2.4 แสดงการแทนกราฟด้วย Adjacency List

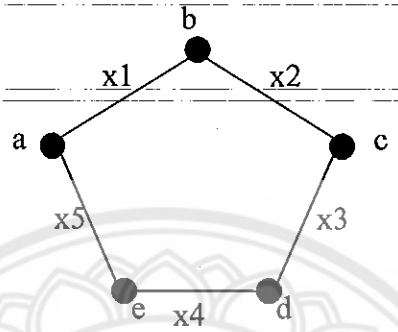
2.4 ดีกรี

ให้ G(V, E) เป็นกราฟ และให้ (u, v) เป็นจุดยอดของกราฟ ดีกรีของจุดยอด V ในกราฟ G จะสามารถเขียนแทนได้ด้วย

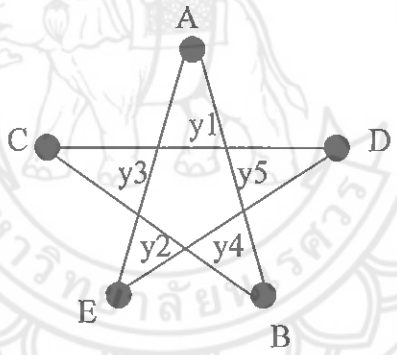
deg(v) หมายถึง จำนวนของด้านที่ตกกระทบบน V หรือจำนวนของด้านซึ่งมี V เป็นจุดปลายนั้นเอง

2.5 การถอดแบบของกราฟ

ในการวาดกราฟ โดยกำหนดให้ วาดกราฟที่มีจุดยอด คือ a, b, c, d, e และลากเส้นเชื่อมระหว่าง ab, bc, cd, de, และ ae ให้เป็นด้านของกราฟ สามารถสร้างรูปกราฟออกมาที่แตกต่างกัน ดังเช่น ในรูปที่ 2.5 และรูป 2.6



รูปที่ 2.5 กราฟ G1



รูปที่ 2.6 กราฟ G2

ซึ่งรูปกราฟทั้งกราฟ G1 และ G2 นั้น จะแทนกราฟเดียวกัน ถึงแม้ว่าภาพที่ปรากฏออกมาจะต่างกัน เราจะเรียกกราฟทั้งสอง ว่า กราฟถอดแบบกัน (Isomorphic) ซึ่งกราฟถอดแบบกันนั้นสามารถพิจารณาจาก

2.4.1 Invariant

กราฟที่จะถอดแบบกันนั้นต้องมีคุณสมบัติ ที่เรียกว่า Invariant คือ จำนวนจุดยอด, จำนวนด้านของกราฟ และจำนวนดีกรีของจุดยอดต้องมีจำนวนเท่ากัน

2.4.2 เมตริกซ์ตกกระทบของกราฟที่นำมาตรวจสอบ

การที่กราฟจะเป็นกราฟที่ถอดแบบกันนั้น ก็ต่อเมื่อ การเรียงอันดับของจุดยอดและด้านเมตริกซ์ตกกระทบของกราฟทั้งสอง เหมือนกัน

จากรูปที่ 2.5 และรูปที่ 2.6 สามารถเขียนเมตริกซ์ตกกระทบ ได้ดังรูปที่ 2.7

	a	b	c	d	e
a	0	1	0	0	1
b	1	0	1	0	0
c	0	1	0	1	0
d	0	0	1	0	1

	A	B	C	D	E
A	0	1	0	0	1
B	1	0	1	0	0
C	0	1	0	1	0
D	0	0	1	0	1

รูปที่ 2.7 เมตริกซ์ตกกระทบของกราฟ G1 และกราฟ G2

2.6 ทฤษฎีเบื้องต้นของอัลกอริทึมทางพันธุศาสตร์ (Genetic Algorithm: GA)

ถึงแม้ว่าในปัจจุบันจะมีการคิดค้นอัลกอริทึม ในการแก้ปัญหาที่หลากหลายและแตกต่างกันออกไป เพื่อให้วิธีการเหล่านั้นเหมะสมกับปัญหาที่เกิดขึ้น และอัลกอริทึมทางพันธุศาสตร์ หรือ Genetic Algorithm (GA) เป็นวิธีการแก้ปัญหาอย่างหนึ่งที่ช่วยให้ได้มาซึ่งคำตอบที่เหมาะสมที่สุดให้กับปัญหา

ในกระบวนการค้นหาคำตอบสำหรับปัญหาของ GA นั้น มีคุณสมบัติของการเลียนแบบการถ่ายทอดทางพันธุกรรม นั่นคือ หลักการคัดเลือกแบบธรรมชาติและหลักการทางสายพันธุ์ ซึ่งในปัจจุบันเป็นที่ยอมรับถึงประสิทธิภาพและมีการนำไปประยุกต์อย่างกว้างขวาง และในการนำเอาความสามารถทางด้านวิวัฒนาการทางพันธุศาสตร์มาประยุกต์ใช้ก็ต้องมีความรู้พื้นฐานเกี่ยวกับปรากฏการณ์ต่างๆ ที่เกิดขึ้นในกระบวนการทางพันธุศาสตร์ เช่น selection เป็นการคัดเลือกโครโมโซมที่สมควรนำไปสืบพันธ์, mating การจับคู่โครโมโซมที่ได้รับการคัดเลือก, crossover คือการผสมกันระหว่างโครโมโซมที่ได้จากการจับคู่กัน และ mutation คือ การปรับเปลี่ยนค่าบางตำแหน่งของโครโมโซม

2.7 ความเป็นมาของอัลกอริทึมทางพันธุศาสตร์

จากทฤษฎีวิวัฒนาการ หรือทฤษฎีการอยู่รอดของสิ่งมีชีวิต ของ Charles Darwin ทำให้ John Holland นักวิทยาศาสตร์สาขาวิทยาการคอมพิวเตอร์ ได้ทำการคิดค้นการลอกเลียนแบบขั้นตอนธรรมชาติของการพัฒนาสิ่งมีชีวิตขึ้นในปี คริสต์ศักราช 1970 โดยพัฒนาขึ้นกับเพื่อนร่วมงานและนักศึกษาของมหาวิทยาลัย Michigan ประเทศสหรัฐอเมริกา โดยที่มีจุดมุ่งหมายเพื่อ

1. อธิบายการเปลี่ยนแปลงกระบวนการทางธรรมชาติของพันธุกรรม
2. เพื่อที่จะนำกลไกการเปลี่ยนแปลงเหล่านี้มาประยุกต์ใช้กับการเขียนโปรแกรม

จากการคิดค้นของ John Holland ทำให้สามารถค้นหาและแก้ปัญหาให้ได้จุดที่เหมาะสมที่สุด สำหรับหลักการของวิธีการค้นหาแบบ GA คือ สิ่งมีชีวิตทั้งหมดจะมีทั้งลักษณะที่ดีและไม่ดี ซึ่งสิ่งมีชีวิตที่มีลักษณะที่ดีนั้นจะได้รับการสนับสนุนให้มีการถ่ายทอดทางพันธุกรรม เพื่อให้ได้สิ่งมีชีวิตใหม่ที่ดีขึ้น ดังนั้น ในหลักการทำงานของ GA จึงถูกนำเสนอในรูปแบบโครโมโซม นั่นหมายความว่า คำตอบที่สามารถเป็นไปได้ทั้งหมดของปัญหาจะถูกนำมาแปลงเป็นโครโมโซม เพื่อนำโครโมโซมไปใช้ในกระบวนการถ่ายทอดลักษณะทางพันธุกรรม โดยจะใช้ Fitness Function (ค่าความเหมาะสม) ที่มีความสอดคล้องกับวัตถุประสงค์กำหนดให้แต่ละโครโมโซม และโครโมโซมเหล่านั้นจะถูกนำมาพิจารณาว่าโครโมโซมใดควรนำมาสืบสายพันธุ์ต่อไป และในแต่ละรุ่นจะมีการสุ่มคำตอบที่เป็นไปได้ทั้งหมดของปัญหา

2.8 องค์ประกอบของอัลกอริทึมทางพันธุศาสตร์

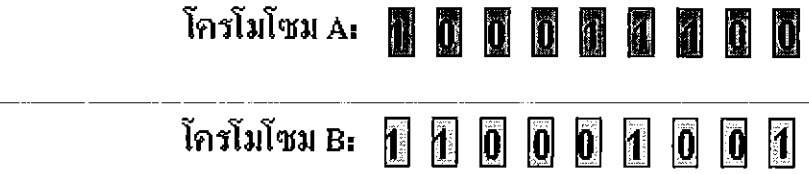
Genetic Algorithm มีองค์ประกอบที่สำคัญ 5 ส่วนด้วยกัน ดังนี้ คือ Chromosome Encoding, Initial Population, Fitness Function, Generic Operator และ Parameter

2.8.1 Chromosome Encoding (การเข้ารหัสโครโมโซม)

Chromosome encoding หรือการเข้ารหัสโครโมโซมที่ใช้ในการนำเสนอทางเลือกที่สามารถเป็นไปได้ของแต่ละปัญหา ซึ่งขึ้นอยู่กับปัญหา และในปัจจุบันมีปัญหามากมายจึงทำให้โครโมโซมมีความแตกต่างกันออกไปตามปัญหานั้น ดังนี้

1. Binary Encoding

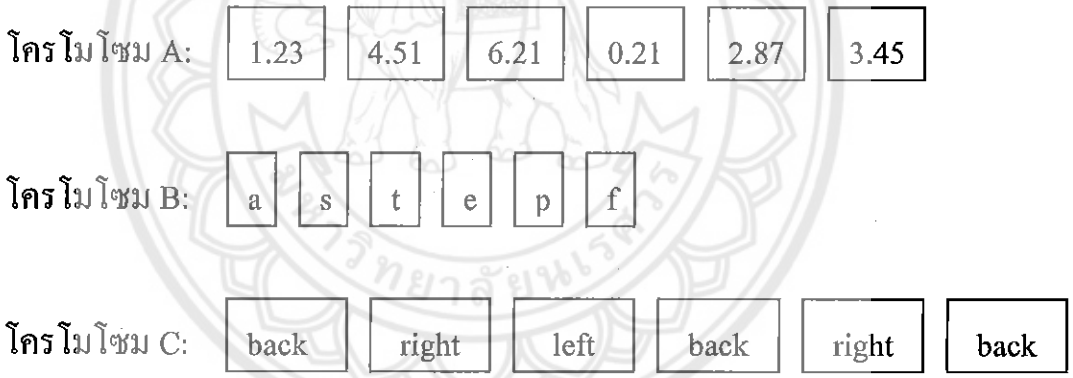
เป็นการเข้ารหัสโครโมโซมเริ่มแรกที่นำมาใช้ในการแก้ปัญหาของ GA ซึ่งลักษณะของ Binary Encoding คือ ทุกตำแหน่งของยีนในโครโมโซมจะมีค่าเป็น 0 หรือ 1



รูปที่ 2.8 ตัวอย่างโครโมโซมแบบ Binary Encoding

2. Value Encoding

Value Encoding หรือ Direct Encoding ทุกตำแหน่งของยีนในโครโมโซมจะมีค่าบางค่าซึ่งสามารถเชื่อมโยงไปยังปัญหาได้ เช่น ตัวอักษร, จำนวนจริง, คำสั่ง หรืออื่นๆ รูปแบบโครโมโซมแบบนี้สามารถใช้ได้กับปัญหาที่ค่อนข้างซับซ้อน



รูปที่ 2.9 ตัวอย่างโครโมโซมแบบ Value Encoding

3. Permutation Encoding

รูปแบบโครโมโซมแบบนี้ใช้ในการลำดับของปัญหา ทุกตำแหน่งของยีนในโครโมโซมจะเป็นค่าของจำนวนนับที่แทนตำแหน่งในลำดับ

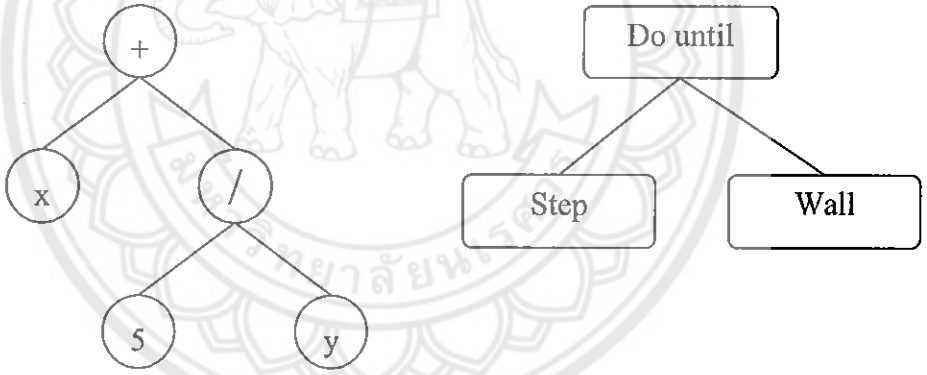
โครโมโซม A: 1 2 3 4 5 6 7 8 9

โครโมโซม B: 9 5 2 1 4 6 7 8 3

รูปที่ 2.10 ตัวอย่างโครโมโซมแบบ Permutation Encoding

4. Tree Encoding

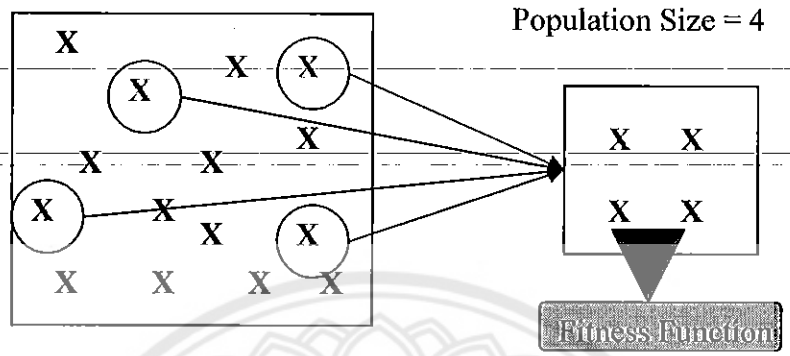
ถูกใช้ในการแก้ปัญหาสำหรับการพัฒนาโปรแกรมสำหรับ Genetic Programming เช่น ภาษา LISP รูปแบบของโครโมโซม คือ ทุกตำแหน่งของยีนจะเป็น node ของต้นไม้



รูปที่ 2.11 ตัวอย่างโครโมโซมแบบ Tree Encoding

2.8.2 Initial Population

คือ ประชากรต้นกำเนิดที่จะนำเข้าไปในกระบวนการถ่ายทอดลักษณะทางพันธุกรรม โดยทำการสุ่มขึ้นมาให้มีจำนวนเท่ากับขนาดของรุ่นที่ได้กำหนดไว้ โดยที่ยังไม่มีการสนใจค่าความเหมาะสมของแต่ละโครโมโซม



ข้อมูลทั้งหมด

รูปที่ 2.12 ตัวอย่างการสุ่มประชากรต้นกำเนิด

2.8.3 Fitness Function

โครโมโซมทุกตัวจะต้องมีค่าซึ่งบ่งบอกถึงความเหมาะสมที่จะพิจารณาว่าสมควรนำไปสืบพันธ์ต่อหรือไม่ ดังนั้น จึงต้องมีการให้ค่าความเหมาะสมกับแต่ละโครโมโซม เพื่อนำค่าความเหมาะสมไปพิจารณา โดยใช้สมการที่สอดคล้องกับปัญหา

ตัวอย่างของฟังก์ชันหาค่าความเหมาะสม เช่น

ค่าความเหมาะสม = จำนวนของ bit 1 ทั้งหมดของโครโมโซม

โครโมโซม A: 1 0 0 0 1 1 1 0 0

ดังนั้น โครโมโซม A มีค่าความเหมาะสมเท่ากับ 4

โครโมโซม B: 1 1 0 1 1 1 0 0 1

ดังนั้น โครโมโซม B มีค่าความเหมาะสมเท่ากับ 6

2.8.4 Generic Operator

Generic Operator ใช้ในการปรับเปลี่ยนองค์ประกอบของข้อมูลตลอดขบวนการ ซึ่งถือได้ว่าเป็นหัวใจสำคัญของ GA ซึ่งมีกระบวนการพื้นฐานที่สำคัญ มี 3 ส่วน ดังนี้

1. Selection – การคัดเลือก

ในการคัดเลือกโครโมโซมเพื่อที่จะนำมาเป็น Parent ในการสืบสายพันธุ์นั้นมีรูปแบบมากมายในการคัดเลือกโครโมโซมที่น่าพอใจที่สุดเพื่อนำไปสืบพันธ์

2. Crossover – การข้ามสายพันธุ์

การ Crossover เป็นกระบวนการที่สำคัญของ GA ซึ่งเมื่อเกิดการ crossover แล้ว จะทำให้เกิดความหลากหลายของคำตอบ จึงสามารถเลือกเอาคำตอบที่เหมาะสมกับความต้องการมากที่สุด ขั้นตอนในการ crossover นำ 2 โครโมโซมที่เป็น parent มาผสมกันเพื่อให้ได้โครโมโซมใหม่ขึ้น วิธีการ คือ สุ่มตำแหน่ง Crossover จากนั้นทำการคัดลอกทุกอย่างที่อยู่หน้าตำแหน่ง Crossover ของพ่อ และคัดลอกทุกอย่างที่อยู่หลังตำแหน่ง Crossover ของแม่ รวมกันจะได้ลูกตัวที่ 1 จากนั้นทำการคัดลอกทุกย่างที่อยู่หลังตำแหน่ง Crossover ของพ่อ และคัดลอกทุกอย่างที่อยู่หน้าตำแหน่ง Crossover ของแม่ รวมกันจะได้ลูกตัวที่ 2

รูปที่ 2.13 แสดงการ Crossover

3. Mutation - การกลายพันธุ์

เป็นกระบวนการที่เกิดหลังจากการ Crossover เสร็จสิ้น นั่นคือ การนำรุ่นลูกที่เกิดจากการผสมพันธ์ของ Parent มาทำการ Mutation ซึ่งจะทำให้เกิดการเปลี่ยนแปลงหรือทำให้เกิดลักษณะใหม่ๆ ขึ้น ขั้นตอนการ Mutation จะสุ่มหาค่าตำแหน่งที่ Mutation แล้วเปลี่ยนค่า ณ ตำแหน่งที่สุ่มนั้น

รูปที่ 2.14 แสดงการ Mutation

สำหรับการ Mutation นั้น สามารถเกิดได้มากกว่า 2 ตำแหน่ง ขึ้นอยู่กับการสุ่มที่ที่อยู่ภายใต้ความน่าจะเป็นของ Mutation

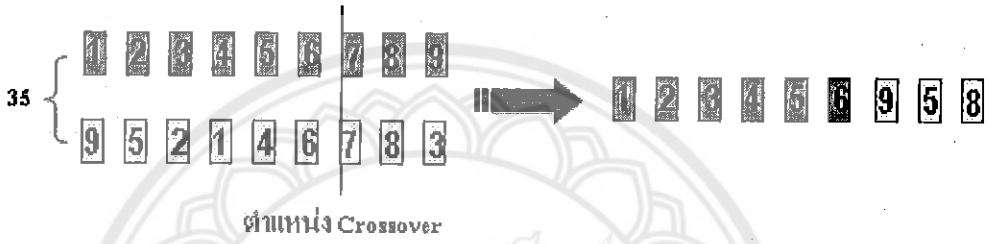
2.8.5 Parameter

พารามิเตอร์ที่สำคัญหรือเป็นพื้นฐานของ Genetic Algorithm มี 3 ตัว คือ

1. Crossover Probability

คือ ความน่าจะเป็นในการ Crossover ซึ่งจะมีค่าตั้งแต่ช่วง 0-100

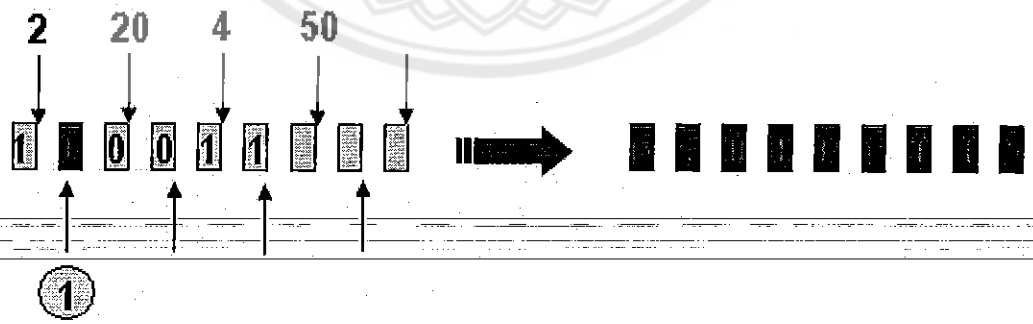
ตัวอย่างเช่นการเกิด Crossover กำหนดความน่าจะเป็นในการเกิด Crossover เป็น 85% ค่าที่สุ่มอยู่ในช่วง 0-100 ถ้ามีค่าน้อยกว่าหรือเท่ากับ 85 ก็จะเกิดการ Crossover นอกนั้นจะไม่เกิด



รูปที่ 2.15 แสดงโอกาสเกิด Crossover

2. Mutation Probability คือ ความน่าจะเป็นในการ Mutation มีค่าตั้งแต่ 0-100 ถ้าไม่มีการ Mutation แสดงว่าผลที่ได้เกิดจากการ Crossover เพียงอย่างเดียว

ตัวอย่างการเกิด Mutation โดยกำหนดความน่าจะเป็นของการเกิด mutation เท่ากับ 1 % ดังนั้น ค่าที่ทำการสุ่มจาก 0-100 ถ้ามีค่าน้อยกว่าหรือเท่ากับ 1 ก็จะเกิดการ mutation



รูปที่ 2.16 แสดงโอกาสเกิด Mutation

3. Population size หรือจำนวนโครโมโซมของแต่ละรุ่น

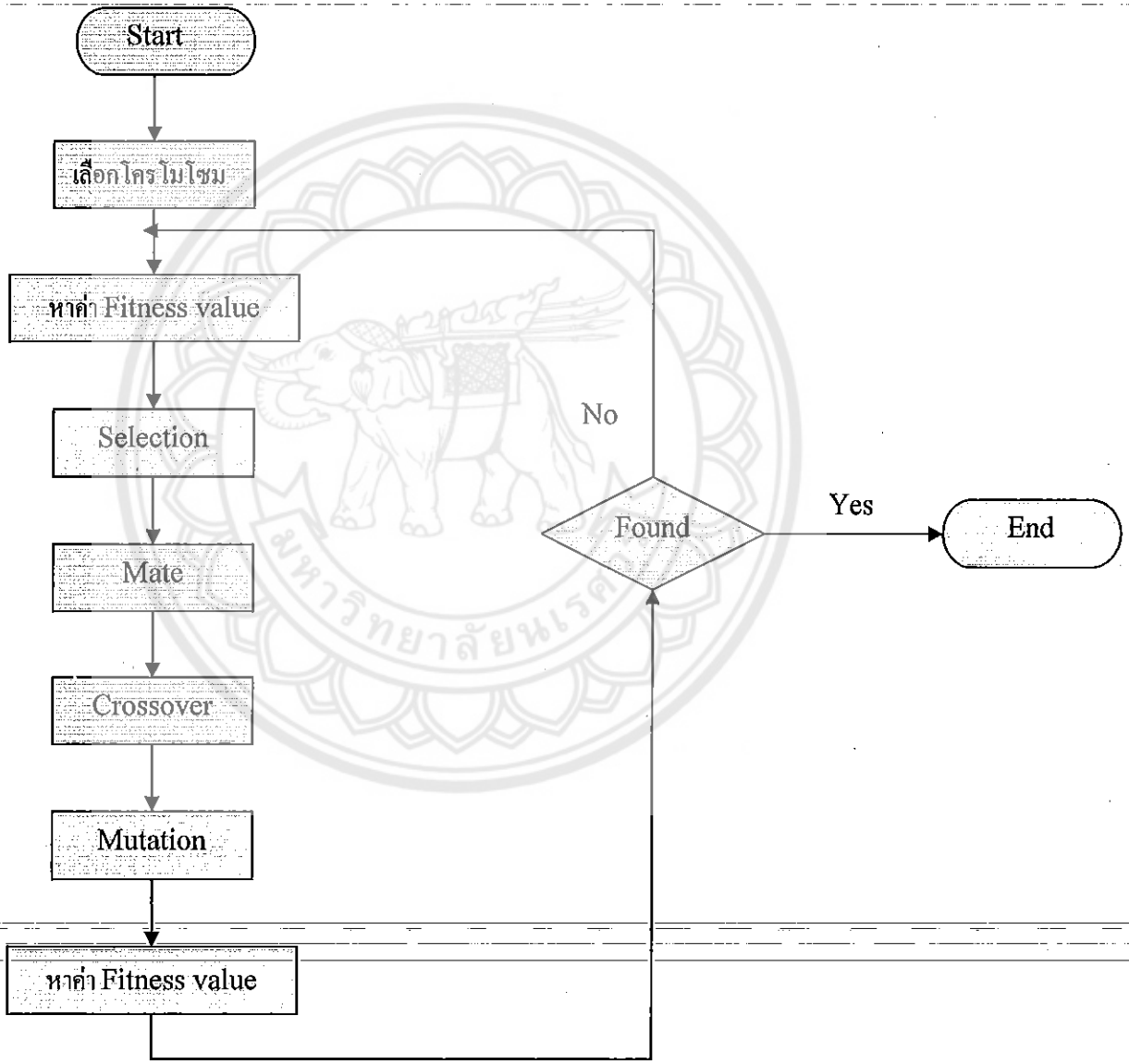
ถ้าจำนวนของ Population size มีจำนวนมากจะทำให้ Genetic Algorithm ประมวลผลช้าลง

2.9 เงื่อนไขในการหยุดกระบวนการหาคำตอบ

ในการหยุดวิธีการหาคำตอบของ Genetic Algorithm มีได้หลายวิธีด้วยกัน คือ

- ครบจำนวนรอบที่กำหนดไว้
- พบเป้าหมายหรือคำตอบที่ต้องการ
- พบคำตอบที่ใกล้เคียงกับที่ต้องการ

2.10 ขั้นตอนการทำงาน



รูปที่ 2.17 แสดงขั้นตอนการทำงานของ Genetic Algorithm

บทที่ 3
วิธีการดำเนินการ

3.1 การตรวจสอบและประเมิน Pattern ว่าเป็น Design Pattern ชนิดใด

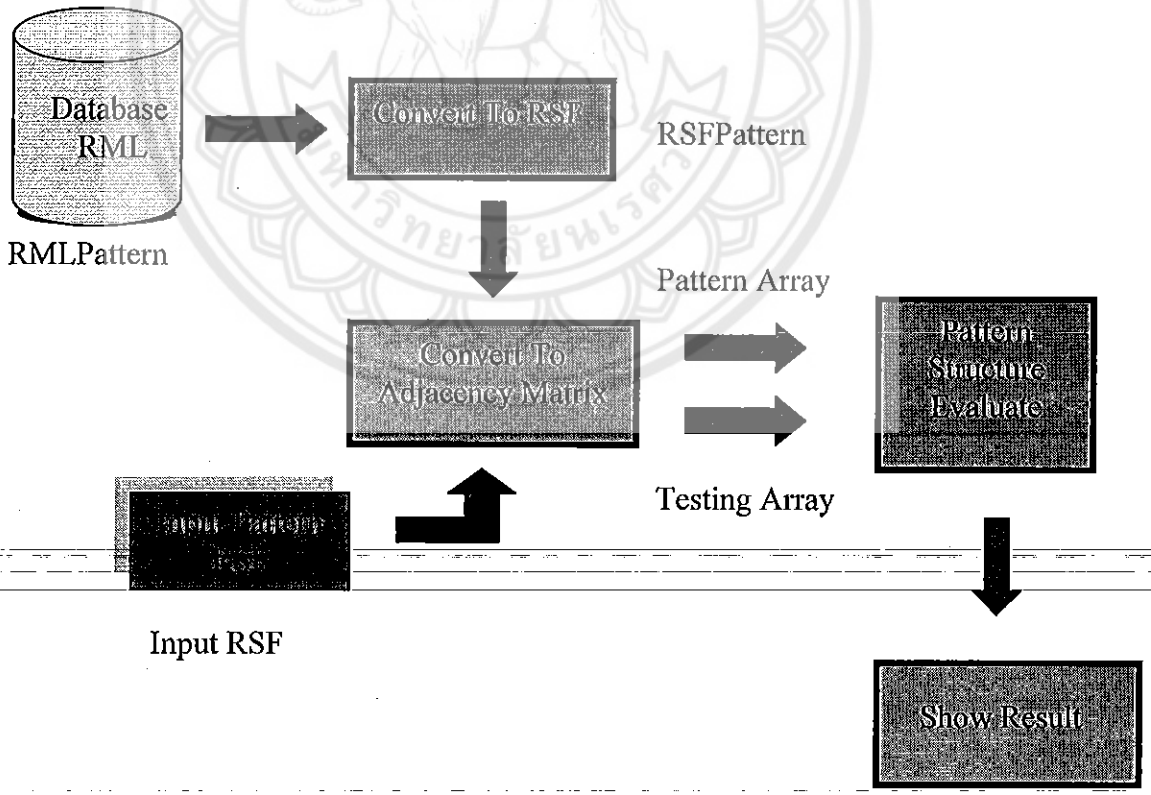
ขั้นตอนการตรวจสอบและประเมิน Design Pattern มีดังนี้

3.1.1 อ่านข้อมูล Design Pattern จากฐานข้อมูลที่เป็นภาษา RML แล้วแปลงข้อมูลนี้เป็น RSF

3.1.2 อ่านข้อมูล Input Pattern ที่เป็น RSF เข้ามาแล้วทำการแปลงข้อมูลนี้ให้อยู่ในรูปกราฟโดยใช้ Adjacency Matrix ในการเก็บข้อมูล

3.2.3 นำ Adjacency Matrix ไปตรวจสอบกับ Design Pattern จาก Database ทีละ Pattern โดยในขั้นตอนนี้จะมีการแปลง Design Pattern จากฐานข้อมูลให้อยู่ในรูปแบบกราฟโดยใช้ Adjacency Matrix ด้วย หากตรวจสอบได้ว่ามีโครงสร้างเหมือนกันก็จะบันทึกว่า Input Pattern มีโครงสร้างเหมือนกับ Design Pattern อะไร

3.2.4 แสดงผลลัพธ์ในการตรวจสอบ



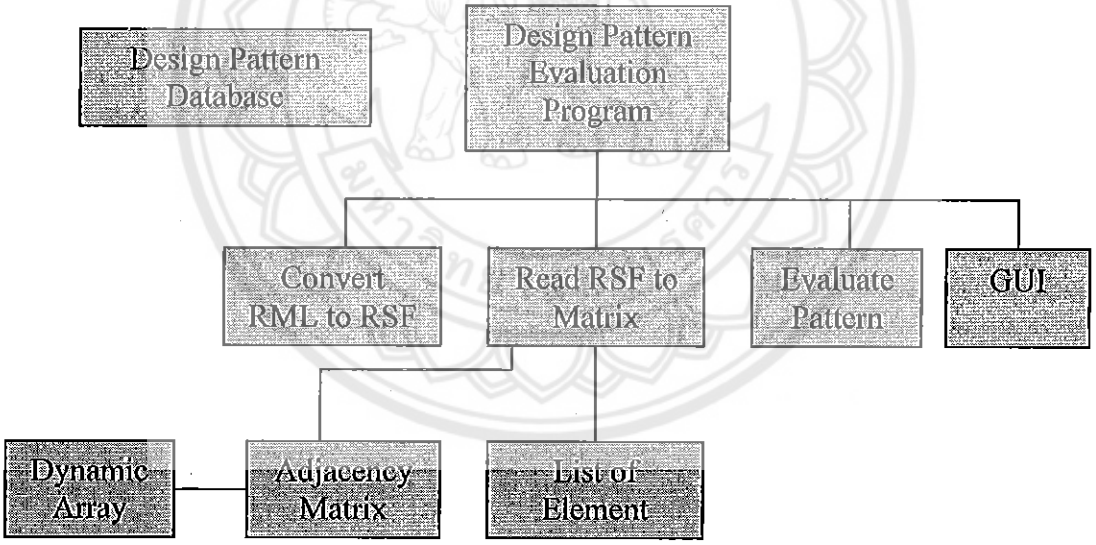
รูปที่ 3.1 แสดงผังการทำงานของการประเมิน Design Pattern

3.2 การออกแบบโปรแกรมประเมิน Design Pattern

ในการออกแบบโปรแกรม ได้แบ่งโปรแกรมออกเป็นส่วนต่างๆ โดยใช้หลักการของ Modular decomposition การออกแบบโปรแกรมจึงเน้นไปที่หน้าที่การทำงานของแต่ละ Module ซึ่งสามารถจำแนกส่วนประกอบหลักต่างๆ ของโปรแกรมได้ดังนี้

1. ส่วนควบคุมการแสดงผลและติดต่อกับผู้ใช้
2. ส่วนแปลงข้อมูลจาก RML เป็น RSF
3. ส่วนแปลงข้อมูลจาก RSF เป็น Adjacency Matrix
4. ส่วนประเมิน Design Pattern
5. ส่วนควบคุมและดำเนินการของ Adjacency Matrix
6. ส่วนฐานข้อมูลของ Design Pattern

3.3 ผังแสดงโครงสร้างส่วนประกอบของโปรแกรม

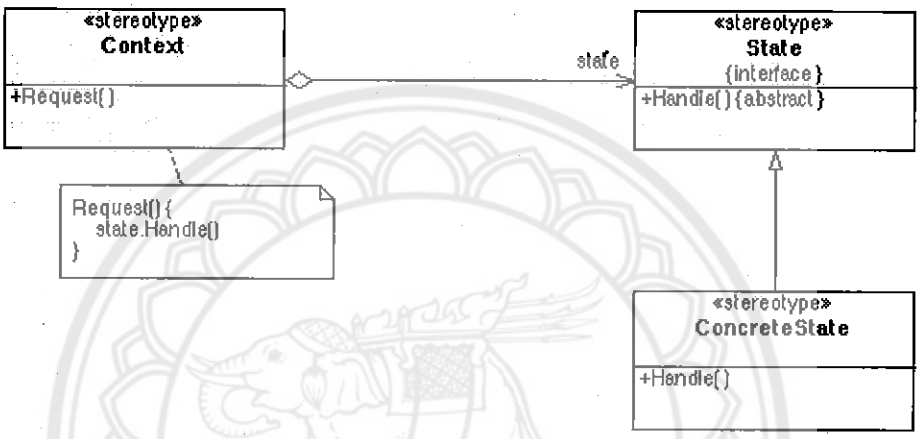


รูปที่ 3.2 ผังแสดงส่วนประกอบของโปรแกรม

3.4 การเตรียมข้อมูลในฐานข้อมูล

ในกระบวนการแรกข้อมูลในฐานข้อมูลจะอยู่ในรูปแบบของภาษา RML เพราะว่าภาษา RML สามารถบรรยายโครงสร้างของ Pattern ได้เข้าใจและครอบคลุมมากกว่า RSF ซึ่งเราจะต้องศึกษารูปแบบของ Design Pattern และภาษา RML เพื่อให้รู้ถึงการบรรยายโครงสร้างโดยใช้ Pattern Language นี้ และสามารถเพิ่ม Design Pattern ที่ต้องการตรวจสอบไปยังฐานข้อมูลได้

ตัวอย่าง ของ RML ที่ แสดงโครงสร้างของ State Pattern



รูปที่ 3. 3 แสดงโครงสร้างของ State Pattern

ภาษา RML ที่ใช้ในการบรรยายโครงสร้างและความสัมพันธ์

ComPat (State, ConcreteState, Context) := Inherit (ConcreteState, State)
& Contain (Context, State);

หลังจากการศึกษาเราได้ทำการเพิ่ม Design Pattern แบบต่างๆ เพิ่มในฐานข้อมูล เช่น Composite Pattern แบบต่างๆ เข้าไป

การเตรียมข้อมูลในฐานข้อมูลจึงทำให้เราสามารถที่จะเพิ่ม Pattern รูปแบบต่างๆ ลงไปในฐานข้อมูลเพื่อใช้ในการตรวจสอบได้สะดวกและรวดเร็วมากยิ่งขึ้น

3.5 การแปลง RML เป็น RSF

เนื่องจาก Design Pattern ในฐานข้อมูลของเป็น RML แต่ Input Pattern เป็น RSF ทำให้เมื่อเราจะนำ Input Pattern ไปประเมินกับ Design Pattern ต่างๆ ในฐานข้อมูลจำเป็นต้องมีการแปลงข้อมูลจาก RML ไปเป็น RSF ก่อนที่จะทำการตรวจสอบ

ข้อแตกต่าง ระหว่างภาษา RML กับ RSF คือภาษา RML จะบอกว่าโครงสร้างนั้นประกอบไปด้วยส่วนประกอบอะไรบ้างด้วยคำว่า ComPat และจะมีเครื่องหมายที่บ่งบอกถึงความหมายต่างๆ ดังนั้นหากเราตัดคำและเครื่องหมายเหล่านี้ออกไป ก็จะได้ RSF นั่นเอง

ตัวอย่างแสดงการแปลงจาก RML ไปเป็น RSF

RML ของ Composite Pattern แบบที่ 1

```
ComPat (Component, Leaf, Composite) :=
                Inherit (Composite, Component)
              & Contain (Composite, Component)
              & Inherit (Leaf, Component)
              & ! Contain (Leaf, Component);
```

เมื่อผ่านการแปลงข้อมูลแล้วจะได้ผลลัพธ์ดังนี้

```
Inherit Composite Component
Contain Composite Component
Inherit Leaf Component
```

เมื่อเราแปลงแต่ละ Design Pattern ในฐานข้อมูลให้เป็น RSF ได้แล้วนั้น เราสามารถที่จะนำข้อมูลอันนี้ไป Encode เป็นโครงสร้างของกราฟโดยใช้ Adjacency Matrix ได้ ซึ่งพร้อมที่จะเข้าสู่ขั้นตอนการประเมิน Design Pattern แล้ว

3.6 วิธีการตรวจสอบ Design Pattern

เมื่อได้ข้อมูล Design Pattern และข้อมูลที่จะตรวจสอบอยู่ในรูปแบบ RSF แล้ว เราจะทำการตรวจสอบโครงสร้างของ Pattern ที่ต้องการตรวจสอบว่ามีโครงสร้างเหมือนกับ Design Pattern แบบใดบ้าง โดยกระบวนการตรวจสอบมีวิธีการดังนี้

ขั้นตอนที่ 1 แปลงข้อมูลในส่วน Input Pattern จาก Text File (RSF) ให้อยู่ในรูปแบบกราฟ โดยใช้Adjacency Matrix

1.1. นับจำนวน ส่วนประกอบของโครงสร้างทั้งหมดที่มีจำนวน N

1.2. สร้าง Array ขนาด N × N เพื่อเก็บ Adjacency Matrix พร้อมระบุตำแหน่งว่าส่วนประกอบของโครงสร้างใดอยู่ ที่ตำแหน่ง Index ที่เท่าไหร่ของ Array

1.3. ตรวจสอบคำบรรยายลักษณะโครงสร้างของภาษา RSF เพื่อสร้างรูปแบบการเชื่อมต่อกันใน Adjacency Matrix ซึ่ง Key Word มีดังนี้

CONTAIN	ตัวเลขแสดงการเชื่อมต่อของโครงสร้าง	3
INHERIT	ตัวเลขแสดงการเชื่อมต่อของโครงสร้าง	1
USE	ตัวเลขแสดงการเชื่อมต่อของโครงสร้าง	5
CREATES	ตัวเลขแสดงการเชื่อมต่อของโครงสร้าง	13
PRODUCT	ตัวเลขแสดงการเชื่อมต่อของโครงสร้าง	11
BUILDS	ตัวเลขแสดงการเชื่อมต่อของโครงสร้าง	21

1.4 ใส่เลขแสดงรูปแบบการเชื่อมต่อลงใน Adjacency Matrix ลงใน Array เช่น

Composite Pattern

Inherit Composite Component

Contain Composite Component

Inherit Leaf Component

จำนวนส่วนประกอบของโครงสร้าง N = 3 ประกอบด้วย Composite, Component, Leaf เราจะได้ Adjacency Matrix ขนาด 3 × 3 โดย แต่ละ Index แสดงถึงส่วนของโครงสร้างดังนี้

Index [0] แทนส่วนของ Composite

Index [1] แทนส่วนของ Component

Index [2] แมนส่วนของ Leaf

โดยในการแทน Index นั้น ส่วนของโครงสร้างจะแทนทั้งในแนวแถวและหลัก โดยเมื่อใส่ค่าตัวเลขแสดงการเชื่อมต่อแล้วสามารถแสดงผลเป็นดังนี้

	Composite	Component	Leaf
Composite	0	1+3	0
Componen	0	0	0
Leaf	0	1	0

รูปที่ 3. 4 แสดง Adjacency Matrix ที่ใช้แทนส่วนประกอบและการเชื่อมต่อของโครงสร้าง

ขั้นตอนที่ 2 นำ Design Pattern ใน Database ที่มีจำนวนส่วนประกอบของโครงสร้างเท่ากันมาตรวจสอบ โดย Design Pattern ใดมีจำนวนส่วนประกอบไม่เท่ากับจำนวนส่วนประกอบของ Input Pattern แสดงว่าเป็นคนละ Pattern กัน

ขั้นตอนที่ 3 แปลง Design Pattern ที่นำมาจาก Database ให้อยู่ในรูปแบบ Adjacency Matrix โดยใช้วิธีการเดียวกันกับข้อที่ 1

ขั้นตอนที่ 4 ตรวจสอบโครงสร้างของ Pattern จาก Adjacency Matrix ของ Input Pattern กับ Design Pattern จาก Database ซึ่งมี 3 กรณีคือ

1. Element ใน Adjacency Matrix เหมือนกันทุกประการทั้งค่าตัวเลขและตำแหน่ง แสดงว่า Input Pattern ที่นำมาตรวจสอบมีโครงสร้างเหมือนกับ Design Pattern นั้น
2. Element ใน Adjacency Matrix มีค่าตัวเลขเหมือนกันแต่อยู่ไม่ตรงตำแหน่งกัน เราจะต้องทำสอบ Isomorphism ซึ่งเป็นการทดสอบความเหมือนกันของโครงสร้าง โดยใช้หลักการของ Genetic Algorithm เข้ามาช่วย โดยวิธีการโดยละเอียดนั้นจะบรรยายอยู่ในหัวข้อของ Genetic Algorithm
3. Element ใน Adjacency Matrix มีค่าตัวเลขบางตัวแตกต่างกัน แสดงว่า Input Pattern ที่นำมาตรวจสอบมีโครงสร้างไม่เหมือนกับ Design Pattern นั้น

ขั้นตอนที่ 5 ตรวจสอบ Design Pattern อื่นๆ ใน Database อีก โดยกลับไปทำข้อ 2 จนกว่าจะตรวจสอบกับทุก Design Pattern ใน Database ครบทุกอัน

3.7 วิธีการนำ Genetic Algorithm มาช่วยในการตรวจสอบโครงสร้างของ Pattern

Genetic Algorithm (GA) เป็นวิธีการแก้ปัญหาอย่างหนึ่งที่ช่วยให้ได้มาซึ่งคำตอบที่เหมาะสมที่สุดให้กับปัญหา

ในการตรวจว่า Adjacency Matrix 2 อันมีคุณสมบัติ Isomorphism คือ มีโครงสร้างเหมือนกันหรือไม่นั้น สามารถใช้วิธีการดำเนินการตามแถวและหลัก (Row and Column Operation) ได้ ซึ่งเราจะนำ Genetic Algorithm เข้ามาช่วยในการดำเนินการตามแถวและหลักกับ Input Pattern ในกรณีที่มีสมาชิกทุกตัวใน Adjacency Matrix เหมือนกัน แต่อยู่คนละตำแหน่งกันในตอนแรก เพื่อดูว่าเมื่อผ่านกระบวนการดำเนินการตามแถวและหลักแล้ว สมาชิกของ Adjacency Matrix ทั้ง 2 เหมือนกันทุกตำแหน่งหรือไม่ ซึ่งถ้าเหมือนกันแสดงว่า Pattern ทั้งสองมีโครงสร้างแบบเดียวกัน และหากมีบางตำแหน่งไม่เหมือนกันแสดงว่า Pattern ทั้งสองมีโครงสร้างไม่เหมือนกันนั่นเอง

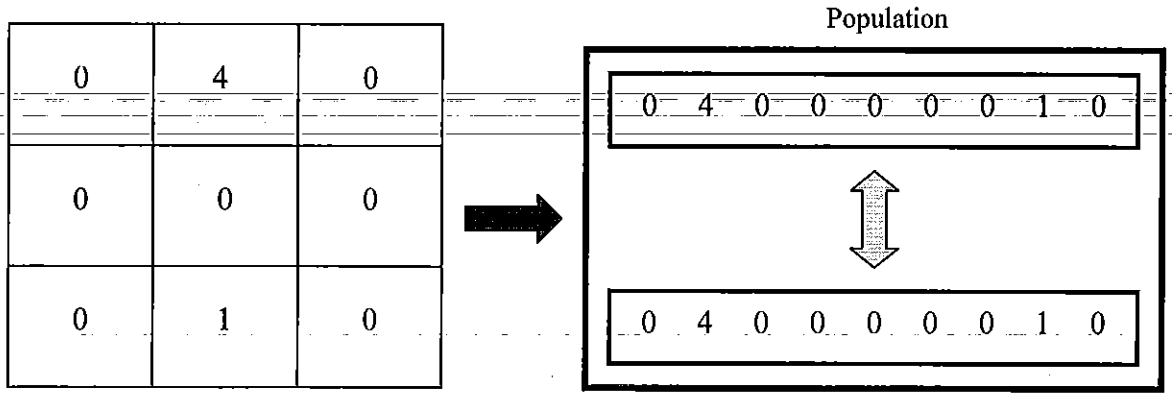
วิธีการนำ Genetic Algorithm มาใช้ในการดำเนินการตามแถวและหลัก

3.7.1 ค่าเริ่มต้นต่างๆ ของการทำงานโดยวิธี Genetic Algorithm

- ใช้ Chromosome จำนวน 500 ตัวต่อ 1 Population
- จำกัดจำนวนครั้งในการทำงานสูงสุด ที่ 20 ครั้ง
- ElitismRate จำนวน Chromosome ที่มีค่า Fitness ดีที่สุดที่คงไว้เป็น 10 %
- MutationRate โอกาสเกิดการสลับที่ตามแถวหรือหลัก 90%
- จำนวนของ Gene จะขึ้นอยู่กับจำนวนข้อมูลตามขนาดของ Adjacency Matrix

3.7.2 การกำหนดค่าเริ่มต้นให้แก่ทุกๆ Chromosome ใน Population

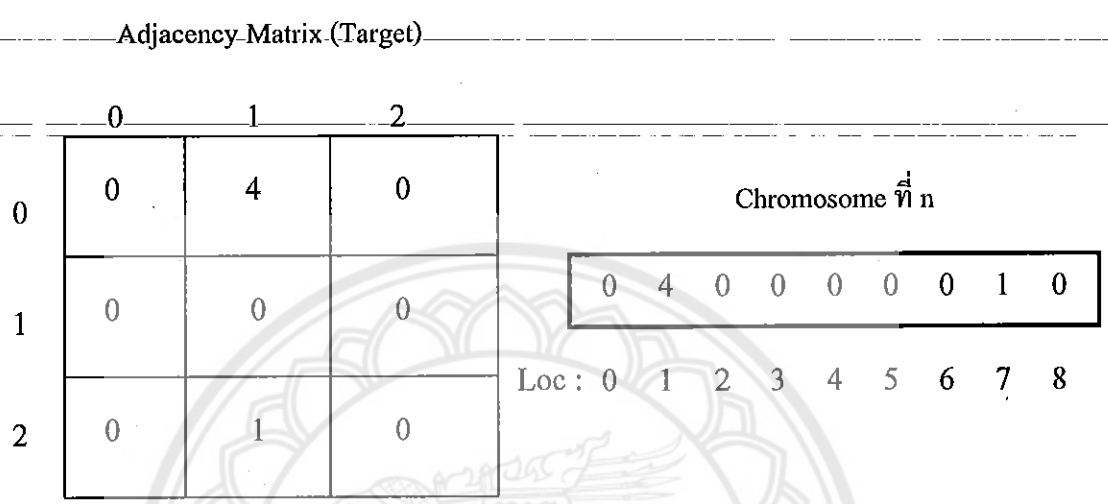
วิธีการ Encode นั้นเป็นการ Encode แบบ Value Encoding โดยจะนำสมาชิกแต่ละแถวจาก Adjacency Matrix ของ Input Pattern มาเรียงต่อกันจนครบ โดยจะให้ทุก Chromosome ในขั้นตอนนี้มีข้อมูลเหมือนกันหมด



รูปที่ 3.5 แสดงผลลัพธ์ Encode ข้อมูลของ Genetic Algorithm

3.7.3 การคำนวณหาค่า Fitness ให้แก่แต่ละ Chromosome

ใช้วิธีการ Sum Square Error ระหว่างค่าในแต่ละ Chromosome กับค่าใน Target โดย Target เป็น ค่าของ Adjacency Matrix ของ Design Pattern ที่นำมาตรวจสอบ ซึ่งถ้าหา ค่า Fitness มีค่าเป็น 0 แสดงว่า Adjacency Matrix ของ Input Pattern และ Design Pattern มีโครงสร้างเหมือนกัน



รูปที่ 3.6 แสดงวิธีการคำนวณหาค่า Fitness ให้แต่ละ Chromosome

จากรูปค่าตำแหน่งของสมาชิกอยู่ในรูปแบบแตกต่างกัน เราสามารถใช้วิธีการแปลงตำแหน่งของสมาชิกใน Adjacency Matrix เป็นค่าตำแหน่งของ Gene ใน Chromosome ที่ถูกต้องได้ดังนี้

$$\text{Location (Gene)} = (\text{Row} \times \text{Size of Matrix}) + \text{Col}$$

เช่น ค่าสมาชิก Adjacency Matrix (0, 1) มีค่า 4 เราจะเปรียบเทียบหา Error จากการนำมาลบกับ Gene ใน Chromosome ในตำแหน่งที่ 1 จากสูตร

$$\text{Location} = \text{Row} \times (\text{Size of Matrix}) + \text{Col}$$

$$= (0 \times 3) + 1$$

$$= 1$$

ซึ่งสามารถนำวิธีการนี้ใช้กับการแปลงตำแหน่ง ชุดของ Chromosome และ Adjacency Matrix ขนาด N × N ใดๆ เพื่อที่จะหา Sum Square Error ได้อย่างถูกต้อง

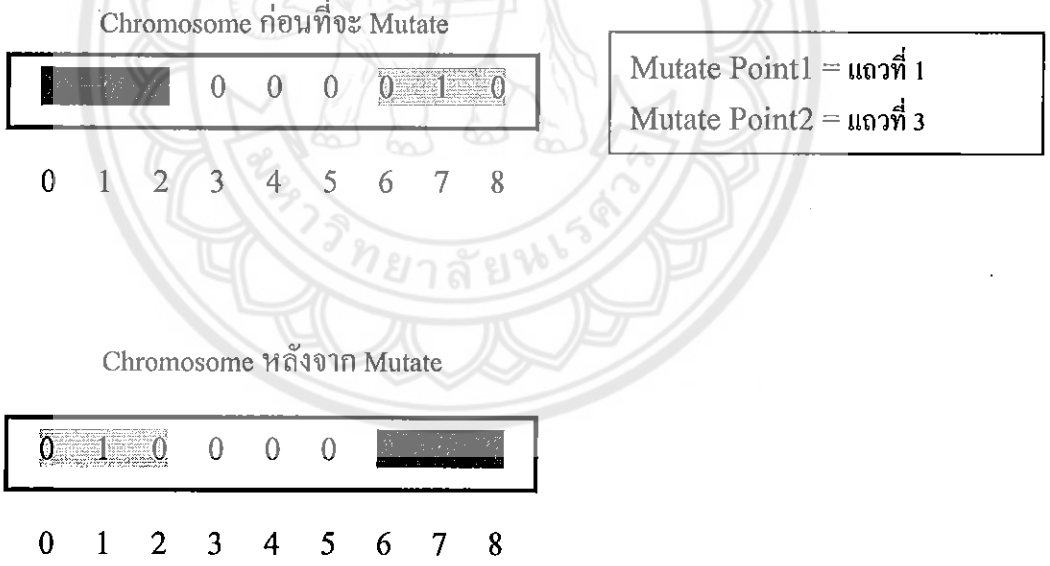
3.7.4 การ Mate เพื่อสุ่มการสลับที่ตามแถวและหลัก

เป็นการวิวัฒนาการจาก Population เดิมให้ไปสู่อีก Population ใหม่ที่มี Chromosome ที่แตกต่างไปจากเดิม ซึ่งอาจจะเป็น Chromosome ซึ่งเป็นคำตอบที่เราต้องการได้

ในกระบวนการนี้ เราจะไม่ใช้วิธีการ Cross Over แต่จะใช้วิธีการ Mutate เท่านั้น เพราะการ Cross Over จะทำให้ รูปแบบของ Chromosome ที่แสดงโครงสร้างของ Pattern เปลี่ยนแปลงไป โดยการ Mutate จะมี 2 แบบ คือ Mutate ตามแนวแถว และ Mutate ตามแนวคอลัมน์

แบบที่ 1 การ Mutate ตามแนวแถว เปรียบเสมือนการดำเนินการสลับที่ของแถวใน Adjacency Matrix แต่จะเป็นการดำเนินการภายใน Chromosome แทน โดยมีวิธีการดังนี้

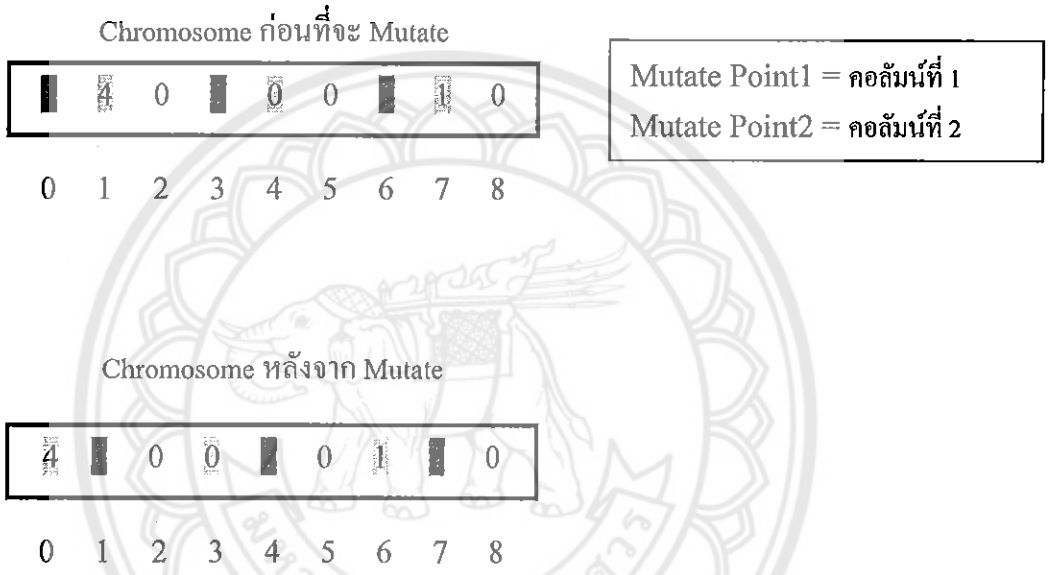
1. สุ่มคู่ของแถวที่จะสลับกัน
2. ทำการสลับค่าระหว่างทั้ง 2 แถว โดยสลับค่าที่อยู่ในแถวที่สุ่มได้แถวแรก คอลัมน์ แรก กับค่าที่อยู่ในแถวที่สุ่มได้แถวที่สองคอลัมน์แรก สลับไปเรื่อยๆ จนถึงแถวที่สุ่มได้แถว แรกคอลัมน์สุดท้ายกับแถวที่สุ่มได้แถวที่สองคอลัมน์สุดท้าย



รูปที่ 3.7 แสดงวิธีการ Mutate ตามแนวแถว

แบบที่ 2 การ Mutate ตามแนวคอลัมน์ เปรียบเสมือนการดำเนินการสลับที่ของคอลัมน์ใน Adjacency Matrix แต่จะเป็นการดำเนินการภายใน Chromosome แทน โดยมีวิธีการดังนี้

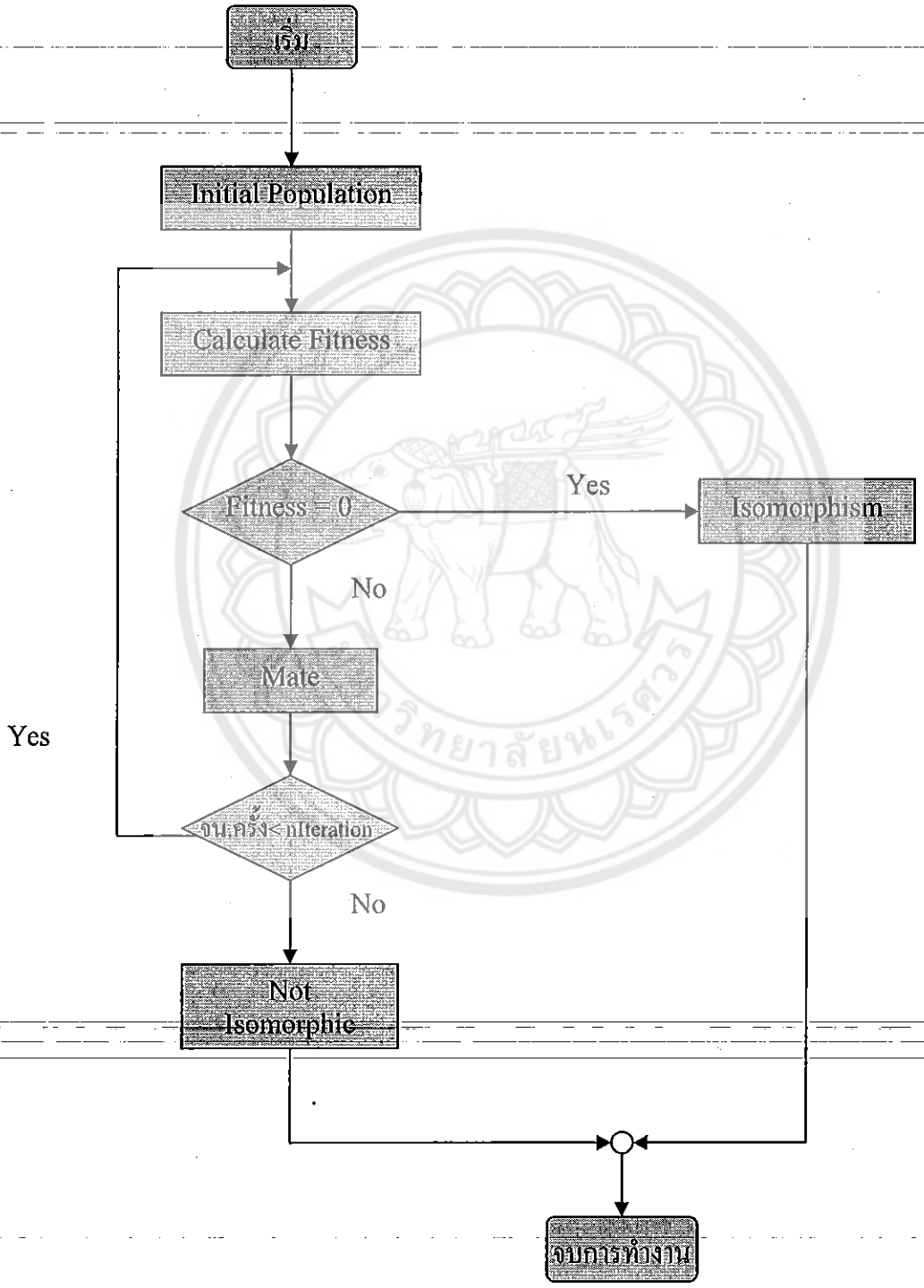
1. สุ่มคู่ของคอลัมน์ที่จะสลับกัน
2. ทำการสลับค่าระหว่างทั้ง 2 คอลัมน์ โดยสลับค่าที่อยู่ในคอลัมน์ ที่สุ่มได้คอลัมน์ แถวแรกกับค่าที่อยู่ในคอลัมน์ที่สุ่มได้คอลัมน์ที่สองแถวแรกแรก สลับไปเรื่อยๆ จนถึงคอลัมน์ ที่สุ่มได้คอลัมน์แรกแถวสุดท้ายกับคอลัมน์ที่สุ่มได้คอลัมน์ที่สองแถวสุดท้าย



รูปที่ 3.8 แสดงวิธีการ Mutate ตามแนวคอลัมน์

3.8 ขั้นตอนในการนำ Genetic Algorithm มาช่วยในการตรวจสอบ

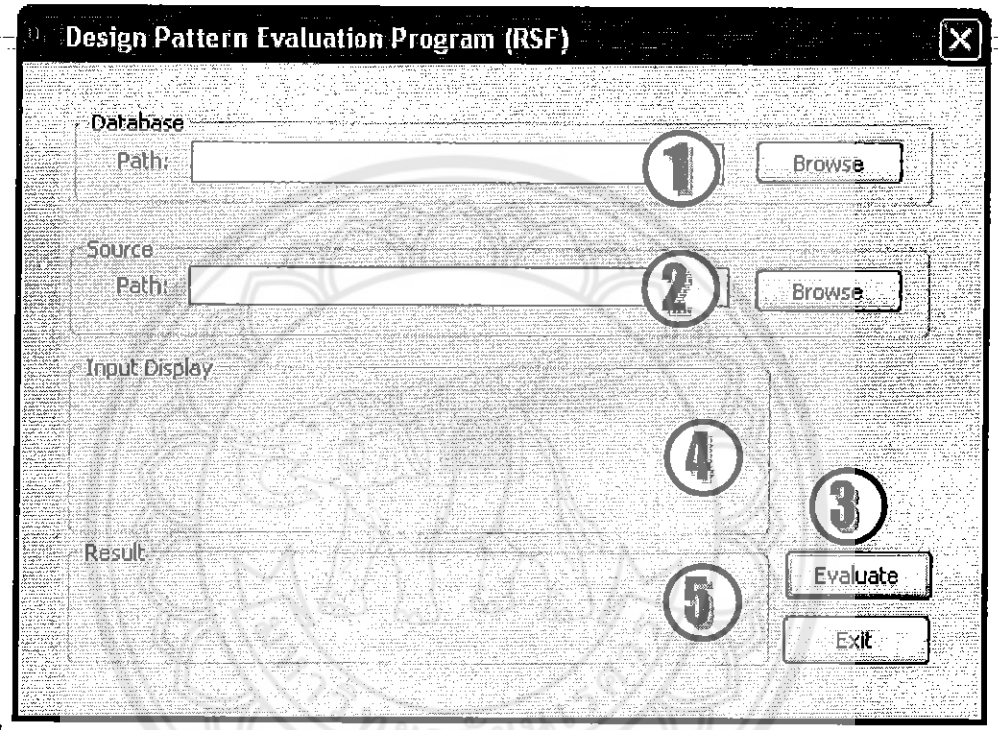
เมื่อการตรวจสอบในขั้นตอนแรกพบว่าตัวเลขใน Adjacency Matrix มีค่าเหมือนกันหมด ต่างกันแค่เพียงตำแหน่ง เราต้องนำ Adjacency Matrix ของ Input Pattern มาทำการสลับที่ตามแนวแถวหรือหลัก เพื่อดูว่า เมื่อผ่านกระบวนการนี้แล้ว ตำแหน่งของค่าตัวเลขตรงกันหรือไม่ โดยสามารถแสดงขั้นตอนการทำงานออกมาเป็นผังการทำงานนี้



รูปที่ 3.9 แสดงผังขั้นตอนในการนำ Genetic Algorithm มาใช้งาน

3.9 ส่วนติดต่อของโปรแกรมการตรวจสอบ Design Pattern จาก Source Code

การสร้างส่วนติดต่อของโปรแกรมการตรวจสอบ Design Pattern จาก Source Code จะมีรูปแบบเป็นวินโดวส์โดยอาศัยลักษณะพิเศษในโปรแกรม Visual Studio 2005 และ MFC ที่มีความสามารถในการสร้างส่วนติดต่อกับผู้ใช้ โดยการรวมเอาทรัพยากรต่างๆ เช่น ปุ่มกด, ช่องรับข้ออวาม, เมนูต่างๆ เป็นต้น มาทำงานร่วมกับโปรแกรมการตรวจสอบ Design Pattern จาก Source Code ได้อย่างกลมกลืน



รูปที่ 3.10 แสดงส่วนการติดต่อกับผู้ใช้ของโปรแกรมตรวจสอบ Design Pattern จาก Source Code

จากรูปที่ 3.1 แสดงให้เห็นว่าส่วนติดต่อกับผู้ใช้นั้น ประกอบด้วยหลายๆ ส่วนด้วยกัน คือ

ส่วนที่ 1 Database จะเป็นส่วนที่ใช้ในการรับ RML ของ Design Pattern ที่รวบรวมรูปแบบโครงสร้างของ Design Pattern หลายๆ แบบไว้ เพื่อนำมาเปรียบเทียบกับ Input Pattern

ส่วนที่ 2 Source เป็นส่วนที่นำเอา RSF ที่อยู่ในรูปของเท็กซ์ไฟล์ (Input Pattern) ที่บรรยายถึงโครงสร้างของโปรแกรมเพื่อนำมาตรวจสอบกับ Design Pattern ใน Database ทั้งหมด

ส่วนที่ 3 Evaluate ปุ่ม Evaluate จะส่ง Input ที่ได้รับ คือ Design Pattern ใน Database และ Input Pattern ไปให้โปรแกรมตรวจสอบ Design Pattern ประมวลผลออกมาว่า Source Pattern เป็น Design Pattern แบบใด

ส่วนที่ 4 Input Display แสดง RSF ของ Input Patternที่ต้องการนำมาตรวจสอบ Design Pattern

ส่วนที่ 5 Result แสดงว่า Input Pattern ที่เรานำไปตรวจสอบว่าเป็น Design Pattern ชนิดใด

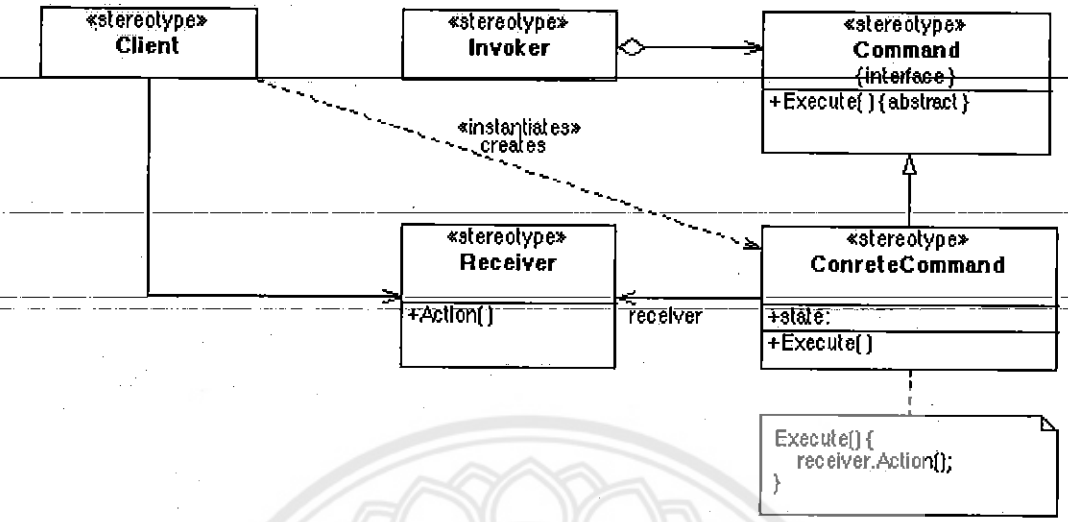
บทที่ 4
ผลการทดลอง

4.1 ผลการทดลองการเปรียบเทียบระหว่าง โครงสร้างของ Design Pattern ในฐานข้อมูลที่เก็บอยู่ในรูปของ RML กับ RSF ของ Pattern ที่นำมาตรวจสอบ

ในการทดลองนั้น ต้องให้อินพุตกับโปรแกรมสองอย่างด้วยกัน คือ RSF ของ Pattern ที่จะนำมาตรวจสอบซึ่งอยู่ในรูปของเท็กซ์ไฟล์ ซึ่งในแต่ละไฟล์นั้นจะประกอบไปด้วย Pattern ที่ต้องการตรวจสอบเพียง Pattern เดียวเท่านั้น อินพุตอีกอย่างหนึ่งก็คือ โครงสร้างของ Design Pattern ที่เก็บไว้ใน ฐานข้อมูลโดยอยู่ในรูปแบบ RML ซึ่งจะเก็บอยู่ในรูปของเท็กซ์ไฟล์เช่นกัน ในเท็กซ์ไฟล์ของฐานข้อมูลนี้ จะประกอบไปด้วย โครงสร้างของทุก Design Pattern

เมื่อโปรแกรมได้รับอินพุตทั้งสองอย่างแล้วจะนำ RSF ของ Pattern ที่ต้องการตรวจสอบไปเปรียบเทียบกับ Design Pattern ที่เป็นฐานข้อมูลทุก Design Pattern แต่เนื่องจาก Design Pattern ที่อยู่ใน ฐานข้อมูล อยู่ในถูกเก็บในรูปของ RML ดังนั้น โปรแกรมจะต้องเปลี่ยนเป็น RSF ก่อนเพื่อที่จะสามารถนำไปเปรียบเทียบกับ RSF ของ Pattern ที่นำมาตรวจสอบได้ เมื่อพบโครงสร้างของ Design Pattern ที่เป็นฐานข้อมูลที่ตรงกันกับ Pattern ที่ต้องการตรวจสอบ ก็จะแสดงชื่อ Design Pattern ชนิดนั้นออกมา

การตรวจสอบ Command Pattern



รูปที่ 4.1 โครงสร้าง Command Pattern

โครงสร้าง ของ Command Pattern ที่อยู่ใน ฐานข้อมูล คือ

ComPat (Command, Invoker, ConreteCommand, Receiver, Client) :=

Contain (Invoker, Command)

& Inherit (ConreteCommand, Command)

& Use (ConreteCommand, Receiver)

& Use (Client, Receiver)

& Creates (Client, ConreteCommand);

RSF ของ Pattern ที่นำมาตรวจสอบ

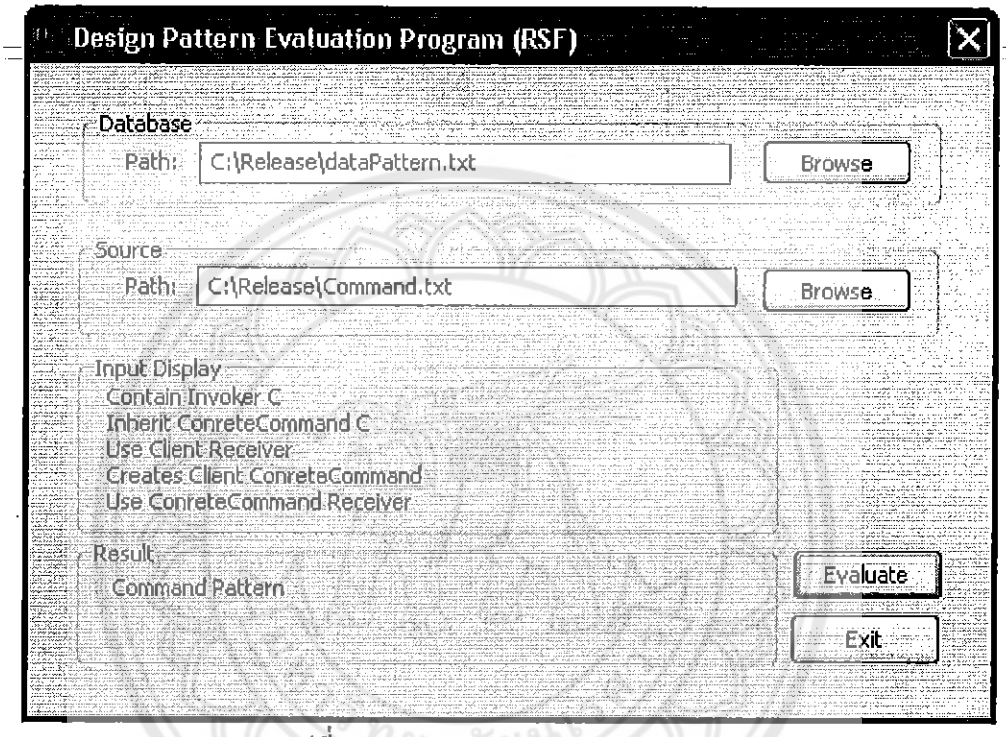
Contain Invoker Command

Inherit ConreteCommand Command

Use Client Receiver

Creates Client ConreteCommand

Use ConreteCommand Receiver



รูปที่ 4.2 แสดงผลการตรวจสอบ Pattern

จากรูปที่ 4.2 RSF ของ Pattern ที่นำมาตรวจสอบนั้น มีลำดับการเขียนและใช้ชื่อส่วนประกอบเหมือนกันกับ Command Pattern ในฐานข้อมูล เมื่อตรวจสอบแล้ว Pattern ที่นำมาตรวจสอบนั้นจึงเป็น Command Pattern

RSF ของ Pattern ที่นำมาตรวจสอบ

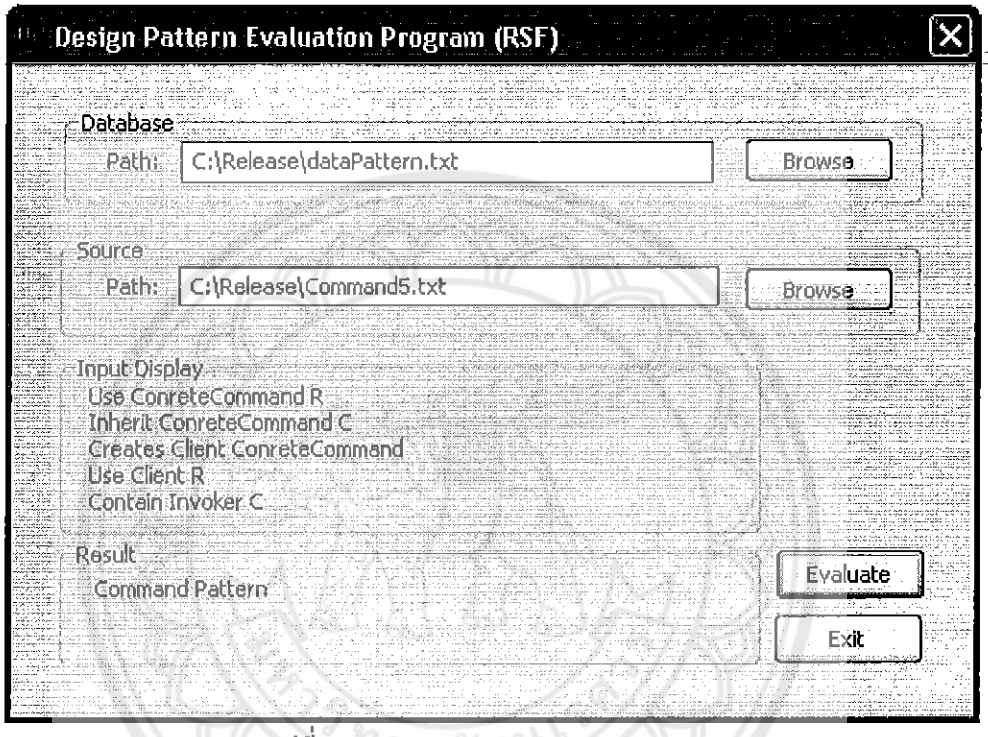
Use ConreteCommand R

Inherit ConreteCommand C

Creates Client ConreteCommand

Use Client R

Contain Invoker C



รูปที่ 4.3 แสดงผลการตรวจสอบ Pattern

จากรูปที่ 4.3 RSF ของ Pattern ที่นำมาตรวจสอบกับ Design Pattern ในฐานข้อมูล พบว่าเป็น Command Pattern ถึงแม้ว่า RSF นั้นจะมีการใช้ชื่อส่วนประกอบและลำดับการเขียนต่างกัน แต่มีความสัมพันธ์ของส่วนประกอบเหมือนกัน

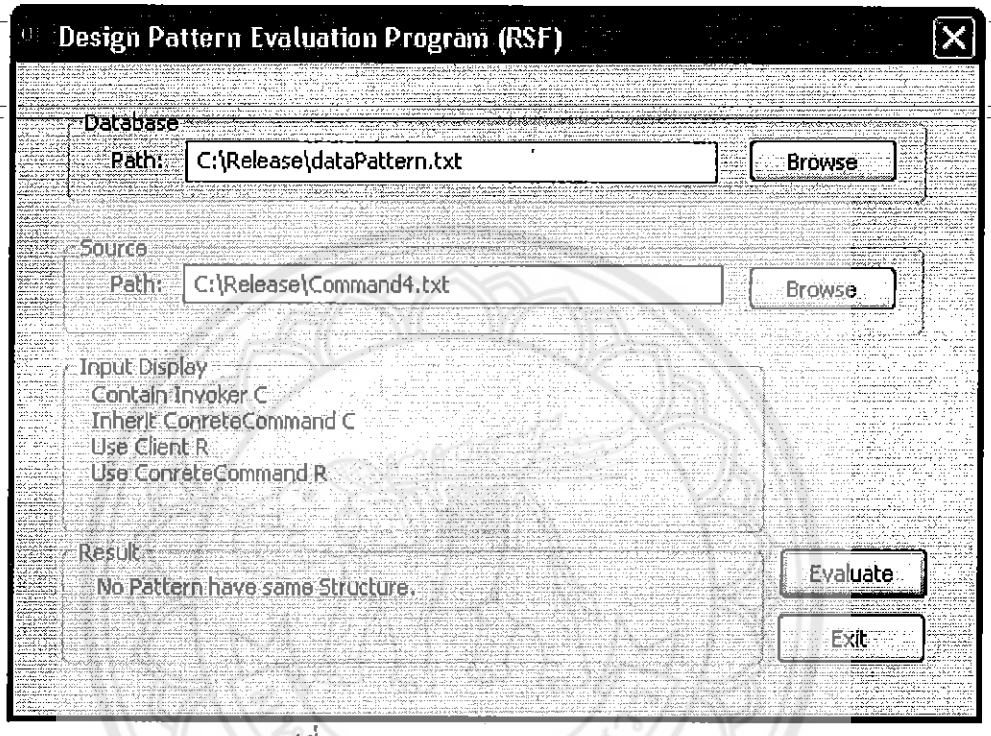
RSF ของ Pattern ที่นำมาตรวจสอบ

Contain Invoker C

Inherit ConreteCommand C

Use Client R

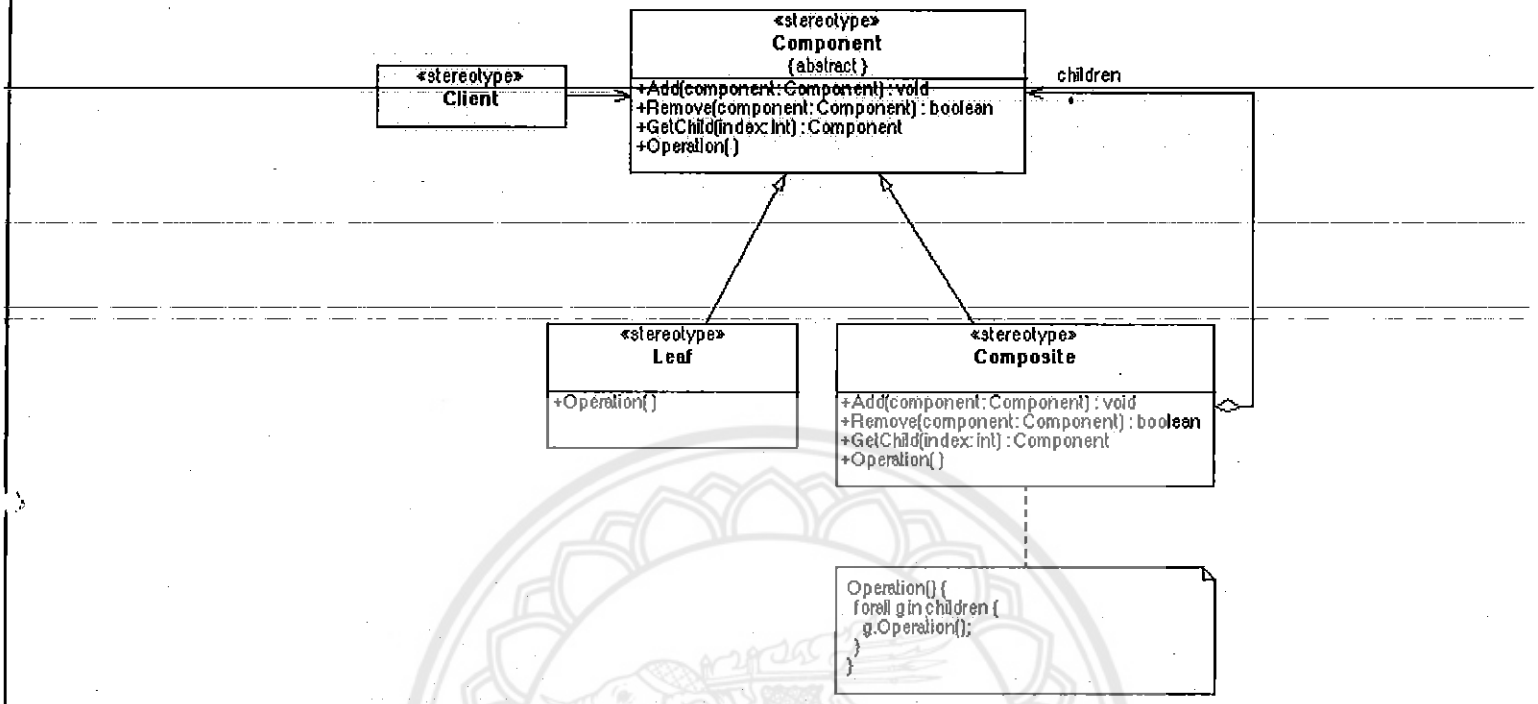
Use ConreteCommand R



รูปที่ 4.4 แสดงผลการตรวจสอบ Pattern

จากรูปที่ 4.4 เมื่อตรวจสอบ RSF ของ Pattern ที่นำมาตรวจสอบกับ Design Pattern ในฐานข้อมูล แล้ว พบว่าไม่มี Design Pattern ใดที่มีส่วนประกอบและความสัมพันธ์ตรงกับ Pattern ที่นำมาตรวจสอบ ดังนั้น RSF ที่นำมาตรวจสอบไม่เป็น Design Pattern

การตรวจสอบ Composite Pattern



รูปที่ 4.5 โครงสร้างของ Composite Pattern

โครงสร้าง ของ Composite Pattern สามารถเขียนได้ 4 แบบ คือ

แบบที่ 1 คือ

ComPat (Component, Leaf, Composite) :=

Inherit (Composite, Component)

& Contain (Composite, Component)

& Inherit (Leaf, Component)

& ! Contain (Leaf, Component);

แบบที่ 2 คือ

ComPat (Component, Leaf, Composite, Client) :=

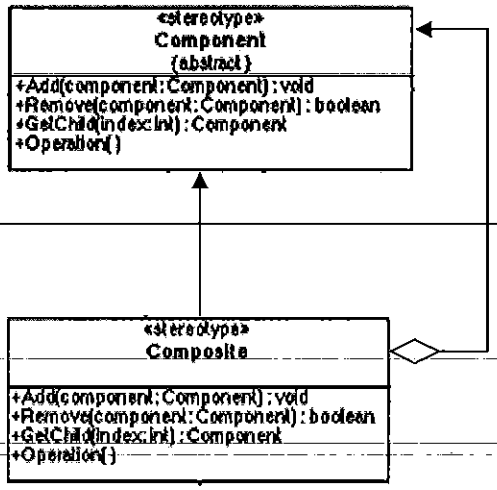
Inherit (Composite, Component)

& Contain (Composite, Component)

& Inherit (Leaf, Component)

& ! Contain (Leaf, Component)

& Use (Client, Component);



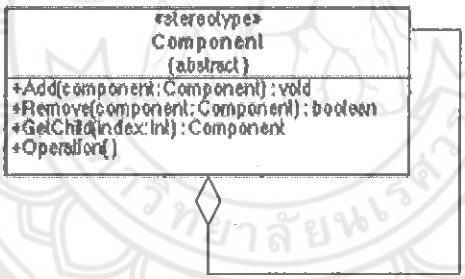
รูปที่ 4.6 โครงสร้าง Composite Pattern แบบที่ 3

แบบที่ 3 คือ

ComPat (Component, Composite) :=

Contain (Composite, Component)

& Inherit (Composite, Component);



รูปที่ 4.7 โครงสร้าง Composite Pattern แบบที่ 3

แบบที่ 4 คือ

ComPat (Component) :=

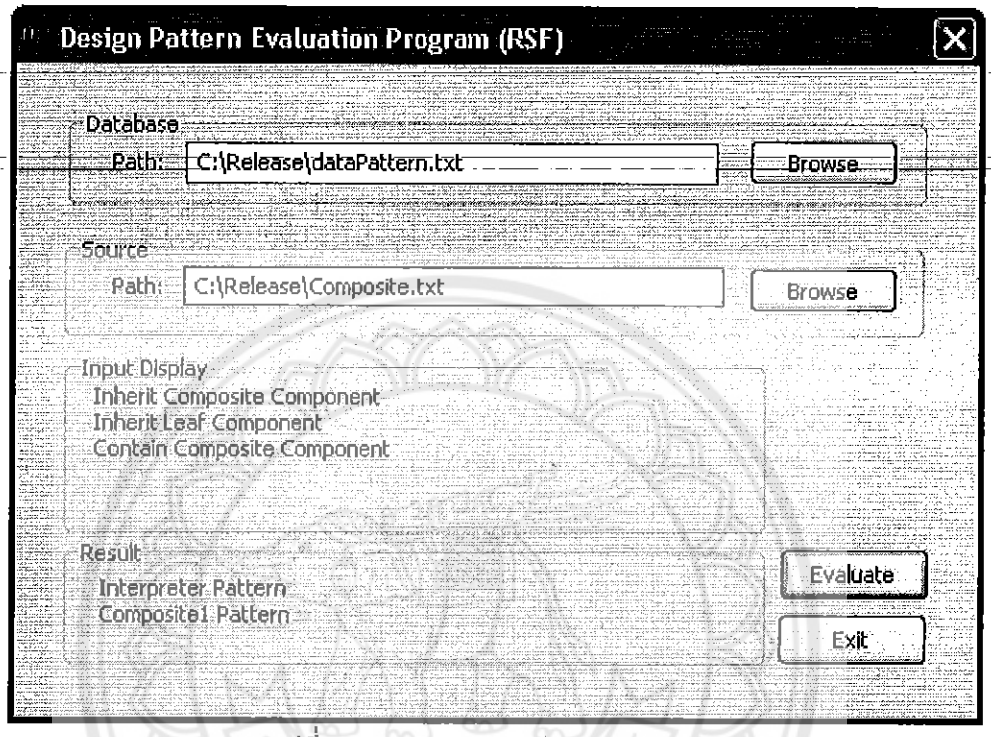
Contain (Component, Component);

RSF ของ Pattern ที่นำมาตรวจสอบ

Inherit Composite Component

Contain Composite Component

Inherit Leaf Component



รูปที่ 4.8 แสดงผลการตรวจสอบ Pattern

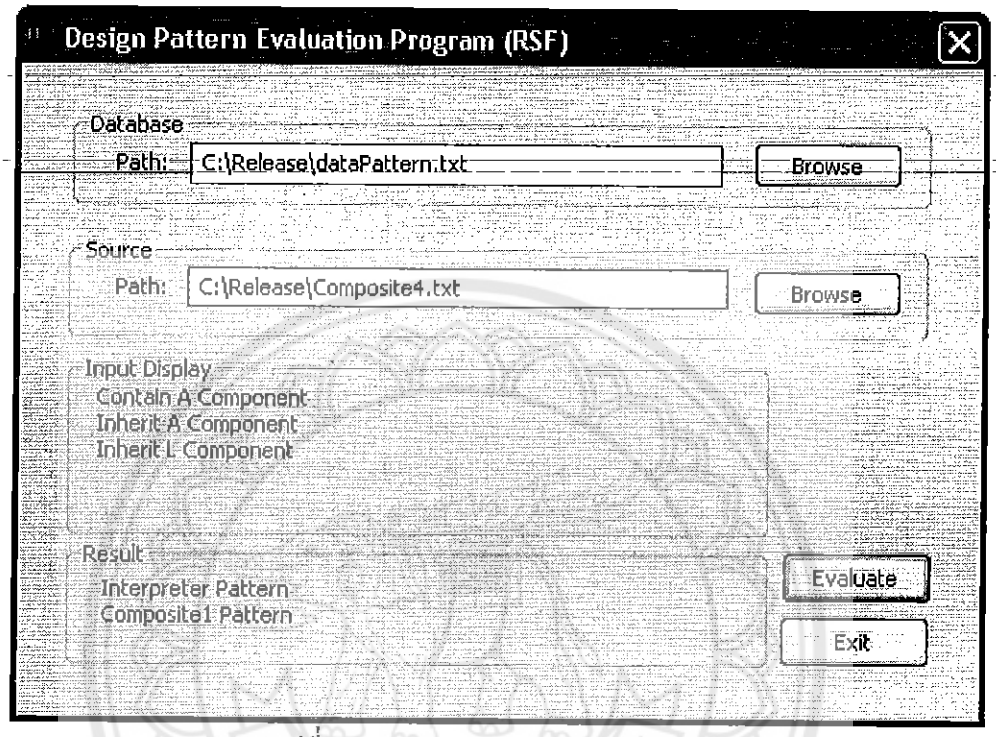
จากรูปที่ 4.8 เมื่อตรวจสอบ RSF ของ Pattern ที่นำมาตรวจสอบกับ Design Pattern ในฐานข้อมูล พบว่าเป็น Interpreter Pattern และ Composite Pattern เนื่องจากมีส่วนประกอบและความสัมพันธ์ของส่วนประกอบตรงกับ Design Pattern ทั้งสอง

RSF ของ Pattern ที่นำมาตรวจสอบ

Contain A Component

Inherit A Component

Inherit L Component



รูปที่ 4.9 แสดงผลการตรวจสอบ Pattern

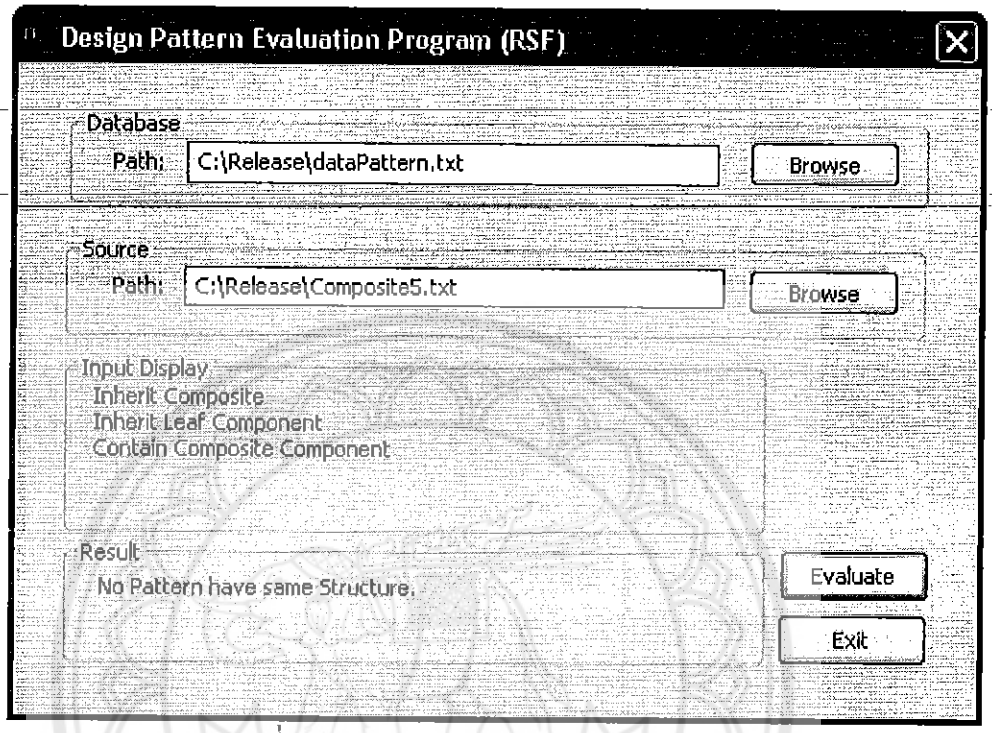
จากรูปที่ 4.9 RSF ของ Pattern ที่นำมาตรวจสอบมีการสลับลำดับการเขียนและใช้ชื่อส่วนประกอบเหมือนกันกับใน Design Pattern ในฐานข้อมูล แต่เมื่อนำมาตรวจสอบกับ Design Pattern ใน ฐานข้อมูล พบว่าเป็น Composite Pattern และ Interpreter Pattern เนื่องจาก RSF ของ Pattern ที่นำมาตรวจสอบนั้นมีส่วนประกอบและความสัมพันธ์ตรงกันกับ Composite Pattern และ Interpreter Pattern ใน ฐานข้อมูล

RSF ของ Pattern ที่นำมาตรวจสอบ

Inherit Composite

Inherit Leaf Component

Contain Composite Component



รูปที่ 4.10 แสดงผลการตรวจสอบ Pattern

จากรูปที่ 4.10 RSF ของ Pattern ที่นำมาตรวจสอบนั้นได้มีการตัดส่วนประกอบบางอย่างออกไป ดังนั้น เมื่อนำมาตรวจสอบกับ Design Pattern ในฐานข้อมูล จึงไม่มี Design Pattern ใดตรงกับ Pattern ที่นำมาตรวจสอบเลย

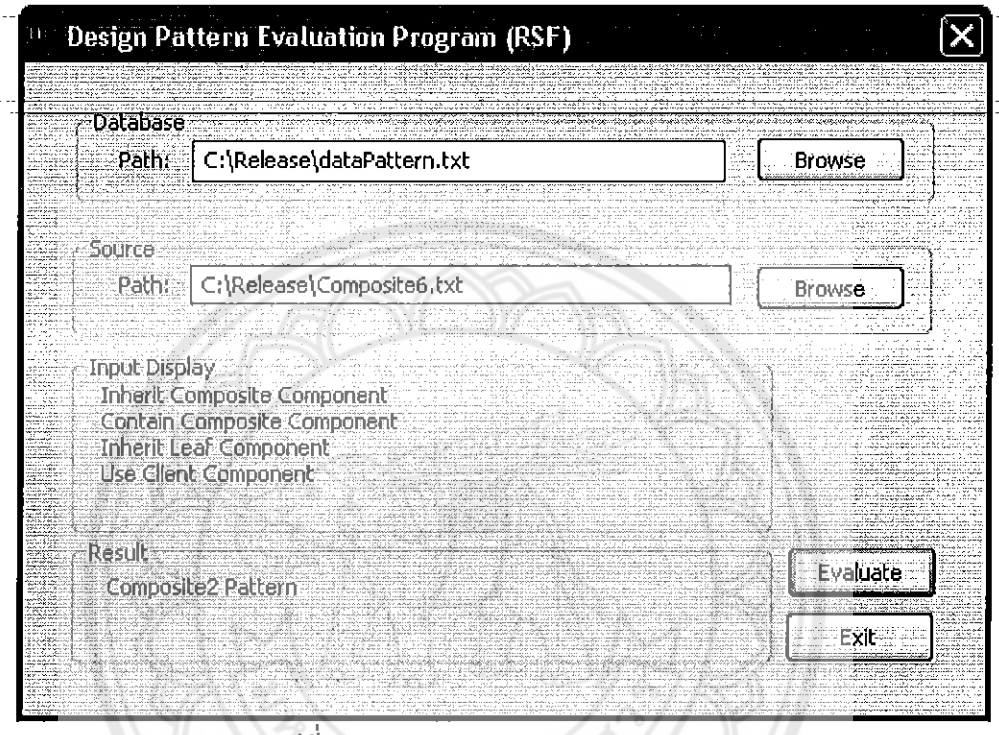
RSF ของ Pattern ที่นำมาตรวจสอบ

Inherit Composite Component

Contain Composite Component

Inherit Leaf Component

Use Client Component



รูปที่ 4.11 แสดงผลการตรวจสอบ Pattern

จากรูปที่ 4.11 RSF ของ Pattern เมื่อนำมาตรวจสอบกับ Design Pattern ที่อยู่ในฐานข้อมูล จึงพบว่าเป็น Composite Pattern แบบที่ 2 เนื่องจากมีส่วนประกอบและความสัมพันธ์ของส่วนประกอบตรงกับ Composite Pattern แบบที่ 2

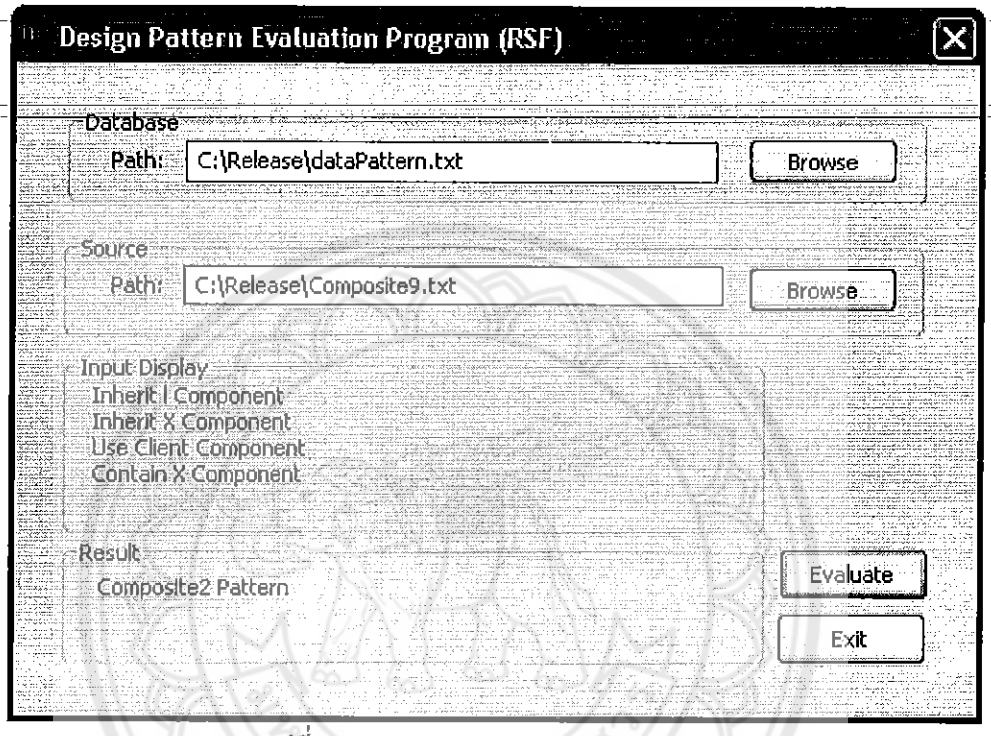
RSF ของ Pattern ที่นำมาตรวจสอบ

Inherit l Component

Inherit X Component

Use Client Component

Contain X Component



รูปที่ 4.12 แสดงผลการตรวจสอบ Pattern

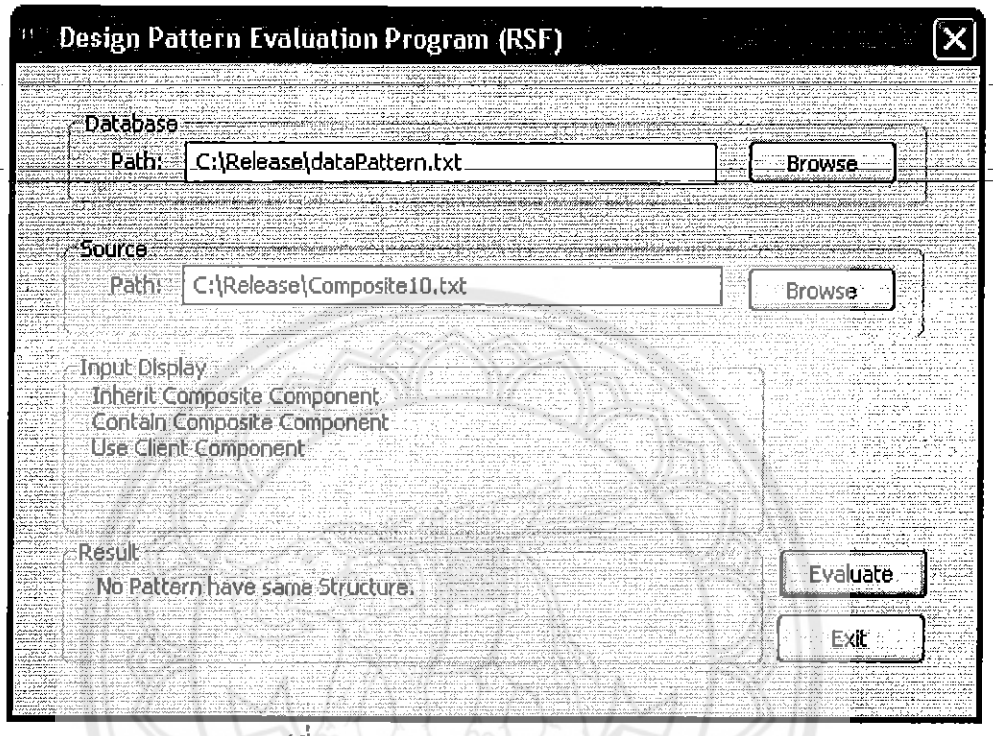
จากรูปที่ 4.12 RSF ของ Pattern ที่นำมาตรวจสอบมีการสลับลำดับการเขียนและใช้ชื่อส่วนประกอบไม่เหมือนกันกับ Design Pattern ในฐานข้อมูล แต่เมื่อนำมาตรวจสอบกับ Design Pattern ใน ฐานข้อมูล พบว่าเป็น Composite Pattern แบบที่ 2 เนื่องจาก RSF ของ Pattern ที่นำมาตรวจสอบนั้นมีความสัมพันธ์ของส่วนประกอบที่เหมือนกัน

RSF ของ Pattern ที่นำมาตรวจสอบ

Inherit Composite Component

Contain Composite Component

Use Client Component



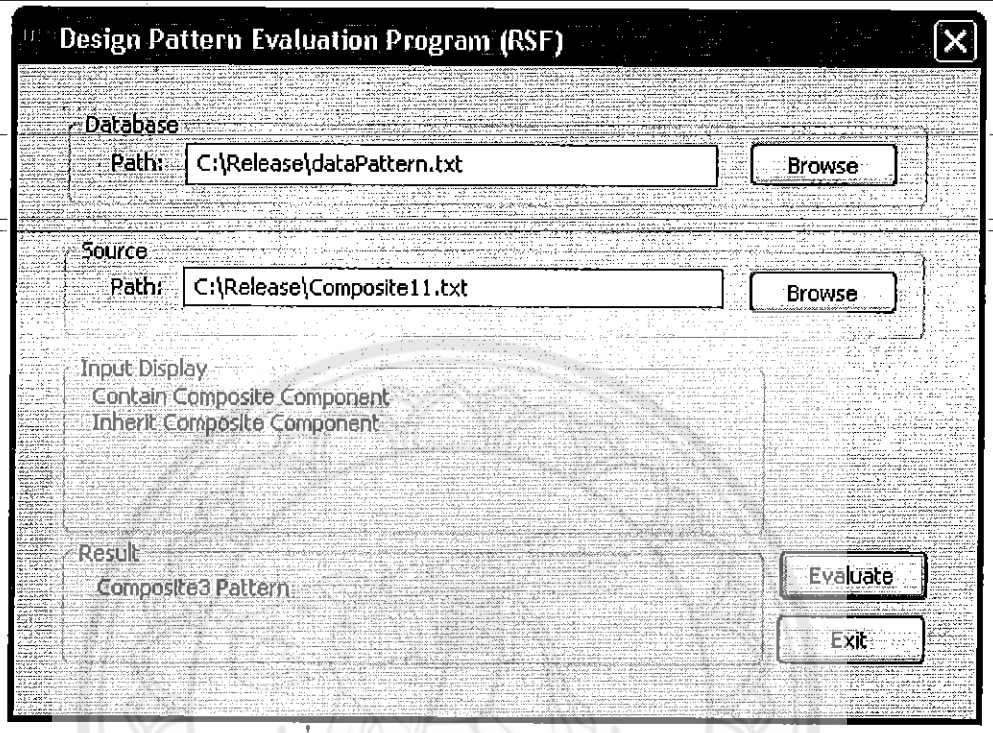
รูปที่ 4.13 แสดงผลการตรวจสอบ Pattern

จากรูปที่ 4.13 RSF ของ Pattern ที่นำมาตรวจสอบถูกตัดโครงสร้างบางส่วนออก เมื่อนำมาตรวจสอบกับ Design Pattern ที่อยู่ในฐานข้อมูล จึงไม่มี Design Pattern ใดเลยที่ตรงกับ Pattern ที่นำมาตรวจสอบ

RSF ของ Pattern ที่นำมาตรวจสอบ

Contain Composite Component

Inherit Composite Component



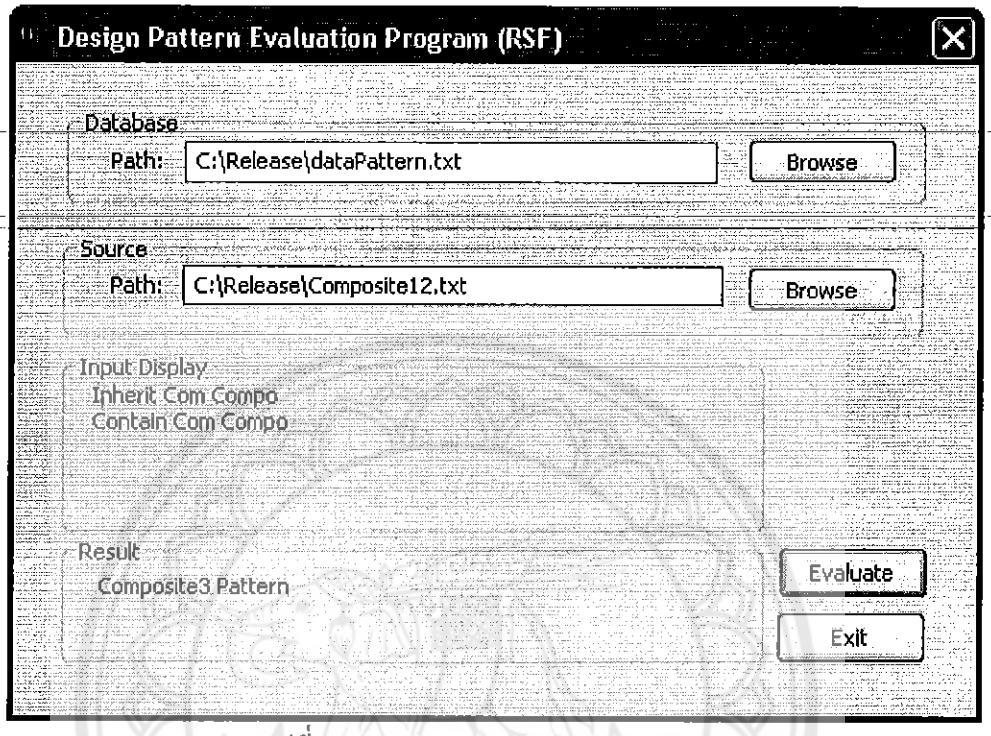
รูปที่ 4.14 แสดงผลการตรวจสอบ Pattern

RSF ของ Pattern ที่นำมาตรวจสอบมีการวางลำดับและการใช้คำเหมือนกับ Composite Pattern แบบที่ 3 ดังนั้น เมื่อนำไปตรวจสอบกับ Design Pattern ในฐานข้อมูล จะพบว่า Pattern ที่นำมาตรวจสอบนั้นตรงกับ Composite Pattern แบบที่ 3

RSF ของ Pattern ที่นำมาตรวจสอบ

Inherit Com Compo

Contain Com Compo



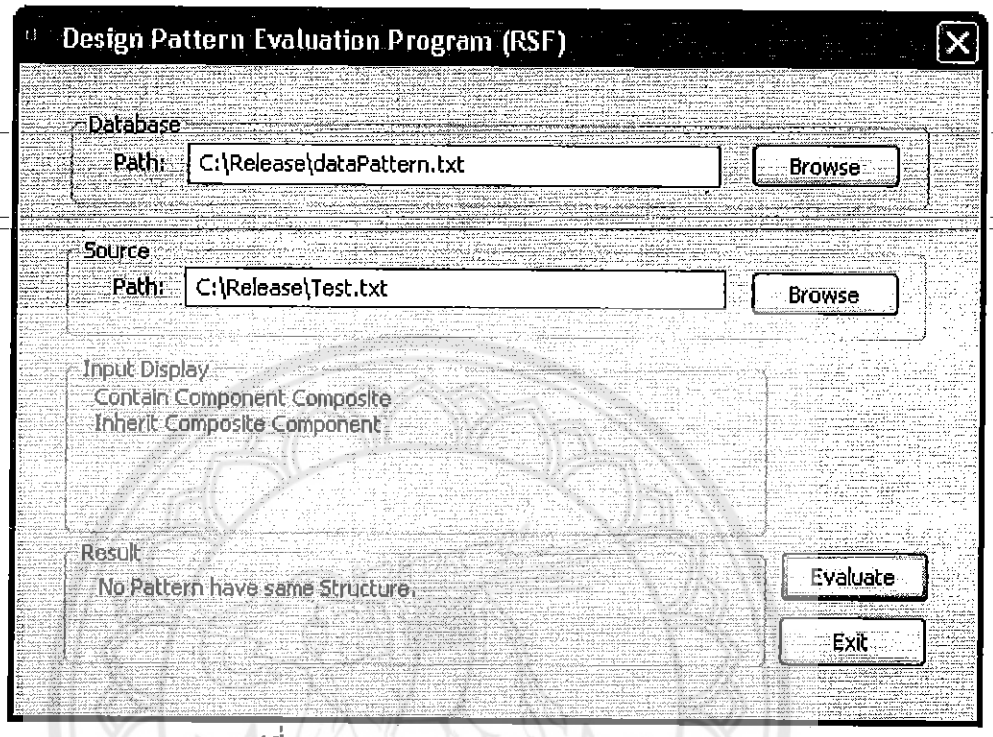
รูปที่ 4.15 แสดงผลการตรวจสอบ Pattern

จากรูปที่ 4.15 RSF ของ Pattern ที่นำมาตรวจสอบมีการสลับลำดับการเขียนและใช้ชื่อส่วนประกอบไม่เหมือนกันกับ Design Pattern ในฐานข้อมูล แต่เมื่อนำมาตรวจสอบกับDesign Pattern ใน ฐานข้อมูล พบว่าเป็น Composite Pattern แบบที่ 3 เนื่องจาก RSF ของ Pattern ที่นำมาตรวจสอบนั้นมีความสัมพันธ์ของส่วนประกอบที่เหมือนกัน

RSF ของ Pattern ที่นำมาตรวจสอบ

Contain Component Composite

Inherit Composite Component

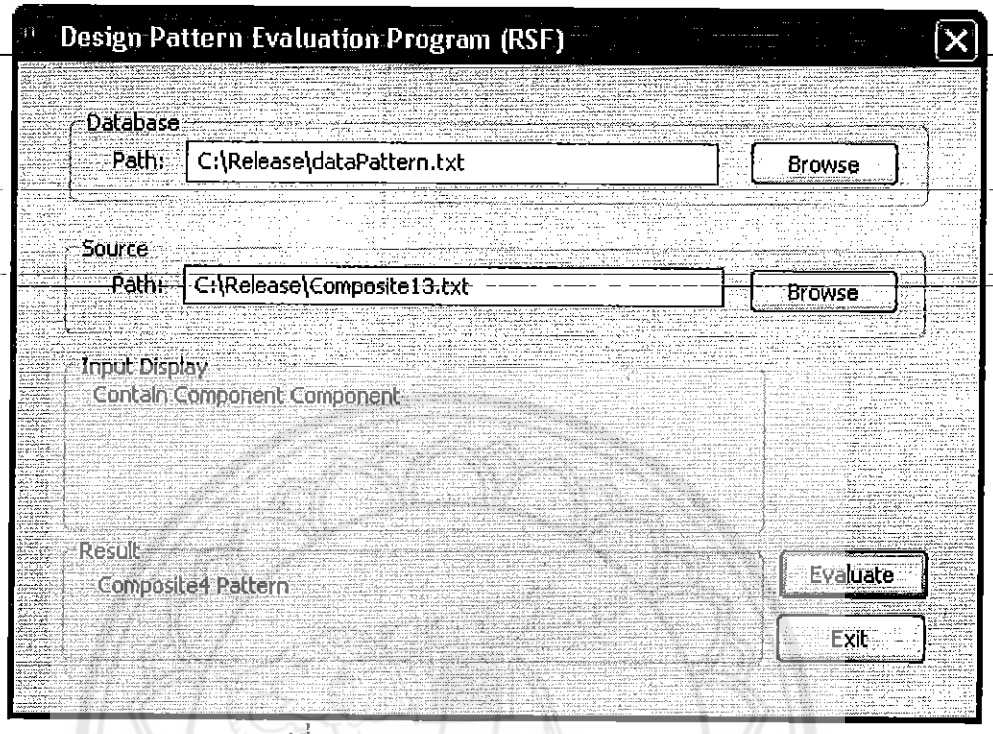


รูปที่ 4.16 แสดงผลการตรวจสอบ Pattern

จากรูปที่ 4.16 RSF ของ Pattern ที่นำมาตรวจสอบมีการสลับตำแหน่งของส่วนประกอบบางตำแหน่ง เมื่อนำมาตรวจสอบกับ Design Pattern ที่อยู่ในฐานข้อมูล จึงไม่มี Design Pattern ใดเลยที่ตรงกับ Pattern ที่นำมาตรวจสอบ

RSF ของ Pattern ที่นำมาตรวจสอบ

Contain Component Component

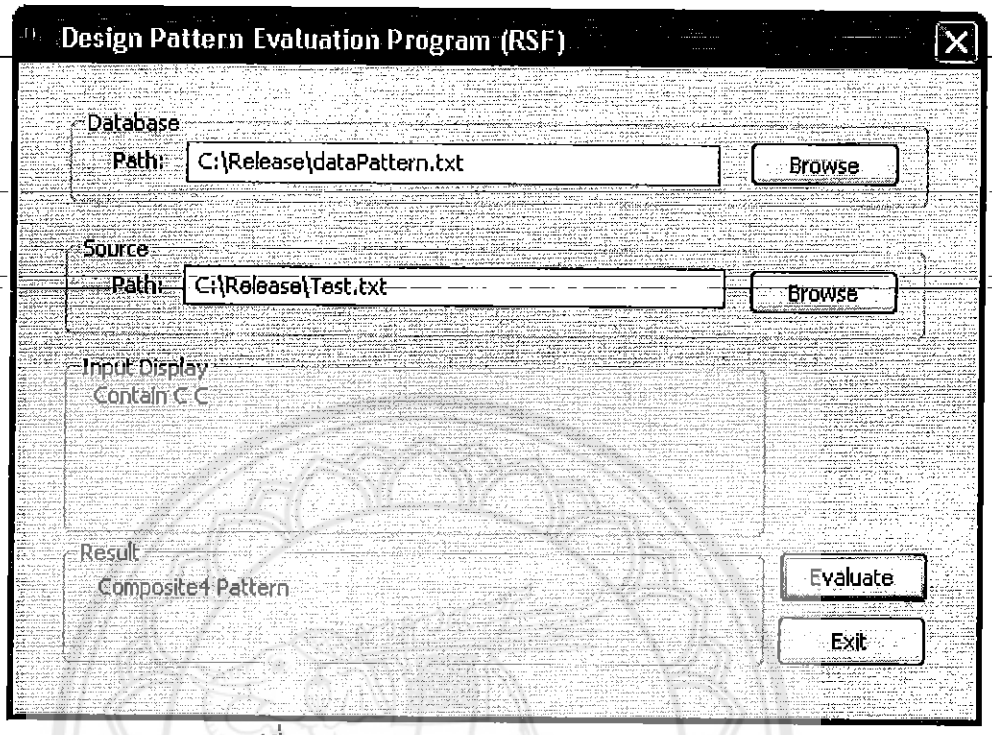


รูปที่ 4.17 แสดงผลการตรวจสอบ Pattern

จากรูปที่ 4.17 RSF ของ Pattern เมื่อนำไปตรวจสอบกับ Design Pattern ในฐานข้อมูล จะพบว่า Pattern ที่นำมาตรวจสอบนั้นตรงกับ Composite Pattern แบบที่ 4 เนื่องจากมีส่วนประกอบและความสัมพันธ์ของส่วนประกอบตรงกับ Composite Pattern แบบที่ 4

RSF ของ Pattern ที่นำมาตรวจสอบ

Contain C C

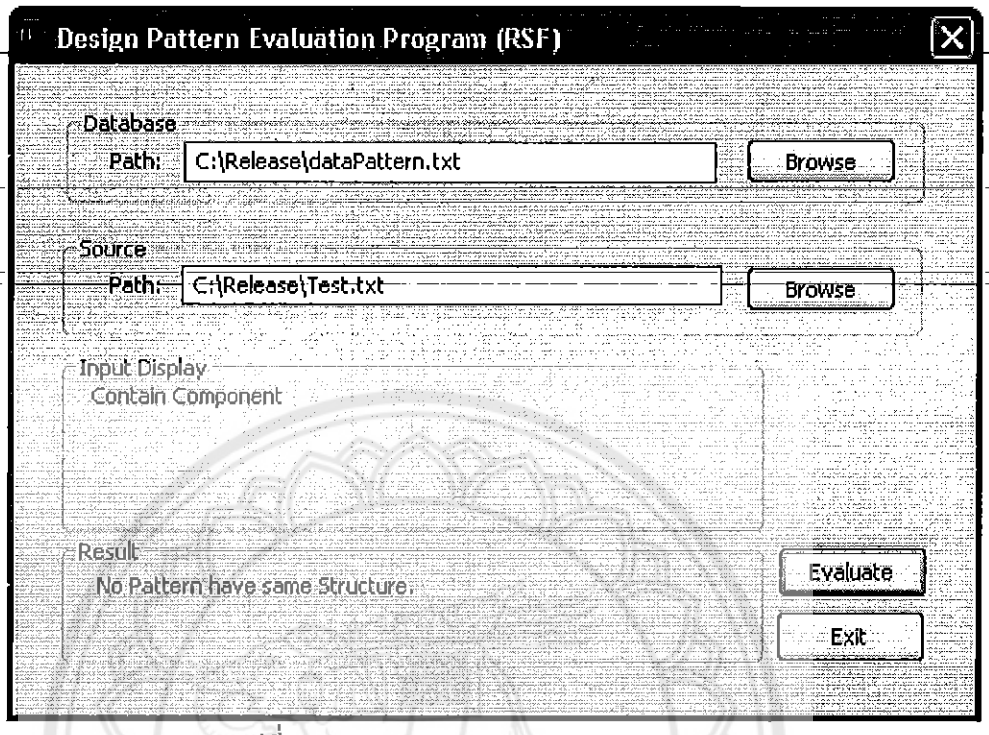


รูปที่ 4.18 แสดงผลการตรวจสอบ Pattern

จากรูปที่ 4.18 RSF ของ Pattern ที่นำมาตรวจสอบมีการสลับลำดับการเขียนและใช้ชื่อส่วนประกอบไม่เหมือนกันกับ Design Pattern ในฐานข้อมูล แต่เมื่อนำมาตรวจสอบกับ Design Pattern ใน ฐานข้อมูล พบว่าเป็น Composite Pattern แบบที่ 4 เนื่องจาก RSF ของ Pattern ที่นำมาตรวจสอบนั้นมีความสัมพันธ์ของส่วนประกอบที่เหมือนกัน

RSF ของ Pattern ที่นำมาตรวจสอบ

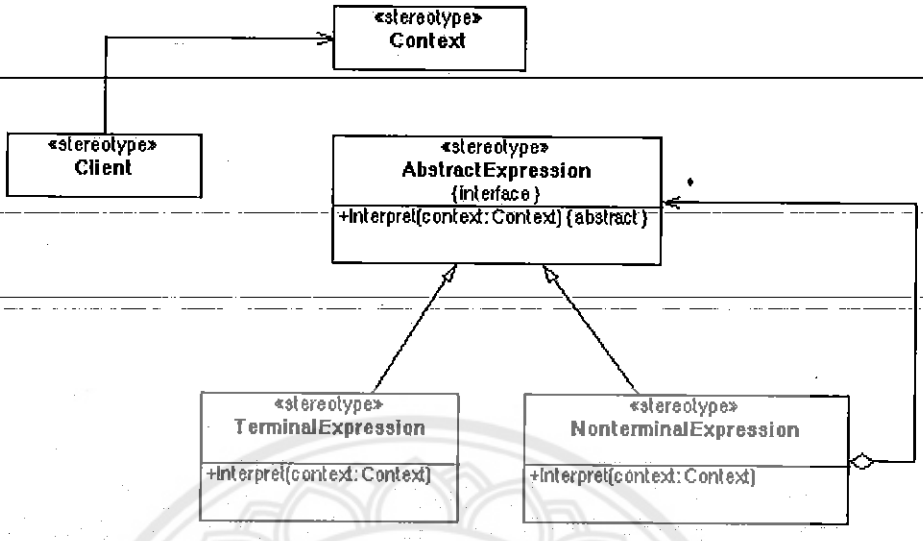
Contain Component



รูปที่ 4.19 แสดงผลการตรวจสอบ Pattern

จากรูปที่ 4.19 RSF ของ Pattern ที่นำมาตรวจสอบถูกตัดโครงสร้างบางส่วนออก เมื่อนำมาตรวจสอบกับ Design Pattern ที่อยู่ในฐานข้อมูล จึงไม่มี Design Pattern ใดเลยที่ตรงกับ Pattern ที่นำมาตรวจสอบ

การตรวจสอบ Interpreter Pattern



รูปที่ 4.20 โครงสร้าง Interpreter Pattern

โครงสร้าง ของ Interpreter Pattern คือ

ComPat (AbstractExpression, TerminalExpression, NonterminalExpression) :=

Inherit (NonterminalExpression, AbstractExpression)

& Contain (NonterminalExpression, AbstractExpression)

& Inherit (TerminalExpression, AbstractExpression)

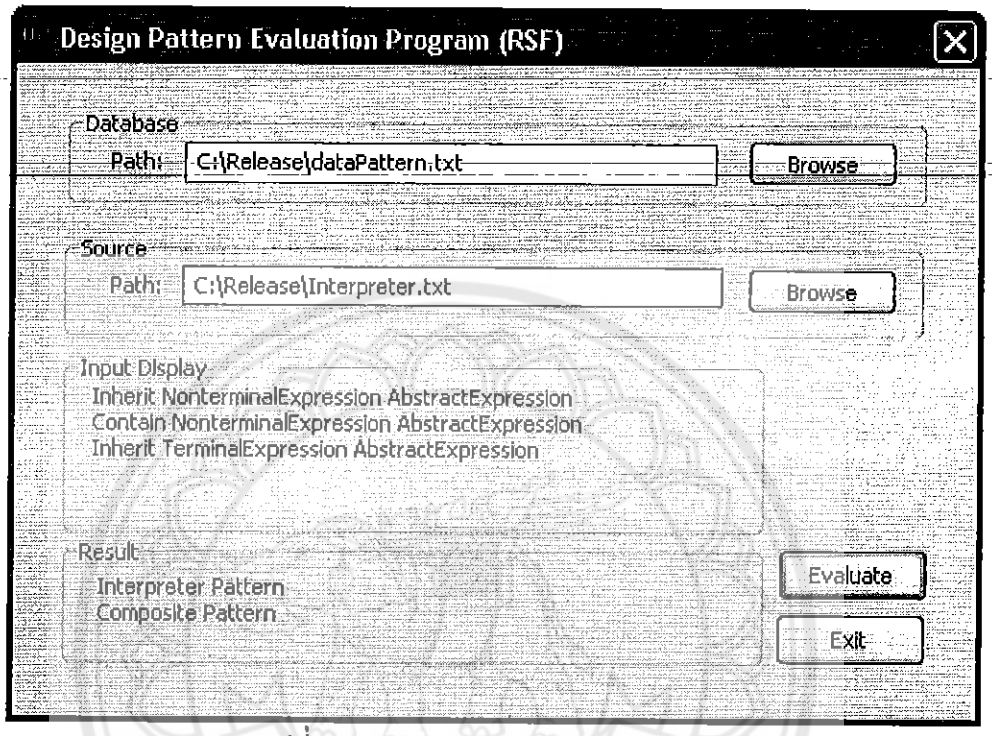
& ! Contain (TerminalExpression, AbstractExpression);

RSF ของ Pattern ที่นำมาตรวจสอบ

Inherit NonterminalExpression AbstractExpression

Contain NonterminalExpression AbstractExpression

Inherit TerminalExpression AbstractExpression



รูปที่ 4.21 แสดงผลการตรวจสอบ Pattern

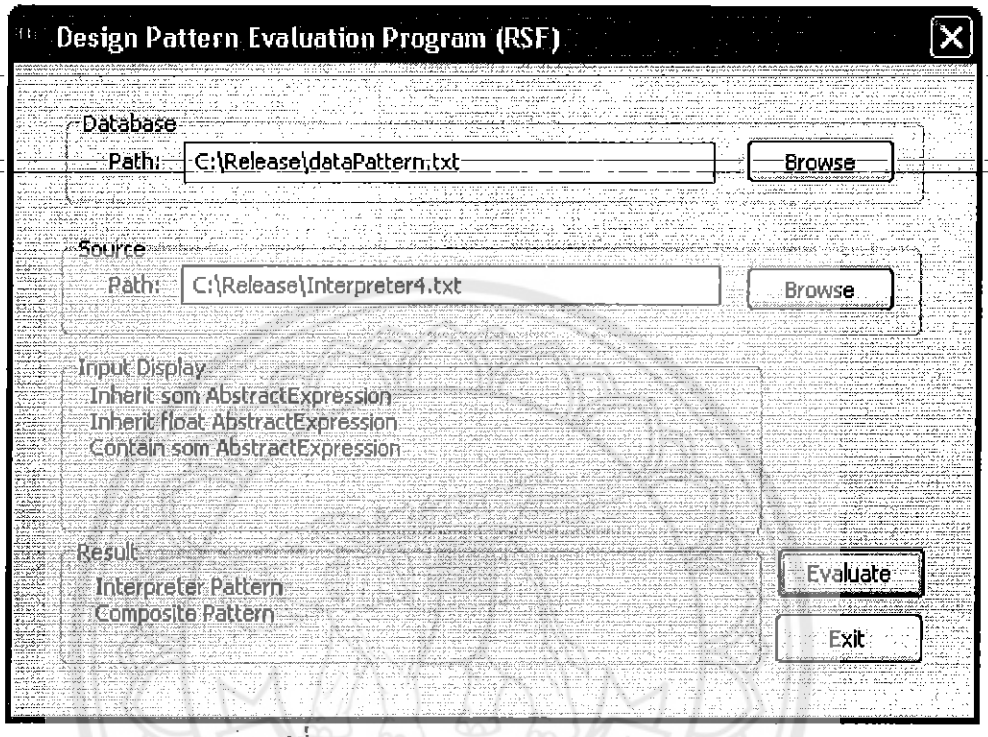
จากรูปที่ 4.21 RSF ของ Pattern เมื่อนำมาตรวจสอบกับ Design Pattern ที่อยู่ในฐานข้อมูล จึงพบว่าเป็น Interpreter Pattern และ Composite Pattern เนื่องจากมีส่วนประกอบและความสัมพันธ์ของส่วนประกอบตรงกับ Design Pattern ทั้งสอง

RSF ของ Pattern ที่นำมาตรวจสอบ

Inherit som AbstractExpression

Inherit float AbstractExpression

Contain som AbstractExpression



รูปที่ 4.22 แสดงผลการตรวจสอบ Pattern

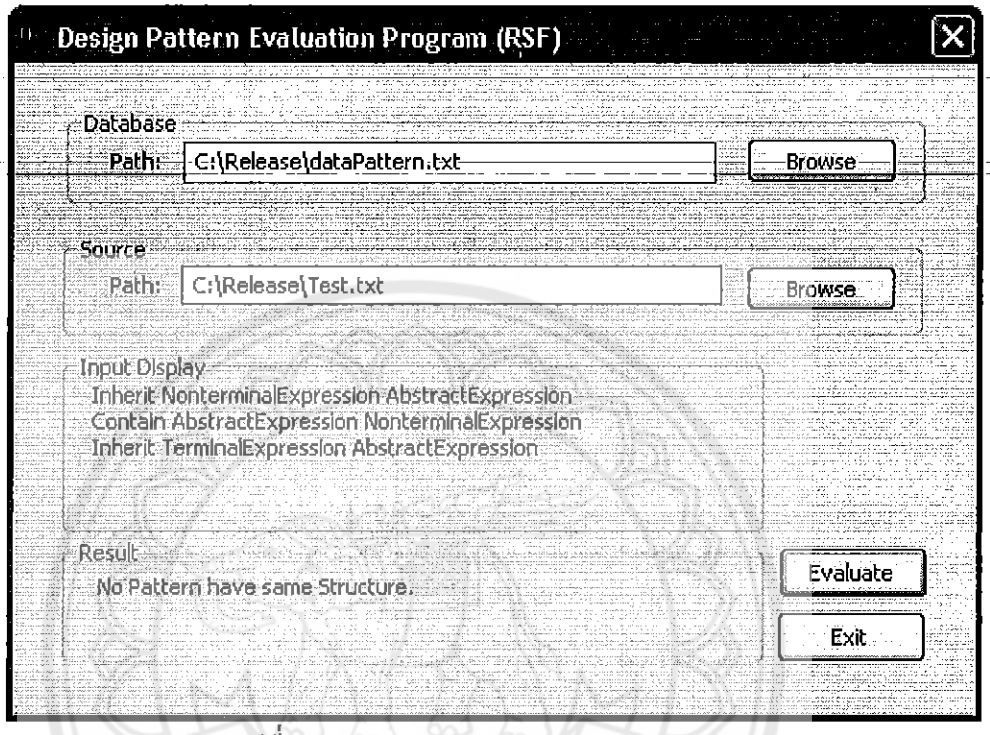
จากรูปที่ 4.22 RSF ของ Pattern ที่นำมาตรวจสอบมีการสลับลำดับการเขียนและชื่อส่วนประกอบไม่เหมือนกันกับ Design Pattern ของฐานข้อมูล แต่เมื่อนำมาตรวจสอบกับ Design Pattern ใน ฐานข้อมูล พบว่าเป็น Interpreter Pattern และ Composite Pattern เนื่องจากมีส่วนประกอบและความสัมพันธ์ของส่วนประกอบตรงกับ Design Pattern ทั้งสอง

RSF ของ Pattern ที่นำมาตรวจสอบ

Inherit NonterminalExpression AbstractExpression

Contain AbstractExpression NonterminalExpression

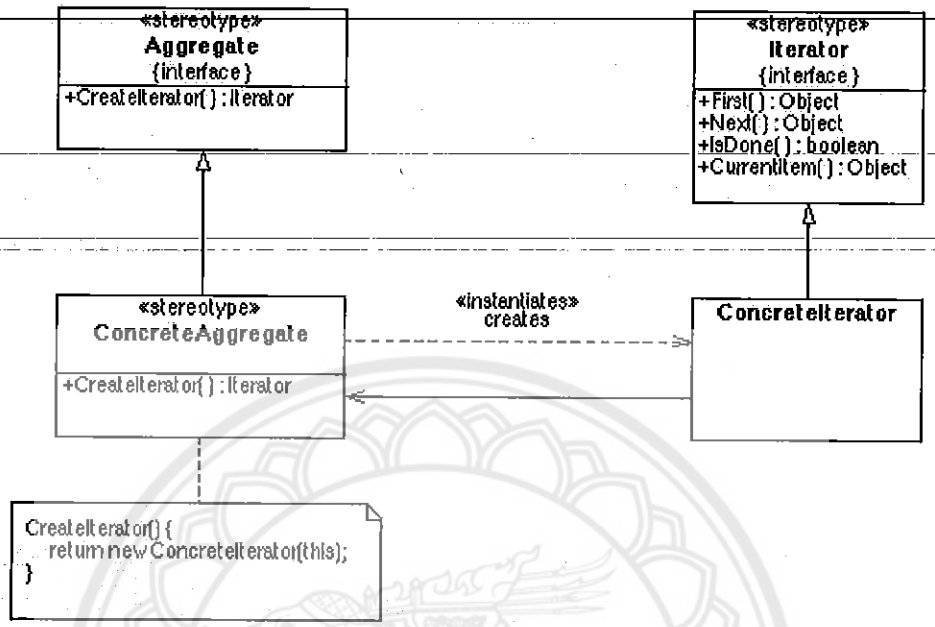
Inherit TerminalExpression AbstractExpression



รูปที่ 4.23 แสดงผลการตรวจสอบ Pattern

จากรูปที่ 4.23 RSF ของ Pattern ที่นำมาตรวจสอบถูกตัดโครงสร้างบางส่วนออก เมื่อนำมาตรวจสอบกับ Design Pattern ที่อยู่ในฐานข้อมูล จึงไม่มี Design Pattern ใดเลยที่ตรงกับ Pattern ที่นำมาตรวจสอบ

การตรวจสอบ Iterator Pattern



รูปที่ 4.24 โครงสร้าง Iterator Pattern

โครงสร้าง ของ Iterator Pattern คือ

ComPat (Iterator, Aggregate, ConcreteIterator, ConcreteAggregate) :=

Inherit (ConcreteIterator, Iterator)

& Use (ConcreteIterator, ConcreteAggregate)

& Creates (ConcreteAggregate, ConcreteIterator)

& Inherit (Aggregate, ConcreteAggregate);

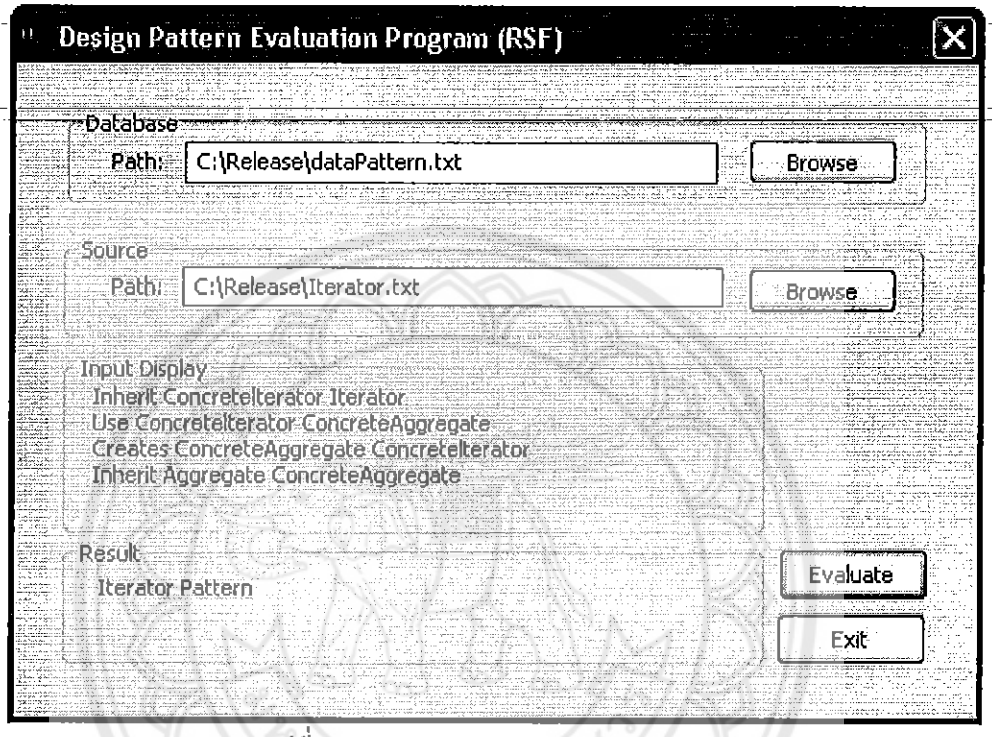
RSF ของ Pattern ที่นำมาตรวจสอบ

Inherit Concretelterator Iterator

Use Concretelterator ConcreteAggregate

Creates ConcreteAggregate Concretelterator

Inherit Aggregate ConcreteAggregate



รูปที่ 4.25 แสดงผลการตรวจสอบ Pattern

จากรูปที่ 4.25 RSF ของ Pattern เมื่อนำมาตรวจสอบกับ Design Pattern ที่อยู่ในฐานข้อมูล จึงพบว่าเป็น Iterator pattern เนื่องจากมีส่วนประกอบและความสัมพันธ์ของส่วนประกอบตรงกับ Iterator Pattern

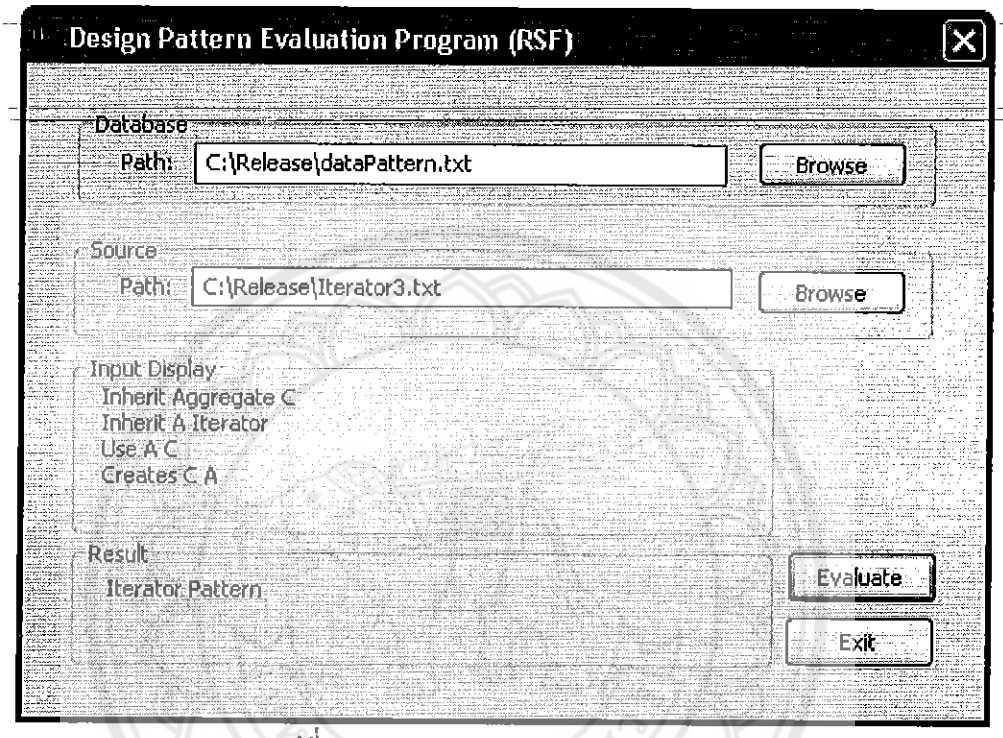
RSF ของ Pattern ที่นำมาตรวจสอบ

Inherit Aggregate C

Inherit A Iterator

Use A C

Creates C A



รูปที่ 4.26 แสดงผลการตรวจสอบ Pattern

จากรูปที่ 4.26 RSF ของ Pattern ที่นำมาตรวจสอบมีการสลับลำดับการเขียนและใช้ชื่อส่วนประกอบไม่เหมือนกันกับใน Design Pattern ของฐานข้อมูล แต่เมื่อนำมาตรวจสอบกับ Design Pattern ใน ฐานข้อมูล พบว่าเป็น Iterator Pattern เนื่องจากมีส่วนประกอบและความสัมพันธ์ของส่วนประกอบตรงกับ Iterator Pattern

RSF ของ Pattern ที่นำมาตรวจสอบ

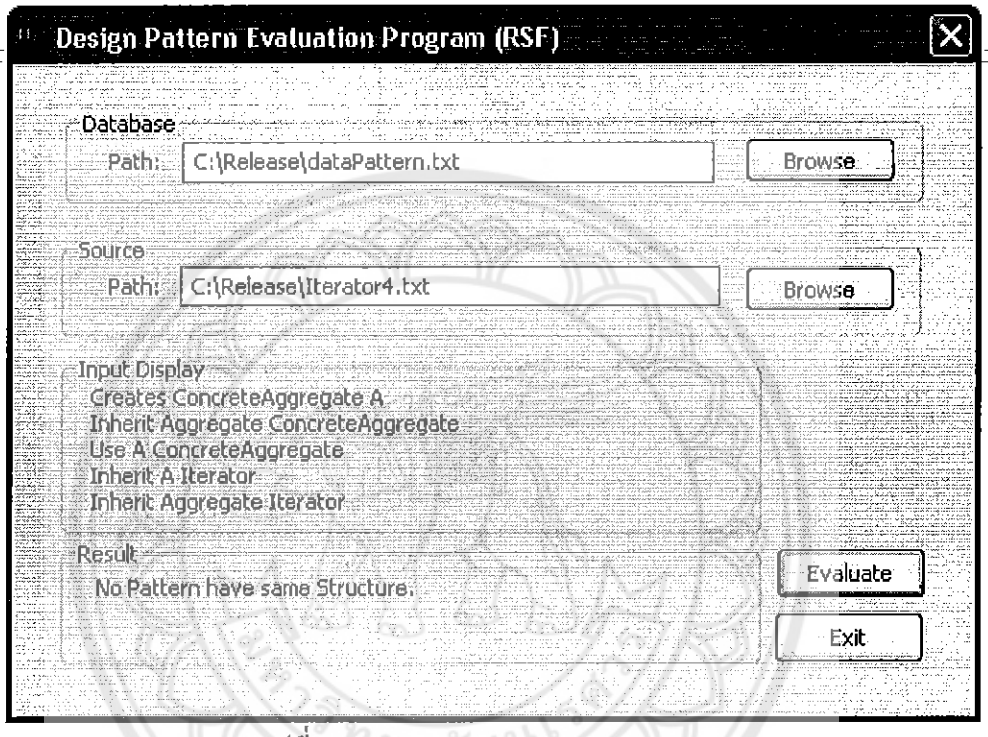
Creates ConcreteAggregate A

Inherit Aggregate ConcreteAggregate

Use A ConcreteAggregate

Inherit A Iterator

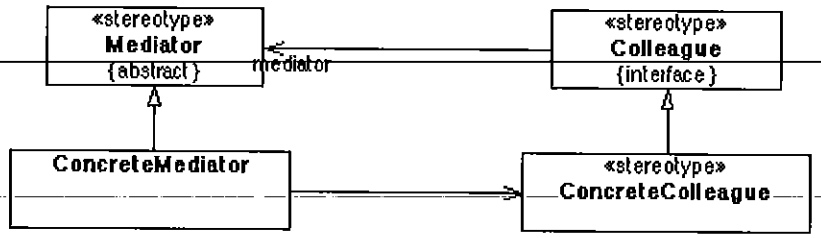
Inherit Aggregate Iterator



รูปที่ 4.27 แสดงผลการตรวจสอบ Pattern

RSF ของ Pattern ที่นำมาตรวจสอบมีการเพิ่มโครงสร้างขึ้น เมื่อนำมาตรวจสอบกับโครงสร้าง ของ Design Pattern ที่อยู่ในฐานข้อมูล จึงไม่มี Design Pattern ใดเลยที่ตรงกับ Pattern ที่นำมาตรวจสอบ

การตรวจสอบ Mediator Pattern



รูปที่ 4.28 โครงสร้าง Mediator Pattern

โครงสร้าง ของ Mediator Pattern คือ

ComPat (Mediator, Colleague, ConcreteMediator, ConcreteColleague) :=

Inherit (ConcreteMediator, Mediator)

& Use (Colleague, Mediator)

& Inherit (ConcreteColleague, Colleague)

& Use (ConcreteMediator, ConcreteColleague);

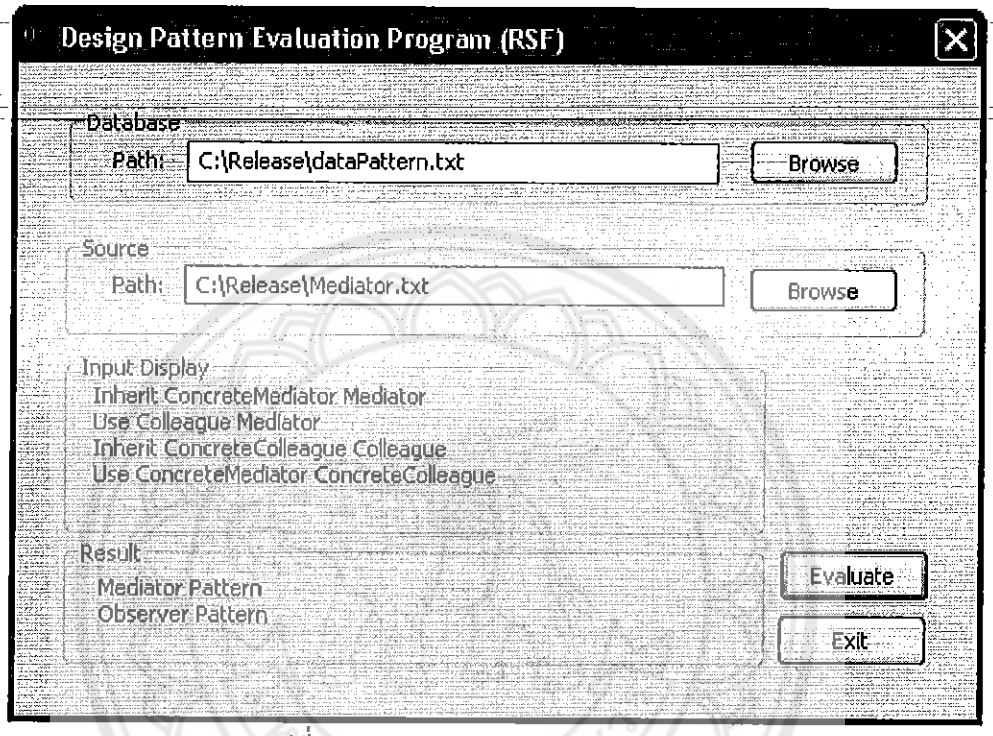
RSF ของ Pattern ที่นำมาตรวจสอบ

Inherit ConcreteMediator Mediator

Use Colleague Mediator

Inherit ConcreteColleague Colleague

Use ConcreteMediator ConcreteColleague



รูปที่ 4.29 แสดงผลการตรวจสอบ Pattern

จากรูปที่ 4.29 RSF ของ Pattern เมื่อนำมาตรวจสอบกับ Design Pattern ที่อยู่ในฐานข้อมูล จึงพบว่าเป็น Mediator pattern และ Observer Pattern เนื่องจากมีส่วนประกอบและความสัมพันธ์ของส่วนประกอบตรงกับ Design Pattern ทั้งสอง

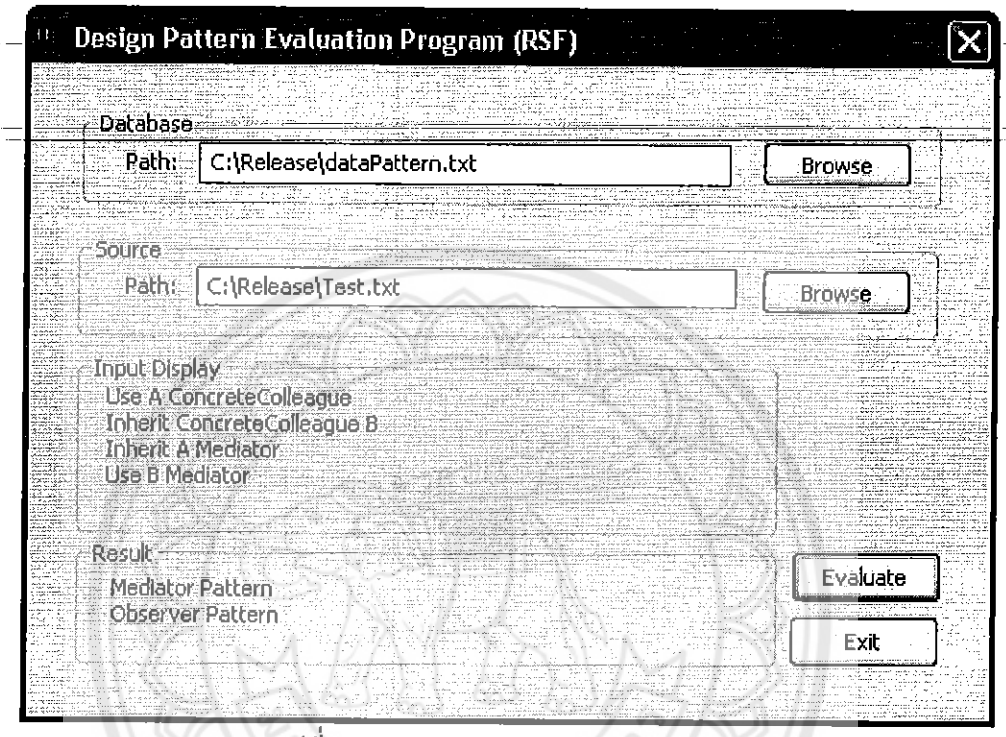
RSF ของ Pattern ที่นำมาตรวจสอบ

Use A ConcreteColleague

Inherit ConcreteColleague B

Inherit A Mediator

Use B Mediator



รูปที่ 4.30 แสดงผลการตรวจสอบ Pattern

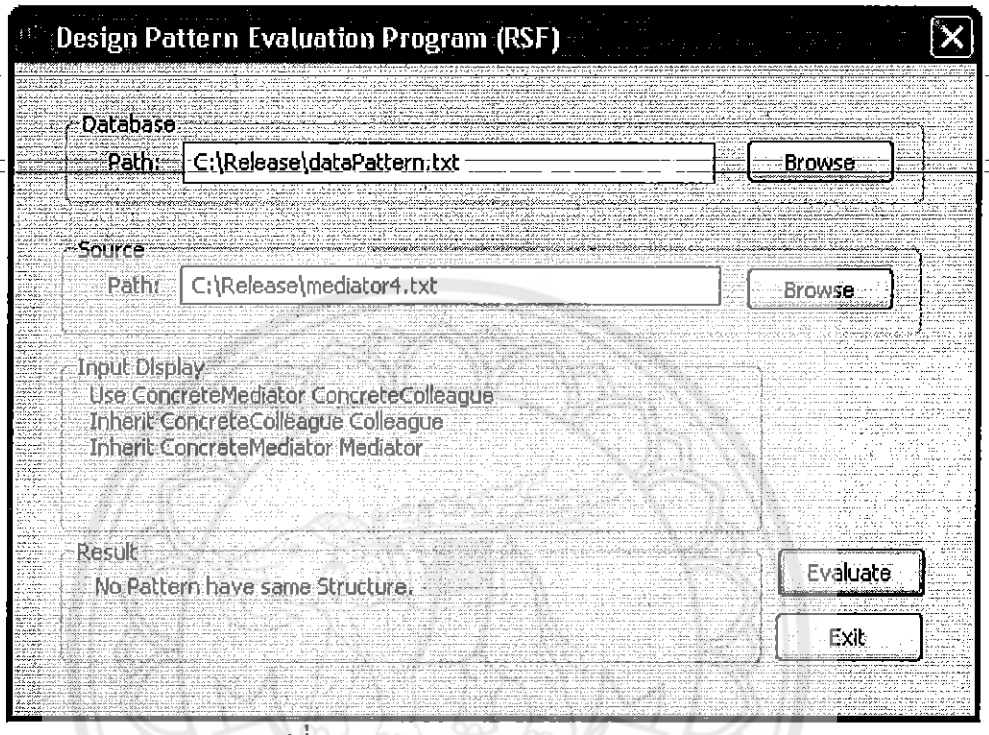
จากรูปที่ 4.30 RSF ของ Pattern ที่นำมาตรวจสอบมีการสลับลำดับการเขียนและใช้ชื่อส่วนประกอบไม่เหมือนกันกับใน Design Pattern ของฐานข้อมูล แต่เมื่อนำมาตรวจสอบกับ Design Pattern ใน ฐานข้อมูล พบว่าเป็น Mediator pattern และ Observer Pattern เนื่องจากมีส่วนประกอบและความสัมพันธ์ของส่วนประกอบตรงกับ Design Pattern ทั้งสอง

RSF ของ Pattern ที่นำมาตรวจสอบ

Use ConcreteMediator ConcreteColleague

Inherit ConcreteColleague Colleague

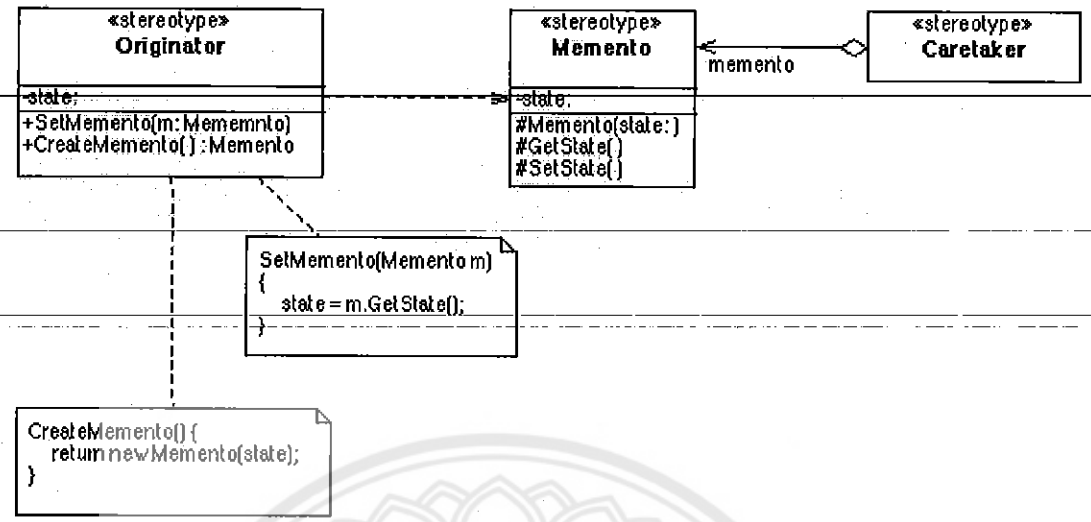
Inherit ConcreteMediator Mediator



รูปที่ 4.31 แสดงผลการตรวจสอบ Pattern

จากรูปที่ 4.31 RSF ของ Pattern ที่นำมาตรวจสอบถูกตัดโครงสร้างบางส่วนออก เมื่อนำมาตรวจสอบกับ โครงสร้าง ของ Design Pattern ที่อยู่ในฐานข้อมูล จึงไม่มี Design Pattern ใดเลยที่ตรงกับ Pattern ที่นำมาตรวจสอบ

การตรวจสอบ Memento Pattern



รูปที่ 4.32 โครงสร้าง Memento Pattern

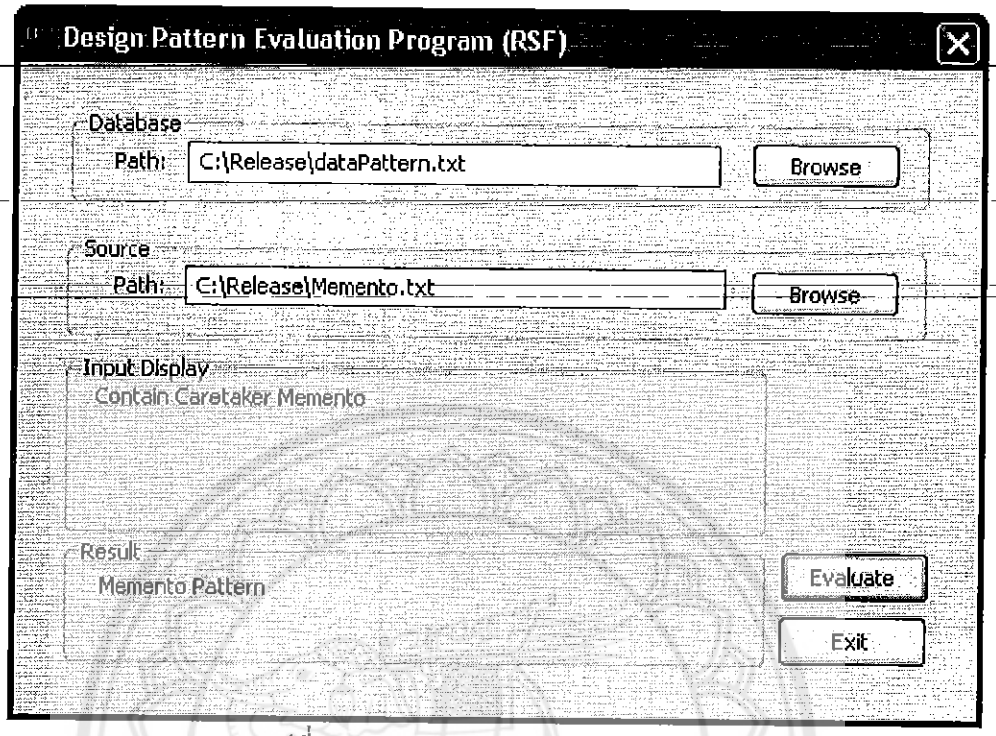
โครงสร้าง ของ Memento Pattern คือ

ComPat (Memento, Caretaker) :=

Contain (Caretaker, Memento);

RSF ของ Pattern ที่นำมาตรวจสอบ

Contain Caretaker Memento

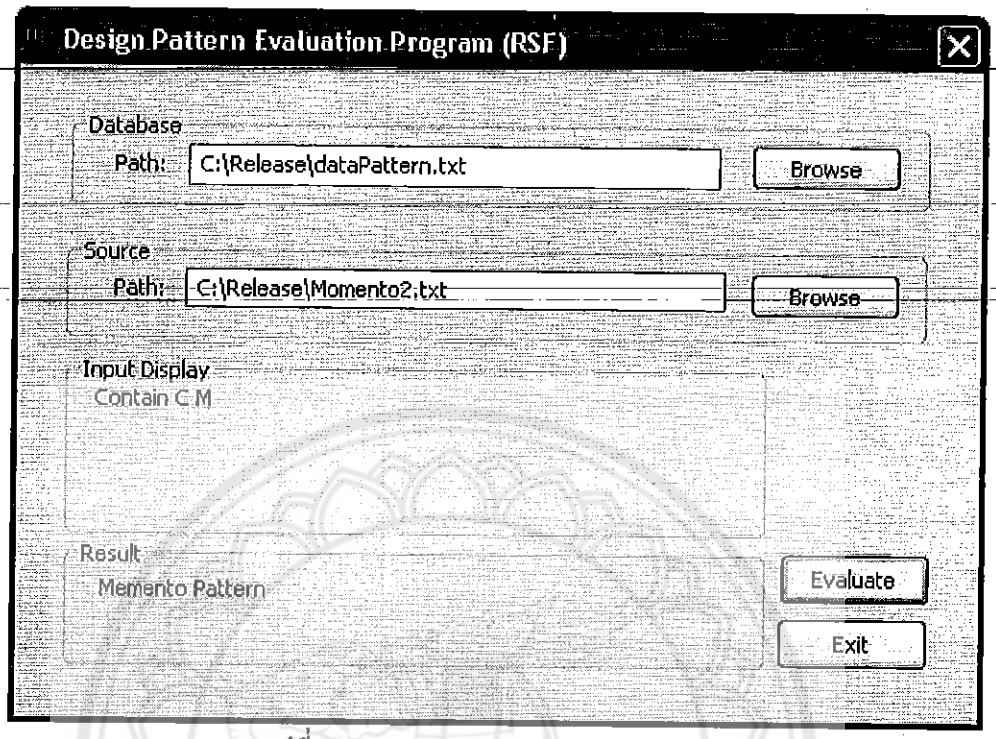


รูปที่ 4.33 แสดงผลการตรวจสอบ Pattern

จากรูปที่ 4.33 RSF ของ Pattern เมื่อนำมาตรวจสอบกับ Design Pattern ที่อยู่ในฐานข้อมูล จึงพบว่าเป็น Memento pattern เนื่องจากมีส่วนประกอบและความสัมพันธ์ของส่วนประกอบตรงกับ Memento pattern

RSF ของ Pattern ที่นำมาตรวจสอบ

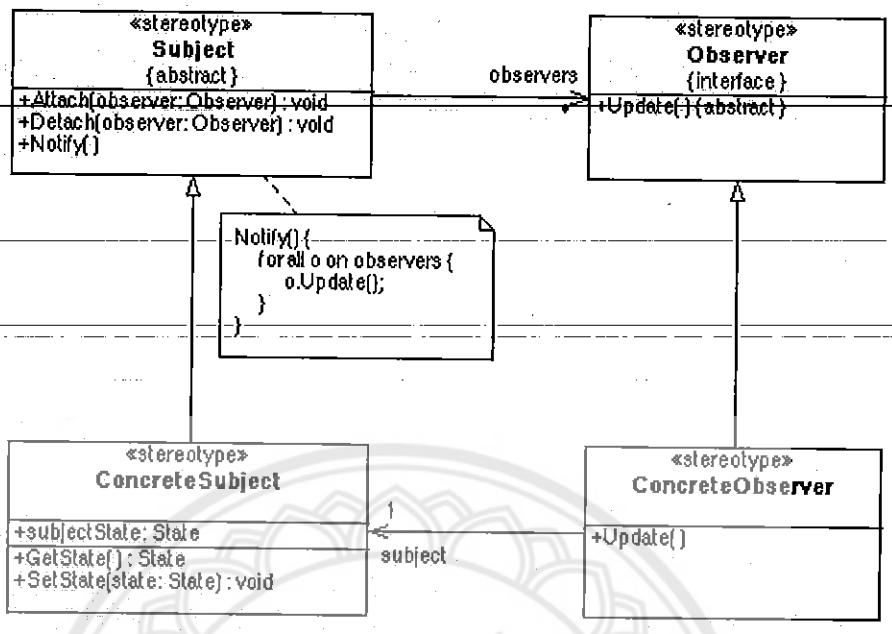
Contain C M



รูปที่ 4.34 แสดงผลการตรวจสอบ Pattern

จากรูปที่ 4.34 RSF ของ Pattern ที่นำมาตรวจสอบมีการใช้ชื่อส่วนประกอบไม่เหมือนกันกับ Design Pattern ของฐานข้อมูล แต่เมื่อนำมาตรวจสอบกับ Design Patternใน ฐานข้อมูล พบว่าเป็นMemento Pattern เนื่องจากมีส่วนประกอบและความสัมพันธ์ของส่วนประกอบตรงกับ Memento pattern

การตรวจสอบ Observer Pattern



รูปที่ 4.35 โครงสร้าง Observer Pattern

โครงสร้าง ของ Observer Pattern คือ

ComPat (Observer, Subject, ConcreteObserver, ConcreteSubject) :=

Inherit (ConcreteObserver, Observer)

& Use (Subject, Observer)

& Inherit (ConcreteSubject, Subject)

& Use (ConcreteObserver, ConcreteSubject);

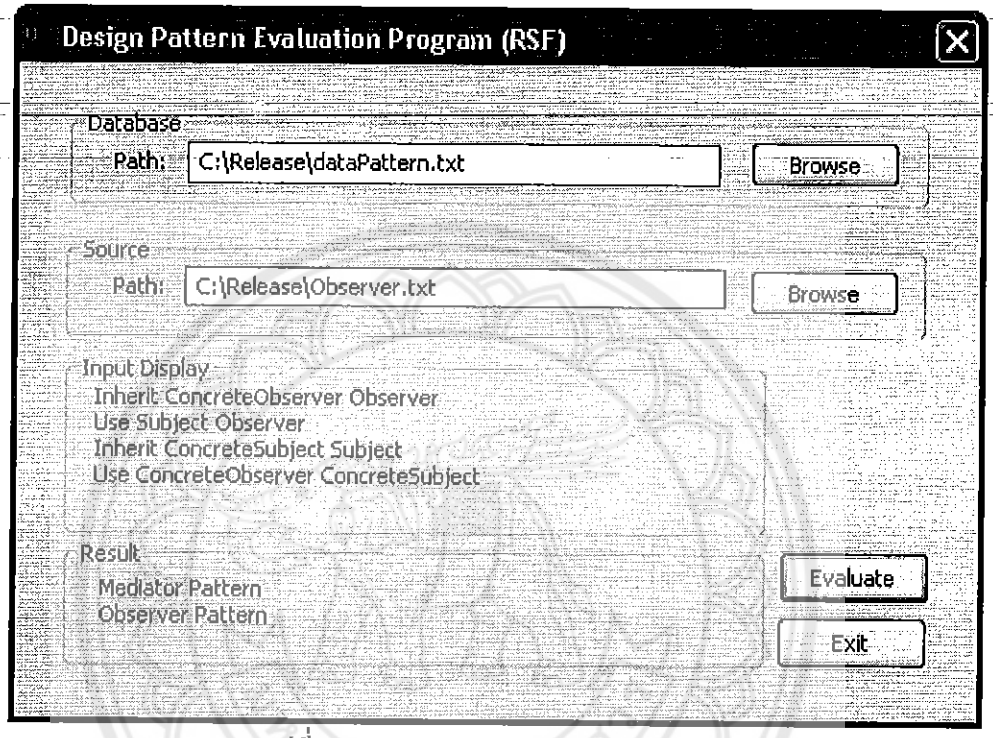
RSF ของ Pattern ที่นำมาตรวจสอบ

Inherit ConcreteObserver Observer

Use Subject Observer

Inherit ConcreteSubject Subject

Use ConcreteObserver ConcreteSubject



รูปที่ 4.36 แสดงผลการตรวจสอบ Pattern

จากรูปที่ 4.36 RSF ของ Pattern เมื่อนำมาตรวจสอบกับ Design Pattern ที่อยู่ในฐานข้อมูล จึงพบว่าเป็น Observer Pattern และ Mediator Pattern เนื่องจากมีส่วนประกอบและความสัมพันธ์ของส่วนประกอบตรงกับ Design Pattern ทั้งสอง

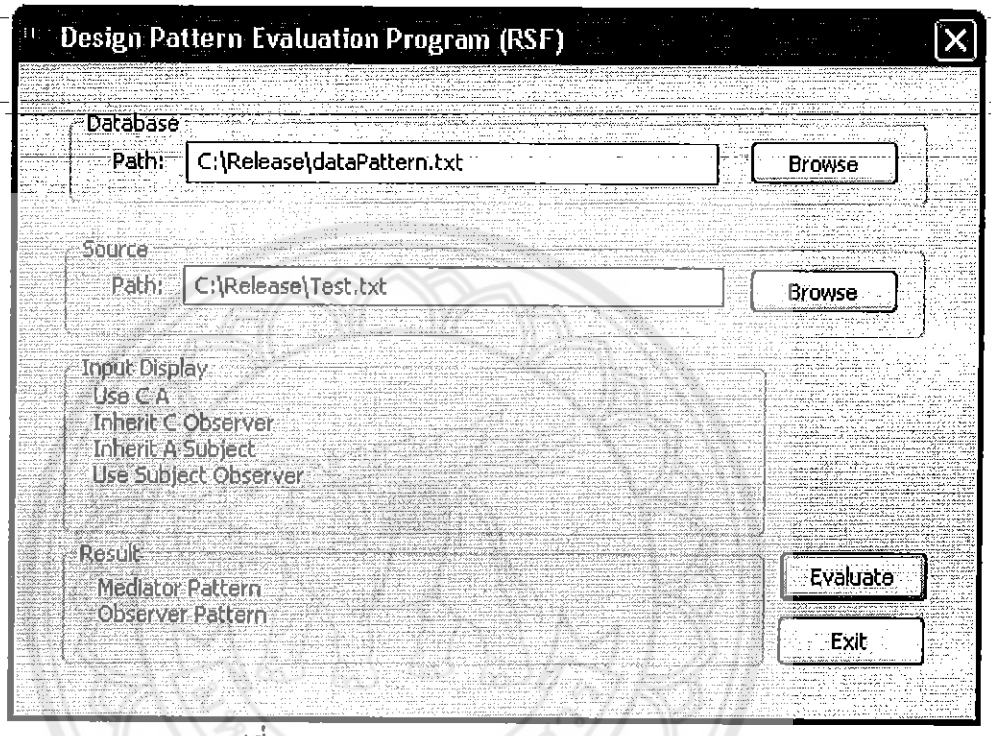
RSF ของ Pattern ที่นำมาตรวจสอบ

Use C A

Inherit C Observer

Inherit A Subject

Use Subject Observer



รูปที่ 4.37 แสดงผลการตรวจสอบ Pattern

จากรูปที่ 4.37 RSF ของ Pattern ที่นำมาตรวจสอบมีการสลับลำดับการเขียนและใช้ชื่อส่วนประกอบไม่เหมือนกันกับ Design Pattern ของฐานข้อมูล แต่เมื่อนำมาตรวจสอบกับ Design Patternใน ฐานข้อมูล พบว่าเป็น Observer Pattern และ Mediator Pattern เนื่องจากมีส่วนประกอบและความสัมพันธ์ของส่วนประกอบตรงกับ Design Pattern ทั้งสอง

RSF ของ Pattern ที่นำมาตรวจสอบ

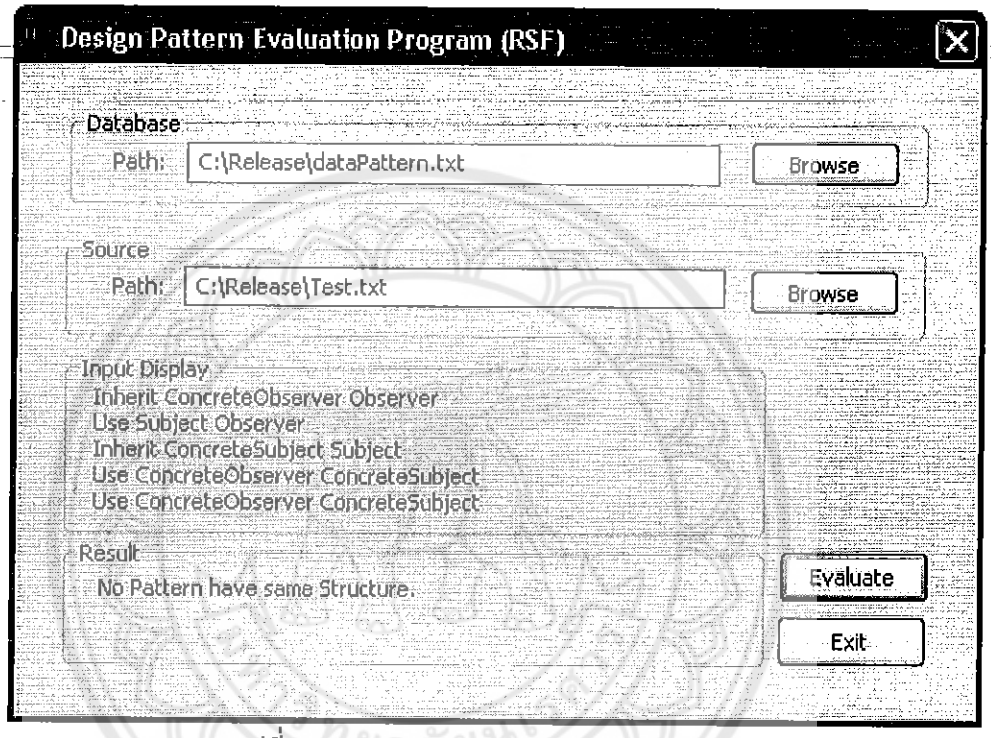
Inherit ConcreteObserver Observer

Use Subject Observer

Inherit ConcreteSubject Subject

Use ConcreteObserver ConcreteSubject

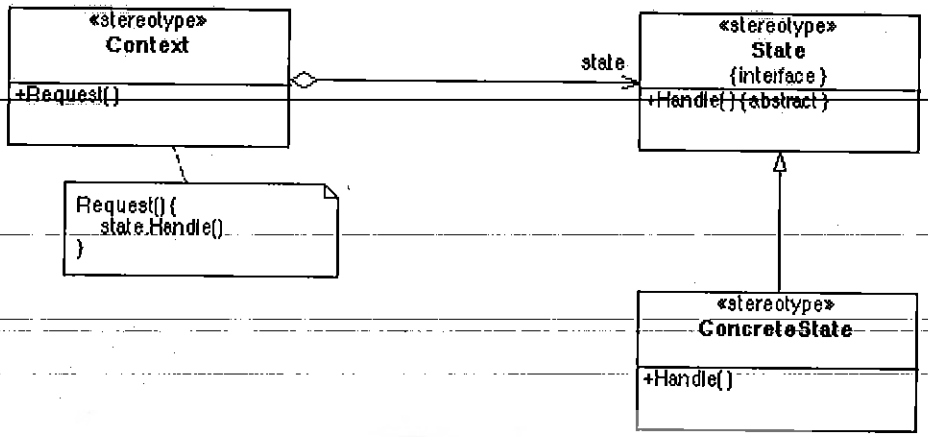
Use ConcreteObserver ConcreteSubject



รูปที่ 4.38 แสดงผลการตรวจสอบ Pattern

จากรูปที่ 4.38 RSF ของ Pattern ที่นำมาตรวจสอบถูกเพิ่มส่วนประกอบบางอย่างเข้ามา เมื่อนำมาตรวจสอบกับ โครงสร้าง ของ Design Pattern ที่อยู่ในฐานข้อมูล จึงไม่มี Design Pattern ใดเลยที่ตรงกับ Pattern ที่นำมาตรวจสอบ

การตรวจสอบ State Pattern



รูปที่ 4.39 โครงสร้าง State Pattern

โครงสร้าง ของ State Pattern คือ

ComPat (State, ConcreteState, Context) :=

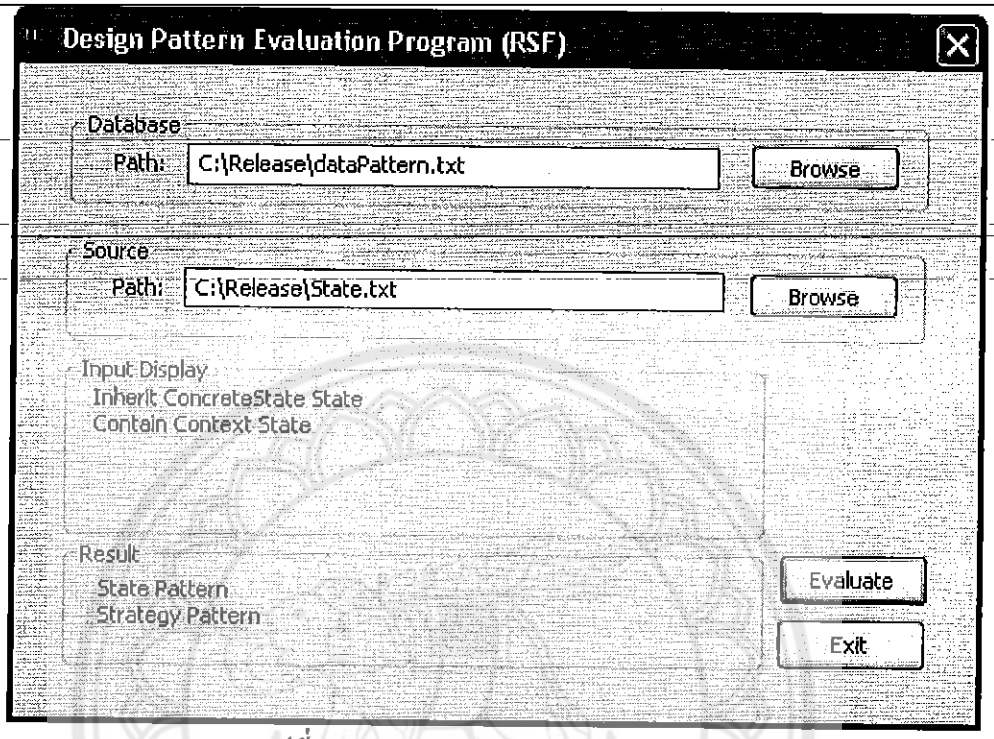
Inherit (ConcreteState, State)

& Contain (Context, State);

RSF ของ Pattern ที่นำมาตรวจสอบ

Inherit ConcreteState State

Contain Context State



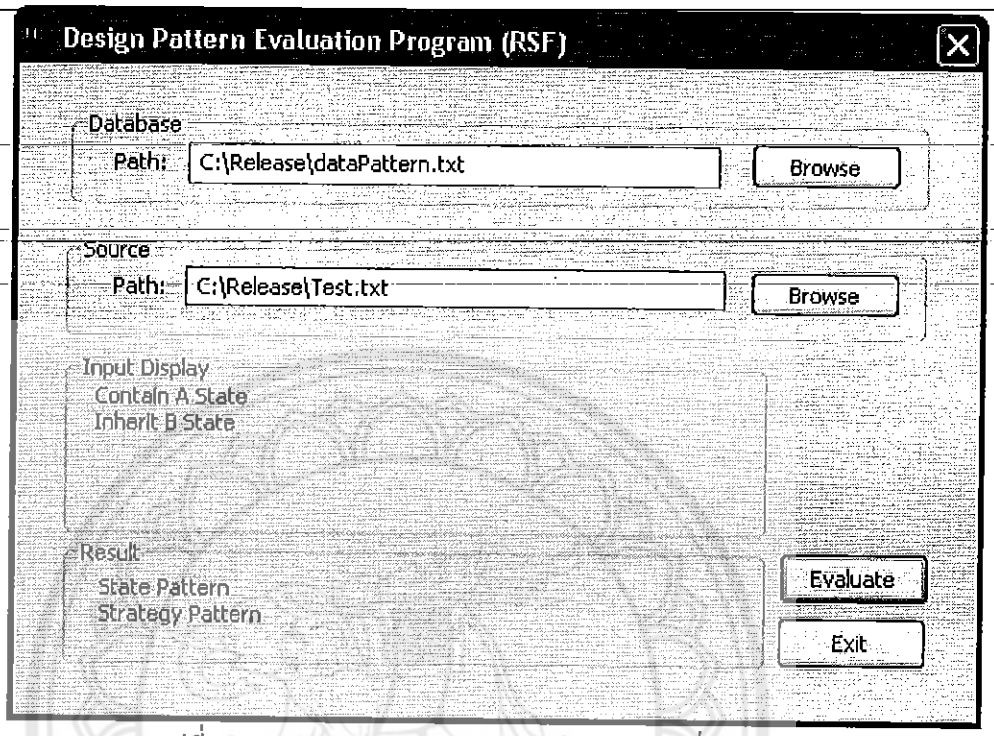
รูปที่ 4.40 แสดงผลการตรวจสอบ Pattern

จากรูปที่ 4.40 RSF ของ Pattern เมื่อนำมาตรวจสอบกับ Design Pattern ที่อยู่ในฐานข้อมูล จึงพบว่าเป็น State pattern เนื่องจากมีส่วนประกอบและความสัมพันธ์ของส่วนประกอบตรงกับ State Pattern

RSF ของ Pattern ที่นำมาตรวจสอบ

Contain A State

Inherit B State



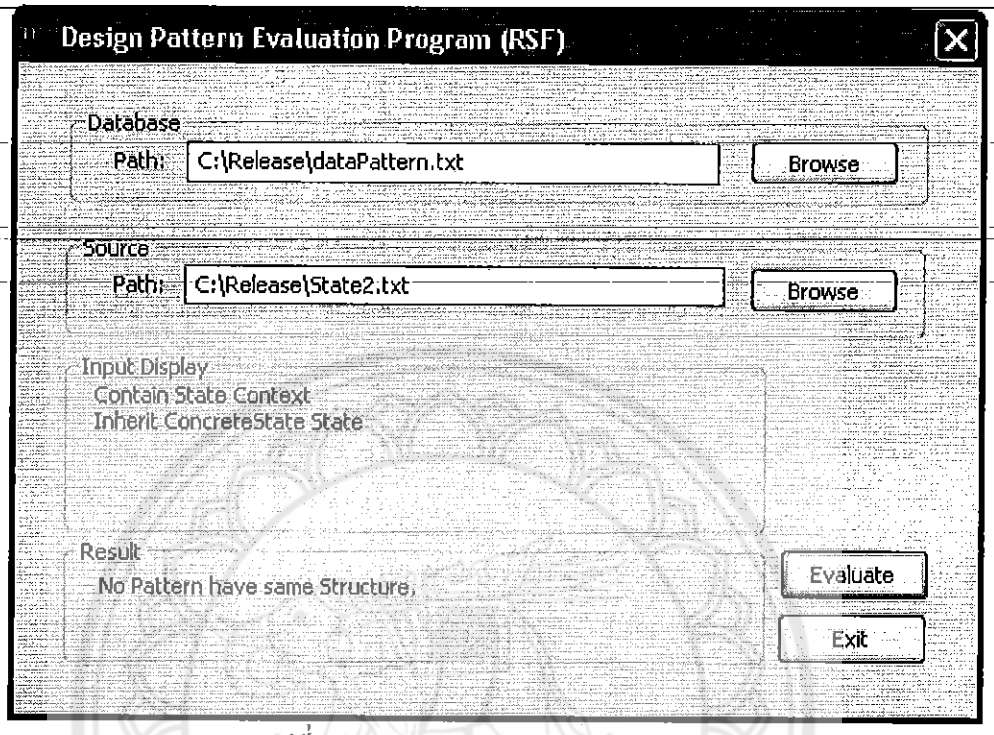
รูปที่ 4.41 แสดงผลการตรวจสอบ Pattern แบบที่ 30

จากรูปที่ 4.41 RSF ของ Pattern ที่นำมาตรวจสอบมีการสลับลำดับการเขียนและใช้ชื่อส่วนประกอบไม่เหมือนกันกับ Design Pattern ของฐานข้อมูล แต่เมื่อนำมาตรวจสอบกับ Design Patternใน ฐานข้อมูล พบว่าเป็น State Pattern เนื่องจากมีส่วนประกอบและความสัมพันธ์ของส่วนประกอบตรงกับ State Pattern

RSF ของ Pattern ที่นำมาตรวจสอบ

Contain State Context

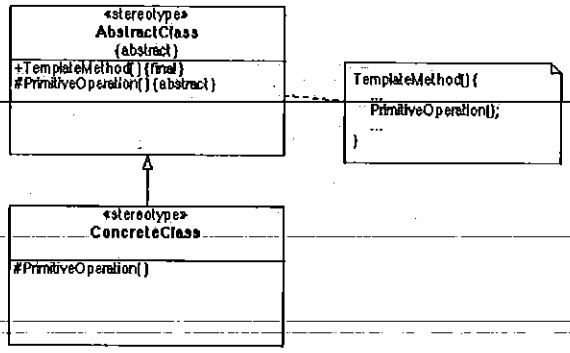
Inherit ConcreteState State



รูปที่ 4.42 แสดงผลการตรวจสอบ Pattern

จากรูปที่ 4.42 RSF ของ Pattern ที่นำมาตรวจสอบมีสลับตำแหน่งของส่วนประกอบ ดังนั้น เมื่อนำมาตรวจสอบกับ Design Pattern ที่อยู่ในฐานข้อมูล จึงไม่พบ Design Pattern ใดเลยที่ตรงกับ Pattern ที่นำมาตรวจสอบเลย

การตรวจสอบ Template-method Pattern



รูปที่ 4.43 โครงสร้าง Template-method Pattern

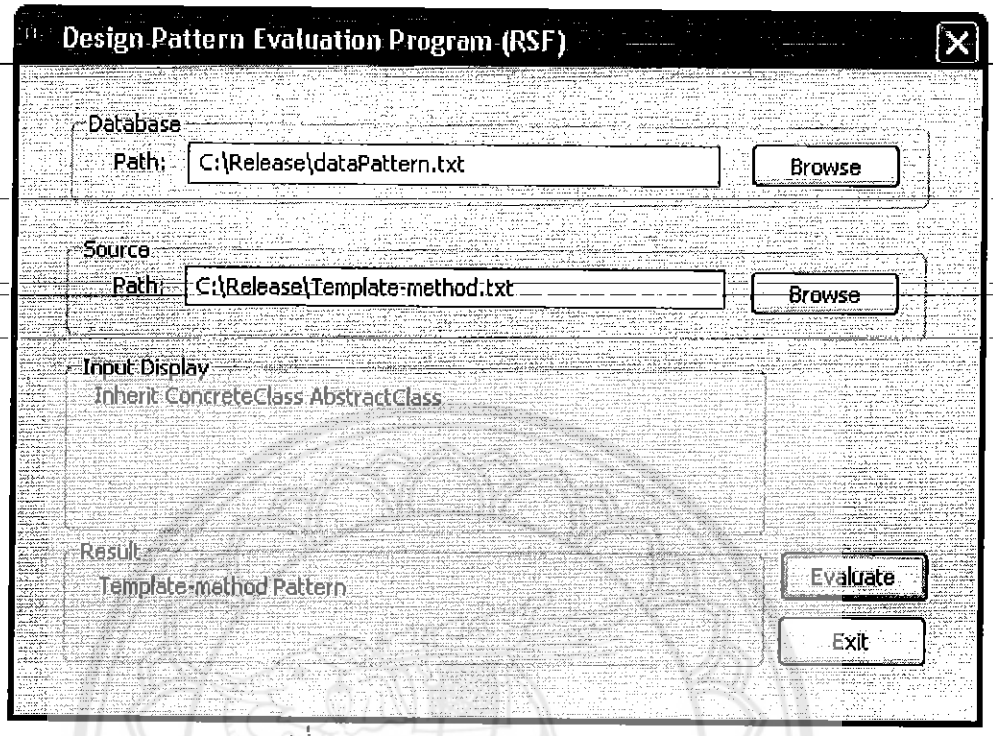
โครงสร้าง ของ Template-method Pattern คือ

ComPat (AbstractClass, ConcreteClass) :=

Inherit (ConcreteClass, AbstractClass);

RSF ของ Pattern ที่นำมาตรวจสอบ

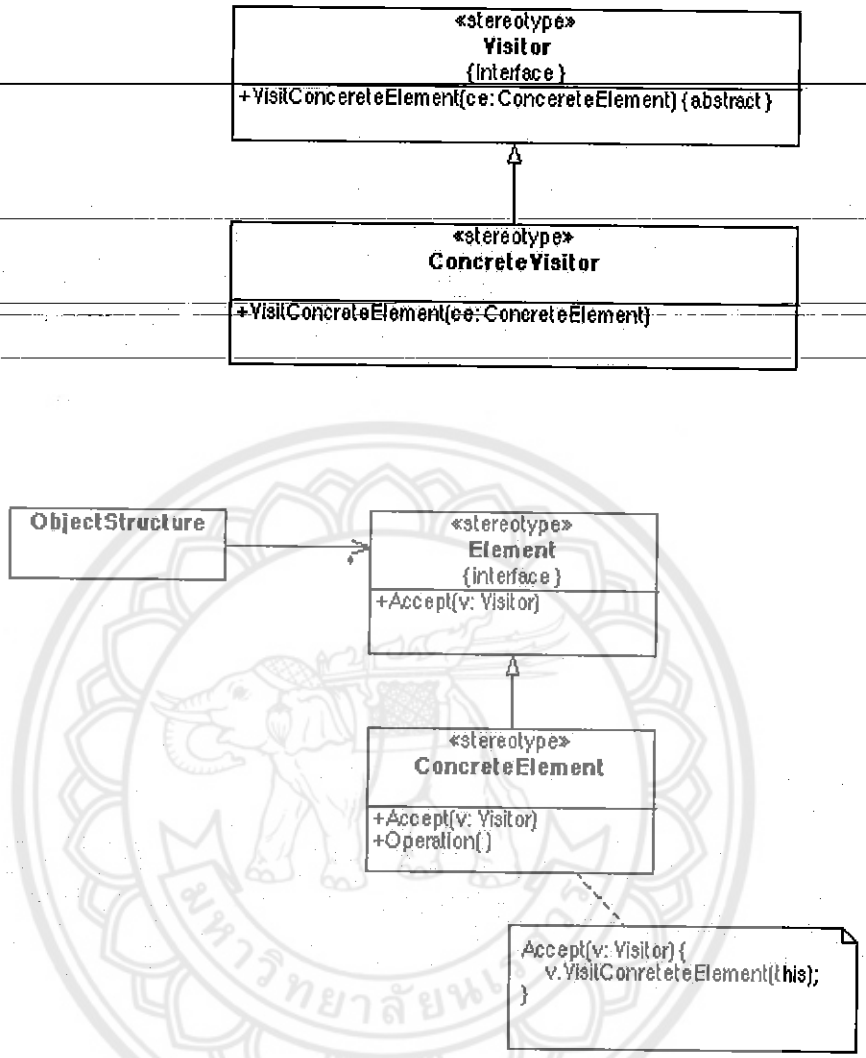
Inherit ConcreteClass AbstractClass



รูปที่ 4.44 แสดงผลการตรวจสอบ Pattern

จากรูปที่ 4.44 RSF ของ Pattern เมื่อนำมาตรวจสอบกับ Design Pattern ที่อยู่ในฐานข้อมูล จึงพบว่าเป็น Template-method Pattern เนื่องจากมีส่วนประกอบและความสัมพันธ์ของส่วนประกอบตรงกับ Template-method Pattern

การตรวจสอบ Visitor Pattern



รูปที่ 4.45 โครงสร้าง Visitor Pattern

โครงสร้าง ของ Visitor Pattern คือ

ComPat (Visitor, ConcreteVisitor, Element, ConcreteElement, ObjectStructure) :=

Inherit (ConcreteVisitor, Visitor)

& Inherit (ConcreteElement, Element)

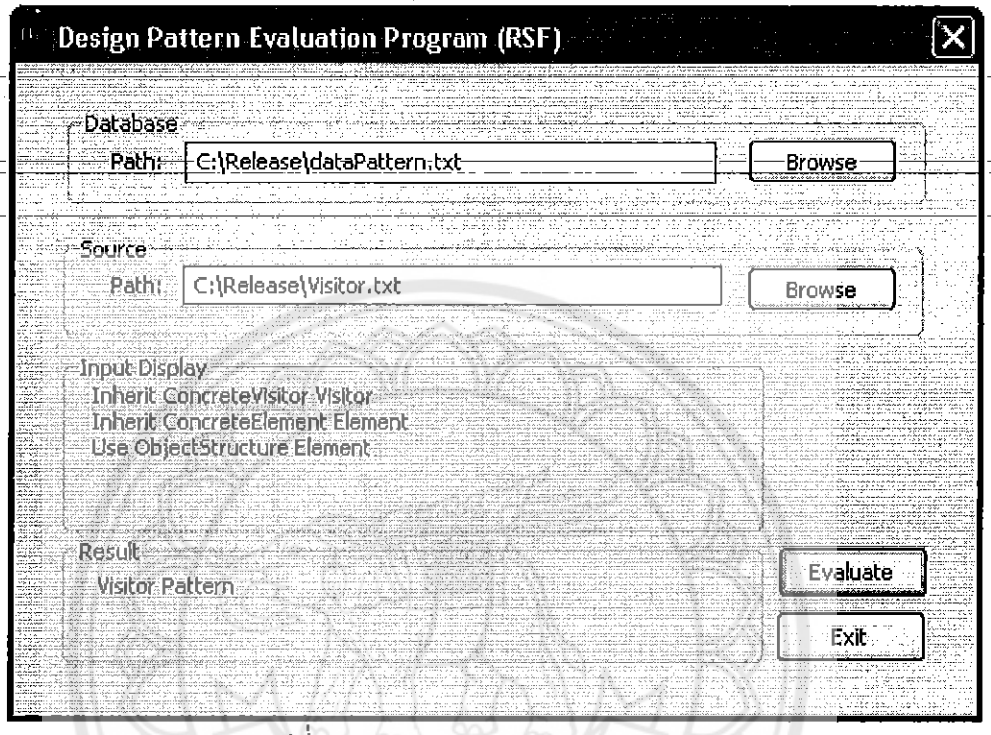
& Use (ObjectStructure, Element);

RSF ของ Pattern ที่นำมาตรวจสอบ

Inherit ConcreteVisitor Visitor

Inherit ConcreteElement Element

Use ObjectStructure Element



รูปที่ 4.46 แสดงผลการตรวจสอบ Pattern

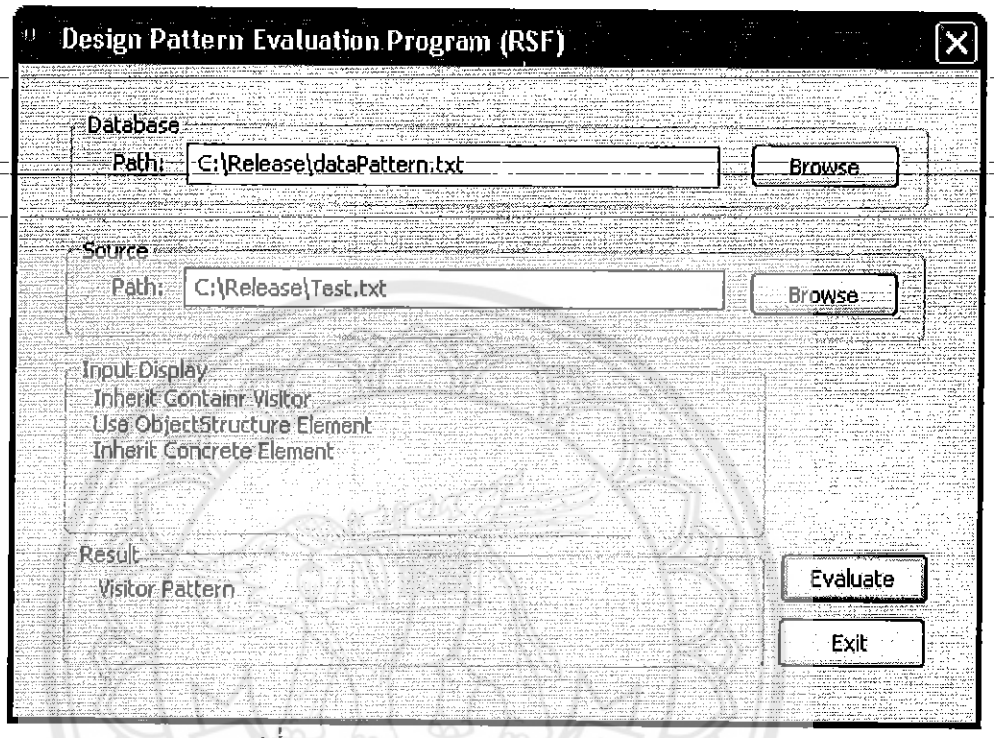
จากรูปที่ 4.46 RSF ของ Pattern เมื่อนำมาตรวจสอบกับ Design Pattern ที่อยู่ในฐานข้อมูล จึงพบว่าเป็น Visitor Pattern เนื่องจากมีส่วนประกอบและความสัมพันธ์ของส่วนประกอบตรงกับ Visitor Pattern

RSF ของ Pattern ที่นำมาตรวจสอบ

Inherit Containr Visitor

Use ObjectStructure Element

Inherit Concrete Element



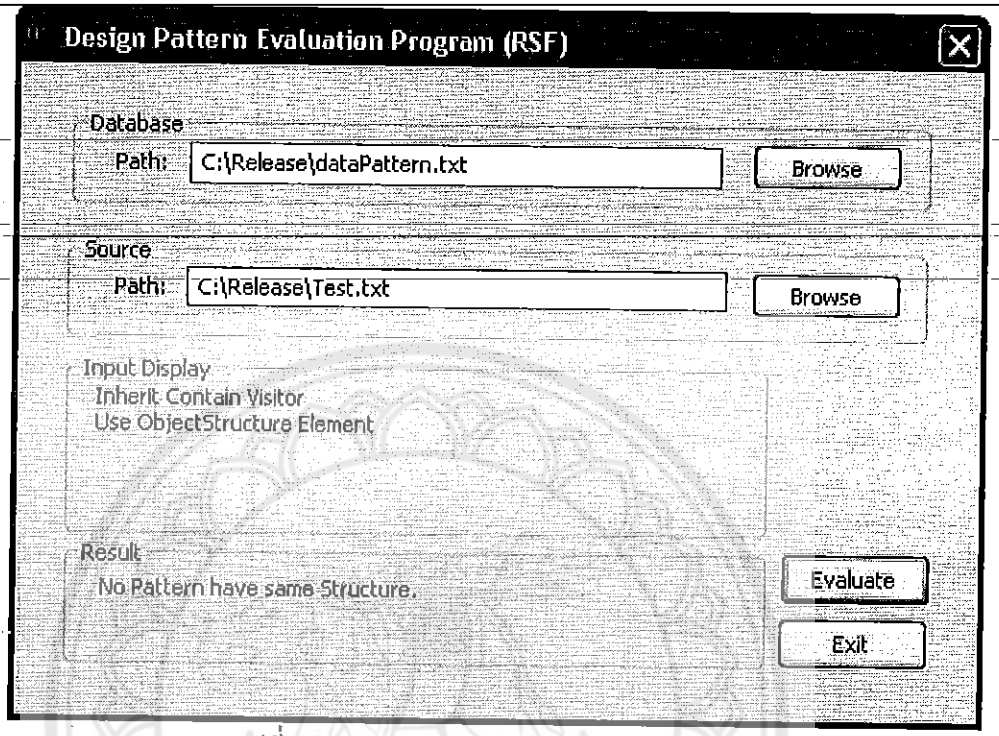
รูปที่ 4.47 แสดงผลการตรวจสอบ Pattern

จากรูปที่ 4.47 RSF ของ Pattern ที่นำมาตรวจสอบมีการสลับลำดับการเขียนและใช้ชื่อส่วนประกอบไม่เหมือนกันกับ Design Pattern ของฐานข้อมูล แต่เมื่อนำมาตรวจสอบกับ Design Patternใน ฐานข้อมูล พบว่าเป็น Visitor Pattern เนื่องจากมีส่วนประกอบและความสัมพันธ์ของส่วนประกอบตรงกับ Visitor Pattern

RSF ของ Pattern ที่นำมาตรวจสอบ

Inherit Contain Visitor

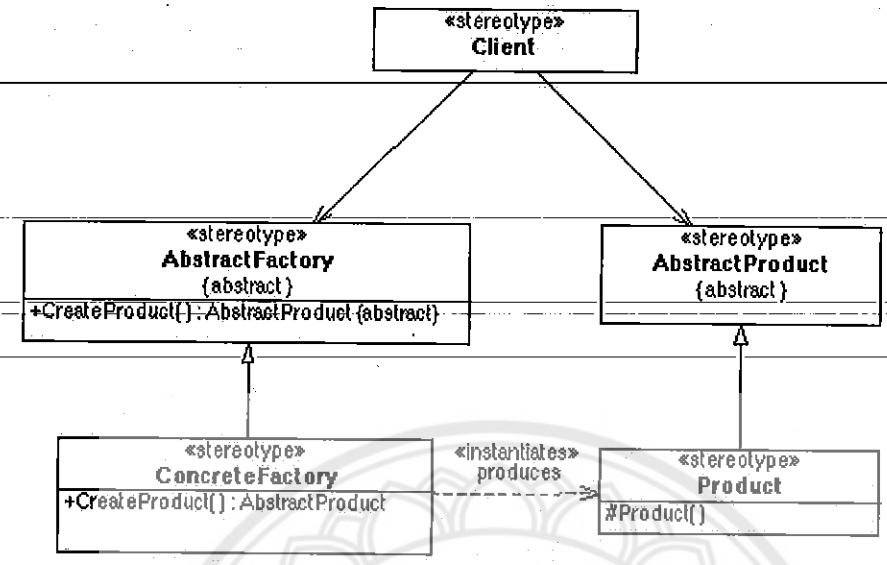
Use ObjectStructure Element



รูปที่ 4.48 แสดงผลการตรวจสอบ Pattern

จากรูปที่ 4.48 RSF ของ Pattern ที่นำมาตรวจสอบถูกตัดองค์ประกอบบางอย่างไป ดังนั้น เมื่อนำมาตรวจสอบกับ Design Pattern ที่อยู่ในฐานข้อมูล จึงไม่พบ Design Pattern ใดเลยที่ตรงกับ Pattern ที่นำมาตรวจสอบเลย

การตรวจสอบ Abstract-factory Pattern



รูปที่ 4.49 โครงสร้าง Abstract-factory Pattern

โครงสร้าง ของ Abstract-factory Pattern คือ

ComPat (Client, AbstractFactory, AbstractProduct, ConcreteFactory, Product) :=

Inherit (Product , AbstractProduct)

& Inherit (ConcreteFactory, AbstractFactory)

& Use (Client, AbstractFactory)

& Use (Client, AbstractProduct)

& Product (ConcreteFactory, Product);

RSF ของ Pattern ที่นำมาตรวจสอบ

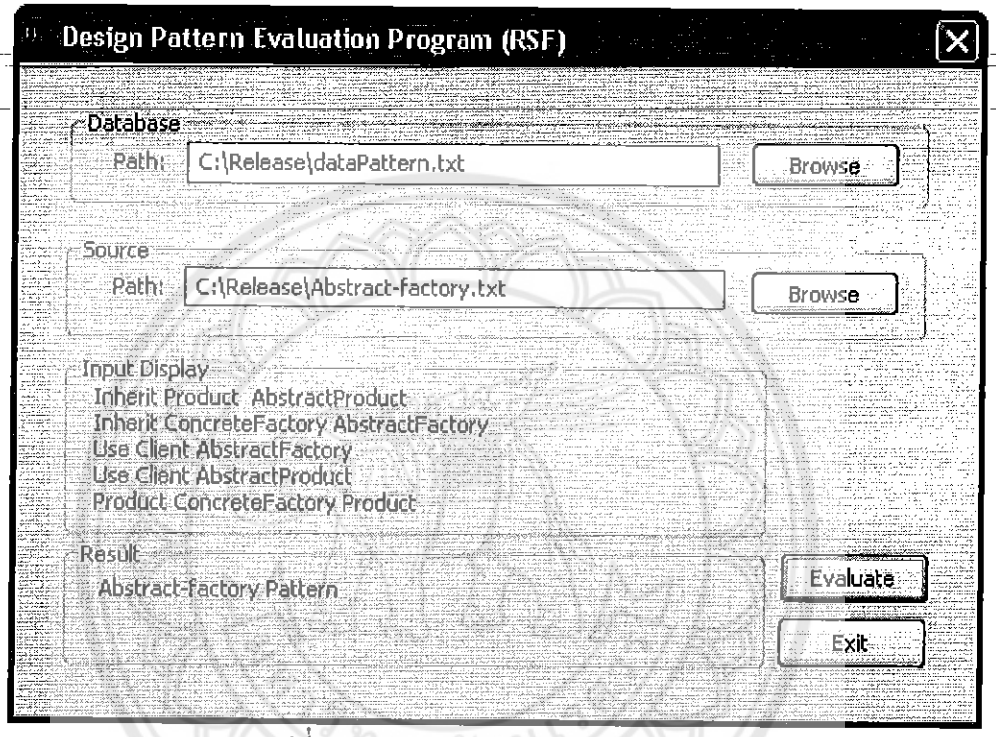
Inherit Product AbstractProduct

Inherit ConcreteFactory AbstractFactory

Use Client AbstractFactory

Use Client AbstractProduct

Product ConcreteFactory Product



รูปที่ 4.50 แสดงผลการตรวจสอบ Pattern

จากรูปที่ 4.50 RSF ของ Pattern ที่มือนำมาตรวจสอบกับ Design Pattern ที่อยู่ในฐานข้อมูล จึงพบว่าเป็น Abstract-factory Pattern เนื่องจากมีส่วนประกอบและความสัมพันธ์ของส่วนประกอบตรงกับ Abstract-factory Pattern

RSF ของ Pattern ที่นำมาตรวจสอบ

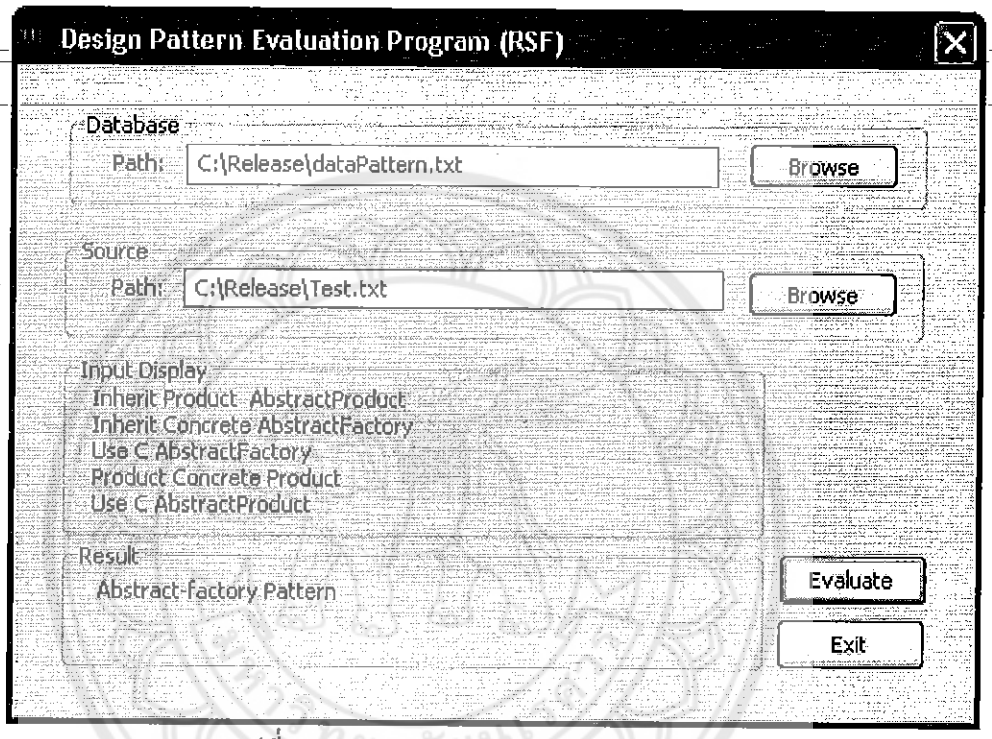
Inherit Product AbstractProduct

Inherit Concrete AbstractFactory

Use C AbstractFactory

Product Concrete Product

Use C AbstractProduct



รูปที่ 4.51 แสดงผลการตรวจสอบ Pattern

จากรูปที่ 4.51 RSF ของ Pattern ที่นำมาตรวจสอบมีการสลับลำดับการเขียนและใช้ชื่อส่วนประกอบไม่เหมือนกันกับ Design Pattern ของฐานข้อมูล แต่เมื่อนำมาตรวจสอบกับ Design Patternใน ฐานข้อมูล พบว่าเป็น Abstract-factory Pattern เนื่องจากมีส่วนประกอบและความสัมพันธ์ของส่วนประกอบตรงกับ Abstract-factory Pattern

RSF ของ Pattern ที่นำมาตรวจสอบ

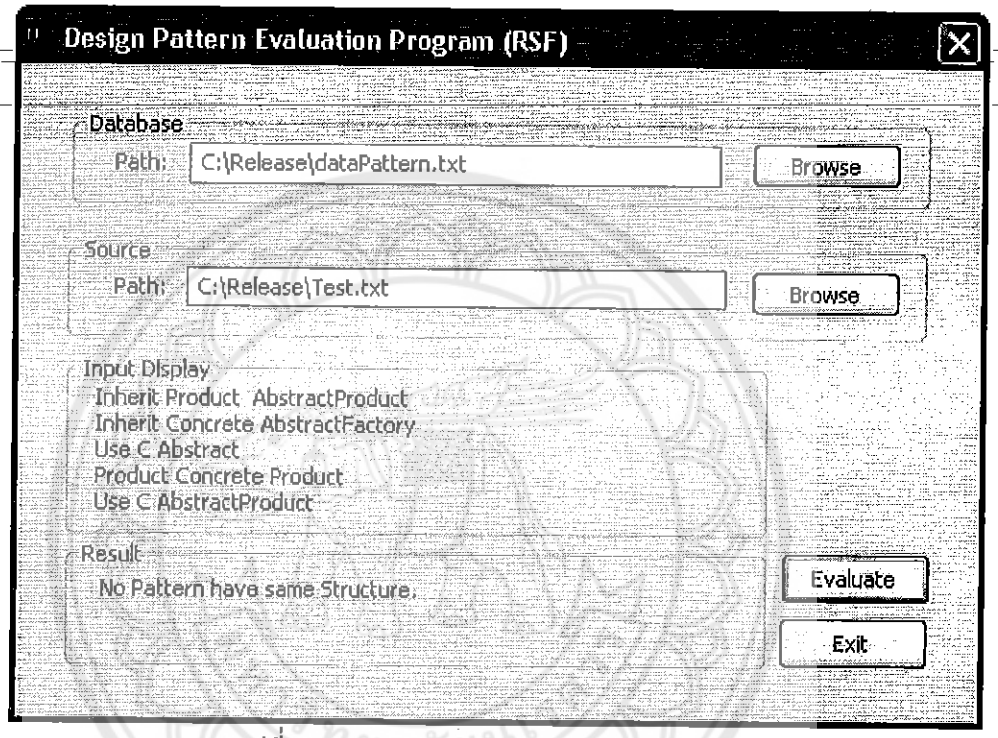
Inherit Product AbstractProduct

Inherit Concrete AbstractFactory

Use C Abstract

Product Concrete Product

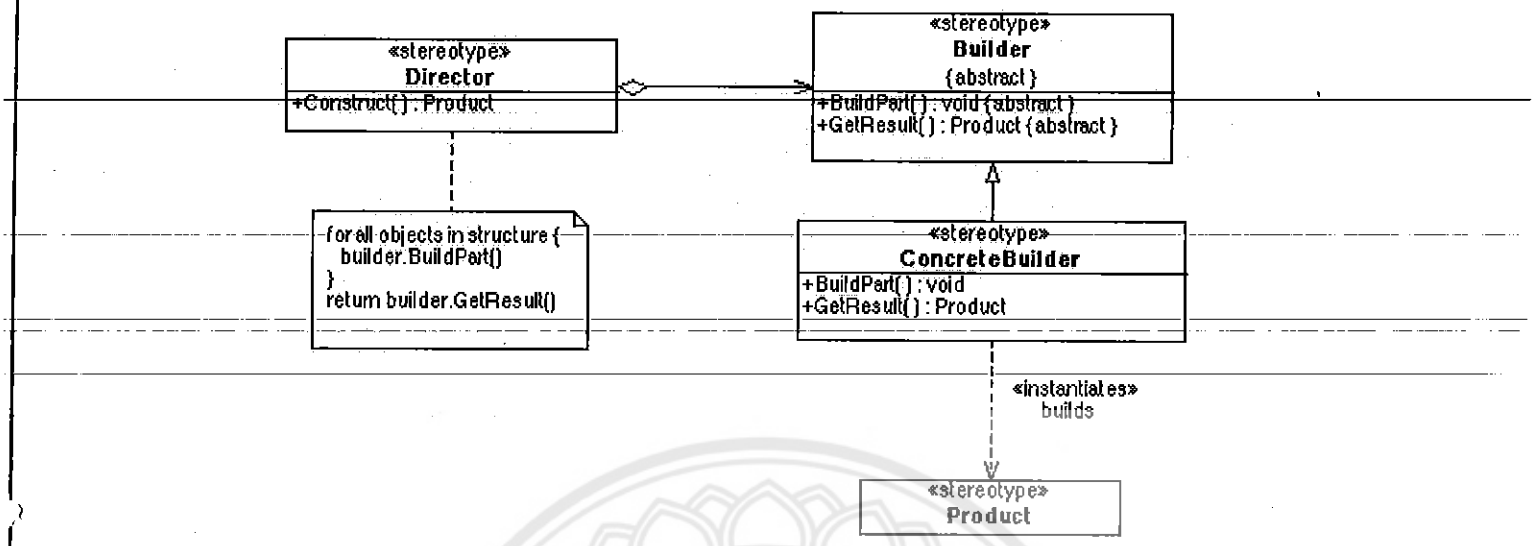
Use C AbstractProduct



รูปที่ 4.52 แสดงผลการตรวจสอบ Pattern

จากรูปที่ 4.52 RSF ของ Pattern ที่นำมาตรวจสอบมีส่วนประกอบเหมือนกัน แต่ใช้ชื่อต่างกัน ดังนั้น เมื่อนำมาตรวจสอบกับ Design Pattern ที่อยู่ในฐานข้อมูล จึงไม่พบ Design Pattern ใดเลยที่ตรงกับ Pattern ที่นำมาตรวจสอบเลย

การตรวจสอบ Builder Pattern



รูปที่ 4.53 โครงสร้าง Builder Pattern

โครงสร้าง ของ Builder Pattern คือ

ComPat (Builder, Director, ConcreteBuilder, Product) :=

Contain (Director, Builder)

& Inherit (ConcreteBuilder, Builder)

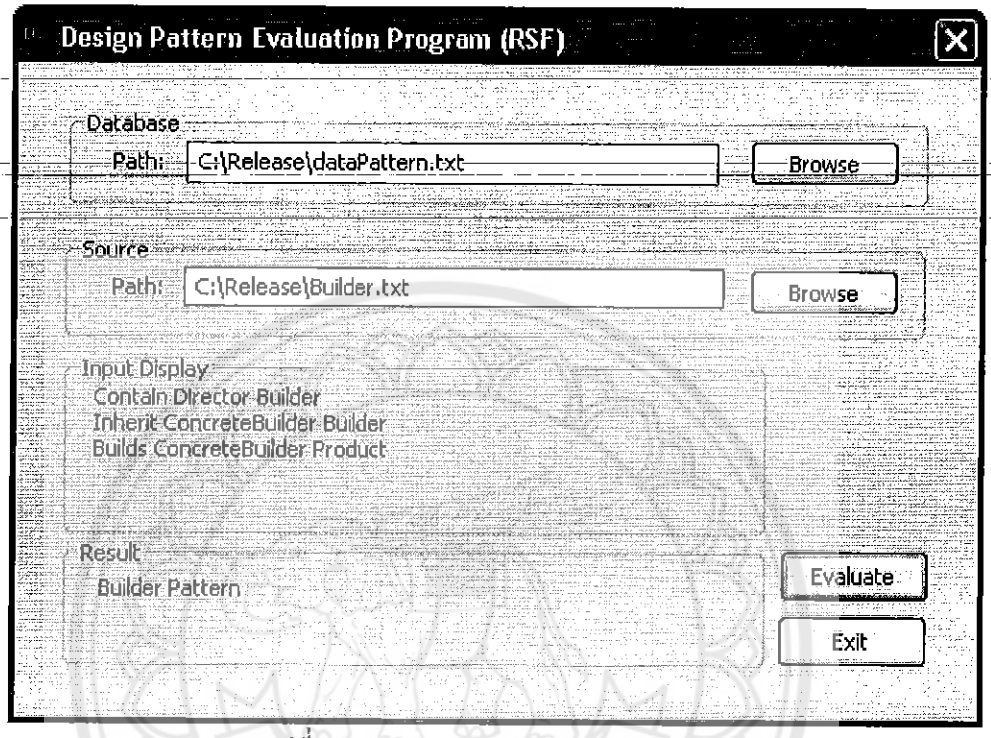
& Builds (ConcreteBuilder, Product);

RSF ของ Pattern ที่นำมาตรวจสอบ

Contain Director Builder

Inherit ConcreteBuilder Builder

Builds ConcreteBuilder Product



รูปที่ 4.54 แสดงผลการตรวจสอบ Pattern

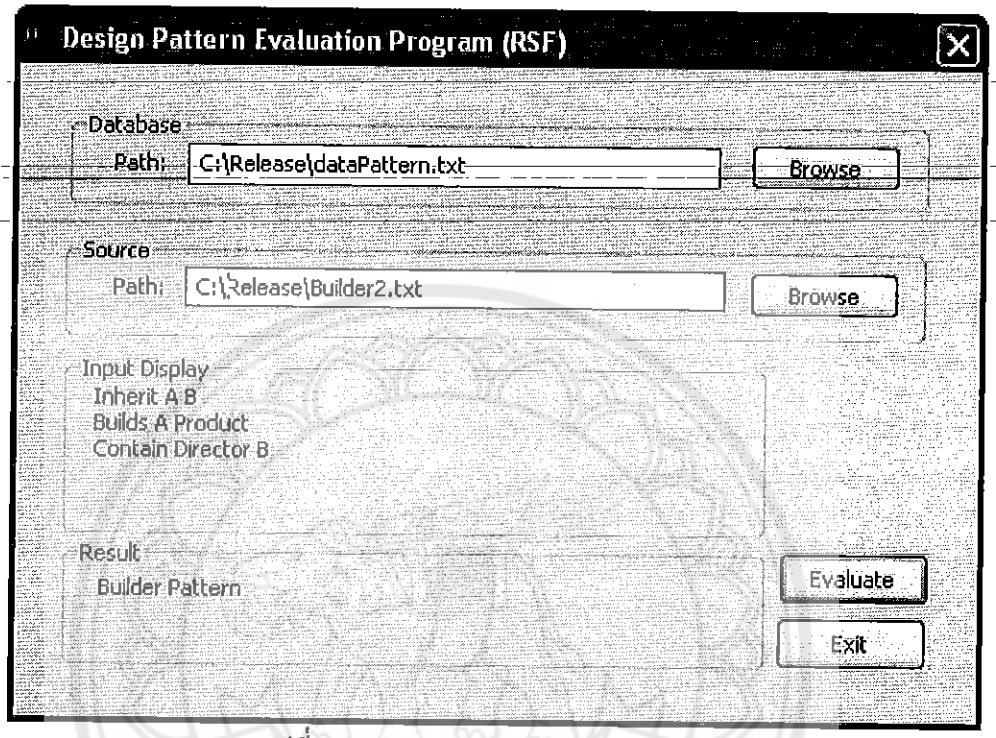
จากรูปที่ 4.54 RSF ของ Pattern เมื่อนำมาตรวจสอบกับ Design Pattern ที่อยู่ในฐานข้อมูล จึงพบว่าเป็น Builder Pattern เนื่องจากมีส่วนประกอบและความสัมพันธ์ของส่วนประกอบตรงกับ Builder Pattern

RSF ของ Pattern ที่นำมาตรวจสอบ

Inherit A B

Builds A Product

Contain Director B



รูปที่ 4.55 แสดงผลการตรวจสอบ Pattern

จากรูปที่ 4.55 RSF ของ Pattern ที่นำมาตรวจสอบมีการสลับลำดับการเขียนและใช้ชื่อส่วนประกอบไม่เหมือนกันกับใน Design Pattern ของฐานข้อมูล แต่เมื่อนำมาตรวจสอบกับ Design Pattern ใน ฐานข้อมูล พบว่าเป็น Builder Pattern เนื่องจากมีส่วนประกอบและความสัมพันธ์ของส่วนประกอบตรงกับ Builder Pattern

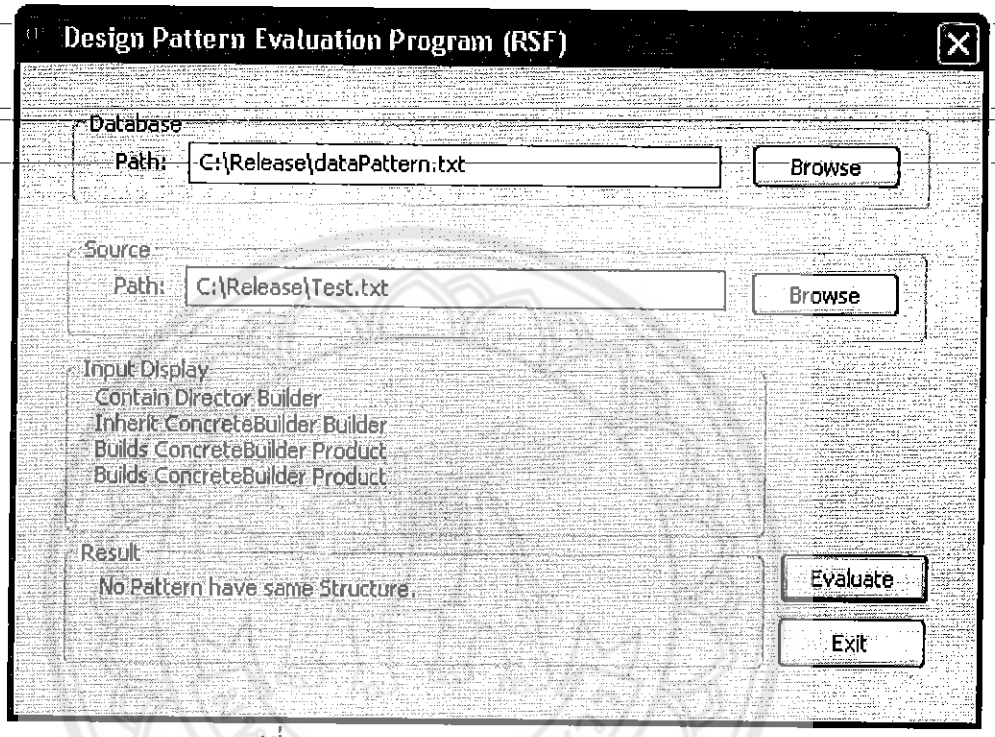
RSF ของ Pattern ที่นำมาตรวจสอบ

Contain Director Builder

Inherit ConcreteBuilder Builder

Builds ConcreteBuilder Product

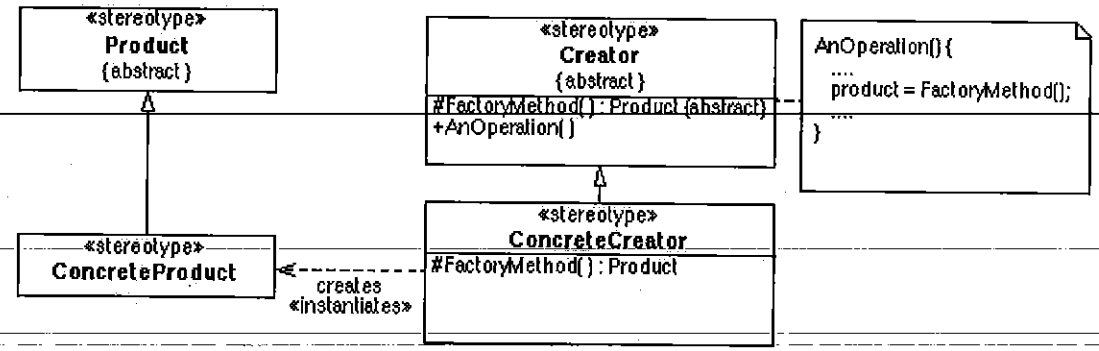
Builds ConcreteBuilder Product



รูปที่ 4.56 แสดงผลการตรวจสอบ Pattern

จากรูปที่ 4.56 RSF ของ Pattern ที่นำมาตรวจสอบมีส่วนประกอบอื่นเพิ่มขึ้นดังนั้น เมื่อนำมาตรวจสอบกับ Design Pattern ที่อยู่ในฐานข้อมูล จึงไม่พบ Design Pattern ใดเลยที่ตรงกับ Pattern ที่นำมาตรวจสอบเลย

การตรวจสอบ Factory-Method Pattern



รูปที่ 4.57 โครงสร้าง Factory-Method Pattern

โครงสร้าง ของ Factory-method Pattern คือ

ComPat (Product, ConcreteProduct, Creator, ConcreteCreator) :=

Inherit (ConcreteProduct, Product)

& Creates (ConcreteCreator, ConcreteProduct)

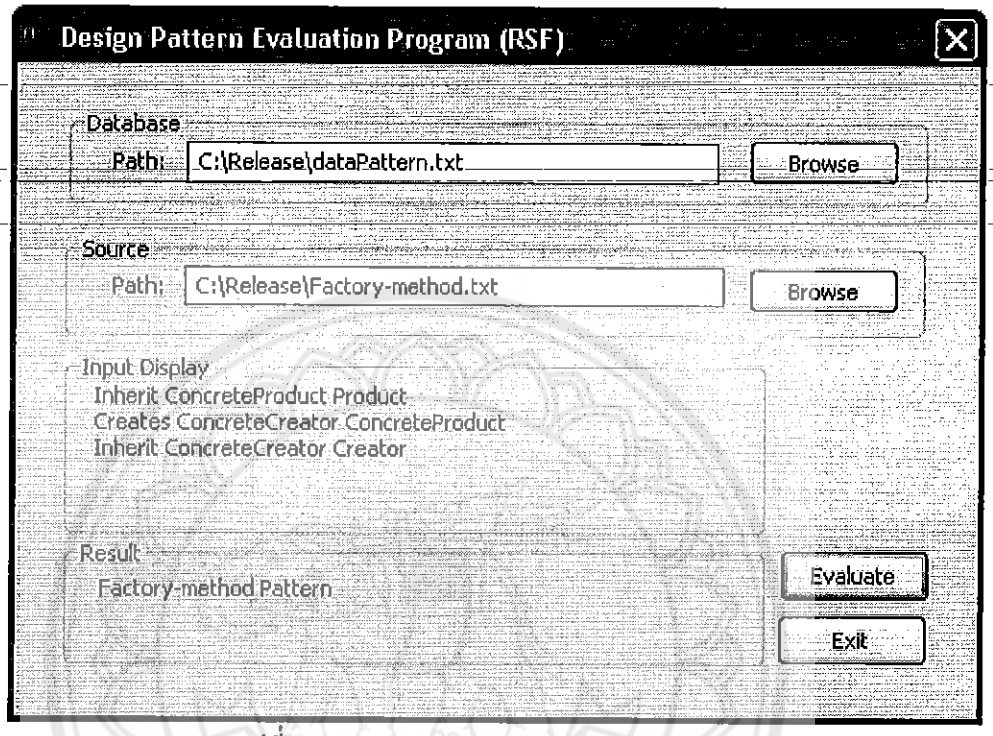
& Inherit (ConcreteCreator, Creator);

RSF ของ Pattern ที่นำมาตรวจสอบ

Inherit ConcreteProduct Product

Creates ConcreteCreator ConcreteProduct

Inherit ConcreteCreator Creator



รูปที่ 4.58 แสดงผลการตรวจสอบ Pattern

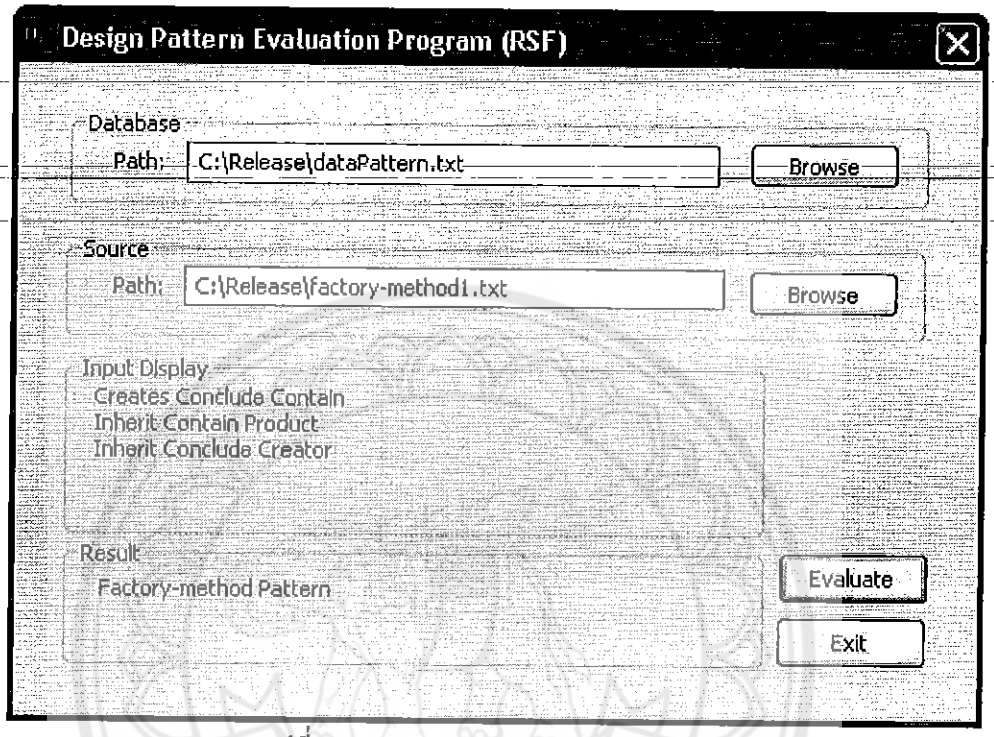
จากรูปที่ 4.58 RSF ของ Pattern เมื่อนำมาตรวจสอบกับ Design Pattern ที่อยู่ในฐานข้อมูล จึงพบว่าเป็น Factory-Method Pattern เนื่องจากมีส่วนประกอบและความสัมพันธ์ของส่วนประกอบตรงกับ Factory-Method Pattern

RSF ของ Pattern ที่นำมาตรวจสอบ

Creates Conclude Contain

Inherit Contain Product

Inherit Conclude Creator



รูปที่ 4.59 แสดงผลการตรวจสอบ Pattern

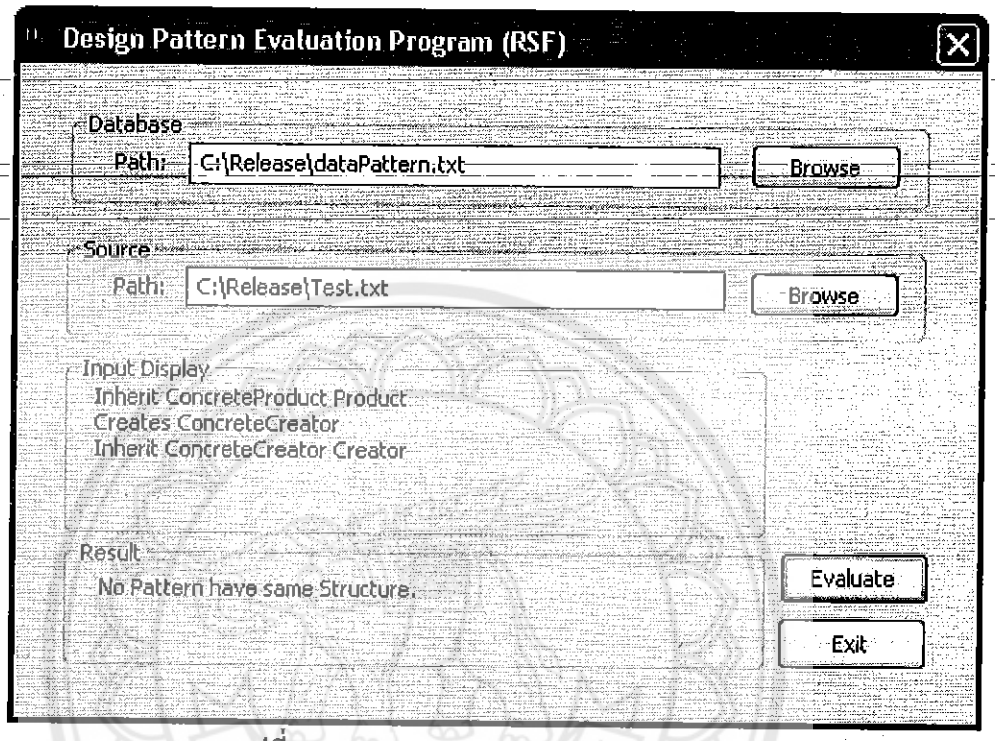
จากรูปที่ 4.59 RSF ของ Pattern ที่นำมาตรวจสอบมีการสลับลำดับการเขียนและใช้ชื่อส่วนประกอบไม่เหมือนกันกับใน Design Pattern ของฐานข้อมูล แต่เมื่อนำมาตรวจสอบกับ Design Pattern ใน ฐานข้อมูล พบว่าเป็น Factory-Method Pattern เนื่องจากมีส่วนประกอบและความสัมพันธ์ของส่วนประกอบตรงกับ Factory-Method Pattern

RSF ของ Pattern ที่นำมาตรวจสอบ

Inherit ConcreteProduct Product

Creates ConcreteCreator

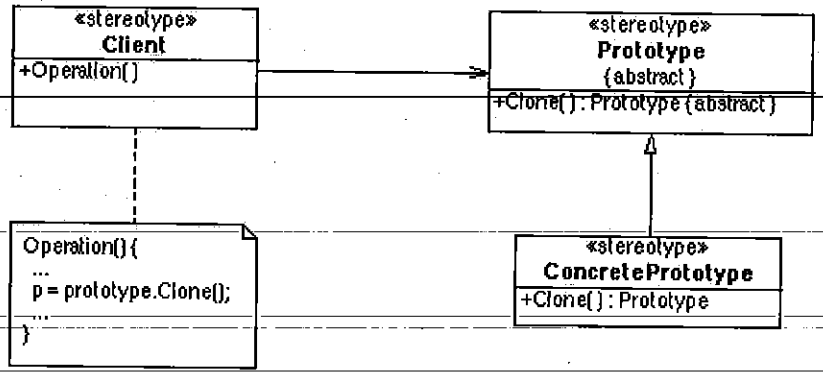
Inherit ConcreteCreator Creator



รูปที่ 4.60 แสดงผลการตรวจสอบ Pattern

จากรูปที่ 4.60 RSF ของ Pattern ที่นำมาตรวจสอบมีส่วนประกอบหายไป ดังนั้น เมื่อนำมาตรวจสอบกับ Design Pattern ที่อยู่ในฐานข้อมูล จึงไม่พบ Design Pattern ใดเลยที่ตรงกับ Pattern ที่นำมาตรวจสอบเลย

การตรวจสอบ Prototype Pattern



รูปที่ 4.61 โครงสร้าง Prototype Pattern

โครงสร้าง ของ Prototype Pattern คือ

ComPat (Prototype, ConcretePrototype, Client) :=

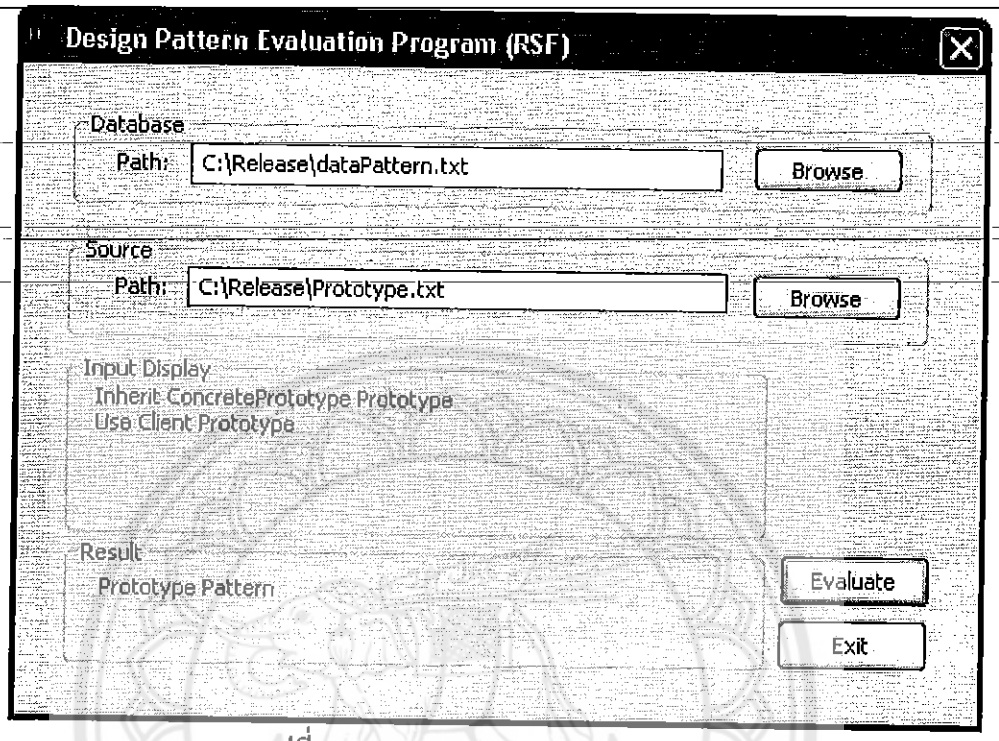
Inherit (ConcretePrototype, Prototype)

& Use (Client, Prototype);

RSF ของ Pattern ที่นำมาตรวจสอบ

Inherit ConcretePrototype Prototype

Use Client Prototype



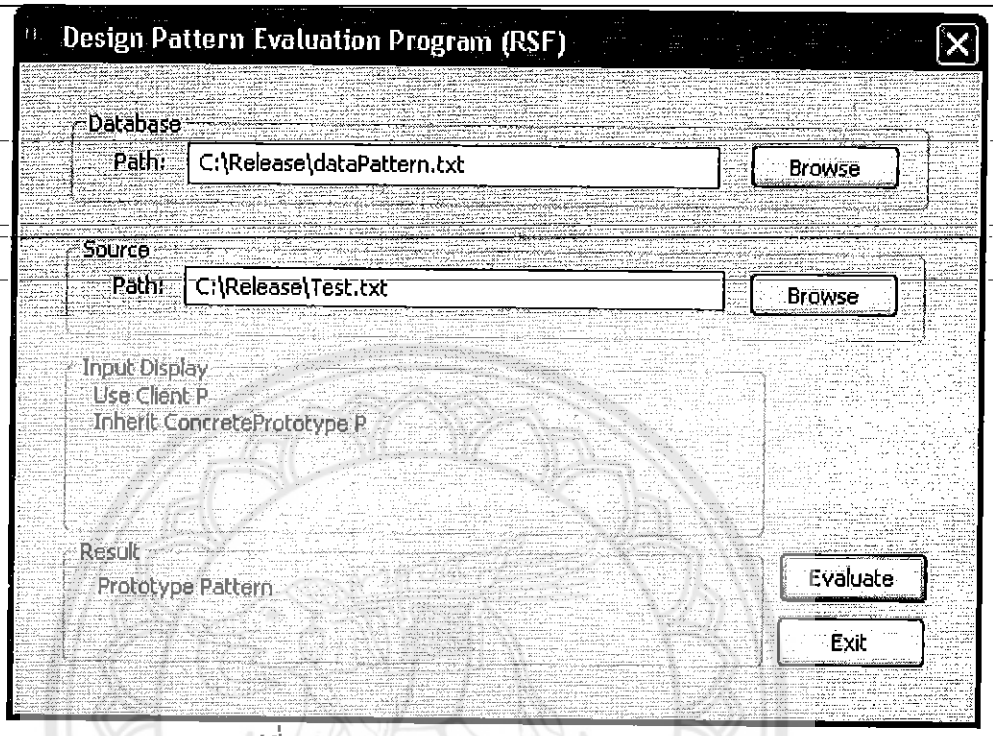
รูปที่ 4.62 แสดงผลการตรวจสอบ Pattern

จากรูปที่ 4.62 RSF ของ Pattern ที่ ดังนั้น เมื่อนำมาตรวจสอบกับ Design Pattern ที่อยู่ในฐานข้อมูล จึงพบว่าเป็น Prototype Pattern เนื่องจากมีส่วนประกอบและความสัมพันธ์ของส่วนประกอบตรงกับ Prototype Pattern

RSF ของ Pattern ที่นำมาตรวจสอบ

Use Client P

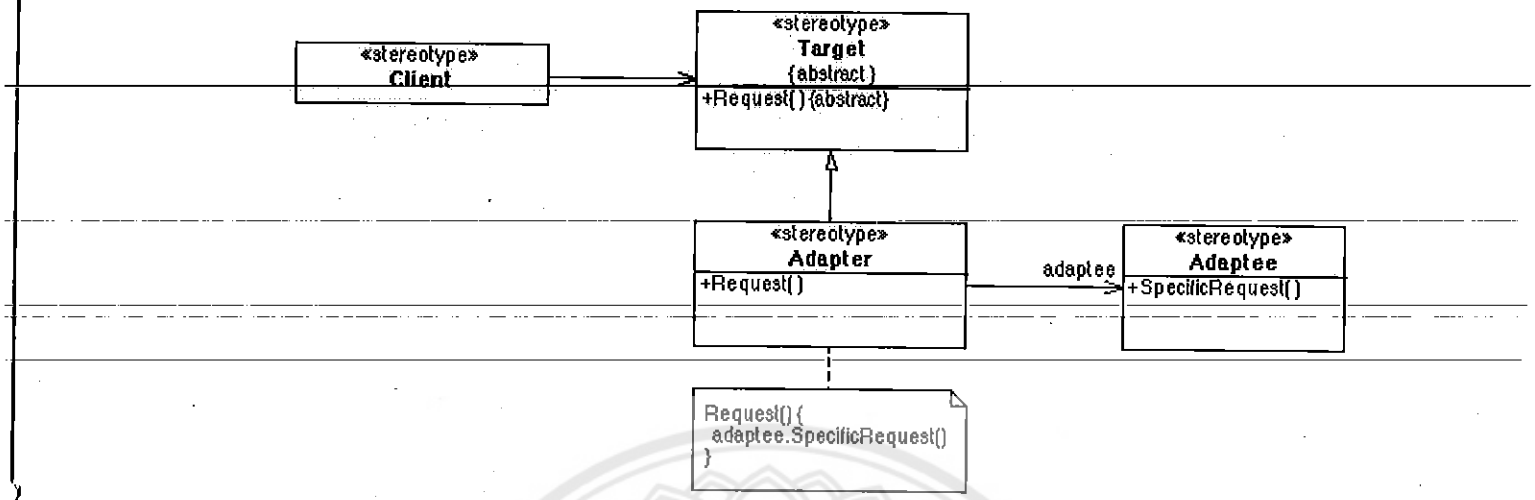
Inherit ConcretePrototype P



รูปที่ 4.63 แสดงผลการตรวจสอบ Pattern

จากรูปที่ 4.63 RSF ของ Pattern ที่นำมาตรวจสอบมีการสลับลำดับการเขียนและใช้ชื่อส่วนประกอบไม่เหมือนกันกับใน Design Pattern ของฐานข้อมูล แต่เมื่อนำมาตรวจสอบกับ Design Pattern ใน ฐานข้อมูล พบว่าเป็น Prototype Pattern เนื่องจากมีส่วนประกอบและความสัมพันธ์ของส่วนประกอบตรงกับ Prototype Pattern

การตรวจสอบ Adapter Pattern



รูปที่ 4.64 โครงสร้าง Adapter Pattern

โครงสร้าง ของ Adapter Pattern คือ

ComPat (Target, Adapter, Adaptee, Client) :=

Inherit (Adapter, Target)

& Use (Adapter, Adaptee)

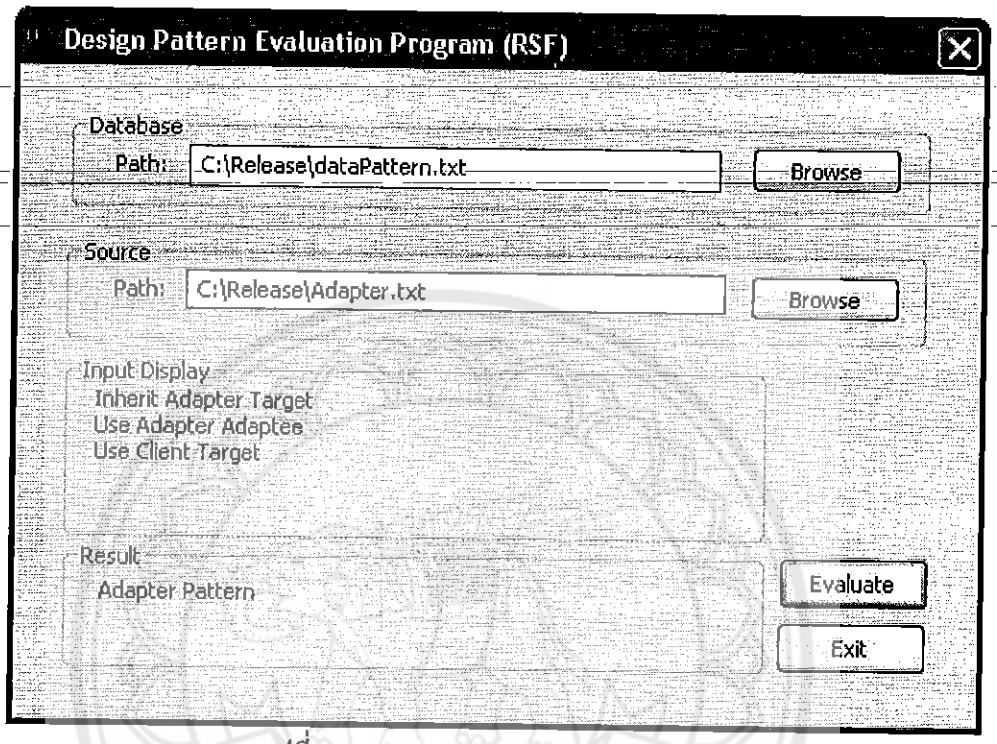
& Use (Client, Target);

RSF ของ Pattern ที่นำมาตรวจสอบ

Inherit Adapter Target

Use Adapter Adaptee

Use Client Target



รูปที่ 4.65 แสดงผลการตรวจสอบ Pattern

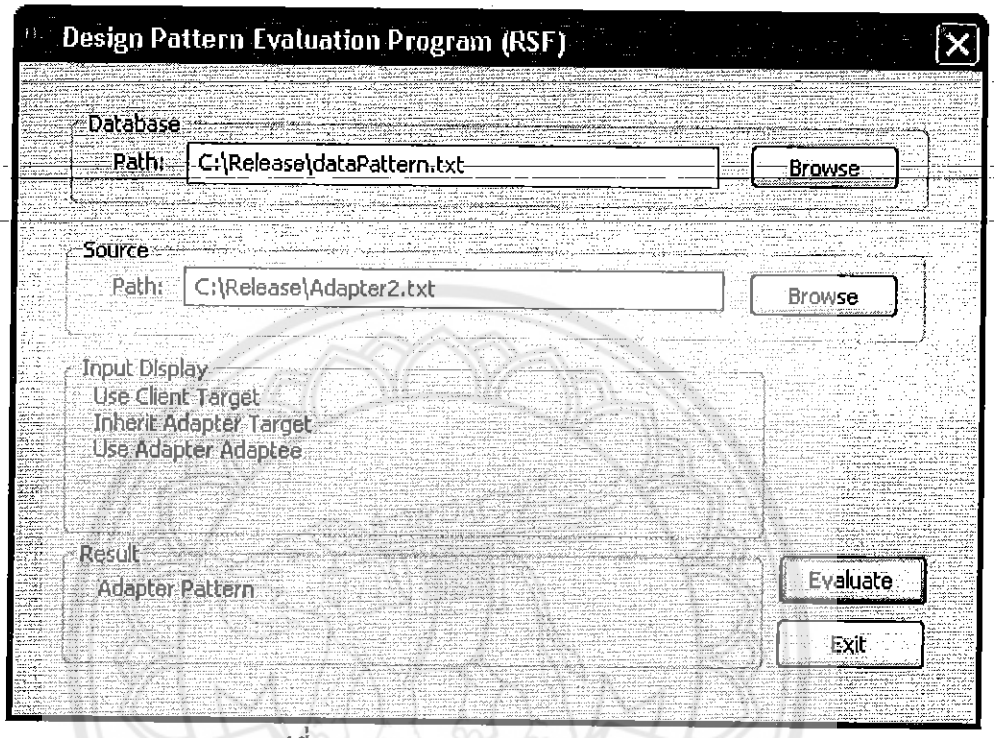
จากรูปที่ 4.65 RSF ของ Pattern เมื่อนำมาตรวจสอบกับ Design Pattern ที่อยู่ในฐานข้อมูล จึงพบว่าเป็น Adapter Pattern เนื่องจากมีส่วนประกอบและความสัมพันธ์ของส่วนประกอบตรงกับ Adapter Pattern

RSF ของ Pattern ที่นำมาตรวจสอบ

Use Client Target

Inherit Adapter Target

Use Adapter Adaptee



รูปที่ 4.66 แสดงผลการตรวจสอบ Pattern

จากรูปที่ 4.66 RSF ของ Pattern ที่นำมาตรวจสอบมีการสลับลำดับการเขียนและใช้ชื่อส่วนประกอบไม่เหมือนกันกับใน Design Pattern ของฐานข้อมูล แต่เมื่อนำมาตรวจสอบกับ Design Pattern ใน ฐานข้อมูล พบว่าเป็น Adapter Pattern เนื่องจากมีส่วนประกอบและความสัมพันธ์ของส่วนประกอบตรงกับ Adapter Pattern

จากผลการทดลองจะเห็นได้ว่า โครงสร้าง ของ Pattern ที่นำมาตรวจสอบนั้นมีลักษณะการเขียนที่แตกต่างกันออกไปจาก Design Pattern ที่มีอยู่จริง เช่น การใช้คำ การเรียงของลำดับแถว แต่ถ้า Pattern นั้นมีโครงสร้างที่เหมือนกันแล้ว โปรแกรมจะสามารถแสดงประเภทของ Design Pattern ออกมาได้ แต่ถ้าในกรณีที่ Pattern ที่นำมาตรวจสอบนั้นมีมากกว่า 1 Pattern โปรแกรมจะไม่สามารถตรวจสอบได้ โครงสร้าง ของ Pattern ที่นำมาตรวจสอบ คือ

Contain Invoker C

Inherit ConreteCommand C

Use Client Receiver

Creates Client ConreteCommand

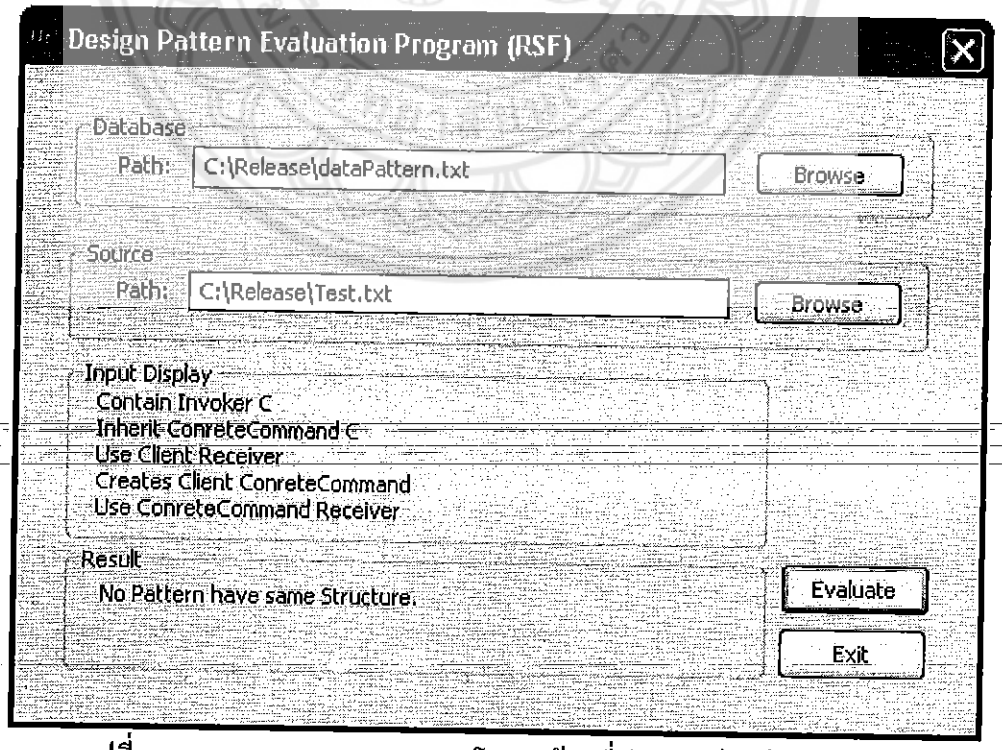
Use ConreteCommand Receiver

Inherit Composite Component

Inherit Leaf Component

Contain Composite Component

ซึ่ง RSF ข้างต้นนี้ เป็น RSF ของ Composite Pattern และ Command Pattern รวมอยู่ด้วยกัน ดังนั้น เมื่อนำไปตรวจสอบกับ Design Pattern ในฐานข้อมูล จะไม่พบ Design Pattern ที่ตรงกับ Pattern ที่นำมาตรวจสอบเลย เนื่องจากโปรแกรมตรวจสอบ Design Pattern จาก Source Code สามารถตรวจสอบได้เพียงครั้งละ 1 Pattern เท่านั้น



รูปที่ 4.67 แสดงผลการตรวจสอบ โครงสร้าง ที่ประกอบไปด้วย 2 Pattern

บทที่ 5
สรุปผล

5.1 สรุปผลการทดลอง

5.1.1 ผลของการประเมิน Design Pattern จากภาษา RSF

โปรแกรมสามารถประเมิน Design Pattern โดยการตรวจสอบโครงสร้างของ Pattern ว่ามีโครงสร้างเหมือนกันหรือไม่ ซึ่งสามารถตรวจสอบ Pattern ที่เขียนด้วยภาษา RSF ที่มีลำดับการเขียนแตกต่างกัน รวมทั้งชื่อของส่วนประกอบที่แตกต่างกันอีกด้วย โดยจากการทดลอง Design Pattern ที่โปรแกรมสามารถตรวจสอบได้มีดังนี้

ตารางที่ 5.1 แสดงรายการ Design Pattern ที่สามารถตรวจสอบได้แยกตามประเภท

Behavioral Patterns	**Creational Patterns**	**Structural Patterns**
Command Pattern	Abstract-factory Pattern	Adapter Pattern
Interpreter Pattern	Builder Pattern	Bridge Pattern
Iterator Pattern	Factory-method Pattern	Composite Pattern type 1
Mediator Pattern	Prototype Pattern	Composite Pattern type 2
Memento Pattern		Composite Pattern type 3
Observer Pattern		Composite Pattern type 4
State Pattern		Decorator Pattern
Strategy Pattern		Flyweight Pattern
Template-method Pattern		Proxy Pattern
Visitor Pattern		

ประกอบด้วย Behavioral Patterns จำนวน 10 ชนิด, Creational Patterns จำนวน 4 ชนิด และ Structural Patterns จำนวน 9 ชนิด ซึ่งรวมทั้งหมดแล้วโปรแกรมสามารถที่จะตรวจสอบ Design Pattern ได้ทั้งหมด 23 ชนิด

จาก Design Pattern ชนิดต่างๆ ที่สามารถตรวจสอบได้นั้น เรายังสามารถที่จะเพิ่มประสิทธิภาพในการตรวจสอบของโปรแกรมให้สามารถที่จะตรวจสอบ Design Pattern ชนิดต่างๆ ได้เพิ่มขึ้นอีก โดยการเพิ่มข้อมูลของ Design Pattern ต่างๆ ที่ต้องการให้โปรแกรมสามารถตรวจสอบได้ลงในฐานข้อมูล

5.1.2 ผลของการตรวจสอบโครงสร้างโดยการใช้ Genetic Algorithm

ในกรณีที่การเขียนบรรยายโครงสร้างของ Pattern ที่นำมาตรวจสอบโดย RSF มีลำดับการเขียนไม่ตรงกับ RSF ที่แปลงมาจากฐานข้อมูล Genetic Algorithm สามารถช่วยในการตรวจสอบได้ว่าโครงสร้างของข้อมูลนั้นมีความเหมือนกันหรือไม่ โดยรอบของการประมวลผล (Iteration) จะแปรผันตรงกับจำนวนส่วนประกอบของโครงสร้างและความซับซ้อนของความสัมพันธ์

ในการทดลองโปรแกรมสามารถที่จะทำการตรวจสอบเปรียบเทียบโครงสร้างของ Design Pattern โดยใช้ Genetic Algorithm ได้กับทุก Design Pattern ที่จัดเก็บอยู่ในฐานข้อมูล

5.1.3 ผลของการแสดงโครงสร้างของ Pattern ด้วย Adjacency Matrix

การแสดงโครงสร้างโดย Adjacency Matrix สามารถที่จะแสดงความสัมพันธ์ของโครงสร้างต่างๆ ได้ด้วยค่าตัวเลขที่มีลักษณะเฉพาะตัวคือไม่มีตัวเลขแสดงความสัมพันธ์อื่นๆ รวมกันแล้วได้ตัวเลขนั้น ถึงแม้ว่าส่วนประกอบหนึ่งๆ อาจจะมีหลายความสัมพันธ์กับส่วนประกอบอื่นๆ เราก็สามารถแยกความสัมพันธ์ออกได้จากข้อมูลที่เป็น Adjacency Matrix

5.2 ปัญหาและอุปสรรค

1. โปรแกรมไม่สามารถตรวจสอบ Pattern ที่มีมากกว่า 1 Design Pattern ใน RSF ที่เป็น Input ได้ จากข้อจำกัดของวิธีการตรวจสอบโดยใช้หลักการแสดงโครงสร้างโดยใช้กราฟและการตรวจสอบโครงสร้างที่เทียบกับตัว Design Pattern ทีละอัน

2. โปรแกรมไม่สามารถตรวจสอบ Design Pattern บางอันที่มีส่วนประกอบเพียงอันเดียวและไม่มีส่วนแสดงแสดงความสัมพันธ์ได้ เช่น Singleton Pattern

3. โปรแกรมสามารถตรวจสอบได้เพียงแต่โครงสร้างภายนอก ทำให้ในการตรวจสอบ Input ที่เป็น RSF อาจจะทำให้ผลลัพธ์ออกว่ามีลักษณะโครงสร้างเหมือน Design Pattern มากกว่า 1 แบบ เช่น Composite Pattern แบบที่ 1 กับ Interpreter Pattern ที่มีโครงสร้างภายนอกเหมือนกัน แต่ส่วนประกอบภายในแตกต่างกัน

4. โปรแกรมไม่สามารถตรวจสอบความคล้ายกันของโครงสร้าง Design Pattern ได้

5.3 ข้อเสนอแนะ

1. ในการตรวจสอบความคล้ายกันของโครงสร้าง Design Pattern สามารถทำได้โดยการปรับค่าน้ำหนักตัวเลขในการแสดงความสัมพันธ์ให้มีค่าที่เหมาะสม ซึ่งเมื่อเราใช้ Genetic Algorithm ในขบวนการตรวจสอบ เราสามารถที่จะตรวจสอบความคล้ายของโครงสร้างจากค่า Fitness ได้

2. ในการตรวจสอบ Design Pattern ที่มีส่วนประกอบเพียงอย่างเดียวและไม่มีส่วนแสดงแสดงความสัมพันธ์จะต้องมีการเพิ่มคำอธิบายโครงสร้างของ Pattern ว่าประกอบไปด้วยอะไรบ้างใน Pattern Language ของ Input Pattern

5.4 สรุป

เราได้ทำการศึกษาทฤษฎีและหลักการเกี่ยวกับโครงสร้างของ Design Pattern ประเภทต่างๆ ว่ามีโครงสร้างในลักษณะใด โดยได้เก็บรวบรวม Design Pattern เหล่านี้อยู่ใน format ของ RML และจัดเก็บไว้ในฐานข้อมูล โดยเราสามารถที่จะตรวจสอบ Input Pattern ว่าเป็น Design Pattern ชนิดใดได้โดยการนำโครงสร้างของทั้งคู่มาเปรียบเทียบกัน

หลังจากนั้น เราได้ทำการพัฒนาโปรแกรมเพื่อใช้ในการตรวจสอบว่า Input Pattern ที่อยู่ในformat ของ RSF ว่าเป็น Design Pattern ชนิดใด โดยนำเอาโครงสร้างของInput Patternไปเปรียบเทียบกับโครงสร้างของ Design Pattern ในฐานข้อมูล โดยใช้หลักการของ Graph Isomorphism, Adjacency Matrix และ Genetic Algorithm ในขบวนการเปรียบเทียบ จากการทดลองโปรแกรมที่พัฒนาขึ้นสามารถที่จะตรวจสอบเปรียบเทียบโครงสร้างของ Design Pattern ได้ทั้งหมด 23 แบบ

บรรณานุกรม

[1] กฤติกา เผื่อนงูเหลือม. "Graph." [Slide]. กรุงเทพฯ : สถาบันเทคโนโลยีพระจอมเกล้าพระนครเหนือ. 2546

[2] ไตรภพ ยศราช. "Reach abilities in graphs." [Online].Available : www.cpe.ku.ac.th/~jtf/204512-47/note4b-org.doc. 2547

[3] นิรุธ อำนวยศิลป์. **คู่มือการเขียนโปรแกรม Microsoft Visual C++.** กรุงเทพฯ : ซัคเซส มีเดีย. 2544.

[4] ปริศนา แช่มสุขี. "เอกสารสัมมนาทางคอมพิวเตอร์ Genetic Algorithm GA." [PDF]. ชลบุรี. มหาวิทยาลัยบูรพา. 2547

[5] ไม่ปรากฏชื่อผู้แต่ง. "ทฤษฎีกราฟ." [Online]. Available : http://www.teched.rmut.ac.th/download/downloads/undirectedgraph.pdf. 2545

ภาคผนวก ก

รูปแบบของ Design Pattern ที่เป็น ฐานข้อมูล

Command Pattern

ComPat (Command, Invoker, ConreteCommand, Receiver, Client) :=

Contain (Invoker, Command)

& Inherit (ConreteCommand, Command)

& Use (ConreteCommand, Receiver)

& Use (Client, Receiver)

& Creates (Client, ConreteCommand);

Interpreter Pattern

ComPat (AbstractExpression, TerminalExpression, NonterminalExpression) :=

Inherit (NonterminalExpression, AbstractExpression)

& Contain (NonterminalExpression, AbstractExpression)

& Inherit (TerminalExpression, AbstractExpression)

& ! Contain (TerminalExpression, AbstractExpression);

Iterator Pattern

ComPat (Iterator, Aggregate, Concretelterator, ConcreteAggregate) :=

Inherit (Concretelterator, Iterator)

& Use (Concretelterator, ConcreteAggregate)

& Creates (ConcreteAggregate, Concretelterator)

& Inherit (Aggregate, ConcreteAggregate);

Mediator Pattern

ComPat (Mediator, Colleague, ConcreteMediator, ConcreteColleague) :=

Inherit (ConcreteMediator, Mediator)

& Use (Colleague, Mediator)

& Inherit (ConcreteColleague, Colleague)

& Use (ConcreteMediator, ConcreteColleague);

Memento Pattern

ComPat (Memento, Caretaker) :=

Contain (Caretaker, Memento);

Observer Pattern

ComPat (Observer, Subject, ConcreteObserver, ConcreteSubject) :=

Inherit (ConcreteObserver, Observer)

& Use (Subject, Observer)

& Inherit (ConcreteSubject, Subject)

& Use (ConcreteObserver, ConcreteSubject);

State Pattern

ComPat (State, ConcreteState, Context) :=

Inherit (ConcreteState, State)

& Contain (Context, State);

Strategy Pattern

ComPat (Strategy, ConcreteStrategy, Context) :=

Inherit (ConcreteStrategy, Strategy)

& Contain (Context, Strategy);

Template-method Pattern

ComPat (AbstractClass, ConcreteClass) :=

Inherit (ConcreteClass, AbstractClass);

Visitor Pattern

ComPat (Visitor, ConcreteVisitor, Element, ConcreteElement, ObjectStructure) :=

Inherit (ConcreteVisitor, Visitor)

& Inherit (ConcreteElement, Element)

& Use (ObjectStructure, Element);

Abstract-factory Pattern

ComPat (Client, AbstractFactory, AbstractProduct, ConcreteFactory, Product) :=

Inherit (Product , AbstractProduct)

& Inherit (ConcreteFactory, AbstractFactory)

& Use (Client, AbstractFactory)

& Use (Client, AbstractProduct)

& Product (ConcreteFactory, Product);

Builder Pattern

ComPat (Builder, Director, ConcreteBuilder, Product) :=

Contain (Director, Builder)

& Inherit (ConcreteBuilder, Builder)

& Builds (ConcreteBuilder, Product);

Factory-method Pattern

ComPat (Product, ConcreteProduct, Creator, ConcreteCreator) :=

Inherit (ConcreteProduct, Product)

& Creates (ConcreteCreator, ConcreteProduct)

& Inherit (ConcreteCreator, Creator);

Prototype Pattern

ComPat (Prototype, ConcretePrototype, Client) :=

Inherit (ConcretePrototype, Prototype)

& Use (Client, Prototype);

Adapter Pattern

ComPat (Target, Adapter, Adaptee, Client) :=

Inherit (Adapter, Target)

& Use (Adapter, Adaptee)

& Use (Client, Target);

Bridge Pattern

ComPat (Abstraction, Implementor, RefineAbstraction, ConcreteImplementor) :=

Inherit (RefineAbstraction, Abstraction)

& Inherit (ConcreteImplementor, Implementor)

& Contain (Abstraction, Implementor);

Composite1 Pattern

ComPat (Component, Leaf, Composite) :=

Inherit (Composite, Component)

& Contain (Composite, Component)

& Inherit (Leaf, Component)

& ! Contain (Leaf, Component);

Composite2 Pattern

ComPat (Component, Leaf, Composite, Client) :=

Inherit (Composite, Component)

& Contain (Composite, Component)

& Inherit (Leaf, Component)

& ! Contain (Leaf, Component)

& Use (Client, Component);

Composite3 Pattern

ComPat (Component, Composite) :=

Contain (Composite, Component)

& Inherit (Composite, Component);

Composite4 Pattern

ComPat (Component) :=

Contain (Component, Component);

Decorator Pattern

ComPat (Component, ConcreteComponent, Decorator, ConcreteDecorator) :=

Inherit (Decorator, Component)

& Contain (Decorator, Component)

& Inherit (ConcreteComponent, Component)

& ! Contain (ConcreteComponent, Component);

Flyweight Pattern

ComPat (FlyweightFactory, Flyweight, ConcreteFlyweight, UnshardConcreteFlyweight, Client):=

Inherit (UnshardConcreteFlyweight, Flyweight)

& Inherit (ConcreteFlyweight, Flyweight)

& Contain (FlyweightFactory, Flyweight)

& Use (Client, UnshardConcreteFlyweight)

& Use (Client, ConcreteFlyweight)

& Use (Client, FlyweightFactory);

Proxy Pattern

ComPat (Subject, RealSubject, Proxy) :=

Inherit (RealSubject, Subject)

& Use (Proxy, RealSubject);

ประวัติผู้เขียนโครงงาน

ชื่อ นายธีรภัทร์ ยังดำรง

ภูมิลำเนา 90/2 ถนนศรีธรรมไตรปิฎก อำเภอเมือง จังหวัดพิษณุโลก

ประวัติการศึกษา

- จบมัธยมศึกษาจากโรงเรียนพิษณุโลกพิทยาคม จังหวัดพิษณุโลก
- ปัจจุบันกำลังศึกษาอยู่ระดับปริญญาตรีชั้นปีที่ 4
  สาขาวิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์
  มหาวิทยาลัยนเรศวร

E-mail: float_teerapat@hotmail.com

ชื่อ นางสาววณิชยา ราชบัณฑิต

ภูมิลำเนา 285/3 หมู่ 12 ตำบลหนองญาติ อำเภอเมือง
จังหวัดนครพนม

ประวัติการศึกษา

- จบมัธยมศึกษาจากโรงเรียนสกลราชวิทยานุกูล จังหวัดสกลนคร
- ปัจจุบันกำลังศึกษาอยู่ระดับปริญญาตรีชั้นปีที่ 4
  สาขาวิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์
  มหาวิทยาลัยนเรศวร

E-mail: porjayy_a@hotmail.com