ระบบผู้เชี่ยวชาญแบบกระจายบนเครือข่ายคอมพิวเตอร์

(Distributed Expert System)

ดร.สุรพงค์ เอื้อวัฒนามงคล[1]

บทคัดย่อ (Abstract)

ในปัจจุบันระบบผู้เชี่ยวชาญ (Expert System) ได้ถูกพัฒนาไปอย่างมาก เพื่อช่วยเหลือมนุษย์ในการตัดสินใจและแก้ไขปัญหาต่าง ๆ ที่ต้องอาศัยฐานความรู้ (Knowledge Base) อย่างไรก็ตาม ระบบผู้เชี่ยวชาญยังมีความจำกัดในขนาดของฐานความรู้ และประสิทธิภาพของระบบ ทั้งนี้เนื่องจากระบบผู้เชี่ยวชาญส่วนใหญ่ ยังเป็นระบบที่ทำงานแบบอิสระโดยลำพัง (Standalone) งานวิจัยของบทความนี้จะเป็นความพยายามที่จะศึกษาและออกแบบระบบผู้เชี่ยวชาญแบบกระจายที่ทำงานบนระบบเครือข่ายคอมพิวเตอร์โดยระบบผู้เชี่ยวชาญจะมีความสามารถที่จะแก้ไขปัญหาร่วมกันได้ ระบบผู้เชี่ยวชาญแต่ละตัวจะแก้ไขปัญหาที่อาศัยฐานความรู้ของตนรวมทั้งสามารถสอบถามไประบบผู้เชี่ยวชาญอื่น เพื่อให้แก้ไขปัญหาที่ตนไม่สามารถทำได้ ซึ่งวิธีการนี้จะช่วยให้การแก้ไขปัญหาของทั้งระบบมีประสิทธิภาพและขยายฐานความรู้ออกไปได้อย่างไม่จำกัด

---

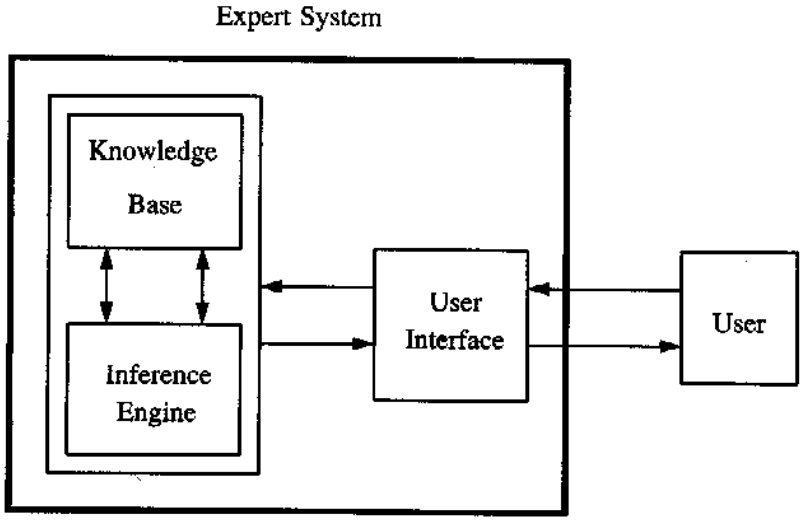
[1] สาขาวิชาวิทยาการคอมพิวเตอร์ คณะสถิติประยุกต์ สถาบันบัณฑิตพัฒนบริหารศาสตร์ สถาบันบัณฑิตพัฒนบริหารศาสตร์

1. บทนำ

การพัฒนาระบบผู้เชี่ยวชาญ เป็นความพยายามที่จะใช้ความรู้ทางด้านปัญญาประดิษฐ์และการแก้ไขปัญหามาสร้างระบบที่ทำตัวเหมือนเป็นผู้เชี่ยวชาญที่ให้คำปรึกษา หรือแก้ปัญหาโดยใช้ความรู้เฉพาะด้านที่ตนมีอยู่ เช่น การวิเคราะห์วินิจฉัยโรค การวิเคราะห์และแก้ไขปัญหาของเครื่องยนต์ วิเคราะห์ตลาดการเงิน เป็นต้น การพัฒนาระบบผู้เชี่ยวชาญได้กระทำอย่างต่อเนื่องมานานกว่า 20 ปี ในปัจจุบันระบบผู้เชี่ยวชาญได้พัฒนามาถึงจุดที่เริ่มมีการใช้ระบบผู้เชี่ยวชาญมาช่วยทำงานแทนมนุษย์อย่างจริงจัง โดยเฉพาะอย่างยิ่งในงานบางอย่างระบบผู้เชี่ยวชาญสามารถทำงานแทนมนุษย์ได้อย่างดีเยี่ยม

ระบบผู้เชี่ยวชาญ โดยทั่วไปจะประกอบด้วยส่วนที่สำคัญ 3 ส่วน [1, 3] คือ

- *Knowledge Base* เป็นฐานความรู้ของระบบผู้เชี่ยวชาญที่เก็บอยู่ในรูปแบบที่พร้อมนำมาใช้ในการแก้ปัญหา
- *Inference Engine* เป็นตัวแก้ปัญหาโดยใช้หลักการใช้เหตุผล (Reasoning) บนพื้นฐานความรู้ของระบบ
- *User Interface* ทำหน้าที่ติดต่อกับผู้ใช้ทั้งในด้านการรับข้อมูล แสดงผลลัพธ์ของการแก้ไขปัญหา



ระบบผู้เชี่ยวชาญที่มีใช้อยู่ในปัจจุบันโดยส่วนใหญ่จะมีลักษณะเป็นระบบ Standalone คือ ทำงานตามลำพังบนฐานของความรู้ที่เก็บอยู่ในระบบ ด้วยข้อจำกัดนี้ ทำให้ความสามารถในการแก้ปัญหา ของระบบผู้เชี่ยวชาญทำได้ในวงแคบ เพราะถูกจำกัด ทั้งในแง่ความเร็วของการทำงาน และขนาดของฐานความรู้ที่จำกัดอยู่ในแต่ละระบบเท่านั้น การขจัดข้อจำกัดเหล่านี้อาจทำได้โดยสร้างระบบผู้เชี่ยวชาญที่สามารถทำงานในลักษณะ Cooperative Work กล่าวคือ ในการแก้ปัญหาใดปัญหาหนึ่ง เราอาจแบ่งงานของระบบผู้เชี่ยวชาญไปทำงานบน Processor หรือคอมพิวเตอร์แต่ละตัว เมื่อทำงานเสร็จจึงส่งคำตอบหรือผลลัพธ์มาประกอบกัน เพื่อสร้างเป็นคำตอบรวมของปัญหาที่ต้องการได้

นอกจากการแบ่งงานไปยังแต่ละ Processor ระบบผู้เชี่ยวชาญแต่ละตัวจะแก้ปัญหาที่อยู่ในกรอบของฐานความรู้ของตนเอง เมื่อพบปัญหาใดที่อยู่นอกฐานความรู้ของตน ก็สามารถสอบถามไประบบผู้เชี่ยวชาญอื่นซึ่งแก้ไขปัญหานั้นได้ ดังนั้นฐานความรู้รวม ของทั้งระบบจะสามารถขยายออกไปได้ไม่จำกัด

บทความนี้จะเสนอการออกแบบระบบผู้เชี่ยวชาญแบบกระจายบนเครือข่ายคอมพิวเตอร์ ตามแนวทาง Cooperating Distributed Expert System ที่ได้กล่าวมา ระบบผู้เชี่ยวชาญที่ทำงานอยู่บนเครื่องคอมพิวเตอร์ต่าง ๆ ในระบบเครือข่ายจะสามารถแก้ไขปัญหาใดปัญหาหนึ่งร่วมกัน โดยระบบผู้เชี่ยวชาญหนึ่งสามารถขอให้ระบบผู้เชี่ยวชาญอีกระบบหนึ่งช่วยแก้ไขปัญหาให้ในลักษณะ Client-to-Server Request [2]

2. โครงสร้างฐานข้อมูล Knowledge-Based Predicates

การเก็บฐานความรู้ (Knowledge Base) สามารถทำได้หลายรูปแบบ เช่น Rules, Semantic Network, Predicates เป็นต้น ในบรรดารูปแบบการเก็บเหล่านี้ Predicate นับเป็นวิธีการหนึ่งที่ง่ายต่อการทำ Reasoning ของ Inference Engine [1, 3, 6] ตัว Predicate จะถูกกำหนดในรูปของ Horn's Clause ดังนี้

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

โดยที่ H คือ Head ของ clause และ $B_1 \wedge B_2 \wedge ... \wedge B_n$ คือ Body ของ Clause Head และ Body อาจจะประกอบด้วย Variable ซึ่งจะใช้อักษรตัวใหญ่ และค่าคงที่ต่างๆ เช่น String , เลขจำนวนเต็ม, Structure, Linked List เป็นต้น ตัวอย่าง เช่น

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

Clause นี้ จะสามารถอ่านได้ 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 Reasoning สำหรับ 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) ← p(X, Z) ∧ p(Z, Y) **(3)**

ถ้าต้องการพิสูจน์ว่า Goal gp(a, c) เป็นจริงหรือไม่ เราจะใช้วิธีตั้งสมมติฐานว่า Goal gp(a, c) เป็นจริง จากสมมติฐานกับ Clause (3) เราสามารถใช้ขบวนการ Resolution เพื่อการพิสูจน์ โดยเริ่มจากการทำ Pattern matching หรือ Unification ระหว่าง Head ของ Clause และ Goal ที่ต้องการพิสูจน์ ซึ่งจะได้ว่า X=a, Y=c และจะสามารถสรุปได้ว่า Goal ต่อไปนี้จะต้องเป็นจริงด้วย (ตามหลักของ Monus Ponen)

p(a, Z) ∧ 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 ระหว่างชื่อของ Predicate และ Arity (จำนวน Arguments) ของ Goal กับ Head ของ Clause และทำ Unification ระหว่าง Arguments ของ Goal และ Head ทีละตัวตามลำดับ ถ้าส่วนใดส่วนหนึ่งไม่สอดคล้องกัน จะถือ ว่า Unification กระทำไม่สำเร็จ ตัวอย่างเช่น

Goal : **gp(a, c)**

Clause : **gp(X, Y) ← p(X, Z) ∧ p(Z, Y)**

Goal และ Head ของ Clause ข้างบนจะสามารถ Unify กันได้ ผลลัพธ์การ Unify จะทำให้เกิดการแทนค่ากันระหว่าง Variables ใน Clause และ Argument ของ Goal ในตัวอย่างข้างบน จะได้การแทนค่า (Variable Substitution) ดังนี้

{X=a, Y=c}

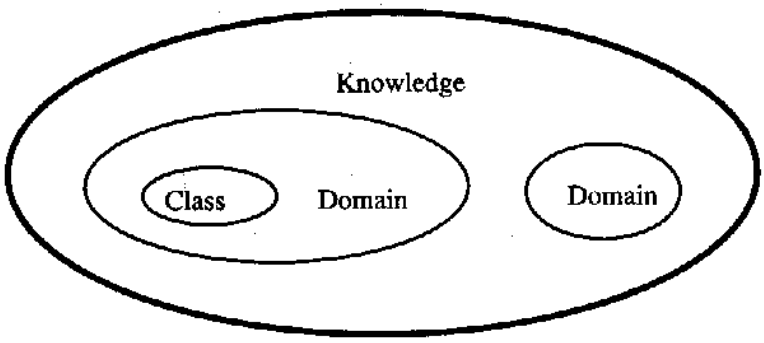
ข้อสังเกต ถ้า Variable A มีค่า (Bound) X จะมีค่าเท่ากับค่าใน A หรือถ้า A ยังไม่มีค่า (Unbound) X ก็จะยังไม่มีค่าเช่นกัน อนึ่ง Variable Z ใน Clause ยังไม่มีค่า (Unbound) เนื่องจากยังไม่มีการแทนค่าเกิดขึ้น ค่าของ Z จะเกิดขึ้นได้ ถ้าหากได้มีการแก้ปัญหาของ Subgoals ใน Body ของ Clause ต่อไป

จากนั้นเราสามารถใช้ ขบวนการ Resolution เพื่อพิสูจน์ Subgoal p(a, Z) และ p(Z, c) ต่อไป สำหรับ Subgoal p(a, Z) จะสามารถ Unify กับ Head ของ Clause (1) จึงสรุปได้ว่า Subgoal นี้เป็นจริง ผลลัพท์ก็คือ Substitution Z=b และ Subgoal p(Z, c) ที่จะต้องพิสูจน์ต่อไป จะถูกเปลี่ยนเป็น p(b, c) ตาม Substitution นี้ เมื่อทำการพิสูจน์ต่อไป Subgoal p(b, c) สามารถ Unify กับ Head ของ Clause (2) จึงสามารถสรุปได้ว่า Subgoal นี้เป็นจริงเช่นกัน เมื่อ Subgoal ทั้งสองสามารถพิสูจน์ได้ว่าเป็นจริง จึงสรุปได้ว่า gp(a, c) เป็นจริงตามสมมติฐาน

4. Hybrid Knowledge Model

ในระบบฐานความรู้แบบกระจาย (Distributed Knowledge Base) Predicate จะต้องกระจายเก็บอยู่ตาม Host ต่าง ๆ ในระบบเครือข่ายคอมพิวเตอร์ Predicate เหล่านี้ ควรจะถูกจัดเก็บในรูปแบบ ที่สามารถเชื่อมโยงระหว่างฐานความรู้ในแต่ละ Host ได้ รวมทั้งมีรูปแบบใกล้เคียงกับความรู้ (Real World Knowledge) ที่มนุษย์เป็นผู้เรียนรู้มา ด้วยเหตุที่มนุษย์มักจะมองสิ่งต่าง ๆ รอบข้างเป็นวัตถุ (Objects) ทำให้ความรู้ต่าง ๆ จะมีลักษณะของความสัมพันธ์ระหว่างวัตถุ การนำหลักการ Object-Oriented Data Model มาผสมผสานกับการเก็บฐานความรู้ในรูป Predicate หรือ Logic-based จะเป็นการช่วยให้มนุษย์สามารถถ่ายทอดความรู้ลงในฐานข้อมูลได้ง่ายขึ้น โดยที่ความสามารถในการ Inference เพื่อแก้ไขปัญหาจากฐานความรู้ยังคงอยู่

การผสมผสานระหว่าง Logic-based และ Object-Oriented Knowledge Model ก่อให้เกิด Hybrid Knowledge Model ซึ่งจะอธิบายพฤติกรรมของ Object ในรูปของ Predicate กล่าวคือ Predicates ใน Class ของ Object จะใช้แสดงหรือเป็นตัวแทนของ Methods และ Fact ที่เกี่ยวข้องกับ Object ภายใน Class นั้น นอกเหนือจากการใช้ Predicate ในการอธิบายพฤติกรรมของ Object เราควรจะมีการจัดกลุ่มของ Class ที่เกี่ยวข้อง ประกอบขึ้นเป็น Domain ซึ่งทำให้สะดวกต่อการจัดเก็บและเรียกใช้ใน Distributed Environment กล่าวคือ ฐานความรู้ของระบบควรจะแยกออกเป็นกลุ่มหรือ Domain แต่ละ Domain สามารถแยกเก็บอยู่ใน Host ต่างๆ บนระบบเครือข่าย ดังนั้นเราอาจมอง Domain หนึ่ง ว่าเป็นฐานความรู้เฉพาะซึ่งประกอบด้วย Classes ของ Objects ที่เกี่ยวข้อง



การจัดเก็บฐานความรู้ในลักษณะนี้ ทำให้สามารถแบ่งฐานความรู้เป็นส่วน ๆ และแยกจัดเก็บส่วนของฐานความรู้ใน Host ต่าง ๆ ตามแหล่งกำเนิดและการเรียกใช้ของฐานความรู้นั้น ๆ

5. Universal Predicate Locator (UPL)

เนื่องจากฐานความรู้ถูกจัดเก็บในรูป Predicates ซึ่งกระจายเก็บอยู่ตาม Host ต่าง ๆ เราจำเป็นต้องมีวิธีการที่ใช้เรียก Predicates เหล่านี้ Universal Predicate Locator จะเป็นวิธีการเรียก Predicate ที่เก็บอยู่ในฐานความรู้ในระบบเครือข่าย UPL จะประกอบด้วยชื่อ Predicate, Arity (จำนวน Arguments ของ Predicate), Class, Domain และ Host Name ที่เก็บ Predicate นั้น ดังรูปแบบต่อไปนี้

Predicate/Arity/Class/Domain/Hostname

ในส่วนของ Class, Domain และ Host Name ของ UPL ถ้าหากไม่ทราบ ก็สามารถละไว้ได้ ตัวอย่าง เช่น p/2///, g/3/c// เป็นต้น

6. Predicate Name Server (PNS)

โดยที่ระบบผู้เชี่ยวชาญแต่ละตัวจะไม่ทราบว่าระบบผู้เชี่ยวชาญบนเครื่องคอมพิวเตอร์ใดประกอบด้วยฐานความรู้ที่มี Predicate ที่ตนต้องการทำ Inference ดังนั้นจึงจำเป็นต้องมีคนกลางที่เก็บข้อมูล UPL ของ Predicates ต่าง ๆ ที่มีอยู่ในระบบเครือข่าย รวมทั้งทำหน้าที่ค้นหา UPL สำหรับ Predicate ที่ ระบบผู้เชี่ยวชาญต้องการ ระบบผู้เชี่ยวชาญอาจส่ง Query เพื่อให้ค้นหา UPL โดยกำหนดบางส่วนของ UPL เช่น ไม่กำหนด Class, Domain หรือ Host Name เป็นต้น PNS เมื่อได้รับ Query จะค้นหา UPL ของ Predicate ที่ต้องการ จากฐานข้อมูลของตน ถ้าค้นพบ จะส่งค่า UPL กลับไปยัง Client นั้น แต่ถ้าไม่พบ จะส่งค่า Null กลับไปเพื่อให้ Client ทราบ และทำการค้นหากับ PNS อื่น ๆ ต่อไป

การติดต่อระหว่าง Client และ PNS ใช้หลักการของ Remote Procedure Call (RPC) ซึ่งเป็นวิธีการเรียกใช้บริการจาก Host หนึ่งไปอีก Host หนึ่ง ในรูปแบบของ Client-to-Server บนระบบเครือข่ายคอมพิวเตอร์ [2]

PNS นอกจากให้บริการในการค้นหา UPL ของ Predicate แล้วยังสามารถรับ Request จาก Client เพื่อบันทึก (Register) หรือแก้ไข UPL ของ Predicate ที่ต้องการบนฐานข้อมูลของ PNS อีกด้วย

7. Extended Horn's Clause

จากการผสมผสาน Object Oriented Concept อันได้แก่ Domain และ Class เข้ากับ Logic-Based Predicate รูปแบบของ Horn's Clause จะต้องเปลี่ยนแปลงไป เพื่อรองรับ Object Oriented Concept เหล่านี้ ส่วนเปลี่ยนแปลงต่อ Horn's Clause ได้แก่

1. Predicate Name ในส่วน Head ของ Clause ควรจะต้องมีชื่อ Domain และ Class ของ Predicate นั้นกำกับอยู่ ดังนี้

Predicate/Class/Domain (Argument List)

2. Predicate Names ในส่วน Body ของ Clause ควรจะต้องกำกับด้วย Domain, Class และ Host Name ซึ่งหมายถึงการใช้ UPL เพื่อเรียก Predicate นั่นเอง ในกรณีที่ไม่กำหนด UPL ที่ครบถ้วน เช่น ไม่กำหนด Host Name ระบบก็สามารถส่ง Query ไปยัง PNS เพื่อให้ค้นหา UPL ที่สมบูรณ์ อันประกอบด้วย Predicate/Arity/Class/Domain/Hostname

จากการเพิ่ม Concepts ของ Domain และ Class เข้ากับ Horn's Clause ผลลัพธ์ที่ได้คือ Extended Horn's Clause ซึ่งจะมีลักษณะดังตัวอย่างต่อไปนี้

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

กรณีตัวอย่างข้างบน gp และ p จะเป็น Predicate ที่อยู่ใน Class ของ "person" และ Domain "family"

8. Searching Rules สำหรับการค้นหา UPL ของ Predicates

การแก้ไขปัญหาด้วย Resolution จำเป็นจะต้องทำการค้นหา UPL ของ Predicates ในส่วน Body ของ Clause เพื่อทราบ Host Name ที่เก็บ Clause ของ Predicate เหล่านั้น เมื่อทราบ Host Name แล้วจึงจะส่ง Request ไปยัง Host นั้น เพื่อขอให้ทำ Inference บน Predicate ที่ต้องการ การค้นหา UPL ของ Predicates จึงเป็นงานที่จะต้องกระทำบ่อย ๆ ในระหว่างการทำ Inference Process บน Clauses ต่าง ๆ

การค้นหา UPL ของ Predicate ได้อย่างมีประสิทธิภาพ จึงเป็นปัจจัยอันหนึ่ง ที่จะทำให้การทำงานของระบบโดยรวมเป็นไปได้อย่างรวดเร็ว และมีประสิทธิภาพ อนึ่ง UPL ของ Predicates ต่าง ๆ อาจจัดเก็บแยกไปตาม PNS ต่าง ๆ ซึ่ง PNS แต่ละตัวจะทำหน้าที่ค้นหา UPL และบำรุงรักษาฐานข้อมูล UPL ของตนเท่านั้น

การสอบถามไปยัง PNS โดย Client Process เอง ควรจะมีการกำหนดลำดับ ว่าจะสอบถาม PNS ตัวใดตามลำดับ Client Process ควรจะเริ่มค้นหา UPL ของ Predicate ในฐานข้อมูลของ Local Host ก่อน เพราะ Predicate ส่วนใหญ่อาจอยู่ที่ Local Host หลังจากนั้นจึงสอบถาม PNS ตามลำดับที่ได้กำหนดไว้ จนกว่าจะได้รับคำตอบ คือ UPL ที่ต้องการ

เนื่องจากการสอบถามไปยัง Remote PNS จะใช้เวลาค่อนข้างมาก ถ้าเราต้องการค้นหา UPL บน Local Database เท่านั้น เราอาจใช้เครื่องหมาย * แทน Host Name บน Query UPL ก็ได้ เช่น gp/2///* หรือถ้าต้องการค้นหา UPL เฉพาะบน PNS แรกที่พบเท่านั้น เราอาจใช้เครื่องหมาย + แทน Host Name ใน Query UPL ก็ได้ เช่น gp/2///+

9. การทำ Inference ในลักษณะ Distributed Processing

Host แต่ละเครื่อง ซึ่งทำหน้าที่เป็น Knowledge Server (KNS) จะมีฐานความรู้ของตนเอง KNS จะทำงานในลักษณะ Daemon Process ที่รอรับ Request เพื่อให้ทำ Inference บน Goal ที่ต้องการ และส่งคำตอบกลับไป หน้าที่อีกอย่างหนึ่งของ KNS ก็คือจะต้องส่งรายชื่อ UPL ของ Predicate ในฐานความรู้ของตนไปบันทึกบน PNS ซึ่งดูแลกำกับตัวเองอยู่ ทั้งนี้เพื่อให้ KNS อื่น สามารถสอบถาม UPL ของ Predicate ที่ตนมีอยู่ จาก PNS นั้นได้

การทำ Inference บน Predicates ที่กระจายอยู่ตามฐานความรู้ของ KNS ต่างๆ ในระบบเครือข่าย นับเป็นสิ่งที่ยุ่งยากต่อการออกแบบเป็นอย่างยิ่ง ทั้งนี้เพราะ การทำ Inference โดยวิธี Resolution อาจได้ผลลัพธ์เป็นหลายคำตอบ แต่ละคำตอบจะถูกนำไปใช้ในการทำ Inference ของ Goals ที่เหลือ คำตอบเหล่านี้จะถูกสร้างจากการ Inference ของ KNS หนึ่ง และถูกนำไปใช้โดยอีก KNS หนึ่ง เพื่อทำ Inference ต่อไปด้วยการ

ทำงานแบบอิสระของ KNS แต่ละตัวเราจะต้องมีวิธีการที่จะประสานการทำงาน (Synchronization) ระหว่าง KNS ซึ่งเป็น Producer และ Consumer ของคำตอบเหล่านี้

10. ขั้นตอนการทำงานของ KNS ในการทำ Inference

ขั้นตอนการทำงานของ KNS ที่มีการประสานการทำ Inference ร่วมกับ KNS อื่น สามารถสรุปได้ดัง Algorithm ต่อไปนี้

1. รอรับ Request จาก KNS อื่น
2. เมื่อได้รับ Request ให้สร้าง Service Process เพื่อที่จะทำ Inference บน Predicate และ Argument ที่ส่งมา หลังจากนั้น จะกลับไปรอรับ Request ดังเดิมในข้อ 1 ส่วน Service Process จะเช็คดูว่า Predicate นั้นอยู่ในฐานความรู้ของตนหรือไม่ ถ้าไม่อยู่ให้ส่งคำตอบ Null กลับไปเพื่อแสดงว่า ไม่สามารถหาคำตอบได้
3. ถ้า predicate นั้นอยู่ในฐานความรู้ของตนให้กระทำขั้นตอนต่อไปนี้
   - ทำ Unification ระหว่าง Predicate นั้น และ Head ของ Clause ที่ Match กัน
   - ในแต่ละ Clause ที่ Unify ได้กับ Predicate สร้าง Clause Process ที่จะทำการ Inference ต่อไปกับ Goals ที่อยู่ใน Body ของ Clause นั้น ๆ ตามลำดับ เมื่อได้ชุดของคำตอบทั้งหมดของ Clause (หรือ Null ถ้าไม่มีคำตอบ) จึงส่งชุดของคำตอบเหล่านั้นกลับไปยัง Service Process หลังจากนั้น จึงหยุดทำงานของ Clause Process
4. Service Process เมื่อได้รับชุดของคำตอบจาก Clause Processes ที่ตนสร้างไว้ทั้งหมดแล้วจึงรวบรวมคำตอบทั้งหมดที่ได้สร้างเป็นชุดของคำตอบรวมของ Predicate แล้วส่งชุดของคำตอบเหล่านั้นกลับไปยัง Client KNS หลังจากนั้นจึงหยุดทำงานตนเอง

11. Clause Representation

การทำ Inference กับ Clause นับเป็นขั้นตอนหลักในการทำงานของ KNS รูปแบบและวิธีการเก็บ Clause ในฐานความรู้ จะมีส่วนสำคัญที่เอื้ออำนวยให้การทำ Inference เป็นไปได้อย่างรวดเร็วและมีประสิทธิภาพ เนื่องจาก Head และ Body ของ Clause มีส่วนประกอบหลักก็คือ Predicate Name และ Argument List ดังนั้น เราอาจใช้ Template ที่เก็บรายละเอียดของทั้งสองส่วน ดังโครงสร้างข้อมูลต่อไปนี้

Predicate Template

Predicate name	Arity	Argument-1 Template	Argument-2 Template

Argument Template

Tag	Data

โดยที่

Tag เป็นตัวระบุชนิดของ Data ใน Field ถัดไป เช่นเป็น Variable Number เป็นต้น Structure หรือ Pointer ไปยังข้อมูลคงที่ต่าง ๆ เช่น String, Number, และ List เป็นต้น

Data เป็นส่วนเก็บข้อมูลตามชนิดที่ระบุด้วย Tag

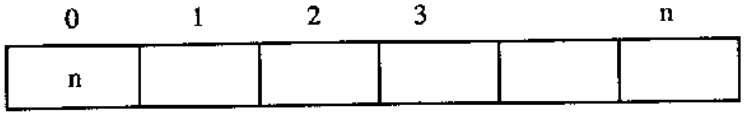
12. Environment ภายใน Clause

จากขั้นตอนการทำงานของ KNS ข้างต้น Service Process จะถูกสร้างโดย KNS เพื่อเป็นผู้กระทำ Unification ระหว่าง Goal และ Head ของ Clause ถ้า Unification กระทำได้สำเร็จ (Success) จะเกิดการแทนค่า ของ Variables ระหว่าง Arguments ที่ส่งมากับ Goal และ Variables ต่าง ๆ ภายใน Clause นั้น ค่า Clause Variables ทั้งหมดนี้เรียกโดยรวมว่า Environment ของ Clause และจะถูกเก็บอยู่ใน Local Frame ซึ่งถูกสร้างขึ้น หลังจากที่มีการกระทำ Unification สำเร็จแล้ว

สำหรับ Arguments ของ Goal ที่ส่งมายัง KNS จะถูกเก็บอยู่ภายในเนื้อที่อีกส่วนหนึ่ง ซึ่งจะเรียกว่า Incoming Frame การเก็บ Arguments ภายใน Incoming Frame นี้ก็เพื่อใช้เป็นรูปแบบ (Template) ในการสร้างคำตอบ (Return Frame) เพื่อจัดส่งกลับไปยัง Client KNS หลังจากที่ Clause Process ได้ทำ Inference ใน Clause เสร็จเรียบร้อยแล้ว โปรดสังเกตการส่งค่า Variables ระหว่าง Client และ Server KNS จะกระทำได้ โดยการส่งค่าผ่านทาง Arguments เท่านั้น

สำหรับ Client KNS เมื่อต้องการส่ง Goal และ Arguments เพื่อไปแก้ปัญหาที่ Server KNS ก็จะต้องสร้าง Outgoing Template เพื่อเป็น Template ที่แสดงว่าได้ส่งค่า Clause Variables ใดไปใน Incoming Frame เมื่อ Client KNS ได้รับ Return Frames จาก Server KNS ก็สามารถนำค่า Variables ใน Return frame มาแทนค่าใน Clause Variables ได้ถูกต้อง ทั้งนี้โดยดูจาก Variable Mapping ที่เก็บอยู่ใน Outgoing Frame

18. โครงสร้างของ Local Frame



Local Frame จะประกอบด้วย Variable slot จำนวน n+1 ช่อง โดยที่

Slot 0 จะเก็บจำนวน Clause Variables ใน Local Frame คือค่า n

Slot 1 to n จะเก็บค่าของ Clause Variables แต่ละตัวตามลำดับ

Slot ใน Local Frame จะประกอบด้วยส่วนต่าง ๆ ดังต่อไปนี้

1. Tag เป็นตัวบอกชนิดของข้อมูลที่เก็บอยู่ใน Slot นั้น ชนิดของข้อมูลสามารถแยกออกเป็น 2 ประเภท คือ

   ก) ค่าคงที่ ซึ่งแยกออกได้ตามชนิดของข้อมูล ดังนี้

   - เลขจำนวนเต็ม
   - Pointer ไปยัง String ของตัวอักษร
   - Pointer ไปยัง Structure
   - Pointer ไปยัง Linked List

   ข) Variable ซึ่งอาจจะมีค่าได้เป็น 2 สถานะ คือ

   - Bound ได้แก่สถานะที่ Variable นี้ได้ถูกกำหนดให้มีค่าเช่นเดียวกับ Variable อื่นใน Local Frame เดียวกัน
   - Unbound ได้แก่สถานะที่ Variable ยังไม่ถูกกำหนดให้มีค่าใดเลย

2. ข้อมูล ซึ่งแยกตามชนิดที่ระบุ โดย Tag ดังนี้

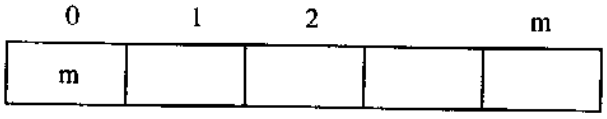
a) ค่าคงที่ต่าง ๆ (ดังระบุข้างต้น) เช่น เลขจำนวนเต็ม, Pointer เป็นต้น

b) Variable Slot

- กรณี Bound ข้อมูลจะเป็นค่า Slot Number ของ Variable อื่นใน Local Frame ที่ Variable นี้มีค่าเช่นเดียวกันกับ Variable นั้น
- กรณี Unbound ข้อมูลจะเป็นค่า -1

เนื่องจาก Local Frame จะเป็นที่เก็บ Environment ภายใน Clause ข้อมูลใน Local Frame จะต้องสมบูรณ์ภายในตัวเอง คือมีลักษณะที่เป็น Closed Environment [7] กล่าวคือ การอ้างอิงไปยัง Variable อื่นนอก Local Frame จะไม่สามารถกระทำได้ การที่ Local Frame มีลักษณะเป็น Closed Environment จะทำให้การ Inference Clause มีความอิสระ ไม่ต้องอ้างอิงข้อมูลนอกเหนือจากข้อมูลภายใน Local Frame ซึ่งจะเหมาะกับการทำงานในลักษณะ Distributed Processing เพราะไม่ต้องอ้างอิงไปยังข้อมูลที่อยู่คนละ KNS

14. โครงสร้างของ Incoming Frame



Incoming Frame จะประกอบด้วย Argument และ Variable Slots จำนวน m+1 ช่อง โดยที่

Slot 0 จะเก็บจำนวน Slot ใน Frame ในที่นี้คือ m ซึ่งเท่ากับ a+p โดย

a = จำนวน Argument (Arity)

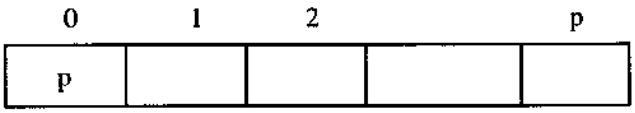
p = จำนวน Variables ใน Frame นี้

Slot 1 to a จะเก็บ Tag และค่าของข้อมูลของ Argument แต่ละตัวตามลำดับ (โครงสร้างเช่นเดียวกับ Slot ใน Local Frame)

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

ข้อมูลใน Incoming Frame จะมีลักษณะ Closed หรือสมบูรณ์ในตัวเอง เช่นเดียวกับข้อมูลใน Local Frame

15. โครงสร้างของ Outgoing Template



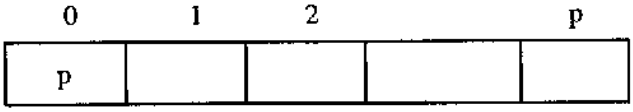
Outgoing Template จะประกอบด้วย Slot จำนวน p ช่อง โดยที่

Slot 0 จะเก็บจ

Variable ใน Incoming Frame ที่ส่งไปยัง Server KNS

Slot 1 to p จะเก็บค่า Variable Number ของ Variable ใน Local Frame ที่ถูก ส่งค่าไปใน Variable ต่าง ๆ ภายใน Incoming Frame

16. โครงสร้างของ Return Frame



Return frame จะประกอบ Variable slot จำนวน p ช่อง โดยที่

Slot 0 จะเก็บจำนวน Variable ใน Frame ในที่นี้คือ p ซึ่งจะมีค่าเท่า กับจำนวน Variable ใน Incoming Frame

Slot 1 to p จะเก็บค่าของ Variable แต่ละตัวตามลำดับ

อนึ่ง Return Frame จะมีลักษณะ Closed เช่นเดียวกับ Local Frame

17. การส่งผ่านค่าของ Variables ระหว่าง Client และ Server Environment

จากการส่งผ่านค่าของ Variables ระหว่าง Client และ Server KNS ดังได้กล่าวมาแล้วข้างต้นอาจพอสรุปได้ว่า การส่งผ่านค่าระหว่าง Client และ Server จะเกิดขึ้นได้ใน 2 ช่วง คือ

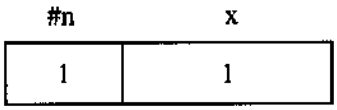
1. เมื่อ Client KNS ต้องการส่ง Request ไปยัง Server KNS เพื่อให้แก้ปัญหาใดปัญหาหนึ่ง Client KNS จะสร้าง Incoming Frame ซึ่งประกอบด้วย Arguments และค่า Variable ที่เกี่ยวข้องกับ Arguments นั้น เพื่อส่งไปพร้อมกับ Request นั้น ในขณะเดียวกันก็จะสร้าง Outgoing Template เพื่อระบุว่า Variables ที่ส่งไปใน Incoming Frame เป็น Variable ใดบ้าง Outgoing Template นี้จะมีบทบาทในการรับผลลัพธ์จาก Server ไปสู่ Client เมื่อได้รับคำตอบ คือ Return Frame กลับมาแล้ว ส่วน Server KNS เมื่อได้รับ Request พร้อม Goal และ Incoming Frame ก็จะสร้าง Service Process ที่จะทำการ Unification ระหว่าง Goal/Arguments และ Head ของ Clause ถ้า Unification กระทำได้สำเร็จ Service Process ก็จะสร้าง Local Frame และ Incoming Frame เฉพาะของ Clause นั้นเท่านั้น เนื่องจากการทำ Unification อาจมีการแทนค่า ระหว่าง Variables ใน Incoming และ Local Frame และเพื่อทำให้ Local Frame มีลักษณะเป็น Closed Environment Variables ใด ใน Incoming Frame ถ้าถูกกำหนดให้มีค่าเท่ากับ Variable ใน Local Frame ค่าของ Variables นั้น ใน Incoming Frame ก็จะมีค่าเป็น -v โดยที่ v คือ Variable Number ของ Variableใน Local Frame ที่ถูกทำให้มีค่าเช่นเดียวกัน สำหรับการที่มีค่าเป็นลบก็เพื่อเป็นการแสดงถึงความเป็น Variable Number ของ Local Frame ซึ่งต่างกับ Variable Number ใน Incoming Frame ซึ่งจะมีค่าบวก ค่า v ใน Incoming Frame จะถูกใช้ในการสร้าง Return Frame เมื่อ Clause Process ได้ทำการ Inference สำเร็จเรียบร้อยแล้ว
2. หลังจาก Clause Process ได้ทำ Inference บน Clause ของตนจนสำเร็จแล้ว ผลลัพธ์ที่ได้ คือ Set ของคำตอบทั้งหมดของ Clause นั้น แต่ละคำตอบจะประกอบด้วย Local Frame ที่มีค่าของ Variable แตกต่างกันตามคำตอบนั้น ๆ

หน้าที่ต่อจากนั้นของ Clause Process ก็คือการสร้าง Return Frame สำหรับแต่ละคำตอบของ Clause โดยการ Unify ค่า Variable ต่าง ๆ ใน Incoming Frame กับ Variable ใน Local Frame ซึ่งจะเห็นได้ว่ามีการส่งผ่านค่าจาก Local Frame กลับไปยัง Caller โดยผ่าน Return Frame นั่นเอง เมื่อได้รับ Return Frame ของทุกคำตอบเรียบร้อยแล้ว Clause Process จะส่งชุดของ Return Frame เหล่านี้กลับไปยัง Service Process ซึ่งจะทำการจัดเรียงชุดของคำตอบให้เป็นไปตามลำดับของ Clauses ที่เป็นผู้ผลิตชุดคำตอบเหล่านั้น ลำดับของ Clause หรือที่เรียกว่า Textual Order สามารถกำหนดได้ด้วย Sequence Number ที่ Clause ส่งมาพร้อมกับชุดของคำตอบ Sequence Numbers ของ Clauses จะถูกใช้ในการจัดเรียงคำตอบโดย Service Process เป็นผู้กระทำ หลังจากนั้นจึงจะส่งคำตอบที่ได้จัดเรียงเรียบร้อยแล้วกลับไปยัง Client Process

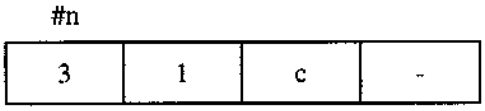
สำหรับ Client Process เมื่อได้รับชุดของ Return Frame ก็จะทำการแทนค่าของ Variables ใน Return Frame กลับไปยัง Local Frame (โดยใช้ Outgoing Template ช่วยในการแทนค่า) Local Frame ใหม่ที่ได้จากการแทนค่าของแต่ละคำตอบ จะถูกนำไปใช้ในการทำ Inference ของ Goal ถัดไปใน Body ของ Clause

ภาพต่อไปนี้เป็นการแสดงตัวอย่างการสร้าง Frame ต่าง ๆ และการส่งค่า Variables ระหว่าง Client และ Server KNS เพื่อทำการแก้ปัญหา gp(X, c)

Goal : gp(X, c)



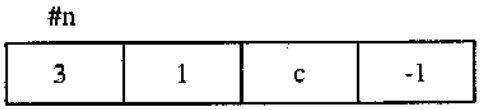
Outgoing Template



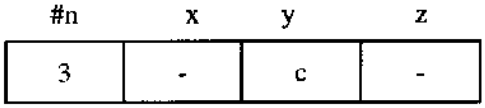
Incoming Frame

gp(X, Y) ← p(X, Z) ∧ p(Z, Y) (1)

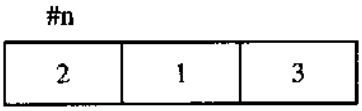
Incoming frame is unified with Head of the clause (1)



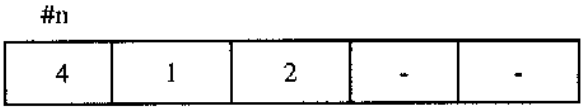
Incoming Frame



Local Frame

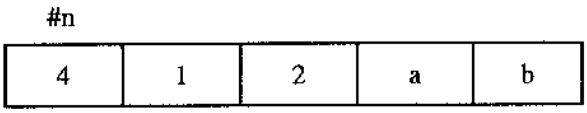


Outgoing Template

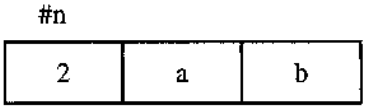


Incoming Frame

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

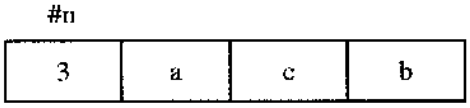


Incoming Frame



Return Frame

Return to Clause (1)



Local Frame

.

.

.

หมายเหตุ : - ในช่อง Variable Slot หมายถึงค่า Unbound

18. บทสรุป

การออกแบบระบบผู้เชี่ยวชาญแบบกระจายบนเครือข่ายคอมพิวเตอร์ ตามที่เสนอในรายงานฉบับนี้มีจุดประสงค์ที่ได้มาซึ่งต้นแบบของระบบผู้เชี่ยวชาญที่ทำงานประสานกันบนเครือข่ายคอมพิวเตอร์ สำหรับโปรแกรมต้นแบบของระบบได้ถูกพัฒนาบนระบบเครือข่ายของ HP Workstation ที่โครงการศูนย์คอมพิวเตอร์ของคณะสถิติประยุกต์ โปรแกรมต้นแบบนี้จะใช้ตรวจสอบความถูกต้อง และรายละเอียดของการออกแบบระบบ ตลอดจนทดสอบประสิทธิภาพ เพื่อเป็นแนวทางในการปรับปรุงระบบให้สมบูรณ์ยิ่งขึ้นต่อไป

19. เอกสารและหนังสืออ้างอิง

1. Intelligent Systems for Business โดย Fatemeh Zahedi, Wadsworth MIS series, 1993.
2. Power Programming with RPC โดย John Bloamer, O'Reilly & Associates, 1992.
3. Expert Systems for Business: Concepts and Applications, โดย D.V. Pigford และ Greg Baur, Boyd & Fraser Publishing Company, 1995.
4. Logic Programming: Prolog and Stream Parallel languages โดย J.D. Newmarch, Prentice Hall, 1990.
5. Prolog++: The Power of Object-Oriented and Logic Programming โดย 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.