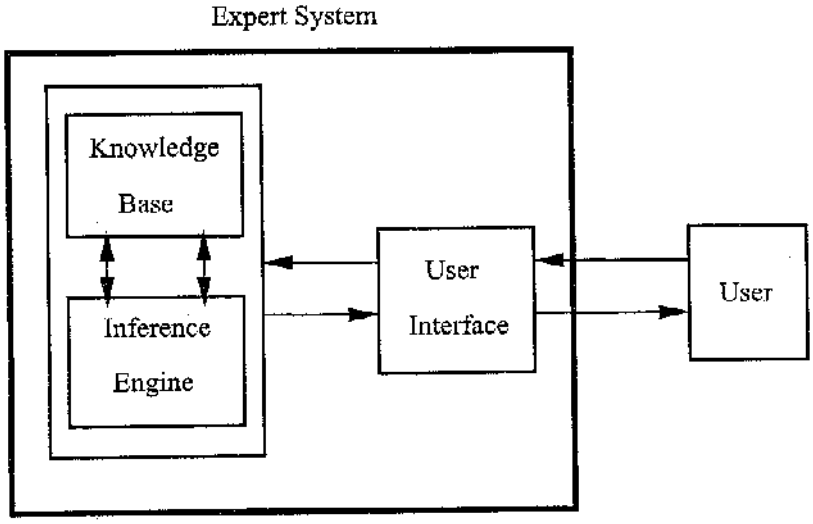
ระบบผู้เชี่ยวชาญแบบกระจายบนเครือข่ายคอมพิวเตอร์

สุรพงศ์ เอื้อวัฒนามงคล*

1. คำนำ

ระบบผู้เชี่ยวชาญ โดยทั่วไปจะประกอบด้วยส่วนที่สำคัญ 3 ส่วน [1, 3] คือ

- *Knowledge Base* เป็นฐานความรู้ของระบบผู้เชี่ยวชาญที่เก็บอยู่ในรูปแบบที่พร้อมนำมาใช้ในการแก้ปัญหา
- *Inference Engine* เป็นตัวแก้ปัญหาโดยอาศัยความรู้ของระบบในการหาคำตอบ (Logical Reasoning)
- *User Interface* ทำหน้าที่ติดต่อกับผู้ใช้ทั้งในด้านการรับข้อมูลและแสดงผลลัพธ์ของการแก้ไขปัญหา



ระบบผู้เชี่ยวชาญโดยส่วนใหญ่จะมีลักษณะเป็นระบบ Stand Alone คือ ทำงานตามลำพัง บนฐานของความรู้ที่เก็บอยู่ในระบบ ด้วยการทำงานในลักษณะนี้ทำให้ความสามารถในการแก้ปัญหาของระบบผู้เชี่ยวชาญถูกจำกัดทั้งในแง่ความเร็วและขนาดของฐานความรู้ การลดข้อจำกัดเหล่านี้อาจทำได้

* ผู้ช่วยศาสตราจารย์ สาขาวิทยาการคอมพิวเตอร์ คณะสถิติประยุกต์ สถาบันบัณฑิตพัฒนบริหารศาสตร์

โดยสร้างระบบผู้เชี่ยวชาญแบบกระจายที่สามารถช่วยกันทำงาน กล่าวคือการแก้ปัญหาของระบบผู้เชี่ยวชาญตัวหนึ่งอาจจะพึ่งระบบผู้เชี่ยวชาญอีกตัวหนึ่งเพื่อให้ช่วยแก้ไขปัญหาบางอย่างที่ตนทำไม่ได้ ปัญหาใดที่อยู่นอกฐานความรู้ของตนก็สามารถสอบถามไประบบผู้เชี่ยวชาญอื่น ซึ่งจะทำให้ฐานความรู้รวมของทั้งระบบขยายออกไปได้อย่างไม่จำกัด

บทความนี้จะเสนอการออกแบบระบบผู้เชี่ยวชาญแบบกระจายบนเครือข่ายคอมพิวเตอร์ตามแนวทาง Cooperating Expert System ที่ได้กล่าวมา ระบบผู้เชี่ยวชาญที่ทำงานอยู่บน Host ต่าง ๆ ในระบบเครือข่ายจะสามารถแก้ไขปัญหาใดปัญหาหนึ่งร่วมกัน โดยระบบผู้เชี่ยวชาญหนึ่งสามารถส่งปัญหาให้ระบบผู้เชี่ยวชาญอีกระบบหนึ่งช่วยแก้ไขปัญหาให้ในลักษณะ Client-Server Distributed Processing ได้

2. โครงสร้างฐานข้อมูล Knowledge-Based Predicates

การเก็บฐานความรู้ (Knowledge Base) สามารถทำได้หลายรูปแบบ เช่น Production Rules, Semantic Networks, Frames, Predicate Logics เป็นต้น ในบรรดารูปแบบการเก็บเหล่านี้ Predicate Logics นับเป็นวิธีการหนึ่งที่ง่ายต่อการทำ Logical Reasoning ของ Inference Engine [1, 3, 6] ในงานวิจัยนี้เราจึงเลือกรูปแบบ Predicate Logics ที่ใช้ Horn's Clause ในการเก็บฐานความรู้ รูปแบบของ Horn's Clause มีลักษณะ ดังนี้

$$H \leftarrow B_1 \wedge B_2 \wedge ... \wedge B_n$$

โดยที่ H คือ Head ของ clause และ $B_1, B_2, ..., B_n$ คือ Subgoals ใน Body ของ Clause ตัวอย่างเช่น

$$gp(X, Y) \leftarrow p(X,Z) \wedge p(Z,Y)$$

เราสามารถอ่าน Clause ข้างบนได้เป็น 2 ลักษณะ คือ

1. gp (X, Y) จะเป็นจริง ถ้า p(X, Z) และ p(Z, Y) เป็นจริง
2. ถ้าจะแก้ปัญหา gp(X,Z) จะต้องแก้ปัญหา p(X, Z) และ p(Z, Y)

การแก้ปัญหาด้วยการทำ Logical Reasoning กับ Horn's Clause สามารถทำได้ด้วยขบวนการ Inference ที่เรียกว่า Resolution ซึ่งเป็นขบวนการในการพิสูจน์ เพื่อหาข้อสรุปจากฐานความรู้ (Knowledge base) ตัวอย่างเช่น กำหนดให้ Clause ทั้งสามต่อไปนี้อยู่ในฐานความรู้

$p(a, b)$ (1)

$p(b, c)$ (2)

$gp(X, Y) \leftarrow p(X, Z) \wedge p(Z, Y)$ (3)

ถ้าต้องการพิสูจน์ว่า Goal gp(a, c) เป็นจริงหรือไม่ เราจะตั้งสมมติฐานว่า Goal gp(a, c) เป็นจริง แต่ใช้ขบวนการ Resolution เพื่อการพิสูจน์โดยเริ่มจากการทำ pattern matching หรือ Unification ระหว่าง Head ของ Clause ต่าง ๆ และ Goal ที่ต้องการพิสูจน์ ปรากฏว่า Head ของ Clause(3) และ Goal สามารถทำ Unification ได้สำเร็จ ซึ่งจะได้ว่า X=a และ Y=c ดังนั้นจึงสามารถสรุปได้ว่า Subgoals ต่อไปนี้จะต้องเป็นจริงด้วย (ตามหลักของ Monus Ponen)

$$\mathbf{p(a, Z) \wedge p(Z, c)}$$

3. Unification Algorithm

การทำ Unification ระหว่าง Predicate หรือ Goal กับ Head ของ Clause นับเป็นขบวนการที่สำคัญในการทำ Inference เพราะเป็นขั้นตอนแรกของการทำ Inference และมีผลต่อประสิทธิภาพของระบบโดยรวม ขบวนการ Unification จะใช้วิธีทำ Pattern Matching ระหว่าง Goal และ Head ของ Clause การทำ Pattern Matching จะเริ่มจากชื่อ Predicate ตามด้วย Arguments ทีละตัวตามลำดับ ถ้าส่วนใดส่วนหนึ่งไม่ Match หรือตรงกัน จะถือว่า Unification กระทำไม่สำเร็จ ตัวอย่างเช่น

Goal : $\mathbf{gp(a, c)}$

Clause : $\mathbf{gp(X, Y) \leftarrow p(X, Z) \wedge p(Z, Y)}$

Goal และ Head ของ Clause ข้างบนจะสามารถ Unify กันได้ ผลลัพธ์จากการ Unify จะทำให้เกิดการแทนค่ากันระหว่าง Variables ใน Clause และ Argument ของ Goal ตัวอย่างข้างบนจะได้การแทนค่า (Variable Substitutions) ดังนี้

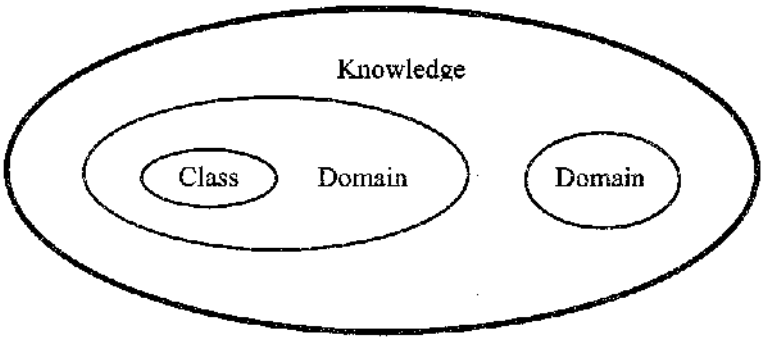
$$\mathbf{\{X=a, Y=c\}}$$

ข้อสังเกต Variable Z ใน Clause ยังไม่มีค่า (Unbound) เนื่องจากยังไม่มีการแทนค่าเกิดขึ้นกับ Z

ด้วยขบวนการ Resolution เราจะต้องพิสูจน์ Subgoals ใน Body ของ Clause คือ p(a, Z) และ p(Z, c) ว่าเป็นจริงต่อไป สำหรับ Subgoal p(a, Z) จะสามารถ Unify กับ Head ของ Clause (1) ซึ่งทำให้ Subgoal นี้เป็นจริง และจะได้ว่า Z=b ดังนั้น Subgoal p(Z, c) จึงถูกเปลี่ยนเป็น p(b, c) ซึ่งจะเห็นว่าสามารถ Unify กับ Head ของ Clause (2) ซึ่งทำให้ Subgoal นี้เป็นจริงเช่นกัน เมื่อ Subgoals ทั้งสองสามารถพิสูจน์ได้ว่าเป็นจริงจึงสรุปได้ว่า gp(a, c) เป็นจริงตามสมมติฐาน

4. Hybrid Knowledge Model

โดยที่ผู้เชี่ยวชาญ (มนุษย์) มักจะมองสิ่งต่าง ๆ รอบตัวเป็นวัตถุ (Objects) ทำให้องค์ความรู้ต่าง ๆ จะเกี่ยวข้องกับคุณสมบัติของวัตถุและความสัมพันธ์ระหว่างวัตถุ การนำหลักการ Object-oriented Data Model มาผสมผสานกับการเก็บฐานความรู้ในรูป Predicate หรือ Logic-based จะเป็นการช่วยให้มนุษย์สามารถถ่ายทอดความรู้เพื่อเก็บในฐานข้อมูลได้ง่ายขึ้น การผสมผสานระหว่าง Logic-based และ Object-Oriented Knowledge Model ก่อให้เกิด Hybrid Knowledge Model ที่ Methods ภายในของ Classes ของ Objects ถูกกำหนดอยู่ในรูปของ Clauses นอกจากนี้ Classes ที่มีความสัมพันธ์กันควรจัดรวมกันเป็นกลุ่มหรือ Domain ทำให้สามารถแบ่งฐานความรู้เป็นส่วน ๆ และแยกจัดเก็บฐานความรู้ (Domain) กระจายไปบน Host ต่าง ๆ ตามแหล่งกำเนิดและการเรียกใช้ของฐานความรู้นั้น ๆ



5. Universal Predicate Locator (UPL)

Universal Predicate Locator เป็นชื่อ (Path) ที่เราจะใช้เรียก Predicate ที่เก็บอยู่ในฐานความรู้ต่าง ๆ บนระบบเครือข่าย ตัว Path จะต้องสามารถระบุชื่อของ Predicate, Arity (จำนวน Arguments ของ Predicate), Class, Domain และ Host Name ที่เก็บ Predicate นั้น เช่น

Predicate/Arity/Class/Domain/Hostname

โดยที่ Class, Domain และ Hostname ใน Path อาจไม่ต้องระบุในกรณีที่ไม่ทราบ (Unknown) เช่น p/2///, g/3/c// เป็นต้น

6. Predicate Name Server (PNS)

ในระบบผู้เชี่ยวชาญแบบกระจายที่ได้ออกแบบนี้ เราจะมี Predicate Name Server (PNS) ทำหน้าที่เสมือน Directory Server คอยค้นหา UPL ของ Predicate ที่Client ต้องการ Client อาจสอบถามโดยกำหนดบางส่วนของ UPL เช่น ไม่กำหนด Class, Domain หรือ Host Name เป็นต้น PNS จะค้นหา UPL ของ Predicate ที่ต้องการจากฐานข้อมูลของตน (Directory Database) ถ้าค้นพบจะส่งค่า UPL ที่สมบูรณ์กลับไปยัง Client แต่ถ้าไม่พบจะส่งค่า Null กลับไปเพื่อบอกให้ Client ทำการค้นหากับ PNS อื่น ๆ ต่อไป การติดต่อกับ PNS อาจใช้หลักการของ Remote Procedure Call (RPC) ซึ่งเป็นวิธีทั่วไปที่ใช้เรียกบริการจาก Host หนึ่งไปยังอีก Host หนึ่ง

7. Extended Horn's Clause

เมื่อมีการผสมผสาน Object Oriented Concept เข้ากับ Logic-Based Predicate รูปแบบของ Horn's Clause จึงต้องมีการเปลี่ยนแปลงเพื่อรองรับ Object Oriented Concept สิ่งที่จะต้องเปลี่ยนแปลงใน Horn's Clauses ได้แก่การเรียกหรืออ้างถึง Predicates ต่าง ๆ ในระบบเครือข่าย ซึ่งพอสรุปได้ดังต่อไปนี้

1. Predicate Name ในส่วน Head ของ Clause ควรจะต้องเพิ่ม Class และ Domain Name เช่น

Predicate/Class/Domain (Argument List)

2. Subgoals ใน Body ของ Clause จะต้องกำหนดด้วย UPL แทน ในกรณีที่ไม่ต้องการกำหนดUPL ที่ครบถ้วนก็สามารถทำได้ ตัวอย่างเช่น

gp/person/family (X,Y) ← p/person/family/ (X,Z) ∧ p/person/family/ (Z,Y)

Clause ดังตัวอย่างข้างบน กำหนด gp ว่าเป็น Predicate ที่อยู่ใน Class "person" และ Domain "family" Subgoals p ทั้งสองใน body มี UPL ที่ไม่กำหนดชื่อ Host แน่นอน ดังนั้นจึงต้องมีการสอบถามไปยัง PNS เพื่อให้ได้ UPL ของ p ที่สมบูรณ์

8. Searching Rules สำหรับการค้นหา UPL ของ Predicates

การค้นหา UPL ของ Predicate ได้อย่างมีประสิทธิภาพ นับเป็นปัจจัยอันหนึ่ง ที่จะทำให้การทำงานของระบบเป็นไปได้อย่างรวดเร็ว และมีประสิทธิภาพ UPL ของ Predicates ต่าง ๆ อาจแยกจัดเก็บไปบน PNS หลายตัว โดยแต่ละตัวจะทำหน้าที่ค้นหา UPL และบำรุงรักษาฐานข้อมูล UPL ของตนเท่านั้น

การสอบถามไปยัง PNS ต่าง ๆ ในระบบควรจะมีการกำหนดลำดับในการสอบถาม เช่น ควรจะสอบถามกับ Local PNS ก่อน เพราะ Predicates ที่ต้องการอาจจัดเก็บอยู่ที่ Local Host เป็นส่วนใหญ่ ในกรณีที่ไม่พบใน Local PNS จึงสอบถาม PNS ที่ห่างไกลออกไปตามลำดับ จนกระทั่งได้รับคำตอบ คือ UPL ที่ต้องการ

เนื่องจากการสอบถามไปยัง Remote PNS จะใช้เวลาค่อนข้างมาก ดังนั้นถ้าต้องการค้นหา UPL บน Local Host เท่านั้น เราอาจใช้เครื่องหมาย * แทน Host Name ก็ได้ เช่น gp/2///* เป็นต้น นอกจากนี้เราสามารถ Cache ตัว UPL ที่เคยสอบถามไปแล้วทำให้ลดการสอบถามไปยัง PNS อันจะทำให้การทำงานของระบบมีความรวดเร็วยิ่งขึ้น อย่างไรก็ตาม UPL แต่ละตัวที่ถูก Cache ไว้ ควรจะมีอายุเวลา (life time) กำกับ เมื่อ UPL ที่ Cache ไว้หมดอายุจะถูกลบทิ้งโดยอัตโนมัติ ซึ่งถ้าต้องการ UPL นี้อีกก็ต้องสอบถาม PNS ใหม่ ทั้งนี้เพื่อให้ UPL ใน Cache เป็นข้อมูลที่ทันสมัยเสมอ

9. การทำ Inference ในลักษณะ Distributed Processing

ในการสร้างระบบผู้เชี่ยวชาญแต่ละตัวในเครือข่าย เราอาจทำได้โดยการสร้าง Daemon Process ที่รอรับ Request จาก Client เราจะขอเรียก Daemon Process นี้ว่า Knowledge Server (KNS) การแก้ปัญหาด้วยวิธีการทำ Inference บน Predicates ที่กระจายอยู่ตามฐานความรู้ของ KNS หลาย ๆ ตัวจะต้องมีการติดต่อเพื่อประสานการทำงานระหว่าง KNS เหล่านี้ การทำ Inference บน Subgoal หนึ่งอาจได้ผลลัพธ์หลายคำตอบ ซึ่งแต่ละคำตอบจะถูกนำไปใช้ในการทำ Inference ของ Subgoals ที่อยู่ถัดไปใน Clause Body คำตอบเหล่านี้อาจถูกสร้างจากการ Inference ของ KNS หนึ่ง และถูกใช้โดยอีก KNS หนึ่ง ดังนั้น Synchronization ระหว่าง KNS ซึ่งเป็น Producer และ Consumer ของคำตอบจึงเป็นสิ่งจำเป็นต่อการทำงานของระบบ

10. ขั้นตอนการทำงานของ KNS ในการทำ Inference

ขั้นตอนการทำงานของ KNS สามารถสรุปได้ดัง Algorithm ต่อไปนี้

1) KNS Daemon รอรับ Request จาก Client

2) เมื่อได้รับ Request แล้ว KNS Daemon จะสร้าง Service Process เพื่อการทำ Inference บน Predicate และ Arguments ที่ได้รับมา หลังจากนั้นจึงกลับไปรอรับ Request ดังเดิมในข้อ 1 ส่วน Service Process จะเช็คดูว่า Predicate นั้นอยู่ในฐานความรู้ของตนหรือไม่ ถ้าไม่อยู่จะส่งคำตอบ Null กลับไปเพื่อแสดงว่า ไม่สามารถหาคำตอบได้

3) ถ้า predicate นั้นอยู่ในฐานความรู้ จึงทำงานตามขั้นตอนต่อไปนี้

a) ทำ Unification ระหว่าง Predicate และ Head ของ Clause ที่ Match กับ Predicate นั้น

b) ในแต่ละ Clause ที่ Unify สำเร็จจะสร้าง Clause Process เพื่อทำการ Inference ต่อไปกับ Subgoals ที่อยู่ใน Body ของ Clause นั้น

c) Clause Process ทำการแก้ปัญหาทีละ Subgoals ใน Body จนกระทั่งหมด Subgoalsใน Body ก็จะได้ Set ของคำตอบทั้งหมด (หรือ Null ถ้าไม่มีคำตอบเลย) หลังจากนั้นส่ง Set ของคำตอบกลับไปยัง Service Process แล้วจึงหยุดทำงาน

4. Service Process เมื่อได้รับ Set ของคำตอบครบถ้วนจากทุก Clause Processes แล้วจึงรวบรวมคำตอบทั้งหมดที่ได้สร้างเป็น Set ของคำตอบรวมของ Predicate และส่ง Set ของคำตอบเหล่านั้นกลับไปยัง Client หลังจากนั้นจึงหยุดทำงาน

11. Clause Representation

การทำ Inference กับ Clause นับเป็นขั้นตอนหลักของการทำงานของ KNS รูปแบบของการเก็บ Clause ที่ง่ายต่อการใช้งานจึงมีส่วนสำคัญที่จะเอื้ออำนวยต่อการทำ Inference ให้เป็นไปได้อย่างรวดเร็วและมีประสิทธิภาพ โดยที่ Head และ Body ของ Clause มีส่วนประกอบหลักคือ Predicate เราจึงอาจใช้ Template เก็บรายละเอียดของ Predicate ดังโครงสร้างข้อมูลต่อไปนี้

Predicate Template

Predicate UPL	Argument-1 Template	Argument-2 Template

Argument Template

Tag	Data

โดยที่

Tag เป็นตัวระบุชนิดของ Data เช่น Variable Id, Pointer ไปยังข้อมูล เช่น String, Integer, List, Structure เป็นต้น

Data เป็นส่วนเก็บข้อมูลตามชนิดที่ระบุด้วย Tag

12. Environment ภายใน Clause

จากขั้นตอนการทำงานของ KNS ที่กล่าวมาข้างต้น Service Process จะถูกสร้างโดย KNS Daemon เพื่อเป็นผู้กระทำ Unification ระหว่าง Goal และ Head ของ Clause ถ้า Unification กระทำได้สำเร็จ (Success) จะเกิดการแทนค่า ระหว่าง Arguments ที่ส่งมากับ Goal และ Variables ต่าง ๆ ภายใน Clause นั้น Variables ที่ได้หลังจากถูกแทนค่ารวมเรียกว่า Environment ซึ่งจะถูกเก็บอยู่ในเนื้อที่ที่เรียกว่า Local Frame

Arguments ของ Goal ที่ส่งมายัง KNS จะถูกเก็บอยู่ภายในเนื้อที่ ที่เรียกว่า Incoming Frame การเก็บ Arguments ไว้ภายใน Incoming Frame ก็เพื่อใช้เป็นรูปแบบ (Template) ในการสร้างคำตอบ (Return Frame) เพื่อจัดส่งกลับไปยัง Client เมื่อการแก้ปัญหาสำเร็จเรียบร้อยแล้ว

ในกรณีที่ Clause Process ต้องการส่ง Goal และ Arguments ไปยัง KNS อื่นเพื่อขอให้ช่วยแก้ไขปัญหา Clause Process นอกจากจะสร้าง Outgoing Frame ซึ่งประกอบด้วย Goal และ Arguments จะสร้าง Outgoing Template ซึ่งเป็น Template ที่บอกว่าได้จัดส่ง Variables ใดไปใน Outgoing Frame ซึ่ง Outgoing Template นี้จะช่วยให้ KNS สามารถนำค่า Arguments จาก Return Frame กลับมาแทนค่าใน Clause Variables ได้อย่างถูกต้อง

13. โครงสร้างของ Local Frame

Local Frame จะประกอบด้วย Variable slots จำนวน n+1 ช่อง โดยที่

Slot 0 จะเก็บจำนวน Variables ใน Local Frame คือค่า n

Slot 1 to n จะเก็บ Tag และค่าของ Variables แต่ละตัวตามลำดับ

Slots ใน Local Frame จะประกอบด้วยส่วนต่อไปนี้

1. Tag เป็นตัวบอกชนิดของข้อมูลที่เก็บอยู่ใน Slot ซึ่งสามารถแยกเป็น 2 ประเภท คือ
   a) Variable ที่ถูก Assign ให้มีค่าคงที่ เช่น
      เลขจำนวนเต็ม (Integer)
      Pointer ไปยัง String ของตัวอักษร
      Pointer ไปยัง Structure หรือ Linked List

b) Variable ที่ไม่ถูก Assign ให้มีค่าคงที่ แยกออกเป็น 2 สถานะ คือ

Bound แสดงถึงสถานะที่ Variable นี้ได้ถูกกำหนดให้มีค่าเท่ากับ Variable อื่นใน Local Frame เดียวกัน

Unbound แสดงถึงสถานะที่ Variable ยังไม่ถูกกำหนดให้มีค่าใดเลย

2. ข้อมูล ตามที่ Tag ระบุ เช่น

a) ค่าคงที่ต่าง ๆ เช่น เลขจำนวนเต็ม, Pointer เป็นต้น

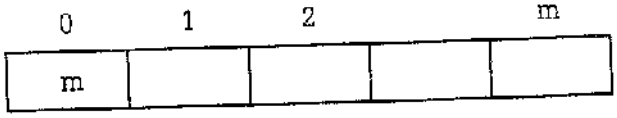
b) Variable

กรณี Bound ข้อมูลจะเป็นค่า Variable Id ของ Variable อื่นที่มีค่าเท่ากับ Variable นี้

กรณี Unbound ข้อมูลจะมีค่า -1 ซึ่งไม่ได้นำไปใช้งานใด ๆ

เนื่องจาก Local Frame เป็นที่เก็บ Environment ภายใน Clause ข้อมูลใน Local Frame จึงควรสมบูรณ์ภายในตัวเองหรือมีลักษณะที่เรียกว่า Closed Environment [7] การอ้างอิงไปยัง Variable อื่นนอก Local frame จะไม่สามารถกระทำได้ การที่ Local Frame มีลักษณะเป็น Closed Environment จะทำให้การ Inference ภายใน Clause อาศัยข้อมูลภายใน Local Frame เท่านั้น ซึ่งวิธีการนี้จะเหมาะกับการทำงานในลักษณะ Distributed Processing เป็นอย่างมาก เนื่องจากไม่มีการ Access ข้อมูลในลักษณะ Global References

4. โครงสร้างของ Incoming และ Outgoing Frames



Incoming และ Outgoing Frames จะประกอบด้วย Arguments และ Variable Slots จำนวน m+1 ช่อง โดยที่

Slot 0 จะเก็บจำนวน Slots ใน Frame นี้ คือค่า m ซึ่งเท่ากับ a+p โดย

a = จำนวน Arguments (Arity)

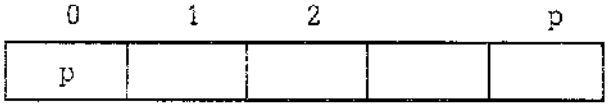
p = จำนวน Variables ใน Frame นี้

Slot 1 to a จะเก็บ Tag และค่าของ Arguments ตามลำดับ (โครงสร้าง เช่นเดียวกับ Slot ใน Local Frame)

Slot (a+1) to m จะเก็บ Tag และค่าของ Variables ใน Frame นี้ (โดยโครงสร้าง มีลักษณะเช่นเดียวกับ Slot ใน Local Frame)

ทั้ง Incoming และ Outgoing Frames จะมีลักษณะ Closed หรือสมบูรณ์ในตัวเอง เช่นกัน

15. โครงสร้างของ Outgoing Template

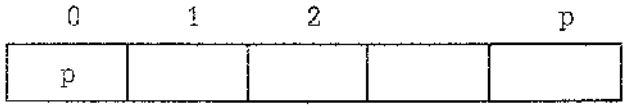


Outgoing Template จะประกอบด้วย Slot จำนวน p + 1 ช่อง โดยที่

Slot 0 จะเก็บจำนวน Slots ใน Frame นี้ คือค่า p ซึ่งจะมีค่าเท่ากับจำนวน Variables ใน Corresponding Outgoing Frame ที่ส่งไปยัง KNS อื่น

Slot 1 to p จะเก็บค่า Id ของ Variables ใน Local Frame ที่ ถูกส่งไปเป็น Variables ใน Corresponding Outgoing Frame

16. โครงสร้างของ Return Frame



Return frame จะประกอบ Variable slots จำนวน p + 1 ช่อง โดยที่

Slot 0 จะเก็บจำนวน Variables ใน Frame นี้ ในกรณีนี้คือ p ซึ่งจะมีค่าเท่ากับจำนวน Variables ใน Incoming Frame

Slot 1 to p จะเก็บค่าของ Variable แต่ละตัวตามลำดับ

Return Frame จะมีลักษณะ Closed เช่นเดียวกับ Local Frame

17. การส่งผ่านค่าของ Variables ระหว่าง Client และ KNS

การส่งผ่านค่าของ Variables ระหว่าง Client และ KNS เกิดขึ้นได้ใน 2 ช่วง คือ

1. ช่วงส่ง Request เมื่อ Client ต้องการส่ง Request ไปยัง KNS เพื่อให้แก้ไขปัญหาหนึ่ง Client จะสร้าง Outgoing Frame ซึ่งประกอบด้วย Arguments และค่า Variables ต่าง ๆ ที่เกี่ยวข้อง เพื่อจัดส่งไปพร้อมกับ Request นั้น ในขณะเดียวกันก็จะสร้าง Outgoing Template เพื่อระบุว่า Local

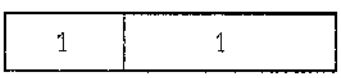
Variables ใดที่ถูกส่งไปใน Outgoing Frame นั้น เมื่อ KNS ได้รับ Goal และ Outgoing Frame ก็จะสร้าง Service Process ที่จะทำการ Unification ระหว่าง Goal/Arguments และ Heads ของ Clauses ต่าง ๆ ในฐานข้อมูลตนถ้า Unification กระทำได้สำเร็จ Service Process จะสร้าง Clause Process พร้อม Local Frame และ Incoming Frame สำหรับแต่ละ Clause ที่ Unification กระทำสำเร็จ แต่เนื่องจากการทำ Unification อาจมีการแทนค่าระหว่าง Variables ใน Incoming และ Local Frame ดังนั้น Variables ใดใน Incoming Frame ถ้าถูก Assign ให้มีค่าเท่ากับ Variables ใน Local Frame ค่าของ Variables นั้น ใน Incoming Frame ก็จะมีค่าเป็น -v โดยที่ v คือ Id ของ Variables ใน Local Frame ที่ถูกทำให้มีค่าเท่ากัน เครื่องหมายลบจะเป็นตัวแสดงถึงความเป็น Variable ใน Local Frame ซึ่งต่างกับ Variable ใน Incoming Frame ซึ่งจะมีเครื่องหมายเป็นบวก

2. ช่วงส่งคำตอบ : หลังจาก Clause Process ได้ทำการ Inference ทุก Subgoals ใน body สำเร็จแล้ว ผลลัพธ์ที่ได้ คือ Set ของคำตอบทั้งหมดของ Clause นั้น แต่ละคำตอบสามารถ Represent ได้ด้วย Local Frame ของคำตอบนั้น ๆ หน้าที่ต่อไปของ Clause Process ก็คือการสร้าง Return Frame ของแต่ละคำตอบซึ่งทำได้โดยการแทนค่า Variables ต่าง ๆ ใน Incoming Frame ด้วยค่า Variables ใน Local Frame เมื่อได้ Return Frames ของทุกคำตอบแล้ว Clause Process จึงส่ง Set ของ Return Frames เหล่านี้กลับไปยัง Service Process คำตอบจากทุก Clauses จะถูกรวบรวมและจัดเรียงตาม Textual Ordering ของ Clauses (ลำดับของ Clauses ที่เก็บใน Database) Client

เมื่อ Client ได้รับ Set ของคำตอบ ก็จะทำการแทนค่าของ Variables ในคำตอบกลับไปยัง Local Frame (โดยใช้ Outgoing Template ช่วยในการแทนค่า) แต่ละ Local Frame ใหม่ที่ได้จากการแทนค่าจะถูกนำไปใช้ในการทำ Inference ของ Subgoal ถัดไปใน Body ของ Clause

ภาพต่อไปนี้จะแสดงตัวอย่างการสร้าง Frames ต่าง ๆ และการส่งค่า Variables ระหว่าง Client และ KNS เพื่อทำการแก้ปัญหา gp(X, c)

Goal : gp(X, c)



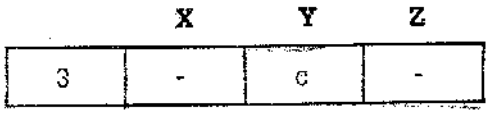
Outgoing Template $_0$

Outgoing Frame $_0$

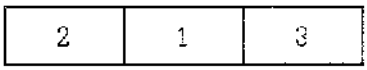
gp(X, Y) ← p(X, Z) ∧ p(Z, Y) (1)

Goal unified with Head of the clause (1)

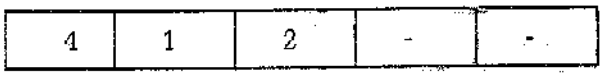
Incoming Frame $_1$



Local Frame $_1$

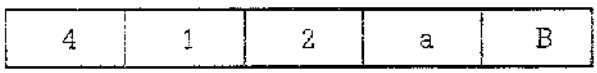


Outgoing Template $_1$

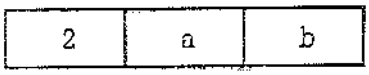


Outgoing Frame $_1$

p(X, Y) unified with p(a, b)

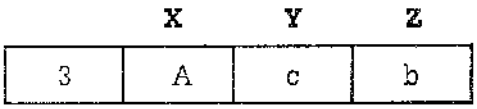


Incoming Frame $_2$



Return Frame $_2$

Return to Clause (1)



Local Frame $_3$

18. บทสรุป

การออกแบบระบบผู้เชี่ยวชาญแบบกระจาย ดังที่เสนอในบทความนี้เป็นเพียงต้นแบบสำหรับการสร้างระบบผู้เชี่ยวชาญที่สามารถทำงานร่วมกันในระบบเครือข่ายคอมพิวเตอร์ โปรแกรมต้นแบบของระบบจะได้มีการพัฒนาขึ้นเพื่อตรวจสอบความถูกต้องและรายละเอียดปลีกย่อยต่าง ๆ ของการออกแบบ รวมทั้งทดสอบประสิทธิภาพ เพื่อการปรับปรุงระบบให้ดียิ่งขึ้นต่อไป

บรรณานุกรม

1. Intelligent Systems for Business, Fatemeh Zahedi, Wadsworth MIS series, 1993.
2. Power Programming with RPC, John Bloamer, O'Reilly & Associates, 1992.
3. Expert Systems for Business: Concepts and Applications, D.V. Pigford และ Greg Baur, Boyd & Fraser Publishing Company, 1995.
4. Logic Programming: Prolog and Stream Parallel languages, J.D.Newmarch, Prentice Hall, 1990.
5. Prolog++: The Power of Object-Oriented and Logic Programming, Chris Moss, Addison-Wesley, 1994.
6. Deductive Databases and Logic Programming, Subrata Kumar Das, Addison Wesley, 1992.
7. J.S. Conery, Parallel Execution of Logic Programs, Kluwer Academic Publishers, 1987.
8. Techniques of Prolog Programming, T.Van Le, John Wiley & Sous Publisher, 1993.
9. Introduction to Expert Systems, Peter Jackson, Addison-Wesley, Second Edition, 1990.
10. Artificial Intelligence:Theory and Practice, Thomas Dean, James Allen and Yiannis Aloimonos, Benjamin/Cummings Publishing, 1995.