การพัฒนาเครื่องมือช่วยออกแบบฐานข้อมูลเชิงวัตถุแบบกระจาย

The Development of a Distributed Object-Oriented Database Design Tool

โดย

นายมานพ เรืองบุตร

รหัส 43067116

อาจารย์ที่ปรึกษา

รศ. ดร. ศุภมิตร จิตตะยโศธร

รายงานนี้เป็นส่วนหนึ่งของโครงการพัฒนาระบบงาน

หลักสูตรวิทยาศาสตรมหาบัณฑิต สาขาวิชาเทคโนโลยีสารสนเทศ

ภาคเรียนที่ 2 ปีการศึกษา 2543

คณะเทคโนโลยีสารสนเทศ

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ชื่อหัวข้อ	การพัฒนาเครื่องมือช่วยออกแบบฐานข้อมูลเชิงวัตถุแบบกระจาย
นักศึกษา	นายมานพ เรืองบุตร
อาจารย์ที่ปรึกษา	รศ.ดร. ศุภมิตร จิตตะยโศธร
ระดับการศึกษา	วิทยาศาสตรมหาบัณฑิต สาขาวิชาเทคโนโลยีสารสนเทศ
แขนงวิชา	วิทยาการสารสนเทศ
ปีการศึกษา	2543

บทคัดย่อ

ปัญหาในการออกแบบฐานข้อมูลเชิงวัตถุแบบกระจาย ก็มีลักษณะที่คล้ายกับปัญหาในการออกแบบฐานข้อมูลเชิงสัมพันธ์แบบกระจาย เพียงแต่เพิ่มรูปแบบการจัดเก็บ object เป็น class และโครงสร้างแบบลำดับชั้นที่ซับซ้อน หากมองแบบง่าย ๆ ระบบฐานข้อมูลเชิงสัมพันธ์ก็คือกรณีพิเศษของระบบฐานข้อมูลเชิงวัตถุที่ปราศจากระบบชั้นของ class และ Attributes ที่ซับซ้อน ดังนั้นอัลกอริธึมในการออกแบบที่พัฒนาขึ้นมาใช้กับระบบฐานข้อมูลเชิงสัมพันธ์และฐานข้อมูลเชิงสัมพันธ์แบบกระจายจึงสามารถประยุกต์ใช้กับระบบฐานข้อมูลเชิงวัตถุแบบกระจายได้ เช่นเดียวกับที่นำเสนอในงานศึกษานี้

รายงานนี้จะอธิบายถึงวิธีการออกแบบ และท้ายสุดสำหรับโครงการพัฒนาระบบงาน จะพัฒนาเป็นโปรแกรมสำเร็จรูปช่วยออกแบบฐานข้อมูลเชิงวัตถุแบบกระจาย โดยสร้างเป็นแบบร่างของฐานข้อมูล ในอนาคตสามารถพัฒนาฐานข้อมูลโดยสร้างจากแบบร่างที่ได้เพื่อเชื่อมต่อเข้ากับ ระบบฐานข้อมูลเชิงวัตถุแบบกระจายจริง

Title	The Development of a Distributed Object-Oriented Database Design Tool
Student	Mr. Manop Ruengbut
Advisor	Dr. Suppamitr Jittayasothorn
Level of Study	Master of Science in Information Technology
Major	Information Science
Academic Year	2000

ABSTRACT

Distributed object-oriented database (DOODB) design inherits the design problems from the relational distributed database (DDB) and presents additional object class and their possible complicated hierarchy structure. A relational database can be simply seen as a special case of an object-oriented database (OODB) without a class hierarchy and complex attributes. Then, many of the design algorithms developed for relational database can be generalized and applied to object-oriented database as presented in this studying.

This seminar study address issues related to design of DOODB. At the end, for a System Development Project, will generate a database schema which in the future can be use as a blueprint of the database management system, so can connects to a real DOODB system.

กิตติกรรมประกาศ

งานวิจัยนี้สามารถสำเร็จได้ เนื่องจากได้รับการสนับสนุนและการช่วยเหลือจากบุคคลหลายฝ่าย ขอกราบพระคุณสำหรับคำแนะนำทางวิชาการจาก รศ. ดร. ศุภมิตร จิตตะยโศธร คำแนะนำสำหรับขั้นตอนต่าง ๆ ของคณะจากเจ้าหน้าที่คณะเทคโนโลยีสารสนเทศทุกท่าน

ในโอกาสนี้ขอกราบขอบพระคุณมารดา ที่ได้ดูแลกระผมเป็นอย่างดีเยี่ยมทำให้สามารถใช้เวลาทั้งหมดเพื่อทำงานวิจัยนี้จนเสร็จลุล่วงไปได้ และขอบคุณในกำลังใจ ความเข้าใจและความอดทน จากคุณจารุวรรณและเด็กชายศศิน พงษ์เกษตรการ ภรรยาและลูก ตลอดเวลาส่วนใหญ่เพื่อการศึกษานี้ รวมทั้งกำลังใจจากเพื่อนร่วมรุ่นทุกคน โดยเฉพาะจากคุณณรรฐพงษ์ พงษ์พานิช ที่ได้ช่วยกันฟันฝ่าอุปสรรคต่าง ๆ จนสำเร็จ

นายมานพ เรืองบุตร

สารบัญ

หน้า

สารบัญตาราง

หน้า

ตารางที่

สารบัญรูปภาพ

บทที่ 1
บทนำ

ปัจจุบันมีความพยายามที่จะประยุกต์ใช้ฐานข้อมูลเชิงสัมพันธ์(Relational Database: RDB) ในการเก็บข้อมูลที่มีความซับซ้อน ตัวอย่างเช่น การเก็บข้อมูลจากระบบคอมพิวเตอร์ช่วยงานออกแบบ ระบบข้อมูลทางภูมิศาสตร์ และระบบฐานข้อมูลที่ใช้เก็บรูปภาพ หรือภาพเคลื่อนไหว แต่ว่าลักษณะของระบบข้อมูลเหล่านี้เป็นชนิดของข้อมูลที่ไม่เหมือนแบบดั้งเดิม ดังนั้นจึงเป็นเรื่องยากถ้าจะใช้งานข้อมูลเหล่านี้กับ RDB

ต่อมาแนวคิดในเชิงวัตถุ(object-oriented approach) ได้รับความนิยมมากขึ้นทั้งในเชิงของการโปรแกรม การออกแบบระบบ และฐานข้อมูลด้วย ความเป็นไปได้ในการใช้วิธีให้ผู้ใช้กำหนด class (user-defined classes) ให้กับ object ที่ซับซ้อน (complex objects) จะง่ายเหมือนกับการโปรแกรมและการออกแบบระบบ จึงเกิดแรงจูงใจให้เกิดการพัฒนาให้เกิดฐานข้อมูลเชิงวัตถุ(Object-Oriented Database: OODB) ขึ้นมา

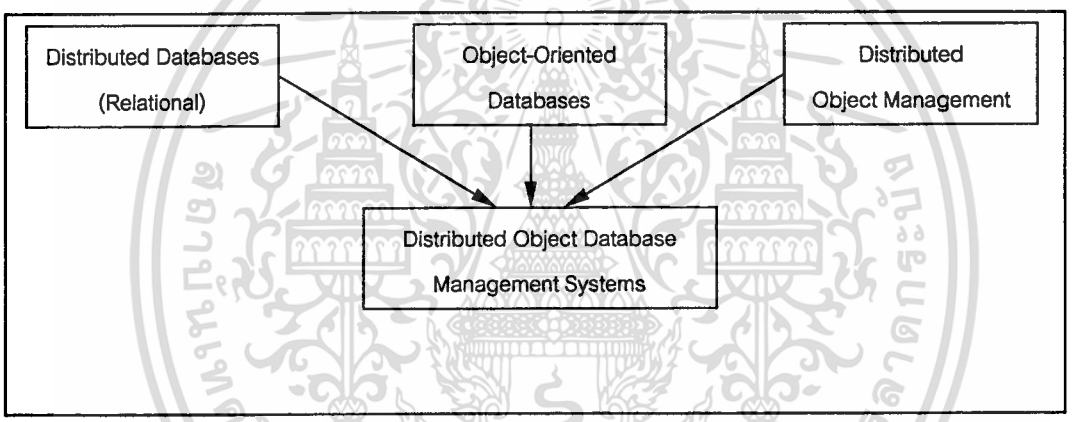
ยิ่งไปกว่านั้นการยอมรับในเรื่องคอมพิวเตอร์เน็ตเวิร์ค (computer network) และผลประโยชน์จากเทคโนโลยีการสื่อสารได้เปลี่ยนทิศทางของการพัฒนาฐานข้อมูลเชิงสัมพันธ์ บทบาทของฐานข้อมูลแบบกระจาย (distributed database: DDB) จึงมีความสำคัญมากขึ้น ในลักษณะการใช้งานฐานข้อมูลที่เป็นธรรมชาติมากขึ้นตามลักษณะโครงสร้างขององค์กรที่มีการกระจาย และยังได้รับประสิทธิภาพในเรื่องของการใช้ข้อมูลร่วมกันอีกด้วย

ประสิทธิผลที่จะได้รับจากการใช้งานในระบบฐานข้อมูลแบบกระจายนั้นขึ้นอยู่กับปัจจัยหลักสองอย่าง คือ การดำเนินการในการให้และการรับข้อมูล และ การถ่ายเทข้อมูล เราสามารถเพิ่มประสิทธิภาพของระบบการจัดการฐานข้อมูล (database management system: DBMS) ได้ ถ้าการออกแบบ DDB ประกอบไปด้วย การแตกกระจายข้อมูล (fragmentation) การทำซ้ำข้อมูล (replication) และการจัดสรรข้อมูลที่แตกกระจาย (fragmentation allocation) อย่างเพียงพอและเหมาะสม ในวิธีแบบเชิงสัมพันธ์ได้แยกการแตกกระจายข้อมูลเป็น 3 ชนิด [1] คือ การแตกกระจายแนวลึก (vertical fragmentation) และการแตกกระจายแนวกว้าง (horizontal fragmentation) และอีกหนึ่งชนิดเป็นการ

ผสมกันระหว่างสองวิธีแรก คือ การแตกกระจายแบบผสม (hybrid fragmentation) ซึ่งสามารถนำมาใช้กับ OODB ได้

ปัจจุบันโปรแกรมสำเร็จรูปในตลาดฐานข้อมูลสำหรับ OODB ยังไม่สามารถตอบสนองกับแนวคิดเชิงวัตถุได้อย่างเต็มรูปแบบ เนื่องจากแนวคิดเชิงวัตถุเองก็ยังไม่มีมาตรฐานที่แน่นอนนัก ทำให้ต้องมีการศึกษาให้ละเอียดยิ่งขึ้น

แม้ว่าจะมีงานวิจัยในเรื่องระบบฐานข้อมูลเชิงวัตถุมากขึ้น แต่ส่วนที่เกี่ยวข้องกับการแตกกระจายข้อมูลนั้นยังมีไม่มาก ซึ่งจะเป็นพื้นฐานสำคัญในการออกแบบฐานข้อมูลชนิดนี้ เพราะสามารถใช้ความรู้จากเรื่องการแตกกระจายข้อมูลไปใช้ในการออกแบบฐานข้อมูลได้



ภาพที่ 1.0.1 สถาปัตยกรรมของฐานข้อมูลแบบ DOODB

1. ระบบสารสนเทศในปัจจุบันแบบ object servers กับ client ซึ่งเชื่อมต่อกันด้วยระบบเน็ตเวิร์คแบบเปิด (Open Networks Systems) การเก็บรักษาข้อมูลจึงต้องอาศัยระบบการจัดการฐานข้อมูลเชิงวัตถุ (Object Oriented Data Manágement Systems : OODBMS) เพื่อที่จะรองรับกับความต้องการนี้ การใช้ระบบฐานข้อมูลเชิงสัมพันธ์ (Relational Database Systems : RDBMS) แบบดั้งเดิม ไม่สามารถรองรับการเก็บข้อมูลที่หลากหลายไปตามลักษณะของระบบงานแบบใหม่นี้ได้ ยิ่งไปกว่านั้นแล้วส่วนใหญ่ของงานประยุกต์เหล่านี้ยังต้องการการจัดการแบบกระจาย (Distributed Management Systems) ด้วย ดังนั้นยุคต่อไปของ OODBMS จึงมีแนวโน้มว่าจะมีลักษณะของ Distributed Systems อยู่ด้วย จึงเกิดเทคโนโลยีทางฐานข้อมูลใหม่ขึ้นมา คือ Distributed Object-Oriented Database Management Systems (DOODBMS) (โปรดพิจารณาภาพที่ 1.1 ประกอบ)

2. หากจะพัฒนาระบบงานที่เชื่อมต่อกับฐานข้อมูลเชิงวัตถุ ให้มีประสิทธิภาพที่ดี ผู้ออกแบบฐานข้อมูลจะต้องให้ความสนใจในเรื่องการออกแบบการกระจาย Object Class, Method ที่จะมีการเรียกใช้งานข้ามไปข้ามมาระหว่างในระบบเน็ตเวิร์ค การออกแบบที่ดีควรต้องคำนึงถึง การแตกกระจายข้อมูล (Fragmentation) การทำซ้ำข้อมูล (Replication) และการจัดสรรข้อมูลที่แตกกระจาย (Fragmentation Allocation) การออกแบบฐานข้อมูลเชิงสัมพันธ์แบบกระจาย ได้แยกการแตกกระจายข้อมูลเป็น 3 ชนิด คือ การแตกกระจายแนวลึก (Vertical Fragmentation) และการแตกกระจายแนวกว้าง (horizontal fragmentation) และอีกหนึ่งชนิดเป็นการผสมกันระหว่างสองวิธีแรก คือ การแตกกระจายแบบผสม (Mixed Fragmentation) ซึ่งสามารถนำมาใช้กับระบบฐานข้อมูลเชิงวัตถุได้ โดยเปรียบเทียบ การแตกกระจายแนวลึก คือ การแยกโครงสร้างของ Class (คือ Attributes และ Methods) ไว้ต่าง site กัน นั่นคือจะมีการเก็บ Object เดียวกันในที่ต่างกันแต่ว่ามีโครงสร้างที่ต่างกัน ส่วนการกระจายแนวนอน คือ การกระจาย Class Instances หรือ Objects ต่าง ๆ ภายใน Class เดียวกัน ไปไว้ยัง site ต่างกัน นั่นคือจะมีกลุ่มของ Objects ใน Class เดียวกัน(โครงสร้างเดียวกัน) ไปอยู่ยังที่ต่างกัน
3. สรุปงานวิจัยที่เกี่ยวข้อง

Characteristics	Savonnet (1998)	Bellatreche (1998)	Ezeife & Barker (1995)	Baiao (1998)	Propose (2001)
Techniques for Horizontal Fragmentation	primary/ derived	primary	primary/ derived	primary/ derived	primary/ derived
Chooses classes indicated to be fragmentation	No	Yes	No	Yes	Yes
Considers relationships between classes	Yes	No	Yes	Yes	Yes
Links between classes are originated by	method calls	-	user queries/	user queries/	method calls
Addressed operations: (N)avigations/ (S)et operations	N, S	S	N, S	N, S	N, S
Takes operation frequencies into account	Yes	No	Yes	Yes	
Develops an algorithm	No	Yes	Yes	Yes	Yes

ตารางที่ 1.1 สรุปงานวิจัยที่เกี่ยวข้อง

4. ขั้นตอนที่จะทำให้เกิด Mixed Fragments ที่ในโครงการพัฒนาเครื่องมือช่วยออกแบบ DOODB ประกอบด้วยส่วนต่าง ๆ ดังนี้ (โปรดพิจารณารูปที่ 4.1 ในบทที่ 4 ประกอบความเข้าใจรายละเอียดจะกล่าวอีกครั้งในบทที่ 4 เช่นกัน)

   A) **Analysis Phase: Object-Oriented Database (OODB) design**

   B) **Fragmentation Phase: Distributed Database (DDB) design**

   C) **Method and Class Allocation (MCA) Phase: Distributed Object-Oriented Database (DOODB) design**

บทที่ 2
ทฤษฎีและหลักการที่เกี่ยวข้อง

ในส่วนนี้ต้องการอธิบายถึงทฤษฎีและหลักการที่เกี่ยวข้องกับงานวิจัยนี้ โดยจะกล่าวถึง 3 ประเด็นหลัก คือ

- **พื้นฐานแนวคิดในเชิงวัตถุ (Fundamental Concepts of Object-Oriented Models)** เพื่อเป็นการทำความเข้าใจกับแนวคิดในเชิงวัตถุ เนื่องจากในปัจจุบันยังไม่มีทฤษฎีที่มารองรับได้เหมือนกับในฐานข้อมูลเชิงสัมพันธ์ จึงต้องมีการอธิบายศัพท์ที่เกี่ยวข้องเอาไว้ก่อน ซึ่งเป็นการอธิบายโดยยึดหลักการโปรแกรมเอาไว้ ยังไม่เข้าไปถึงความหมายที่เกี่ยวกับระบบฐานข้อมูล
- **การออกแบบในฐานข้อมูลเชิงวัตถุ** จะแบ่งเป็น 2 หัวข้อ
  - **ความแตกต่างระหว่างการออกแบบฐานข้อมูลเชิงสัมพันธ์กับฐานข้อมูลเชิงวัตถุ** เพื่อแยกความแตกต่างระหว่างการออกแบบในระบบฐานข้อมูลสองระบบนี้ได้ดียิ่งขึ้น โดยนำเอาความหมายของศัพท์ต่าง ๆ ในเชิงการโปรแกรมมาใช้ในระบบฐานข้อมูล และชี้ให้เห็นถึงประโยชย์ของการออกแบบในฐานข้อมูลเชิงวัตถุโดยเปรียบเทียบกับฐานข้อมูลเชิงสัมพันธ์
  - **การออกแบบฐานข้อมูลเชิงวัตถุในปัจจุบัน**
- **Fragmentation ในฐานข้อมูลเชิงวัตถุแบบกระจาย** จะอธิบายรูปแบบของฐานข้อมูลเชิงวัตถุแบบกระจาย (Distributed Object-Oriented Database : DOODB) โดยจะกล่าวถึงการทำ Fragmentation ในสองแบบหลัก คือ vertical fragmentation และ horizontal fragmentation

2.1. พื้นฐานแนวคิดในเชิงวัตถุ (Fundamental Concepts of Object-Oriented Models)

การโปรแกรมในเชิงวัตถุแพร่หลายมาก เนื่องจากได้เปรียบเรื่องการจัดระเบียบโปรแกรมที่ดีกว่าและพัฒนาเป็นซอฟแวร์ที่น่าเชื่อถือกว่า ต่อมาจึงมีการนำแนวคิดนี้เข้ามาใช้กับระบบฐานข้อมูล เพื่อทำความเข้าใจกับแนวคิดนี้ จึงจะอธิบายถึงความหมายของศัพท์ต่าง ๆ ที่เกี่ยวข้อง

2.1.1. Type

ชนิดของข้อมูลในการโปรแกรมเชิงวัตถุ มีได้หลายแบบ คือ ชนิดของข้อมูลแบบพื้นฐาน(base type) ได้แก่ จำนวนเต็ม(Integer) จำนวนจริง(real numbers) จำนวนตรรกะ(Boolean) ตัวอักษร

(character strings) และยังสามารถสร้างชนิดอื่นได้อีกมากมาย โดยใช้ตัวสร้าง(constructors) เป็นตัวสร้างชนิดข้อมูลใหม่ขึ้นมา เช่น

- **Record structures :** เหมือนกับการกำหนด "structs" ในการเขียนโปรแกรมในภาษา C หรือ C++ กล่าวคือ กำหนดให้ $T_1, T_2, ..., T_n$ เป็นรายการของ type และ $f_1, f_2, ..., f_n$ เป็นรายการของ field names ตัวสร้างหนึ่งตัวสามารถสร้าง record type ที่มีจำนวน n ตัวประกอบ และส่วนประกอบลำดับที่ ith จะมี type คือ T_i และมี field name คือ f_i เรียงกันไป
- **Collection types :** กำหนดได้ว่า type T เพียงอย่างเดียวสามารถนำไปสร้างเป็น type ใหม่ได้ โดยใช้ collection operator (เช่น lists, array และ sets) กับ type T (ซึ่งสมมุติให้เป็น integer) กลายเป็น type ใหม่ คือ "list of integers", "array of integers" หรือ "set of integers"
- **Reference types :** การอ้างถึง type T คือ type ที่ค่าของมันใช้เป็นที่เก็บค่าของ type T (สำหรับในภาษา C, C++ ก็คือ "pointer") การเก็บข้อมูลในระบบฐานข้อมูลนั้นจะเก็บบนดิสก์หลายดิสก์ หรือบางครั้งอาจจะกระจายกันอยู่ตาม hosts ต่าง ๆ การอ้างอิงก็จะเป็นสิ่งจำเป็นมากกว่าการเป็น pointer ตัวอย่างเช่น อาจจะต้องเก็บชื่อของ host, หมายเลขของดิสก์, block ภายในดิสก์นั้น และเก็บตำแหน่งภายใน block ที่ค่าอ้างอิงเก็บอยู่

2.1.2. Classes and Objects

class ประกอบด้วย type เพียงชนิดเดียวและอาจมี function, procedure (เรียกว่า method) หนึ่งหรือมากกว่านั้น ซึ่งสามารถดำเนินการเกี่ยวกับ objects ของ class นั้นได้ objects ของ class หนึ่งอาจจะเป็นค่าของ type นั้นเอง(เรียกว่า *immutable objects*) หรือเป็นตัวแปรซึ่งมีค่าเป็น type เดียวกัน(เรียกว่า *mutable objects*)

ตัวอย่างเช่น ถ้าเรากำหนด class A มีชนิดเป็น "set of integers" จะได้ว่า {2, 4, 9} เป็น immutable object ของ class A ขณะที่ตัวแปร S ก็สามารถถูกกำหนดให้เป็นสมาชิกของ class A ได้ โดยให้ S มีค่าเป็น {2, 4, 9}

2.1.3. Object Identity

ขณะที่ tuples ในฐานข้อมูลเชิงสัมพันธ์ถูกแยกแยะโดย key ที่กำหนดให้แต่ละ tuples นั้น objects ก็ถูกแยกแยะโดย ID ประจำ objects นั้นเหมือนกัน โดยที่ objects จะมีคุณสมบัติดังนี้

- ทุกละ objects จะถูกกำหนด object identity หรือ OID
- ไม่มี objects ใด ๆ ที่มี OID ซ้ำกัน
- ไม่มี objects ใด ๆ ที่จะมี 2 OID ได้

- OID คือ ค่าที่ใช้อ้างอิง object ได้
- เราคิดว่า OID เป็น pointer ในหน่วยความจำเสมือนที่ชี้ไปยัง object แต่ว่าในระบบฐานข้อมูลแล้ว OID สามารถเป็นสิ่งที่ซับซ้อนกว่า pointer ได้

2.1.4. Methods

method จะเป็นสิ่งที่ดำเนินการกับ class และมีหน้าที่ที่แน่นอน method สำหรับ class C จะมี อย่างน้อยหนึ่ง argument ที่เป็น object ของ class C และ argument อื่นจะเป็นของ class อื่นหรือ class C ก็ได้ ตัวอย่างเช่น การดำเนินการกับ class C ซึ่งมี type เป็น "set of integers" เราอาจมี method ที่คำนวณ power set ของ set ที่กำหนด หรือหา union ระหว่างสอง set ก็ได้

2.1.5. Abstract Data Types

มีหลาย ๆ กรณีที่ classes จะเป็น "abstract data types" หมายความว่า อาจถูก ***encapsulate*** หรือถูกจำกัดขอบเขตในการเข้าถึง object ของ class ดังนั้นจะมีเพียง methods ที่ถูกระบุไว้เท่านั้นที่สามรถแก้ไข objects ของ class ได้โดยตรง การจำกัดแบบนี้เพื่อเป็นการป้องกันว่า objects ของ class นั้นจะไม่ถูกเปลี่ยนแปลงอย่างพละการ แนวคิดนี้เป็นข้อดีในการพัฒนาโปรแกรมสำเร็จรูปที่น่าเชื่อถือได้

2.1.6. Class Hierarchies

เราสามารถประกาศเอาไว้ว่า class C อาจจะเป็น ***subclass*** ของ class D ได้ นั่นคือ class C จะ ***inherits*** คุณสมบัติของ class D ทั้งหมดไว้ รวมทั้ง type ของ class D และ functions ต่าง ๆ ที่ระบุไว้สำหรับ class D แต่ว่า class C ก็สามารถที่จะเพิ่มคุณสมบัติของตัวเองได้ ตัวอย่างเช่น อาจจะระบุ method ใหม่ให้ใช้สำหรับ objects ใน class C ซึ่งอาจจะเป็นการเพิ่มหรือแทนที่ methods ที่ได้มาจาก class D ก็ได้ ในลักษณะเดียวกันก็อาจจะเป็นการเพิ่ม type จากที่ class D มี โดยเฉพาะถ้า class D เป็น record-structure type เราสามารถจะเพิ่ม fields ใหม่ให้กับ type นี้ได้เลย แต่มีผลใช้ได้เฉพาะกับ objects ใน class C เท่านั้น

2.2. การออกแบบในฐานข้อมูลเชิงวัตถุ

2.2.1. ความแตกต่างระหว่างการออกแบบฐานข้อมูลเชิงสัมพันธ์กับฐานข้อมูลเชิงวัตถุ

ในทางปฏิบัติแล้วเราจะออกแบบ RDB โดยเริ่มจากหา entities(รวมทั้ง entities types) และหาความสัมพันธ์ระหว่างกัน(รวมทั้ง relationship types) ผลลัพธ์ที่ได้จากขั้นตอนแรกนี้ คือ "Entity-Relationship :ER" schema แล้วจะถูกแปลงไปเป็น relational schema อีกทีและสามารถใช้โปรแกรม

สำเร็จรูปทำได้โดยอัตโนมัติ แต่ว่าบางครั้งก็จะได้ relational schema ที่ยังไม่สมบูรณ์ดีและต้อง normalization อีกขั้นหนึ่ง ขั้นตอนสุดท้ายในทางใดทางหนึ่งก็ตามจะต้องได้ชุดของ relational schema ที่สมมูลกับ ER schema ถึงกระนั้นก็ตาม schema ของ RDB ที่ได้ก็ยังไม่ดีพอ เพราะยากที่จะรู้ความหมายของความสัมพันธ์โดยไม่รู้การออกแบบ ER ที่สอดคล้องกัน

2.2.2. การออกแบบฐานข้อมูลเชิงวัตถุในปัจจุบัน

ในแง่ของระบบฐานข้อมูลแล้ว OID ของ objects จะต้องมีคุณสมบัติเพิ่มเติม ดังนี้

- OID จะสามารถใช้เป็นตัวแยกแยะ objects ในระบบฐานข้อมูลได้ด้วย นอกจากเป็นตัวแยกแยะ objects ภายใน class แล้ว
- OID จะเป็นค่าคงที่ตลอดวงจรชีวิตของ object
- ผู้ใช้จะมองไม่เห็น OID
- OID ถูกสร้างและจัดการโดยระบบ

ขั้นตอนต่าง ๆ นี้ไม่ต้องทำตามลำดับ ยกตัวอย่างเช่น การอธิบายพฤติการณ์อาจเป็นผลให้เปลี่ยนโครงสร้างภายในของ class ได้ ดังนั้นระหว่างกระบวนการออกแบบจึงเลี่ยงไม่ได้ที่จะต้องทำขั้นตอนต่าง ๆ ซ้ำไปซ้ำมา เรียกได้ว่าเป็นการออกแบบด้วยวิธีแบบวงจร มากกว่าที่จะเป็นวิธี top-down หรือ bottom-up

2.3. Fragmentation ในฐานข้อมูลเชิงวัตถุแบบกระจาย

ตามวิธีแบบเชิงสัมพันธ์แล้ว fragmentation จะเกี่ยวข้องกับอัลกอริธึมที่ประยุกต์ใช้กับการกระจาย attributes ซึ่งขึ้นอยู่กับ 1) ชนิดของ fragmentation 2) สารสนเทศเกี่ยวกับความถี่ของ transaction 3) การใช้ attributes 4) ชนิดของ attributes ที่แน่นอน เราต้องการสารสนเทศ 4 อย่าง เพื่อใช้ในการออกแบบที่เหมาะสม

1. สารสนเทศของฐานข้อมูลที่รวม global conceptual schema
2. สารสนเทศของโปรแกรมประยุกต์ที่เรียกใช้ข้อมูล จะแบ่งเป็น 2 อย่าง คือ 1) การใช้ที่แน่นอน สำหรับ horizontal fragmentation และ 2) แมทริกซ์ในการเรียกใช้ attributes และความถี่ของ transaction สำหรับ vertical fragmentation
3. สารสนเทศของการติดต่อสื่อสารระหว่างเน็ตเวิร์ค
4. สารสนเทศของระบบคอมพิวเตอร์

สารสนเทศสองข้อแรกใช้สำหรับการ fragmentation ส่วนสองข้อสุดท้ายใช้ในการ allocation

แต่ว่าตามวิธีแบบเชิงวัตถุแล้ว รูปแบบทางความคิดจะมีความซับซ้อนมากขึ้นกว่าในเชิงสัมพันธ์ และเพิ่มการพิจารณาในเรื่อง method โครงสร้างลำดับชั้น(hierarchical structure) และ attributes ที่ซับซ้อน(complex attributes) สำหรับการทำ fragmentation

ตัวอย่างของลักษณะใหม่ที่เพิ่มขึ้นมาสำหรับการ fragmentation ใน DOODB คือ

- หา type ของ class ที่เหมือนกัน : วิธีการทำสามารถทำได้เช่นเดียวกับการทำกับ attributes ในเชิงสัมพันธ์
- การทำ flattening กับ class และ object แล้วหาความเหมือนกันระหว่าง attributes
- การกระจายความถี่การใช้ method ของการเข้าใช้ method โดยการกำหนด transactions แล้วตามด้วยการ fragmentation ของ attributes ที่ method เรียกใช้
- ใช้ความถี่ของการเรียกใช้ attributes เพื่อแบ่ง object ของ class ตามด้วยการกระจาย method อย่างเหมาะสม
- สุดท้ายคือ การจัดโครงสร้างลำดับชั้นของ class

2.4. ลักษณะของฐานข้อมูลเชิงวัตถุแบบกระจาย (DOODB Model)

DOODB คือ การรวมกันของ local object base ที่กระจายกันอยู่ตามที่ตั้งต่าง ๆ แล้วเชื่อมโยงกันด้วยระบบเน็ตเวิร์ค สมมติให้ระบบการจัดการฐานข้อมูล (Database Management System : DBMS) กระจายกันอยู่และมี objects อยู่ทั่ว ๆ ระบบเน็ตเวิร์ค สรุปลักษณะของ DBMS แบบกระจายได้ดังนี้

- ที่ตั้งแต่ละที่จะมี schema ภายในเป็นของตัวเอง เรียกว่า "local internal schema : LIS" และใช้อธิบายถึงการจัดระบบข้อมูลทางกายภาพของที่ตั้งนั้น
- ภาพโดยรวมของข้อมูลทั้งหมด จะถูกอธิบายด้วย "global conceptual schema : GCS" ซึ่งจะอธิบายถึงโครงสร้างทางตรรกะของข้อมูลในทุก ๆ ที่ตั้ง
- เมื่อข้อมูลทั้งหมดถูกแบ่งแยกและทำซ้ำ ณ ที่ตั้งเฉพาะถิ่นแล้ว การจัดระบบของข้อมูลทางตรรกะของแต่ละที่ตั้งจะอธิบายด้วย "local conceptual schema : LCS"
- ผู้ใช้ที่เข้าใช้ฐานข้อมูลจะได้รับการช่วยเหลือโดย "external schemas : Exs" และถูกกำหนดอยู่เหนือกว่า GCS

ขณะที่ข้อมูลใน DOODB จะเป็นชุดของ encapsulated objects ที่ค่าของข้อมูล(attribute values) ถูกรวมกับ methods(procedures, functions) objects กับ attributes แบบธรรมดาและ methods จะเป็นของ class เดียวกัน class เป็นความสัมพันธ์ของ $C = (K, A, M, I)$ ซึ่งมีลักษณะเด่น ดังนี้

2.4.1. Fragmentation Type : มี 3 ชนิด คือ

1. Horizontal Fragmentation ของ class แทนด้วย C_h จะประกอบไปด้วย ID ของ class, attributes และ methods ของ class แต่ว่ามีเพียงบาง instance objects ($I' \subseteq I$) ของ class เท่านั้น

   $C_h = (K, A, M, I')$

2. Vertical Fragmentation ของ class แทนด้วย C^v โดยประกอบด้วย ID ของ class, instance objects ทั้งหมด และมีบาง methods ($M' \subseteq M$) และบาง attributed ($A' \subseteq A$)

   $C^v = (K, A', M', I)$

3. Hybrid Fragmentation ของแต่ละ class แทนด้วย C_h^v ประกอบด้วย ID ของ class และมีบาง instance objects($I' \subseteq I$) บาง methods($M' \subseteq M$) กับบาง attributes($A' \subseteq A$)

   $C_h^v = (K, A', M', I')$

2.4.2. Attribute Structures : **ใน class จะมี attribute ได้สองชนิด คือ**

1. Simple Attributes คือ มีเพียง attribute ชนิดเดียว ไม่รวมกับ classes อื่น ๆ หรือส่วนใดส่วนหนึ่งของ classes อื่น ๆ
2. Complex hierarchy คือ ขอบเขตของ attribute อาจจะเป็นของ classes อื่น กำหนดค่าโดยใช้องค์ประกอบของ class หรือลำดับชั้นของ classes ทั้งหมด

2.4.3. Method Structures : มี 3 ชนิด คือ

1. Simple methods คือ methods ที่ไม่ได้ไปเรียกใช้ methods อื่น ๆ ของ classes อื่น ๆ จะใช้แต่เพียงข้อมูลของ object เฉพาะถิ่นหรือ ข้อมูลที่ถูกส่งผ่านค่าตัวแปรเท่านั้น
2. Contained Simple methods คือ simple methods ของ class ที่มีการเรียก class อื่น ๆ Complex methods คือ methods ที่สามารถเรียกใช้ methods ของ classes อื่นได้ ซึ่งอาจจะได้ object ที่มีชนิดแตกต่างกัน

บทที่ 3
วัตถุประสงค์ของการวิจัย

3.1. วัตถุประสงค์และข้อสมมติที่จำเป็นในงานวิจัย

การวิจัยครั้งนี้เบื้องต้นทำการศึกษาว่า การออกแบบฐานข้อมูลเชิงวัตถุแบบกระจาย มีอุปสรรคอะไรบ้าง โดยมุ่งศึกษาว่าอัลกอริธึมและเทคนิคที่ใช้กับฐานข้อมูลแบบ RDB สามารถประยุกต์ใช้กับฐานข้อมูลแบบ DOODB ได้อย่างไร

สุดท้ายเป้าหมายของงานโครงการพัฒนานี้คือ ผลิตเครื่องมือที่จะช่วยนักออกแบบ ได้ออกแบบฐานข้อมูลภายใต้สิ่งแวดล้อมแบบกระจายและลดต้นทุนโดยรวมของการประมวลผลข้อมูลข้ามเน็ตเวิร์ก การคำนวณการลดต้นทุนและการกำหนดการจัดเก็บข้อมูลอย่างความเหมาะสม จำเป็นต้องอาศัยอัลกอริธึมหลาย ๆ อัลกอริธึม และใช้ตัวแปรที่เกี่ยวข้องหลายตัว

โดยอาศัยอัลกอริธึมในเรื่องของการแตกกระจาย Class และ Method จาก [2] ส่วนในเรื่องของการจัดสรร Class และ Method ได้นำสูตรการคำนวณต้นทุนและวิธีการมาจาก [3] และ [4] โดยที่ปัญหาเรื่องการจัดสรรสามารถคิดได้ว่าเป็นปัญหาเรื่องของการจับคู่ Class ให้กับ Site และการจับคู่ Method ให้กับ Site ประเด็นหลัก ๆ คือ

1. ในส่วนปัญหาของการจัดสรรนี้จะรวมการจัดสรร Method และ Class เข้าด้วยกัน
2. แบบจำลองคำนวณต้นทุนที่ใช้ เกี่ยวข้องกับการขึ้นต่อกันอย่างซับซ้อนระหว่าง Queries, Classes, Methods และ Sites
3. จะนำเอาสูตรคำนวณที่ให้ผลดีที่สุดใน [3] มาใช้ โดยจะไม่อธิบายลงไปที่รายละเอียดว่ามีที่มาที่ไปอย่างไร เพราะประเด็นปัญหาตรงนี้เป็นเรื่องค่อนข้างกว้าง

การวิจัยจะพิจารณาแบบจำลองของ OODB และสถาปัตยกรรมทางเน็ตเวิร์คแบบกระจายแบบง่าย ๆ กล่าวคือ สมมติว่ามีลักษณะที่สนับสนุน

- Object Identify
- Encapsulation
- Class Hierarchy (IsA relationship)
- Class Composition Hierarchy (PartOf relationship, aggregation graph)

- Method สามารถสืบทอดมาจาก superclass(es) และ subclass สารมรถเปลี่ยนแปลงคำสั่งทำงานของ method ที่สืบทอดมาได้ หรือสามารถเพิ่ม method ได้ สมมติให้มี method แค่สองชนิด คือ simple (method ที่ไม่ได้เรียก method อื่น) และ complex (method ที่เรียก method อื่นได้)

3.2. การแบ่งประเภทการประมวลผลของ Method

ข้อสมมติอีกประการสำหรับจำลองแบบทดลองที่ต้องการ คือ การแบ่งประเภทของการประมวลผลของ Method ใน DOODB ในการศึกษานี้จะแบ่งเป็น 4 รูปแบบ คือ

1. Local Method – Local Object หมายถึง ทั้ง method และ object ต่างถูกเก็บไว้ที่ site เดียวกันกับผู้เรียกใช้ การประมวลผลต่าง ๆ จะถูกกระทำที่ local
2. Local Method – Remote Object หมายถึง ผู้เรียกใช้อยู่ที่เดียวกับ method แล้ว method ถูกระบุให้ประมวลผล objects จำนวนหนึ่งยัง site ที่ต่างกัน กรณีนี้ต้องขนย้าย objects ทั้งหมดที่ต้องการไปยัง site ที่ method สถิตอยู่เพื่อจะประมวลผล
3. Remote Method – Local Object หมายถึง การประมวลผลจะเกิดได้ โดยต้องย้าย method ไปยัง site ที่ objects สถิตอยู่ซึ่งก็เป็น site เดียวกับผู้เรียกใช้อยู่ หลังจากนั้นจึงประมวลผล
4. Remote Method – Remote Object หมายถึง จะต้องย้ายทั้ง method และ objects ไปยัง site ที่ทำการเรียกใช้ ซึ่งแตกต่างจาก sites ของ method และ objects

ดังนั้นเพื่อให้บรรลุวัตถุประสงค์ของงานวิจัย จึงควรต้องคำนึงถึงปัจจัยในส่วนนี้ด้วย เพื่อลดค่าใช้จ่ายในการขนย้าย Method หรือ Objects ไปมาระหว่างเน็ตเวิร์คให้มากที่สุด

บทที่ 4

กรรมวิธีการวิจัยและผลการทดลอง

4.1. รูปแบบของ DOODB design tools

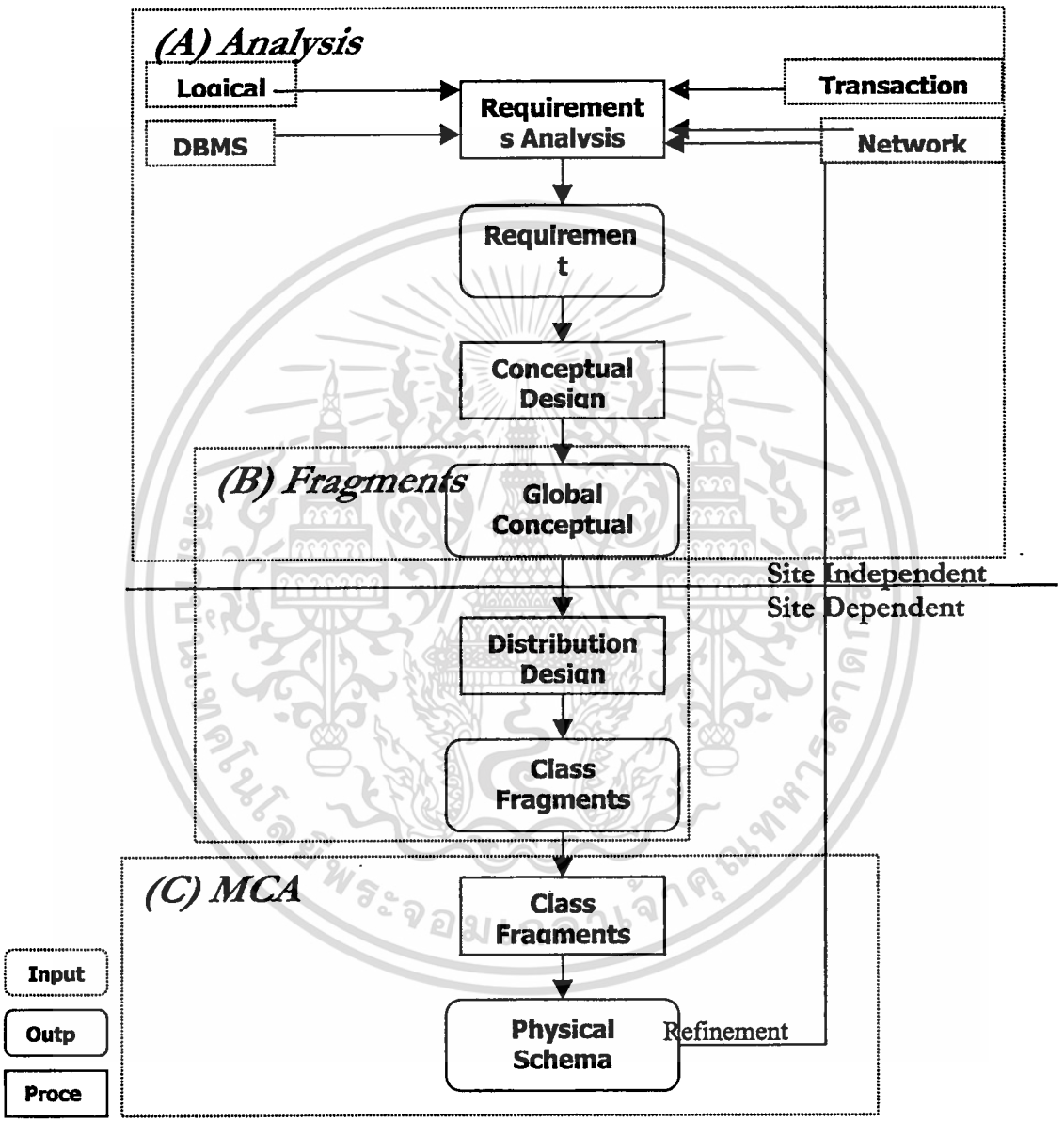
ในการบริหารระบบฐานข้อมูลแบบกระจาย มีปัจจัยหลัก 2 ประการที่นักออกแบบฐานข้อมูลจะต้องพิจารณาเวลาออกแบบฐานข้อมูล เพราะจะไปกระทบประสิทธิภาพการทำงานของระบบฐานข้อมูล อันจะส่งผลไปถึงระบบงานโดยร่วม (set of transaction) ด้วย ปัจจัยหลักสองประการที่ว่านี้คือ จำนวนการเข้าถึงพื้นที่เก็บข้อมูล (Disk Accesses) และ ต้นทุนในการถ่ายโอนข้อมูลจากเน็ตเวิร์คกกลุ่มหนึ่งไปอีกกลุ่มหนึ่ง หรือระหว่างเครื่องคอมพิวเตอร์ภายในเน็ตเวิร์คกกลุ่มเดียวกัน

จากปัจจัยหลัก 2 ประการ สามารถแยกออกเป็นตัวแปรที่เกี่ยวข้องได้เป็น 4 หมวด

1) Logical Parameters
2) Transaction Parameters
3) Database Systems Parameters
4) Network System Parameters

4.2. ขั้นตอนต่าง ๆ ของ DOODB design tools มีดังนี้

(โปรดพิจารณาภาพที่ 4.1 ประกอบความเข้าใจ)



ภาพที่ 4.1 แสดง Diagram ของขั้นตอนต่าง ๆ ในการออกแบบ DOODB

4.2.1. Analysis Phase: OODB design

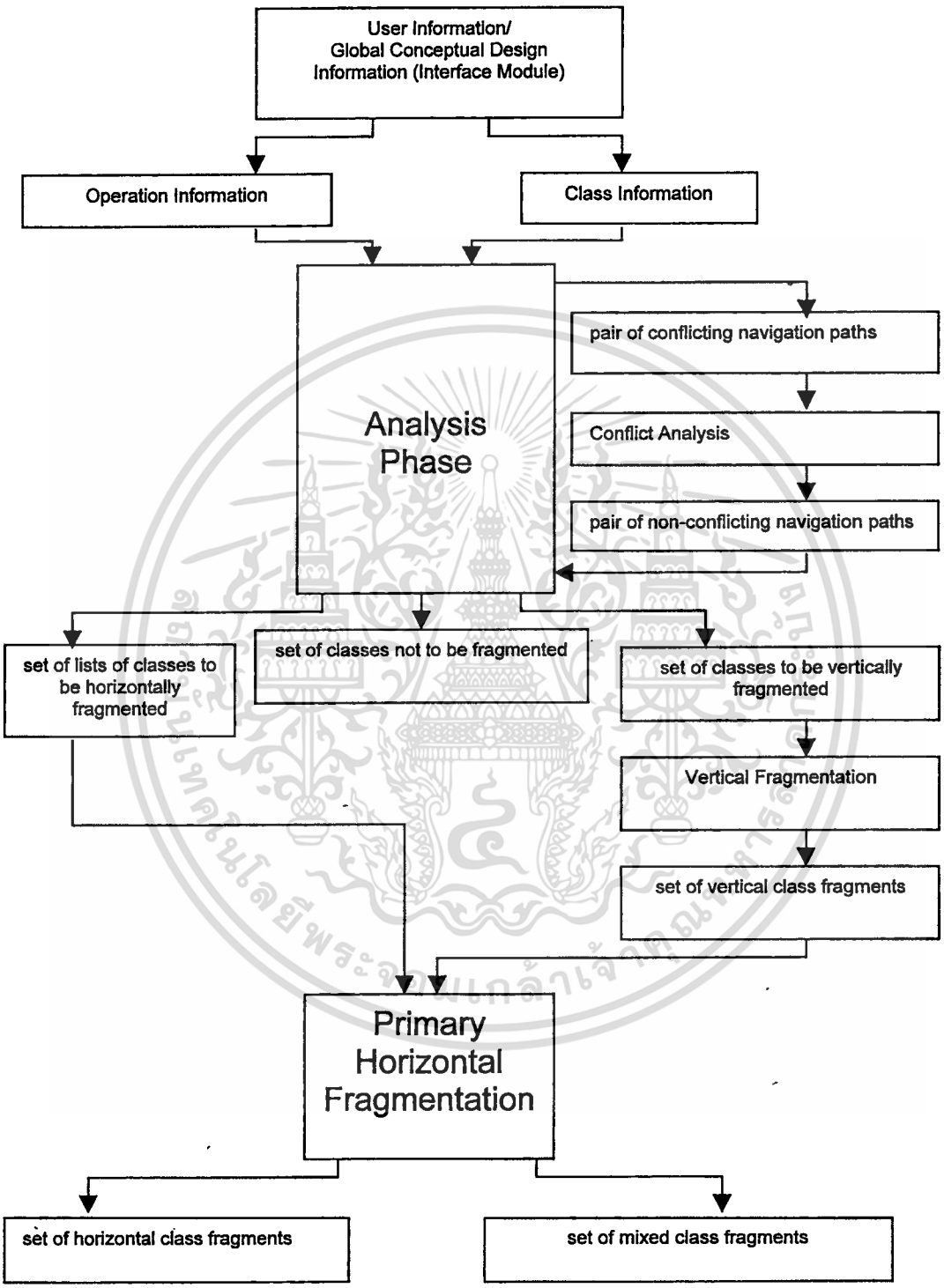
1. ใช้กระบวณการวิเคราะห์และออกแบบเชิงวัตถุ ในการออกแบบ DOODB โดยเริ่มจาก
   - การเก็บ requirements โดยการป้อนค่าตัวแปรที่เกี่ยวข้อง ดังนี้
     1) Logical Parameters
        - Object Database Schema
        - Application Processing Characteristics
     2) Transaction Parameters
        - Site of Origin
        - Frequency of Accessing
     3) Database Systems Parameters
        - การหา Objects และ Classes Structure
        - แจกแจงโครงสร้างภายในของ classes
        - กำหนดรูปแบบการติดต่อแบบคงที่ คือ การสื่อสารของ instance และการสื่อสารของ inheritance
        - กำหนดรูปแบบการติดต่อแบบโครงสร้างรวมทั้งหมด คือ การสื่อสารแบบโดยรวม เช่น ความสัมพันธ์แบบ **has_a**
        - อธิบายการส่งผ่าน message ระหว่าง objects
        - อธิบายพฤติกรรมภายในของ classes
        - Query Processing Strategy
        - Taxonomy of Method Execution
     4) Network System Parameters
        - Network Topology
        - Baud Rates
        - Node Parameters (MIPS, Disk I/O time)

4.2.2. Fragmentation Phase: DDB design

ช่วงขั้นตอนทั้งหมดนี้รวมเรียกว่าเป็นการแตกกระจายทั้ง Class และ Method หรือ Operations แต่จะเป็นการแตกไปเป็น Vertical, Horizontal หรือ Mixed Fragments นั้นก็ขึ้นอยู่กับค่าของตัวแปร Input แล้วก็ลักษณะของ Class กับ Method เอง โดยจะแบ่งช่วงการทำงานนี้เป็น 3 ขั้นตอน คือ

1. ขั้นตอนที่หนึ่ง เริ่มจาก การวิเคราะห์ข้อมูลจากช่วง A) แล้วใช้ Algorithm One ตัดสินว่าจะใช้เทคนิคในการแบ่งแบบไหน ผลลัพธ์ที่ได้ คือ ชุดของ Classes ที่ถูกแบ่งโดยวิธี Vertical และ ชุดของ Classes ที่ถูกแบ่งโดยวิธี Horizontal แล้วจะให้ความสำคัญกับ Operation ที่มีการทำงานถี่ที่สุด นั่นคือเรียงจากมากลงไปหาน้อย จากการเลือกวิธีในการกระจายนี้ หากมีบาง Class ที่ถูกรวมอยู่ในชุดของ Vertical และ Horizontal (โดยทั่วไปมีแค่ Root Class เท่านั้นที่จะเป็นได้) ก็อาจจะถูกแบ่งด้วยวิธีผสม กรณีนี้จะแบ่งด้วย Vertical ก่อน หลังจากนั้นจึงใส่ Horizontal Algorithm เข้าไปอีกที
2. ขั้นตอนที่สอง Vertical Fragmentation คือ การแตกข้อมูลในขั้นตอนที่หนึ่งออกเป็น Vertical Fragments
3. ขั้นตอนที่สาม Horizontal Fragmentation คือ การแตกข้อมูลในขั้นตอนที่หนึ่งออกเป็น Horizontal Fragments และ Mixed Fragments

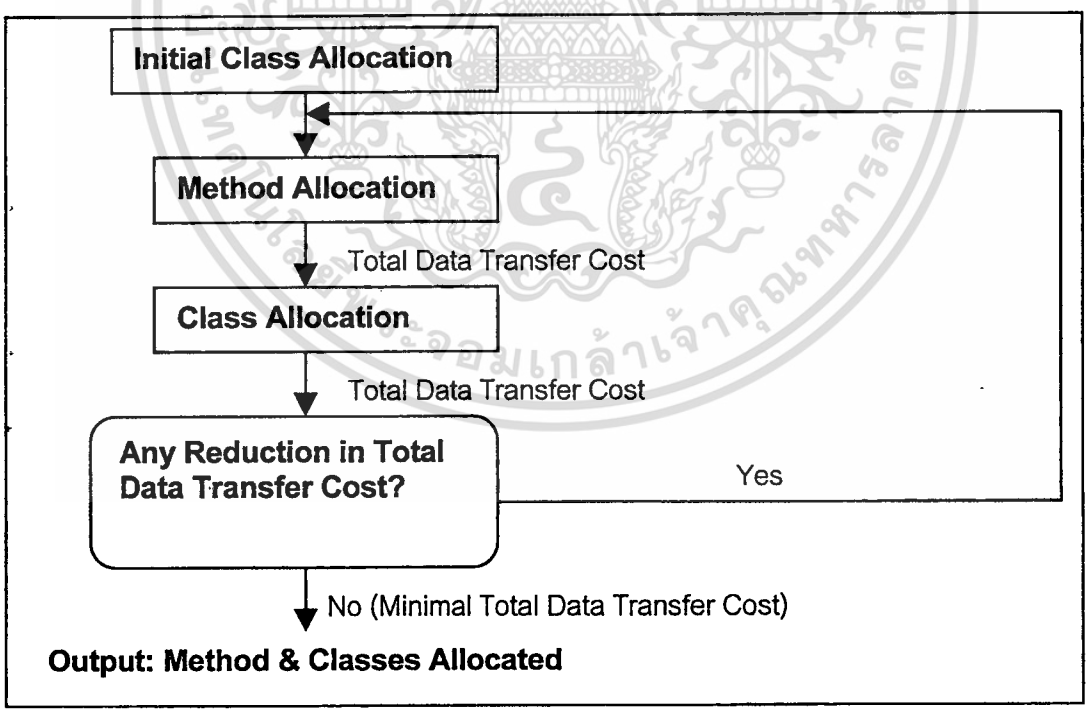
(พิจารณาภาพที่ 4.2 เพื่อประกอบความเข้าใจ)



ภาพที่ 4.2 ภาคขยายของส่วน Fragmentation Phase: DDB design

4.2.3. Method and Class Allocation (MCA) Phase: DOODB design

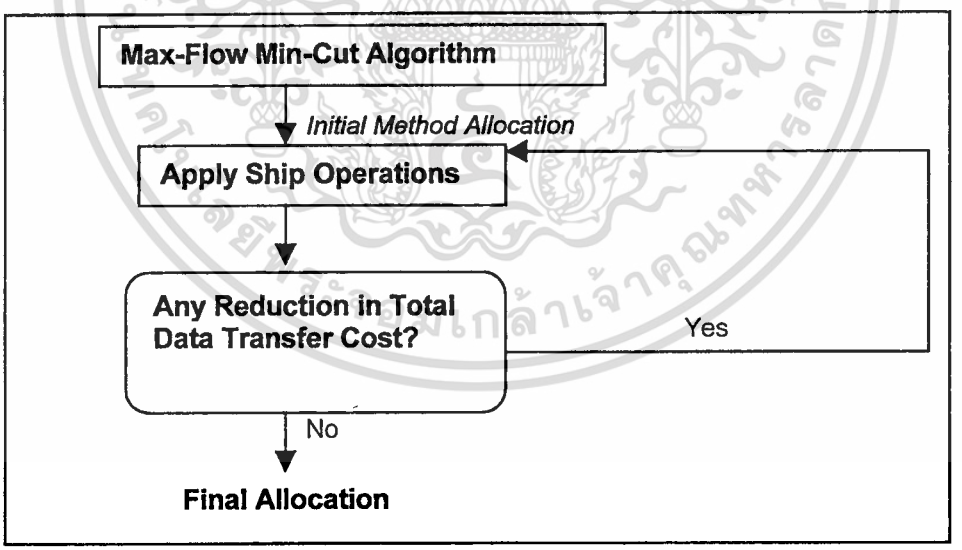
1. สร้างเป็นรูปแสดงความสัมพันธ์ระหว่าง class ต่าง ๆ โดยอาศัยสัญลักษณ์ประกอบ
   - มีสัญลักษณ์แสดงวิธีการออกแบบ OODB (UML)
   - อาจจะมีการทำ OODB Normalization (optional) ถ้ามีการทำ normalization ที่ดีแล้ว จะทำให้ลดความซ้ำซ้อนของข้อมูล
   - Notation แบบ Link Graph จะใช้ข้อมูลจาก Global Conceptual schema โดยที่
     - graph nodes แทน database classes
     - edges แทน ส่วนประกอบของความสัมพันธ์
   - ทำ Hybrid Fragmentation โดยการทำ Horizontal Fragmentation แล้วตามด้วย Vertical Fragmentation โดยการใช้ Link Graph เป็น Notation ในการ design
   - Allocation Hybrid Fragments โดย
     - Disjoint จะดูที่ data
     - Overlapping จะดูที่ method



ภาพที่ 4.3 อัลกอริธึมสำหรับทำ Methods and Class Allocation (MCA)

```
begin
   for j:= 1 to t do /*This step initializes weight between every site and method*/
      for i:=  1 to n do
         w(Sj, mi) = 0; /* w(Sj, mi) is the global weight including all MDGs (1) */
      for j:=  1 to t do /* j is for sites (2) */
         for i:=  1 to n do /* i for methods (3) */
            for h:= 1 to N do /* h for queries (4) */
               for i':=  1 to n do /* i' for methods (5) */
                     if (Ghi' = 1) then (6)
                        if (i = i') and (Vhj ≠ j) then (7)
                           w(Sj, mi) = w(Sj, mi) + wj' (Sj, mi) X fhj + Cost3
                        else
                           w(Sj, mi) = w(Sj, mi) + wj' (Sj, mi) X fh, site (Gh)
else
```

ภาพที่ 4.4 Pseudo Code การแปลง Method Dependency Graph (MDG) เป็น Weighted Graph โดยใช้อัลกอริธึมนี้



ภาพที่ 4.5 สร้าง Final Allocation

```
begin
  for j:= 1 to t do /*This step initializes weight between every site and class*/
    for i:= 1 to s do
      w(Sj, Ck) = 0; /* w(Sj, Ck) is the global weight including all MDGs (1) */
    for j:= 1 to t do /* j is for sites (2) */
      for k:= 1 to s do /* k for classes (3) */
        for h:= 1 to N do /* k for queries (4) */
          for i':= 1 to n do /* i' for methods (5) */
              if (Ghi' = 1) then (6)
                w(Sj, Ck) = w (Sj, Ck) + wj' (Sj, Ck) X fh, site (Gh)
else
```

ภาพที่ 4.6 Pseudo Code การทำ Class Re-Allocation

4.3. ผลการทดลอง

```
class( employee ).
class( department ).
class( project ).
class( location ).
class( dependent ).

cardinality( employee, large ).
cardinality( department, small ).
cardinality( project, large ).
cardinality( location, small ).
cardinality( dependent, large ).

relationship( employee, department, worksIn, '1' ).
relationship( department, employee, holds, 'N' ).
relationship( employee, project, worksOn, 'N' ).
relationship( project, employee, isCarriedOutBy , 'N' ).
relationship( employee, employee, isManagedBy, '1' ).
relationship( employee, dependent, has, '0,N' ).
relationship( dependent, employee, dependsOn, '1' ).
relationship( department, location, isLocatedAt, '1' ).
relationship( location, department, holds, 'N' ).
relationship( project, location, isLocatedAt, '1' ).

operation( employeesWhoWorkInDepartmentAtLocation, 100 ).
operation( projectsControlledByDepartmentAtLocation, 100 ).
operation( mainEmployeeInformation, 85 ).
operation(insuranceInformation, 85 ).

accessedClasses( employeesWhoWorkInDepartmentAtLocation,[location,
                 department, employee] ).
accessedClasses( projectsControlledByDepartmentAtLocation,[location,
                 department, project] ).
accessedClasses( mainEmployeeInformation, [employee] ).
accessedClasses( insuranceInformation, [employee, dependent] ).
```

ภาพที่ 4.7 ตัวอย่างข้อมูล (Input) แบบ DOODBMS พร้อมทั้งเงื่อนไขที่กำหนด โดยใช้ Company Database Schema

********** BEGIN **********

Ch =[]
Cv =[]
Analyzing Operation :projectsControlledByDepartmentAtLocation
Going to Choose Fragmentation Method for classes accessed by
:projectsControlledByDepartmentAtLocation
accessed classes :[location, department, project]
[location, department, project]:will be horizontally fragmented, if possible
removing All Conflicts...
Ch =[[location, department, project]]
Cv =[]
Analyzing Operation :employeesWhoWorkInDepartmentAtLocation
Going to Choose Fragmentation Method for classes accessed by
:employeesWhoWorkInDepartmentAtLocation
accessed classes :[location, department, employee]
[location, department, employee]:will be horizontally fragmented, if possible
removing All Conflicts...
Ch =[[location, department, project], [location, department, employee]]
Cv =[]
Analyzing Operation :insuranceInformation
Going to Choose Fragmentation Method for classes accessed by :insuranceInformation
accessed classes :[employee, dependent]
[employee, dependent]:will be horizontally fragmented, if possible
removing All Conflicts...
Ch =[[location, department, project], [location, department, employee], [employee, dependent]]
Cv =[]
Analyzing Operation :mainEmployeeInformation
Going to Choose Fragmentation Method for classes accessed by :mainEmployeeInformation
accessed classes :[employee|_G882]
employee:will be vertically fragmented
Ch = [[location, department, project], [location, department, employee], [employee, dependent]]
Cv = [employee]

********** COMPLETED **********

ภาพที่ 4.8 ผลการประมวลผล (โดยใช้ PHP)

บทที่ 5
บทสรุป

จากความแตกต่างระหว่างระบบฐานข้อมูลเชิงสัมพันธ์ กับฐานข้อมูลเชิงวัตถุ ในเชิงแนวคิดและทฤษฎีที่ใช้ จึงทำให้ไม่สามารถนำเอาวิธีการออกแบบฐานข้อมูลเชิงสัมพันธ์แบบกระจายมาใช้กับฐานข้อูลเชิงวัตถุได้ตรง ๆ เพราะว่าจะเป็นการออกแบบที่คำนึงถึงเรื่องของการ Query และ เรื่องความสัมพันธ์ระหว่าง Entities ส่วนในเชิงวัตถุจะคำนึงถึงกลไกในการสืบทอดคุณสมบัติ การทำงานของ Method และระดับความซับซ้อนของความสัมพันธ์ระหว่าง Objects

ดังนี้จะเห็นว่า

Relational → Set Oriented

OO → Pointer Oriented และยังอาจมีเรื่องของรูปแบบการเข้าถึง Object คือ Set Operation (เช่น search) และ Set Navigation (เช่น traversals)

จากที่ได้ศึกษาถึงความสมบูรณ์ของอัลกอริธึมต่าง ๆ ที่เกี่ยวข้องในงานศึกษานี้ เช่น allocation algorithm, fragmentation algorithm ฯลฯ ได้ทดลองทำเป็น prototype ของระบบงานทั้งหมด หลังจากนั้นในโครงการพัฒนาระบบงานที่สมบูรณ์ ก็ได้พัฒนาเป็นเครื่องมือสำเร็จรูปที่สามารถใช้ออกแบบ DOODB ได้จริง แต่เนื่องจากข้อจำกัดด้านเวลาจึงไม่สามารถพัฒนาเครื่องมือตัวนี้ได้อย่างสมบูรณ์ โดยสำเร็จเพียงแค่ ตัว User Interface ที่ผู้ใช้สามารถออกแบบ Class Diagram แล้วบันทึกเป็นรูปแบบของ text file ได้ หลังจากนั้นงานวิจัยยังได้ทดลองนำเอา text file ที่ได้จากการสร้างโดย D2O2 แล้วใช้ PHP script เพื่อทดสอบ Algorithm ในการแบ่งแยก Fragments อย่างคราว ๆ ก็ปรากฏดังผลการทำงานในบทที่ผ่านมา ทั้งนี้ในส่วนของการผนวกเอา Algorithm ในการจัดสรร Fragments ที่ได้ ก็มิได้นำมาใส่ไว้ในโปรแกรมเวอร์ชั่นแรกนี้ แต่เป็นหัวข้อการวิจัยที่น่าสนใจหากว่าจะมีผู้สนใจนำไปพัฒนาต่อให้สมบูรณ์ยิ่งขึ้น ทั้งนี้ได้เพิ่มเติมในส่วนของรายละเอียดของ Algorithm ต่าง ๆ ไว้ในภาคผนวกแล้ว

สำหรับเครื่องมือในการพัฒนาโปรแกรม ได้เลือกใช้ Jbuilder 3.5 ในการสร้างตัว Interface ส่วนผลการทำงานได้ใช้ PHP (อย่างที่ได้กล่าวไปแล้ว)

ในการพัฒนาลำดับต่อไป ผู้วิจัยมีความเห็นว่าควรที่จะมีการนำเอา schema ที่ได้จากการสร้างของ D2O2 ไปสร้างจริง ๆ ในระบบฐานข้อมูลแบบ DOODBMS

บรรณานุกรม

Baiao, Fernanda and Mattoso, Marta. 1998. **A Mixed Fragmentation Algorithm for Distributed Object Oriented Database**. Department of Computer Science, Federal University of Rio de Janeiro, Brazil.

Bellatreche, Ladjel and Karlapalem, Kamalakar. 1998. **Complex Methods and Class Allocation in Distributed OODBSs**. University of Science and Technology Clear Water Bay, Kowloon, Hong Kong.

Date, C.J. 2000. **An Introduction to Database Systems**. Seventh edition, New York: Addison-Wesley Publishing Company.

Ezeife, Christiana I. **Class Fragmentation in a Distributed Object Based System**. Department of Computer Science, University of Manitoba, Canada, 1995.

Ezeife, Christiana I. and Barker, Ken. 1993. **A Distributed Object Based Design Technique**. Department of Computer Science, University of Manitoba, Canada.

Ezeife, Christiana I. and Barker, Ken. 1994. **A Comprehensive Approach to Horizontal Class Fragmentation in a Distributed Object Based System.** Technical Report. Advanced Database System Laboratory, Department of Computer Science, University of Manitoba, Canada, October.

Ezeife, Christiana I. and Barker, Ken. 1994. **Vertical Class Fragmentation in a Distributed Object Based System.** Technical Report. Advanced Database System Laboratory, Department of Computer Science, University of Manitoba, Canada.

Hong, Shuguang. 1991. **A Class Normalization Approach to the Design of Object-Oriented Database.** Department of Computer Information systems, Georgia State University.

Koreichi, A. and Le Cum, B. 1997. **On Data Fragmentation and Allocation in Distributed Object Oriented Database.**

Malinowski, Elzbieta. 1996. **Fragmentation Techniques for Distributed Object-Oriented Database.** Master Thesis : Department of Computer and Information Science and Engineer, University of Florida.

Mitchell, Gail and B. Zdonik, Stanley. 1991. **Object-Oriented Query Optimization: What's the Problem?.** Technical Report No. CS-91-41, Department of Computer Science, Brown University.

Özsu, M.T. and Valduriez ,P. and Dayal, Umeshwar. 1994. **An Introduction to Distributed Object Management.** Morgan-Kaufmann, pages 1-24.

Özsu, M.T. and Valduriez ,P. 1999. **Principles of Distributed Database Systems.** Second Editions: Prentice-Hall, ISBN 0-13-607938-5

Rasmussen, Henderson-Sellers and G.C. Low. 1996. **An Object-Oriented Analysis and Design Notation for Distributed Systems.** JOOP. http://www.sigs.com/publications/docs/joop/9610/j10.sellers_friends.html

Saake, Gunter; Conrad, Stefan; Schmitt, Ingo and Turker Can. 1995. **Object-Oriented Database Design: What is the Difference with Relational Database Design?.** Institut för Technische Informationssysteme, Otto-von-Guericke-Universität Magdeburg, Germany. http://wwwiti.cs.uni-magdeburg.de/iti_db/veroeffentlichungen/95/SaaConSch95.html

Semeczko, George. 1995. **Design Issues in Distributed Databases for Wide Area Networks.** PhD Thesis : Department of Computer Science, University of Queensland.

Semeczko, George. 1996. **A Network Model for Resource Allocation Problems in WANs.** Technical Report #96/5, Faculty of Information Technology, Queensland University of Technology.

Semeczko, George. 1996. **Using A Double Weighted Clustering Technique for Fragment Allocation in Wide Area Networks.** Technical Report, Faculty of Information Technology, Queensland University of Technology.

Semeczko, George.; Y. W. Su, Stanley; Yu Tsae-Feng; Wang, Fang. 1996. **Supporting Distributed Query Processing in a Heterogeneous Environment.** Technical Report, Database System Research and Development Center, University of Florida,Gainesville.

Semeczko, George.; Y. W. Su, Stanley. 1997. **Query Plans - Their Cost and Generation in a Distributed Heterogeneous Environment.** Technical Report, Database System Research and Development Center, University of Florida,Gainesville.

Semeczko, George.; Y. W. Su, Stanley. July, 1996. **Supporting Object Migration in Distributed System.** Technical Report 96-029, Department of Computer and Information Science and Engineering, University of Florida,Gainesville, http://www.cis.ufl.edu/research/tech-reports/tr96-abstracts.html

Ullman, D. Jeffrey and Widom, Jennifer. 1997. **A First Course in Database Systems.** International Edition : Prentice-Hall, ISBN 0-13-887-647-9

ภาคผนวก

I. Algorithm One สำหรับการวิเคราะห์: เพื่อเลือกใช้เทคนิคการ Fragment ที่เหมาะสมที่สุดสำหรับแต่ละ Class

```
function AnalisysPhase (      C : the set of classes in the schema,
                              O : the set of operations)
returns  Ch = set of lists of classes to be horizontally fragmented
         Cv = set of classes to be vertically fragmented
         Cn = set of classes not to be fragmented

begin
   sort the set O in a descending way according to the operation frequency
   for each Oi that is in O do
      if Oi is an "extension operation" then
         if (C(Oi) is in C) and (cardinality of C(Oi) = "large") then
            Cv += C(Oi) ; C -= C(Oi)
      else
         if Oi is a "navigation operation" then
            for each non-root class c that is in C(Oi) do
               if c is in Cv then
                  break the list of classes C(Oi) at class c, forming 2 sublists
                     sub1 and sub2
                  sublistsOfOi += sub1
                  C(Oi) = sub2
               sublistsOfOi += C(Oi)
            for each sublist s that is in sublistsOfOi do
               for each list of classes l that is in Ch do
                  if s and l are two conflicting navigational paths then
                     let Oj be the operation that originated l
                     if freq(Oi) and freq(Oj) are "almost the same" then
                        for each class Ck that is in l and conflicts with s
                           Ch -= list l ; C += all the elements from l
                           call procedure conflictAnalysis( Ck, s, l,freq(Oi), freq(Oj) )
                           Ch += returned lists s, l
                           C -= all the non-root classes from s and l
               Ch += s ; C -= all the elements from s
   Cn = C
   return Ch, Cv, Cn
end
```

II. Algorithm Two สำหรับวิเคราะห์ความขัดแย้ง: เพื่อตัดสินใจเลือก Links ที่สัมพันธ์กันมากที่สุดระหว่าง Class

```
function ConflictAnalysis (     Y: Class, N1 = (Ci, Cj, ..., X, Y, ..., Cn),
                                N2 = (Cp, Cq, ..., Z, Y, ..., Cm),
                                f1: frequency of N1, f2: frequency of N2)
returns N1', N2': final navigation paths
begin
   let d1 be the clustering dependency between X and Y
   let d2 be the clustering dependency between Z and Y
   if d1 = d2 then
      if f1 > f2 then
         N1' = N1; N2' = (Cp, Cq, ..., Z)
      else
         N1' = (Ci, Cj, ..., X); N2' = N2
   else
      select
         case d1 = "non-shared dependent"
            N1' = N1; N2' = (Cp, Cq, ..., Z)
         case d1 = "shared dependent"
            if d2 = "non-shared dependent" then
               N1' = (Ci, Cj, ..., X); N2' = N2
            else
               N1' = N1; N2' = (Cp, Cq, ..., Z)
         case d1 = "independent"
            N1' = (Ci, Cj, ..., X); N2' = N2
   return N1', N2'
end
```

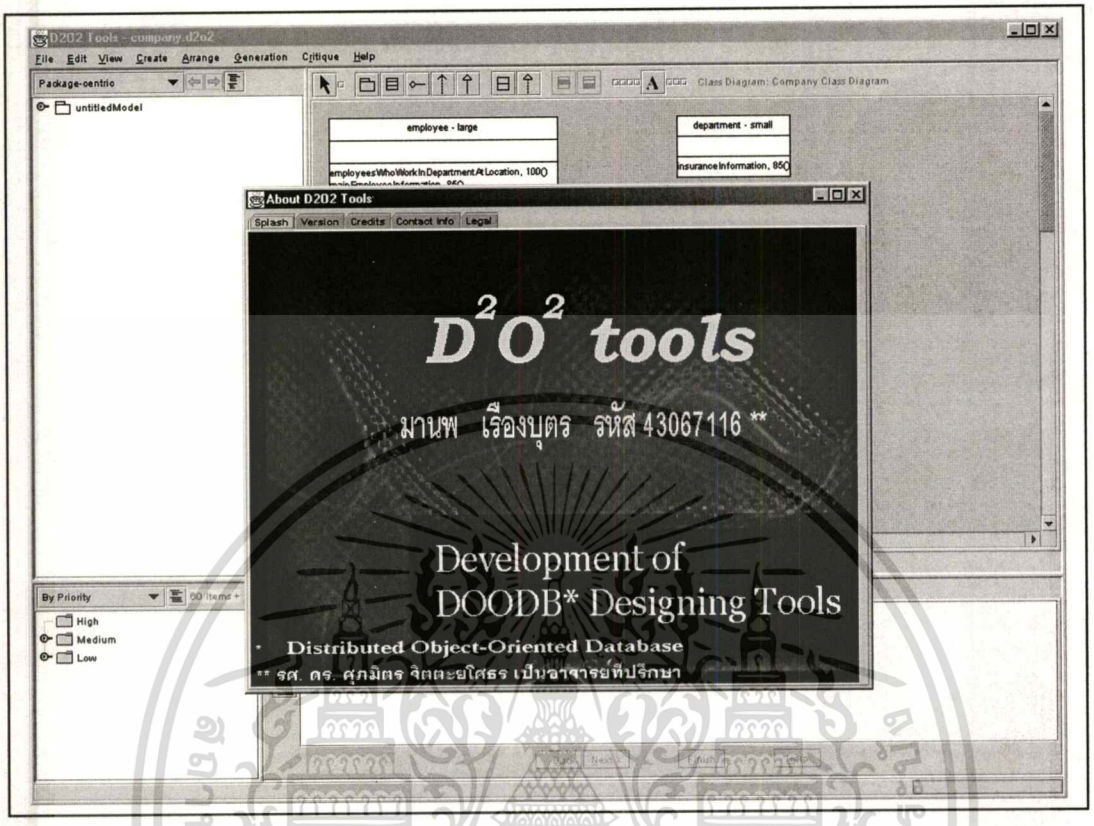
III. Algorithm Three สำหรับการทำ Vertical Fragmentation: เพื่อกำหนด Vertical Fragments ของ class ใน Cv

```
function VerticalFragmentation( Cv: set of classes to be vertically fragmented,
                                O: the set of operations)
returns Fv : set of vertical class fragments
begin
    for each Ck that is in Cv do
        for each Oi that is in O do
            for each element (attribute or method) ei of Ck that is accessed by Oi do
                for each element (attribute or method) ej of Ck that is accessed by Oi do
                    if there is a link between ei and ej then
                        value of this link += freq (Oi)
                    else
                        create a link between ei and ej
                        value of this link = freq (Oi)
        N = empty set of nodes; A = empty set of links; G = (N, A)
        firstNode = any element of Ck
        N += {firstNode}
        while there is an element of Ck that is not in N do
            chosenLink = the link with the greatest value to one of the graph extremities
            if chosenLink forms a cycle in the graph G then
                let cp be this cycle
                if cp can be an affinity cycle then
                    mark cp as a fragment candidate
            else
                if there is a fragment candidate then
                    let cf be this candidate
                    if cf cannot be extended then
                        mark cf as a fragment
                        Fv (Ck)+= cf
        Fv += Fv(Ck)
        if Ck is a root class in Ch then
            substitute Ck in Ch by its fragment that contains the relevant element for the
            derived→
                fragmentation
    return Fv
end
```

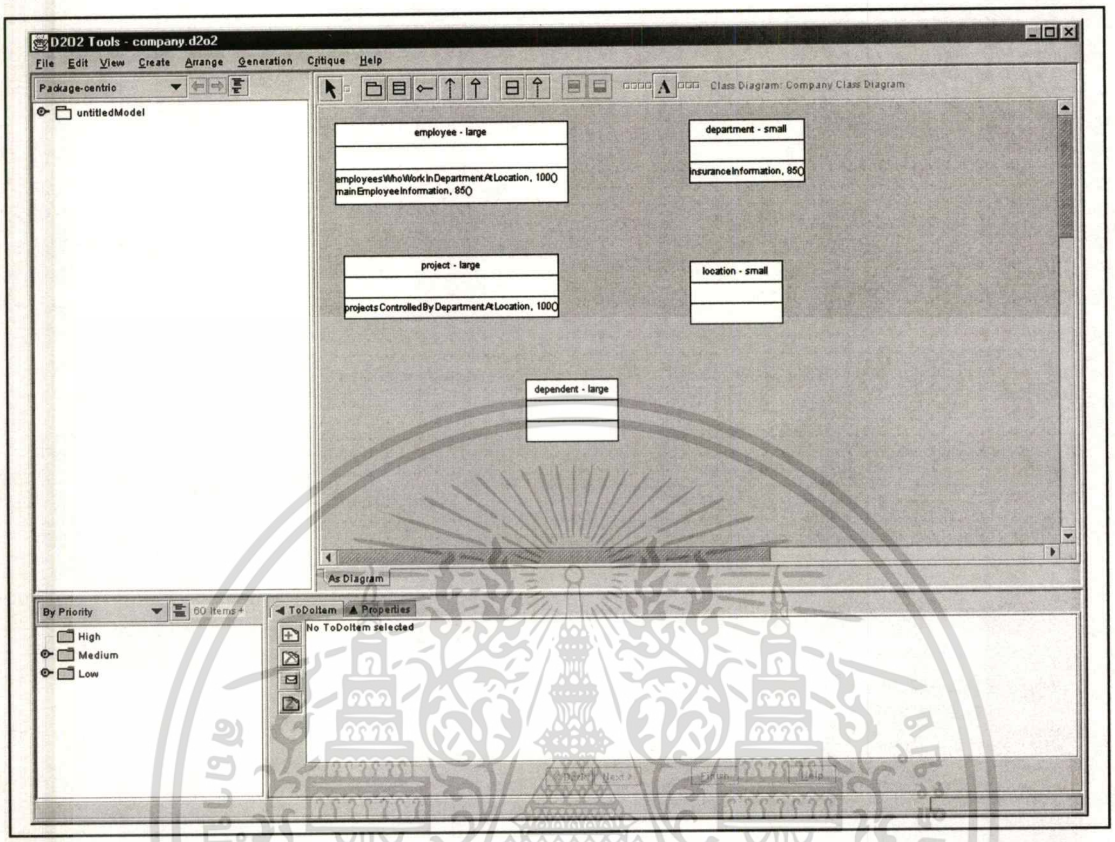
IV. Algorithm Four สำหรับการทำ Horizontal Fragmentation: เพื่อกำหนด Primary Horizontal Fragments และ/หรือ Mixed Fragments ของ class ใน Ch

```
function PrimaryHorizontalFragmentation( Ch: set of classes to be horizontally fragmented,
                                         O: set of operations,
                                         Fv: set of vertical class fragments )
returns Fh : set of horizontal class fragments
        Fm : set of mixed class fragments
begin
   Cr = empty set;
   for each list Li that is in Ch do
      Cr += root class of Li
   for each Ck that is in Cr do
      for each pair of operations Oi, Oj extracted from the same transaction→
         such that C(Oi) = C(Oj) = Ck do
         Oext += Oi; Oext += Oj
         if there is a link between Oi and Oj then value(link) += freq (Oi )
         else create a link between Oi and Oj with value(link) = freq (Oi)
      if Oext is empty then
         if the class has a large extension then
            define horizontal fragments of Ck in a circular manner
      else
         for each operation Oi that is in Oext do
            for each operation Oj that is in Oext do
               if Oi => Oj then // logic predicate
                  create a logic implication link between Oi and Oj
               if Oi and Oj are next to each other then
                  create a proximity link between Oi and Oj
         N = empty set of nodes; A = empty set of links; G = (N, A)
         N += any operation of Oext
         while there is an operation of Oext that is not in N do
            chosenLink = the link with the greatest value to one of the graph extremities
            if chosenLink forms a cycle in the graph G then
               let cp be this cycle
               if cp can be an affinity cycle then
                  mark cp as a fragment candidate
            else
               if there is a fragment candidate then
                  let cf be this candidate
                  if cf cannot be extended then
                     mark cf as a group of operations
         for each group of operations g = (O1, O2, .., Oq) do
            for each operation Oi that is in g do
               for each operation Oj that is in g do
                  if (Oi != Oj) and (Oi => Oj) then g -= Oi
            mark g as an operation term
         TO = {t1, t2, ..., tt} // set of operation terms
         Tab = empty table // table of operation terms on Ck
         E = empty set // set of Ck elements in Tab
         while there is a Ck element in TO which is not in E do
            let e be the less frequent element of Ck in TO such
               that e is not in E
            create a new column in Tab
```

```
            fill each element of the new column with operations over e such that the
                combination of elements in the same row defines a term in TO
        for each row r in Tab do
            if there are vertical fragments of Ck in Fv then
                // Mixed Fragmentation!!!
                Fm += combination of all elements in r applied to vertical fragments of Ck
                    containing these elements
            else
                Fh += the combination of all the elements in row r of Tab applied to the
            whole
                    class Ck
    return Fh, Fm
end
```



ภาพที่ 6.1 แสดงรูปหน้าจอของ About D^2O^2 ซึ่งจะมี Tab ให้ผู้ใช้ได้เข้าไปเลือกอ่านข้อมูลที่สนใจได้



ภาพที่ 6.2 แสดงพื้นที่ใช้สอย สำหรับนักออกแบบฐานข้อมูลในการทำงานกับ D^2O^2