CL : ภาษาสำหรับการสั่งงานคอมพิวเตอร์ด้วยเวลาและเหตุการณ์

CL : A LANGUAGE FOR PROGRAMMING WITH TIME AND EVENTS

สุพัณณดา โชติพันธ์

SUPANNADA CHOTIPANT

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชาวิศวกรรมคอมพิวเตอร์

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ. 2547

ISBN 974-15-1321-5

CL : ภาษาสำหรับการสั่งงานคอมพิวเตอร์ด้วยเวลาและเหตุการณ์

CL : A LANGUAGE FOR PROGRAMMING WITH TIME AND EVENTS

สุพัณณดา โชติพันธ์
SUPANNADA CHOTIPANT

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์
บัณฑิตวิทยาลัย
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ. 2547

ISBN 974-15-1321-6

CL : A LANGUAGE FOR PROGRAMMING WITH TIME AND EVENTS

SUPANNADA CHOTIPANT

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

ISBN 974-15-1321-6

หัวข้อวิทยานิพนธ์	CL : ภาษาสำหรับการสั่งงานคอมพิวเตอร์ด้วยเวลาและเหตุการณ์
นักศึกษา	นางสาว สุพัณณดา โชติพันธ์
รหัสนักศึกษา	44061624
ปริญญา	วิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชา	วิศวกรรมคอมพิวเตอร์
พ.ศ.	2547
อาจารย์ผู้ควบคุมวิทยานิพนธ์	ดร.วิศิษฏ์ หิรัญกิตติ

บทคัดย่อ

วิทยานิพนธ์ฉบับนี้นำเสนอการออกแบบและพัฒนาภาษาชั้นสูงสำหรับการสื่อสารของเอเจนต์ โดยให้ชื่อว่า "Communication Language" หรือเรียกสั้นๆ ว่า "ภาษา CL" ภาษา CL สร้างมาจากแนวความคิดใหม่เพื่อให้เป็นภาษาสำหรับสร้างเอเจนต์สำหรับการสื่อสารและขณะเดียวกันก็เป็นภาษาสำหรับสั่งงานคอมพิวเตอร์(เอเจนต์)ด้วยเวลาและเหตุการณ์ ในความสามารถเรื่องการสื่อสาร ผู้เขียนโปรแกรม CL สามารถสร้างเอเจนต์ให้สื่อสารกัน ตามบทสนทนา ตามโปรโตคอลประเภทต่างๆ และตามช่วงเวลาที่ต้องการผ่านทาง Communication Scripts การสื่อสารของแต่ละเอเจนต์เกิดจาก Inference ภายในที่ทำงานตามวัฏจักร Observe – Fire Comm. Script – Actการพัฒนาภาษาในส่วนการสื่อสารนี้ใช้หลักการของ Meta-reasoning สำหรับความสามารถในเรื่องการสั่งงานคอมพิวเตอร์(เอเจนต์)ด้วยเวลาและเหตุการณ์ ผู้เขียนโปรแกรม CL สามารถระบุการทำคำสั่งในโปรแกรมของเอเจนต์ให้เป็นตามกำหนดเวลา นอกจากนี้ยังสามารถระบุให้ทำคำสั่งในโปรแกรมเมื่อมีเหตุการณ์หนึ่งๆเกิดขึ้นในรูปแบบ Event -> Action การออกแบบและพัฒนาความสามารถดังกล่าวนี้ได้มาผลการศึกษาความหมายของเวลา ความสัมพันธ์ระหว่างเวลา การทำงานและการตอบสนองต่อเหตุการณ์ในชีวิตประจำวันของเรา

Thesis Title	CL : A Language for Programming with Time and Events
Student	Miss. Supannada Chotipant
Student ID.	44061624
Degree	Master of Engineering
Programme	Computer Engineering
Year	2004
Thesis Advisor	Dr. Visit Hirankitti

ABSTRACT

In this thesis we have proposed a design and development of a high-level language for agent communication. We named this language "Communication Language" or briefly "CL". CL is developed based on a novel idea to be a language for constructing communication agents and simultaneously to be a language with which a programmer can issue instructions governed by time and can program a computer (agent) to take an action to response to an event. For the CL's communication ability, a programmer can use CL to create an agent according to certain conversation scripts via certain protocols and at specific time using Communication Scripts. The agent communicates when its Inference operates in the observe–fire Comm. Script–act cycle. This CL's ability is built upon meta-reasoning. For the other ability in CL which supports the (agent) programming with time and events, the programmer can also issue any CL statement constrained by time and a CL statement conditioned by an event in the form of "Event -> Action." This latter ability is developed based on the result of our investigation on the meaning of time as well as some relationships among time, actions, and events in our everyday life.

กิตติกรรมประกาศ

วิทยานิพนธ์เล่มนี้สำเร็จได้ด้วยความกรุณาจากอาจารย์ที่ปรึกษา ดร.วิศิษฏ์ หิรัญกิตติ ที่ให้ความช่วยเหลือ ให้คำชี้แนะช่วยแก้ปัญหาตลอดจนให้ความรู้และประสบการณ์ที่ดีแก่ข้าพเจ้า บ่อยครั้งที่อาจารย์เหน็ดเหนื่อยและมีธุระมากมาย แต่ท่านก็ยังเสียสละเวลาเพื่อให้คำปรึกษาในการทำวิทยานิพนธ์เสมอ

ขอขอบพระคุณอาจารย์ทุกท่านที่ได้ประสิทธิ์ประสาทวิชาความรู้ และกรุณาชี้แนะและให้คำแนะนำในเรื่องต่างๆ ไม่ว่าจะเป็นเรื่องการเรียน การงาน หรือแม้แต่การใช้ชีวิต

ขอขอบคุณเพื่อนนักศึกษาที่ช่วยเหลือและเป็นกำลังใจให้เสมอมา

ขอขอบคุณบิดามารดาอันเป็นที่รักและเคารพยิ่ง ที่ให้กำเนิด อบรมเลี้ยงดู และคอยสนับสนุนข้าพเจ้าในทุกๆ ด้าน

สำหรับคุณความดีอันเกิดขึ้นจากวิทยานิพนธ์ฉบับนี้ ข้าพเจ้าขอมอบให้กับบิดามารดา ครูอาจารย์ และผู้ที่สนับสนุนและช่วยเหลือข้าพเจ้าทุกท่าน เพราะหากปราศจากท่านทั้งหลาย วิทยานิพนธ์ฉบับนี้ก็คงไม่สามารถสำเร็จลุล่วงได้

สุพัณณดา โชติพันธ์

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

ตารางที่ หน้า

สารบัญรูป

สารบัญรูป (ต่อ)

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ในปัจจุบัน มนุษย์สามารถทำกิจกรรมหลายๆ อย่างได้สะดวกและง่ายดายยิ่งขึ้น เนื่องด้วยการอาศัยโปรแกรมคอมพิวเตอร์ในรูปแบบต่างๆ ภาษาคอมพิวเตอร์ที่ใช้ในปัจจุบันมีมากมาย เช่น ภาษา Java, C, Visual C, C++ เป็นต้น ซึ่งเมื่อพิจารณาดูแล้ว จะพบว่าภาษาคอมพิวเตอร์ในปัจจุบันส่วนมากเหมาะสมสำหรับการสั่งงานเป็นชุดคำสั่งเท่านั้น ไม่เหมาะสำหรับการสั่งงานและการทำงานที่เกี่ยวข้องกับการสื่อสารซึ่งเป็นกิจกรรมที่มนุษย์ทำอยู่เป็นประจำ

ข้อจำกัดดังกล่าวจะนำไปสู่ความไม่ยืดหยุ่นของการเขียนและการทำงานของโปรแกรม ซึ่งนับเป็นปัญหาที่น่าสนใจและควรแก่การแก้ไขปัญหาหนึ่ง เนื่องจากมนุษย์มีกิจกรรมการทำงานที่หลากหลายนอกเหนือจากการทำงานในลักษณะชุดคำสั่งในภาษาคอมพิวเตอร์ในทั่วไป และหากสามารถปรับปรุงรูปแบบของภาษาให้มีความยืดหยุ่นและสามารถทำงานได้คล้ายคลึงกับการทำงานโดยทั่วไปของคนมากขึ้นได้ ย่อมทำให้ผู้เขียนสามารถเขียนโปรแกรมและสื่อความหมายของโปรแกรมได้ง่าย หลากหลาย และครอบคลุมมากยิ่งขึ้น

เพื่อแก้ไขปัญหาดังกล่าวข้างต้น งานวิจัยนี้จึงขอนำเสนอการออกแบบและพัฒนาภาษาคอมพิวเตอร์รูปแบบใหม่ ที่เรียกว่า "ภาษาเพื่อการสื่อสาร" ซึ่งเป็นภาษาคอมพิวเตอร์ที่สนับสนุนการทำงานตามเวลาและสามารถตอบสนองต่อเหตุการณ์ภายนอก ตลอดจนการพัฒนาเอเจนต์เพื่อการสื่อสาร

1.2 ความมุ่งหมายและวัตถุประสงค์ของการศึกษา

1. เพื่อพัฒนาภาษาคอมพิวเตอร์รูปแบบใหม่ที่มีสามารถทำงานตามเวลา และตอบสนองต่อเหตุการณ์ภายนอกได้
2. นำเอาความเข้าใจเกี่ยวกับธรรมชาติการสื่อสาร และการทำงานที่ต้องเกี่ยวข้องกับเวลามาเป็นหลักในการออกแบบภาษาคอมพิวเตอร์ที่สามารถทำงานเลียนแบบมนุษย์ได้มากขึ้น
3. เพื่อเพิ่มขีดความสามารถให้กับภาษาไพธอน (Python)
4. พัฒนาภาษาที่ใช้สร้างเอเจนต์เพื่อการสื่อสารที่ง่ายจ่อการใช้งาน เมื่อกำหนดสคริปต์สำหรับสื่อสารให้กับเอเจนต์ แต่ละเอเจนต์ก็จะสามารถทำการสื่อสารที่ซับซ้อนได้

1.3 สมมติฐานของการศึกษา

การพัฒนาภาษาคอมพิวเตอร์ชนิดใหม่ จะมีลักษณะดังนี้

1. อาศัยการศึกษาธรรมชาติของการสื่อสารและการทำงานที่เกี่ยวข้องกับเวลาในชีวิตประจำวัน แล้วพัฒนาเป็นแนวความคิดและหลักการ จากนั้นจึงนำเอาหลักการนี้ไปใช้พัฒนาเป็นภาษาคอมพิวเตอร์

2. ภาษาที่ใช้พัฒนาเป็นภาษาไพธอน (Python) โดยภาษา CL ที่พัฒนาขึ้นจะเป็นส่วนของ Meta-Program บนภาษาไพธอนอีกชั้นหนึ่ง แต่เราสามารถทำความเข้าใจกับ CL เสมือนภาษาคอมพิวเตอร์ภาษาหนึ่งที่ไม่ต้องพึ่งพาภาษาไพธอนได้

1.4 ทฤษฎีหรือแนวคิดที่ใช้ในงานวิจัย

ภาษาเพื่อการสื่อสาร (Communication Language) หรือขอเรียกสั้นๆ ว่า "ภาษา CL" เป็นภาษาคอมพิวเตอร์ในรูปแบบที่ใช้บรรยายถ่ายทอดลำดับการทำงานรวมทั้งการติดต่อสื่อสาร นั่นคือได้ออกแบบให้เหมาะสำหรับการทำงานที่เกี่ยวข้องกับการสื่อสารโดยเฉพาะ รวมทั้งยังมีความสามารถอย่างที่ปรากฏในภาษาคอมพิวเตอร์ทั่วไป ภาษา CL เป็นภาษาที่มีลักษณะเฉพาะตัว เพื่อให้ง่ายต่อการใช้ของผู้เขียนโปรแกรม จึงได้นำเอาไวยากรณ์ของภาษา Python [1,2] มาใช้ในการออกแบบภาษา CL โดยลักษณะเด่นที่สำคัญของภาษา CL คือมีความสามารถในการประมวลผลการทำงานต่างๆ ที่กำหนดตามเวลา เนื่องจากผู้วิจัยได้ตระหนักว่า ในธรรมชาตินั้น เวลาได้เข้ามาเกี่ยวข้องกับพฤติกรรมและการทำกิจกรรมต่างๆ อยู่เสมอ อีกทั้งโปรแกรมภาษา CL ยังสนับสนุนความสามารถในการสื่อสารโดยการโปรแกรมการทำงานให้ตอบสนองตามเหตุการณ์ (Event) ซึ่งพิจารณาจากเวลาที่สัมพันธ์กับเหตุการณ์นั้นๆ อีกด้วย ความสามารถดังกล่าว ทำให้ภาษา CL มีความยืดหยุ่นและสนับสนุนการประมวลผลที่คล้ายคลึงกับการทำงานของมนุษย์มากขึ้น ที่ผ่านมาภาษา CL ได้นำไปใช้พัฒนาเว็บบราวเซอร์ที่สามารถโปรแกรมได้ [3, 4, 5, 6, 7]

ในงานวิจัยนี้ จะกล่าวเน้นถึงแนวคิดที่นำมาใช้ในการประมวลผลเวลาและการตอบสนองต่อเหตุการณ์ที่สนับสนุนการสื่อสารในภาษา CL เป็นสำคัญ ดังนี้

1) แนวคิดเกี่ยวกับเวลา

ในวิทยานิพนธ์นี้ได้พัฒนาภาษา CL ที่มีความสามารถทำงานตามเวลา และเสนอความคิดเกี่ยวกับเวลาว่า เวลาเกิดจากการดำเนินไปของกระบวนการ (Process) อันหนึ่งอย่างต่อเนื่อง เมื่อไรก็ตามที่เกิดกระบวนการ ก็จะมีเวลาเกิดขึ้น ดังนั้น เราสามารถใช้เวลาเพื่อแสดงจุดกำเนิดของกระบวนการการทำงานได้

บทความที่เกี่ยวข้องกับเวลา [8, 9, 10, 11] ของนักวิจัยหลายท่าน ได้อธิบายถึงรูปแบบ

ของเวลา ความสัมพันธ์ของเวลา และการประมวลผลเวลาเชิงคณิตศาสตร์ รวมไปถึงความสัมพันธ์ระหว่างเวลาและการทำงาน อีกทั้งการวิจัยถึงความสัมพันธ์ของเวลาและเหตุการณ์ [12] อีกด้วย ซึ่งในวิทยานิพนธ์นี้ได้นำแนวคิดดังกล่าวมาใช้และอธิบายเวลาให้อยู่ในรูปแบบของภาษาคอมพิวเตอร์และได้เพิ่มเติมรูปแบบและความสัมพันธ์ของเวลาตามความเหมาะสม

2) แนวความคิดเกี่ยวกับการสื่อสาร

ในวิทยานิพนธ์นี้ได้พัฒนาภาษา CL ที่มีแนวความคิดว่า โปรแกรมควรมีความสามารถที่จะตอบสนองต่อเหตุการณ์ภายนอกได้ ซึ่งได้พิจารณาเหตุการณ์ภายนอกให้อยู่ในรูปแบบของข้อความสื่อสารที่ส่งมาจากเอเจนต์หรือโปรแกรมอื่น นั่นหมายถึง โปรแกรมภาษา CL จะมีความสามารถที่ทำการสื่อสารได้

ในปัจจุบัน ได้มีการวิจัยและพัฒนาภาษาสำหรับการสื่อสารของเอเจนต์ [13, 14, 15] ขึ้นมากมาย เช่น FIPA, KQML, DALI โดยสนับสนุนให้เอเจนต์สื่อสารกันได้โดยส่งข้อความถึงกันในรูปแบบการพูดของคน (Speech act) ซึ่งก็มีความสอดคล้องใกล้เคียงกับลักษณะข้อความสื่อสารในภาษา CL แต่อย่างไรก็ตาม แนวความคิดบางส่วนก็เห็นแตกต่างกันไป ซึ่งจะได้อธิบายไว้ในบทที่ 3

1.5 ขอบเขตของการวิจัย

พัฒนาภาษา CL ที่มีรูปแบบของภาษาที่ง่ายต่อการเข้าใจและการใช้งาน

1. ภาษา CL ต้องมีความสามารถพื้นฐานเช่นเดียวกับภาษาคอมพิวเตอร์ทั่วไป
2. ผู้เขียนโปรแกรมสามารถกำหนดเวลาการทำงานให้กับโปรแกรม และสามารถโปรแกรมให้มีการทำงานตอบสนองต่อเหตุการณ์ที่เกิดขึ้นได้ ด้วยภาษา CL
3. ผู้เขียนโปรแกรมสามารถโปรแกรมการทำงานที่ต้องมีการติดต่อสื่อสารผ่านเครือข่ายคอมพิวเตอร์ได้
4. สามารถใช้ CL ในการเขียนโปรแกรมเพื่อสร้างเอเจนต์สื่อสารได้
5. อินเตอร์พรีตเตอร์สำหรับภาษา CL ต้องสามารถทำงานได้อย่างถูกต้อง

1.6 ประโยชน์ที่เกิดขึ้นจากงานวิจัย

1. ได้ภาษาคอมพิวเตอร์รูปแบบใหม่ที่มีความสามารถประมวลผลการทำงานตามเวลา และสนับสนุนการโปรแกรมที่เกี่ยวข้องกับการสื่อสารและนำไปใช้สร้างเอเจนต์สื่อสาร (Communication agent) ได้

2. ได้ภาษาคอมพิวเตอร์ที่มีความสามารถมากกว่าภาษาสำหรับการสื่อสารของเอเจนต์ในงานวิจัยอื่น คือ มีความสามารถในการโปรแกรมสคริปต์ และ สามารถทำงานตามเวลาที่กำหนดได้
3. ภาษาไพธอน(Python) ได้รับการเพิ่มขีดความสามารถในการสร้างเอเจนต์สื่อสารและประมวลผลคำสั่งการทำงานตามเวลา

1.7 ขั้นตอนของการศึกษา

การทำงานแบ่งออกเป็น 3 ส่วน ดังนี้

1. รวบรวมและศึกษาข้อมูลที่สำคัญ ดังต่อไปนี้
    1.1 ความรู้พื้นฐานของภาษาคอมพิวเตอร์
    1.2 หลักการการทำงานของอินเตอร์พรีตเตอร์และคอมไพล์เลอร์
    1.3 ค้นหาและศึกษาโปรแกรมที่ช่วยในการสร้างอินเตอร์พรีตเตอร์
    1.4 การเขียนโปรแกรมภาษา Python ที่นำมาใช้ในการสร้างอินเตอร์พรีตเตอร์ และเป็นต้นแบบในการพัฒนาภาษา CL
    1.5 งานวิจัยที่เกี่ยวข้องกับการพัฒนาภาษาคอมพิวเตอร์
    1.6 งานวิจัยที่เกี่ยวข้องกับการประมวลผลเวลา
    1.7 งานวิจัยที่เกี่ยวข้องกับวิธีการติดต่อสื่อสารของเอเจนต์
2. วิเคราะห์ข้อมูลที่ได้ทำการรวบรวมมาทั้งหมด เพื่อกำหนดขอบเขตและลักษณะของชิ้นงาน
3. พัฒนาตัวอย่างงานวิจัย โดยมีขั้นตอน ดังนี้
    3.1 ออกแบบโครงงานตัวอย่าง
    3.2 ตรวจสอบลักษณะการทำงานของโครงงานตัวอย่างว่าสามารถทำงานได้
    3.3 ถ้าโครงงานตัวอย่างทำงานไม่ตรงตามลักษณะของงานที่กำหนดไว้ ต้องทำการแก้ไขให้โครงงานตัวอย่างทำงานได้ถูกต้อง
    3.4 ถ้าโครงงานตัวอย่างทำงานได้ถูกต้องตามลักษณะของงานที่กำหนดไว้ ต้องทำการพัฒนาความสามารถของโครงงานของโครงงานตัวอย่างให้เพิ่มขึ้น

1.8 ภาพรวมของเนื้อหาบทต่าง ๆ ในวิทยานิพนธ์

เนื้อหาในวิทยานิพนธ์นี้แบ่งเป็นบทได้ทั้งหมด 10 บท โดยเริ่มว่าด้วยเรื่องของภาษาไพธอน (Python) ซึ่งเป็นภาษาที่มีบทบาทอย่างมากในการพัฒนาภาษา CL ใน**บทที่** 2 โดยได้อธิบายเริ่มตั้งแต่ไวยากรณ์ภาษาไพธอน ลักษณะการแปลภาษา จนถึงกระบวนการสร้างตัวแปลภาษาด้วยภาษาไพธอน หลังจากนั้นจึงกล่าวถึงงานวิจัยที่เกี่ยวข้องกับวิทยานิพนธ์นี้ทั้งหมด อันประกอบด้วย

งานวิจัยที่เกี่ยวข้องกับเวลา และงานวิจัยที่เกี่ยวข้องกับภาษาเพื่อการสื่อสารของเอเจนต์ (ACL : Agent Communication Language) ใน**บทที่** 3 ตามด้วยการกล่าวถึงภาพรวมทั้งหมดของภาษา CL ทั้งสถาปัตยกรรมของ CL, ลักษณะของโปรแกรมภาษา CL, วิธีการสร้างเอเจนต์สื่อสารด้วยการโปรแกรมภาษา CL จนถึงตัวอย่างการทำงานของเอเจนต์ ใน**บทที่** 4

หลังจากที่อธิบายภาพรวมการทำงานและส่วนที่เกี่ยวข้องไปแล้ว จะเริ่มอธิบายถึงรายละเอียดทั้งหมดของ CL โดยเริ่มต้นด้วยเรื่องรายละเอียดของภาษา CL ตั้งแต่โครงสร้างภาษา CL, ไวยากรณ์ภาษา CL, ชนิดข้อมูล จนถึงรายละเอียดของการโปรแกรม ใน**บทที่** 5 แล้วจึงกล่าวถึงการสื่อสารใน CL ใน**บทที่** 6 โดยได้บรรยายถึง การทำงานของ CL ที่เป็นเอเจนต์สื่อสาร, รูปแบบการโต้ตอบข้อความสื่อสารของเอเจนต์, สัญลักษณ์ที่ใช้ในการสื่อสารใน CL, การส่ง-การตอบสนองต่อข้อความสื่อสาร จนถึงการสื่อสารที่มีลักษณะเป็นบทสนทนา ตามด้วยการประมวลผลโปรแกรมภาษา CL ใน**บทที่** 7 ซึ่งเป็นรายละเอียดของส่วนประกอบต่างๆ ในเอเจนต์สื่อสาร, กระบวนการทำงานของเอเจนต์ จนถึงการประมวลผลโปรแกรมภาษา CL ในระดับล่างด้วยอินเตอร์พรีตเตอร์

เมื่อกล่าวถึงรายละเอียดการทำงานของ CL เรียบร้อยแล้ว จึงเป็นส่วนที่แสดงผลการทดลองการใช้งานภาษา CL ใน**บทที่** 8 โดยทำการเสนอตัวอย่างการโปรแกรมภาษา CL อย่างง่ายที่สะท้อนให้เห็นการสื่อสารระหว่างกลุ่มเอเจนต์ แล้วแสดงการนำแนวคิดภาษา CL ไปใช้ในงานวิจัยอื่นๆ ตามด้วยการวิเคราะห์วิจารณ์การทำงานของเอเจนต์สื่อสารที่เกิดจากการโปรแกรมภาษา CL เปรียบเทียบกับภาษาเพื่อการสื่อสารของเอเจนต์ (ACL) ที่ปรากฏในปัจจุบัน และบทสรุปของวิทยานิพนธ์ อันกล่าวถึงจุดเริ่มต้นของการทำวิทยานิพนธ์ ประโยชน์ จนถึงการทำงานส่วนต่างๆ ในวิทยานิพนธ์ ปิดท้ายด้วยการเสนอแนวทางที่สามารถนำไปใช้ในการพัฒนาในอนาคต ใน**บทที่** 9 **และ** 10 ตามลำดับ

บทที่ 2
ภาษาไพธอน

ในบทนี้จะกล่าวถึงภาษาที่ใช้สร้างภาษา CL ได้แก่ ภาษาไพธอน (Python) โดยการโปรแกรมภาษา CL จะกระทำบนภาษาไพธอนอีกที หรือเรียกว่า ภาษา CL เป็น Meta-Program บนภาษาไพธอน เหตุที่เลือกใช้ภาษาไพธอนนั้นเพราะภาษาไพธอนเป็นภาษาที่มีการทำงานแบบอินเตอร์พรีตเตอร์ (Interpreter) เหมาะสำหรับแนวคิดที่ต้องมีการตอบสนองต่อเหตุการณ์ภายนอกของภาษา CL อีกทั้งภาษาไพธอนยังมีไวยากรณ์ที่เข้าใจได้ง่าย ทำให้การพัฒนาภาษา CL เป็นไปได้สะดวกยิ่งขึ้น

เนื้อหาในบทจะเริ่มอธิบายที่รูปแบบภาษาไพธอนในหัวข้อที่ 2.1 โดยการกำหนดไวยากรณ์ในภาษา CL ส่วนหนึ่งได้มาจากการอ้างอิงไวยากรณ์ในภาษาไพธอน หลังจากนั้นจะกล่าวถึงกระบวนการแปลภาษาแบบอินเตอร์พรีตเตอร์ ซึ่งเป็นการแปลภาษาที่ใช้ทั้งในภาษาไพธอนและภาษา CL ในหัวข้อที่ 2.2 ตามด้วยวิธีการสร้างตัวแปลภาษา CL ด้วยภาษาไพธอน โดยการใช้ PLY (Python Lex-Yacc) ในหัวข้อที่ 2.3 เป็นลำดับสุดท้าย

2.1 ไวยากรณ์ภาษาไพธอน [1,2]

ภาษาไพธอนคือภาษาสคริปต์ซึ่งมีการทำงานแบบอินเตอร์พรีตเตอร์(Interpreter), เป็นภาษาระดับสูง (High-level language), มีความสามารถเชิงวัตถุ (Object-oriented) และมีความสามารถในการเป็นภาษาสคริปต์ฝั่งเซิร์ฟเวอร์ (Server-side scripting language) มีการออกแบบให้ไม่ขึ้นอยู่กับระบบปฏิบัติการใดๆจึงเหมาะกับการใช้งานเป็นภาษาสคริปต์ ภาษาไพธอนมีซินแทคที่เข้าใจได้ง่ายและมีความยืดหยุ่นสูง สามารถนำมาใช้งานได้หลากหลาย และนำกลับมาใช้ใหม่ได้

2.1.1 คุณสมบัติของภาษาไพธอน

ภาษาไพธอนมีคุณสมบัติ และมีความสามารถหลายประการดังนี้

- **High-level** ภาษาไพธอนมีโครงสร้างข้อมูลระดับสูงซึ่งช่วยลดระยะเวลาในการพัฒนาโปรแกรมลง เช่น Python's list(Resizeable arrays) และ Dictionary(Hash table)
- **Object-oriented** ภาษาไพธอนเป็นภาษาเชิงวัตถุ(Object Oriented Language) เช่นเดียวกับ C++ และ JAVA และสามารถสร้างคลาสย่อยจากภาษาทั้งสองได้อีกด้วย จึงเหมาะแก่การเป็นภาษาสคริปต์ให้กับภาษาทั้งสอง

- Scalable ภาษาไพธอนมีการเขียนซอร์สโค้ดที่เข้าใจง่าย มีโครงสร้างระดับสูง และสามารถสร้างแพ็คเก็จที่บรรจุหลายๆคอมโพเน็น
- Extensible การเขียนโปรแกรมในภาษาไพธอนสามารถแบ่งซอร์สโค้ดออกเป็นโมดูลย่อย ซึ่งสามารถเรียกมาใช้ได้
- Portable โปรแกรมที่เขียนจากภาษาไพธอนจะสามารถใช้ได้กับระบบปฏิบัติการทั่วไป โดยตัวแปลภาษาจะแปลคำสั่งให้เป็น bytecode ที่ระบบปฏิบัติการนั้นเข้าใจ
- Robustภาษาไพธอนมีการจัดการข้อผิดพลาดที่ดี และผู้เขียนสามารถโปรแกรมเพื่อจัดการกับข้อผิดพลาด(Exception handler) ได้อีกด้วย
- Effective as a Rapid Prototyping Tool ผู้เขียนสามารถเขียนโปรแกรมไพธอนได้บนระบบใดๆ อีกทั้งยังมี library จำนวนมากให้ใช้ทำให้ลดระยะเวลาในการพัฒนาลงได้
- A Memory Manger ตัวแปลภาษาไพธอนจะรับผิดชอบในการจัดการกับหน่วยความจำแทนผู้เขียน ทำให้โปรแกรมมีข้อผิดพลาดน้อยลง และใช้เวลาในการพัฒนาน้อยลง
- Interpreted and (Byte-) Compiled ไพธอนเป็นภาษาที่มีการทำงานแบบอินเตอร์พรีตเตอร์ (Interpreter) แต่ภาษาไพธอนก็สามารถทำ byte-compiled ได้ซึ่งผลจากการ compile จะอยู่ในรูปแบบที่ใกล้เคียงกับภาษาเครื่องทำให้ประสิทธิภาพการทำงานดี

ตารางที่ 2.1 ตารางเปรียบเทียบคุณสมบัติต่างๆ ของภาษาไพธอนกับภาษาอื่นๆ

Language	Execution Speed	Coding Speed	Object-Oriented	GUI Coding	Dev Environment	Suitability for large tasks	Libraries available
Python	Fair	Excellent	Excellent	Good	Fair	Excellent	Good
Perl	Fair	Excellent	Fair	Good	Fair	Fair	Excellent
Vis.Basic	Good	Excellent	Fair	Excellent	Excellent	Poor	Fair
C	Excellent	Poor	Poor	n/a	Excellent	Good	Good
C++	Excellent	Fair	Excellent	n/a	Excellent	Excellent	Good
Java	Fair	Good	Excellent	Good	Excellent	Excellent	Good

2.1.2 ชนิดของข้อมูล

ชนิดของข้อมูลในภาษาไพธอนจะอยู่ในอ๊อบเจ็คที่ชื่อว่า ไพธอนอ๊อบเจ็ค การกำหนดชนิดของข้อมูลให้กับตัวแปรในภาษาไพธอนจะถูกกระทำโดยอัตโนมัติ ผู้เขียนโปรแกรมสามารถเปลี่ยนแปลงตัวแปรหนึ่งให้เก็บค่าที่ต่างไปได้โดยไม่ต้องสนใจชนิดของตัวแปรที่เก็บอยู่เดิม ชนิดของข้อมูลทั่วไปได้แก่

1) **ตัวเลข** (Numeric) ข้อมูลชนิดตัวเลข ได้แก่

- เลขฐานสิบ (Interger) คือเลขที่ประกอบด้วยเลข 0-9 ไม่ว่าจะเป็นจำนวนบวกหรือจำนวนลบ แต่ต้องไม่มีจุดทศนิยม ตัวอย่างเช่น 1, -3, 8784 ฯลฯ
- เลขฐานแปด คือ ตัวเลขที่ขึ้นต้นด้วย 0 เช่น 0715 ฯลฯ
- เลขฐานสิบหก คือ ตัวเลขที่ขึ้นต้นด้วย 0x เช่น 0x9FAC, 0x12A ฯลฯ
- เลขทศนิยม (Float number) เช่น 3.2, -45e12 ฯลฯ
- เลขจำนวนเต็มขนาดยาว (Long integer) คือตัวเลขที่ลงท้ายด้วย L หรือ l เช่น 4890549579L, 5l ฯลฯ
- เลขจำนวนเชิงซ้อน (Complex number) เช่น 9j, 5i-4j, 45i ฯลฯ

2) **จำนวนทางตรรกะ** (Boolean) โดยปกติจำนวนทางตรรกะจะมีค่าเท็จ หรือจริง สำหรับภาษาไพธอนจะแทนค่าเท็จด้วย เลขศูนย์, โครงสร้างเปล่า หรือ None value เช่น 0, [], {}, (), None และแทนค่าจริงด้วย ค่าใดๆที่ไม่ใช่ศูนย์, โครงสร้างที่มีข้อมูลอยู่ เช่น 1, [5], (7,8,98,347), "xyz"

3) **สายอักขระ** (String) เป็นข้อมูลชนิดของตัวอักษร เช่น 'This is a string.', "This is another string" สายอักขระนี้จะอยู่ภายในสัญลักษณ์ Single quoted หรือ Double quoted ก็ได้ ในข้อมูลชนิดสายอักขระนั้น จะมีอักขระพิเศษอยู่ด้วย ได้แก่

\n = Newline	\' = Single quoted	\b = Backspace	\t = Tab
\" = Double quoted	\f = Formfeed	\\ = Backslash	\a = Bell
\r = Carriage return	\v = Vertical tab		

4) **ลิสต์**(Lists) มีลักษณะคล้ายกับข้อมูลชนิดอาเรย์(Array) แต่จะต่างกันตรงที่ข้อมูลที่เก็บในลิสต์ไม่จำเป็นต้องเก็บข้อมูลชนิดเดียวกัน ข้อมูลที่สามารถเก็บในลิสต์ได้แก่ข้อมูลชนิดต่างๆ ของไพธอน เช่น ตัวเลข, ตัวอักษร, สายอักขระ หรือแม้แต่จะเป็นลิสต์ด้วยกันเองก็ได้ การเขียนอ้างอิงถึงข้อมูลชนิดลิสต์ทำได้ดังนี้

```
list2 = [1,"two",[3,4]]
```

เป็นการสร้างลิสต์ที่ประกอบด้วยสมาชิก 3 ตัวโดยตัวแรก เป็น ตัวเลข, ตัวที่สองเป็นสายอักขระ, ตัวที่สามเป็นลิสต์ที่มีสมาชิกเป็นตัวเลขสองตัว

การอ้างถึงสมาชิกในลิสต์ทำได้ดังนี้

```
list2[0]
```

จะได้ผลลัพธ์เป็น 1

ด้วยคุณสมบัติและลักษณะข้างต้น จึงสามารถใช้ลิสต์ในการสร้างสแตค (Stack) และคิว (Queue) ได้ ซึ่งจะมีประโยชน์ต่อการสร้างอินเตอร์พรีตเตอร์ภาษา CL (CL-Interpreter) ต่อไป

5) **ทัปเปิล** (Tuples) ข้อมูลชนิดทัปเปิลมีลักษณะเหมือนข้อมูลชนิดลิสต์ แต่จะต่างกันตรงที่ไม่สามารถแก้ไขข้อมูลของสมาชิกภายในทัปเปิลได้ ทัปเปิลจึงเหมาะแก่การใช้เป็นข้อมูลอ้างอิง การกำหนดทัปเปิลมีวิธีการดังนี้

tuple2 = (“one”,2,"three",4)

การจัดการกับทัปเปิลนั้นมีวิธีการเช่นเดียวกับการจัดการกับลิสต์

6) **ดิกชันนารี** (Dictionary) ข้อมูลชนิดดิกชันนารีจะประกอบไปด้วยคีย์(Keys) และค่าที่เก็บ(Values) ตัวอย่างของดิกชันนารีเป็นดังนี้

dict = {1: “one”, 2 : “two”, 3: “three”}

เราสามารถอ้างอิงข้อมูลในดิกชันนารีทำได้โดย

dict[key] จะให้ผลลัพธ์เป็นอ๊อบเจ็คที่ถูกเก็บโดยคีย์นั้น

เช่น dict[1] จะให้ผลลัพธ์เป็น “one” เป็นต้น

โดยปกติข้อมูลชนิดลิสต์จะสามารถเรียงลำดับข้อมูลได้ แต่ข้อมูลชนิดดิกชันนารีนั้นจะไม่มีการเรียงลำดับข้อมูล การอ้างอิงทำได้โดยผ่านทางคีย์เท่านั้น

7) **ข้อมูลเปล่า** (None) ข้อมูลชนิดนี้ใช้ในการระบุคาเริ่มต้นของตัวแปร หรือแสดงว่าไม่มีข้อมูล ซึ่งในการเปรียบเทียบค่านั้น None จะมีค่าเท่ากับ None เท่านั้น

2.1.3 **ตัวกระทำ** (Operators)

ตัวกระทำต่างๆ ในภาษาไพธอน แบ่งออกเป็น 4 ประเภทคือ ตัวกระทำทางตรรกะ, ตัวกระทำทางการเปรียบเทียบ, ตัวกระทำทางบิตไวส์ และตัวกระทำทางคณิตศาสตร์

1) **ตัวกระทำทางตรรกะ** (Logical operators) ตัวกระทำทางตรรกะมีทั้งหมด 3 ตัวด้วยกันคือ and, or, not

2) **ตัวกระทำทางการเปรียบเทียบ** (Comparison operators) ตัวกระทำที่ใช้เปรียบเทียบค่าต่างๆ ได้แก่ <, >, <=, >=, ==, <>, !=, in, not in, is, is not

3) **ตัวกระทำทางบิตไวส์** (Bitwise operators) เป็นตัวกระทำที่ใช้คำนวณเลขฐานสองได้แก่ << (Shift left), >> (Shift right), &(and), | (or), ^ (xor), ~ (not)

4) **ตัวกระทำทางคณิตศาสตร์** (Arithmatic-Style operators) ตัวกระทำทางคณิตศาสตร์นี้สามารถใช้งานไม่เพียงกับตัวเลขเท่านั้น แต่ยังสามารถใช้งานกับข้อมูลชนิดอื่นๆ เช่นสายอักขระได้ ตัวกระทำดังกล่าวได้แก่ ตัวกระทำ +, -, *, /, **, %

5) **ลำดับของตัวกระทำ** (Precedence) ลำดับของตัวกระทำเป็นดังเช่นโปรแกรมคอมพิวเตอร์ทั่วไปซึ่งมีลำดับดังนี้คือ or, and, not, (<, <=, ==, >=, >, !=, <>, is, in, not, not in), |, ^, &, (<<, >>), (+, -), (*, /, %), **, (unary+, unary-, unary~)

2.1.4 **คำสงวน** (Reserved words)

คำสงวนในภาษาไพธอนแบ่งเป็น คีย์เวิร์ด(Keywords) และบิวท์อินฟังก์ชั่น(Built-in function)

1) **คีย์เวิร์ด** (Keywords) ได้แก่

'and', 'assert', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'exec', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'not', 'or', 'pass', 'print', 'raise', 'return', 'try', 'while'

2) **บิวท์อินฟังก์ชั่น** (Built-in function) ได้แก่

'import', 'abs', 'apply', 'buffer', 'callable', 'chr', 'cmp', 'coerec', 'compile', 'complex', 'delattr', 'dir', 'divmod', 'eval', 'execfile', 'filter', 'float', 'getattr', 'globals', 'hasattr', 'hash', 'hex', 'id', 'input', 'int', 'intern', 'isins', 'issubclass', 'len', 'list', 'locals', 'long', 'map', 'max', 'min', 'oct', 'open', 'ord', 'pow', 'range', 'raw_input', 'reduce', 'reload', 'repr', 'round', 'setattr', 'slide', 'str', 'tuple', 'type', 'vars', 'xrange'

2.1.5 **กลุ่มของโค้ด** (Code block)

การกำหนดกลุ่มของซอร์สโค้ดจะใช้การย่อหน้า(Indentation) แทนสัญลักษณ์ ซึ่งต่างจากภาษาอื่นๆ เช่น ภาษา C และ Java ใช้วงเล็บปีกกา และภาษา Pascal ใช้ begin end เป็นตัวกำหนดกลุ่มของซอร์สโค้ด

การที่ไพธอนใช้การย่อหน้าในการกำหนดกลุ่มของซอร์สโค้ดทำให้ช่วยลดความยุ่งยาก และข้อผิดพลาดที่มักเกิดขึ้นจากการลืมพิมพ์สัญลักษณ์ที่ใช้ในการกำหนดกลุ่ม และข้อดีอีกอย่างหนึ่งคือทำให้ซอร์สโค้ดอ่านง่ายเนื่องจากทุกครั้งที่ขึ้นกลุ่มใหม่จะต้องทำการย่อหน้าซึ่งทำให้ซอร์สโค้ดมีความเป็นระเบียบ

2.1.6 **การกำหนดค่าให้ตัวแปร** (Variable assignment)

การกำหนดค่าให้ตัวแปรใช้เครื่องหมายเท่ากับ(=) ซึ่งการกำหนดค่าให้ตัวแปรนั้นไม่ได้เป็นการนำค่าไปใส่ให้ตัวแปรจริงๆ แต่เป็นการกำหนดให้ตัวแปรอ้างอิงไปยังไพธอนอ๊อบเจ็คที่มีค่านั้นอยู่ โดยไพธอนอ๊อบเจ็คนั้นอาจถูกสร้างขึ้นมาใหม่ตอนที่กำหนดค่า หรือเป็นไพธอนอ๊อบเจ็คที่มีอยู่แล้วจากตัวแปรอื่นก็ได้

การกำหนดค่าให้ตัวแปรในภาษาไพธอน นอกจากจะกำหนดค่าหนึ่งค่าให้กับตัวแปรในรูปแบบ Var = Value แล้ว ยังสามารถกำหนดค่าหนึ่งค่าให้กับตัวแปรหลายๆ ตัวได้ เช่น Var1 = Var2 = Value หรือสามารถกำหนดค่าหลายๆ ค่าให้กับตัวแปรหลายตัวได้ด้วยประโยคเดียว ซึ่งในกรณีหลังนี้ จำนวนค่าที่กำหนดจะต้องเท่ากับจำนวนตัวแปร เช่น (Var1, Var2) = (Value1, Value2)

2.1.7 **คำสั่งควบคุมการทำงาน** (Control flow)

คำสั่งลูปเป็นคำสั่งที่ใช้ในการควบคุมลำดับการประมวลผลของคำสั่งต่างๆ โดยให้ประมวลผลวนรอบไปเรื่อยๆจนกว่าจะถึงเงื่อนไขที่กำหนด คำสั่งลูปต่างๆได้แก่ IF, WHILE, FOR

2.1.7.1 **คำสั่ง** IF การประมวลผลประโยค if นั้น จะทำการตรวจสอบเงื่อนไขซึ่งกำหนดโดย expression ถ้าเงื่อนไขถูกต้อง จึงทำสเตทเมนต์ที่กำหนด รูปแบบของคำสั่ง IF เป็นดังนี้

```
if <EXPRESSION>:
       <STATEMENT>
elif <EXPRESSION>:
       <STATEMENT>
else:
       <STATEMENT>
```

ตัวอย่างของการใช้คำสั่ง if แสดงได้ดังนี้

```
if  x == 0:
       print 'x equal 0'
elif x >0:
       print 'x is positive number'
else:
       print 'x is negative number'
```

2.1.7.2 **คำสั่ง** WHILE ใช้สำหรับการประมวลผลแบบวนรอบ ซึ่งจะทำงานไปเรื่อยๆ จนกว่าเงื่อนไขหรือ expression จะเป็นเท็จ คำสั่ง WHILE นั้นมีการตรวจสอบเงื่อนไขก่อนการทำคำสั่งในลูปทุกครั้ง รูปแบบคำสั่ง while เป็นดังนี้

```
while <EXPRESSION> :
       <STATEMENT>
```

ตัวอย่างของการใช้คำสั่ง while แสดงได้ดังนี้

```
while 1:
       print 'Infinite loop'
```

2.1.7.3 **คำสั่ง** FOR เป็นประโยคที่ทำการประมวลผลแบบวนรอบจนครบตามจำนวนที่กำหนด คำสั่ง FOR เป็นคำสั่งที่จะวนรอบการทำงานจนครบจำนวนที่กำหนด รูปแบบของคำสั่ง for เป็นดังนี้

```
for <VARIABLE> in <RANGE>:
       <STATEMENT>
```

ตัวอย่างของการใช้คำสั่ง for แสดงได้ดังนี้

```
list1 = [1,'a',[b,c]]
for var1 in list1:
       print var1
```

โดยผลลัพธ์ที่ได้จากการรันคำสั่งเป็นดังนี้

```
1
'a'
[b,c]
```

โดย <RANGE> คือตัวแปรที่เป็นลิสต์ หรือคำสั่งเฉพาะบางคำสั่ง ส่วน <VARIABLE> คือตัวแปรที่เก็บค่าของสมาชิกทุกตัวของ <RANGE> การทำงานของคำสั่ง FOR จะทำงานวนรอบเป็นจำนวนเท่ากับจำนวนสมาชิกของ <RANGE>

2.1.8 **การสร้างคำนิยามของฟังก์ชัน** (Function Definition)

ผู้ใช้สามารถสร้างฟังก์ชันใหม่ขึ้นเองได้ เพื่อความสะดวกในการเขียนโปรแกรม ซึ่งการสร้างฟังก์ชันใหม่นั้น มีรูปแบบคำสั่ง ดังนี้

```
def <Function name> (<Parameter>) :
        <Statement>
```

สำหรับ <Parameter> หมายถึง พารามิเตอร์ต่างๆ ของฟังก์ชันตามที่ผู้เขียนกำหนด อาจจะมากกว่า 1 ตัวก็ได้ โดยใช้เครื่องหมาย "," แยกระหว่างพารามิเตอร์ต่างๆ เช่น

```
def compare(x,y) :
        if x<y : print("OK")
```

2.1.9 **การสร้างคลาส** (Class Definition)

ผู้ใช้สามารถสร้างคลาสขึ้นเองได้ด้วยรูปแบบคำสั่ง ดังนี้

```
class <Class name> :
      <Method>
```

เมื่อทำการนิยามคลาสแล้ว ผู้เขียนสามารถสร้างอินสแตนส์(instance) ของคลาส และเรียกเมธอด (Method) จากอินสแตนส์นั้นได้ ตัวอย่างเช่น

```
class MyDataWithMethod :
      def printFoo(self) :
           print 'You invoked printFoo()!'
```

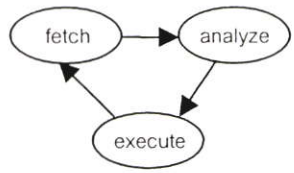
ผู้เขียนสามารถสร้างอินสแตนส์ของคลาสนี้ได้ด้วยการใช้คำสั่ง myObj = MyDataWithMethod() เมื่อต้องการเรียกเมธอดในคลาสก็ทำได้โดย myObj.printFoo() หน้าจอก็จะแสดง You invoked printFoo()! ออกมาเป็นผลลัพธ์

2.2 **อินเตอร์พรีตเตอร์** (Interpreter)

ภาษา CL จะถูกแปลความหมายด้วยตัวแปลภาษาแบบอินเตอร์พรีตเตอร์ (Interpreter) เรียกว่า CL-Interpreter ซึ่งเป็นอินเตอร์พรีตเตอร์ที่พัฒนาขึ้นด้วยภาษาไพธอน

อินเตอร์พรีตเตอร์ คือ โปรแกรมที่ทำการรับซอร์สโปรแกรม (source program) แล้วทำการรันซอร์สโปรแกรมนั้นทันที ซึ่งถ้าเปรียบเทียบกับ CL-Interpreter แล้วซอร์สโปรแกรม ก็คือ ภาษา CL เมื่อ CL-Interpreter ได้รับภาษา CL แล้วก็จะประมวลผลคำสั่งทันที

อินเตอร์พรีตเตอร์มีขั้นตอนการทำงานทั้งหมด 3 ขั้นตอน คือ เฟช (fetch), อนาไลซ์ (analyze) และ เอ็กซิคิวต์ (execute) ตามลำดับ ดังในรูปที่ 2.1



รูปที่ 2.1 ภาพการทำงานของ Interpreter

ซึ่งแสดงเป็นโค้ดได้ดังนี้

```
do {
            next_instruction = fetch()
            analyze(next_instruction)
            execute(next_instruction)
} while (still running)
```

นั่นคือ อินเตอร์พรีตเตอร์จะทำการเฟชคำสั่งจากแหล่งเก็บหรือคำสั่งที่ได้รับจากผู้ใช้ เมื่อได้รับคำสั่งแล้ว อินเตอร์พรีตเตอร์จะทำการวิเคราะห์ (Analyze) คำสั่งนั้นออกเป็นส่วนต่างๆ เสร็จแล้วจึงทำการประมวลผลคำสั่งนั้นต่อไป

สำหรับขั้นตอนในการวิเคราะห์ของอินเตอร์พรีตเตอร์นั้น จะเริ่มด้วยการอ่านสายอักขระ (Characters stream) ของคำสั่งหนึ่งๆ จากด้านซ้ายไปด้านขวา แล้วจัดกลุ่มเป็น token ซึ่งหมายถึงลำดับของอักขระที่มีความหมาย เรียกว่าเป็นการทำ Lexical analysis เมื่อทำ Lexical analysis เสร็จแล้ว จะเป็นขั้นตอนของ Syntax analysis หรือ Parsing เพื่อทำการจัดกลุ่ม token ต่างๆ ตามรูปแบบของแกรมม่าซึ่งถูกระบุด้วย Context Free Grammar(CFG) แล้วเขียนอยู่ในรูปแบบของทรีเรียกว่า Abstract Syntax Tree เมื่อได้ผลลัพธ์จากการวิเคราะห์แล้ว อินเตอร์พรีตเตอร์จะพิจารณาความหมายของคำสั่ง แล้วประมวลผลคำสั่งนั้นต่อไป

การทำงานในขั้นตอนต่างๆ ของอินเตอร์พรีตเตอร์นั้น จำเป็นต้องมีการจัดการตารางสัญลักษณ์ (Symbol Table) เพื่อเก็บ identifier ที่ใช้ในซอร์สโปรแกรมรวมถึงทำการรวบรวมข้อมูลต่างๆ ของแอททริบิวต์สำหรับแต่ละ identifier ด้วย ซึ่งแอททริบิวต์เหล่านี้ อาจมีข้อมูลของตำแหน่ง identifier, type, scope เป็นต้น

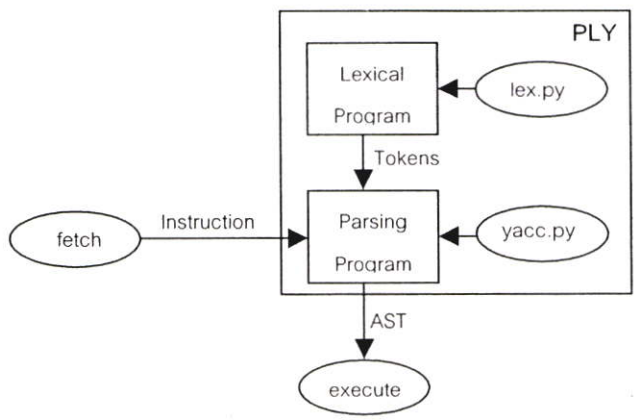
2.3 PLY (Python Lex-Yacc) [16]

เพื่อความสะดวกในการสร้างอินเตอร์พรีตเตอร์สำหรับภาษา CL นั้น ผู้เขียนจำเป็นต้องใช้เครื่องมือเพื่อทำ lexical analysis และ parsing (syntax analysis) ซึ่งเป็นขั้นตอนสำคัญของการทำงานของอินเตอร์พรีตเตอร์ โดย เครื่องมือที่ผู้เขียนนำมาใช้นี้เป็น lex และ yacc ที่อิมพลีเมนต์ด้วยภาษาไพธอน (Python) เรียกว่า PLY (Python Lex-Yacc)

PLY ประกอบด้วยส่วนสำคัญ 2 ส่วน คือ lex.py และ yacc.py โดยที่ lex.py ทำหน้าที่ตัด token จากซอร์สโปรแกรมซึ่ง token ต่างๆ นั้นจะถูกระบุด้วย regular expression rule ส่วน yacc.py นั้น ทำหน้าที่จัดการ parsing ซึ่ง yacc.py ใช้การ parse แบบ LR-Parsing และทำการสร้าง parsing table ด้วยอัลกอริทึม SLR

ผู้เขียนได้อิมพลีเมนต์อินเตอร์พรีตเตอร์ โดยการ import lex.py และ yacc.py ของ PLY มาใช้ (มอง lex.py และ yacc.py เป็นโมดูล) โดยอินเตอร์พรีตเตอร์ที่สร้างขึ้นจะถูกแบ่งออกเป็น 2 โปรแกรม คือ โปรแกรมที่ทำ lexical analysis และโปรแกรมที่ทำ Parsing

สำหรับโปรแกรมในส่วนแรกนั้นต้อง import lex.py เพื่อสร้าง lexer และผลลัพธ์ที่ได้จากโปรแกรมส่วนนี้คือ token ต่างๆ ในโปรแกรม CL สำหรับโปรแกรมที่ทำ parsing จะดึงเอา token ที่ได้จากโปรแกรมในส่วนแรก (ด้วยวิธีการ import) มาใช้พิจารณาในกฎไวยากรณ์ (Grammar rule) นอกจากนี้ยังต้อง import yacc.py เพื่อสร้าง Parser อีกด้วย ส่วนผลลัพธ์ที่ได้จากการ Parsing นั้นขึ้นกับผู้พัฒนา ซึ่งอาจจะเป็น Abstract Syntax Tree หรืออาจจะทำการประมวลผลไปเลยก็ได้ แต่สำหรับการสร้าง CL-Interpreter นี้ กำหนดให้ผลลัพธ์ที่ได้เป็น Abstract Syntax Tree การสร้างอินเตอร์พรีตเตอร์ด้วย PLY แสดงได้ดังรูปที่ 2.2



รูปที่ 2.2 การสร้างอินเตอร์พรีตเตอร์ด้วย PLY

2.3.1 ตัวอย่างการสร้างอินเตอร์พรีตเตอร์ด้วย PLY

ตัวอย่างที่นำมาแสดงนี้ เป็นการสร้างอินเตอร์พรีตเตอร์สำหรับเอ็กเพรสชันของตัวเลขอย่างง่ายๆ เพื่อให้ผู้อ่านได้เข้าใจขั้นตอนการสร้างอินเตอร์พรีตเตอร์ด้วย PLY ได้ชัดเจนมากขึ้น ดังที่ได้อธิบายข้างต้น ถึงโปรแกรมที่ทำ Lexical Analysis และ Parsing จะมีลักษณะดังนี้

2.3.1.1 โปรแกรมที่ทำ Lexical Analysis (calclex.py)

lex.py เป็นโมดูลที่ใช้สำหรับการตัด token ของโปรแกรม ซึ่ง token แต่ละตัวนั้นจะถูกกำหนดได้ด้วย regular expression rule และไฟล์ที่ได้นำมาแสดงนี้เป็น lexer อย่างง่ายที่ทำการตัด token สำหรับเอ็กเพรสชันของตัวเลขทั่วไป

```
# ------------------------------------------------------------
# calclex.py
#
# tokenizer for a simple expression evaluator for
# numbers and +,-,*,/
# ------------------------------------------------------------
import lex

# List of token names.   This is always required
tokens = (
   'NUMBER',
   'PLUS',
   'MINUS',
   'TIMES',
   'DIVIDE',
   'LPAREN',
   'RPAREN',
)

# Regular expression rules for simple tokens
t_PLUS    = r'\+'
t_MINUS   = r'-'
t_TIMES   = r'\*'
t_DIVIDE  = r'/'
t_LPAREN  = r'\('
t_RPAREN  = r'\)'

# A regular expression rule with some action code
def t_NUMBER(t):
    r'\d+'
    try:
        t.value = int(t.value)
    except ValueError:
        print "Line %d: Number %s is too large!" %
      (t.lineno,t.value)
  t.value = 0
    return t

# Define a rule so we can track line numbers
def t_newline(t):
    r'\n+'
    t.lineno += len(t.value)

# A string containing ignored characters (spaces and tabs)
t_ignore  = ' \t'

# Error handling rule
```

```
def t_error(t):
    print "Illegal character '%s'" % t.value[0]
    t.skip(1)

# Build the lexer
lex.lex()

# Test it out
data = '''
3 + 4 * 10
  + -20 *2
'''

# Give the lexer some input
lex.input(data)

# Tokenize
while 1:
    tok = lex.token()
    if not tok: break      # No more input
    print tok
```

จากตัวอย่าง ลิสต์ " tokens " เป็นลิสต์ที่ใช้กำหนดชื่อของ token เท่าที่มีทั้งหมดไว้ การกำหนด regular expression สำหรับ token แต่ละตัวที่ปรากฏในลิสต์ "tokens" จะเขียนขึ้นต้นด้วย "t_" เพื่อบอกการกำหนด token ให้กับโปรแกรม สำหรับ token ทั่วไป สามารถกำหนด regular expression เป็นสตริงได้ เช่น

```
t_PLUS = r'\+'
```

ในกรณีของ token ที่จำเป็นต้องมีการประมวลผลหรือ มีการทำงานบางอย่างก่อน กฎที่ใช้กำหนด token นั้น จะเป็นรูปแบบของฟังก์ชัน ตัวอย่างเช่น

```
def t_NUMBER(t):
        r'\d+'
        try:
            t.value = int(t.value)
        except ValueError:
            print "Line %d: Number %s is too large!"
            %(t.lineno,t.value)
            t.value = 0
        return t
```

ฟังก์ชันที่ใช้กำหนด token นี้จะมีอาร์กิวเมนต์เพียง 1 อาร์กิวเมนต์เสมอ ซึ่งอาร์กิวเมนต์ดังกล่าวเป็นอินสแตนส์ของ LexToken ซึ่งมีแอททริบิวต์ทั้งหมด 3 ตัว ได้แก่ t.type, t.value และ t.lineno เพื่อใช้บอกประเภท, ค่า และบรรทัดปัจจุบันของ token ตามลำดับ โดยปกติแล้ว จะกำหนด t.type เป็นชื่อที่เขียนตามหลัง t_ สำหรับการกำหนด regular expression ด้วยฟังก์ชันนั้น ผู้เขียนสามารถเปลี่ยนแปลงค่าแอททริบิวต์ในวัตถุ LexToken ได้ แต่อย่างไรก็ตาม เมื่อฟังก์ชันทำงานเสร็จ จะต้องมีการคืนค่าของ token ออกมาด้วย

t_ignore นั้นเป็นส่วนที่สงวนไว้สำหรับ lex.py โดยใช้สำหรับตัดช่องว่างหรืออักขระที่ไม่จำเป็นออกไป ส่วนฟังก์ชัน t_error() นั้นใช้สำหรับตรวจจับความผิดพลาดของการตัด token (lexing) ซึ่งจะเกิดขึ้นเมื่อได้รับอักขระที่ผิดปกติไปจากกฎที่กำหนดไว้

ในการสร้าง Lexer นั้น ต้องทำการเรียกฟังก์ชัน lex.lex() โดยฟังก์ชันนี้จะทำการอ่าน regular expression แล้วสร้าง lexer ขึ้น ฟังก์ชัน 2 ฟังก์ชันที่ใช้ควบคุม lexer ได้แก่

- lex.input(data) ทำการตั้งค่าและเก็นอินพุทใหม่ให้กับ lexer
- lex.token() จะคืนค่า token ถัดไปออกมา

เมื่อทำการประมวลผลโค้ดที่แสดงไว้เป็นตัวอย่างข้างต้น จะได้ผลลัพธ์ดังนี้

```
$ python example.py
LexToken(NUMBER,3,2)
LexToken(PLUS,'+',2)
LexToken(NUMBER,4,2)
LexToken(TIMES,'*',2)
LexToken(NUMBER,10,2)
LexToken(PLUS,'+',3)
LexToken(MINUS,'-',3)
LexToken(NUMBER,20,3)
LexToken(TIMES,'*',3)
LexToken(NUMBER,2,3)
```

2.3.1.2 **โปรแกรมที่ทำ** Parsing (parsing.py)

โปรแกรมนี้จะทำการรับ token จาก calclex.py มาพิจารณากับกฎไวยากรณ์ (Grammar Rule) โดยที่ token นั้นจะแทนด้วยอักษรตัวใหญ่ทั้งหมด แต่ถ้าเป็น non-terminal นั้นจะแทนด้วยอักษรตัวเล็กทั้งหมด แต่ไม่ได้แสดงให้เห็นในที่นี้ ผู้พัฒนาต้องเรียกฟังก์ชัน yacc() เพื่อทำการสร้าง Parser และเรียกฟังก์ชัน parse() เพื่อทำการ Parse

```
# Yacc example

import yacc

# Get the token map from the lexer.  This is required.
from calclex import tokens

def p_expression_plus(t):
    'expression : expression PLUS term'
    t[0] = t[1] + t[3]

def p_expression_minus(t):
    'expression : expression MINUS term'
    t[0] = t[1] - t[3]

def p_expression_term(t):
    'expression : term'
    t[0] = t[1]

def p_term_times(t):
    'term : term TIMES factor'
    t[0] = t[1] * t[3]

def p_term_div(t):
    'term : term DIVIDE factor'
```

```
    t[0] = t[1] / t[3]

def p_term_factor(t):
    'term : factor'
    t[0] = t[1]

def p_factor_num(t):
    'factor : NUMBER'
    t[0] = t[1]

def p_factor_expr(t):
    'factor : LPAREN expression RPAREN'
    t[0] = t[2]

# Error rule for syntax errors
def p_error(t):
    print "Syntax error in input!"

# Build the parser
yacc.yacc()

while 1:
   try:
       s = raw_input('calc > ')
   except EOFError:
       break
   if not s: continue
   result = yacc.parse(s)
   print result
```

จากตัวอย่างดังกล่าว จะเห็นว่ากฎไวยากรณ์สามารถกำหนดได้ด้วยฟังก์ชันในภาษาไพธอน ซึ่งใช้สตริงเป็นตัวระบุ Context-Free-Grammar ในทุกๆ ฟังก์ชันจะรับอาร์กิวเมนต์ t เพียง 1 ตัว ซึ่งประกอบด้วยค่าต่างๆ ของสัญลักษณ์ในไวยากรณ์ โดยค่าของ t[i] จะแมปเข้ากับสัญลักษณ์ต่างๆ ดังนี้

```
def p_expression_plus(t):
    'expression : expression PLUS term'
    #   ^            ^        ^    ^
    #  t[0]          t[1]     t[2] t[3]

    t[0] = t[1] + t[3]
```

ค่าของ t[i] เป็นค่าของ t.value ที่กำหนดไว้ในโมดูล lexer สำหรับส่วนที่เป็น non-terminal นั้น ค่าที่ได้จะเป็นค่าในตำแหน่ง t[0] เมื่อมีการรีดิวซ์(Reduce) กฎ ในตัวอย่างนี้ token NUMBER จะเก็บค่า integer ไว้ในส่วน value

กฎแรกที่กำหนดไว้ใน yacc จะถือเป็นสัญลักษณ์เริ่มต้น เมื่อไรก็ตามที่กฎเริ่มต้นนี้ถูกรีดิวซ์ โดย parser และไม่มีอินพุทอื่นอีก กระบวนการ parsing จะจบลงและคืนค่าที่ได้ออกมา (นั่นคือค่าที่อยู่ใน t[0] ของกฎที่เขียนไว้บนสุด)

ในการสร้างตัว parser นั้น ทำได้โดยการเรียกฟังก์ชัน yacc.yacc() ฟังก์ชันนี้จะพยายามสร้าง ตาราง LR parsing ทั้งหมดสำหรับไวยากรณ์ที่ได้กำหนดไว้ในโปรแกรม เมื่อมีการเรียก yacc.yacc() จะมีข้อความปรากฏดังนี้

```
$ python calcparse.py
yacc: Generating SLR parsing table...
calc >
```

ผลลัพธ์ที่ได้จากการ parse คือ parsing table จะถูกเก็บไว้ในไฟล์ parsetab.py ในการประมวลผลแต่ละครั้ง yacc จะทำการโหลดตารางจาก parsetab.py มาใช้ซ้ำอีก หากไม่ได้มีการเปลี่ยนแปลงไวยากรณ์ของภาษา ในทางกลับกัน yacc จะทำการสร้าง parsing table เก็บในไฟล์ parsetab.py ใหม่อีกครั้ง

บทที่ 3
งานวิจัยที่เกี่ยวข้อง

ภาษาเพื่อการสื่อสาร หรือ ภาษา CL เป็นภาษาที่สนับสนุนการทำงานตามเวลาและเหตุการณ์ นำไปสู่ความสามารถในการสร้างเอเจนต์ที่ติดต่อสื่อสารกันได้

เนื้อหาในบทนี้ จะกล่าวถึงงานวิจัยที่สำคัญซึ่งเกี่ยวข้องกับวิทยานิพนธ์นี้ทั้งหมด อันได้แก่ งานวิจัยที่กล่าวถึงเวลาในหัวข้อที่ 3.1 และงานวิจัยที่เกี่ยวข้องกับภาษาเพื่อการสื่อสารของเอเจนต์ที่พบในปัจจุบันในหัวข้อที่ 3.2 โดยอันดับแรกจะกล่าวแนวคิดและต้นกำเนิดภาษาเพื่อการสื่อสารของเอเจนต์ (ACL : Agent Communication Language) ในหัวข้อที่ 3.2.1 ตามด้วยภาษาเพื่อการสื่อสารของเอเจนต์ที่มีการพัฒนาในปัจจุบัน ได้แก่ ภาษา KQML และ ภาษา DALI ในหัวข้อที่ 3.2.2 และ 3.2.3 ตามลำดับ

3.1 งานวิจัยที่เกี่ยวข้องกับเวลา

เนื่องจากเวลาเป็นสิ่งสำคัญที่เกี่ยวข้องกับการทำงาน(Action) ซึ่งถือว่าเป็นกิจกรรมทั่วไปของมนุษย์ จึงมีนักวิจัยหลายท่านให้ความสนใจศึกษาค้นคว้าและหาวิธีในการอธิบายเวลาในรูปแบบคณิตศาสตร์ อันได้แก่งานวิจัยของ Allen [8, 9, 10] และ Kowalski ร่วมกับ Sergot [12] ที่ได้เสนอแนวคิดในการแทนรูปแบบและการประมวลผลเวลาในเชิงคณิตศาสตร์ ซึ่งนับเป็นจุดเริ่มต้นและเป็นพื้นฐานสำหรับงานวิจัยนี้

Allen ได้ศึกษาเวลาในรูปแบบต่างๆ แล้วสรุปว่า เวลาสามารถเป็นได้ใน 2 ลักษณะ คือ จุดเวลา (Dating Schemes) [9] และ ช่วงเวลา (Interval) [10] ดังนี้

1) **จุดเวลา** ใช้สำหรับระบุเวลาที่เป็นจุดเริ่มต้นของสิ่งต่างๆ กำหนดเป็น Tuple ที่ประกอบด้วย ปี เดือน วัน ชั่วโมง นาที วินาที และอาจรวมถึงเศษเสี้ยวของวินาที เช่น (2003 9 21 10 0 0) หมายถึง ณ วันที่ 21 กันยายน 2003 เวลา 10.00 น. เป็นต้น
2) **ช่วงเวลา** ใช้กราฟเส้นตรงแสดงเหตุการณ์ที่สัมพันธ์กับช่วงเวลาต่างๆ ซึ่งแต่ละจุดปลายของเส้นตรง แสดงถึงจุดเวลาเริ่มต้นและจุดเวลาสิ้นสุดสำหรับเหตุการณ์หนึ่งๆ

Allen ได้ศึกษาความสัมพันธ์ของช่วงเวลา โดยใช้กราฟเส้นตรงในการแสดงความสัมพันธ์ของเวลาในช่วงต่างๆ เช่น ถ้า e1 < e2 และ e2 < e3 แสดงว่า e1 < e3 หมายถึง ถ้าเหตุการณ์ที่ 1 เกิดก่อนเหตุการณ์ที่ 2 และ เหตุการณ์ที่ 2 เกิดก่อนเหตุการณ์ที่ 3 จะสรุปได้ว่า เหตุการณ์ที่ 1 เกิดก่อนเหตุการณ์ที่ 3 เป็นต้น และ Allen ยังได้นำเสนออัลกอริทึมเพื่อหาความสัมพันธ์ต่างๆ เหล่านี้อีกด้วย [8] นอกจากนี้ Allen [10] ยังได้เสนอผลงานวิจัยเกี่ยวกับการทำงาน (Action) ซึ่งมีความ

สัมพันธ์กับเวลาในรูปตรรกศาสตร์ โดยพิจารณาจากเหตุการณ์และเวลาที่เกิดขึ้นในการทำงาน ซึ่งใกล้เคียงกับงานวิจัยของ Kowalski และ Sergot [12] ที่เสนอแนวความคิดในการจัดการเวลาและเหตุการณ์ในรูปแบบตรรกศาสตร์ ซึ่งเวลาที่ใช้ในงานวิจัยของทั้งสองเป็นลักษณะช่วงเวลา

สำหรับการประมวลผลเวลาของภาษา CL ในงานวิจัยนี้ได้ศึกษาเพิ่มเติมนอกเหนือจากงานของ Allen และของ Kowalski และ Sergot โดยได้แจกแจงประเภทของเวลาได้เป็น 3 ประเภท คือ จุดเวลา(Time Point), ช่วงเวลา(Time Interval) และระยะเวลา(Time Duration) รวมทั้งได้นำเสนอพีชคณิตสำหรับประมวลผลเวลาทั้ง 3 ประเภทนี้ด้วย ซึ่งผลที่ได้นำไปใช้ในการออกแบบภาษา CL

3.2 งานวิจัยที่เกี่ยวข้องกับภาษาเพื่อการสื่อสารของเอเจนต์

3.2.1 ภาษาเพื่อการสื่อสารของเอเจนต์ (ACL:Agent Communication Languages) [13]

ปัจจุบันได้มีความพยายามที่จะนำรูปแบบของภาษามนุษย์มาเป็นมาตรฐานที่ใช้สื่อสารในซอฟต์แวร์เอเจนต์ (Software agent) เอเจนต์ (ในที่นี้ หมายถึง Software agent) จะเสนอรูปแบบสำหรับการพัฒนาซอฟต์แวร์ที่มีความสามารถแบบ autonomy, adaptivity และ cooperation ซึ่งแนวคิดนี้ได้พบเห็นอยู่ทั่วไป ภาษาที่ใช้สำหรับสื่อสารในเอเจนต์นั้นต้องสามารถสื่อสารกับเอเจนต์ได้คล้ายภาษาธรรมชาติ (Natural Language) ที่สามารถสื่อสารในสังคมมนุษย์ เราเรียกภาษาที่ใช้สำหรับสื่อสารในเอเจนต์นี้ว่า ACL (Agent Communication Language)

1) หลักการพื้นฐานของ ACL

ACL ถูกพัฒนาขึ้นเพื่อเพิ่มความสามารถในการแลกเปลี่ยนความรู้และข่าวสารให้กับเอเจนต์ ซึ่งหมายถึง ได้มีการวิจัยและพัฒนาถึงการแลกเปลี่ยนความรู้และข่าวสารระหว่างแอพพลิเคชันนั่นเอง อันมีจุดประสงค์คล้ายกับ RPC(Remote Procedure Call), RMI(Remote Method Invocation) หรือแม้แต่ CORBA ที่ได้มีการพัฒนาขึ้นแล้ว แต่ ACL มีลักษณะเด่นที่เหนือกว่า CORBA อยู่ 2 ประการ ดังนี้

- ACL มีความสามารถในการจัดการ proposition, rule and action
- ACL Message อธิบาย state ที่ต้องการด้วย declarative language มากกว่าในรูปของ procedure หรือ method

จากข้อดีดังกล่าว ทำให้เอเจนต์สามารถแลกเปลี่ยนข่าวสารที่มีความซับซ้อน เช่น สามารถใช้แผนงานและเป้าหมายร่วมกัน หรือสามารถใช้ประสบการณ์และวิธีการทำงานต่างๆ ร่วมกันได้

ในด้านเทคนิค เอเจนต์ที่ใช้ ACL จะส่งข้อความผ่านเครือข่ายโดยใช้โปรโตคอลระดับล่าง เช่น SMTP, TCP/IP, IIOP หรือ HTTP เป็นต้น ACL สามารถกำหนดให้เอเจนต์แลกเปลี่ยนข้อความได้ทั้งแบบส่งครั้งเดียวหรือแบบส่งข้อความไปมาในรูปแบบของการสนทนา ในขณะเดียวกัน

ที่ระดับบน (High-level) จะเป็นแนวคิด(concept) ของวิธีการทำงานและพฤติกรรมของเอเจนต์ซึ่งนำไปสู่ลักษณะวิธีการสื่อสารของเอเจนต์

ข้อความสื่อสารที่ใช้ใน ACL นั้นอาศัยหลักการพูดของมนุษย์ (Speech act) ซึ่งสามารถอธิบายได้ในรูปแบบของ belief, desire, intention เพื่อใช้อธิบายโมเดลระดับแนวคิด (conceptual model) ของความรู้, เป้าหมาย และข้อตกลงของเอเจนต์ เรียกว่าเป็น BDI theory

2) ความเป็นมาของการพัฒนา ACL

เริ่มต้นจากการถือกำเนิดของ The Knowledge Sharing Effort (KSE) ราวๆ ปี ค.ศ. 1990 โดยแนวคิดสำคัญของ KSE คือ การพัฒนาความสามารถของการใช้ความรู้ร่วมกันโดยอาศัยการติดต่อสื่อสารผ่านภาษาพื้นฐาน (Common Language) ทีมงานวิจัยจึงได้มีความพยายามที่จะนิยามภาษาพื้นฐานนั้นขึ้น ในโมเดล KSE ระบบซอฟต์แวร์จะทำการแลกเปลี่ยน proposition ผ่านการกำหนด propositional attitude ซึ่งประกอบด้วย 3 ส่วน ดังนี้

- เอเจนต์
- เนื้อหาของ proposition เช่น ฝนกำลังจะตก
- ความรู้สึกของเอเจนต์ที่มีต่อ proposition เช่น ความเชื่อ (believing)

ตัวอย่างของ propositional attitude คือ <a, fear, raining(t_{now})> หมายถึง เอเจนต์ a กลัวว่าฝนจะตก

ภาษาที่ใช้ในการสื่อสารระหว่างเอเจนต์ ควรมีความสามารถทำงานต่างๆ ดังนี้

- สามารถแปลไวยากรณ์ระหว่างภาษาในกลุ่มเดียวกันหรือต่างกลุ่มภาษาได้
- ต้องรับประกันว่าในแอพพลิเคชัน (application) ต้องมีการสงวนความหมายของ token เอาไว้ นั่นคือ concept, object หรือ entity เดียวกันจะต้องมีความหมายเหมือนกันเมื่อมีการใช้งานข้ามแอพพลิเคชันอื่นๆ แม้ว่าในแอพพลิเคชันต่างๆ จะมีการอ้างอิงชื่อที่แตกต่างกัน เอเจนต์ทุกตัวจะมีการรวบรวมโดเมนความรู้ที่มันจะต้องทำการตอบข้อความสื่อสารเมื่อเกิดเหตุการณ์ดังกล่าวไว้ด้วย ความรู้พื้นฐานนี้ เรียกว่า ontology ซึ่งเป็นแนวคิดของเซตของ object, concept และ entity อื่นๆ เกี่ยวกับความรู้ที่ได้และความสัมพันธ์ระหว่างความรู้เหล่านั้น ontology จะประกอบด้วย term, definition และ axiom ที่สัมพันธ์กับ term เหล่านั้น
- เอเจนต์ควรจะสามารถมีการติดต่อสื่อสารที่เกี่ยวกับข่าวสารและเนื้อหาความรู้ที่ซับซ้อนได้ เอเจนต์จำเป็นจะต้องมีการทำงานหลายอย่าง เช่น การถามคำถามเอเจนต์อื่น, แจ้งข่าวสาร, ขอบริการ, ค้นหาเอเจนต์อื่นๆ เป็นต้น ซึ่งการทำ RPC นั้นไม่สามารถจัดการการทำงานเหล่านี้ได้

ซึ่งการทำงานทั้งสามส่วนนี้ต่างเป็นอิสระต่อกัน สำหรับ ACL นั้นเกี่ยวข้องเฉพาะ proposition attitude แต่ไม่ได้คำนึงว่าจะแสดง proposition นั้นออกมาอย่างไร ซึ่ง proposition ก็คือเรื่องราวที่เอเจนต์พูดคุยกันนั่นเอง

KSE ได้เสนอให้มีการใช้ Knowledge Interchange Format (KIF) ซึ่งเขียนในรูปแบบของภาษาเชิงตรรกศาสตร์ (Logic language) เพื่อเป็นมาตรฐานสำหรับการอธิบายสิ่งต่างๆภายในระบบคอมพิวเตอร์ เช่น expert system, databases, intelligent agents เป็นต้น นอกจากนั้น นักวิจัยของ KSE ยังได้ทำการออกแบบ KIF ให้สามารถใช้เป็นสื่อกลางในการแปลเป็นภาษาอื่นๆ KIF เขียนในรูปแบบของ first-order predicate-order calculus เพื่อสนับสนุน meta-operators และ definitions รายละเอียดของภาษาจะประกอบทั้งการกำหนดไวยากรณ์และความหมายของภาษา

นักวิจัยได้พัฒนาเครื่องมือและบริการสำหรับใช้ในกระบวนการใช้ ontology ร่วมกัน โดยผู้ใช้สามารถใช้เครื่องมือนี้ผ่านอินเตอร์เน็ตเพื่อทำการแก้ไข ontology ต่างๆ ที่เก็บไว้ในเซิร์ฟเวอร์ ผู้ใช้สามารถเข้าไปดู ontology ใหม่ๆ, เพิ่มเติม definition และ constraint, ตรวจสอบ logical consistency จากห้องสมุดที่จัดเตรียมไว้ให้ได้

3.2.2 ภาษา KQML (Knowledge Query and Manipulation Language) [14]

ภาษา KQML เป็นภาษาและโปรโตคอลที่ใช้สำหรับแลกเปลี่ยนข่าวสาร (information) และความรู้ (Knowledge) พัฒนาขึ้นโดย ARPA Knowledge Sharing Effort จุดประสงค์เพื่อสร้างฐานความรู้ (Knowledge base) ที่สามารถใช้งานร่วมกันและนำกลับมาใช้ใหม่ได้ หรืออาจกล่าวได้ว่า KQML เป็นภาษาที่ถูกออกแบบมาเพื่อสนับสนุนการติดต่อสื่อสารกันระหว่างซอฟต์แวร์เอเจนต์ที่สามารถสื่อสารกับเอเจนต์อื่น, สามารถทำงานร่วมกันกับเอเจนต์อื่น, สามารถใช้ความรู้และข่าวสารในการจัดการการร้องขอจากเอเจนต์อื่นได้

เมื่อกล่าวถึง KQML จะหมายความรวมถึงรูปแบบข้อความสื่อสาร (Message format) และโปรโตคอลที่ใช้จัดการกับข้อความเหล่านั้น โดยมีการขยายเซตของ performative ซึ่งกำหนดเพื่อให้เอเจนต์สามารถติดต่อสื่อสารกันได้ตามหลักการของ Speech Act นอกจากนี้ KQML ยังประกอบด้วยสถาปัตยกรรมสำหรับการใช้ความรู้ร่วมกัน (Knowledge Sharing) เรียกว่า Communication Facilitator ปัจจุบัน KQML สามารถนำมาใช้งานได้ในหลากหลายสาขา เช่น การออกแบบ, การวางแผน, การจัดตาราง เป็นต้น

1) การสื่อสารใน KQML

เอเจนต์ 2 ตัวจะสามารถพูดคุยสื่อสารกันได้นั้นจะต้องกระทำผ่านภาษาที่เอเจนต์เข้าใจทั้งสองฝ่าย (Common representation language) หรือใช้ภาษาที่สามารถแปลความความหมายได้ด้วยตัวมันเอง (inter-translatable) เพื่อใช้แปลความหมายของข้อความสื่อสารที่เอเจนต์ได้ทำการ

แลกเปลี่ยนกัน ซึ่งในที่นี้ไม่ได้หมายความแต่เพียงการใช้ความหมายของข้อความร่วมกันเท่านั้น แต่หมายรวมถึงการใช้ ontology ร่วมกันด้วย

กระบวนการสื่อสารระหว่างคอมพิวเตอร์(เอเจนต์) ประกอบด้วยขั้นตอนการปฏิบัติ ดังนี้

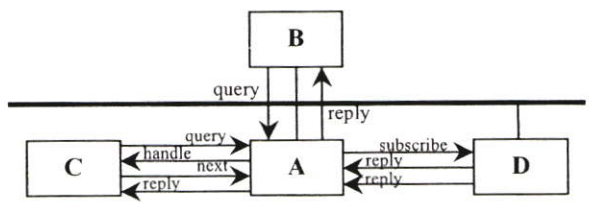
1) เราจะต้องรู้ว่าจะคุยกับใคร และจะพบผู้นั้นได้อย่างไร

2) เราจะต้องรู้ว่าจะแลกเปลี่ยนข่าวสารอย่างไร

KQML จะเกี่ยวข้องกับกระบวนการแรกในแง่ของการปฏิบัติ และในกระบวนการที่สองในแง่ของความหมาย นั่นคือ KQML เป็นทั้งภาษาและเซตของโปรโตคอลที่ทำให้โปรแกรมคอมพิวเตอร์สามารถติดต่อและแลกเปลี่ยนข่าวสารกับโปรแกรมอื่นได้

2) โปรโตคอลสำหรับสื่อสารใน KQML

KQML ได้สนับสนุนให้มีโปรโตคอลสำหรับแลกเปลี่ยนความรู้ระหว่างเอเจนต์ได้มากมาย แบบง่ายที่สุดคือ เอเจนต์หนึ่งทำหน้าที่เสมือนเป็นไคลเอนต์ (client) และส่ง query ไปยังเอเจนต์อีกตัวหนึ่งที่ทำหน้าที่เป็นเสมือนเซิร์ฟเวอร์ (server) แล้วรอการตอบสนองกลับ แสดงได้ดังการพูดคุยระหว่างเอเจนต์ A และ B ในรูปที่ 3.1 โดยคำตอบที่ได้จากเซิร์ฟเวอร์อาจเป็นคำตอบเดียวหรือเป็นกลุ่มหรือเซตคำตอบก็ได้ ในกรณีอื่นสามารถแสดงได้ดังการพูดคุยระหว่างเอเจนต์ A และ C คือ คำตอบของเซิร์ฟเวอร์ยังไม่ใช่คำตอบที่สมบูรณ์ แต่เมื่อไคลเอนต์ได้รับคำตอบแล้วก็สามารถขอคำตอบถัดไปจากเซิร์ฟเวอร์ได้ ซึ่งจะสังเกตว่าการสื่อสารที่กล่าวไปข้างต้น จะทำการรอการตอบกลับก่อนที่จะส่งข้อความอื่นออกไป เรียกว่าเป็น การสื่อสารแบบซิงโครนัส (synchronous) แต่สำหรับในบางกรณี อาจไม่เป็นเช่นนั้นดังที่แสดงในการพูดคุยระหว่างเอเจนต์ A และ D ในรูปที่ 3.1 ที่ไคลเอนต์ไม่สามารถรู้ได้ว่าจะได้รับการตอบกลับเมื่อไร และในขณะเดียวกันก็สามารถทำงานอย่างอื่นได้



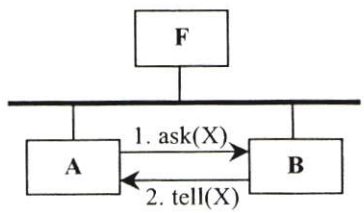
รูปที่ 3.1 รูปแบบการสื่อสารใน KQML

3) เอเจนต์ตัวกลาง Facilitators

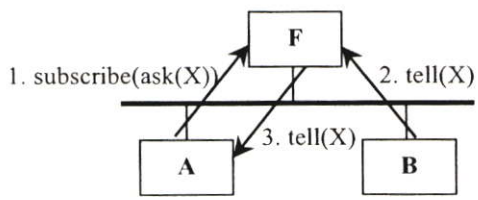
หนึ่งในมาตรฐานการออกแบบ KQML คือการสร้างภาษาที่สามารถสนับสนุนสถาปัตยกรรมเอเจนต์ที่หลากหลาย โดยได้เสนอ KQML performative ซึ่งใช้โดยเอเจนต์เพื่ออธิบายข้อมูลระดับเมต้า (meta-data) ที่กำหนดความต้องการและความสามารถของข่าวสาร นอกจากนี้ ยังได้เสนอส่วน Communication Facilitator ที่เป็นเอเจนต์ที่ทำหน้าที่บริการการทำงาน

ต่างๆ เกี่ยวกับการติดต่อสื่อสารของเอเจนต์ เช่น เก็บรีจีสทรี (registry) สำหรับการบริการ, ทำการจับคู่ข่าวสารระหว่างผู้ส่งและผู้รับ เป็นต้น

ตัวอย่างเช่น เอเจนต์ A อยากจะทราบถึงค่าความจริงของประโยค X โดยที่เอเจนต์ B เป็นเอเจนต์ที่มี X อยู่ในฐานความรู้ และมี F เป็นเอเจนต์ตัวกลาง ถ้า A รู้ว่าควรจะส่ง query เกี่ยวกับ X ไปยัง B แล้ว ก็สามารถส่ง query ไปยัง B ได้โดยตรง ดังแสดงในรูปที่ 3.2 แต่ถ้า A ไม่สามารถรู้ว่าควรจะส่ง query ไปที่ให้ใคร (ที่จะมี X ในฐานความรู้) A จะต้องส่ง subscribe performative ไปยัง F เพื่อให้ค้นหาค่าความจริงของ X ให้ ถ้า B แจ้งให้ F ทราบว่ามันเชื่อว่า X เป็นจริงแล้ว F จะแจ้งข่าวสารนั้นไปยัง A อีกที ดังแสดงไว้ในรูปที่ 3.3



รูปที่ 3.2 A ส่ง query โดยตรง



รูปที่ 3.3 A ส่ง query ผ่าน Facilitator agent

4) **รูปแบบภาษา KQML**

การทำการสื่อสารจะประกอบด้วยขั้นตอนหลายระดับ เนื้อหาของข้อความที่ส่งเป็นเพียงส่วนประกอบหนึ่งของการสื่อสารเท่านั้น นั่นคือ เอเจนต์จะต้องทำการระบุถึงตำแหน่งเอเจนต์ที่ต้องการจะสื่อสารด้วย แล้วจึงส่งข้อความสื่อสารไปให้ซึ่งกำหนดในรูปแบบของข้อความ KQML

ข้อความ KQML จะประกอบด้วยส่วนต่างๆ คือ performative และ อาร์กิวเมนต์ต่างๆ ของ performative ซึ่งกำหนดเป็นคู่ของคีย์/ค่า (keyword/value) โดยอาร์กิวเมนต์ดังกล่าวเป็นข้อมูลที่มีความสำคัญและสามารถนำไปใช้ประโยชน์ในการสื่อสารได้ เช่น ชื่อ และที่อยู่ของเอเจนต์ผู้รับและเอเจนต์ผู้ส่ง, หมายเลขข้อความ, และสัญลักษณ์อื่นๆ นอกจากนี้ ยังประกอบด้วยข้อมูลที่ระบุถึงภาษาหรือ ontology ที่ใช้ ซึ่งรูปแบบของส่วนต่างๆ ในข้อความ KQML อาจมีได้หลากหลายรูปแบบขึ้นกับกลไกการส่งข้อความ KQML นั้น ตัวอย่างของข้อความ KQML ที่ทำการ query หาราคาของ IBM stock เป็นดังนี้

```
( ask-one
   : content (PRICE IBM ?price)
   : receiver stock-server
   : langauge LPROLOG
   : ontology NYSE-TICKS)
```

ในข้อความนี้ประกอบด้วย

- KQML performative คือ ask-one
- เนื้อหา คือ (price ibm ?price)
- ontology คือ nyse-ticks
- เอเจนต์ผู้รับทำหน้าที่เป็นเซิร์ฟเวอร์ คือ stock-server
- ภาษาที่ใช้ในการเขียน query คือ LPROLOG

ตัวอย่างของ communication performative ในภาษา KQML มีดังนี้

Basic query performatives :

- Evaluate, ask-if, ask-in, ask-one, ask-all, ...

Multi-response query performatives:

- stream-in, stream-all, ...

Response performatives :

- reply, sorry, ...

Generic informational performatives :

- tell, achieve, cancel, untell, unachieve, ...

Generator performatives :

- standby, ready, next, rest, discard, generator, ...

Capability-definition performatives :

- advertise, subscribe, monitor, import, export, ...

Networking performatives :

- register, unregister, forward, broadcast, route, ...

3.2.3 ภาษา DALI [15]

DALI เป็นภาษาเชิงตรรกศาสตร์ (Logic programming) ที่ออกแบบมาเพื่อทำการสื่อสารสำหรับเอเจนต์โดยเฉพาะ เอเจนต์ DALI คือโปรแกรมเชิงตรรกะที่สร้างขึ้นเพื่อทำการติดต่อสื่อสารกับสิ่งแวดล้อมในรูปแบบของเหตุการณ์ภายนอก (External event) เอเจนต์ DALI จะสามารถทำงานได้เมื่อมีเหตุการณ์ (Event) ต่างๆ เกิดขึ้น โดยเหตุการณ์เหล่านั้นแบ่งออกเป็น External

event, Internal event, Present event และ Past event จะสังเกตว่าเหตุการณ์และการทำงานต่างๆ ของเอเจนต์ DALI นั้นจะเกิดขึ้น ณ เวลาใดเวลาหนึ่ง ซึ่งเวลาดังกล่าวจะถูกบันทึกเก็บไว้ด้วย

1) **เหตุการณ์ภายนอก** (External event)

คือ เหตุการณ์ที่เกิดขึ้นจากสภาพแวดล้อมภายนอก เมื่อเอเจนต์ได้รับเหตุการณ์จากโลกภายนอก เอเจนต์สามารถตัดสินใจที่จะทำการตอบสนองต่อเหตุการณ์นั้นได้ด้วยการกำหนดกฎการตอบสนอง (Reaction rule) ซึ่งทำการประมวลผลแบบ forward reasoning ตัวอย่างเช่น bell_ringE :> open_the_doorA. หมายถึง เมื่อมีเหตุการณ์กระดิ่งหน้าบานดัง (bell_ringE) แล้ว เอเจนต์จะทำการเปิดประตู (open_the_door) เมื่อมีการตอบสนองต่อเหตุการณ์ภายนอกแล้ว เหตุการณ์นั้นจะกลายเป็นเหตุการณ์อดีต (Past event) ทันที

2) **เหตุการณ์ภายใน** (Internal event)

คือ เหตุการณ์ที่เกิดขึ้นภายในเอเจนต์ DALI เหตุการณ์นี้จะประกอบด้วยกฎ 2 ส่วน โดยส่วน แรกประกอบด้วยเงื่อนไขต่างๆ (ความรู้, เหตุการณ์อดีต, โพรซีเยอร์ และอื่นๆ) ที่เมื่อเป็นจริงแล้ว จะมีการกระทำการตอบสนองเกิดขึ้นตามกฎในส่วนที่สอง ดังตัวอย่างที่แสดงต่อไปนี้ เป็นสถานการณ์ที่เอเจนต์ทำการเตรียมซุป โดยจะต้องทำการตั้งไฟเป็นเวลา K นาที โดยที่ predicate ที่มี P ตามหลังนั้น หมายถึง เหตุการณ์อดีต (Past event) นั่นคือ เหตุการณ์ที่เกิดขึ้นและถูกบันทึกไว้แล้ว จากตัวอย่าง กฎในส่วนแรกกล่าวว่า การเตรียมน้ำซุปจะเสร็จเรียบร้อยก็ต่อเมื่อ เอเจนต์ต้องเปิดไฟ แล้วตั้งซุปเป็นเวลานาน K นาที เมื่อเตรียมน้ำซุปเสร็จเรียบร้อย เอเจนต์จะกระทำการตอบสนองต่อเหตุการณ์นี้ตาม กฎในส่วนที่สองคือเอาหม้อซุปลง แล้วปิดไฟ

```
soup_ready :- tuen_on_the_fireP, put_pan_on_the_stoveP :T,
              cooking_time(K), time_elapsed(T,K).
soup_readyI :> take_off_pan_from_stoveA, turn_off_the_fireA.
```

3) **เหตุการณ์ปัจจุบัน** (Present event)

เมื่อเอเจนต์ได้รับเหตุการณ์จากภายนอก เอเจนต์ไม่จำเป็นจะต้องตอบสนองต่อเหตุการณ์นั้นทันที โดยเอเจนต์สามารถวิเคราะห์เหตุผลของเหตุการณ์ก่อนทำการตอบสนองได้ ในสถานการณ์แบบนี้ เราเรียกเหตุการณ์ดังกล่าวว่าเหตุการณ์ปัจจุบัน (Present Event)

4) **การทำงาน** (Action)

เป็นวิธีการทำงานที่ส่งผลต่อสภาพแวดล้อมของเอเจนต์ ซึ่งอาจเป็นการตอบสนองต่อเหตุการณ์ภายนอก หรือ เหตุการณ์ภายในเอเจนต์ ใน DALI การทำงานจะมีเงื่อนไขเบื้องต้น (precondition) หรือไม่ก็ได้ โดยการทำงานจะถูกกำหนดโดยกฎการทำงาน (Action rule) ด้วย

ไวยากรณ์ action :< precondition และเป็นเช่นเดียวกับเหตุการณ์ภายนอกและภายใน การทำงานจะถูกบันทึกเป็นอดีตเมื่อมีการทำงานนั้นแล้ว

5) **เหตุการณ์อดีต** (Past event)

เป็นส่วนที่แสดงถึงความจำของเอเจนต์ โดยเอเจนต์สามารถกระทำงานในอนาคตได้ด้วยประสบการณ์จากเหตุการณ์ที่เคยเกิดขึ้น เหตุการณ์อดีต หรือ Past event นี้ จะเขียนลงท้ายด้วย P ตัวอย่างเช่น alarm_clock_ringsP เป็นเหตุการณ์ที่เอเจนต์ได้กระทำการตอบสนองไปแล้วและยังคงถูกบันทึกไว้ในความจำของเอเจนต์ โดยเหตุการณ์อดีตจะมีเวลา T ระบุไว้เมื่อเกิดเหตุการณ์นั้นขึ้น

3.2.3.1 ข้อความสื่อสารของ DALI

ดังที่กล่าวไปแล้วข้างต้นว่า ACL คือ เซตของ primitive และ rule ที่ทำให้เอเจนต์สามารถติดต่อสื่อสารกันได้ ซึ่งมาตรฐานของภาษาสำหรับสื่อสารนี้มีมากมาย หนึ่งในนั้นคือ FIPA ACL ที่มีการใช้งานอย่างแพร่หลาย เพื่อความสะดวก DALI จึงได้พัฒนาขึ้นจากการขยาย FIPA โดยข้อความที่ใช้สื่อสารมีโครงสร้าง ดังนี้

- receiver
- language
- ontology
- sender
- content

3.2.3.2 การกรองข้อความสื่อสารใน DALI

ในการทำงานจริงนั้น จะพบว่า การติดต่อสื่อสารระหว่างเอเจนต์นั้นต้องคำนึงถึงปัญหาด้านความปลอดภัย (security) ของข้อมูล ถ้าเอเจนต์ไม่มีการป้องกันปัญหาดังกล่าว อาจทำให้มีผลกระทบต่อการทำงาน จนถึงความรู้และกฎต่างๆ ของเอเจนต์ได้

วิธีแก้ปัญหาด้านความปลอดภัยใน DALI นั้น ทำโดยการกำหนดขอบเขตของปัญหา นั่นคือ เมื่อเอเจนต์ได้รับข้อความ เอเจนต์จะทำการตรวจสอบข้อความนั้นด้วยเงื่อนไขการรับข้อความ หากเงื่อนไขเป็นเท็จ ก็จะกำจัดข้อความนั้นออกโดยพิจารณาว่าเป็นข้อความที่ผิด

และในทำนองเดียวกัน สำหรับการส่งข้อความไปยังเอเจนต์ หรือสภาพแวดล้อมภายนอก ก็สามารถตรวจสอบข้อความที่จะส่งว่าสมควรส่งข้อความนั้นออกไปหรือไม่ โดยพิจารณาจากเงื่อนไขที่เอเจนต์กำหนด แต่ถึงอย่างไรก็ตามผู้ใช้ก็สามารถเลือกได้ว่า จะให้มีการตรวจสอบข้อความก่อนที่จะส่งออกไปยังภายนอกหรือไม่

3.2.3.3 การทำ Meta-reasoning

ในกรณีที่มีการติดต่อสื่อสารระหว่างเอเจนต์หลายๆ ตัว (Multi-agent system) ก็เป็นไปได้ที่เอเจนต์ทุกตัวไม่ได้คุยด้วยภาษาเดียวกัน และเป็นไปได้ที่เอเจนต์แต่ละตัวจะอธิบายถึงสิ่งเดียวกันด้วยคำพูดที่แตกต่างกัน ซึ่งหากเอเจนต์อื่นไม่สามารถเข้าใจความหมายของคำต่างๆ นั้นได้ย่อมทำให้เอเจนต์ทำงาน หรือตอบสนองเหตุการณ์ได้อย่างผิดพลาด แต่ DALI ได้ทำแก้ปัญหาดังกล่าวโดยอาศัยความสามารถของการคิดระดับเมต้า (meta-reasoning) ในภาษาเชิงตรรกะ ซึ่งใช้ ontology ประกอบการพิจารณา โดย ontology ใน DALI เป็นเสมือนพจนานุกรมของคำเหมือนต่างๆ ตัวอย่างเช่น ontology(bob, rain, water_falling_from_sky) เป็น ontology ของ Bob ที่บอกว่า rain กับ water_falling_from_sky มีความหมายเหมือนกัน

3.2.3.4 สถาปัตยกรรมการสื่อสารของภาษา DALI

จากที่กล่าวมาทั้ง สามารถสรุปได้ว่า สถาปัตยกรรมการสื่อสาร DALI ประกอบด้วยการทำงาน 3 ขั้นตอน คือ

1) การกรองข้อความสื่อสาร
2) การคิดที่ระดับเมต้า (Meta-reasoning)
3) การประมวลผลด้วยอินเตอร์พรีตเตอร์ของ DALI

นั่นคือ เมื่อเอเจนต์ได้รับเหตุการณ์ภายนอกแล้ว อินเตอร์พรีตเตอร์ของ DALI จะเรียกส่วนกรองข้อความสื่อสาร (Filter layer) เพื่อทำการตรวจสอบความปลอดภัยของข้อความหรือเหตุการณ์ที่ได้รับ หากผ่านการตรวจสอบในขั้นตอนนี้ได้ อินเตอร์พรีตเตอร์ของ DALI จะเรียกการทำงานที่ระดับเมต้าให้ทำความเข้าใจต่อเหตุการณ์นั้นโดยอัตโนมัติ หากเหตุการณ์ที่ได้รับนั้นเป็นเซตของเหตุการณ์ที่เอเจนต์รู้อยู่แล้วก็ไม่จำเป็นต้องผ่านกระบวนการคิดที่ระดับเมต้า (Meta reasoning) และเอเจนต์สามารถตอบสนองได้โดยตรง แต่ในทางกลับกัน หากเหตุการณ์ที่ได้รับ เอเจนต์ยังไม่เคยรู้จักมาก่อน อินเตอร์พรีตเตอร์ของ DALI ก็จะทำการคิดเพื่อค้นหากฎที่ใช้ตอบสนอง (Reactive rule) ที่เหมาะสมกับเหตุการณ์นั้นต่อไป

บทที่ 4
ภาพรวมของภาษา CL

ภาษา CL (Communication Language) เป็นภาษาที่นำไปใช้เพื่อสร้างเอเจนต์ที่มีความสามารถในการสื่อสาร โดยเอเจนต์สามารถทำงานคำสั่งได้ตามเวลาที่กำหนด อีกทั้งยังสามารถตอบสนองต่อเหตุการณ์ภายนอก หรือข้อความสื่อสารที่ได้รับจากเอเจนต์อื่นได้

CL ที่กล่าวถึงในงานวิจัยนี้ ไม่ได้หมายถึงเฉพาะรูปแบบการเขียนโปรแกรมของภาษาเท่านั้น แต่หมายรวมถึงกระบวนการประมวลผลโปรแกรมภาษา CL ซึ่งมีลักษณะการทำงานที่เป็นเอเจนต์ ดังนั้น นับจากนี้ไป เมื่อกล่าวถึง CL จะหมายความถึง เอเจนต์ซึ่งเป็นระบบที่รับโปรแกรมภาษา CL แล้วประมวลผลได้ตามที่กำหนด

เนื้อหาในบทนี้จะกล่าวถึงโครงสร้างโดยรวมของเอเจนต์ CL ในหัวข้อที่ 4.1 แล้วจึงกล่าวถึงสถาปัตยกรรมของ CL ในหัวข้อ 4.2 หลังจากนั้นจึงกล่าวถึงรูปแบบของโปรแกรมภาษา CL ในหัวข้อที่ 4.3 ตามด้วยวิธีการสร้างเอเจนต์ CL ในหัวข้อที่ 4.4 และแสดงถึงตัวอย่างการทำงานของเอเจนต์ ในหัวข้อที่ 4.5 เป็นลำดับสุดท้าย

4.1 โครงสร้างของเอเจนต์ CL

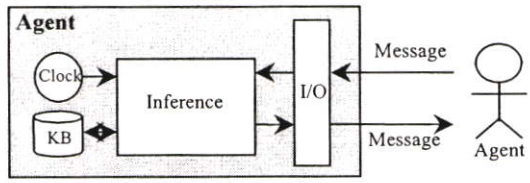
ความสามารถในการตอบสนองต่อเหตุการณ์ภายนอก ทำให้ CL มีการทำงานที่สนับสนุนต่อการติดต่อสื่อสาร อันเป็นลักษณะการทำงานที่เป็นเอเจนต์ โดยเอเจนต์ CL มีโครงสร้างดังที่แสดงไว้ในรูปที่ 4.1 ซึ่งประกอบด้วยส่วนต่างๆ ดังนี้

1) ส่วนวินิจฉัย (Inference) ทำหน้าที่ประมวลผลสารที่ได้รับจากเอเจนต์ภายนอกหรือตัวเอเจนต์เอง แล้วทำการตอบสนอง

2) ส่วนฐานความรู้ (Knowledge) เป็นส่วนเก็บความรู้ของเอเจนต์ เพื่อนำไปช่วยในการวินิจฉัยของระบบ

3) ส่วนอินพุท/เอาท์พุท (Input/Output) ทำหน้าที่รับและส่งสารไปสู่เอเจนต์ภายนอกหรือตัวเอเจนต์เอง

4) ส่วนนาฬิกา (Clock) เป็นนาฬิกาที่ใช้อ้างอิงในตัวเอเจนต์

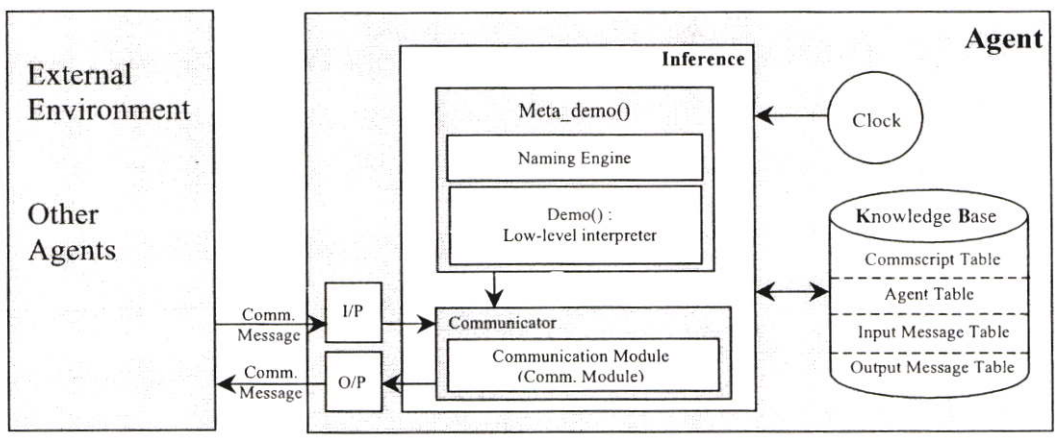


รูปที่ 4.1 โครงสร้างของเอเจนต์ CL

เอเจนต์ CL จะทำการรับสารผ่านส่วนอินพุท (Input) แล้วทำการประมวลผลสารนั้นด้วยส่วนวินิจฉัย (Inference) ซึ่งต้องมีการเก็บและดึงข้อมูลในฐานความรู้ (Knowledge Base) มาใช้ในการประมวลผล โดยผลลัพธ์ที่ได้จะถูกส่งกลับไปในรูปของสารผ่านส่วนเอาท์พุท (Output) ออกสู่ภายนอก

4.2 สถาปัตยกรรมของเอเจนต์ CL

ในหัวข้อที่แล้วได้กล่าวถึงโครงสร้างของเอเจนต์ CL จุดประสงค์เพื่อให้ผู้อ่านได้เข้าใจถึงภาพรวมของการทำงานตลอดจนส่วนประกอบหลักของเอเจนต์ สำหรับเนื้อหาในหัวข้อนี้จะต่อเนื่องจากหัวข้อที่แล้ว ซึ่งจะอธิบายถึงรายละเอียดของส่วนประกอบต่างๆ ของเอเจนต์ โดยสถาปัตยกรรมของเอเจนต์ CL สามารถแสดงได้ดังรูปที่ 4.2 อันประกอบด้วยส่วนต่างๆ ดังนี้



รูปที่ 4.2 สถาปัตยกรรมของเอเจนต์ CL

1) อินพุท/เอาท์พุท(Input/Output) ใช้รับและส่งสารกับภายนอก ซึ่งเอเจนต์สามารถเป็นทั้งผู้รับและผู้ส่งสารได้ในเวลาเดียวกัน(สื่อสารกับตัวเอง) อีกทั้งอาจมีอินพุทและเอาท์พุทได้มากกว่า 1 ช่องทาง

2) ส่วนวินิจฉัย (Inference) ทำหน้าที่ประมวลผลและตอบสนองต่อสารหรือข้อความที่ได้รับจากภายนอกตามที่ผู้ใช้กำหนดด้วยรูปแบบของสคริปต์ภาษา CL ซึ่งในการวินิจฉัยแต่ละครั้งอาจต้องอาศัยความรู้ที่มีในฐานความรู้ (KB) เพื่อใช้ค้นหาคำตอบต่างๆ ให้แก่เอเจนต์ ส่วนวินิจฉัยประกอบด้วยส่วนต่างๆ ได้แก่

- *ส่วน Communicator* ทำหน้าที่สื่อสาร รับ/ส่งข้อความสื่อสาร
- *ส่วน Naming Engine* ทำหน้าที่แปลงสัญลักษณ์ไปมาระหว่างระดับวัตถุและเมต้า โดยฟังก์ชัน objtoname() และ nametoobj()
- *ส่วน meta_demo()* ทำหน้าที่ประมวลผลคำสั่งตอบสนองที่ระดับเอเจนต์ (ส่วน demo () ใช้ประมวลผลคำสั่งที่เป็นโค้ดโปรแกรม)

3) ส่วนฐานความรู้ ใช้เก็บความรู้ของเอเจนต์ ประกอบด้วย

- *ส่วนตารางสคริปต์สื่อสาร (Commscript Table) หรือส่วนตารางสคริปต์ตอบสนองต่อเหตุการณ์ (Event-action script table)* ใช้บันทึกสคริปต์ที่ใช้ตอบสนองต่อข้อความสื่อสาร ซึ่งถือเป็นเหตุการณ์ที่ได้รับจากภายนอก
- *ตารางเอเจนต์ (Agent Table)* ใช้บันทึกชื่อและที่อยู่ของเอเจนต์ที่ทำการติดต่อมายังเอเจนต์สื่อสาร
- *ตารางข้อความนำเข้า (Input Message Table)* ใช้บันทึกเวลาและข้อความสื่อสารที่เอเจนต์ได้รับในแต่ละครั้ง
- *ตารางข้อความส่งออก (Output Message Table)* ใช้บันทึกเวลาและข้อความสื่อสารที่เอเจนต์ส่งออกไปภายนอก

4) ส่วน Clock เป็นนาฬิกาของเอเจนต์

4.3 รูปแบบภาษา CL

โปรแกรมภาษา CL จะประกอบด้วย 3 ส่วนหลักๆ ได้แก่

- การนิยามฟังก์ชัน (Function definition) ผู้ใช้สามารถสร้างฟังก์ชันขึ้นเพื่อใช้งานในโปรแกรมได้
- สคริปต์ตอบสนองต่อเหตุการณ์ (Event-action script) เป็นส่วนที่กำหนดการตอบสนองต่อเหตุการณ์ที่ได้รับจากภายนอก ประกอบด้วย ประโยคการตอบสนองเหตุการณ์ และ ประโยคชุดของการตอบสนองต่อเหตุการณ์ โดยประโยคการตอบสนองเหตุการณ์จะมีรูปแบบ Event → Action หมายถึง ถ้ามีเหตุการณ์ (Event) เกิดขึ้น ภาษา CL จะประมวลผลการทำงาน (Action) เพื่อตอบสนอง แต่สำหรับในวิทยานิพนธ์นี้ เหตุการณ์ภายนอกจะเน้นไปถึงข้อความสื่อสารโดยเฉพาะ ดังนั้นจากนี้ไป

เมื่อกล่าวถึงสคริปต์ตอบสนองต่อเหตุการณ์แล้ว จะหมายถึงสคริปต์ที่ใช้สำหรับตอบสนองต่อการสื่อสาร เรียกว่า สคริปต์สื่อสาร (Commscript) โดยปริยาย

- ประโยค (statement) ใช้กำหนดการทำงานต่างๆ ของเอเจนต์ ประกอบด้วย ประโยคกำหนดค่าตัวแปร, ประโยคคำสั่ง, ประโยคควบคุมลำดับการทำงาน และประโยคชุดคำสั่ง โดยผู้ใช้สามารถกำหนดช่วงเวลาการทำงานได้ด้วยรูปแบบ "**ช่วงเวลา : ประโยค**"

ในการเขียนโปรแกรมภาษา CL จะประกอบด้วยลิสต์ของกำหนดการนิยามฟังก์ชัน, สคริปต์ตอบสนองต่อเหตุการณ์ และประโยค

ตัวอย่างโปรแกรมภาษา CL

```
## Function definition ##
def display_meta_html(meta_html_data) :
      html_data = nametoobject(meta_html_data)
      return launchBrowser(html_data)

## Event-Action script ##
exist(tell_msg(?msg, ?snd, _, id, _)) →
     {res = display_meta_html(msg); send(Yes(res, agent("myself"),
snd, 5555, id))}

## Statement ##
<20:35:00|03/10/2004>: send(please_tell("index.html", agent("myself"),
agent("kmitl"), 0), 5555)
```

จากตัวอย่าง เป็นโปรแกรมภาษา CL ที่มีส่วนประกอบครบทั้ง 3 ส่วน ซึ่งในการเขียนโปรแกรมจริงๆ นั้น อาจไม่มีความจำเป็นต้องเป็นเช่นนั้น ขึ้นอยู่กับจุดประสงค์ในการเขียนโปรแกรม เช่น ถ้าต้องการเขียนโปรแกรมที่มีการทำงานโดยไม่สนใจต่อเหตุการณ์ภายนอก ก็ไม่จำเป็นต้องมีส่วนตอบสนองต่อเหตุการณ์ หรือในกรณีที่ต้องการเขียนโปรแกรมที่ใช้ตอบสนองต่อเหตุการณ์อย่างเดียว ก็ไม่จำเป็นต้องมีส่วนการทำงานของเอเจนต์ที่โปรแกรมโดยใช้ประโยคก็ได้

4.4 วิธีการสร้างเอเจนต์ CL

ในวิทยานิพนธ์นี้ ได้กำหนดโครงสร้างของเอเจนต์ CL ให้อยู่ในรูปแบบของคลาสที่มีชื่อว่า คลาส Agent และจากที่ได้กล่าวถึงส่วนประกอบต่างๆ ในหัวข้อก่อนหน้านี้ สามารถสรุปได้ว่า คลาส Agent จะประกอบด้วยแอททริบิวท์ (Attribute) ที่สำคัญต่างๆ ดังนี้

1) *Name* คือ ชื่อของเอเจนต์
2) *Input channel* คือ ช่องทางอินพุทของเอเจนต์ เพื่อกำหนดว่าเอเจนต์สามารถรับข้อความเข้ามาผ่านช่องทางและวิธีใดได้บ้าง
3) *Output channel* คือ ช่องทางเอาท์พุทของเอเจนต์ เพื่อกำหนดว่าเอเจนต์สามารถส่งข้อความออกไปผ่านช่องทางและวิธีใดได้บ้าง

4) *CL program* คือโปรแกรมที่เขียนในรูปแบบของภาษา CL ซึ่งใช้กำหนดการทำงาน รวมไปถึงการตอบสนองต่อเหตุการณ์หรือข้อความที่ได้รับจากภายนอกของเอเจนต์สื่อสาร

5) *Knowledge base* คือ ความรู้ของเอเจนต์

ดังนั้น คลาส Agent เขียนในรูปแบบภาษา Python ได้ ดังนี้

```
Class Agent :
   def __init__(self, name, input, output, cl_program, kb) : …
```

สำหรับการสร้างเอเจนต์ CL ในวิทยานิพนธ์นี้ ทำได้โดยการเขียนโปรแกรมภาษาไพธอนให้เรียกคลาส Agent มาใช้ด้วยวิธีการ import เพื่อใช้ในการสร้างอินสแตนส์ (instance) ของคลาส Agent แล้วสั่งให้อินสแตนส์นั้นทำงาน ซึ่งหมายถึง ให้เอเจนต์ทำการประมวลผลโปรแกรมภาษา CL นั่นเอง ตัวอย่างกระบวนการสร้างเอเจนต์สื่อสารซึ่งเขียนด้วยภาษาไพธอนเป็นดังนี้

```
import Agent
john = Agent("john",[5555,6666], [5555,6666], "cl_script.cl",
john_kb)
john.exist()
```

ในบรรทัดแรกแสดงการ import คลาส Agent แล้วจึงทำการสร้างอินสแตนส์ของคลาส Agent ชื่อ "john" ในบรรทัดที่สอง ซึ่งสามารถรับและส่งข้อความหรือเหตุการณ์ผ่านพอร์ต 5555 และ 6666 และจะทำงานตามโปรแกรมภาษา CL ในไฟล์ "cl_script.cl" โดยเอเจนต์ "john" จะเริ่มทำการประมวลผลตามโปรแกรมภาษา CL เมื่อเรียกฟังก์ชัน exist() ในบรรทัดที่สาม

4.5 ตัวอย่างการทำงานของเอเจนต์

เพื่อให้เกิดความเข้าใจในภาพรวมของงานวิจัย ในหัวข้อนี้จะขอเสนอตัวอย่างการทำงานของเอเจนต์สื่อสาร ดังนี้

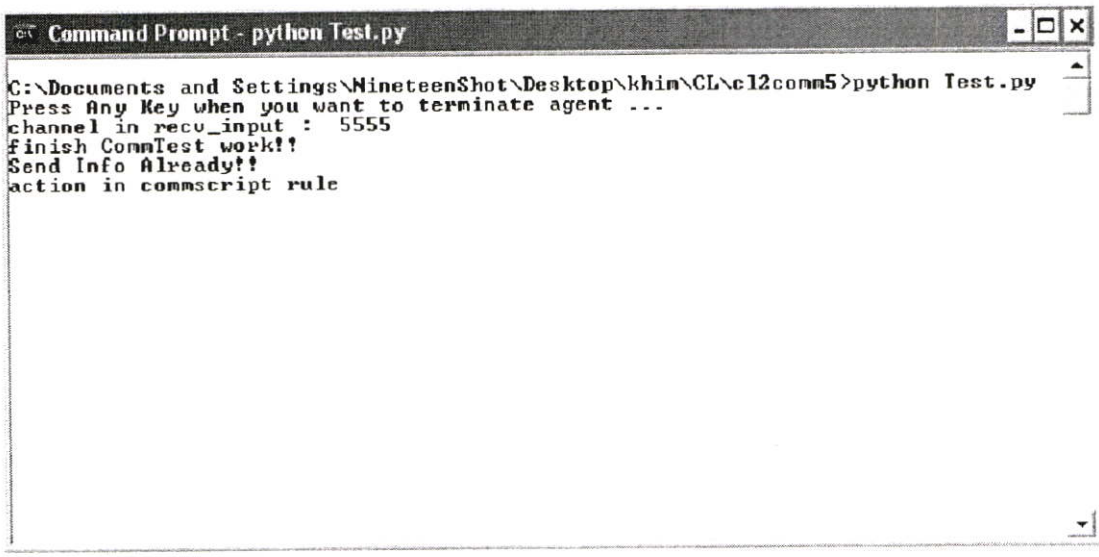
```
##  Initiate Agent (Test.py) ##
from Agent import *
if __name__ == '__main__' :
   test = Agent("test", [5555], [5555], "CommTest.cl", None)
   test.exist()

## CL Program (CommTest.cl)##
commscript :
   exist(please_tell_msg(?msg1,_,_,?id,_)) --> {
         result = find(msg1);send(tell('''result''', agent("myself"),
         agent("myself"),id),5555);print("Send Info Already!!");}
   exist(tell_msg(?msg,_,_,_,_)) --> {res = assert(msg);
         print("action in commscript rule");}
send(please_tell('''"testFile.txt"''', agent("myself"), agent("myself"), 0),
5555)
print("finish CommTest work!!")
```

ตัวอย่างนี้ แสดงถึงการทำงานของเอเจนต์ "test" ที่สามารถรับและส่งข้อความผ่านพอร์ต 5555 โดยเอเจนต์ไม่มีฐานความรู้ของตนเอง และได้กำหนดการทำงานด้วยไฟล์ "CommTest.cl" ซึ่งกำหนดคำสั่งการทำงานให้เอเจนต์ "test" ขอไฟล์ข้อมูลชื่อ "testFile.txt" จากตัวเอเจนต์เอง แล้วทำการพิมพ์ "finish CommTest work!!" ออกทางจอภาพ ในขณะเดียวกัน ก็กำหนดการตอบสนองต่อเหตุการณ์หรือข้อความสื่อสารไว้ในส่วนของ commscript ซึ่งกำหนดการตอบสนองไว้ 2 กรณีดังนี้

- ถ้าเอเจนต์ได้รับการขอไฟล์ (please_tell_msg) เข้ามา เอเจนต์จะค้นหาไฟล์นั้นแล้วส่งผลลัพธ์ที่ได้กลับไปยังผู้ขอด้วยคำสั่ง send แล้วจึงพิมพ์ "Send Info Already!!" ทางจอภาพ
- ถ้าเอเจนต์ได้รับไฟล์ข้อมูล (tell_msg) เอเจนต์จะเก็บข้อมูลนั้นไว้ แล้วจึงพิมพ์ "action in commscript rule" ออกทางจอภาพ

โดยผลที่ได้จากการทำงานของเอเจนต์ เป็นดังรูปที่ 4.3



รูปที่ 4.3 ผลการทดลองสร้างเอเจนต์ "test"

บทที่ 5
ภาษา CL

เนื้อหาในบทนี้จะกล่าวถึงโครงสร้างและไวยากรณ์ทั้งหมดของภาษา CL โดยเริ่มจากการอธิบายโครงสร้างภาษาในหัวข้อ 5.1 แล้วจึงแสดงไวยากรณ์ทั้งหมดของภาษา CL ในหัวข้อที่ 5.2 ตามด้วยตัวอย่างของภาษา CL เพื่อให้ผู้อ่านเห็นภาพรวมและมีความเข้าใจในภาษา CL มากขึ้นในหัวข้อที่ 5.3 หลังจากนั้น จึงกล่าวถึงองค์ประกอบต่างๆ ของภาษา CL โดยละเอียด เริ่มด้วยชนิดข้อมูลของภาษา CL ในหัวข้อที่ 5.4 ซึ่งส่วนมากจะเป็นชนิดข้อมูลเดิมที่ปรากฏในภาษาไพธอน โดยในวิทยานิพนธ์จะเน้นอธิบายเฉพาะชนิดข้อมูลที่เพิ่มเติมเข้ามา ได้แก่ ข้อมูลประเภทเวลา หลังจากนั้น ในหัวข้อที่ 5.5 จึงกล่าวถึงรายละเอียดของการเขียนโปรแกรมภาษา CL เช่น การกำหนดประโยคต่างๆ เป็นต้น สุดท้ายจึงกล่าวถึงโอเปอเรเตอร์และเอ็กซ์เพรสชั่นของภาษา CL ในหัวข้อที่ 5.6 และ 5.7 ตามลำดับ

5.1 โครงสร้างของภาษา CL

ภาษา CL เกิดจากการเพิ่มขยายภาษาไพธอน (Python) [1,2] ให้มีความสามารถในการประมวลผลเกี่ยวกับเวลาและการสื่อสาร อันเป็นแนวคิดใหม่ซึ่งไม่มีในภาษาคอมพิวเตอร์ปัจจุบัน ไวยากรณ์หลักๆ ในภาษา CL มีดังนี้

- **ประเภทข้อมูล** (Data Type) ประเภทข้อมูลในภาษา CL ส่วนใหญ่จะเป็นประเภทข้อมูลเดิมที่มีในภาษาไพธอน ได้แก่ Numbers, Boolean values, String, Lists, Tuples, Dictionary และ None ส่วนประเภทข้อมูลใหม่ที่เพิ่มเติม ได้แก่ Time
- **โอเปอเรเตอร์** (Operators) โอเปอเรเตอร์ที่ใช้ในภาษา CL แบ่งเป็น 4 ประเภทคือ โอเปอเรเตอร์ทางตรรกะ, โอเปอเรเตอร์ทางการเปรียบเทียบ, โอเปอเรเตอร์คำนวณเลขฐานสอง และโอเปเรเตอร์ทางคณิตศาสตร์ ซึ่งโอเปอเรเตอร์ทั้ง 4 มาจากภาษาไพธอน แต่ก็ได้เพิ่มเติมโอเปอเรเตอร์ที่ประมวลผลเกี่ยวกับเวลาเข้ามาด้วย
- **เอ็กซ์เพรสชั่น** (Expression) เป็นกลุ่มของโอเปอเรเตอร์และโอเปอแรนด์ ซึ่งเมื่อผ่านการประเมินค่า (Evaluation) จะให้ผลลัพธ์เป็นค่าออกมา ซึ่งค่าที่ได้สามารถนำไปใช้ในส่วนของประโยคได้ โดยค่าที่ได้จะเป็นข้อมูลประเภทใดนั้น ขึ้นกับประเภทโอเปอเรเตอร์และโอเปอแรนด์ที่ใช้ในเอ็กเพรสชั่นนั้นๆ
- **ประโยค** (Statement) ในโปรแกรมภาษา CL มีความแตกต่างจากในภาษาคอมพิวเตอร์ทั่วไป คือ มีการกำหนดช่วงเวลาทำงานให้กับทุกประโยคในรูปแบบของ "**ช่วงเวลา** : **ประโยค**" ซึ่ง

หมายถึง ประโยคจะทำงานในช่วงเวลานี้ ถ้าผู้เขียนโปรแกรมไม่ต้องการระบุช่วงเวลานี้ ก็จะได้ประโยคที่ใช้กันในภาษาคอมพิวเตอร์ทั่วไป ประโยคมีหลายแบบ คือ ประโยคสำหรับกำหนดค่าให้ตัวแปร, ประโยคคำสั่ง, ประโยคสำหรับควบคุมลำดับการทำงานในโปรแกรม และประโยคชุดคำสั่ง โดยอธิบายได้ดังนี้

- **ประโยคสำหรับการกำหนดค่าให้ตัวแปร** (Assignment Statement) การกำหนดค่าให้ตัวแปรใช้โอเปอเรเตอร์กำหนดค่า ('=') ด้วยรูปแบบ Var = Value

- **ประโยคคำสั่ง** เป็นคำสั่งที่ใช้สั่งงานในโปรแกรม สำหรับในภาษาไพธอน ประโยคคำสั่งมี 3 รูปแบบคือ function(), module.function() และ class.function() ซึ่งภาษา CL ณ ปัจจุบันสามารถรองรับรูปแบบประโยคคำสั่งในแบบหนึ่งและสองเท่านั้น ในอนาคตจะขยายให้ครอบคลุมแบบที่สาม

- **ประโยคสำหรับควบคุมลำดับการทำงานในโปรแกรม** (Control Statement) ใน CL มีประโยคสำหรับควบคุมลำดับการทำงานของโปรแกรมทั้งหมด 4 รูปแบบ คือ รูปแบบประโยค if, ประโยค while และประโยค for ซึ่งใช้ตามภาษาไพธอนทั้งหมด รวมทั้งประโยค repeat-until ที่ได้เพิ่มเติมเข้ามา

- **ประโยคชุดคำสั่ง** เกิดจากการนำประโยคแบบต่างๆ มารวมกัน ประโยคชุดคำสั่งแบ่งเป็น 4 แบบ คือ แบบ Sequence, Unordered, Alternative, Parallel และ Nested

- **การนิยามฟังก์ชัน** ใน CL ผู้ใช้สามารถนิยามฟังก์ชันขึ้นใหม่ได้ซึ่งมีรูปแบบเดียวกับในไพธอน

- **สคริปต์ตอบสนองต่อเหตุการณ์** (Event-action script) ใน CL ผู้ใช้สามารถกำหนดการตอบสนองต่อเหตุการณ์ที่ได้รับจากภายนอกด้วยสคริปต์ตอบสนองต่อเหตุการณ์ อันประกอบด้วย ประโยคการตอบสนองเหตุการณ์ และ ประโยคชุดของการตอบสนองต่อเหตุการณ์ ซึ่งการทำงานในส่วนนี้ยังไม่มีปรากฏในภาษาไพธอน

  - **ประโยคการตอบสนองเหตุการณ์** อยู่ในรูปแบบ

    เหตุการณ์ → การทำงาน

    หรือ Event → Action

    โดยที่ "เหตุการณ์" ในที่นี้ คือ Expression หรือ ประโยคคำสั่งที่ให้ค่าออกมาเป็นค่า Boolean ซึ่งในวิทยานิพนธ์นี้ พิจารณาเหตุการณ์ที่เป็นข้อความสื่อสาร (Communication message) ที่ได้รับจากภายนอกเป็นหลัก และ "การทำงาน" ในที่นี้ คือ ประโยคคำสั่งใดๆ สำหรับ "เหตุการณ์ → การทำงาน" หมายความว่า ไม่ว่าเมื่อไรก็ตามที่ "เหตุการณ์" (Event) ได้ค่า Boolean เป็น True ออกมา อินเตอร์พรีตเตอร์ภาษา CL จะลงมือกระทำ "การทำงาน" (Action) ทันที

  - **ประโยคชุดของการตอบสนองเหตุการณ์** เกิดจากการนำประโยคการตอบสนองต่อเหตุ

การณ์มารวมกันเข้าเป็นชุด มีอยู่ 6 แบบ คือ แบบ Independent Repeatable and Partial, Independent Repeatable and Total, Sequence, Unordered, One และ Nested ดังนั้น จากที่กล่าวมาทั้งหมด โปรแกรมภาษา CL จะประกอบด้วยลิสต์ของประโยค, คำนิยามฟังก์ชัน และ/หรือ สคริปต์ตอบสนองต่อเหตุการณ์

5.2 ไวยากรณ์ภาษา CL

หลังจากที่ได้อธิบายโครงสร้างของภาษา CL ในหัวข้อที่แล้ว ต่อไปจะกล่าวถึงไวยากรณ์ทั้งหมดของภาษา CL อย่างละเอียด ซึ่งแสดงไว้ในรูปแบบ BNF (Backus Naur Form) ดังนี้

```
1  <cl-program> ::= <define-function> | <comm-script> | <action-statement>

2  <define-function> ::= def <id> ( [<id> (, <id>)*] ) : <action-statement>+
   [return [<expression>]]

3  <comm-script> ::= commscript : <comm_grp>+
4  <comm_grp> ::= <comm_sequence> | <comm_unordered> | <comm_indp_total> |
   <comm_one> | <comm_stmt>
5  <comm_sequence> ::= comm_sequence : <comm_stmt>+
6  <comm_unordered> ::= comm_unordered : <comm_stmt>+
7  <comm_indp_total> ::= comm_independent_total : <comm_stmt>+
8  <comm_one> ::= comm_one : <comm_stmt>+
9  <comm_stmt> ::= <comm_msg> → <action-statement>+
10 <comm_msg> ::= [<interval_time> :] <comm_expr>
11 <comm_expr> ::= exist(<comm_type>)

12 <comm_type> ::= <mode> (<msg>,<snd>,<recv>,<msg_id>,<msg_ret_id>)
13 <mode>  ::=  tell_msg  |  query_msg  |  request_msg  |  delete_msg  |
   please_tell_msg  |  please_query_msg  |  please_request_msg  |
   please_delete_msg | yes_msg | no_msg
14 <msg> ::= <string> | <id> | <retid> | _
15 <snd> ::= <string> | <id> | <retid> | _
16 <recv> ::= <string> | <id> | <retid> | _
17 <msg_id> ::= <number> | <id> | <retid> | _
18 <msg_ret_id> ::= <number> | <id> | <retid> | _

19 <action-statement> ::= <module_stmt> | <statement> | <comment>
20 <module_stmt> ::= import <id>
21 <statement> ::= [<interval_time> :] <statement_type>
22 <statement_type> ::= <assign_stmt> | <control_stmt> | <group_stmt>
23 <assign_stmt> ::= <left_value> (= | += | -= | *= | /= | %= | ^^= | <<= |
   >>= | &= | |= | ^= ) <expression>
24 <left_value> ::= <id> | <access>
25 <control_stmt> ::= <if_stmt> | <while_stmt> | <for_stmt> | <repreat_stmt>

26 <if_stmt> ::= if <expression> : <action-statement>+ (elif <expression> :
   <action-statement>+)* [else : <action-statement>+ ]
27 <while_stmt> ::= while <expression> : <action-statement>+
28 <for_stmt> ::= for <id> in <expression> : <action-statement>+
29 <repreat_stmt> ::= repeat : <action-statement>+ until <expression>

30 <group_stmt> ::= (sequence | unordered | alternative | parallel) :
   <action- statement>+
31 <expression> ::= <unary_expr> | <binary_expr> | <bitwise_expr> |
   <shift_expr> | <extend_expr> | <shrink_expr> | <boolean_expr> |
   <logic_expr> | <group_expr> | <term>
32 <unary_expr> ::= (- | + | ~) <expression>
33 <binary_expr> ::= <expression> (+ | - | * | / | ** | %) <expression>
34 <bitwise_expr> ::= <expression> (& | | | ^) <expression>
```

```
35 <shift_expr> ::= <expression> (<< | >>) <expression>
36 <extend_expr> ::= <expression> (<- | ->) <expression>
37 <shrink_expr> ::= <expression> (>- | -<) <expression>
38 <boolean_expr> ::= <expression> ( < | > | <= | >= | == | != | mi | m | oi
   | o | di | d | si| s | fi | f ) <expression>
39 <logic_expr> ::= <expression> ( and | or ) <expression>
40 <group_expr> ::= ( <expression> )

41 <term> ::= <number> | <function> | <id> | <retid> | <string> | <meta> |
   <time> | <list> | <tuple> | <dictionary> | <access> | <boolean> | <none>
42 <number> ::= <int_number>|<float_number>|<long_number>|<complex_number>
43 <int_number> ::= (<digit>)+
44 <float_number> ::= (<digit>)+ . (<digit>)+
45 <long_number> ::= (<digit>)+ (l|L)
46 <complex_number> ::= (<digit>)+ [. <digit>+] (+|-) <digit>+ [. <digit>+]
   (j|J)
47 <function> ::= <id> ([<expression> (, <expression>)*])
48 <id> ::= (<upper>|<lower>) (<upper>|<lower>|<digit>|_)*
49 <retid> ::= ? (<upper>|<lower>) (<upper>|<lower>|<digit>|_)*
50 <string> ::= "(<upper>|<lower>|<digit>|<alpha>|<special>)* "
51 <meta> ::= ''' (<upper>|<lower>|<digit>|<alpha>|<special>)* '''
52 <time> ::= <point> | <duration> | <interval>
53 <point> ::= ((0|1)(<digit>)|2(0...3)):(0...5)(<digit>):(0...5)(<digit>)| ((0...2)
   <digit>)|3(0|1)) / ((1...9)|1(0...2)) / (<digit>)^4
54 <duration> ::= <digit>+ [.<digit>+] (min|d|h|m|s)(, <digit>+ [.<digit>+]
   (min|d|h|m|s))*
55 <interval> ::= < (<point>|_) , (<point>|_) >
56 <list> ::= [ [<expression> (, <expression>)*] ]
57 <tuple> ::= ( [<expression> (, <expression>)*] )
58 <dictionary> ::= { [ <expression> : <expression> ( , <expression> :
   <expression>)* ] }
59 <access> ::= <id> [ <index> [: <index>] ]
60 <index> ::= <int_number> | <id> | <string>
61 <boolean> ::= true | false
62 <none> ::= none

63 <upper> ::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
64 <lower> ::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z
65 <digit> ::= 0|1|2|3|4|5|6|7|8|9
66 <alpha> ::= !|$|%|&|*|+|-|.|/|<|>|=|?|@|_|~
67 <special> ::= "|#|'|(|)|,|\|^
```

ไวยากรณ์ของภาษา CL แสดงด้วย BNF โดยสัญลักษณ์ที่แสดงถึง nonterminal จะถูกกำหนดภายใต้เครื่องหมาย <> ส่วน terminal จะแสดงด้วยอักษรตัวเข้ม นอกจากนี้ ยังมีสัญลักษณ์ที่ใช้กำหนดจำนวนการเกิดของส่วนต่างๆ อันได้แก่ การไม่เกิดหรือเกิดเพียง 1 ครั้ง (zero or one) การไม่เกิดหรือเกิดตั้งแต่ 1 ครั้งขึ้นไป (zero or more) การเกิดตั้งแต่ 1 ครั้งขึ้นไป (one or more) และการเกิดทั้งหมด n ครั้ง ซึ่งแสดงด้วยสัญลักษณ์ [<nonterminal>], <nonterminal>*, <nonterminal>+ และ <nonterminal>^n ตามลำดับ

ภาษา CL ประกอบด้วยส่วนประกอบหลักๆ 3 ส่วน ได้แก่ การนิยามฟังก์ชัน (<define-function>), สคริปต์ตอบสนองการสื่อสาร (<comm-script>) และส่วนการทำงานคำสั่ง (<action-statement>) ดังที่แสดงในบรรทัดที่ 1 โปรแกรมภาษา CL จะต้องประกอบด้วยส่วนประกอบนี้อย่างน้อย 1 ส่วน และต้องกำหนดส่วนประกอบตามลำดับ เริ่มจาก <define-function>, <comm-script> และ <action-statement>

การนิยามฟังก์ชัน (<define-function>) มีไวยากรณ์ตามบรรทัดที่ 2 โดยผู้เขียนสามารถกำหนดให้มีการคืนค่าจากฟังก์ชันหรือไม่ก็ได้ ตัวอย่างเช่น def add(x,y) : return (x+y) เป็นการนิยามฟังก์ชันที่ทำการบวกเลข 2 จำนวน และคืนค่าผลบวกนั้นออกมา ซึ่งมีรายละเอียดในหัวข้อ 5.5.4

สำหรับส่วนสคริปต์ตอบสนองการสื่อสาร (<comm-script>) แสดงไว้ในบรรทัด 3-11 ซึ่งจะประกอบด้วยลิสต์ของการตอบสนองการสื่อสารในแบบต่างๆ ทั้งที่เป็นคำสั่งตอบสนอง (<comm_stmt>) และแบบบทสนทนา รูปแบบไวยากรณ์ของคำสั่งตอบสนอง แสดงไว้ในบรรทัดที่ 9 ซึ่งเป็นลักษณะ exist(comm_msg) → action นั่นคือ เมื่อได้รับข้อความสื่อสารแล้วให้ตอบสนองด้วยการทำคำสั่งต่างๆ ซึ่งข้อความสื่อสารจะแบ่งออกเป็นโหมด เช่น tell, query, request เป็นต้น และข้อความจะประกอบด้วยส่วนสาร (<msg>), ผู้ส่ง (<snd>), ผู้รับ (<recv>), หมายเลขสาร (msg_id), หมายเลขสารที่ตอบสนอง (msg_ret_id) ตามไวยากรณ์ที่แสดงในบรรทัดที่ 12-18 เช่น exist(tell_msg(?msg,_,_,_,_)) → assert(msg) เป็นต้น สำหรับการตอบสนองแบบบทสนทนาในบรรทัดที่ 4-8 จะประกอบด้วยการตอบสนองแบบ sequence, unordered, independent repeatable and total, one ซึ่งจะประกอบด้วยลิสต์ของคำสั่งตอบสนอง (<comm_stmt>) ที่อธิบายไปข้างต้นอีกที รายละเอียดของการทำงานในส่วนนี้จะอยู่ในหัวข้อที่ 5.5.5

สำหรับส่วนคำสั่งการทำงาน (<action-statement>) หลักๆ แล้วจะประกอบด้วยประโยค (statement) ซึ่งผู้เขียนสามารถกำหนดช่วงเวลาสำหรับการทำคำสั่งได้ตามบรรทัดที่ 21 ตัวอย่างเช่น <10:00:00|21/12/2004, 11:00:00|21/12/2004> : play(movie) เป็นการกำหนดให้ทำการเปิดภาพยนตร์ movie ภายในเวลา 10.00-11.00 น. ของวันที่ 21 ธันวาคม 2004 เป็นต้น ประโยคในภาษา CL จะแบ่งเป็นประโยคกำหนดค่า(<assign_stmt>), ประโยคควบคุมลำดับการทำงาน (<control_stmt>), ประโยคคำสั่ง และประโยคชุดคำสั่ง (<group_stmt>) ซึ่งปรากฏในหัวข้อ 5.5.3

ไวยากรณ์ของประโยคกำหนดค่าแสดงในบรรทัดที่ 23 ซึ่งเป็นการกำหนดค่าที่ได้จากเอ็กซเพรสชัน (<expression>) ให้กับตัวแปร เช่น x = 10 เป็นต้น ส่วนประโยคควบคุมลำดับการทำงานในโปรแกรมจะประกอบด้วย ประโยค if, for, while และ repeat ดังแสดงไว้ในบรรทัดที่ 25-29 ตัวอย่างเช่น repeat : print(x); x+=1; until x>5 กำหนดให้พิมพ์ค่า x แล้วเพิ่มค่า x ขึ้นทีละ 1 จนกว่า x จะมีค่ามากกว่า 5 เป็นต้น สำหรับส่วนประโยคชุดคำสั่ง (<group_stmt>) จะแบ่งเป็นแบบ sequence, unordered, alternative และ parallel ซึ่งมีไวยากรณ์ตามที่กำหนดในบรรทัดที่ 30

ในบรรทัดที่ 31-40 แสดงเอ็กซ์เพรสชันต่างๆ ใน CL ตามประเภทของโอเปอเรเตอร์อันได้แก่ โอเปอเรเตอร์ทางตรรกะ, โอเปอเรเตอร์ทางการเปรียบเทียบ, โอเปอเรเตอร์คำนวณเลขฐานสอง,

โอเปอเรเตอร์ทางคณิตศาสตร์ และโอเปอเรเตอร์สำหรับเวลา ตัวอย่างเช่น 3+2 เป็นเอ็กซ์เพรสชันของการทำโอเปอเรเตอร์ทางคณิตศาสตร์ ซึ่งค่าที่ประเมินออกมาได้คือ 5 เป็นต้น โดยในวิทยานิพนธ์นี้จะเน้นอธิบายถึงโอเปอเรเตอร์สำหรับเวลาในหัวข้อที่ 5.6 ซึ่งเป็นส่วนที่เพิ่มมาจากภาษาไพธอน

ในบรรทัดที่ 41-66 แสดงเทอมต่างๆ ใน CL ซึ่งประกอบด้วย ตัวเลข (<number>) เช่น 12 เป็นเลขจำนวนเต็ม (<int_number>), ฟังก์ชัน (<function>) เช่น add(2,3), ตัวแปร (<id>) เช่น id1 , ตัวแปรแบบคืนค่า (<retid>) เช่น ?id1, สายอักขระ (<string>) เช่น "Hi", ข้อมูลเมต้า (<meta>) เช่น "'12'" เป็นข้อมูลเมต้าของจำนวนเต็ม 12, จุดเวลา (<point>) เช่น 10:00:00|12/12/2004, ระยะเวลา (<duration>) เช่น 2d, ลิสต์ (<list>) เช่น [1,"john"], ทัปเปิล (<tuple>) เช่น (1,"john"), ดิกชันนารี (<dictionary>) เช่น {1:"john",2:"mary"}, บูลีน (<boolean>) เช่น true และ ข้อมูลเปล่า (<none>) โดยขยายมาจากภาษาไพธอน ยกเว้นข้อมูลประเภทเวลาซึ่งจะกล่าวถึงรายละเอียดในหัวข้อที่ 5.4

5.3 ตัวอย่างโปรแกรมภาษา CL

เพื่อให้เข้าใจในโครงสร้างของภาษา CL จึงขอยกตัวอย่างโปรแกรมภาษา CL ดังรูปที่ 5.1 ซึ่งเป็นการเขียนโปรแกรมเพื่อขอไฟล์ "index.html" จากเอเจนต์ "kmitl" ณ เวลา 20.35 น. วันที่ 3 ตุลาคม ค.ศ. 2004 (4) จากตัวอย่างได้มีการกำหนดขั้นตอนการสื่อสาร (2) ไว้ว่า หากได้รับสารจากเอเจนต์ใดๆ แล้วให้เรียกฟังก์ชัน display_meta_html() ที่นิยามไว้ (1) เพื่อแสดงสารที่ได้ออกทางจอภาพ แล้วจึงส่งข้อความตอบรับกลับไปยังผู้ส่ง (3)

```
def display_meta_html(meta_html_data) :                                          (1)
    html_data = nametoobject(meta_html_data)
    return launchBrowser(html_data)

commscript :                                                                     (2)
    exist(tell_msg(?msg, ?snd, _, id, _)) →
        {res = display_meta_html(msg); send(Yes(res, agent("myself"), snd, id), 5555)}   (3)

<20:35:00|03/10/2004> : send(please_tell("index.html", agent("myself"), agent("kmitl"), 0), 5555)   (4)
```

รูปที่ 5.1 ตัวอย่างการเขียนโปรแกรมภาษา CL

5.4 ชนิดข้อมูลในภาษา CL

ดังที่กล่าวไว้ข้างต้นว่า ประเภทข้อมูลในภาษา CL นั้นประกอบด้วยประเภทข้อมูลเดิมที่มีในภาษาไพธอน ได้แก่ Numbers, Boolean values, String, Lists, Tuples, Dictionary และ None รวมกับประเภทข้อมูลชนิดใหม่ที่กำหนดเพิ่มเติมเข้ามา ได้แก่ ข้อมูลประเภท Time (เวลา) โดยจะกล่าวถึงรายละเอียดในหัวข้อถัดไป สำหรับชนิดข้อมูลอื่น เช่น Numbers ได้อธิบายไปแล้วในบทที่ 2 ที่ว่าด้วยภาษาไพธอน

5.4.1 ข้อมูลประเภทเวลา (Time)

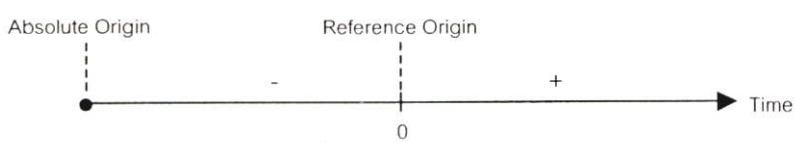
เป็นข้อมูลที่ใช้แสดงเวลาในภาษา CL ซึ่งแบ่งได้ 3 ประเภท ได้แก่ จุดเวลา(Point), ช่วงเวลา (Interval) และระยะเวลา(Duration) รายละเอียดของข้อมูลประเภทเวลา เป็นดังนี้

5.4.1.1 ความหมายของเวลา

ในวิทยานิพนธ์นี้ เราได้ทำการศึกษาและแยกแยะประเภทของเวลา ตลอดจนนำเสนอการประมวลผลและวินิจฉัยเวลาทุกประเภท เพื่อสื่อความหมายการทำงานที่เกี่ยวกับเวลาในภาษา CL รวมทั้งนำผลไปใช้ในการสร้างอินเตอร์พรีตเตอร์ภาษา CL ที่ใช้ประมวลผลเวลาและการทำคำสั่งที่เกี่ยวข้องกับเวลา

การศึกษาเพื่อเข้าใจธรรมชาติของเวลามีมาตั้งแต่ยุคโบราณ จาก [11] กล่าวถึงแนวคิดเกี่ยวกับเวลาไว้หลายรูปแบบ ได้แก่ เวลาสัมพันธ์กับการเปลี่ยนแปลงอย่างเป็นจังหวะ เช่น การเกิดกลางวัน-กลางคืน, เวลาสัมพันธ์กับการเคลื่อนที่และความเร็ว และเวลาสัมพันธ์กับเหตุการณ์ในอดีต เป็นต้น

ในวิทยานิพนธ์นี้จึงขอเสนอความคิดที่ว่า เวลาเกิดจากการดำเนินไปของกระบวนการ (Process) อันหนึ่งอย่างต่อเนื่อง เมื่อไรที่เกิดมีกระบวนการ ก็จะเกิดเวลาขึ้น แต่ถ้าไม่มี เวลาก็ไม่ปรากฏ เนื่องจากการดำเนินไปของกระบวนการมีความต่อเนื่อง จึงสมมุติให้เวลามีความต่อเนื่องด้วย โดยให้มีค่าเพิ่มขึ้นโดยต่อเนื่องอยู่ตลอดไม่ขาดตอน จึงอาจอธิบายได้โดยเส้นลูกศรในระบบ co-ordinate โดยให้เป็นแกนของเวลา ดังรูปที่ 5.2 ซึ่งถือว่าเป็นแกนที่เพิ่มเติมจากแกน xyz ในระบบเรขาคณิต ซึ่งอาจถือว่าเป็นแกนที่ 4 หรือมิติที่ 4

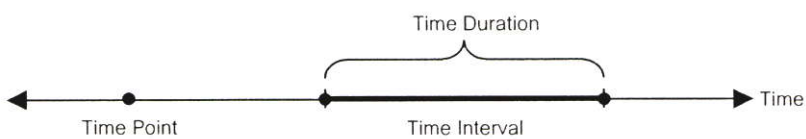


รูปที่ 5.2 เส้นแกนเวลา

ดังนั้นเวลาจะมีค่าเพิ่มขึ้นเรื่อยๆ เมื่อนับจากจุดเริ่มต้นในอุดมคติของการเกิดเวลา (Absolute Origin) แต่ในความเป็นจริงนั้น เราไม่สามารถรู้ถึงจุดกำเนิดของเวลาที่แท้จริงได้ มนุษย์จึงได้กำหนดจุดเริ่มต้นสมมุติของเวลาขึ้นมาเพื่อง่ายต่อการอ้างอิงเวลาเท่านั้น และขอเรียกจุดเริ่มต้นเวลานี้ว่า "จุดเริ่มอ้างอิง"(Reference Origin) ซึ่งในแต่ละอารยธรรมอาจมีจุดเริ่มอ้างอิงที่ต่างกันตามลัทธิความเชื่อทางศาสนา เช่น จุดเริ่มอ้างอิงตามระบบคริสตศักราช และระบบพุทธศักราช เป็นต้น ดังนั้น เวลาที่สนใจเมื่อเทียบกับจุดเริ่มอ้างอิงนี้อาจมีค่าเป็นลบได้ เช่น เมื่อกล่าวถึงเวลาก่อนคริสตศักราช (B.C.) เป็นต้น

จุดต่างๆ บนเส้นแกนเวลานั้น แสดงการอ้างอิงเวลา ณ ขณะใดขณะหนึ่ง จุดนี้เรียกว่า ***จุดเวลา (Time Point)*** โดยจุดเวลาที่อยู่ขวามือของจุดเริ่มอ้างอิง (Reference Origin) มีค่าเป็นบวก และจุดที่อยู่ซ้ายมือของจุดเริ่มอ้างอิง มีค่าเป็นลบ เช่น จุดเวลาที่ 15:00:00 น. วันที่ 10 มิ.ย. พ.ศ. 2546 เป็นจุดเวลาที่เป็นบวกเมื่อเทียบกับจุดเริ่มอ้างอิงตามระบบพุทธศักราช เป็นต้น

ในกรณีที่มีการลากเส้นเชื่อมต่อระหว่าง 2 จุดเวลา จากจุดเวลาที่น้อยกว่าซึ่งเรียกว่าจุดเริ่มต้น ไปยังจุดเวลาที่มีค่ามากกว่าซึ่งเรียกว่าจุดสิ้นสุด จะทำให้เกิด ***ช่วงเวลา(Time Interval)*** ซึ่งประกอบด้วยจุดเวลาหลายๆ จุด ที่เรียงต่อกันอย่างต่อเนื่องไม่ขาดสายจากจุดเริ่มไปจนถึงจุดสิ้นสุด โดย**ความยาวของช่วงเวลา**หรือของเส้นตรงนี้ ถือว่าเป็น ***ระยะเวลา(Time Duration)*** จะสังเกตว่า ในกรณีที่จุดเริ่มต้นของช่วงเวลาอยู่ที่ตำแหน่งจุดเริ่มอ้างอิง(Reference Origin) ค่าของจุดสิ้นสุดของช่วงเวลาจะมีค่าเท่ากับระยะเวลาของช่วงเวลานั้น เช่น ณ เวลา 9:00:00 น. วันที่ 17 มิถุนายน พ.ศ. 2546 จะมองได้ว่าเป็น จุดเวลานี้ห่างจากจุดอ้างอิงเป็นระยะเวลา 2546 ปี 7 เดือน 17 วัน และ 9 ชั่วโมง ได้เช่นกัน ในทำนองเดียวกัน จะได้ว่าจุดเริ่มอ้างอิง หมายถึง จุดเวลาที่มีระยะเวลาเป็นศูนย์เมื่อเทียบกับตำแหน่งอ้างอิง หรือตัวมันเอง ดังแสดงในรูปที่ 5.3



รูปที่ 5.3 แสดงเวลาประเภทต่างๆ บนแกนเวลา

จากแนวคิดเรื่องเวลาที่ได้อธิบายไปแล้ว หัวข้อต่อไปจะกล่าวถึงรูปแบบของเวลาทั้ง 3 ประเภท คือ จุดเวลา, ช่วงเวลา และระยะเวลา

5.4.1.2 **รูปแบบของเวลา**

1. **รูปแบบจุดเวลา** (Time Point) จุดเวลาสามารถกำหนดด้วยรูปแบบดังนี้

1) รูปแบบทางคณิตศาสตร์ของจุดเวลา

ในทางคณิตศาสตร์ จุดเวลาจะถูกแสดงให้อยู่ในรูปของหน่วยเวลาที่เล็กที่สุดเท่าที่นัยสำคัญจะมีผลกระทบต่อการทำงาน (ในทางทฤษฎีควรจะมีค่าน้อยที่สุดเท่าที่จะทำได้) ซึ่งเริ่มต้นนับจำนวนหน่วยเวลานั้นตั้งแต่จุดเริ่มอ้างอิง ในที่นี้กำหนดให้เป็นหน่วยวินาที เพื่อง่ายต่อการสั่งงานคอมพิวเตอร์ เช่นจุดเวลา ณ วันที่ 1 มกราคม ค.ศ. 0001 เวลา 10.00 น. หมายถึง จุดเวลา ณ วินาทีที่ 36000 ในรูปแบบทางคณิตศาสตร์ (เวลาผ่านไปแล้ว 10 ชั่วโมง) เมื่อจุดอ้างอิงคือ วันที่ 1 มกราคม ค.ศ. 0001 เวลา 00.00 น. เป็นต้น จะสังเกตว่า การกำหนดจุดเวลาด้วยรูปแบบนี้ เป็นรูปแบบพื้นฐานที่เหมาะสำหรับการคำนวณโดยเฉพาะ ถ้าเป็นรูปของเดือน ปี การเพิ่มของเดือน ปีจะไม่สม่ำเสมอ เนื่องจากบางเดือนมี 28 วัน บางเดือนมี 30 วัน เป็นต้น

2) รูปแบบจุดเวลาที่ใช้ในชีวิตประจำวัน

เป็นการกำหนดจุดเวลาในรูปแบบของ วัน เดือน ปี ชั่วโมง นาที และวินาที แทนที่จะกำหนดในหน่วยวินาทีเพียงหน่วยเดียว เช่น จุดเวลา ณ วินาทีที่ 360000 หมายถึงจุดเวลา ณ วันที่ 1 มกราคม ค.ศ. 0001 เวลา 10.00 น. เป็นต้น รูปแบบนี้จะเหมาะกับการแสดงผลหรือสื่อสารกับคน

3) รูปแบบจุดเวลาในภาษา CL

เป็นการผสมผสานรูปแบบของจุดเวลาใน 2 แบบข้างต้น สามารถกำหนดจุดเวลาด้วยรูปแบบ ดังนี้

hh : minmin : ss | dd / mm / yyyy

โดยที่ h, min, s, d, m, y แทน ชั่วโมง, นาที, วินาที, วันที่, เดือน และ ปี ตามลำดับ โดยกำหนดให้ใช้จุดเริ่มอ้างอิง ณ ตำแหน่ง epoch หรือวันที่ 1 มกราคม ค.ศ. 1970 ดังที่กำหนดไว้ในภาษาไพธอน เช่น 11:00:00|28/08/2002 หมายถึง ณ เวลา 11.00 น. ของวันที่ 28 สิงหาคม ค.ศ. 2002 ในรูปแบบจุดเวลาในชีวิตประจำวัน หรือหมายถึง จุดเวลา ณ วินาทีที่ 1030507200 เมื่อเทียบกับจุดเริ่มอ้างอิง (epoch) ในรูปแบบทางคณิตศาสตร์ เป็นต้น

รูปแบบในภาษา CL ข้างต้นสอดคล้องกับรูปแบบจุดเวลาที่ใช้ในชีวิตประจำวัน แต่ในการประมวลผลจุดเวลา อินเตอร์พรีตเตอร์จะแปลงจุดเวลาในรูปแบบนี้ให้เป็นรูปแบบคณิตศาสตร์ก่อนแล้วจึงทำการคำนวณ จากนั้น ผลที่ได้ซึ่งอยู่ในรูปแบบทางคณิตศาสตร์จะถูกแปลงกลับให้อยู่ในรูปเดิมเพื่อการแสดงผล

2. **รูปแบบช่วงเวลา** (Time Interval) สามารถกำหนดเป็นรูปแบบต่างๆ ได้ดังนี้

1) รูปแบบทางคณิตศาสตร์ของช่วงเวลา

ช่วงเวลา ประกอบด้วย จุดเวลาหลายๆ จุดที่เรียงต่อกันอย่างต่อเนื่องจากจุดเริ่มต้นจนถึงจุดสิ้นสุด ในทางคณิตศาสตร์ จึงกำหนดให้ช่วงเวลาอยู่ในรูปแบบของเซตต่อเนื่องที่มีสมาชิกเป็นจุดเวลา คือ [P_s , P_e] โดยที่ P_s คือ จุดเวลาเริ่มต้นซึ่งมีค่าน้อยกว่า P_e จุดเวลาสิ้นสุด หรือ [P_s , $P_s + d$] โดย d คือระยะเวลาจาก P_s และ d>0 หรือสามารถเขียนช่วงเวลาในรูปของเซตต่อเนื่อง $\{ t \mid P_s \leq t \leq P_e \}$ ได้ด้วย เช่น ช่วงเวลาตั้งแต่ เวลา 11.00 น. วันที่ 28 สิงหาคม ค.ศ. 2002 ถึง เวลา 12.00 น. วันที่ 28 สิงหาคม ค.ศ. 2002 หมายถึง [1030507200 , 1030510800] หรือ $\{ t \mid 1030507200 \leq t \leq 1030510800 \}$ ในรูปแบบทางคณิตศาสตร์

2) รูปแบบช่วงเวลาในภาษา CL

ในภาษา CL รูปแบบช่วงเวลา ประกอบด้วยจุดเวลาเริ่มต้น และจุดเวลาสิ้นสุดของช่วงเวลา โดยแสดงไว้ภายในเครื่องหมาย <...> ดังนี้

$< P_s , P_e >$

โดยที่ P_s และ P_e คือ ข้อมูลชนิดจุดเวลาแสดงจุดเวลาเริ่มต้นและสิ้นสุดตามลำดับ เช่น <11:00:00|28/08/2002, 12:00:00|28/08/2002> หมายถึง ช่วงเวลาตั้งแต่ เวลา 11.00 – 12.00 น. ของวันที่ 28 สิงหาคม ค.ศ. 2002 สาเหตุที่ในภาษา CL เลือกใช้เครื่องหมาย < > แทน [ ] เนื่องจาก เครื่องหมาย [ ] ได้ถูกใช้กับข้อมูลประเภท List ในภาษาไพธอนแล้ว

ในกรณีที่โปรแกรมเมอร์กำหนดช่วงเวลา $< P_s , P_e >$ ที่ไม่เหมาะสมเข้าไปในโปรแกรมภาษา CL ด้วยการกำหนดให้ P_e เกิดก่อน P_s จะทำให้เกิดความผิดพลาดขึ้น แม้ว่าการกำหนดดังกล่าวจะถูกต้องที่ระดับไวยากรณ์ (Syntax) คือประกอบด้วยจุดเวลา 2 จุด แต่ไม่ถูกต้องที่ระดับความหมาย (Semantic) ที่กำหนดว่าจุดเวลาเริ่มต้น (P_s) ต้องเกิดขึ้นก่อนจุดเวลาสิ้นสุด (P_e)

3. **รูปแบบระยะเวลา** (Time Duration) ระยะเวลามีรูปแบบต่างๆ ดังนี้

1) รูปแบบทางคณิตศาสตร์ของระยะเวลา

รูปแบบของระยะเวลาในทางคณิตศาสตร์ จะคล้ายคลึงกับรูปแบบของจุดเวลาในทางคณิตศาสตร์ คือจะใช้หน่วยวินาทีซึ่งเป็นหน่วยเวลาที่เล็กที่สุดในการกำหนดระยะเวลา เช่นระยะเวลา 10 ชั่วโมง หมายถึง ระยะเวลา 36000 วินาที ในทางคณิตศาสตร์

2) รูปแบบระยะเวลาในภาษา CL ในภาษา CL ผู้เขียนสามารถกำหนดรูปแบบของระยะเวลาได้ดังนี้

$Number_1$ $TimeUnit_1$, $Number_2$ $TimeUnit_2$, $Number_3$ $TimeUnit_3$, ...

โดยที่ Number คือ เลขจำนวนเต็ม หรือ เลขทศนิยม และ TimeUnit คือ หน่วยของเวลา ซึ่งประกอบด้วย หน่วยวัน (d), ชั่วโมง (h), นาที (min) และวินาที (s) เช่น 2d หมายถึง ระยะเวลานาน 2 วัน นอกจากนี้ยังสามารถกำหนดช่วงเวลาที่มีหลายหน่วยได้ โดยใช้เครื่องหมาย "," คั่น เช่น 3d,4h หมายถึง ระยะเวลานาน 3 วัน 4 ชั่วโมง เป็นต้น การกำหนดรูปแบบระยะเวลาในภาษา CL นั้น ไม่อนุญาตให้ระบุระยะเวลาในหน่วยเดือน หรือหน่วยปี เนื่องจากในแต่ละเดือนมีจำนวนวัน (ระยะเวลา) ไม่เท่ากัน

5.5 การโปรแกรมด้วยภาษา CL

ในหัวข้อนี้ จะกล่าวถึงรายละเอียดของการเขียนโปรแกรมด้วยภาษา CL ในส่วนต่างๆ เริ่มจากวิธีการเขียนโปรแกรม, คำที่ใช้ในโปรแกรม จนถึงรายละเอียดของความหมายและวิธีการใช้ประโยค, การนิยามฟังก์ชัน และสคริปต์ตอบสนองต่อเหตุการณ์ ตามลำดับ

5.5.1 วิธีการเขียนโปรแกรมภาษา CL

5.5.1.1 กลุ่มอักขระ (Character set) ในภาษา CL ประกอบด้วย

1) ตัวอักษรภาษาอังกฤษ (Alphabetic characters) ทั้งขนาดเล็กและใหญ่รวม 52 ตัว ได้แก่ a, b, ..., z, A, B, ..., Z
2) ตัวเลข (numeric digits) 10 ตัว ได้แก่ 0, 1, ..., 9
3) อักขระพิเศษ (special characters) ได้แก่ _ & | ^ > < = ~ + - * / % () ! ; , : {} []

5.5.1.2 รูปแบบการเขียนโปรแกรม

รูปแบบการเขียนโปรแกรมในภาษา CL นั้น จะคล้ายคลึงกับรูปแบบการเขียนโปรแกรมของภาษาไพธอน ซึ่งมีการใช้ย่อหน้า, การขึ้นบรรทัดใหม่ และการเขียนคอมเมนต์ ดังนี้

1) *การใช้ย่อหน้า (indenting)* ในภาษา CL ได้ทำการกำหนดสโคปของซอร์สโค้ดโดยใช้การย่อหน้าแทนสัญลักษณ์เช่นเดียวกับภาษาไพธอน ซึ่งต่างจากภาษาอื่นๆ เช่น ภาษา C และ Java ใช้วงเล็บปีกกา และภาษา Pascal ใช้ begin end เป็นตัวกำหนดสโคปของซอร์สโค้ด
2) *การขึ้นบรรทัดใหม่* ใช้สำหรับบอกตำแหน่งสิ้นสุดของการเขียนประโยค (statement) แต่ละประโยคเหมือนกับภาษาไพธอน แทนที่จะลงท้ายเครื่องหมาย ";" ดังที่พบทั่วไปในภาษา Pascal, Java

3) *คอมเมนต์ (Comment)* ใช้สำหรับอธิบายโปรแกรมเพิ่มเติม โดยไม่นำไปประมวลผลรวมกับประโยคต่างๆ ในภาษา CL กำหนดให้สามารถเขียนคอมเมนต์ได้ภายใต้เครื่อง "#" เช่นเดียวกับในภาษาไพธอน

5.5.2 **คำที่ใช้ในภาษา** CL

คำที่ใช้ในโปรแกรม อาจจะเป็นคำที่กำหนดไว้ก่อนแล้วอย่างคีย์เวิร์ด (Language Keywords) ของภาษา หรือเป็นคำที่โปรแกรมเมอร์เป็นคนสร้างขึ้น (Identifier)

5.5.2.1 Identifier

โปรแกรมเมอร์จะกำหนด identifier ให้กับตัวแปร(variable), ฟังก์ชัน (function) และตัวแปรแกรมเองได้

ชื่อ หรือ identifier อาจกำหนดด้วยอักษรภาษาอังกฤษทั้งหมด หรืออาจมีตัวเลขหรือเครื่องหมาย "_" เข้าไปด้วยก็ได้ แต่จะต้องเริ่มต้นด้วยอักษรภาษาอังกฤษเท่านั้น เช่น name, gender, student_name, name1 เป็นต้น สำหรับการกำหนดชื่อ หรือ identifier นั้นก็ควรมีการสื่อความหมายได้ดี เช่น การตั้งชื่อตัวแปรที่เก็บค่าที่บอกอายุว่า age เป็นต้น

5.5.2.2 **คำสงวน** (Reserved words)

คำสงวนในภาษา CL แบ่งเป็น คีย์เวิร์ด(Keywords) และ บิวท์อินฟังก์ชั่น(Built-in function)

1) คีย์เวิร์ด (Keywords) คีย์เวิร์ดใน CL ส่วนหนึ่งนำมาจากภาษาไพธอน นอกจากนี้ยังได้เพิ่มคีย์เวิร์ดที่ใช้สำหรับเทียบเวลา และกำหนดชุดคำสั่งเข้าไว้ด้วย คีย์เวิร์ดในภาษา CL มีดังนี้

'def', 'elif', 'else', 'for', 'if', 'return', 'while', 'to', 'repeat', 'until', 'sequence', 'unordered', 'parallel', 'alternative', 'import', 'and', 'or', 'not', 'mi', 'm', 'oi', 'o', 'di', 'd', 'si', 's', 'fi', 'f', 't', 'independent_rep_part', 'independent_total', 'one'

2) บิวท์อินฟังก์ชั่น(Built-in function) เช่นเดียวกับคีย์เวิร์ด คือได้มีการเพิ่มเติมบิวท์อินฟังก์ชันที่ใช้สำหรับการสื่อสารนอกเหนือจากบิวท์อินฟังก์ชันในไพธอน บิวท์อินฟังก์ชันใน CL มีดังนี้

'print', 'abs', 'divmod', 'round', 'pow', 'real', 'imag', 'len', 'min', 'max', 'append', 'extend', 'count', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort', 'clear', 'has_key', 'keys', 'update', 'values', 'open', 'read', 'readline', 'readlines', 'write', 'writelines', 'close', 'tell', 'query', 'request', 'delete', 'please_tell', 'please_query', 'please_request', 'please_delete',

'yes', 'no', 'deny', 'dontknow', 'ignore', 'find', 'send', 'assert', 'demo', 'agent', 'tell_msg', 'query_msg', 'request_msg', 'delete_msg', 'please_tell_msg', 'please_query_msg', 'please_request_msg', 'please_delete_msg'

5.5.3 ประโยค (Statement)

ประโยค (Statement) เป็นส่วนประกอบหลักของโปรแกรมภาษา CL โดยผู้เขียนจะใช้ประโยคเพื่อกำหนดการทำงานให้กับโปรแกรม ซึ่งในภาษา CL ได้มีการกำหนดการเขียนประโยค (statement) ไว้หลายประเภท ได้แก่ ประโยคสำหรับกำหนดค่าให้ตัวแปร, ประโยคคำสั่ง, ประโยคสำหรับควบคุมลำดับการทำงานในโปรแกรม และ ประโยคชุดคำสั่ง โดยอธิบายรายละเอียดได้ดังนี้

5.5.3.1 ประโยคสำหรับการกำหนดค่าให้ตัวแปร (Assignment Statement)

โปรแกรมเมอร์สามารถกำหนดค่าต่างๆ ให้กับตัวแปร ผ่านโอเปอเรเตอร์ "=" รูปแบบการใช้งานของประโยคประเภทนี้คือ

<variable-identifier> = <expression>

ตัวอย่างเช่น

```
anInt = 4
aString = "hello"
aFloat = -1.1111 * (5.0 ** 2)
anotherString = "shop" + "ping"
aList = [123, "world", 5+4j]
```

ในภาษา CL ได้กำหนดให้สามารถนำโอเปอเรเตอร์ = เข้าไปใช้รวมกับโอเปอเรเตอร์ที่ใช้ในการคำนวณ (Arithmatic operation) ได้แก่ โอเปอเรเตอร์ +=, -=, *=, /=, **=, %= ซึ่งใช้ในกรณีที่มีการกำหนดค่าที่คำนวณได้ใหม่ให้กับตัวแปรเดิมที่มีอยู่ (reassign) ตัวอย่างเช่น a = a + 1 หรือสามารถเขียนได้อีกแบบว่า a += 1

5.5.3.2 ประโยคคำสั่ง

เป็นคำสั่งที่ใช้สั่งงานในโปรแกรม ซึ่งภาษา CL ณ ปัจจุบัน สามารถรองรับรูปแบบประโยคคำสั่งแบบ function() และ module.function() เท่านั้น และในอนาคตจะขยายให้ครอบคลุมคำสั่งแบบclass.function()

1) **แบบ function()** เป็นประโยคคำสั่งที่ใช้เรียกฟังก์ชันให้ทำงานต่างๆ ตามที่กำหนดไว้ในฟังก์ชัน ฟังก์ชันในภาษา CL แบ่งเป็น 2 แบบ ได้แก่

- *Built-in function* เป็นฟังก์ชันที่กำหนดไว้ภายในภาษา CL อยู่แล้ว เช่น ฟังก์ชัน abs, print เป็นต้น
- *Defined function* เป็นฟังก์ชันที่ถูกกำหนดโดยโปรแกรมเมอร์

รูปแบบของประโยคคำสั่งเป็นดังนี้

<function-identifier>(<argument1>, <argumentN>)

เช่น abs(-11) เป็นการเรียกฟังก์ชัน abs เพื่อหาค่าสัมบูรณ์ (absolute) ของอาร์กิวเมนต์ซึ่งกำหนดให้เป็น –11 เป็นต้น

2) **แบบ** module.function() มีการทำงานเช่นเดียวกับแบบแรก แตกต่างกันเพียงฟังก์ชันที่ถูกเรียกนั้นไม่ได้ปรากฏในโปรแกรมปัจจุบัน หากแต่เป็นการเรียกฟังก์ชันจากโปรแกรมหรือโมดูลอื่นมาใช้แทนด้วยการใช้คำสั่ง import

5.5.3.3 ประโยคสำหรับควบคุมลำดับการทำงานในโปรแกรม (Control Statement)

ใน CL มีประโยคสำหรับควบคุมลำดับการทำงานของโปรแกรมทั้งหมด 4 รูปแบบ คือ รูปแบบประโยค if, ประโยค while, ประโยค for และประโยค repeat-until ซึ่งประโยคในสามแบบแรกใช้รูปแบบตามภาษาไพธอนทั้งหมด

1) ประโยค if (if statement)

เป็นการควบคุมลำดับแบบเลือก (Selective) ของโปรแกรม หมายถึง ในโปรแกรมสามารถกำหนดให้มีการเลือกการทำงานอย่างใดอย่างหนึ่ง เมื่อเงื่อนไขที่กำหนดไว้ในโปรแกรมเป็นจริง รูปแบบของประโยค if เป็นดังนี้

```
if expression1 :
   expr1_true_statement
        [ elif expression2 :
   expr2_true_statement
          :
elif expressionN :
   exprN_true_statement ]
        [ else :
   none_of_the_above_statement ]
```

ในการกำหนดประโยค if นั้น โปรแกรมเมอร์สามารถสร้างทางเลือกของการทำงานของโปรแกรมได้มากกว่า 2 ทางเลือก (เลือกที่จะทำประโยคในประโยค if หรือไม่ทำ) ด้วยประโยค elif และ else ซึ่งประโยค elif อาจมีได้มากกว่า 1 ประโยค ต่างจากประโยค else ที่มีได้เพียง 1 ประโยค แต่ถ้าต้องการสร้างเส้นทางการทำงานเพียง 2 เส้นทาง ก็ไม่จำเป็นต้องมีประโยค elif และ else

*2) **ประโยค** while (while statement)*

เป็นการควบคุมลำดับแบบวนรอบ (Iterative) ของโปรแกรม ที่มีการตรวจสอบเงื่อนไข (expression) ก่อนประมวลผลประโยค (statement) ทุกครั้ง หรือเรียกว่า test-before loop รูปแบบของประโยค while เป็นดังนี้

```
while expression :
   statement
```

ประโยค while เป็นประโยคที่ใช้กำหนดให้มีการทำงานประโยค (statement) ซ้ำ ได้เมื่อเงื่อนไข (expression) ที่กำหนดไว้ในโปรแกรมเป็นจริง และจะหยุดการทำซ้ำเมื่อเงื่อนไขเป็นเท็จ

*3) **ประโยค** for (for statement)*

เป็นการควบคุมลำดับแบบวนรอบ (Iterative) ของโปรแกรม ที่มีการใช้อินเด็กซ์กำหนดจำนวนของการวนรอบ หรือเรียกว่า indexed loop รูปแบบของประโยค for เป็นดังนี้

```
for variable in sequence :
   statement
```

ประโยค for เป็นประโยคที่ใช้กำหนดให้มีการทำงานประโยค (statement) ซ้ำ ที่มีจำนวนการวนรอบที่แน่นอน

*4) **ประโยค** repeat-until (repeat-until statement)*

เป็นการควบคุมลำดับแบบวนรอบ (Iterative) ของโปรแกรม ที่มีการตรวจสอบเงื่อนไข (expression) หลังการประมวลผลประโยค (statement) หรือเรียกว่า test-after loop รูปแบบของประโยค repeat-until เป็นดังนี้

```
repeat :
   statement
until expression
```

ประโยค repeat-until เป็นประโยคที่มีการประมวลผลประโยค (statement) ก่อนที่จะตรวจสอบเงื่อนไข (expression) ที่กำหนด หากเงื่อนไขเป็นเท็จ จะทำการประมวลผลประโยคซ้ำ แต่ถ้าเงื่อนไขเป็นจริง ก็จะหยุดการวนรอบ นั่นคือ การประมวลผลประโยคจะดำเนินไปเรื่อยๆ จนกว่าเงื่อนไขจะเป็นจริง

5.5.3.4 **คำสั่งการทำงานในรูป ช่วงเวลา : การทำงาน** (Time interval : Action)

ในภาษา CL ผู้เขียนได้มีการเพิ่มความสามารถในการกำหนดช่วงเวลาให้กับการทำงาน ซึ่งยังไม่ปรากฏในภาษาคอมพิวเตอร์อื่นๆ จุดประสงค์เพื่อเพิ่มความยืดหยุ่นในการเขียนโปรแกรม ทั้งยังสนับสนุนให้โปรแกรมเมอร์พัฒนาโปรแกรมได้อย่างสอดคล้องกับการทำงานในชีวิตประจำวัน

ทำให้โปรแกรมที่เขียนด้วยภาษา CL สามารถเขียนและเข้าใจได้ง่าย แม้ผู้อ่านไม่มีความรู้ด้านการเขียนโปรแกรมคอมพิวเตอร์มากนัก

จะสังเกตได้ว่า การทำงานหรือการทำกิจกรรมทุกอย่างของคนนั้น ล้วนแล้วแต่เกี่ยวข้องกับเวลาทั้งสิ้น ไม่ว่าจะเป็นจุดเวลาเริ่มต้นและจุดเวลาสิ้นสุดของการทำงานนั้น ๆ หรือกล่าวได้ว่า การทำงานคำสั่งย่อมทำในช่วงเวลาเสมอ ในภาษา CL จึงมีการกำหนดรูปแบบคำสั่งการทำงานหรือรูปแบบประโยคที่การทำงานกำหนดอยู่ภายใต้ช่วงเวลา คือมีการกำหนดช่วงเวลาทำงานให้กับทุกประโยค ดังนี้

$[P_s, P_e]$: statement

หมายถึง การทำ statement จะเริ่มต้น ณ จุดเวลา P_s และสิ้นสุด ณ จุดเวลา P_e โดย statement หมายถึงประโยคแบบต่างๆ ที่ได้อธิบายไปแล้ว และ $[P_s , P_e]$ เป็นข้อมูลชนิดช่วงเวลา

ในภาษา CL จะใช้สัญลักษณ์ < > แทน [ ] ตัวอย่างเช่น

<11:00:00|21/10/2004, 12:00:00|21/10/2004> แทนช่วงเวลาตั้งแต่เวลา 11.00 น. ของวันที่ 21 ตุลาคม 2004 ถึงเวลา 12.00 น. วันเดียวกัน

นอกจากนี้ จุดเวลาเริ่มต้นและจุดเวลาสิ้นสุดสามารถใช้เป็น expression ใดๆ ที่ให้ค่าจุดเวลาออกมาได้ นอกเหนือจากการกำหนดโดยใช้ข้อมูลชนิดจุดเวลาโดยตรง เช่น ถ้ากำหนดให้ x = 11:00:00|21/10/2004 แล้ว ช่วงเวลาที่อธิบายไว้ข้างต้นสามารถเขียนได้เป็น < x, x+1h > ซึ่งแสดงถึงช่วงเวลาที่มีจุดเวลาเริ่มต้นและสิ้นสุดห่างกันเป็นระยะเวลา 1 ชั่วโมง นั่นเอง

1. ประเภทของคำสั่งการทำงาน

เพื่อให้ผู้เขียนโปรแกรมภาษา CL มีความยืดหยุ่นในการกำหนดช่วงเวลาให้กับคำสั่ง จึงได้แบ่งคำสั่งการทำงานออกเป็น 3 แบบ แสดงเป็นตารางที่ 5.1

1) คำสั่งการทำงานแบบระบุเวลา เป็นคำสั่งที่ผู้เขียนโปรแกรมต้องการกำหนดเวลาที่แน่นอนในการเริ่มต้นและสิ้นสุดการทำคำสั่งนั้น เช่น ต้องการให้คอมพิวเตอร์เริ่มดาวน์โหลดไฟล์ ณ เวลา 10.00 น. และสิ้นสุดการดาวน์โหลด ณ เวลา 11.00 น. เป็นต้น

2) คำสั่งการทำงานแบบกึ่งระบุเวลา เป็นคำสั่งที่ผู้เขียนโปรแกรมต้องการกำหนดเวลาที่แน่นอนสำหรับการเริ่มต้นหรือสิ้นสุดในการทำคำสั่ง คำสั่งการทำงานแบบกึ่งระบุเวลาจึงแบ่งเป็น 2 แบบ คือคำสั่งการทำงานที่ระบุเวลาเริ่มต้น และคำสั่งการทำงานที่ระบุเวลาสิ้นสุดแต่เพียงอย่างเดียว ส่วนจุดเวลาที่เหลือที่ไม่ได้ระบุจะถูกกำหนดโดยระบบเอง เช่น ต้องการให้ส่งไฟล์เสร็จ ณ เวลา 10.00 น. เป็นคำสั่งการทำงานที่ระบุเวลาสิ้นสุด ซึ่งจะเริ่มทำงานต่อเนื่องจากคำสั่งก่อนหน้า เป็นต้น

3) **คำสั่งการทำงานแบบไม่ระบุเวลา** เป็นคำสั่งที่ผู้เขียนโปรแกรมกำหนดขึ้นโดยไม่ต้องการระบุเวลาเริ่มต้นและสิ้นสุดในการทำคำสั่ง แต่ปล่อยให้เวลาเริ่มต้นถูกกำหนดโดยเวลาสิ้นสุดของการทำคำสั่งก่อนหน้า ส่วนเวลาสิ้นสุดถูกกำหนดตามระยะเวลาที่ระบบใช้ในการทำคำสั่งนั้น

คำสั่งแบบนี้จะเหมือนกับคำสั่งในภาษาคอมพิวเตอร์ทั่วไปที่ผู้เขียนไม่ต้องการระบุเวลาเริ่มและเวลาสิ้นสุดของแต่ละคำสั่ง แต่ให้เวลาถูกกำหนดในขณะที่คอมพิวเตอร์ทำคำสั่งนั้นๆ

ตารางที่ 5.1 แสดงคำสั่งการทำงานในแบบต่างๆ

ประเภทคำสั่ง	รูปแบบทั่วไป	อธิบาย	ตัวอย่างใน CL
1. แบบระบุเวลา	[P_s, P_e] : statement	เริ่มทำงาน ณ เวลา P_s และ สิ้นสุดการทำงาน ณ เวลา P_e	<10:00:00|10/7/2003 , 10:10:00|10/7/2003> : Download('p.mp3')
2. แบบกึ่งระบุเวลา	[_, P_e] : statement	เริ่มทำงานถัดจากคำสั่งก่อนหน้า และ สิ้นสุดการทำงาน ณ เวลา P_e	< _,10:10:00|10/7/2003> : Download('p.mp3')
	[P_s, _] : statement	เริ่มทำงาน ณ เวลา P_s โดยไม่ระบุเวลาสิ้นสุดการทำงาน	<10:00:00|10/7/2003, _ > : Download('p.mp3')
3. แบบไม่ระบุเวลา	[_ , _] : statement	เริ่มทำงานถัดจากคำสั่งก่อนหน้า โดยไม่ระบุเวลาสิ้นสุดการทำงาน	<_ , _>:Download('p.mp3') หรือ Download('p.mp3')

2. ตัวอย่างการใช้คำสั่งการทำงานในโปรแกรม

ในโปรแกรมภาษา CL ผู้เขียนโปรแกรมสามารถกำหนดช่วงเวลาให้กับคำสั่ง โดยที่ช่วงเวลาดังกล่าวสามารถกำหนดโดยใช้ตัวแปร หรือโดยการคำนวณจากเอ็กซ์เพรสชั่นในโปรแกรมได้ ดังตัวอย่างต่อไปนี้

```
X = 11:00:00|21/9/2003
<X, X+10min> : download('hello.mp3')
```

หมายถึง ให้ทำคำสั่ง download นาน 10 นาที ตั้งแต่จุดเริ่มต้น ณ เวลา X หรือถูกกำหนดเป็นเวลา 11.00 น. วันที่ 21 กันยายน 2003

```
for X in range(0,24) :
   < X:00:00 , _ > : alert()
```

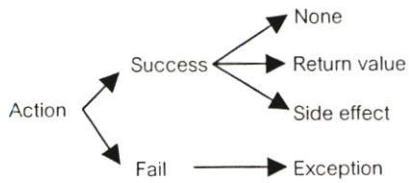
หมายถึง ให้โปรแกรมเตือนทุกๆ ชั่วโมง

3. การทำ Action

เมื่อทำการประมวลผลคำสั่งนั้นแล้ว ผลที่เกิดขึ้นจะเป็นได้ใน 2 แบบ คือ

- ผลการทำ Action ที่ทำได้สำเร็จ (Success) ซึ่งหมายถึง การประมวลผลตามคำสั่งการทำงานได้อย่างครบถ้วน ซึ่งอาจมีการส่งผลกลับหรือไม่ก็ได้ คือ ถ้าส่งผลกลับจะเป็นลักษณะที่ผลที่ได้เป็นค่าข้อมูล (Value) หรือเป็นผลกระทบ (Side effect) เช่นการแสดงผลหน้าจอ เป็นต้น ส่วนถ้าไม่ส่งผลอะไรกลับให้ถือว่าสิ่งที่ส่งกลับคือ None

- ผลการทำ Action ที่ประสบความล้มเหลว (Failed) หมายถึง การประมวลผลตามคำสั่งนั้นได้ไม่สมบูรณ์ ซึ่งไม่ว่าจะด้วยสาเหตุใดก็ตาม ระบบจะจัดการกับผลลัพธ์นี้ในลักษณะของ Exception ซึ่งวิธีการจัดการกับ Exception กำหนดได้เองโดยผู้เขียนโปรแกรม ผลที่เกิดจากการทำ Action สามารถสรุปได้ดังรูปที่ 5.4



รูปที่ 5.4 สรุปผลที่เกิดจาก Action

5.5.3.5 **ประโยคชุดคำสั่ง** (Compound statement)

ในภาษา CL ได้มีการกำหนดประโยครูปแบบหนึ่ง ซึ่งเกิดจากการนำประโยคแบบต่างๆ มารวมกัน เรียกว่า **ประโยคชุดคำสั่ง** ซึ่งแบ่งได้เป็น 5 แบบ คือ แบบ Sequence, Unordered, Alternative, Parallel และ Nested

*1) **ชุดคำสั่งแบบ** Sequence (Sequence compound statement)*

เป็นการทำงานตามคำสั่งทีละคำสั่งอย่างเป็นลำดับ โดยคำสั่งถัดไปจะทำงานได้ เมื่อคำสั่งก่อนหน้าได้ทำงานเสร็จแล้วเท่านั้น ซึ่งมีรูปแบบดังนี้

sequence :
$[s_1,e_1]$: statement$_1$
$[s_2,e_2]$: statement$_2$
:
$[s_n,e_n]$: statement$_n$

หรือ

$[s_1,e_1]$: statement$_1$
$[s_2,e_2]$: statement$_2$
:
$[s_n,e_n]$: statement$_n$

โดยที่ s_i และ e_i เป็นจุดเวลาเริ่มและสิ้นสุดของแต่ละ statement$_i$ โดยที่ $s_1 < e_1 < s_2 < e_2 < \ldots < s_n < e_n$ ชุดคำสั่งแบบ sequence สามารถแบ่งได้เป็น 2 แบบ คือ

1. **ชุดคำสั่งแบบ** Sequence **ที่ต่อเนื่อง** (Continuous Sequence) คือ sequence ที่ซึ่ง statement หลัง ทำงานต่อเนื่องจาก statement ก่อนหน้าไปเรื่อยๆ จนครบ กรณีนี้จะเหมือน

กับที่มีในภาษาคอมพิวเตอร์ทั่วไป คือ s_2 จะต่อกับ e_1 และ s_3 จะต่อกับ e_2 ตามลำดับ จนถึง s_n ต่อกับ e_{n-1}

2. **ชุดคำสั่งแบบ** Sequence **ที่ขาดตอน** (Discrete Sequence) คือ sequence ที่ซึ่ง statement หลัง ไม่จำเป็นต้องทำงานต่อเนื่องจาก statement ก่อนหน้านั้น คือ s_{i+1} ไม่จำเป็นต้องต่อจาก e_i หมายถึงว่า มีอย่างน้อย 1 $statement_{i+1}$ ใน sequence ที่ ไม่ต่อเนื่องกับ $statement_i$

*2) **ชุดคำสั่งแบบ** Unordered (Unordered compound statement)*

เป็นชุดคำสั่งที่ไม่ได้ให้ความสำคัญต่อลำดับการทำงานของแต่ละ statement หรือคำสั่งโดย statement ใดจะทำก่อนหลังก็ได้ แต่ให้ทำได้ทีละ 1 statement ในแต่ละครั้ง ซึ่งมีรูปแบบดังนี้

unordered : [_ , _] : $statement_1$ [_ , _] : $statement_2$: [_ , _] : $statement_n$	หรือ	**unordered :** $statement_1$ $statement_2$: $statement_n$

สังเกตว่า ในชุดคำสั่งแบบ Unordered และช่วงเวลาที่กำกับในทุกๆ statement จะเป็นแบบไม่ระบุเวลาเริ่มต้นและสิ้นสุด

*3) **ชุดคำสั่งแบบ** Alternative (Alternative compound statement)*

เป็นการประมวลผลประโยคในชุดคำสั่งไปทีละประโยคตามลำดับ แต่ไม่จำเป็นที่จะต้องประมวลผลประโยคทั้งหมด นั่นคือ หากประมวลผลประโยคได้สมบูรณ์ ไม่มีข้อผิดพลาดเกิดขึ้นแล้ว จะไม่มีการประมวลผลประโยคอื่นอีก ในทางกลับกัน หากประมวลผลประโยคได้ไม่สมบูรณ์จะทำการประมวลผลประโยคถัดไปเรื่อยๆ ซึ่งมีรูปแบบดังนี้

alternative :
[s_1,e_1] : $statement_1$
[s_2,e_2] : $statement_2$
:
[s_n,e_n] : $statement_n$

ช่วงเวลาของแต่ละ statement จะต้องเรียงกันเป็นลำดับ เช่นเดียวกับชุดคำสั่งแบบ Sequence นั่นคือ $s_1 < e_1 < s_2 < e_2 < \ldots < s_n < e_n$

*4) **ชุดคำสั่งแบบ** Parallel (Parallel compound statement)*

เป็นชุดคำสั่งที่แต่ละคำสั่งจะทำงานไปพร้อมๆ กันได้อย่างเป็นอิสระต่อกัน มีรูปแบบดังนี้

```
parallel :
    [s1,e1] : statement1
    [s2,e2] : statement2
                :
    [sn,en] : statementn
```

โดยที่ช่วงเวลาของแต่ละ statement มีค่าที่เป็นอิสระต่อกัน แต่จุดเวลาเริ่มต้นของทุกคำสั่งจะต้องมากกว่าหรือเท่ากับจุดเวลา ณ ขณะที่ชุดคำสั่งนี้ถูกเรียกให้ทำงาน

*5) **ชุดคำสั่งแบบ** Nested (Nested compound statement)*

ให้ถือว่าชุดคำสั่งแบบ sequence, unordered, alternative และ parallel ที่เพิ่งอธิบายไปเป็นเสมือน 1 statement ดังนั้นการสร้างชุดคำสั่งแบบ Nested ทำได้โดยใส่ชุดคำสั่งทั้งสี่เหมือน 1 statement ไว้ภายในชุดคำสั่งแบบ sequence หรือ unordered หรือ alternative หรือ parallel ที่สำคัญ ผู้ใช้ต้องยึดถือความหมายของแต่ละประเภทชุดคำสั่งอย่างเคร่งครัด ได้แก่ ในชุดคำสั่งแบบ unordered แต่ละ statement ภายใน จะไม่มีการระบุเวลาเริ่มต้นและสิ้นสุดของ statement เป็นต้น ตัวอย่างของชุดคำสั่งแบบ Nested แสดงได้ดังนี้

```
[s1,e1] : statement1
[s2,e2] : statement2
unordered :
    [ _ , _ ] : statement3
    [ _ , _ ] : statement4
```

หรือ

```
parallel :
    [s1, e1] : statement1
    [s2, e2] : statement2
    unordered :
        [ _ , _ ] : statement3
        [ _ , _ ] : statement4
```

5.5.3.6 ช่วงเวลารวมของชุดคำสั่ง

ดังที่ได้อธิบายไปข้างต้นว่า โปรแกรมเมอร์สามารถกำหนดเวลาการทำประโยคใดๆ ได้ ประโยคชุดคำสั่งก็เป็นประโยคในรูปแบบหนึ่งของภาษา CL ทำให้เราสามารถกำหนดช่วงเวลาเริ่มต้นและสิ้นสุดของการทำงานของทั้งชุดคำสั่งได้ในรูปแบบ

```
[Ps, Pe] : compound statement
```

โดยหมายถึงว่า P_s คือจุดเวลาเริ่มต้นที่ชุดคำสั่งเริ่มทำงาน และ P_e คือจุดเวลาที่ชุดคำสั่งสิ้นสุดการทำงานชุดคำสั่ง นั่นหมายถึง ต้องทำงานประโยคทั้งหมดภายในชุดคำสั่งให้เสร็จภายในเวลา P_e

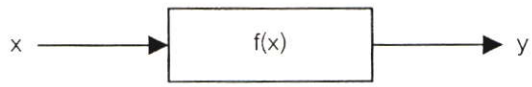
ในกรณีที่สามารถทำงานประโยคชุดคำสั่งได้เสร็จก่อนเวลา P_e โปรแกรมก็ต้องรอจนกว่าถึงเวลาที่กำหนดจึงจะสามารถประมวลผลประโยคถัดไปได้ ในทางกลับกัน หากไม่สามารถประมวล

ผลประโยคทั้งหมดในประโยคชุดคำสั่งได้ภายในเวลา P_e จะเกิดความผิดพลาดและไม่ทำการประมวลผลประโยคใดๆ ต่อ

5.5.4 การนิยามฟังก์ชันในภาษา CL

5.5.4.1 ฟังก์ชัน (Function)

ฟังก์ชันเป็นการทำงานที่มีการคืนค่าผลลัพธ์ออกมา 1 ค่า ซึ่งเขียนอยู่ในรูปแบบทางคณิตศาสตร์ได้ว่า y = f(x) เมื่อ f คือ ชื่อฟังก์ชัน, x คือ อินพุทของฟังก์ชัน และ y คือผลลัพธ์ที่ได้จากการทำฟังก์ชัน ฟังก์ชันสามารถแสดงเป็นโมเดล ได้ดังนี้



รูปที่ 5.5 การทำงานของฟังก์ชัน

ค่าหรือผลลัพธ์ต่างๆ ที่เกิดขึ้นจากการกำหนดของฟังก์ชัน เรียกโดยรวมว่า พารามิเตอร์ (Parameters) แบ่งได้เป็น 3 ประเภท ดังนี้

1) *Input parameters* หมายถึง ค่าที่ใช้สำหรับเป็นอินพุทของฟังก์ชัน
2) *Output parameters* หมายถึง ค่าที่ได้จากการทำฟังก์ชัน หรือผลลัพธ์นั่นเอง
3) *Input/output parameters* เป็นส่วนที่เป็นทั้งอินพุทของฟังก์ชันและเป็นผลลัพธ์ของการทำฟังก์ชันด้วย

ตัวอย่างเช่น ให้ f(x) คือ ฟังก์ชันหาค่าสแควร์รูท (Square root) โดยที่ $f(x) = x^2$ จะได้ว่า f(2) = 4 นั่นคือ x = 2 และ y = 4 เป็นต้น

5.5.4.2 การนิยามฟังก์ชัน

โปรแกรมเมอร์สามารถกำหนดการทำงานในโปรแกรมภาษา CL ได้หลายวิธี หนึ่งในนั้นคือการเรียกฟังก์ชันที่กำหนดไว้ในและนอกโปรแกรมด้วยรูปแบบ function() และ module.function() ตามลำดับ

ซึ่งฟังก์ชันที่ถูกเรียกใช้นั้นอาจเป็นฟังก์ชันที่เกิดจากการสร้างขึ้นของโปรแกรมเมอร์ (define function) หรือเป็นฟังก์ชันที่กำหนดไว้ในภาษา CL อยู่แล้ว (Built-in function) ก็ได้ ในหัวข้อนี้จึงขอกล่าวถึงการกำหนดหรือการสร้างนิยามฟังก์ชันในภาษา CL ซึ่งมีรูปแบบดังนี้

```
def function_name(param1, param2, ...) :
        definition
```

โดยกำหนดให้

- function_name เป็นชื่อของฟังก์ชัน ซึ่งกำหนดได้ตามรูปแบบของ identifier
- param เป็นอินพุทพารามิเตอร์ของฟังก์ชัน ซึ่งสามารถกำหนดด้วยข้อมูลชนิดต่างๆ ในภาษา CL
- definition จะประกอบด้วยประโยครูปแบบต่างๆ ซึ่งเป็นส่วนที่ระบุการทำงานทั้งหมดของฟังก์ชัน

ในกรณีที่ definition เป็น $<P_s,P_e>$: statement ซึ่งหมายถึงว่า function นี้เริ่มทำงาน ณ เวลา P_s และสิ้นสุดการทำงานที่เวลา P_e ดังนั้น การเรียกใช้งาน function นี้จะเป็นรูปแบบ $<P_s,P_e>$: function เช่นกัน โดยผู้เขียนโปรแกรมจะต้องกำหนด P_s และ P_e และเรียกใช้ function ให้สอดคล้องกับ P_s และ P_e ที่กำหนดใน definition เช่น

```
def function1(x, y) :
   < x, x+10min > : z = do something with y
   return(z)
```

จากตัวอย่างเป็นการนิยามฟังก์ชันชื่อ function1 ที่มีอินพุทพารามิเตอร์ 2 ตัว คือ x ซึ่งเป็นข้อมูลชนิดจุดเวลา และ y ซึ่งไม่ได้ระบุชนิดข้อมูลไว้ ในการทำฟังก์ชัน function1 ได้กำหนดให้ทำงานภายในเวลา 10 นาที นับตั้งแต่จุดเวลา x ดังนั้น หากมีการกำหนดช่วงเวลาให้กับการทำงานฟังก์ชัน (เรียกด้วยประโยคคำสั่ง) ช่วงเวลาที่ใช้จะต้องมีระยะเวลาอย่างน้อย 10 นาที เป็นต้น

5.5.4.3 ขอบเขตของตัวแปร

ในภาษา CL ได้มีการกำหนดขอบเขตของตัวแปร (Variable scope) ไว้ 2 ขอบเขต ได้แก่ ขอบเขตแบบหลัก (Global) และขอบเขตย่อย (Local) โดยตัวแปร หรือ identifier ที่ถูกนิยามไว้ในขอบเขตดังกล่าว จะเรียกว่า ตัวแปรหลัก (Global variable) และ ตัวแปรย่อย (Local variable) ตามลำดับ

1) *ตัวแปรหลัก (Global variable)* เป็นตัวแปรที่ถูกนิยามไว้ในระดับบนสุด(highest level) ของโปรแกรม หรือโปรแกรมหลัก ซึ่งตัวแปรนี้จะสามารถอ้างอิงถึงได้ทั้งในโปรแกรมหลัก และโปรแกรมย่อย (ฟังก์ชัน)
2) *ตัวแปรย่อย (Local variable)* เป็นตัวแปรที่ถูกนิยามอยู่ในโปรแกรมย่อยอย่างฟังก์ชัน ซึ่งตัวแปรนี้จะสามารถอ้างอิงถึงได้ในโปรแกรมย่อยเท่านั้น

ตัวอย่างเช่น

```
global_str = "hello"
def foo() :
   local_str = " world"
   return global_str + local_str
```

จะได้ว่า global_str เป็นตัวแปรหลัก ขณะที่ local_str เป็นตัวแปรย่อยของโปรแกรมฟังก์ชัน foo มีการอ้างอิงถึงทั้งตัวแปรหลักและตัวแปรย่อย ผลลัพธ์ที่ได้จากการทำฟังก์ชัน foo คือ สตริง "hello world"

จากตัวอย่างข้างต้น ฟังก์ชัน foo สามารถอ้างอิงใช้ค่าของตัวแปรหลักเพียงอย่างเดียว ไม่สามารถเปลี่ยนแปลงค่าของตัวแปรหลักได้ ในกรณีที่ต้องการเปลี่ยนแปลงค่าตัวแปรหลักด้วยคำสั่งภายในฟังก์ชัน หรือโปรแกรมย่อย ทำได้โดยการใช้คำสั่ง global ด้วยรูปแบบดังนี้

```
global var1[, var2[, … varN]]
```

ตัวอย่างเช่น

```
is_this_global = "xyz"
def foo() :
   global is_this_global
   this_is_local = "abc"
   is_this_global = "def"
   print this_is_local + is_this_global
foo()
print is_this_global
```

จะได้ผลลัพธ์ดังนี้

```
abcdef      //  this_is_local + is_this_global
def         //  is_this_global
```

5.5.5 **สคริปต์ตอบสนองต่อเหตุการณ์** (Event-action script)

นอกจากความสามารถในการประมวลผลเวลา และการกำหนดช่วงเวลาให้กับการทำงานในโปรแกรมได้แล้ว ภาษา CL ยังได้เพิ่มความสามารถในการตอบสนองต่อเหตุการณ์ภายนอกเข้าไปอีกด้วย โดยเหตุการณ์ที่กล่าวถึงนี้ จะมุ่งเน้นไปที่ข้อความสื่อสารที่ได้รับจากเอเจนต์หรือโปรแกรมอื่น นั่นคือ ภาษา CL สนับสนุนการติดต่อสื่อสารระหว่างเอเจนต์ซึ่งสอดคล้องกับการทำงานในชีวิตประจำวันของมนุษย์ได้ ด้วยความสามารถดังกล่าว ทำให้โปรแกรมที่เขียนด้วยภาษา CL เขียนและเข้าใจได้ง่าย แม้ผู้อ่านจะไม่มีชำนาญในการเขียนโปรแกรมคอมพิวเตอร์มากนัก

ผู้เขียนโปรแกรมสามารถกำหนดการตอบสนองต่อเหตุการณ์ภายนอกด้วยการใช้สคริปต์ตอบสนองต่อเหตุการณ์ (Event-action script) ซึ่งประกอบด้วย ประโยคตอบสนองเหตุการณ์ และประโยคชุดของการตอบสนองเหตุการณ์

5.5.5.1 **เหตุการณ์** (Event)

เหตุการณ์ คือสิ่งที่เกิดจากสภาพแวดล้อมภายนอก ซึ่งสิ่งนี้ไม่ได้เกิดขึ้นจากชุดคำสั่ง โดยเหตุการณ์จะปรากฏในรูปข่าวสาร (Information) และเราสามารถตอบสนองต่อการเกิดขึ้นของข่าวสารนั้นได้โดยให้เอเจนต์ทำคำสั่งตามที่เราต้องการ เช่น ถ้ามีเอเจนต์หนึ่งส่งข้อความมาเพื่อถามคำถามเรา เราจะค้นหาคำตอบ แล้วส่งคำตอบที่ได้กลับไป เป็นต้น

เวลาที่เกี่ยวข้องกับเหตุการณ์คือจุดเวลาที่เหตุการณ์หรือข่าวสารนั้นเกิด แต่เพื่อให้ภาษา CL สามารถทำงานได้อย่างมีประสิทธิภาพ เราจะให้ผู้เขียนโปรแกรมสามารถระบุช่วงเวลาที่เหตุการณ์นั้นเกิดขึ้นกำกับไว้ในทุกๆ เหตุการณ์ได้ ทำให้รูปแบบเหตุการณ์ที่เกิดในระหว่างช่วงเวลาเป็น ดังนี้

$[P_s, P_e]$: event

หมายถึง เหตุการณ์นี้เกิดขึ้นภายในช่วงเวลา $[P_s, P_e]$ ซึ่งช่วงเวลานี้ มีลักษณะเช่นเดียวกับที่ใช้ในกรณีคำสั่ง คือแบ่งได้เป็นแบบระบุเวลา, แบบกึ่งระบุเวลา และแบบไม่ระบุเวลา

สำหรับในวิทยานิพนธ์นี้ ได้กำหนดให้เหตุการณ์ในภาษา CL เป็นการได้รับข้อความสื่อสาร (Communication message) จากเอเจนต์อื่น

5.5.5.2 **ข้อความสื่อสาร** (Communication Message)

ข้อความสื่อสาร คือ ข้อความที่เอเจนต์(โปรแกรม) ได้ส่งไปยังเอเจนต์(โปรแกรม) อื่น เพื่อใช้ในการสื่อสารบางสิ่งบางอย่าง เช่น เพื่อแจ้งข้อมูล ถามคำถาม ขอร้อง ลบข้อมูล เป็นต้น โดยข้อความสื่อสารนี้แบ่งได้เป็น 2 ประเภท คือ

- ข้อความส่งสาร คือ ข้อความที่ใช้เริ่มต้นในการสื่อสารของโปรแกรม ประกอบด้วย ข้อความ Tell, Query, Request, Delete, Please_tell, Please_query, Please_request และ Please_delete
- ข้อความตอบสนอง คือ ข้อความที่ทำการตอบสนองต่อข้อความส่งสาร ประกอบด้วย ข้อความ Yes, I know, Ignore, No, Tell, No Answer, I don't know, Failed, No Info, Query, No Question, Request, No Command, Delete และ No Delete

การที่เอเจนต์ได้รับข้อความสื่อสารจากเอเจนต์อื่นถือว่าเป็นการเกิดเหตุการณ์ภายนอก ซึ่งสามารถกำหนดการเกิดเหตุการณ์ดังกล่าวได้ด้วยรูปแบบ exist(Comm. msg) เช่น exist (tell_msg()) หมายถึง เหตุการณ์ที่ได้รับข้อความสื่อสารประเภท Tell เป็นต้น

5.5.5.3 ประโยคตอบสนองต่อเหตุการณ์

ผู้เขียนโปรแกรมสามารถกำหนดคำสั่งการทำงานเพื่อตอบสนองต่อข้อความสื่อสารที่ได้รับได้ด้วย **ประโยคตอบสนองต่อเหตุการณ์** ซึ่งมีรูปแบบดังนี้

$[P_s,P_e]$: Event → Action

หรือ

$[P_s,P_e]$: exist(Comm message) → Action

หมายถึง เมื่อได้รับข้อความสื่อสาร (Comm message) ซึ่งมีเหตุการณ์เกิดขึ้น ในช่วงเวลา $[P_s,P_e]$ ระบบจะทำ Action เพื่อตอบสนองตามที่ผู้เขียนกำหนด โดยที่ Action คือ คำสั่งการทำงานที่เป็นประโยค(statement)ใดๆ และเช่นเดียวกับการกำหนดเวลาในคำสั่ง ช่วงเวลา $[P_s,P_e]$ สามารถเป็นแบบระบุเวลา แบบกึ่งระบุเวลา และ แบบไม่ระบุเวลา ตัวอย่างประโยคตอบสนองต่อเหตุการณ์ เช่น exist(comm_msg) → statement, [_ , P_{e1}] : exist(comm_msg) → [P_{s2}, P_{e2}] : statement, [P_{s1},_] : exist(comm_msg) → [P_{s2}, _] : statement

5.5.5.4 ประโยคชุดของการตอบสนองต่อเหตุการณ์

ในภาษา CL ผู้เขียนสามารถกำหนดสคริปต์ตอบสนองเหตุการณ์ เพื่อใช้ตอบสนองข้อความสื่อสารจนเกิดเป็นเสมือนการสนทนาที่มีเรื่องราวได้ ด้วยการนำประโยคตอบสนองเหตุการณ์มารวมกัน เรียกว่า **ประโยคชุดของการตอบต่อสนองเหตุการณ์** ซึ่งแบ่งออกเป็น 6 ประเภท ได้แก่ Independent Repeatable and Partial, Independent Repeatable and Total, Sequence, Unordered, One และ Nested ซึ่งในหัวข้อนี้ จะกล่าวถึงหลักการทำงานของประโยคชุดการตอบสนองต่อเหตุการณ์แบบต่างๆ อย่างคร่าวๆ ส่วนรายละเอียดการทำงานจะกล่าวถึงในบทที่ 6 ว่าด้วยเรื่องการสื่อสารในภาษา CL

1) ประโยคชุดการตอบสนองแบบ Independent Repeatable and Partial

เมื่อเกิดเหตุการณ์ขึ้น เอเจนต์จะทำการประมวลผลประโยคตอบสนองต่อเหตุการณ์ที่กำหนดไว้ในประโยคชุดการตอบสนองได้ครั้งละตั้งแต่ 1 ประโยคขึ้นไป ซึ่งจะทำประโยคใดก่อนหลังก็ได้ ไม่มีลำดับ อีกทั้งยังสามารถทำประโยคตอบสนองนั้นซ้ำเมื่อเกิดเหตุการณ์ที่สอดคล้องขึ้นอีก

*2) **ประโยคชุดการตอบสนองแบบ** Independent Repeatable and Total*

เมื่อเกิดเหตุการณ์ขึ้น เอเจนต์จะทำการประมวลผลประโยคตอบสนองต่อเหตุการณ์ที่กำหนดไว้ในประโยคชุดการตอบสนองได้ครั้งละตั้งแต่ 1 ประโยคขึ้นไป อย่างไม่มีลำดับ และทำซ้ำกี่ครั้งก็ได้ จนกว่าจะทำครบทุกประโยคในชุดการตอบสนอง

*3) **ประโยคชุดการตอบสนองแบบ** Sequence*

เมื่อเกิดเหตุการณ์ขึ้น เอเจนต์จะทำการประมวลผลประโยคตอบสนองต่อเหตุการณ์ที่กำหนดไว้ในประโยคชุดการตอบสนองได้ทีละประโยค ตามลำดับการทำงานจากบนลงล่างจนครบทุกประโยค

*4) **ประโยคชุดการตอบสนองแบบ** Unordered*

เมื่อเกิดเหตุการณ์ขึ้น เอเจนต์จะทำการประมวลผลประโยคตอบสนองต่อเหตุการณ์ที่กำหนดไว้ในประโยคชุดการตอบสนองได้ทีละประโยค โดยไม่มีลำดับจนครบทุกประโยค

*5) **ประโยคชุดการตอบสนองแบบ** One*

เมื่อเกิดเหตุการณ์ขึ้น เอเจนต์จะทำการประมวลผลประโยคตอบสนองต่อเหตุการณ์ที่กำหนดไว้ในประโยคชุดการตอบสนองได้เพียงครั้งละ 1 ประโยคเท่านั้น ซึ่งจะทำประโยคใดก็ได้

*6) **ประโยคชุดการตอบสนองแบบ** Nested*

เป็นการนำประโยคชุดการตอบสนองต่อเหตุการณ์ในแบบ Independent Repeatable and Partial, Sequence, Independent and Total และ One มาผสมกัน

5.6 **โอเปอเรเตอร์** (Operators)

โอเปอเรเตอร์ที่ใช้ในภาษา CL แบ่งเป็น 4 ประเภทคือ โอเปอเรเตอร์ทางตรรกะ, โอเปอเรเตอร์ทางการเปรียบเทียบ, โอเปอเรเตอร์คำนวณเลขฐานสอง และโอเปเรเตอร์ทางคณิตศาสตร์ โดยส่วนหนึ่งนำมาจากภาษาไพธอน ซึ่งได้กล่าวถึงไปแล้วในบทที่ 2 จึงไม่ขอกล่าวถึงในหัวข้อนี้อีก แต่จะอธิบายถึงโอเปอเรเตอร์ใหม่ที่เพิ่มเข้ามานอกเหนือจากภาษาไพธอน ได้แก่ โอเปอเรเตอร์ที่ประมวลผลเกี่ยวกับเวลา โดยแบ่งได้เป็น โอเปอเรเตอร์ทางการเปรียบเทียบสำหรับเวลา และโอเปอเรเตอร์ทางคณิตศาสตร์สำหรับเวลา ซึ่งมีรายละเอียดดังนี้

5.6.1 **โอเปอเรเตอร์ทางการเปรียบเทียบของเวลา** (Comparison operators of Time)

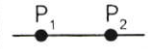
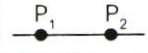
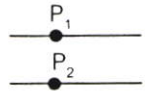
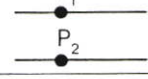
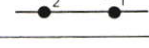
ได้แก่ โอเปอเรเตอร์ระหว่างจุดเวลา, โอเปอเรเตอร์ระหว่างระยะเวลา, โอเปอเรเตอร์ระหว่างช่วงเวลา และ โอเปอเรเตอร์ระหว่างจุดเวลากับช่วงเวลา ซึ่งความสัมพันธ์ดังกล่าวนำไปใช้ในการ

คำนวณและเปรียบเทียบข้อมูลประเภทเวลาแบบต่างๆ ในภาษา CL

5.6.1.1 โอเปอเรเตอร์ทางการเปรียบเทียบระหว่างจุดเวลา

ประกอบด้วย โอเปอเรเตอร์แบบ ก่อน(before), หลัง(after) และเท่ากัน(equal) เพื่อเปรียบเทียบว่าจุดเวลา 2 จุด จุดแรกเกิดก่อน หรือเกิดหลัง หรือ เกิดพร้อมกัน สามารถเปรียบเทียบโดยใช้เครื่องหมาย $<, \leq, >, \geq, =, \neq$ ในทางคณิตศาสตร์ได้ ดังแสดงในตารางที่ 5.2 ซึ่งผู้เขียนได้กำหนดขึ้นเองโดยอาศัยความสัมพันธ์ที่ใช้ในชีวิตประจำวัน

ตารางที่ 5.2 แสดงความสัมพันธ์ระหว่างจุดเวลา

รูปแบบคณิตศาสตร์	ความสัมพันธ์	รูปแบบภาษา CL	รูปแบบกราฟ
1. $P_1 < P_2$	P_1 before P_2	$P_1 < P_2$	
2. $P_1 \leq P_2$	P_1 before or equal P_2	$P_1 <= P_2$	
3. $P_1 > P_2$	P_1 after P_2	$P_1 > P_2$	
4. $P_1 \geq P_2$	P_1 after or equal P_2	$P_1 >= P_2$	
5. $P_1 = P_2$	P_1 equal P_2	$P_1 == P_2$	
6. $P_1 \neq P_2$	P_1 not equal P_2	$P_1 != P_2$	

5.6.1.2 โอเปอเรเตอร์ทางการเปรียบเทียบระหว่างระยะเวลา

เนื่องจากข้อมูลระยะเวลาเป็นข้อมูลเชิงปริมาณ ดังนั้น โอเปอเรเตอร์ระหว่างข้อมูลประเภทนี้ จะเป็นไปตามความสัมพันธ์ระหว่างตัวเลข ดังแสดงได้ในตารางที่ 5.3

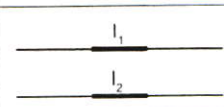
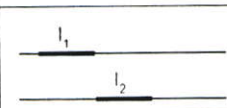
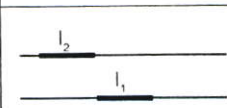
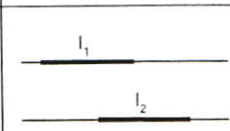
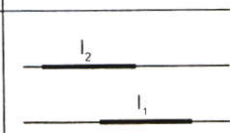
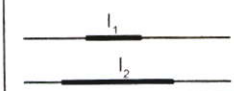
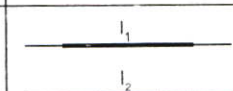
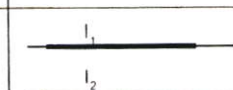
ตารางที่ 5.3 แสดงความสัมพันธ์ระหว่างระยะเวลา

รูปแบบคณิตศาสตร์	ความสัมพันธ์	รูปแบบภาษา CL	ตัวอย่าง
1. $D_1 < D_2$	D_1 less than D_2	$D_1 < D_2$	2h < 4h
2. $D_1 > D_2$	D_1 more than D_2	$D_1 > D_2$	4h > 2h
3. $D_1 \leq D_2$	D_1 less than or equal D_2	$D_1 <= D_2$	2h <= 2h
4. $D_1 \geq D_2$	D_1 more than or equal D_2	$D_1 >= D_2$	2h >= 2h
5. $D_1 = D_2$	D_1 equal D_2	$D_1 == D_2$	5s == 5s
6. $D_1 \neq D_2$	D_1 not equal D_2	$D_1 != D_2$	5s != 3s

5.6.1.3 โอเปอเรเตอร์ทางการเปรียบเทียบระหว่างช่วงเวลา

โอเปอเรเตอร์ที่แสดงทั้งหมดนำมาจากงานวิจัยของ James Allen [8] ดังแสดงในตารางที่ 5.4 ซึ่งขอกำหนดให้ start(I) หมายถึง จุดเวลาเริ่มต้นของช่วงเวลา I และ end(I) หมายถึง จุดเวลาสิ้นสุดของช่วงเวลา I

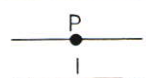
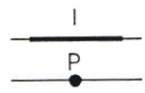
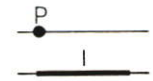
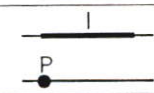
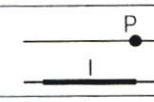
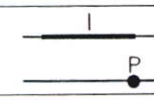
ตารางที่ 5.4 แสดงความสัมพันธ์ระหว่างช่วงเวลา

ความสัมพันธ์	ความหมาย	รูปแบบ CL	รูปแบบกราฟ
1. I_1 before I_2	$end(I_1) < start(I_2)$	$I_1 < I_2$	
2. I_1 after I_2	$start(I_1) > end(I_2)$	$I_1 > I_2$	
3. I_1 equal I_2	$(start(I_1) = start(I_2))$ & $(end(I_1) = end(I_2))$	$I_1 == I_2$	
4. I_1 meets I_2	$end(I_1) = start(I_2)$	$I_1 m I_2$	
5. I_1 met-by I_2	$start(I_1) = end(I_2)$	$I_1 mi I_2$	
6. I_1 overlaps I_2	$(start(I_1) < start(I_2))$ & $(start(I_2) < end(I_1))$ & $(end(I_1) < end(I_2))$	$I_1 o I_2$	
7. I_1 overlapped-by I_2	$(start(I_1) > start(I_2))$ & $(end(I_2) > start(I_1))$ & $(end(I_1) > end(I_2))$	$I_1 oi I_2$	
8. I_1 during I_2	$(start(I_1) > start(I_2))$ & $(end(I_1) < end(I_2))$	$I_1 d I_2$	
9. I_1 contains I_2	$(start(I_1) < start(I_2))$ & $(end(I_1) > end(I_2))$	$I_1 di I_2$	
10. I_1 starts I_2	$(start(I_1) = start(I_2))$ & $(end(I_1) < end(I_2))$	$I_1 s I_2$	
11. I_1 started-by I_2	$(start(I_1) = start(I_2))$ & $(end(I_1) > end(I_2))$	$I_1 si I_2$	
12. I_1 finishes I_2	$(end(I_1) = end(I_2))$ & $(start(I_1) > start(I_2))$	$I_1 f I_2$	
13. I_1 finished-by I_2	$(end(I_1) = end(I_2))$ & $(start(I_1) < start(I_2))$	$I_1 fi I_2$	

5.6.1.4 โอเปอเรเตอร์ทางการเปรียบเทียบระหว่างจุดเวลาและช่วงเวลา

โอเปอเรเตอร์ระหว่างข้อมูลประเภทจุดเวลากับข้อมูลประเภทช่วงเวลานั้น มีลักษณะใกล้เคียงกับโอเปอเรเตอร์ระหว่างข้อมูลประเภทช่วงเวลาในตาราง 5.4 บางส่วน ทำให้ไม่สามารถนำโอเปอเรเตอร์ทุกแบบมาใช้ได้ ได้แก่ โอเปอเรเตอร์ overlap, meet และ equal เนื่องจากความแตกต่างระหว่างข้อมูลทั้งสองประเภทนั่นเอง โอเปอเรเตอร์ที่แสดงในตาราง 5.5 นี้ ผู้เขียนได้กำหนดขึ้นมาใหม่โดยได้ดัดแปลงมาจากแนวความคิดของ Allen ที่อธิบายไปในหัวข้อที่แล้ว โดยกำหนดให้ P เป็นสัญลักษณ์แทนข้อมูลประเภทจุดเวลา และ I เป็นสัญลักษณ์แทนข้อมูลประเภทช่วงเวลา

ตารางที่ 5.5 แสดงความสัมพันธ์ระหว่างจุดเวลา และ ช่วงเวลา

ความสัมพันธ์	ความหมาย	รูปแบบ CL	รูปแบบกราฟ
1. P before I	P < start(I)	P < I	
2. P after I	P > start(I)	P > I	
3. P during I	(P> start(I)) & (P< end(I))	P d I	
4. I contains P	(start(I) < P) & (end(I) > P)	I di P	
5. P starts I	P = start(I)	P s I	
6. I started-by P	start(I) = P	I si P	
7. P finishes I	P = end(I)	P f I	
8. I finished-by P	end(I) = P	I fi P	

5.6.2 โอเปอเรเตอร์ทางคณิตศาสตร์สำหรับเวลา (Arithmetic operators of Time)

โอเปอเรเตอร์ทางคณิตศาสตร์สำหรับเวลาที่นำเสนอนี้ ผู้เขียนได้กำหนดขึ้นมาใหม่ทั้งหมดซึ่งเป็นรูปแบบโอเปอเรเตอร์อย่างง่ายที่พิจารณาจากการประมวลผลเวลาทั่วไปสำหรับกิจกรรมต่างๆ ในชีวิตประจำวัน ในกรณีที่เป็นข้อมูลเชิงปริมาณ โอเปอเรเตอร์ที่ใช้จะเป็นโอเปอเรเตอร์พื้นฐานทางคณิตศาสตร์ นั่นคือ บวก (+), ลบ (—), คูณ (*) และหาร (/) ในกรณีที่เป็นข้อมูลประเภทเซต โอเปอเรเตอร์ที่ใช้จะเป็นโอเปอเรเตอร์พื้นฐานสำหรับเซต ได้แก่ ยูเนียน ($\cup$), อินเตอร์เซคชัน ($\cap$), เซตผลต่าง (—) และคอมพลีเมนต์ (') และเป็นที่น่าสังเกตว่า ข้อมูลที่นำมาประมวลผลและผลลัพธ์

จากการประมวลผลที่ได้ไม่จำเป็นต้องเป็นข้อมูลชนิดเดียวกัน ดังจะแสดงให้เห็นในลำดับต่อไป เพื่อความชัดเจนในการอธิบายในหัวข้อนี้ จึงกำหนดใช้สัญลักษณ์ดังต่อไปนี้

P แทน ข้อมูลประเภทจุดเวลา , D แทน ข้อมูลประเภทระยะเวลา , I แทน ข้อมูลประเภทช่วงเวลา

5.6.2.1 โอเปอเรเตอร์ทางคณิตศาสตร์สำหรับจุดเวลา

โอเปอเรเตอร์ที่ใช้สำหรับจุดเวลานี้มีเพียง 2 ชนิด คือ โอเปอเรเตอร์บวก และ ลบ เท่านั้น จุดเวลาสามารถบวก หรือ ลบ กับข้อมูลประเภทระยะเวลาได้ อีกทั้งผลลัพธ์ที่ได้ก็สามารถเป็นได้ทั้งจุดเวลาและระยะเวลา ดังแสดงในตารางที่ 5.6

พีชคณิตที่ได้จะไม่ยุ่งยาก คือ ผู้ใช้สามารถหาจุดเวลาหนึ่งด้วยการลดหรือเพิ่มด้วยระยะเวลา และผู้ใช้สามารถหาระยะเวลาได้จากการลบกันของจุดเวลา 2 จุด

ตารางที่ 5.6 แสดงการคำนวณและโอเปอเรเตอร์สำหรับจุดเวลา

โอเปอเรเตอร์	พีชคณิต	ตัวอย่าง
1. บวก (+)	$P_1 = P_2 + D$	11:00:00|30/08/2002 = 11:00:00|28/08/2002 + 2d
2. ลบ (—)	$D = P_2 - P_1$	2d = 11:00:00|30/08/2002 - 11:00:00|28/08/2002
	$P_1 = P_2 - D$	11:00:00|28/08/2002 = 11:00:00|30/08/2002 – 2d

5.6.2.2 โอเปอเรเตอร์ทางคณิตศาสตร์สำหรับระยะเวลา

โอเปอเรเตอร์ที่ใช้สำหรับข้อมูลประเภทนี้มีทั้ง 4 แบบ คือ บวก, ลบ, คูณ และหาร ซึ่งง่ายต่อการเข้าใจ เนื่องจากเวลาประเภทนี้เป็นข้อมูลเชิงปริมาณปกติและมีการคำนวณเช่นเดียวกับการคำนวณตัวเลขปริมาณทั่วไป ทั้งยังสามารถนำตัวเลขไปคูณหรือหารกับระยะเวลาได้ ดังแสดงในตารางที่ 5.7 ซึ่งมีพีชคณิตสำหรับโอเปอเรเตอร์บวกที่ได้มาจากตารางที่ 5.6 ด้วย

ตารางที่ 5.7 แสดงการคำนวณและโอเปอเรเตอร์สำหรับระยะเวลา

โอเปอเรเตอร์	พีชคณิต	ตัวอย่าง
1. บวก (+)	$D_3 = D_1 + D_2$	8d,2h = 8d + 2h
	$P_2 = P_1 + D$	11:00:00|30/08/2002 = 11:00:00|28/08/2002 + 2d
2. ลบ (—)	$D_3 = D_1 - D_2$	4d,2h = 8d,2h – 4d
3. คูณ (*)	$D_2 = D_1 * Number$	4d = 2d * 2
4. หาร (/)	$D_2 = D_1 / Number$	2d = 4d / 2
	$Number = D_1 / D_2$	2 = 4d / 2d

นอกจากนี้ ยังมีโอเปอเรเตอร์พื้นฐานสำหรับข้อมูลประเภทระยะเวลานอกเหนือจากในตาราง ได้แก่ ยกกำลัง mod และอื่นๆ ข้อมูลประเภทระยะเวลาจะมีคุณสมบัติเช่นเดียวกับคุณสมบัติของจำนวนจริงทั่วไป ได้แก่ คุณสมบัติปิดการบวก, คุณสมบัติการสลับที่การบวก และคุณสมบัติการเปลี่ยนกลุ่มการบวก นอกจากนี้ ยังมีเอกลักษณ์การบวก คือ 0d,0h,0min,0s

5.6.2.3 **โอเปอเรเตอร์ทางคณิตศาสตร์สำหรับช่วงเวลา**

1) **โอเปอเรเตอร์ทั่วไปสำหรับช่วงเวลา** โอเปอเรเตอร์ที่ใช้เป็นโอเปอเรเตอร์เกี่ยวข้องการขยาย หด และเลื่อนตำแหน่งของช่วงเวลา ได้แก่ โอเปอเรเตอร์ extend, shrink และ shift ตามลำดับ ทั้งยังมีโอเปอเรเตอร์บวก สำหรับต่อช่วงเวลาเข้าด้วยกัน ดังแสดงไว้ในตารางที่ 5.8

ตารางที่ 5.8 แสดงการคำนวณและโอเปอเรเตอร์ทั่วไปสำหรับช่วงเวลา

โอเปอเรเตอร์	พีชคณิต	ตัวอย่าง
1.Extend left (<—)	$I_2 = I_1$ <- D	<12:00:00|28/08/2002, 13:00:00|28/08/2002> = <11:00:00|28/08/2002, 13:00:00|28/08/2002> <- 1h
2.Extend right (—>)	$I_2 = I_1$ -> D	<10:00:00|28/08/2002, 13:00:00|28/08/2002> = <10:00:00|28/08/2002, 12:00:00|28/08/2002> -> 1h
3.Shrink left (>—)	$I_2 = I_1$ >- D	<11:00:00|28/08/2002, 14:00:00|28/08/2002> = <13:00:00|28/08/2002, 14:00:00|28/08/2002> >- 2h
4.Shrink right (—<)	$I_2 = I_1$ -< D	<11:00:00|28/08/2002, 12:00:00|28/08/2002> = <11:00:00|28/08/2002, 14:00:00|28/08/2002> -< 2h
5.Shift left (<<)	$I_2 = I_1$ << D	<10:00:00|28/08/2002, 13:00:00|28/08/2002> = <11:00:00|28/08/2002, 14:00:00|28/08/2002> << 1h
6.Shift right (>>)	$I_2 = I_1$ >> D	<12:00:00|28/08/2002, 13:00:00|28/08/2002> = <11:00:00|28/08/2002, 12:00:00|28/08/2002> >> 1h
7.Concat (+)	$I_3 = I_1 + I_2$	<11:00:00|28/08/2002, 15:00:00|28/08/2002> = <11:00:00|28/08/2002, 12:00:00|28/08/2002> + <12:00:00|28/08/2002, 15:00:00|28/08/2002>

2) **เซตโอเปอเรเตอร์สำหรับช่วงเวลา** เซตโอเปอเรเตอร์เป็นโอเปอเรเตอร์ที่เพิ่มเติมขึ้นเพื่อสนับสนุนความสามารถของเซต ซึ่งใช้กับช่วงเวลา เซตโอเปอเรเตอร์ที่ใช้ ได้แก่ ยูเนียน, อินเตอร์เซคชัน, ผลต่างเซต และคอมพลีเมนต์ ดังแสดงไว้ในตารางที่ 5.9

การประมวลผลเกี่ยวกับเวลาที่ได้อธิบายไปทั้งหมดจะมีความเกี่ยวเนื่องกับการทำงานคำสั่ง (Action) และการตอบสนองต่อเหตุการณ์ในภาษา CL ซึ่งจะได้อธิบายไปแล้วข้างต้น

ตารางที่ 5.9 แสดงการคำนวณและเซตโอเปอเรเตอร์สำหรับช่วงเวลา

โอเปอเรเตอร์	พีชคณิต	ตัวอย่าง
1.ยูเนียน ($\cup$)	$I_1 = I_2 \cup I_3$	<11:00:00|28/08/2002, 15:00:00|28/08/2002> = <11:00:00|28/08/2002, 13:00:00|28/08/2002> $\cup$ <12:00:00|28/08/2002, 15:00:00|28/08/2002>
2.อินเตอร์เซคชัน ($\cap$)	$I_1 = I_2 \cap I_3$	<12:00:00|28/08/2002, 14:00:00|28/08/2002> = <11:00:00|28/08/2002, 14:00:00|28/08/2002> $\cap$ <12:00:00|28/08/2002, 15:00:00|28/08/2002>
3.เซตผลต่าง (—)	$I_3 = I_1 - I_2$	<11:00:00|28/08/2002, 11:59:59|28/08/2002> = <11:00:00|28/08/2002, 14:00:00|28/08/2002> - <12:00:00|28/08/2002, 16:00:00|28/08/2002>
4.คอมพลีเมนต์ (')	$I_2 = I_1'$	<_ ,10:59:59|28/08/2002>=<11:00:00|28/08/2002, _>'

5.7 **เอ็กซเพรสชัน** (Expression)

เป็นกลุ่มของโอเปอเรเตอร์และโอเปอแรนด์ ซึ่งเมื่อผ่านการประเมินค่า (Evaluation) จะให้ผลลัพธ์เป็นค่าออกมา ซึ่งค่าที่ได้สามารถนำไปใช้ในส่วนของประโยคได้ โดยค่าที่ได้จะเป็นข้อมูลประเภทใดนั้น ขึ้นกับประเภทโอเปอเรเตอร์และโอเปอแรนด์ที่ใช้ในเอ็กเพรสชั่นนั้นๆ

5.7.1 **ลำดับของโอเปอเรเตอร์** (Precedence)

ลำดับของตัวกระทำในภาษา CL ส่วนหนึ่งจะเหมือนกับลำดับของโอเปอเรเตอร์ในภาษาไพธอน แต่เนื่องจากได้มีการเพิ่มเติมโอเปอเรเตอร์สำหรับเวลาเข้ามา ทำให้ลำดับของโอเปอเรเตอร์ในภาษา CL เป็นดังนี้ or, and, not, (<, <=, >, >=, ==, !=, mi, m, oi, o, di, d, si, s, fi, f), |, ^, &, (<<, >>, <-, ->, >-, -<), (+, -), (*, /, %), **, (unary+, unary-), unary~

โดยที่โอเปอเรเตอร์ทางการเปรียบเทียบสำหรับเวลา เช่น mi, m, oi จะอยู่ในระดับเดียวกับโอเปอเรเตอร์ทางการเปรียบเทียบทั่วไปในภาษาไพธอน และเช่นเดียวกัน สำหรับโอเปอเรเตอร์ทางคณิตศาสตร์ของเวลา ก็จะอยู่ระดับเดียวกับโอเปอเรเตอร์ทางคณิตศาสตร์ของภาษาไพธอน ซึ่งจะพิจารณาตามความหมายของการทำโอเปอเรเตอร์เป็นหลัก เช่น โอเปอเรเตอร์บวกของเวลา ก็จะอยู่ระดับเดียวกับโอเปอเรเตอร์บวกในภาษาไพธอน เป็นต้น

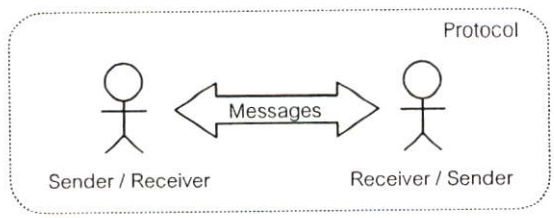
บทที่ 6
การสื่อสารในภาษา CL

ภาษา CL เป็นภาษาที่สนับสนุนการทำงานตามเวลาและเหตุการณ์ ซึ่งเหตุการณ์ที่ปรากฏในวิทยานิพนธ์นี้จะเน้นถึงเหตุการณ์ที่เอเจนต์หรือโปรแกรมได้รับข้อความสื่อสาร ทำให้ภาษา CL ออกแบบมาสนับสนุนการสื่อสารของเอเจนต์

ในบทนี้จะกล่าวถึงรายละเอียดการทำงานที่เกี่ยวข้องกับการสื่อสารทั้งหมด ซึ่งกล่าวถึงมาบ้างแล้วในบทที่ 5 เริ่มด้วยการอธิบายถึงองค์ประกอบและกระบวนการของการสื่อสารทั่วไปในหัวข้อที่ 6.1 ซึ่งหากต้องการให้มีความสามารถในการสื่อสารแล้ว ภาษา CL จำเป็นจะต้องมีการทำงานพื้นฐานที่สอดคล้องกับเหตุการณ์ดังกล่าว หลังจากนั้น จึงอธิบายถึงโครงสร้างการทำงานของ CL ที่เป็นเอเจนต์ในหัวข้อที่ 6.2 เมื่อผู้อ่านได้เข้าใจถึงภาพรวมของการสื่อสารใน CL แล้ว จะกล่าวถึงองค์ประกอบสำคัญในการสื่อสาร ได้แก่ ข้อความที่ใช้สื่อสารระหว่างเอเจนต์ และรูปแบบการโต้ตอบข้อความในหัวข้อที่ 6.3 ตามด้วยสัญลักษณ์ที่ใช้สื่อสารในภาษา CL, การส่งข้อความสื่อสาร, การตอบสนองต่อข้อความสื่อสาร และบทสนทนาสำหรับการสื่อสาร ในหัวข้อที่ 6.4, 6.5, 6.6 และ 6.7 ตามลำดับ

6.1. การสื่อสาร

การสื่อสาร เป็นการส่งสารจากผู้ส่งไปยังผู้รับโดยการส่งต้องกระทำโดยกระบวนการและรูปแบบที่เข้าใจกันทั้งสองฝ่าย องค์ประกอบของการสื่อสาร แบ่งเป็น 4 ส่วน คือ ผู้ส่งสาร(Sender) ผู้รับสาร(Receiver) สาร(Message) และกระบวนการและรูปแบบในการสื่อสาร(Protocol)



รูปที่ 6.1 การสื่อสาร

จุดประสงค์สำคัญในการพัฒนาภาษา CL คือ ความต้องการสร้างภาษาคอมพิวเตอร์ให้สนับสนุนการโปรแกรมที่มีการติดต่อสื่อสาร ดังนั้น ภาษา CL จะต้องออกแบบรูปแบบภาษาให้สอดคล้องกับองค์ประกอบและกระบวนการสื่อสารข้างต้น

6.2 โครงสร้างของเอเจนต์สื่อสารใน CL

เพื่อสนับสนุนความสามารถในการสื่อสาร ภาษา CL ต้องสนับสนุนการทำงานที่เป็นผู้ส่งและผู้รับ หรือเอเจนต์ได้ คือ สามารถส่งและรับข้อความสื่อสารได้จากภายนอก อีกทั้งยังต้องมีความสามารถในการประมวลผลข้อความสื่อสารได้อย่างถูกต้อง เราขอเรียกการทำงานที่เป็นเอเจนต์ในภาษา CL นี้ว่า "เอเจนต์สื่อสาร" ซึ่งได้อธิบายภาพรวมไปบ้างแล้วในบทที่ 4 สำหรับในหัวข้อนี้จะบรรยายถึงส่วนประกอบและรายละเอียดการทำงานที่เกี่ยวข้องกับการสื่อสารของเอเจนต์โดยเฉพาะ ดังต่อไปนี้

1) ส่วนวินิจฉัย (Inference) ทำหน้าที่ประมวลผลข้อความสื่อสารที่ได้รับจากเอเจนต์ภายนอกหรือตัวเอเจนต์เอง แล้วทำการตอบสนองต่อข้อความสื่อสารนั้นตามที่ได้โปรแกรมไว้ในสคริปต์ตอบสนองต่อเหตุการณ์ (Event-action script) ซึ่งประกอบด้วย ประโยคที่มีรูปแบบ $[P_s, P_e]$: exist (Comm msg) → Action หมายถึง หากได้รับข้อความ Comm.msg ภายในช่วงเวลา $[P_s, P_e]$ แล้วจะตอบสนองด้วยการทำ Action

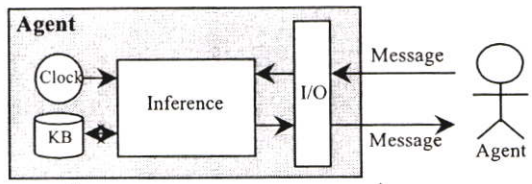
การทำงานในส่วนนี้ จะต้องสามารถเข้าใจความหมายของข้อความสื่อสารได้อย่างถูกต้องว่าข้อความที่ได้รับ คือ ข้อความประเภทไหน มีเนื้อหาอย่างไร และตรงกับข้อความที่ระบุให้มีการตอบสนองในสคริปต์ตอบสนองต่อเหตุการณ์หรือไม่ หากว่าตรง ก็ทำการตอบสนองตามที่กำหนด แต่ถ้าไม่ตรง ก็ทิ้งข้อความสื่อสารนั้นไป และไม่มีการตอบสนองแต่อย่างใด

นอกจากนี้ ส่วนวินิจฉัยยังต้องทำหน้าที่เลือกการทำงานตอบสนองต่อข้อความสื่อสาร ในกรณีที่ข้อความสื่อสารนั้นสอดคล้องกับประโยคตอบสนองมากกว่า 1 ประโยค แต่โปรแกรมต้องการให้ทำการตอบสนองเพียงประโยคเดียว

2) ส่วนฐานความรู้ (Knowledge) เป็นส่วนเก็บความรู้ของเอเจนต์ เพื่อนำไปช่วยในการวินิจฉัยของระบบ เช่น ตารางบันทึกการตอบสนองต่อเหตุการณ์ที่กำหนดในสคริปต์ตอบสนองเหตุการณ์ หรือสคริปต์สื่อสาร, ตารางบอกสถานะของข้อความสื่อสารทั้งหมดที่ปรากฏในเอเจนต์, ตารางบันทึกหมายเลขของข้อความสื่อสาร เป็นต้น

3) ส่วนอินพุท/เอาท์พุท (Input/Output) ทำหน้าที่รับและส่งข้อความสื่อสารไปสู่เอเจนต์ภายนอกหรือตัวเอเจนต์เอง ซึ่งส่วนรับหรือส่งข้อความสื่อสารนี้ อาจมีได้มากกว่า 1 ช่องทาง ขึ้นอยู่กับว่า ผู้เขียนโปรแกรมจะกำหนดให้รับหรือส่งข้อความสื่อสารด้วยวิธีและรูปแบบ (Protocol) ใดได้บ้าง ซึ่งก็จะมีจำนวนช่องทางรับ/ส่งข้อความ ตามจำนวนรูปแบบการรับ/ส่งที่กำหนดให้เอเจนต์

4) ส่วนนาฬิกา (Clock) เป็นนาฬิกาที่ใช้อ้างอิงในตัวเอเจนต์ เพื่อให้เอเจนต์สามารถตรวจสอบเวลาที่ได้รับ/ส่งข้อความสื่อสาร อีกทั้งยังใช้กำหนดช่วงเวลาของการทำงานคำสั่งของเอเจนต์ได้อีกด้วย



รูปที่ 6.2 โครงสร้างของเอเจนต์สื่อสารใน CL

เอเจนต์สื่อสารที่นำเสนอนี้ จะทำการรับข้อความสื่อสารผ่านส่วนอินพุท(Input) แล้วทำการประมวลผลข้อความสื่อสารนั้นด้วยส่วนวินิจฉัย(Inference) ซึ่งจะต้องมีการบันทึกข้อความสื่อสารที่ได้รับลงในฐานความรู้ และทำการเปรียบเทียบกับสคริปต์ตอบสนองเหตุการณ์ (Event-action script) ที่ถูกบันทึกไว้ในฐานความรู้ตั้งแต่ตอนประมวลผลโปรแกรมภาษา CL ของเอเจนต์ ในกรณีที่ข้อความสื่อสารที่เข้ามาสอดคล้องกับเหตุการณ์ที่กำหนดในประโยคตอบสนองเหตุการณ์ เอเจนต์ก็จะประมวลผลการทำงานตอบสนอง ซึ่งอาจมีการส่งข้อความสื่อสารตอบกลับไปยังเอเจนต์ผู้ส่ง โดยกระทำผ่านส่วนเอาท์พุท (Output) ตามรูปแบบ (Protocol) ที่กำหนด

6.3 รูปแบบการสื่อสารของเอเจนต์ใน CL

ข้อความสื่อสาร คือ ข้อความหรือสารที่เอเจนต์ใช้สื่อสารกัน ซึ่งแบ่งได้เป็นข้อความส่งสาร และข้อความตอบกลับ เมื่อเอเจนต์ต้องการเริ่มต้นการสนทนา เอเจนต์จะส่งข้อความส่งสารไปให้เอเจนต์ที่ต้องการสื่อสารด้วย เมื่อเอเจนต์นั้นได้รับข้อความส่งสาร ก็จะทำการตอบสนองต่อข้อความนั้นด้วยการส่งข้อความตอบกลับกลับไปยังเอเจนต์ที่ส่งข้อความมา

ในหัวข้อนี้ จะกล่าวถึงรูปแบบการโต้ตอบข้อความสื่อสารในภาษา CL โดยได้อธิบายถึงข้อความส่งสารและข้อความตอบกลับในแต่ละแบบ ตามประเภทของข้อความส่งสาร ที่ประกอบด้วยข้อความส่งสารแบบ Active และแบบ Passive ดังนี้

6.3.1 ข้อความส่งสารแบบ Active และข้อความตอบกลับ

ข้อความส่งสารแบบ Active เป็นข้อความสื่อสารที่ผู้ส่งเป็นผู้กระทำ มีรูปแบบการโต้ตอบด้วยข้อความตอบกลับ 4 แบบดังนี้

1) Tell ผู้ส่งต้องการแจ้งสารให้กับผู้รับ คือส่งประโยคบอกเล่าให้ เมื่อผู้รับได้รับ ก็จะทำงานและส่งข้อความตอบกลับ(Reply message) ด้วยข้อความซึ่งมีความหมายต่างๆ ดังนี้

- "Yes" : ผู้รับไม่เคยมีสารนั้น จึง**รับรู้สาร**ไว้
- "I know" : ผู้รับมีสารนั้นแล้ว จึง**ไม่เก็บสาร**
- "Ignore" : ผู้รับไม่สนใจสารนั้น จึง**ไม่ได้ทำอะไรกับสาร**

- "No" : สารนั้นขัดแย้งกับความรู้ของผู้รับ ฉะนั้น ผู้รับจึง**ปฏิเสธที่จะรับสาร**ถ้าเชื่อว่าความรู้ของตนถูก
- "Yes" : สารนั้นขัดแย้งกับความรู้ของผู้รับ แต่ผู้รับคิดว่าความรู้ของตนผิด จึงทำการ**แก้ไขความรู้เดิมเพื่อรับรู้สาร**นั้น

2) Query ผู้ส่งต้องการถามคำถามผู้รับ โดยส่งคำถามให้ โดยผู้รับจะหาคำตอบและส่งสารเป็นคำตอบกลับไป ดังนี้

- "Tell" : ผู้รับหาคำตอบได้ แล้ว**ส่งคำตอบกลับไป**
- "No Answer" : ผู้รับทราบว่า**ไม่มีคำตอบสำหรับคำถามนี้เลย**
- "I don't know" : ผู้รับ**ไม่สามารถหาคำตอบได้** เพราะมีความรู้ไม่สมบูรณ์
- "Ignore" : ผู้รับ**ไม่สนใจที่จะตอบคำถาม**นี้
- "No" : ผู้รับ**ปฏิเสธที่จะตอบคำถาม**นี้

3) Request ผู้ส่งสารต้องการให้ผู้รับกระทำคำสั่งที่ปรากฏในสารที่ส่งไป โดยส่งประโยคคำสั่งให้ โดยผู้รับจะทำคำสั่งและส่งผลกลับไป ด้วยกรณีต่างๆ ดังนี้

- "Yes" : ผู้รับ**กระทำคำสั่งนั้นได้อย่างสมบูรณ์**
- "Failed" : ผู้รับ**กระทำคำสั่งไม่สำเร็จ**
- "Ignore" : ผู้รับ**ไม่สนใจที่จะกระทำคำสั่ง**
- "No" : ผู้รับ**ปฏิเสธที่จะทำคำสั่ง**นั้นให้

4) Delete ผู้ส่งส่งสารขอลบความรู้ของผู้รับ โดยผู้รับจะตอบสนอง ด้วยกรณีต่างๆ ดังนี้

- "Yes" : ผู้รับ**ลบความรู้เดิม**ตามระบุที่ในสาร
- "No" : ผู้รับ**ไม่มีความรู้ดังกล่าว** จึงไม่ได้ลบออก
- "Ignore" : ผู้รับ**ไม่สนใจที่จะลบความรู้**ของตน
- "No" : ผู้รับ**ปฏิเสธที่จะลบความรู้**ตามที่ระบุ

ในกรณีของการเปลี่ยนแปลงความรู้เดิมของผู้รับด้วยความรู้ใหม่ (Revise) สามารถทำได้โดยการใช้ Delete แล้วตามด้วยการ Tell และทุกๆ การสื่อสารข้างต้น ถ้าส่งข้อความมาผิดไวยากรณ์ (Syntax) ผู้รับก็จะแจ้งความผิดพลาดของการสื่อสารกลับไป

6.3.2 ข้อความส่งสารแบบ Passive และข้อความตอบกลับ

ข้อความส่งสารแบบ Passive เป็นข้อความสื่อสารที่ผู้ส่งขอให้ผู้รับกระทำให้ มีรูปแบบการโต้ตอบด้วยข้อความตอบกลับ 4 แบบดังนี้

1) Please Tell ผู้ส่งสารขอรับข้อมูลจากผู้รับ ซึ่งผู้ส่งจะระบุข้อมูลที่ต้องการหรือไม่ก็ได้ โดยผู้รับจะส่งเป็นข้อความกลับไป ด้วยกรณีต่างๆ ดังนี้

- "Tell" : กรณีที่ระบุข้อมูลที่ต้องการ ถ้า**ผู้รับมีข้อมูลนั้น**
- "Tell" : กรณีที่ไม่ระบุข้อมูลที่ต้องการ และ**ผู้รับมีข้อมูล**ให้
- "No Tell" : ผู้รับ**ไม่มีข้อมูลนั้นในฐานความรู้ของตน** หรือ ผู้รับ**ไม่ต้องการให้ข้อมูล**
- "Ignore" : ผู้รับ**ไม่สนใจการร้องขอ**นั้น

2) Please Query ผู้ส่งสารขอให้ผู้รับถามคำถาม โดยผู้รับจะส่งเป็นข้อความกลับไป ด้วยกรณีต่างๆ ดังนี้

- "Query" : ผู้รับ**มีคำถาม**
- "No Query" : ผู้รับ**ไม่ต้องการถามคำถามใดๆ**
- "Ignore" : ผู้รับ**ไม่สนใจการร้องขอ**นั้น

3) Please Request ผู้ส่งสารขอให้ผู้รับสั่งงาน โดยผู้รับจะส่งเป็นข้อความกลับไป ด้วยกรณีต่างๆ ดังนี้

- "Request" : ผู้รับ**ต้องการสั่งงาน**ให้แก่ผู้ส่ง
- "No Request" : ผู้รับ**ไม่ต้องการสั่งงานใดๆ**
- "Ignore" : ผู้รับ**ไม่สนใจการร้องขอ**นั้น

4) Please Delete ผู้ส่งสารขอให้ผู้รับแจ้งให้มีการลบความรู้ของผู้ส่ง โดยผู้รับจะส่งเป็นข้อความกลับไป ด้วยกรณีต่างๆ ดังนี้

- "Delete" : ผู้รับ**ต้องการลบความรู้ของผู้ส่ง**
- "No Delete" : ผู้รับ**ไม่ต้องการลบความรู้ใดๆ ของผู้ส่ง**
- "Ignore" : ผู้รับ**ไม่สนใจการร้องขอ**นั้น

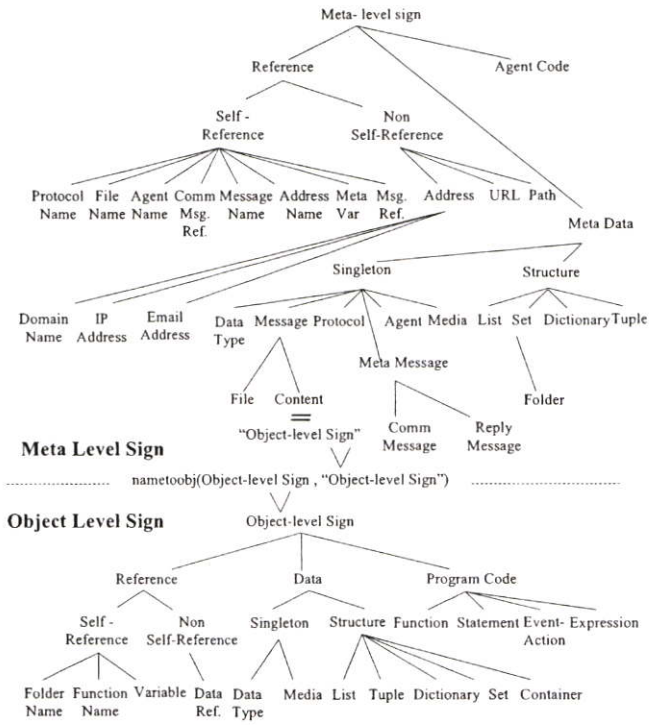
สรุปแล้ว ข้อความสื่อสารแบบ Passive ประกอบด้วยข้อความสื่อสารประเภท Please Tell, Please Query, Please Request และ Please Delete ซึ่งเป็นรูปแบบที่ขอให้ผู้รับ ส่งข้อมูล ถาม สั่งงาน และลบความรู้ของตน ตามลำดับ โดยข้อความที่ส่งกลับจะคล้ายคลึงกัน คือ ประกอบด้วย Comm message, No Comm message และ Ignore

6.4 **สัญลักษณ์** (Communication Sign)

สัญลักษณ์ที่ใช้ในการสื่อสารใน CL แบ่งเป็น สัญลักษณ์ที่ระดับเมต้า (Meta-level Sign) และที่ระดับวัตถุ (Object-Level Sign)

6.4.1 **สัญลักษณ์ระดับเมต้า** (Meta-Level Sign)

เป็นสัญลักษณ์ที่เข้าใจได้ในระดับเอเจนต์หรือระดับเมต้า ส่วนสัญลักษณ์ระดับวัตถุที่ใช้สื่อสารระหว่างเอเจนต์นั้น เป็นเพียงข้อความที่เป็นสายอักขระ หรือ bit stream ไม่มีความหมายที่ระดับเมต้า แต่จะสามารถสื่อความหมายได้ เมื่อมีการตีความหมายในระดับวัตถุ ซึ่งเป็นขบวนการถอดเครื่องหมาย " " คือ เปลี่ยนจากการเป็นชื่อเป็นความหมายระดับวัตถุทำโดยฟังก์ชัน nametoobj() เนื่องจาก nametoobj("x") จะให้ผลเป็นสัญลักษณ์ x ที่ระดับวัตถุ เช่น nametoobj ("print(1)") จะให้คำสั่งการพิมพ์ print(1) ในทางกลับกันก็สามารถแปลงสัญลักษณ์ระดับเมต้าให้เป็นสัญลักษณ์ระดับวัตถุได้ด้วยฟังก์ชัน objtoname() สัญลักษณ์ระดับเมต้าประกอบด้วยส่วนต่างๆ ตามรูปที่ 6.2 ด้านบน



รูปที่ 6.3 Communication Sign

1) ส่วน Reference ใช้อ้างอิงถึงสัญลักษณ์อื่นๆ ที่ระดับเมต้า แบ่งเป็น *Self-Reference* ซึ่งเป็นการอ้างอิงที่เป็นชื่อ (สังเกตว่าชื่อเป็นการอ้างอิงถึงตัวเองไม่ชี้ไปยังสิ่งอื่น) ได้แก่ ชื่อ Protocol, Agent และ *Non-Self-Reference* ที่เป็นการอ้างอิงถึงชื่ออีกทีผ่าน Link การอ้างอิงแบบนี้ได้แก่ URL, Path และ Address

2) ส่วน Meta Data เป็นส่วนข้อมูลในระดับเมต้า และสารที่ใช้สื่อสารกัน ซึ่งสารประกอบด้วยคุณสมบัติ ประเภท และ เนื้อหาของสาร โดยส่วนเนื้อหาจะเป็นที่เก็บสารที่เป็นชื่อของสัญลักษณ์ระดับวัตถุในรูป "Object-level sign"

3) ส่วน Agent Code เป็นส่วนโค้ดโปรแกรมที่ใช้สร้างส่วนต่างๆ ของเอเจนต์ ซึ่งจะอธิบายไว้ในบทที่ 7

6.4.1.1 ข้อความสื่อสารระดับเมต้า

จากที่อธิบายในหัวข้อที่ 6.3 สรุปได้ว่า การสื่อสารจะเกิดขึ้นได้ต้องประกอบด้วยอย่างน้อย 5 ส่วน คือ โหมดการสื่อสาร(Mode), สาร(Message), ผู้ส่ง(Sender), ผู้รับ(Receiver) และโปรโตคอล(Protocol) ดังนั้น จึงกำหนดให้ข้อความสำหรับสื่อสาร (Communication Message) มีรูปแบบดังนี้

Mode(Message, Sender, Receiver, id, Resp_id)

โดย id คือเลขอ้างอิงถึงข้อความสื่อสารที่ส่งไป และ Resp_id คือหมายเลขสารที่ข้อความนี้ใช้ตอบกลับ ซึ่งทั้งคู่จำเป็นต้องใช้เพื่อให้การโต้ตอบของเอเจนต์ได้ใช้อ้างอิงกับข้อความสื่อสารได้อย่างถูกต้อง ตัวอย่างเช่น

tell("/"Hi/"", "john@mail.com", "mary@mail.com", 1, None)

หมายถึง ข้อความสื่อสารที่ใช้แจ้งว่า "Hi" (ใน Python ใช้ /" เพื่อใส่ " ภายในสตริง) ไปยัง "mary@mail.com" โดยมี "john@mail.com" เป็นผู้ส่งสาร ซึ่งหมายเลขของข้อความคือ 1 และไม่ได้ส่งข้อความนี้เพื่อตอบกลับข้อความใด ส่วนประกอบต่างๆ ในข้อความสื่อสาร มีดังนี้

- *โหมด* บอกรูปแบบของการติดต่อ ได้แก่ Tell, Query, Request, Delete, Please Tell, Please Query, Please Request, Please Delete, Yes, I know, Ignore, No, Tell, No Answer, I don't know, Failed, No Info, Query, No Question, Request, No Command, Delete และ No Delete
- *สาร* เป็นสิ่งที่ใช้ส่งกันระหว่างเอเจนต์ แบ่งได้ 3 แบบ คือ โค้ด, ข้อมูล และตัวอ้างอิง (Reference) ซึ่งกำหนดเป็นชื่อของ Object-level sign เช่น "checkEmail()" เป็นโค้ด, "/"Hello/"" เป็นข้อมูล เป็นต้น
- *ผู้ส่งและผู้รับสาร* ทำหน้าที่ส่งและรับสาร แทนด้วยข้อมูลอ้างอิงในระดับเมต้า ได้แก่ ชื่อเอเจนต์, ชื่อคน(User), IP-Address และ E-mail Address
- *หมายเลขข้อความสื่อสาร* ใช้กำหนดหมายเลขของข้อความสื่อสารที่ส่งออกไป เป็นเลขจำนวนเต็ม ซึ่งจะถูกกำหนดโดยอินเตอร์พรีตเตอร์

- *หมายเลขข้อความที่ตอบสนอง* ใช้กำหนดหมายเลขของข้อความที่ข้อความสื่อสารนี้ต้องการตอบสนอง เป็นเลขจำนวนเต็ม

6.4.1.2 **ข้อความตอบกลับในระดับเมต้า** (Reply Message)

การสื่อสารของเอเจนต์จะมีข้อความตอบกลับแตกต่างกัน

1) **ข้อความตอบกลับของข้อความสื่อสารแบบ** Active

- *กรณี Tell* มี "Yes", "I know", "Ignore" และ "No"
- *กรณี Query* มี "Tell", "No Answer", "I don't know" และ "Ignore"
- *กรณี Request* มี "Yes", "Failed" และ "Ignore"
- *กรณี Delete* มี "Yes", "No" และ "Ignore"

2) **ข้อความตอบกลับของข้อความสื่อสารแบบ** Passive

- *กรณี Please Tell* มี "Tell", "No Info" และ "Ignore"
- *กรณี Please Query* มี "Query", "No Question" และ "Ignore"
- *กรณี Please Request* มี "Request", "No Command" และ "Ignore"
- *กรณี Please Delete* มี "Delete", "No Delete" และ "Ignore"

6.4.2 **สัญลักษณ์ระดับวัตถุ** (Object-Level Sign)

สัญลักษณ์ระดับวัตถุใช้แทนคน สัตว์ สิ่งของ และความสัมพันธ์ของสิ่งเหล่านี้ ซึ่งแท้จริงก็คือ ส่วนโปรแกรมคอมพิวเตอร์ และข้อมูลที่โปรแกรมใช้ประมวลผล สัญลักษณ์ระดับวัตถุ ประกอบด้วยส่วนต่างๆ ดังรูปที่ 6.2 ด้านล่าง

1) ส่วน Reference เป็นส่วนที่ทำการอ้างอิงถึงสิ่งต่างๆ ที่อยู่ในระดับวัตถุ แบ่งเป็น *Self-Reference* ซึ่งเป็นการอ้างอิงที่เป็นชื่อ ได้แก่ ชื่อฟังก์ชัน ชื่อตัวแปร และ *Non Self-Reference* เป็นการอ้างอิงถึงสิ่งหนึ่งๆ ผ่าน Link และสามารถใช้การติดตามไปยังสิ่งที่ Link อ้างถึง ซึ่งสุดท้ายก็จะอ้างอิงไปถึงชื่อหรือวัตถุ ซึ่งใน CL เราใช้ Non Self-Reference ในการอ้างอิงถึงข้อมูลระดับวัตถุที่ใช้สื่อสารกันระหว่างเอเจนต์

2) ส่วน Data เป็นข้อมูลในระดับวัตถุ แบ่งเป็น Singleton และ Structure ซึ่งเป็นที่คุ้นเคยในโลกของโปรแกรม ได้แก่ Number, String, Boolean, Time, None, List, Tuple และ Dictionary

3) ส่วนโค้ดโปรแกรม เป็นโค้ดโปรแกรมคอมพิวเตอร์ ได้แก่ ฟังก์ชันและประโยคคำสั่ง

6.5 การส่งข้อความสื่อสาร

เอเจนต์จะสามารถส่งข้อความสื่อสารทั้งที่เป็นข้อความส่งสารและข้อความตอบกลับไปยังเอเจนต์อื่นได้นั้น นอกจากาจะมีสารหรือข้อความสื่อสารที่ต้องการส่งแล้ว เอเจนต์จำเป็นจะต้องมีการระบุกระบวนการและรูปแบบของการสื่อสาร ที่เรียกว่า โปรโตคอล (Protocol) เพื่อใช้กำหนดช่องทางในการส่งข้อความสื่อสารไปยังสภาพแวดล้อมภายนอกได้นั่นเอง ในภาษา CL สามารถส่งข้อความสื่อสารได้ด้วยคำสั่ง send ซึ่งมีรูปแบบ ดังนี้

```
send(Comm. msg., protocol)
```

โดย Comm. msg. คือ ข้อความสื่อสารที่กำหนดด้วยรูปแบบ Mode(Message, Sender, Receiver, id, Resp_id) และ Protocol คือ โปรโตคอล ซึ่งประกอบด้วยชื่อโปรโตคอล และหมายเลขพอร์ต เช่น (http,80) คือโปรโตคอล http ใช้พอร์ต 80 เป็นต้น หรือสามารถกำหนดหมายเลขพอร์ตเพียงอย่างเดียวก็ได้

การทำคำสั่ง send จะเสร็จสิ้นได้ก็ต่อเมื่อทำการส่งข้อความสื่อสารไปยังเอเจนต์อื่นได้สำเร็จ ผู้เขียนโปรแกรมสามารถกำหนดช่วงเวลาสำหรับการส่งข้อความสื่อสารได้ด้วยรูปแบบ

$[P_s, P_e]$: send(Comm. Msg., protocol)

หมายถึง ให้ทำการส่งข้อความสื่อสาร Comm. Msg. ผ่านโปรโตคอล Protocol โดยจะต้องเริ่มต้นและเสร็จสิ้นการส่งข้อความภายในช่วงเวลา $[P_s, P_e]$ ในกรณีที่ไม่สามารถทำงานให้สำเร็จภายในช่วงเวลานี้ได้ โปรแกรมจะแสดงความผิดพลาดที่เกิดขึ้น

6.6 การตอบสนองต่อข้อความสื่อสาร

ภาษา CL สามารถเขียนโปรแกรมให้เอเจนต์สามารถทำการตอบสนองต่อข้อความสื่อสารที่ได้รับได้ ด้วยการกำหนดสคริปต์ตอบสนองเหตุการณ์ (Event-action script) ซึ่งมีประโยคตอบสนองต่อเหตุการณ์เป็นองค์ประกอบพื้นฐาน ประโยคตอบสนองต่อเหตุการณ์มีรูปแบบดังนี้

$[P_s, P_e]$: exist(Comm. Msg.) → Action

หมายถึง หากได้รับข้อความสื่อสาร Comm. Msg. แล้ว CL จะทำการประมวลผลการทำงาน Action เพื่อตอบสนองต่อข้อความสื่อสารนั้น ในกรณีที่กำหนดช่วงเวลา $[P_s, P_e]$ ให้กับเหตุการณ์การเกิดข้อความสื่อสาร หมายความว่า ข้อความสื่อสาร (Comm. Msg.) จะต้องเกิดขึ้นในช่วงเวลา $[P_s, P_e]$ โปรแกรมจึงจะทำงานตอบสนอง ดังนั้นจึงกล่าวได้ว่า ภายในช่วงเวลา $[P_s, P_e]$ สามารถเกิดข้อความสื่อสารที่กำหนดได้มากกว่าหนึ่งครั้ง ต่างจากการทำงานคำสั่งหรือประโยค (Statement) ตามช่วงเวลาที่กำหนดว่าจะต้องทำงานให้เสร็จภายในช่วงเวลาเท่านั้น ทำให้ภายใน

ช่วงเวลาดังกล่าวจะมีการทำงานคำสั่งเพียงหนึ่งครั้งและต้องรอจนกว่าจะหมดช่วงเวลาจึงจะไปทำคำสั่งอื่นได้ ช่วงเวลา [P_s, P_e] นี้อาจเป็นช่วงเวลาแบบระบุเวลา, แบบกึ่งระบุเวลา หรือแบบไม่ระบุเวลาก็ได้

นอกจากนี้ ผู้เขียนยังสามารถกำหนดช่วงเวลาการทำงานคำสั่งได้ด้วยรูปแบบ [P_{s1}, P_{e1}] : Action จึงทำให้รูปแบบการตอบสนองต่อข้อความสื่อสาร เป็นได้ดังนี้

$$[P_s, P_e] : \text{exist(Comm. Msg.)} \rightarrow [P_{s1}, P_{e1}] : \text{Action}$$

แต่ถึงอย่างไรก็ตาม โปรแกรมเมอร์ต้องมีความระมัดระวังในการกำหนดช่วงเวลาของการทำงานตอบสนอง เนื่องจากเราไม่สามารถทราบได้ว่า จะได้รับข้อความสื่อสารเมื่อไร และจะสังเกตได้ว่า ช่วงเวลาของการทำงานตอบสนองควรจะกำหนดขึ้นภายหลังการเกิดข้อความสื่อสาร ดังนั้นจึงควรกำหนดให้ $P_e < P_{s1}$ นั่นเอง แต่ถึงอย่างไรก็ตาม ข้อความสื่อสารอาจเกิดขึ้นก่อนจุดเวลา P_e ก็ได้ โดยโปรแกรมเมอร์สามารถกำหนดช่วงเวลาด้วยวิธีการอ้างอิงจากจุดเวลา now ซึ่งเป็นจุดเวลาปัจจุบันขณะที่ทำการประมวลผล เช่น exist(Comm. Msg.) → [(now + 10min), _] : Action หมายถึง เมื่อมีการเกิดข้อความ Comm. Msg. ขึ้น จะเริ่มทำการตอบสนองด้วยการทำ Action ณ จุดเวลาในอีก 10 นาทีข้างหน้า เป็นต้น

6.7 การตอบสนองแบบบทสนทนา

เนื่องจากในชีวิตประจำวัน การสื่อสารของเรามักมีลักษณะต่อเนื่องเป็นชุด ทำให้สคริปต์สื่อสารใน CL มีลักษณะเป็นชุดของการตอบสนองต่อเนื่องที่คล้ายบทสนทนา ซึ่งเกิดจากการนำประโยคตอบสนองต่อเหตุการณ์ในหัวข้อที่แล้วมารวมกัน เรียกว่า ประโยคชุดการตอบสนองต่อเหตุการณ์ หรือประโยคชุดการสื่อสาร แบ่งเป็น 6 แบบได้แก่ แบบ Independent Repeatable and Partial, Independent Repeatable and Total, Sequence, Unordered, One และ Nested

สำหรับการกำหนดช่วงเวลา [s_i, e_i] ในการตรวจสอบเหตุการณ์หรือข้อความสื่อสารที่ได้รับในแต่ละชุดการตอบสนองนั้น โปรแกรมเมอร์สามารถกำหนดได้อย่างอิสระ เนื่องจากช่วงเวลาดังกล่าวนำไปใช้เป็นเงื่อนไขในการตรวจสอบค่าความจริงของการได้รับข้อความเท่านั้น ไม่เหมือนการกำหนดช่วงเวลาสำหรับประโยค (Statement) ในประโยคชุดคำสั่ง (Compound Statement) แบบต่างๆ ซึ่งมีเงื่อนไขในการกำหนดแตกต่างกันไป ดังที่อธิบายไปในบทที่แล้ว ยกเว้นในบทสนทนาแบบ Sequence ที่มีข้อจำกัดอยู่บ้างเล็กน้อย

1) **แบบ** Independent Repeatable and Partial มีรูปแบบดังนี้

$[s_1, e_1]$: exist(Comm. Msg_1) → $statement_1$
:
$[s_n, e_n]$: exist(Comm. Msg_n) → $statement_n$

ใช้ระบุให้เอเจนต์รับรู้ข้อความสื่อสาร Comm. Msg. แล้วตอบสนองด้วยการกระทำ statement ซึ่งสามารถตอบสนองครั้งละกี่บรรทัดก็ได้ โดยจะเริ่มตอบสนองจากบรรทัดใดก่อนก็ได้ และทำซ้ำกี่ครั้งก็ได้ ซึ่งประโยคชุดการสื่อสารแบบนี้จะอยู่ที่ระดับบน (Top Level) ของสคริปต์สื่อสารเสมอ

2) **แบบ** Independent Repeatable and Total มีรูปแบบดังนี้

comm_independent_total :
$[s_1, e_1]$: exist(Comm. Msg_1) → $statement_1$
:
$[s_n, e_n]$: exist(Comm. Msg_n) → $statement_n$

ใช้ระบุให้เอเจนต์รับรู้ข้อความสื่อสาร Comm. Msg. แล้วตอบสนองด้วยการกระทำ statement ซึ่งสามารถตอบสนองครั้งละกี่บรรทัดก็ได้ โดยจะเริ่มตอบสนองจากบรรทัดใดก่อนก็ได้ และทำซ้ำกี่ครั้งก็ได้ โดยการตอบสนองตามประโยคชุดการสื่อสารนี้จะสมบูรณ์ได้ก็ต่อเมื่อมีการตอบสนองต่อข้อความสื่อสารจนครบทุกบรรทัดแล้ว

3) **แบบ** Sequence มีรูปแบบดังนี้

comm_sequence :
$[s_1, e_1]$: exist(Comm. Msg_1) → $statement_1$
:
$[s_n, e_n]$: exist(Comm. Msg_n) → $statement_n$

ใช้ระบุให้เอเจนต์รับรู้ข้อความสื่อสาร Comm. Msg. แล้วตอบสนองด้วยการกระทำ statement ตามลำดับจากบนลงล่าง ทีละบรรทัดตามลำดับจนครบทั้งชุดการสื่อสารจึงจะถือว่ามีการตอบสนองตามประโยคชุดการสื่อสารแบบนี้ได้อย่างสมบูรณ์

เนื่องจากประโยคชุดการสื่อสารนี้กำหนดให้การตอบสนองเกิดขึ้นอย่างเป็นลำดับ ดังนั้นช่วงเวลาของการได้รับข้อความสื่อสารก็ควรมีลำดับด้วยเช่นกัน นั่นคือต้องกำหนดให้ $s_1 < e_1 < s_2 < e_2 < \ldots < s_n < e_n$ โดยที่ s_i และ e_i เป็นจุดเวลาเริ่มและสิ้นสุดของของการได้รับ Comm. Msg.

4) **แบบ** Unordered มีรูปแบบดังนี้

```
comm_unordered :
    [s1, e1] : exist(Comm. Msg1) → statement1
                        :
    [sn, en] : exist(Comm. Msgn) → statementn
```

ใช้ระบุให้เอเจนต์รับรู้ข้อความสื่อสาร Comm. Msg. แล้วตอบสนองด้วยการกระทำ statement ทีละบรรทัด โดยไม่ต้องเรียงลำดับ เมื่อมีการตอบสนองจนครบทั้งชุดการสื่อสารจึงจะถือว่ามีการตอบสนองตามประโยคชุดการสื่อสารแบบนี้ได้อย่างสมบูรณ์

5) **แบบ** One มีรูปแบบดังนี้

```
comm_one :
    [s1, e1] : exist(Comm. Msg1) → statement1
                        :
    [sn, en] : exist(Comm. Msgn) → statementn
```

ใช้ระบุให้เอเจนต์รับรู้ข้อความสื่อสาร Comm. Msg. แล้วตอบสนองด้วยการกระทำ statement ที่บรรทัดใดก็ได้เพียงบรรทัดเดียว ก็จะถือว่าสามารถตอบสนองตามชุดการสื่อสารแบบนี้ได้สำเร็จ ในกรณีที่ข้อความสื่อสารที่ได้รับตรงกับกฎการตอบสนอง exist(Comm. Msg.) → statement มากกว่า 1 บรรทัด เอเจนต์จะทำการเลือกตอบสนองต่อข้อความนั้นด้วยการใช้บรรทัดใดบรรทัดหนึ่งเท่านั้น

6) **แบบ** Nested

จะถือว่าประโยคชุดการสื่อสารแบบ Independent Repeatable and Partial, Independent Repeatable and Total, Sequence, Unordered และ One เป็นเสมือน 1 ประโยคตอบสนองต่อเหตุการณ์ ดังนั้นการสร้างประโยคชุดการสื่อสารแบบ Nested ทำได้โดยใส่ประโยคชุดการสื่อสารทั้งสี่เหมือน 1 ประโยคตอบสนอง ไว้ภายในประโยคชุดการสื่อสารแบบ Independent Repeatable and Partial หรือ Independent Repeatable and Total หรือ Sequence หรือ Unordered หรือ One ตัวอย่างเช่น

```
comm_sequence :
    [s1, e1] : exist(Comm. Msg1) → statement1
    [sn, e2] : exist(Comm. Msg2) → statement2
    comm_one :
        [s3, e3] : exist(Comm. Msg3) → statement3
        [s4, e4] : exist(Comm. Msg4) → statement4
```

บทที่ 7
การประมวลผลของ CL

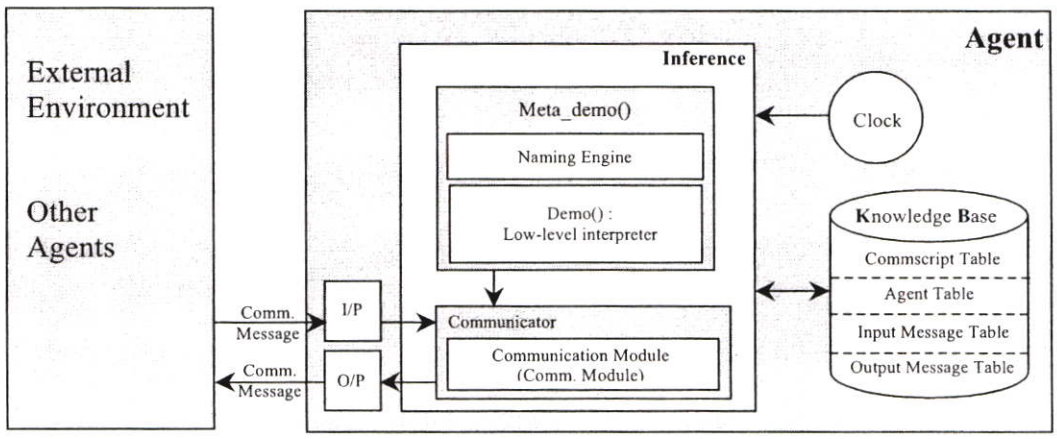
เนื้อหาในบทนี้จะกล่าวถึงรายละเอียดการประมวลผลของเอเจนต์ CL ตามโครงสร้างโดยรวมที่ได้อธิบายไว้ในบทที่ 4 โดยกระบวนการประมวลผล แบ่งออกได้เป็น 2 ระดับ คือ ระดับเมต้า (ระดับบน) และระดับวัตถุ (ระดับล่าง) ซึ่งจะได้อธิบายโครงสร้างการทำงานไว้ในหัวข้อที่ 7.1 และ 7.2 ตามลำดับ

สำหรับการประมวลผลภาษา CL ที่ระดับวัตถุ ภาษา CL จะถูกแปลความหมายโดยอินเตอร์พรีตเตอร์เช่นเดียวกับในภาษาไพธอน การทำงานของอินเตอร์พรีตเตอร์ภาษา CL มี 3 ส่วน คือ การประมวลผลประโยคต่างๆ (statements), การให้คำนิยามฟังก์ชัน และ การประมวลผลสคริปต์ตอบสนองต่อเหตุการณ์ภายนอก หรือสคริปต์สื่อสาร ดังที่ได้อธิบายในหัวข้อที่ 7.3

เนื่องจาก อินเตอร์พรีตเตอร์ของ CL จะทำการประมวลผลคำสั่งตามเวลาที่กำหนดในโปรแกรม ซึ่งทำงานเหมือน Deduction [17] จึงขอเรียกคำสั่งอินเตอร์พรีตเตอร์ว่าเป็น Demonstrator หรือเรียกสั้นๆ ว่า demo และต่อจากนี้ไป เราจะแทนอินเตอร์พรีตเตอร์นี้ด้วย demo

7.1 **เอเจนต์สื่อสาร** (Communication Agent)

ภาษา CL มีการทำงานที่สามารถรับส่งและประมวลผลข้อความสื่อสารได้ ซึ่งเป็นลักษณะการทำงานของเอเจนต์ เราจึงนำภาษา CL มาใช้พัฒนาเอเจนต์ที่สื่อสารกันได้ เรียกว่า "Communication Agent" หรือ "เอเจนต์สื่อสาร" ซึ่งได้ถ่ายทอดสถาปัตยกรรมจากเอเจนต์ CL ตามรูปที่ 7.1



รูปที่ 7.1 เอเจนต์สื่อสาร

7.1.1 **องค์ประกอบของเอเจนต์สื่อสาร** เอเจนต์สื่อสาร ประกอบด้วยส่วนต่างๆ ดังนี้

1) **อินพุท/เอาท์พุท**(Input/Output) เป็นช่องทางที่ใช้รับและส่งข้อความสื่อสารกับสิ่งแวดล้อมภายนอกหรือแม้แต่กับตัวเอเจนต์เอง ช่องทางที่ใช้รับ/ส่งนี้สามารถมีได้หลายช่องทาง ขึ้นอยู่กับการกำหนดไว้ในกระบวนการสร้างเอเจนต์ ซึ่งได้อธิบายไปแล้วในบทที่ 4 โดยผู้เขียนโปรแกรมสามารถกำหนดอินพุท/เอาท์พุทให้กับเอเจนต์ได้ด้วยลิสต์ของพอร์ต(Port) และโปรโตคอล (Protocol) เช่น กำหนดให้ input = [4444, 5555] หมายถึง เอเจนต์จะมีช่องทางรับข้อมูลได้ 2 ทาง คือ ทางพอร์ต 4444 และพอร์ต 5555

2) **ส่วนวินิจฉัย**(Inference) ใช้รับข้อความสื่อสารเพื่อประมวลผลแล้วทำการตอบสนองต่อข้อความที่ได้รับตามที่ระบุไว้ในสคริปต์ตอบสนองต่อเหตุการณ์ (Event-action script) โดยทำหน้าที่วินิจฉัยตามความรู้ที่มีในฐานความรู้ (KB) เพื่อค้นหาคำตอบต่างๆ ให้กับเอเจนต์ผู้ส่ง ส่วนวินิจฉัยจะประกอบด้วย

- *ส่วน Communicator*

ทำหน้าที่สื่อสาร รับ/ส่งข้อความสื่อสาร ในกรณีที่ได้รับข้อความ ส่วนนี้จะทำการบันทึกรายละเอียดของข้อความสื่อสารที่ได้รับเก็บไว้ในตารางข้อความนำเข้า (Input Message Table) ในฐานความรู้ และในทางกลับกัน สำหรับการส่งข้อความก็จะทำการบันทึกรายละเอียดของข้อความสื่อสารที่ส่งไว้ในตารางข้อความส่งออก (Output Message Table) ที่เก็บไว้ในฐานความรู้เช่นกัน

- *Naming Engine*

ทำหน้าที่แปลงสัญลักษณ์ไปมาระหว่างระดับวัตถุและเมต้า จากที่กล่าวไปในบทที่ 6 ว่าสัญลักษณ์ในภาษา CL มี 2 ระดับ ได้แก่ สัญลักษณ์ระดับเมต้า และสัญลักษณ์ระดับวัตถุ ข้อความสื่อสารที่ได้รับจะเป็นสัญลักษณ์ในระดับเมต้า ส่วนวินิจฉัยต้องทำหน้าที่ประมวลผลข้อความสื่อสารนั้นด้วยการแปลงให้เป็นสัญลักษณ์ระดับวัตถุ เพื่อให้เข้าใจถึงความหมายของข้อความที่เอเจนต์ผู้ส่งต้องการจะสื่อ ในทางกลับกัน หากมีการส่งข้อความออกไป ส่วนวินิจฉัยก็ต้องทำการแปลงสัญลักษณ์ในระดับวัตถุให้เป็นระดับเมต้าเสียก่อน

โดยฟังก์ชันที่ใช้แปลงสัญลักษณ์ทั้ง 2 ระดับ ได้แก่ ฟังก์ชัน objtoname() และ nametoobj() เป็นฟังก์ชันที่แปลงสัญลักษณ์จากระดับวัตถุให้เป็นเมต้า และจากระดับเมต้าให้เป็นวัตถุ ตามลำดับ ซึ่งฟังก์ชันทั้งสองนี้จะส่วนเรียกใช้โดย meta_demo() ที่ทำหน้าที่ประมวลผล

- *meta_demo()*

เป็นอินเตอร์พรีตเตอร์ในระดับเอเจนต์ ทำหน้าที่ประมวลผลข้อความและการตอบสนองที่ระดับเอเจนต์ โดยทำการเรียกส่วน Naming Engine ให้ทำการแปลงข้อความและคำสั่งระดับเมต้าให้เป็นระดับวัตถุ แล้วจึงส่งให้ demo ประมวลผลข้อความและคำสั่งนั้นที่อยู่ในระดับวัตถุต่อไป

จะสังเกตว่า demo ก็คืออินเตอร์พรีตเตอร์ในระดับล่าง ซึ่งใช้ประมวลผลคำสั่งที่เป็นโค้ดโปรแกรม ส่วนการทำงานของ demo หรืออินเตอร์พรีตเตอร์ระดับล่างจะอธิบายไว้ในหัวข้อ 7.2

3) **ส่วนฐานความรู้** ใช้เก็บความรู้ของเอเจนต์ ประกอบด้วย

- *ส่วนตารางการตอบสนองเหตุการณ์ (Event-action script table)*

สคริปต์ตอบสนองต่อเหตุการณ์ (Event-action script) เป็นสคริปต์ที่ระบุว่าเมื่อมีเหตุการณ์เกิดขึ้นแล้วให้ทำการตอบสนองต่อเหตุการณ์นั้นอย่างไร ซึ่งสคริปต์จะประกอบด้วยประโยคตอบสนองต่อเหตุการณ์ซึ่งมีรูปแบบ

Event → Action

แต่เนื่องจาก เหตุการณ์ในภาษา CL จะเน้นที่การได้รับ(เกิด)ข้อความสื่อสาร ซึ่งสามารถกำหนดการตอบสนองการสื่อสารได้ว่า

exist(Comm_msg) → Action

หมายถึง เมื่อปรากฏมีข้อความสื่อสาร Comm_msg แล้วให้กระทำ Action เช่น

```
exist(Tell(?msg, agent("John"), _, ?id, _)) →
        {x=assert(msg); send(Yes(meta(x),agent("CL"),agent(snd),id),
(smtp,25))}
```

หมายถึง ถ้าได้รับข้อความจาก "John" แล้วให้รับรู้สารนั้น สังเกตว่า msg เป็นตัวแปรที่แทนสาร และ "_" เป็นส่วนที่ไม่ระบุค่า จะเป็นอะไรก็ได้ และ "John" คือชื่อเอเจนต์ และจากความสัมพันธ์ข้างต้น สามารถสรุปได้ว่า *เมื่อมีการกล่าวถึงสคริปต์ตอบสนองต่อเหตุการณ์ จะหมายถึงสคริปต์สำหรับการสื่อสาร (Commscript : Communication script) ทันที*

สคริปต์ตอบสนองต่อเหตุการณ์จะถูกบันทึกในฐานความรู้ด้วยโครงสร้างของลิสต์ของประโยคตอบสนองต่อเหตุการณ์ซึ่งประกอบด้วย ช่วงเวลาของการเกิดเหตุการณ์ (Time) เหตุการณ์(Event) และการทำงานตอบสนอง(Action) เช่นจากตัวอย่างก่อนหน้าจะได้ตารางการตอบสนองเหตุการณ์ ดังนี้

Time	Event	Action
[_,_]	Tell(?msg, agent("John"), _, ?id, _)	{x=assert(msg); send(Yes(meta(x), agent("CL"), agent(snd), id), (smtp,25))}

เมื่อเอเจนต์ได้รับข้อความสื่อสาร ก็จะนำมาเปรียบเทียบกับตารางนี้ เพื่อค้นหาประโยคตอบสนองที่สอดคล้องตรงกัน ในกรณีที่ประโยคตอบสนองต่อเหตุการณ์เป็นประโยคชุดการตอบสนองที่มีลักษณะเป็นบทสนทนานั้น ลิสต์จะทำการเพิ่มตัวชี้ที่ชี้ไปยังประโยคตอบสนองทั้งหมดในบทสนทนานั้นอีกที ซึ่งกำหนดด้วยโครงสร้างแบบเดียวกัน

- *ส่วนตารางเอเจนต์ (Agent table)*

เป็นตารางที่ระบุชื่อ(Name) และที่อยู่(Address) ของเอเจนต์ ซึ่งที่อยู่ของเอเจนต์นั้นสามารถมีได้มากกว่า 1 ที่อยู่ ในฐานความรู้จึงเก็บที่อยู่เป็นลักษณะของลิสต์ที่อยู่ ตารางเอเจนต์มีรูปแบบดังนี้

Name	Address
"John"	[$Address_1$, $Address_2$, …, $Address_n$]

เมื่อได้รับข้อความสื่อสาร เอเจนต์จะทราบว่าข้อความนั้นส่งมาจากเอเจนต์ชื่ออะไร และอยู่ที่ไหน (เอเจนต์ผู้รับจะทราบที่อยู่ เช่น IP ของเอเจนต์ผู้ส่ง) ในกรณีที่เอเจนต์ผู้รับไม่เคยรู้จักกับเอเจนต์ผู้ส่งมาก่อน นั่นคือ ไม่ปรากฏชื่อเอเจนต์ผู้ส่งในตารางเอเจนต์ เอเจนต์ผู้รับจะบันทึกข้อมูลของเอเจนต์ผู้ส่งเพิ่มเข้าไปในตารางเอเจนต์ แต่ถ้ารู้จักเอเจนต์ผู้ส่งอยู่แล้ว ก็จะพิจารณาที่อยู่ของเอเจนต์นั้น หากไม่ปรากฏที่อยู่นั้นในตารางเอเจนต์ก็จะถูกเพิ่มเข้าไปเช่นกัน ด้วยการทำงานเช่นนี้ ทำให้เอเจนต์มีลักษณะการอยู่ในสังคมคล้ายมนุษย์ และสนับสนุนทำให้การสื่อสารในภาษา CL มีประสิทธิภาพมากขึ้น

สำหรับการดึงข้อมูลของเอเจนต์จะเกิดขึ้นในกรณีที่ต้องการส่งข้อความสื่อสารออกไปยังเอเจนต์อื่น โดยทำการอ้างอิงถึงเอเจนต์ผู้รับด้วยชื่อเอเจนต์นั้น ซึ่งทำให้เอเจนต์ทราบที่อยู่ของเอเจนต์ผู้รับและสามารถส่งข้อความสื่อสารไปยังเอเจนต์ผู้รับได้อย่างถูกต้อง

- *ส่วนตารางข้อความนำเข้า (Input message table)*

เมื่อเอเจนต์ได้รับข้อความสื่อสาร เอเจนต์จะทำการบันทึกข้อความสื่อสารนั้นลงในฐานความรู้ ซึ่งมีโครงสร้างเป็นตารางอันประกอบด้วย

- หมายเลขของข้อความสื่อสาร (Communication Message ID)

-หมายเลขของข้อความสื่อสารที่ข้อความสื่อสารนี้ทำการตอบสนอง (Communication Message Response ID) ในกรณีที่เป็นเลข 0 จะหมายความว่า ข้อความสื่อสารที่ได้รับเป็นการเริ่มต้นการสื่อสาร ไม่ได้ทำการตอบสนองต่อข้อความสื่อสารใดๆ

- สาร(Message) หมายถึงเนื้อหาของข้อความสื่อสารที่เอเจนต์ผู้ส่งต้องการจะสื่อ

- เวลาที่ได้รับข้อความสื่อสาร (Received time)

- สถานะของการตอบสนองข้อความสื่อสารนี้ (Response status) ใช้บอกว่าข้อความสื่อสารนี้ได้ทำการตอบสนองด้วยเอเจนต์หรือยัง

เอเจนต์สามารถนำข้อมูลในตารางนี้ไปช่วยในการตรวจสอบหาค่าความจริงต่างๆ หรือแม้แต่ช่วยประมวลผลการทำงานให้เหมาะสมสอดคล้องกับข้อความที่ได้รับ

- *ส่วนตารางข้อความส่งออก (Output message table)*

เมื่อเอเจนต์ทำการส่งข้อความสื่อสารออกไป เอเจนต์จะทำการบันทึกข้อความสื่อสารนั้นลงในฐานความรู้ ซึ่งมีโครงสร้างเป็นตารางดังนี้

- หมายเลขของข้อความสื่อสาร (Communication Message ID)

-หมายเลขของข้อความสื่อสารที่ข้อความสื่อสารนี้ทำการตอบสนอง (Communication Message Response ID) ในกรณีที่เป็นเลข 0 จะหมายความว่า ข้อความสื่อสารที่ได้รับเป็นการเริ่มต้นการสื่อสาร ไม่ได้ทำการตอบสนองต่อข้อความสื่อสารใดๆ

- สาร(Message) หมายถึงเนื้อหาของข้อความที่เอเจนต์ผู้ส่งต้องการจะสื่อ

- เวลาที่ได้ส่งข้อความสื่อสาร (Received time)

- สถานะของการตอบกลับข้อความสื่อสารนี้ (Response status) ใช้บอกว่าข้อความสื่อสารนี้ได้มีการตอบกลับการส่งข้อความไปหรือยัง

จะสังเกตว่า ส่วนประกอบของตารางข้อความส่งออกจะใกล้เคียงกับตารางของข้อความนำเข้า แตกต่างกันที่ความหมายและการเก็บสถานะในการตอบสนองต่อข้อความนั้น ซึ่งในการประมวลผลการรับ/ส่งข้อความสื่อสารในภาษา CL จะสมบูรณ์ได้ ต้องอาศัยตารางทั้งสองในการแสดงทิศทางของการสนทนาโต้ตอบของเอเจนต์ เพื่อสื่อให้ทราบว่าเอเจนต์ใดส่งข้อความอะไรไปให้ใคร และได้รับข้อความตอบกลับมาหรือไม่ อย่างไร ถ้าได้รับจะมีการส่งข้อความกลับไปหรือไม่ เป็นต้น ทำให้การประมวลผลสำหรับการสื่อสารในภาษา CL เป็นไปได้อย่างถูกต้อง

4) **ส่วน** Clock เป็นนาฬิกาของเอเจนต์ ซึ่งใช้ตรวจสอบเวลาในการรับ/ส่งข้อความสื่อสารของเอเจนต์ และใช้กำหนดการเริ่มต้นการทำงานต่างๆ ในโปรแกรม CL นาฬิกาของเอเจนต์จะอ้าง

อิงมาจากนาฬิกาของเครื่องคอมพิวเตอร์ที่รันโปรแกรม CL นั้นอยู่ โดยกำหนดให้หน่วยการนับเวลาที่เล็กที่สุด คือ หน่วยวินาที (second)

7.1.2 การทำงานของเอเจนต์สื่อสาร

จากที่กล่าวมาข้างต้น เราสามารถนิยามคลาส Agent เพื่อสร้างเอเจนต์สื่อสารในรูปแบบภาษาไพธอน ได้ดังรูปที่ 7.2

```
Class Agent :
  def __init__(self, name, input, output, comm_script, KB) :
    self.name = name
    self.input = [keyboard, (protocol1,port1), ..., (protocoln, portn)]
    self.output = [console, (protocol1,port1), ..., (protocolm, portm)]
    self.comm_script = [(CommMsg1, Action1), ..., (CommMsgl, Actionl)]
    self.KB = [self.comm_script, self.log, fn.def., built-in fn, variable]
    self.comm_msg_input = {Name1:comm_msg1, ..., Namen:comm_msgn}
    self.comm_msg_output = {Name1:comm_msg1, ..., Namem:comm_msgm}
    self.log = { time1:comm_msg_id1, ..., timen:comm_msg_idn }
          self.open_comm_channel(input)
          while 1 : self.inference(self.comm_msg_input)
  def open_comm_channel(self, input) :
          for protocoln in input :
            create_new_thread(receive_comm_msg_input(protocoln))
  def receive_comm_msg_input(self, protocoln) :
          while 1 : self.comm_msg_input[protocoln] = observe(protocoln) at timen
  def send_comm_msg_output(self, protocoln, comm_msg_output[protocoln]) :
          send(comm_msg_output[protocoln]) via protocoln
  def inference(self, comm_msg_input) :
          self.log[timen] = comm_msg_input[protocoln].id
          Actions = fire(Comm_msg , comm_script)
          meta_demo_for_list(Actions)
  def meta_demo_for_list(self, [stmt1, stmt2, ..., stmtn]) :
          for eachStmt in [stmt1, stmt2, ..., stmtn]) :
            meta_demo(eachStmt)
  def meta_demo(self, stmt) :
          if (stmt = Mode(msg,snd,recv,prot,resp_id)) :
            content = Message(type, msg, msg_id) // create message
            self.comm_msg_output[prot] = Comm_msg(Mode,snd,recv,content,prot,resp_id)
            send_comm_msg_output(prot, self. comm_msg_output[prot])
          elif (stmt = meta_fn(meta_msg)) : // meta_fn = [assert, solve, demo, delete]
            obj_msg = nametoobj(meta_msg)
            result = execute obj_msg by object function
            return(objtoname(result)) // return meta-message
Class Comm_msg() :
          def __init__(self,id,mode,sender,receiver,message,protocol,response_id) :
Class Message() :
          def __init__(self, type, content, content_id)
```

รูปที่ 7.2 คลาสเอเจนต์

การทำงานสื่อสารของเอเจนต์นี้สามารถอธิบายเป็นขั้นตอนการทำงานที่มีวัฏจักร Observe–Fire Comm. script–Act ได้ ดังรายละเอียดต่อไปนี้

1. **กระบวนการ** Observe

เอเจนต์จะต้องมีการตรวจสอบการเข้ามาของข้อความสื่อสารที่ส่วนอินพุท(Input) อยู่ตลอดเวลา เราเรียกการทำงานนี้ว่าเป็นกระบวนการ Observe ของเอเจนต์ เมื่อเอเจนต์ได้รับข้อความสื่อสารจากภายนอก ส่วนวินิจฉัยเอเจนต์จะทำการประมวลผลข้อความสื่อสารนั้น และบันทึกเวลาที่ได้รับรวมถึงรายละเอียดต่างๆ ของข้อความลงในตารางข้อความนำเข้าของฐานความรู้

2. **กระบวนการ** Fire Comm. Script

เมื่อบันทึกการได้รับข้อความสื่อสารแล้ว ส่วนวินิจฉัยของเอเจนต์จะต้องทำการตอบสนองต่อเหตุการณ์หรือข้อความสื่อสารนั้น โดยพิจารณาข้อความนั้นว่าสอดคล้องหรือจับคู่ได้กับสคริปต์ exist(Comm_msg) → Action ประโยคใดในตารางสคริปต์ตอบสนองเหตุการณ์ซึ่งเป็นส่วนหนึ่งของฐานความรู้ของเอเจนต์ เมื่อจับคู่ได้แล้วจึงส่ง Action เพื่อนำไปประมวลผลตอบสนองต่อไป กระบวนการจับคู่ข้อความสื่อสารที่ได้รับกับสคริปต์ตอบสนองหรือการสื่อสารนี้เรียกว่า กระบวนการ Fire Comm. Script

แต่เนื่องจากในภาษา CL ได้สนับสนุนให้มีการตอบสนองต่อข้อความสื่อสารที่มีการทำงานเป็นชุดคล้ายบทสนทนาได้ ดังนั้น เอเจนต์จึงต้องมีกระบวนการจับคู่ข้อความสื่อสารที่ซับซ้อนมากขึ้นตามประเภทของบทสนทนาอันได้แก่ แบบ Independent Repeatable and Partial, Independent Repeatable and Total, Sequence, Unordered และ One ซึ่งได้อธิบายรายละเอียดการทำงานไปแล้วในบทที่ 6 จึงไม่ขออธิบายไว้ในหัวข้อนี้

3. **กระบวนการ** Act

เมื่อจับคู่ข้อความสื่อสารกับสคริปต์ตอบสนองได้แล้ว เอเจนต์จะประมวลผลคำสั่ง Action เพื่อเป็นการตอบสนองต่อเหตุการณ์ที่เกิดขึ้น ด้วยส่วน Meta_demo() ซึ่งเป็นอินเตอร์พรีตเตอร์ในระดับเอเจนต์

เอเจนต์สามารถตอบสนองข้อความสื่อสารได้ด้วยการกระทำฟังก์ชันพื้นฐานระดับเมต้า ได้แก่ assert, solve, demo, delete ซึ่งทำหน้าที่รับรู้ แก้ปัญหา ประมวลผล และลบสาร ตามลำดับ โดยสารที่ใช้รับส่งโต้ตอบกันจะเป็นระดับเมต้าทั้งหมด ดังนั้น ฟังก์ชันเหล่านี้อาจจะต้องทำการแปลงสารจากระดับเมต้าให้เป็นสัญลักษณ์ระดับวัตถุ ก่อนนำไปประมวลผล แล้วจะต้องแปลงผลลัพธ์ที่ได้กลับไปเป็นสารระดับเมต้าก่อนแล้วจึงส่งตอบกลับไปได้โดยผ่านข้อความสื่อสารโหมดต่างๆ

กระบวนการที่ทำงานเป็นวัฏจักรดังกล่าว คล้ายกับกระบวนการคิดของเอเจนต์ที่ใช้ "Cycle predicate" เสนอโดย Kowalski [18] แต่เอเจนต์ในงานวิจัยดังกล่าว ยังไม่สามารถใช้งาน

โปรโตคอลสื่อสารได้ และยังไม่มีการศึกษาถึงวิธีรับส่งสารระหว่างเอเจนต์ นอกจากนี้ การสื่อสารในโหมด Tell ที่เป็นการรับรู้สารของเอเจนต์นั้น มีลักษณะคล้ายกับ Knowledge Assimilation [18, 19] ซึ่ง Kowalski ได้อธิบายกระบวนการรับรู้สารของเอเจนต์ไว้ด้วยวิธีทางตรรกศาสตร์ ดังนั้น ฟังก์ชัน assert ของ CL สามารถแทนด้วยฟังก์ชัน assimilate ในอนาคต

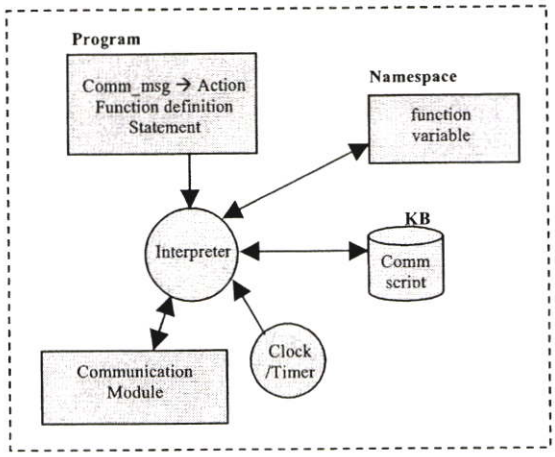
โดยหลักการของเอเจนต์สื่อสารใน CL เราสามารถมองได้ว่า ขบวนการเขียนและรันโปรแกรมผ่านอินเตอร์พรีตเตอร์ (ดังเช่นกรณีของ Python) เป็นการสนทนาระหว่างโปรแกรมเมอร์และอินเตอร์พรีตเตอร์ ดังนั้น เอเจนต์ของ CL ก็สามารถโปรแกรมให้เพิ่มความรู้ใหม่โดยโปรแกรมเมอร์ได้ โดยกำหนดอินพุทของเอเจนต์เป็นคีย์บอร์ด และเอาท์พุทเป็นคอนโซลหรือจอภาพ ซึ่งเราสามารถกำหนดให้เป็นพื้นฐานสำหรับทุกๆ เอเจนต์ของ CL ได้ ดังนั้น Comm. Script สำหรับเอเจนต์ก็จะเป็น

```
exist( tell_msg(?declare,"User","CL",?id1,_))→
   {x = assert(declare); send(Yes(x,"CL","User",idR1,id1), console)}
exist( request_msg(?stmt,"User","CL",?id2,_)) →
   {x = demo(stmt); send(Yes(x, "CL","User",idR2,id2), console)}
```

ซึ่งระบุว่า ถ้าเอเจนต์ได้รับ tell_msg(?declare,"User","CL",?id1,_) จากการ Observe ที่คีย์บอร์ดแล้วให้ Declare ฟังก์ชันนั้น (ใช้ได้กับการนิยามตัวแปรและคลาสได้ด้วย) แล้วตอบกลับว่า Yes แต่ถ้าได้รับ request_msg(?stmt,"User","CL",?id2,_) ให้ประมวลผล statement นั้น แล้วนำผลตอบไปที่คอนโซล ตัวอย่างเช่น โปรแกรมเมอร์พิมพ์ข้อความ Tell("def add(x,y) : return(x+y)", "User","CL",ids1,_) คุยกับเอเจนต์ แล้วจะได้รับคำตอบ Yes ที่หน้าจอ จากนั้นพิมพ์ข้อความ Request("add(3,5)","User", "CL",ids2,_) แล้วเอเจนต์จะตอบ 8 กลับมา โดยวิธีการนี้ทำให้เราสามารถเขียนโปรแกรมและสั่งงานเอเจนต์ของ CL ได้เหมือนกับการเขียนและรันโปรแกรมคอมพิวเตอร์โดยทั่วไป ทำให้ CL สามารถใช้แทนภาษาคอมพิวเตอร์ทั่วไปได้

7.2 ระบบการประมวลผลโปรแกรมภาษา CL ที่ระดับล่าง

จากรูปที่ 7.3 แสดงถึงระบบการประมวลผลโปรแกรมภาษา CL ที่มีการติดต่อกับสภาพแวดล้อมภายนอก (External environment) อันได้แก่ เอเจนต์(Agent) ระบบการประมวลผลโปรแกรมภาษา CL ประกอบด้วยส่วนต่างๆ ดังนี้



รูปที่ 7.3 ระบบการประมวลผลโปรแกรมภาษา CL

7.2.1 **โปรแกรม** (Program)

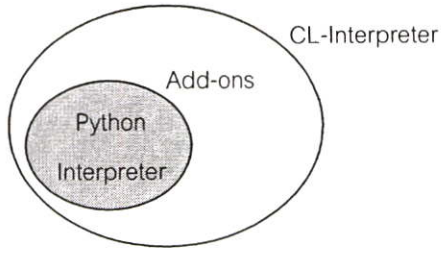
เป็นส่วนโปรแกรมภาษา CL ที่โปรแกรมตามไวยากรณ์ภาษาที่ได้อธิบายไว้ในบทที่ 5 โดยโปรแกรมภาษา CL จะประกอบด้วย 3 ส่วน ได้แก่ การนิยามฟังก์ชัน (Function definition) สคริปต์ตอบสนองต่อเหตุการณ์ (Event-action script) หรือสคริปต์สื่อสาร (Communication script) และ ประโยค (statement)

7.2.2 **อินเตอร์พรีตเตอร์** (Interpreter)

อินเตอร์พรีตเตอร์เป็นส่วนที่ทำหน้าที่แปลความหมายของโปรแกรมภาษา CL หมายถึงเมื่ออินเตอร์พรีตเตอร์ได้รับซอร์สโค้ด (Source code) หรือโปรแกรมภาษา CL (Program) ซึ่งได้อธิบายไปในหัวข้อที่แล้ว อินเตอร์พรีตเตอร์จะประมวลผล (execute) โปรแกรมทันที

ในงานวิจัยนี้ อินเตอร์พรีตเตอร์ที่พัฒนาขึ้นนั้นเป็นอินเตอร์พรีตเตอร์แบบเมต้าอินเตอร์พรีตเตอร์ (Meta-Interpreter) ซึ่งมีหลักการต่างจากอินเตอร์พรีตเตอร์ทั่วไปที่จะทำการแปลภาษาคอมพิวเตอร์ให้อยู่ในรูปของภาษาเครื่อง (Machine language) โดยอินเตอร์พรีตเตอร์ภาษา CL นี้จะทำการแปลความหมายภาษา CL ให้อยู่ในรูปของภาษาไพธอน แล้วใช้อินเตอร์พรีตเตอร์ภาษาไพธอนในการประมวลผลคำสั่งอีกที แสดงได้ดังรูปที่ 7.4

การแปลความหมายภาษา CL ให้อยู่ในรูปแบบภาษาไพธอนนั้น ต้องอาศัยขั้นตอนการทำ lexical analysis และ parsing ของ PLY (Python Lex-Yacc) ซึ่งเป็นโมดูลภาษาไพธอนที่พัฒนาเพื่อการสร้างอินเตอร์พรีตเตอร์โดยเฉพาะ โดยรายละเอียดของการพัฒนาอินเตอร์พรีตเตอร์ด้วย PLY ได้อธิบายไปแล้วในบทที่ 2



รูปที่ 7.4 CL-Interpreter

การทำงานของอินเตอร์พรีตเตอร์ภาษา CL จะมีรายละเอียดขั้นตอนการประมวลผลแตกต่างกันไปตามส่วนประกอบที่กำหนดไว้ในโปรแกรม อันได้แก่ ส่วนของการนิยามฟังก์ชัน, สคริปต์ สื่อสาร และ ประโยค ตามลำดับ และด้วยเหตุนี้ การประมวลผลคำสั่งในแต่ละครั้งก็จะมีการติดต่อกับส่วนอื่นๆ ของระบบอย่าง เนมสเปส (Namespace), ฐานความรู้ (Knowledge Base) และ โมดูลสำหรับการสื่อสาร (Communication Module) แตกต่างกันไปด้วย

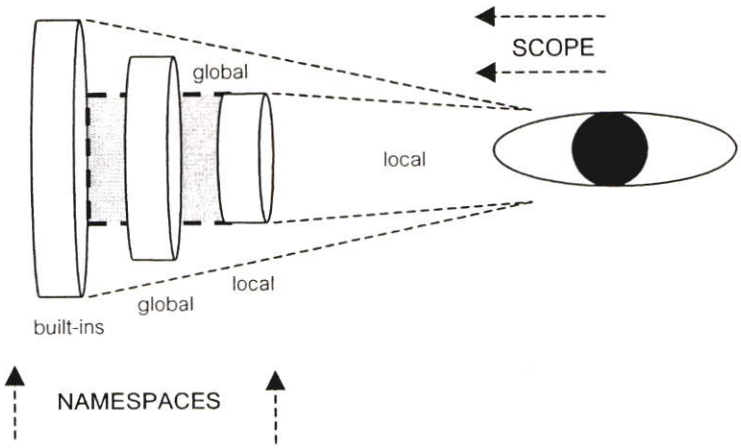
7.2.3 **เนมสเปส** (Namespace)

หลักการพื้นฐานของเนมสเปส (Namespace) คือ เซ็ตของการแมป (mapping) จากชื่อ หรือ identifier ให้เป็นวัตถุ โดยกระบวนการที่ทำการเพิ่มชื่อเข้าไปในเนมสเปส เรียกว่า การ binding ชื่อเข้าไปยังวัตถุ ในทำนองเดียวกัน กระบวนการที่ทำการเปลี่ยนแปลงการแมปชื่อในเนมสเปส เรียกว่า การ rebinding และเรียกกระบวนการที่ทำการลบชื่อออกจากเนมสเปสว่า การ unbinding

เนมสเปสในภาษา CL นี้ได้เลียนแบบการทำงานของเนมสเปสในภาษาไพธอน (Python) มาทั้งหมด โดยได้กำหนดให้มีเนมสเปสทั้งหมด 3 เนมสเปส ได้แก่ โลคอลเนมสเปส (Local namespaces), โกลบอลเนมสเปส (Global namespaces) และ บิวท์อินเนมสเปส (Built-ins namespaces) และการเข้าถึงชื่อ (identifier) ในเนมสเปสนั้นจะขึ้นอยู่กับลำดับของการโหลดเนมสเปสนั้น หรือลำดับที่เกิดเนมสเปสนั้นในระบบ

อินเตอร์พรีตเตอร์ภาษา CL จะทำการโหลดบิวท์อินเนมสเปสขึ้นมาเป็นลำดับแรก แล้วจึงโหลดโกลบอลเนมสเปสสำหรับการประมวลผลโมดูล โกลบอลเนมสเปสนี้จะทำงาน (active) เมื่อมีการประมวลผลโมดูลเกิดขึ้น และในลำดับสุดท้าย คือ การสร้างโลคอลเนมสเปส ซึ่งจะเกิดขึ้นเมื่อมีการเรียกฟังก์ชันระหว่างทำการประมวลผล

จากหลักการของเนมสเปสข้างต้น เราสามารถสรุปความสัมพันธ์ระหว่างเนมสเปสและขอบเขตของตัวแปร (Variable scope) ได้ดังนี้



รูปที่ 7.5 การกำหนดเนมสเปสและขอบเขตของตัวแปร

เมื่อมีการเข้าถึง (access) แอททริบิวต์ต่างๆ อินเตอร์พรีตเตอร์จะต้องหาแอททริบิวต์ หรือ identifier นั้น ในเนมสเปสทั้งสาม โดยจะเริ่มหาจาก โลคอลเนมสเปส หากไม่พบแอททริบิวต์นั้นก็จะหาในโกลบอลเนมสเปส และถ้ายังไม่พบก็จะหาในบิวท์อินเนมสเปสเป็นลำดับสุดท้าย ถ้าไม่พบในทั้งสามเนมสเปสถือว่าเป็นการทำงานที่ผิดพลาด และจะแจ้งข้อผิดพลาดกลับมายังผู้เขียน

7.2.4 **ฐานความรู้** (Knowledge Base)

เป็นส่วนที่ใช้เก็บความรู้ที่ได้จากการประมวลผลโปรแกรม CL ในส่วนของสคริปต์สื่อสาร นั่นคือ ระบบการประมวลผลจะเก็บลิสต์ของคำสั่งสื่อสาร(มีรูปแบบ [P_s, P_e] : exist(Comm. message) → Action) ไว้ เพื่อให้สามารถตรวจสอบข้อความสื่อสารและสามารถตอบสนองได้อย่างถูกต้อง โดยฐานความรู้ที่กล่าวถึงนี้ ก็คือฐานความรู้ของเอเจนต์ที่อธิบายไปในหัวข้อที่ 7.1 นั่นเอง

7.2.5 **นาฬิกา** (Clock/Timer)

เป็นนาฬิกาของระบบ เพื่อใช้ในการอ้างอิงสำหรับการทำงานให้เป็นไปตามเวลาที่ถูกต้องและเหมาะสม ซึ่งในระบบประมวลผลนี้ได้ใช้นาฬิกาของเครื่องคอมพิวเตอร์ในการการกำหนดนาฬิกาของระบบ และมีหน่วยวินาทีเป็นหน่วยที่เล็กที่สุดของนาฬิกา

7.2.6 **โมดูลการสื่อสาร** (Communication module)

เป็นโมดูลการทำงานที่ช่วยให้ระบบสามารถติดต่อสื่อสารกับสภาพแวดล้อมภายนอกได้ โดยได้พัฒนาขึ้นจากโมดูลสำหรับการสื่อสารของภาษาไพธอน ซึ่งมีความสามารถในการจัดการซ็อกเก็ต (socket) แต่ถึงอย่างไรก็ตาม ก็ยังได้พัฒนาการทำงานที่สามารถแปลงข้อมูลระดับวัตถุให้

เป็นระดับเมต้า และแปลงข้อมูลระดับเมต้าให้เป็นระดับวัตถุเพิ่มเติมเข้ามา อีกทั้งยังมีการสร้างข้อความสื่อสารเพื่อใช้ในการติดต่อกับสภาพแวดล้อมภายนอกอีกด้วย

7.3 อินเตอร์พรีตเตอร์ภาษา CL (CL Interpreter)

ในหัวข้อนี้ จะกล่าวถึงอินเตอร์พรีตเตอร์ของภาษา CL ซึ่งเป็นส่วนประกอบที่สำคัญของระบบประมวลผลโปรแกรม CL อย่างละเอียด เริ่มต้นด้วยหลักการทำงานของอินเตอร์พรีตเตอร์, การประมวลผลการนิยามฟังก์ชัน, การประมวลผลสคริปต์สื่อสาร และการประมวลผลประโยคตามลำดับ

7.3.1 หลักการทำงานของอินเตอร์พรีตเตอร์

เมื่ออินเตอร์พรีตเตอร์ได้รับโปรแกรมภาษา CL ก็จะทำการแปลความหมายและประมวลผลไปทีละส่วนประกอบ (Element) ซึ่งอาจเป็นการนิยามฟังก์ชัน, สคริปต์สื่อสาร หรือประโยคใดๆ

หากอินเตอร์พรีตเตอร์ได้รับส่วนประกอบประเภทการนิยามฟังก์ชัน อินเตอร์พรีตเตอร์จะจัดการโดยเก็บนิยามของฟังก์ชันไว้ในเนมสเปส แต่ถ้าอินเตอร์พรีตเตอร์ได้รับส่วนประกอบประเภทสคริปต์สื่อสาร อินเตอร์พรีตเตอร์จะเก็บสคริปต์สื่อสารนั้นไว้ในฐานความรู้ ในกรณีสุดท้ายที่ส่วนประกอบเป็นประโยคใดๆ อินเตอร์พรีตเตอร์จะประมวลผลคำสั่งตามเวลาที่กำหนดไว้ในโปรแกรมการทำงานดังกล่าว แสดงเป็นโค้ดได้ดังนี้

```
def cl_interpreter([element1, ..., elementn]):
    for elementi in [element1, ..., elementn] :
        if (elementType(elementi) == "function_definition") :
            define_function(elementi)
        elif (elementType(elementi) == "comm script") :
            assert_comm_script(elementi)
        else :    process_stack = []; process_id = 0
                  execute_interval_statement(elementi)
def main() :
    program = read(ProgramFile)
    ready_program = add_unspec_interval(program)
    error = checkSyntax&Parser(program)
    if error != 0 : print(error)
    else : cl_interpreter(program)
```

รูปที่ 7.6 การทำงานหลักของอินเตอร์พรีตเตอร์

7.3.2 การประมวลผลการนิยามฟังก์ชัน

เมื่ออินเตอร์พรีตเตอร์ได้รับส่วนประกอบในรูปแบบ "def f^nname([arg_1, ..., arg_m]) : [$action_1$, ..., $action_n$]" อินเตอร์พรีตเตอร์จะพิจารณาว่าฟังก์ชันที่นิยามขึ้นนี้อยู่ภายในการทำงานของฟังก์ชันอื่นหรือไม่ ถ้าอยู่ อินเตอร์พรีตเตอร์จะเก็บนิยามของฟังก์ชันซึ่งเป็นคลาสประกอบด้วยชื่อฟังก์ชัน(f^nname), ลิสต์ของอาร์กิวเมนต์([arg_1, ..., arg_m]) และ ลิสต์ของประโยคที่นิยามใน

ฟังก์ชัน([action$_1$, ..., action$_n$]) ไว้ในโลคอลเนมสเปส(Local namespace) แต่ในทางกลับกัน ก็จะเก็บนิยามฟังก์ชันไว้ในโกลบอลเนมสเปส(Global namespace) การทำงานดังกล่าวแสดงเป็นโค้ดได้ดังนี้

```
def define_function (def f"name([arg1, ..., argm]) : [action1, ..., actionn]) :
    if this function is defined in other function :
        //save f" in local namespace
        local_namespace[f"name] = defineFunction(f"name, [arg1, ..., argm], [action1, ..., actionn])
    else :
        //save f" in global namespace
        global_namespace[f"name] = defineFunction(f"name, [arg1, ..., argm], [action1, ..., actionn])
```

รูปที่ 7.7 การประมวลผลการนิยามฟังก์ชัน

7.3.3 การประมวลผลสคริปต์สื่อสาร

เมื่ออินเตอร์พรีตเตอร์ได้รับส่วนประกอบที่เป็นสคริปต์สื่อสารซึ่งมีรูปแบบ "commscript : [[P_{s1},P_{e1}] : exist(Comm_msg$_1$) → Action$_1$, ..., [P_{sn},P_{en}] : exist(Comm_msg$_n$) → Action$_n$]" แล้ว อินเตอร์พรีตเตอร์จะทำการเก็บสคริปต์สื่อสารลงในฐานความรู้ ในลักษณะของตารางที่เรียกว่า ตารางสำหรับสคริปต์สื่อสาร (Commscript Table) หรือตารางการตอบสนองต่อเหตุการณ์ (Event-action Table) ซึ่งจะบันทึกรายละเอียดของแต่ละ [P_{si},P_{ei}] : exist(Comm_msg$_i$) → Action$_i$ ในสคริปต์สื่อสาร อันประกอบด้วย ข้อมูลของช่วงเวลาในการตรวจสอบคำสั่ง([P_{si},P_{ei}]), ข้อความสื่อสาร(Comm_msg$_i$) และ การตอบสนองต่อข้อความสื่อสาร(Action$_i$)

```
def assert_comm_script(commscript : [CommAction1, ..., CommActionn ]) :
    for CommActioni in [CommAction1, ..., CommActionn ] :
        if CommAction.type == "independent_repeatable_partial" :
            [SubCommAction1, ..., SubCommActionn] = CommAction
            [Psi,Pei] : exist(Comm_msgi) → Actioni = SubActioni
            CommScriptTable.append(Indp_Rept_Part([SubCommAction1, ..., SubCommActionn]))
        elif CommAction.type == "independent_repeatable_total" :
            comm_independent_total : [SubCommAction1, ..., SubCommActionn] = CommAction
            [Psi,Pei] : exist(Comm_msgi) → Actioni = SubActioni
            CommScriptTable.append(Indp_Total([SubCommAction1, ..., SubCommActionn]))
        elif CommAction.type == "sequence" :
            comm_sequence : [SubCommAction1, ..., SubCommActionn] = CommAction
            [Psi,Pei] : exist(Comm_msgi) → Actioni = SubActioni
            CommScriptTable.append(CommSequence([SubCommAction1, ..., SubCommActionn]))
        elif CommAction.type == "unordered" :
            comm_unordered : [SubCommAction1, ..., SubCommActionn] = CommAction
            [Psi,Pei] : exist(Comm_msgi) → Actioni = SubActioni
            CommScriptTable.append(CommUnordered([SubCommAction1, ..., SubCommActionn]))
        elif CommAction.type == "one" :
            comm_one : [SubCommAction1, ..., SubCommActionn] = CommAction
            [Psi,Pei] : exist(Comm_msgi) → Actioni = SubActioni
            CommScriptTable.append(One([SubCommAction1, ..., SubCommActionn]))
        else :
            [Psi,Pei] : exist(Comm_msgi) → Actioni = CommActioni
            CommActionRecordi = [[Psi,Pei], Comm_msgi, Actioni]
            CommScriptTable.append(CommActionRecordi)
    return CommScriptTable
```

รูปที่ 7.8 การประมวลผลสคริปต์สื่อสาร

ในกรณีที่มีการกำหนดสคริปต์สื่อสารเป็นบทสนทนา ซึ่งแบ่งได้เป็น 5 แบบ คือ Independent Repeatable and Partial, Independent Repeatable and Total, Sequence, Unordered และ One อินเตอร์พรีตเตอร์จะเก็บการกำหนดบทสนทนาด้วยคลาสเฉพาะแต่ละแบบ ซึ่งคลาสนี้จะเก็บข้อมูลลิสต์ของประโยคทั้งหมดในบทสนทนา การทำงานดังกล่าวสามารถแสดงได้เป็นโค้ดดังรูปที่ 7.8

7.3.4 การประมวลผลประโยคใดๆ

เมื่ออินเตอร์พรีตเตอร์ได้รับส่วนประกอบในรูปแบบ "[P_s,P_e] : statement" อินเตอร์พรีตเตอร์จะประมวลผลคำสั่งตามเวลาที่กำหนด หากมีการระบุเวลาสิ้นสุด P_e อินเตอร์พรีตเตอร์จะแบ่งการทำงานเป็น 2 ส่วน คือ ส่วนประมวลผลคำสั่ง และส่วนตรวจสอบผลการทำคำสั่งเมื่อถึงเวลา P_e โดยอินเตอร์พรีตเตอร์จะแตกเธรดสำหรับการทำงานในส่วนแรก และเมื่อถึงเวลา P_e อินเตอร์พรีตเตอร์จะดูว่าเธรดที่แตกไปนั้นประมวลผลเสร็จหรือยัง ถ้ายังก็จะทำลายเธรดนั้นแล้วแจ้งความผิดพลาดให้กับระบบ แต่ถ้าเธรดทำงานเสร็จเรียบร้อย อินเตอร์พรีตเตอร์ก็จะรับ statement ถัดไปมาประมวลผล

```
def execute_interval_statement([Ps,Pe]:statement) :
    checkStatus = process_status
    if (Pe != "_") :
        Pe' = expr_eval(Pe)
        if (Ps != "_") :
            Ps' = expr_eval(Ps)
            wait(Ps' - getCurrentTime())
        thread = createThread(execute_statement(statement))
        wait(Pe' - getCurrentTime())
        if (process_stack != []) :
            clear(process_stack)
            exit()
    else :  if (Ps != "_") :
            Ps' = expr_eval(Ps)
            wait(Ps' - getCurrentTime())
        execute_statement(statement)
```

รูปที่ 7.9 การจัดการช่วงเวลาที่กำหนดการทำงาน

หากไม่ได้ระบุเวลา P_e อินเตอร์พรีตเตอร์จะไม่มีการแตกเธรดแยกการประมวลผล ซึ่งในทั้งสองกรณี จะเริ่มทำการประมวลผลคำสั่งเมื่อถึงเวลาเริ่มต้น P_s ซึ่งอินเตอร์พรีตเตอร์จะรอเป็นเวลา P_s – current_time เมื่อระบุเวลาเริ่มต้น P_s และหากไม่มีการระบุ P_s อินเตอร์พรีตเตอร์ก็จะประมวลผลคำสั่งนั้นทันที จากที่กล่าวไปเป็นการจัดการช่วงเวลาเพื่อกำหนดการทำงาน เขียนเป็นโค้ดได้ดังรูปที่ 7.9

ต่อไปจะกล่าวถึงรายละเอียดการประมวลผล statement (execute_statement) เนื่องจากประโยคแบ่งได้เป็น ประโยคคำสั่ง, ประโยคชุดคำสั่ง และประโยคควบคุมลำดับการทำงาน อินเตอร์พรีตเตอร์จึงมีการประมวลผล statement ต่างๆ ดังนี้

```
def execute_statement(statement) :
    process_id += 1
    id = process_id
    process_stack.push(id)
    if statementType(statement) == "single" :
        exec(statement)
    if statementType(statement) == "sequence" :
        [[Ps1,Pe1] : stmt1, ..., [Pn,Pn] : stmtn] = statement
        for [Psi,Pei] : stmti in [[Ps1,Pe1] : stmt1, ..., [Pn,Pn] : stmtn]:
            execute_interval_statement([Psi,Pei] : stmti)
    if statementType(statement) == "unordered" :
        {[_,_] : stmt1, ..., [_,_] : stmtn} = statement
        for [_,_] : stmti ∈ {[_,_] : stmt1, ..., [_,_] : stmtn} :
            execute_interval_statement([_,_] : stmti)
    if statementType(statement) == "alternative" :
        [[Ps1,Pe1] : stmt1, ..., [Pn,Pn] : stmtn] = statement
        for [_,_] : stmti in [[Ps1,Pe1] : stmt1, ..., [Pn,Pn] : stmtn] = statement :
            if (execute_interval_statement([_,_] : stmti)) : break
    if statementType(statement) == "parallel" :
        [[Ps1,Pe1] : stmt1, ..., [Pn,Pn] : stmtn] = statement
        for each [Psi,Pei] : stmti in [[Ps1,Pe1] : stmt1, ..., [Pn,Pn] : stmtn]:
            createThread(Execute_interval_statement([Psi,Pei] : stmti))
        while process_stack != id : pass
    if statementType(statement) == "if-statement" :
        "IF" (cond) : then-statement = statement
        if expr_eval(cond) : execute_interval_statement(then-statement)
    if statementType(statement) == "while-statement" :
        "WHILE" (cond) : while-statement = statement
        while expr_eval(cond) : execute_interval_statement(while-statement)
    if statementType(statement) == "send built-in function" :
        "SEND" (comm_msg, protocol)
        mode(msg, snd, recv, resp_id) = comm_msg
        meta_msg = obj2name(msg)
        content = Message(type, meta_msg, msg_id) //create message
        id = make_new_comm_msg_id()
        send(Comm_msg(mode, snd, recv, content, id, resp_id), protocol)
    if statementType(statement) == "meta function" :
        meta f" name (meta_msg) //meta f" name = [assert, solve, demo, delete]
        obj_msg = name2obj(meta_msg)
        execute_interval_statement(obj f" name(obj_msg)) // obj f" name = [obj_assert, ...]
    process_stack.pop(id)
Class Comm_msg() :
    def __init__(self, mode, sender, receiver, message , id, response_id)
Class Message() :
    def   init   (self, type, content, content id)
```

รูปที่ 7.10 การประมวลผลประโยคใดๆ

ในกรณีที่ statement เป็นประโยคคำสั่ง(single) จะประมวลผลคำสั่งนั้น แต่ถ้า statement เป็นประโยคชุดคำสั่งแบบ sequence([[P_{s1},P_{e1}] : $stmt_1$, ..., [P_{sn},P_{en}] : $stmt_n$]) อินเตอร์พรีตเตอร์จะประมวลผลทีละ [P_{si},P_{ei}] : $stmt_i$ จนหมดชุดคำสั่ง ในกรณีที่ statement เป็นประโยคชุดคำสั่งแบบ unordered ([[_,_] : $stmt_1$, ..., [_,_] : $stmt_n$]) อินเตอร์พรีตเตอร์จะเลือกประมวลผลทีละ [_ , _] : $stmt_i$ จนหมดชุดคำสั่ง ในกรณีที่ statement เป็นประโยคชุดคำสั่งแบบ alternative ([[_,_] : $stmt_1$, ..., [_,_] : $stmt_n$]) อินเตอร์พรีตเตอร์จะเลือกประมวลผลทีละ [_ , _] : $stmt_i$ ถ้าทำสำเร็จจะยกเลิกการประมวลผลประโยคที่เหลือ แต่ถ้าล้มเหลวก็จะประมวลผลประโยคถัดไปทำเช่นนี้ไปเรื่อยๆ จนกว่าจะหมดชุดคำสั่ง และหาก statement เป็นชุดคำสั่งแบบ parallel ([[P_{s1},P_{e1}] : $stmt_1$, ..., [P_{sn},P_{en}] : $stmt_n$]) อินเตอร์พรีตเตอร์จะแตกเธรดแยกการทำงานให้แต่ละ [P_{si},P_{ei}] : $stmt_i$ ในชุดคำสั่ง

สำหรับการประมวลผล statement ที่เป็นประโยคควบคุมลำดับการทำงานไม่ว่าจะเป็นประโยค if หรือ while จะเป็นลักษณะเดียวกัน คือ อินเตอร์พรีตเตอร์จะคำนวณค่า condition ที่

กำหนดในโปรแกรม และให้ทำการประมวลผล then_statement หรือ while_statement เมื่อ condition เป็นจริง

ในกรณีที่ statement เป็นคำสั่งที่เกี่ยวข้องกับการสื่อสาร อินเตอร์พรีตเตอร์จะต้องทำการแปลงสารในระดับวัตถุให้เป็นสารในระดับเมต้าก่อนที่จะทำการส่งข้อความสื่อสารโดยเรียกบิวท์อินฟังก์ชัน send ซึ่งมีรูปแบบ send(comm_msg, protocol) และในทำนองเดียวกัน หากมีการเรียกฟังก์ชันในระดับเมต้า อันได้แก่ ฟังก์ชัน assert, solve, demo, delete จะต้องทำการแปลงสารที่ใช้จากสารระดับเมต้าให้เป็นระดับวัตถุก่อน แล้วจึงเรียกบิวท์อินฟังก์ชันนั้นในระดับวัตถุ (obj_assert, obj_solve, obj_demo, obj_delete) เพื่อประมวลผลอีกที

ทุกครั้งก่อนเริ่มประมวลผล statement หนึ่งๆ ของฟังก์ชัน execute_statement จะต้อง push ค่า process_id สำหรับ statement นั้น (ค่า process_id กำหนดเป็นเลขจำนวนเต็ม) ใน process_stack และต้อง pop ค่า process_id เดิมออกเมื่อประมวลผลเสร็จ ดังนั้น หาก process_stack ว่าง นั่นจะหมายถึง อินเตอร์พรีตเตอร์ได้ประมวลผล statement นั้นเสร็จในเวลา P_e แสดงโค้ดได้ดังรูปที่ 7.10

บทที่ 8
ผลการทดลองการใช้งานภาษา CL

จากที่กล่าวมาทั้งหมด จะเห็นว่าภาษา CL มีโครงสร้างภาษาที่สนับสนุนการทำงานคำสั่งได้เช่นเดียวกับภาษาคอมพิวเตอร์ทั่วไป อีกทั้งยังสามารถประมวลผลและจัดการการทำงานที่เกี่ยวข้องกับเวลาได้ และลักษณะที่สำคัญอีกอย่างหนึ่งคือ ความสามารถในการสนับสนุนการติดต่อสื่อสารระหว่างเอเจนต์ เพื่อแสดงให้ผู้อ่านเห็นภาพรวมการทำงานตลอดจนประโยชน์ที่จะได้รับจากการพัฒนาภาษา CL จึงได้เสนอตัวอย่างการสร้างเอเจนต์สื่อสารในหัวข้อที่ 8.1 หลังจากนั้น จึงกล่าวถึง การนำแนวคิดภาษา CL ไปใช้ในงานวิจัยอื่นๆ ในหัวข้อที่ 8.2

แต่เพื่อไม่ให้เกิดความสับสนในการอ่านโปรแกรมภาษา CL จะขอเน้นถึงโครงสร้างของโปรแกรมภาษา CL ว่าเป็นลิสต์ของส่วนประกอบหลัก 3 ส่วน ได้แก่ การนิยามฟังก์ชัน (Function definition), สคริปต์ตอบสนองต่อเหตุการณ์ (Event response script) ซึ่งประกอบด้วยประโยคตอบสนองเหตุการณ์ที่มีรูปแบบ [P_s, P_e] : exist(Comm. Msg.) → Action และ ประโยค (Statement) ซึ่งกำหนดเวลาการทำงานได้ด้วย [P_{s1}, P_{e1}] : Statement

8.1 ตัวอย่างการสร้างเอเจนต์สำหรับสื่อสารด้วย CL

8.1.1 ตัวอย่างการทำงานระหว่างบราวเซอร์และเซิร์ฟเวอร์

เพื่อแสดงการใช้งานภาษา CL เราจะสร้างเอเจนต์สื่อสาร john และ mary โดยให้เป็น Web Browser และ Web Server อย่างง่ายตามลำดับ บทสนทนาการสื่อสารของ john กำหนดให้เป็น cscript1 ของ mary เป็น cscript2 ดังนี้

John

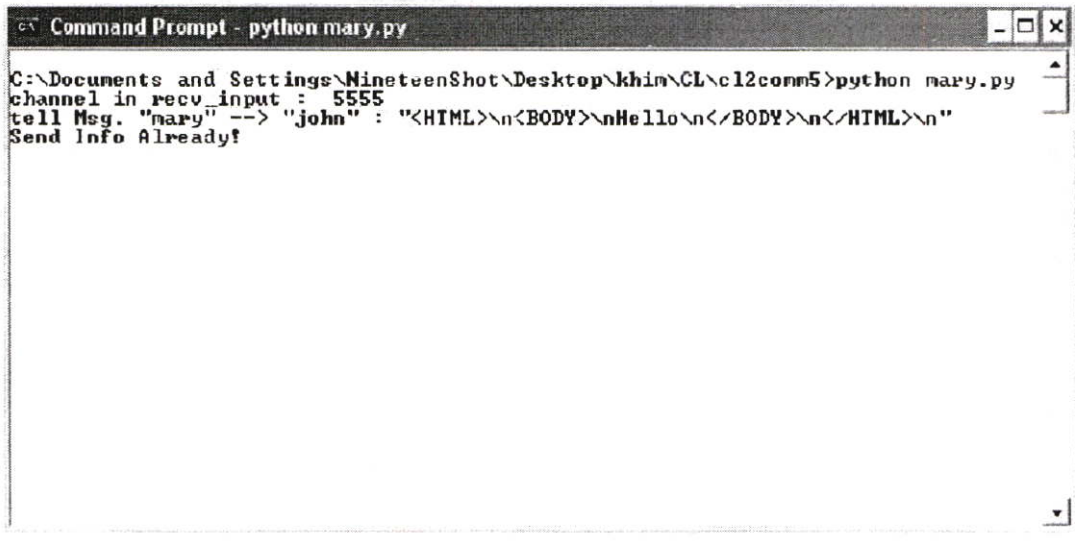
```
// cscript1 : Comm. Script for Agent John
commscript :
    exist(tell_msg(?msg, ?snd, _, ?id, _)) →
            {res = display_html_doc(msg);
             send(Yes(res, agent("myself"), agent(snd), id), 5555);}
send(please_tell("" "hello.html" "", agent("myself"), agent("mary"), 0), 5555)
```

Mary

```
// cscript2 : Comm. Script for Agent Mary
commscript :
   <_, 09:00|today> : exist(please_tell_msg(?msg, ?snd, _, ?x, _)) →
                send(Ignore(msg, agent("myself"), agent(snd), x), 6666)
   <10:00|today, _> : exist(please_tell_msg(?msg, ?snd, _, ?x, _)) →
                { result = find(msg);
                  send(Tell(result, agent("myself"), agent(snd), x), 6666); }
```

เราสามารถสร้างเอเจนต์ John และ Mary โดยคำสั่ง CL john = Agent("john", [6666], [5555,6666,7777], cscript1, john_kb) และ mary = Agent("mary", [6666], [5555,6666], cscript2, mary_kb) จากนั้น Inference ของเอเจนต์ทั้งสองจะทำงานเป็นวัฏจักร เกิดการสื่อสารกันผ่านพอร์ต 5555 และ 6666 โดย john จะขอสารเป็นไฟล์ "hello.html" จาก mary เมื่อได้รับไฟล์แล้วก็จะนำมาแสดงเป็นหน้าเว็บเพจ ส่วน mary ถูกโปรแกรมว่าหากมีใครขอไฟล์ข้อมูลมาก่อนเวลา 9.00 น. จะปฏิเสธ แต่จะเริ่มให้ได้ตั้งแต่เวลา 10.00 น. เป็นต้นไป ซึ่งผลที่ได้จากการรันโปรแกรมเป็นดังนี้

รูปที่ 8.1 หน้าจอแสดงการทำงานเป็นบราวเซอร์ของ "John"



รูปที่ 8.2 หน้าจอแสดงการทำงานเป็นเซิร์ฟเวอร์ของ "Mary"

8.1.2 ตัวอย่างการจองตั๋วภาพยนตร์

ตัวอย่างนี้แสดงการสื่อสารที่กำหนดโดยสคริปต์สื่อสารแบบบทสนทนาที่ซับซ้อนมากขึ้น จากตัวอย่างนี้เป็นการสื่อสารระหว่างเอเจนต์ John กับเอเจนต์ Cinema โดยบทสนทนาการสื่อสารของ John กำหนดให้เป็น cscript1 ของ Cinema เป็น cscript2 ดังนี้

John

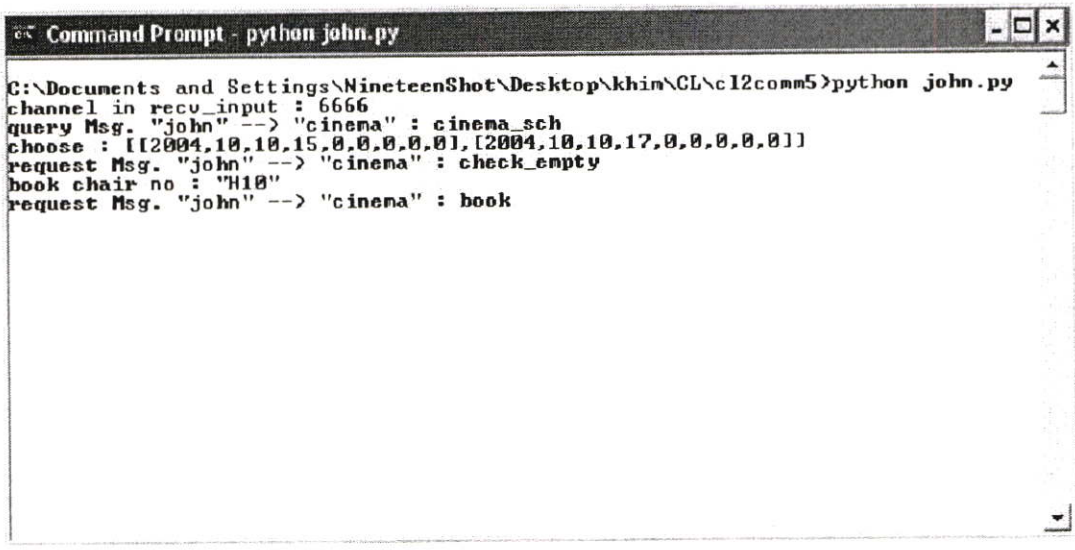
```
// cscript1 : Comm. Script for Agent John
commscript :
   sequence :
      exist(tell_msg(?ans,?snd,?recv,?id,id1)) →
         { conv_time = pickup(ans,mytable);
           send(Request('"check_empty(title,conv_time)"',recv,snd,id),5555);}
      exist(yes_msg(?result,?snd,?recv,?id,id2)) →
         { chair_no = choose_chair_no(result);
           send(Request('"book(title,conv_time,chair_no)"',recv,snd,id),5555);}
// script1 : Command Script for Agent John
title = "the terminal"
send(Query('"cinema_sch(title,today)"',agent( "john"), agent("cinema"),0),5555)
```

Cinema

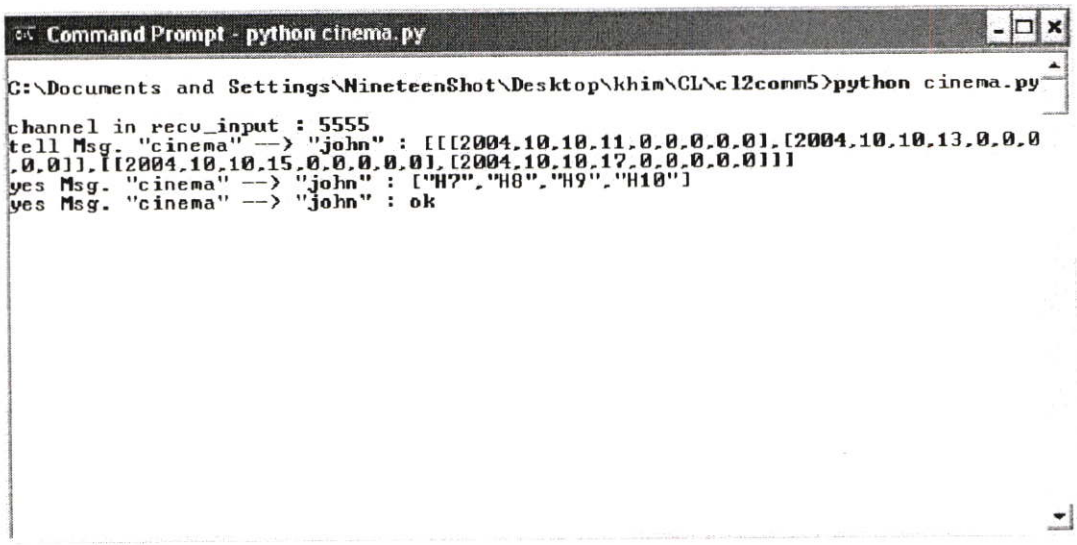
```
// cscript2 : Comm. Script for Agent Cinema
commscript :
   exist(query_msg(?question,?snd,?recv,?id,_)) →
      { result = solve(question);
        send(Tell('"result"',recv,snd,id),6666);}
   exist(request_msg(?order,?snd,?recv,?id,_)) →
      { result = demo(order);
        send(Yes ('"result"',recv,snd,id),6666);}
```

จากตัวอย่างที่เสนอขึ้นนี้ เอเจนต์ John จะทำการสื่อสารกับเอเจนต์ Cinema เพื่อจองตั๋วภาพยนตร์เรื่อง the terminal โดยเริ่มต้นที่ John จะถามถึงตารางการฉายภาพยนตร์จากเอเจนต์ Cinema เมื่อ John ได้รับตารางการฉายภาพยนตร์ของเรื่องที่สนใจแล้วก็จะตรวจสอบว่าสามารถชมภาพยนตร์ในรอบใดได้บ้าง (ด้วยการเรียกฟังก์ชัน pickup()) เมื่อ John เลือกรอบการฉายภาพยนตร์ที่ต้องการแล้ว จึงขอให้ Cinema ตรวจสอบว่ารอบการฉายดังกล่าวว่ามีที่นั่งเหลืออยู่หรือไม่ และมีที่ใดบ้าง เมื่อได้รับคำตอบซึ่งจะประกอบด้วยหมายเลขที่นั่งชมภาพยนตร์ที่ยังว่างอยู่ John ก็จะเลือกที่นั่งแล้วสั่งจองตั๋วภาพยนตร์

ส่วนเอเจนต์ Cinema นั้นจะมีหน้าที่ให้บริการตอบคำถาม ตรวจสอบและสั่งจองที่นั่งและตั๋วภาพยนตร์ให้ สำหรับการทำงานในส่วนแรกซึ่งเป็นบริการการตอบคำถามนั้น จะถูกโปรแกรมให้ตอบสนองต่อข้อความสื่อสารประเภทคำถาม (Query_msg) ในตัวอย่างนี้จะหมายถึงการถามรอบการฉายภาพยนตร์ สำหรับการทำงานในส่วนหลัง ได้แก่ หน้าที่ตรวจสอบและสั่งจองที่นั่งนั้น จะถูกโปรแกรมให้ตอบสนองต่อข้อความสื่อสารประเภทคำสั่ง (Request_msg) ซึ่งในตัวอย่างนี้จะหมายถึงคำสั่งให้ทำการตรวจสอบว่าในรอบการฉายภาพยนตร์ที่ลูกค้าสนใจมีที่นั่งว่างหรือไม่ และคำสั่งให้ทำการจองตั๋วภาพยนตร์ตามที่นั่งที่ลูกค้าต้องการ ผลที่ได้จากการรันโปรแกรมเป็นดังนี้



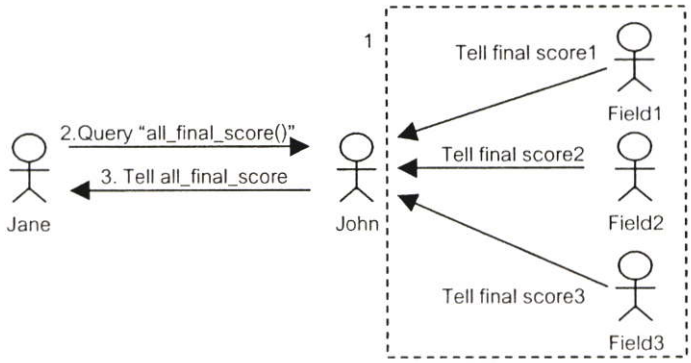
รูปที่ 8.3 หน้าจอแสดงการจองตั๋วภาพยนตร์ของ "John"



รูปที่ 8.4 หน้าจอแสดงการรับจองตั๋วภาพยนตร์ของ "Cinema"

8.1.3 ตัวอย่างการตรวจสอบผลการแข่งขันฟุตบอล

ตัวอย่างนี้แสดงการสื่อสารที่ใช้รูปแบบบทสนทนาประเภท Unordered ตัวอย่างนี้เป็นการสื่อสารระหว่างเอเจนต์ Jane กับเอเจนต์ John เพื่อทำการตรวจสอบผลการแข่งขันฟุตบอล โดยเอเจนต์ John ก็จะต้องสื่อสารกับเอเจนต์ Field ซึ่งเป็นเอเจนต์ที่ทำหน้าที่รายงานผลการแข่งขัน เมื่อการแข่งขันเสร็จสิ้นลง โดยในตัวอย่างนี้กำหนดให้มีการแข่งขันฟุตบอลทั้งหมด 3 คู่ ดังนั้นเอเจนต์ Field จะมี 3 ตัว ได้แก่ เอเจนต์ Field1, Field2 และ Field3 ตามลำดับ การทำงานในตัวอย่างนี้สามารถแสดงได้ตามรูปที่ 8.5



รูปที่ 8.5 การสนทนาเพื่อตรวจสอบผลการแข่งขันฟุตบอล

บทสนทนาการสื่อสารของเอเจนต์ทั้ง 5 เอเจนต์ เป็นดังนี้

Jane

```
commscript :
    exist(tell_msg(?score,"john3",_,_,_)) --> {meta_print(score);}
send(query(" "all_score" ", agent("myself"), agent("john3"), 0), 5555)
```

Field1

```
score = " Singapore vs. Vietnam 0 : 2 \n"
send(tell(" score ", agent("myself"), agent("john3"), 0), 5555)
```

Field2

```
score = " Thai vs. Malaysia 1 : 2 \n"
send(tell(" score ", agent("myself"), agent("john3"), 0), 5555)
```

Field3

```
score = " Indonesia vs. Phillipines 4 : 1\n"
send(tell(" score ", agent("myself"), agent("localhost"), 0), 5555)
```

John

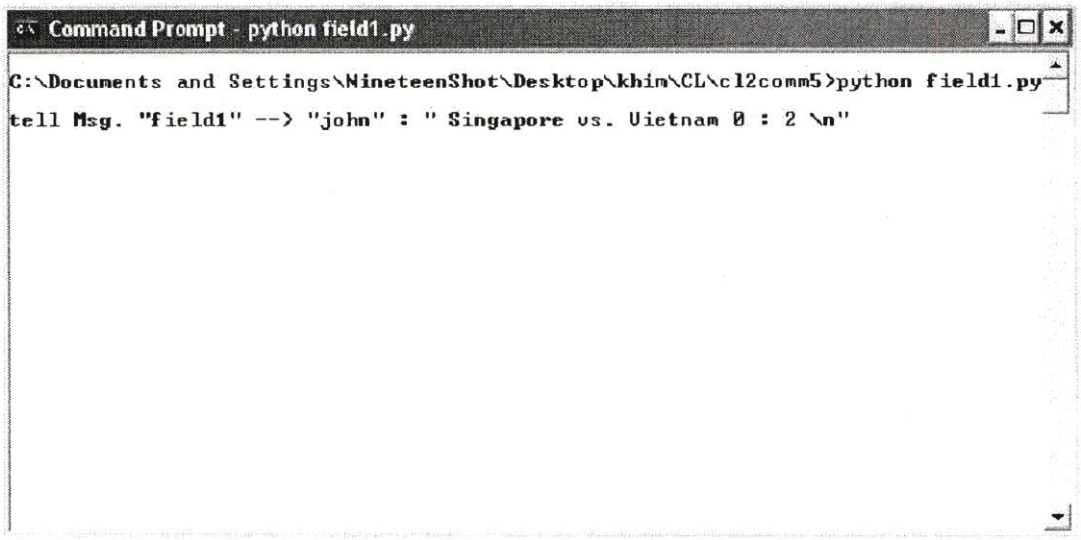
```
commscript :
    comm_unordered :
        exist(tell_msg(?score,"field1",_,_,_)) --> {score_f1 = meta2Obj(score);
            print("Received Score from agent Field1");}
        exist(tell_msg(?score,"field2",_,_,_)) --> {score_f2 = meta2Obj(score);
            print("Received Score from agent Field2");}
        exist(tell_msg(?score,"field3",_,_,_)) --> {score_f3 = meta2Obj(score);
            print("Received Score from agent Field3");}
    exist(query_msg(_,?snd,_,_,_)) --> {all_score = "The Football Score : \n" + score_f1 + score_f2 + score_f3;
        send(tell(" all_score ", agent("myself"), agent(snd), 0), 7777);}
```

จากตัวอย่าง เอเจนต์ Jane จะติดต่อกับเอเจนต์ John เพื่อสอบถามรายงานผลการแข่งขันฟุตบอลทั้งหมด 3 คู่ และเมื่อเอเจนต์ John ได้รับผลการแข่งขันแล้วก็จะแสดงผลนั้น

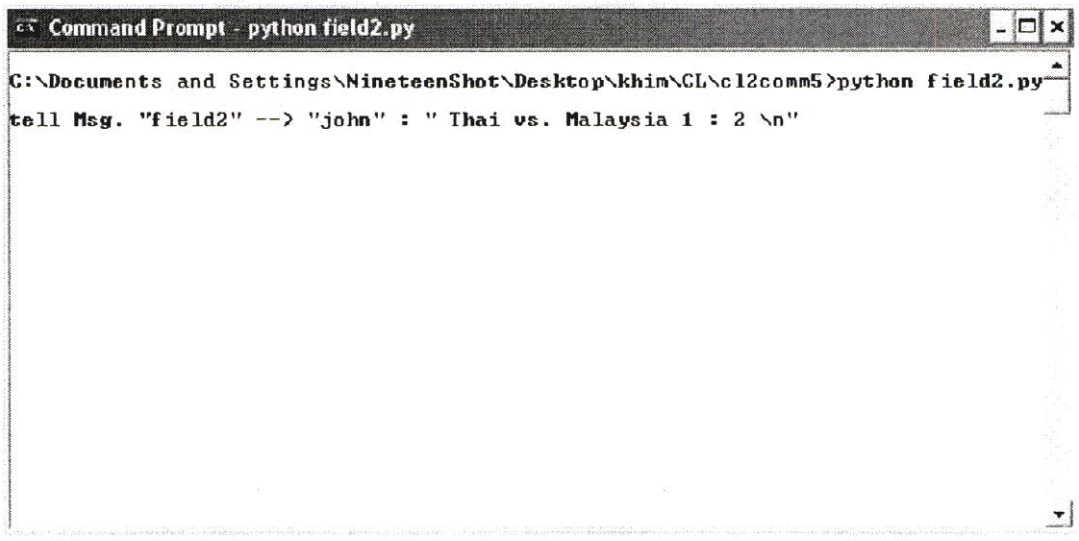
สำหรับเอเจนต์ John นั้น เป็นเอเจนต์ที่รวบรวมผลการแข่งขันทั้ง 3 สนาม โดยเอเจนต์ John จะรอรับผลการแข่งแต่ละคู่จากเอเจนต์ Field ทั้ง 3 เอเจนต์ เมื่อเอเจนต์ John ได้รับผลการแข่งขันก็จะบันทึกเก็บไว้ ซึ่งเอเจนต์ John สามารถรับผลการแข่งจากสนามใดก่อนก็ได้ แต่ต้องได้ผลครบทุกสนามจึงจะถือว่าทำงานได้สำเร็จ และเมื่อมีเอเจนต์อื่นสอบถามถึงผลการแข่งขัน เอเจนต์ John ก็จะส่งผลที่รวบรวมแล้วกลับไป

เอเจนต์ Field1, Field2, Field3 มีการทำงานเหมือนกัน นั่นคือ เมื่อมีการยุติการแข่งขันในสนาม เอเจนต์ในแต่ละสนามจะส่งผลการแข่งขันไปยังเอเจนต์ John

ผลจากการรันโปรแกรมเป็นดังแสดงไว้ในรูปที่ 8.6 ถึง 8.10 โดยเริ่มต้นต้องทำการรันโปรแกรมเอเจนต์ John ซึ่งเป็นเอเจนต์ที่คอยรับผลการแข่งฟุตบอลค้างเอาไว้ก่อน แล้วจึงรันโปรแกรมของเอเจนต์ Field ต่างๆ แล้วจึงรันโปรแกรมของเอเจนต์ Jane เพื่อถามผลการแข่งขันจาก John ในตัวอย่างนี้ทำการรันเอเจนต์ Field3, Field1 และ Field2 ตามลำดับ

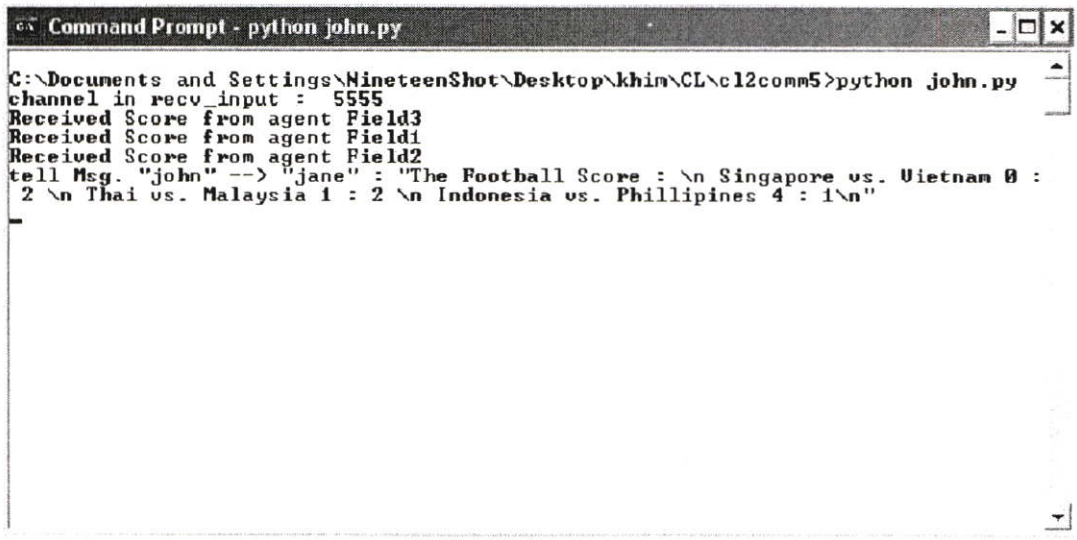


รูปที่ 8.6 หน้าจอแสดงการส่งผลการแข่งฟุตบอลของเอเจนต์ Field1



รูปที่ 8.7 หน้าจอแสดงการส่งผลการแข่งฟุตบอลของเอเจนต์ Field2

รูปที่ 8.8 หน้าจอแสดงการส่งผลการแข่งฟุตบอลของเอเจนต์ Field3

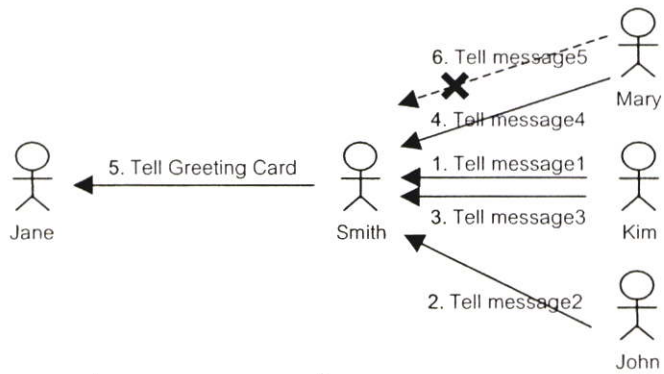


รูปที่ 8.9 หน้าจอแสดงการรับผลการแข่งฟุตบอลของเอเจนต์ John

รูปที่ 8.10 หน้าจอแสดงการตรวจสอบผลการแข่งฟุตบอลของเอเจนต์ Jane

8.1.4 ตัวอย่างการเขียนบัตรอวยพรร่วมกัน

ตัวอย่างนี้แสดงการสื่อสารที่ใช้รูปแบบบทสนทนาประเภท Independent Repeatable and Total ซึ่งเป็นการสื่อสารระหว่างกลุ่มเอเจนต์เพื่อร่วมกันเขียนบัตรอวยพร ในตัวอย่างนี้จะประกอบด้วยเอเจนต์ 5 ตัว ได้แก่ เอเจนต์ John, Mary, Kim, Smith, Jane โดยเอเจนต์ John, Mary, Kim และ Smith จะต้องร่วมกันเขียนบัตรอวยพรให้กับเอเจนต์ Jane และเมื่อเขียนคำอวยพรครบทุกเอเจนต์แล้วจึงส่งให้ Jane การทำงานในตัวอย่างนี้สามารถแสดงได้ตามรูปที่ 8.8



รูปที่ 8.11 การสนทนาเพื่อการเขียนบัตรอวยพรร่วมกัน

บทสนทนาของเอเจนต์ทั้ง 5 เอเจนต์ เป็นดังนี้

Jane

```
commscript :
    exist(tell_msg(?card,"smith",_,_,_)) --> {meta_print(card);}
```

Smith

```
commscript :
    comm_independent_total :
        exist(tell_msg(?msg,"john",_,_,_)) --> {john_msg = "";
            if (send_already==0) : {john_msg = john_msg+meta2Obj(msg); send_flag[0] = 1;
                print("Received Message from agent John");}
            else: {send(tell(""sended card already"",agent("myself"), agent("john"),0),6666);}}
        exist(tell_msg(?msg,"mary",_,_,_)) --> {mary_msg = "";
            if (send_already==0) : {mary_msg = mary_msg + meta2Obj(msg); send_flag[1] = 1;
                print("Received Message from agent Mary");}
            else: {send(tell(""sended card already"",agent("myself"), agent("mary"),0),6666);}}
        exist(tell_msg(?msg,"kim",_,_,_)) --> {kim_msg = "";
            if (send_already==0) :;{kim_msg = kim_msg + meta2Obj(msg);send_flag[2] = 1;
                print("Received Message from agent Kim");}
            else: {send(tell(""sended card already"",agent("myself"), agent("kim"),0),6666);}}
send_already = 0
send_flag = [0,0,0]
all_msg = "For you : \n Smith : Happy birthday! \n"
while (send_flag != [1,1,1]) :  x = 1
all_msg = all_msg + john_msg + mary_msg + kim_msg
send(tell(" all_msg ",agent("myself"), agent("jane"),0),7777)
print("send card already.....")
send_already = 1
```

John

```
commscript :
    exist(tell_msg(?msg,"smith",_,_,_)) --> {meta_print(msg);}
msg = " John : Congratuation! \n"
send(tell(" msg ", agent("myself"), agent("smith"), 0), 5555)
```

Mary

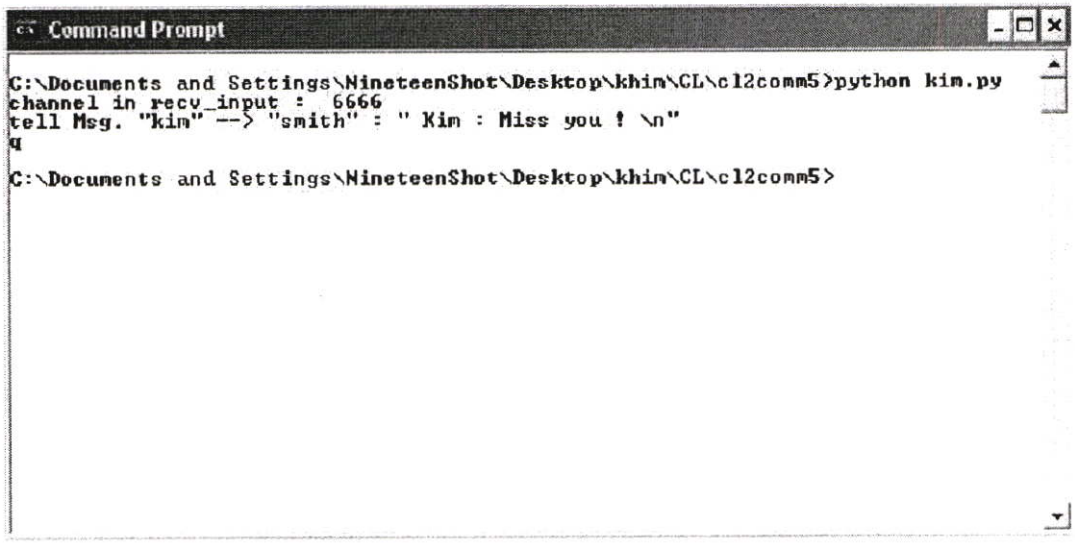
```
commscript :
    exist(tell_msg(?msg,"smith",_,_,_)) --> {meta_print(msg);}
msg = " Mary : Wish you have everything nice! \n"
msg1 = " Mary : Have a nice weekend! \n"
send(tell(" msg ", agent("myself"), agent("smith"), 0), 5555)
<10:00:00|12/12/2004,_> : send(tell(" msg1 ", agent("myself"), agent("smith"), 0), 5555)
```

Kim

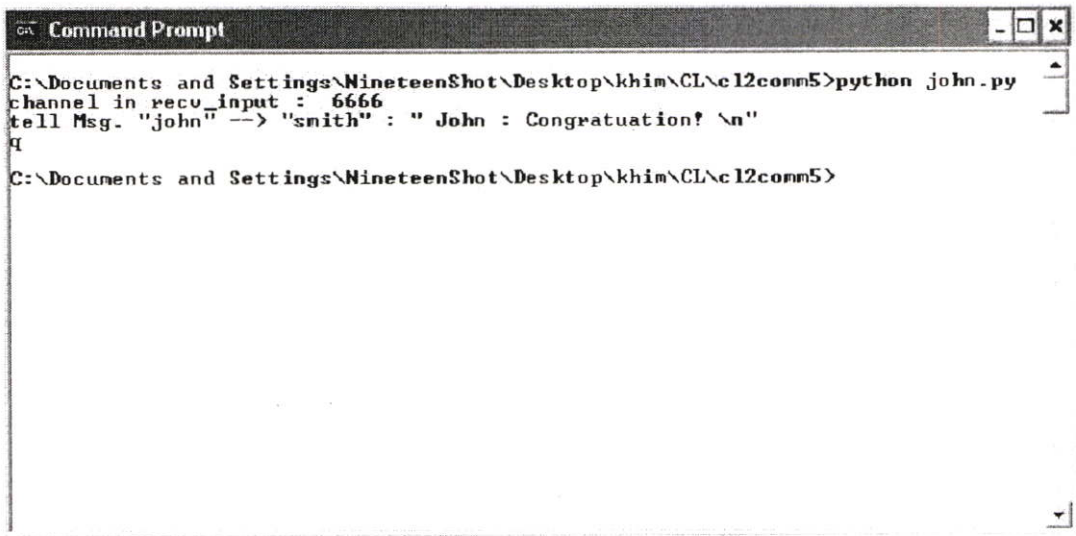
```
commscript :
    exist(tell_msg(?msg,"smith",_,_,_)) --> {meta_print(msg);}
msg = " Kim : Miss you ! \n"
send(tell(" msg ", agent("myself"), agent("smith"), 0), 5555)
```

จากตัวอย่าง เอเจนต์ John, Mary และ Kim จะมีโปรแกรมการทำงานเหมือนกัน นั่นคือ จะทำการส่งข้อความไปให้เอเจนต์ Smith ซึ่งเป็นเอเจนต์ที่มีหน้าที่รวบรวมข้อความในบัตรอวยพร โดยเอเจนต์ทั้งสามจะสามารถส่งข้อความเพื่อบันทึกลงไปในบัตรอวยพรได้กี่ครั้งก็ได้ตามต้องการ หากยังมีเอเจนต์ที่ยังไม่ได้เขียนคำอวยพรลงไป

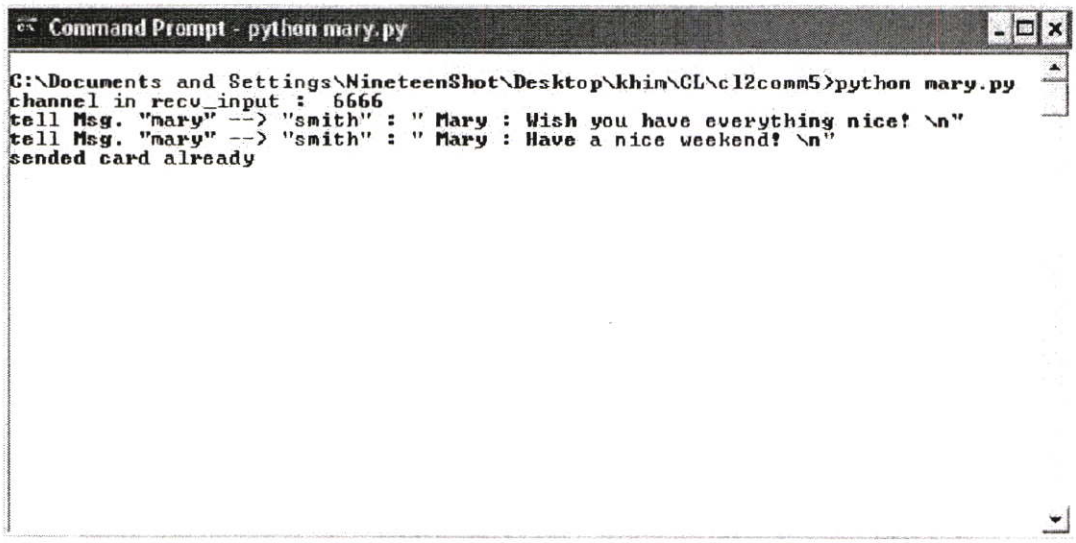
ผลการรันโปรแกรมทั้งสามเป็นดังรูปที่ 8.12 ถึง 8.16 โดยเริ่มต้นจากการรันโปรแกรมเอเจนต์ Jane ขึ้นมาค้างไว้เสียก่อนเพื่อเปิดให้รอรับการ์ดที่ได้จากเอเจนต์ Smith หลังจากนั้นจึงรันโปรแกรมเอเจนต์ John ค้างไว้เพื่อให้ทำการรอรับการส่งคำอวยพรจากเอเจนต์ Kim, John และ Mary แล้วจึงทำการรันโปรแกรมเอเจนต์ Kim, John และ Mary เพื่อตรวจสอบการสื่อสารระหว่างเอเจนต์ ซึ่งในตัวอย่างนี้ได้ทำการรันโปรแกรมเอเจนต์ Kim, John และ Mary ตามลำดับ โดยกำหนดให้เอเจนต์ Mary จะทำการส่งคำอวยพรมา 2 ครั้ง แต่ไม่สามารถเพิ่มเติมคำอวยพรในครั้งหลังได้เนื่องจาก Smith ได้ส่งการ์ดไปให้ Jane แล้ว



รูปที่ 8.12 หน้าจอแสดงลำดับการส่งคำอวยพรของเอเจนต์ Kim

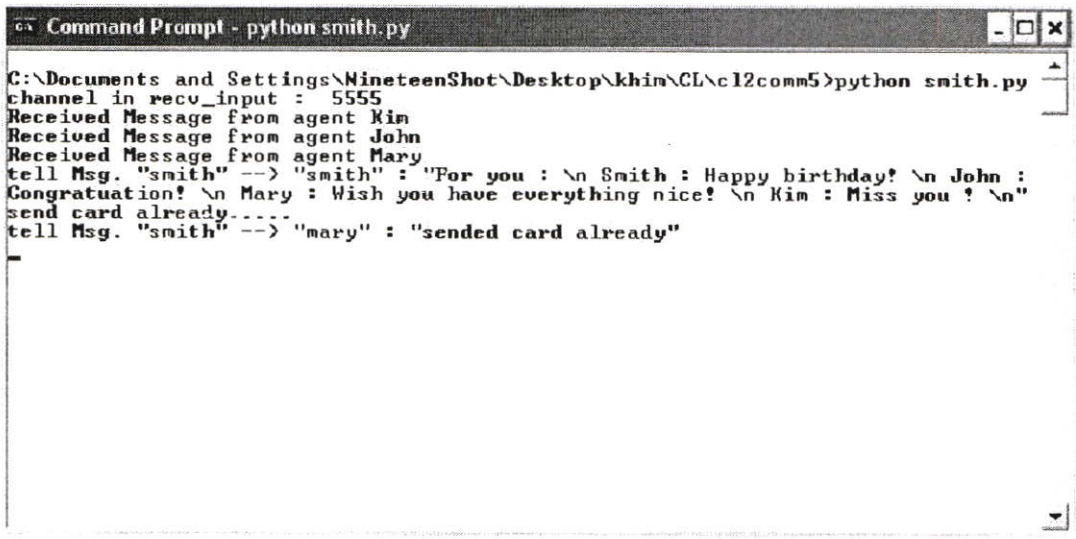


รูปที่ 8.13 หน้าจอแสดงลำดับการส่งคำอวยพรของเอเจนต์ John



รูปที่ 8.14 หน้าจอแสดงลำดับการส่งคำอวยพรของเอเจนต์ Mary

สำหรับเอเจนต์ Smith ผู้เป็นศูนย์กลางของการเขียนบัตรอวยพร โดยมีหน้าที่รับคำอวยพรจากเอเจนต์ John, Mary และ Kim ทั้งยังต้องคอยตรวจสอบว่ามีเอเจนต์มาบันทึกคำอวยพรครบหรือยัง เมื่อครบแล้ว เอเจนต์ Smith จะส่งบัตรอวยพรนั้นไปให้กับเอเจนต์ Jane ในกรณีที่ส่งบัตรอวยพรไปแล้ว แต่ยังคงมีเอเจนต์ส่งคำอวยพรมา Smith ก็จะส่งข้อความเพื่อบอกว่าได้ส่งบัตรอวยพรให้ Jane แล้ว



รูปที่ 8.15 หน้าจอแสดงลำดับการรับคำอวยพรของเอเจนต์ Smith

รูปที่ 8.16 หน้าจอแสดงการรับบัตรคำอวยพรของเอเจนต์ Jane

8.2 การนำแนวคิดภาษา CL ไปใช้ในงานวิจัยอื่น

แนวคิดการสร้างภาษาสำหรับการสื่อสาร หรือภาษา CL ในงานวิจัยนี้ คือ การสร้างภาษาคอมพิวเตอร์ที่สนับสนุนการทำงานตามเวลาและการตอบสนองต่อเหตุการณ์ภายนอก ซึ่งนำไปสู่ความสามารถในการติดต่อสื่อสารของเอเจนต์ แนวคิดดังกล่าวได้นำไปพัฒนาและใช้ในงานวิจัยอื่นๆ ได้แก่ บราวเซอร์ที่สามารถโปรแกรมได้, บราวเซอร์ที่สามารถโปรแกรมได้บนโทรศัพท์เคลื่อนที่ และ บราวเซอร์ที่โปรแกรมได้บนเครื่อง Pocket PC ซึ่งทั้งหมดนี้เป็นการสร้างบราวเซอร์ที่สามารถทำงานได้ตามที่โปรแกรมตามคำสั่งสคริปต์บนอุปกรณ์ต่างๆ

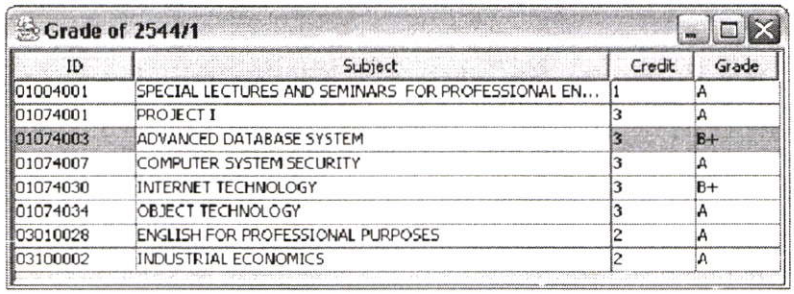
8.2.1 บราวเซอร์ที่สามารถโปรแกรมได้ [4]

หลักการการทำงานของ Programmable Browser หรือ บราวเซอร์ที่สามารถโปรแกรมได้ คือเอเจนต์ที่จะทำงานตามที่ผู้ใช้สั่ง และจะสามารถตอบสนองต่อสิ่งแวดล้อมตามที่ผู้ใช้กำหนดได้ ซึ่งการทำงานตามที่ผู้ใช้สั่งนั้นสามารถทำได้โดยให้ผู้ใช้สั่งงานเป็นคำสั่งสคริปต์แล้วให้เอเจนต์ประมวลผลสคริปต์นั้น ส่วนการตอบสนองต่อสิ่งแวดล้อมหรือเหตุการณ์ภายนอกนั้น ผู้ใช้สามารถทำได้โดยการกำหนดเหตุการณ์และสิ่งที่เอเจนต์ต้องการตอบสนอง โดยกำหนดว่าเมื่อเหตุการณ์ใดเกิดขึ้นแล้วจะให้ตอบสนองโดยกระทำอะไร ซึ่งในการทำงานเอเจนต์จะมีฐานความรู้เป็นของตนเองซึ่งเก็บรวบรวมสคริปต์และความรู้เกี่ยวกับเหตุการณ์และการตอบสนองของเอเจนต์ ผู้ใช้สามารถเพิ่มเติมความรู้ใหม่ๆเข้าไปให้เอเจนต์ได้

1) **ตัวอย่างการทำงานแบบ** Sequential

ตัวอย่างนี้แสดงการดึงข้อมูลเกรดจากเว็บของพระจอมเกล้าลาดกระบัง มาแสดงผลในรูปแบบตาราง โดยส่วนที่ 1 เป็นการกำหนดค่าตัวแปรต่างๆที่ต้องใช้ดูเกรด เช่น รหัสนักศึกษา ในส่วนที่ 2 เป็นส่วนที่ทำการดึงข้อมูลเกรดจากเว็บพระจอมเกล้าลาดกระบัง ซึ่ง KMITL คือบิวท์อินคลาสที่ผู้พัฒนาสร้างขึ้น และในส่วนที่ 3 เป็นการแสดงผลข้อมูลเกรดที่ได้ในรูปแบบตาราง ซึ่งฟังก์ชันในการสร้างตารางนั้นผู้พัฒนาได้สร้างเอาไว้แล้ว

รูปที่ 8.17 สคริปต์ดึงข้อมูลเกรด



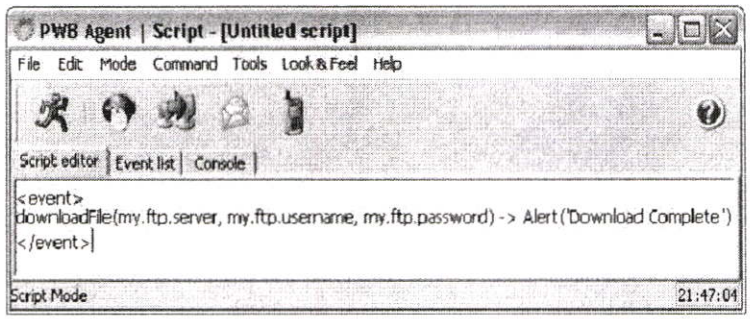
Grade of 2544/1

ID	Subject	Credit	Grade
01004001	SPECIAL LECTURES AND SEMINARS FOR PROFESSIONAL EN...	1	A
01074001	PROJECT I	3	A
01074003	ADVANCED DATABASE SYSTEM	3	B+
01074007	COMPUTER SYSTEM SECURITY	3	A
01074030	INTERNET TECHNOLOGY	3	B+
01074034	OBJECT TECHNOLOGY	3	A
03010028	ENGLISH FOR PROFESSIONAL PURPOSES	2	A
03100002	INDUSTRIAL ECONOMICS	2	A

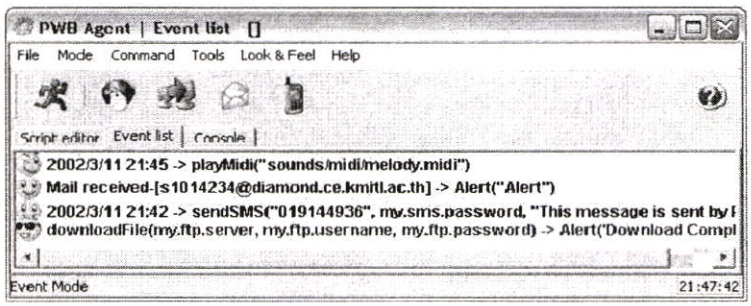
รูปที่ 8.18 ผลลัพธ์ของการเขียนสคริปต์ดึงข้อมูลเกรด

2) **ตัวอย่างการทำงานของส่วนจัดการเหตุการณ์** (Event Handler)

จากตัวอย่างเป็นการสั่งงานในรูปแบบ Event->Action เมื่อเหตุการณ์ที่ระบุเป็นจริง บราวเซอร์จะทำการตอบสนองเหตุการณ์นั้นตามที่กำหนดไว้ จากตัวอย่างเป็นเหตุการณ์ประเภทเอ็กเพรสชั่นใดๆ ซึ่งอนุโลมใช้แทน Boolean Expression ได้ ซึ่งคำสั่งมีความหมายว่าหากทำการถ่ายโอนข้อมูลสำเร็จแล้วให้บราวเซอร์แจ้งให้ผู้ใช้ทราบด้วย และได้มีการแสดงรายการ Event-> Action ใน Event List Monitor รวมถึงบอกสถานะของรายการ Event->Action ด้วยไอคอน



รูปที่ 8.19 การสั่งงานด้วยเหตุการณ์



รูปที่ 8.20 ผลลัพธ์ของการจัดการเหตุการณ์

8.2.2 บราวเซอร์ที่สามารถโปรแกรมได้บนโทรศัพท์เคลื่อนที่ [5]

งานวิจัยนี้มีแนวคิดที่พัฒนาต่อจากบราวเซอร์ที่สามารถโปรแกรมได้ข้างต้น โดยนำบราวเซอร์นั้นมาใช้งานบนโทรศัพท์เคลื่อนที่ ทำให้ผู้ใช้งานสามารถเข้าถึงข้อมูลได้ทุกเวลาทุกสถานที่ โดยใช้ภาษา Java 2 Micro Edition หรือ J2ME และภาษา XHTML ในการสร้างบราวเซอร์ดังกล่าว

8.2.3 บราวเซอร์ที่โปรแกรมได้บนเครื่อง PocketPC [6]

งานวิจัยนี้ เป็นงานวิจัยที่พัฒนาต่อเนื่องมาจากบราวเซอร์ที่สามารถโปรแกรมได้ โดยได้พัฒนาบราวเซอร์ขึ้นโดยอาศัยภาษา PythonCE บนเครื่อง PocketPC แต่ชุดคำสั่งสคริปต์ที่นำมาใช้ควบคุมการทำงานของบราวเซอร์ทั้งหมดเป็นส่วนหนึ่งในภาษา CL ที่ได้นำเสนอในงานวิจัยชิ้นนี้ ซึ่งมีรูปแบบที่แตกต่างจากคำสั่งสคริปต์ของบราวเซอร์ในหัวข้อ 8.2.1 คือ ได้เพิ่มการประมวลผลด้านเวลา และ เพิ่มความสามารถในการประมวลผลชุดคำสั่งที่หลากหลายมากขึ้น ทำให้บราวเซอร์นี้มีความสามารถในการทำงานและตรวจสอบเหตุการณ์ตามเวลาที่กำหนดได้

บทที่ 9
วิเคราะห์และวิจารณ์

9.1 วิเคราะห์ผลการทดลอง

จากการทดลองสร้างเอเจนต์สื่อสารที่ใช้โปรแกรมภาษา CL ในบทที่ 8 นั้น เมื่อเปรียบเทียบกับภาษาที่พัฒนาสำหรับการสื่อสารของเอเจนต์ (ACL : Agent Communication Language) ที่มีอยู่ในปัจจุบัน เช่น KQML[14], DALI[15] เป็นต้น ทำให้สามารถวิเคราะห์ข้อดีและข้อบกพร่องของงานวิจัยได้หลายประการ ดังนี้

9.1.1 ข้อดีของงานวิจัย

1) *ความสามารถในการประมวลผลเวลา* โดยโปรแกรมเมอร์สามารถระบุเวลาในการทำคำสั่ง และสามารถระบุช่วงเวลาในการพิจารณาข้อความสื่อสารที่เข้ามาได้ นับเป็นลักษณะเด่นที่สำคัญของงานวิจัย เนื่องจาก เวลา เป็นองค์ประกอบสำคัญในการกระทำกิจกรรมต่างๆ การพัฒนาภาษาคอมพิวเตอร์ให้มีความสามารถในการประมวลผลทางด้านเวลา ทำให้ภาษาคอมพิวเตอร์นั้นมีความยืดหยุ่น และสามารถนำไปใช้ในการทำงานได้หลากหลาย โดยจะสังเกตว่าในงานวิจัยอื่นๆ ยังไม่สามารถจัดการเกี่ยวกับการทำงานที่สัมพันธ์กับเวลาได้
2) *ความสามารถในการสื่อสาร* โปรแกรมเมอร์สามารถกำหนดบทสนทนาของโปรแกรมได้ด้วยสคริปต์สื่อสาร ทำให้โปรแกรมสามารถสื่อสารกับโปรแกรมอื่นๆ ได้ นอกจากนี้ภาษา CL ยังสนับสนุนแนวความคิดที่มองว่า การสนทนาแต่ละเรื่องจะต้องมีรูปแบบ (Pattern) ที่ใช้ในการติดต่อสื่อสารเสมอ เช่น จะต้องถามคำถามอย่างไร และควรจะตอบอย่างไร เมื่อได้คำตอบจะถามต่อเช่นไร เป็นต้น ซึ่งแนวคิดนี้ยังไม่ปรากฏในงานวิจัยอื่นๆ ทำให้การสื่อสารในโปรแกรมภาษา CL มีความซับซ้อนและสอดคล้องกับการสนทนาของมนุษย์มากกว่างานวิจัยอื่นๆ
3) *สนับสนุนการประมวลผลสัญลักษณ์ในระดับเมต้า* ในภาษา CL มีการแบ่งสัญลักษณ์ออกเป็น 2 ระดับ คือ ระดับวัตถุ(Object Level) และระดับเมต้า(Meta Level) ซึ่งเป็นสัญลักษณ์ที่เข้าใจได้ในระดับเอเจนต์เท่านั้น โดยในการสื่อสารแต่ละครั้ง จะต้องมีการแปลงสัญลักษณ์ระดับวัตถุให้เป็นสัญลักษณ์ระดับเมต้าก่อนที่จะส่งสารออกไป และทำนองเดียวกันก็ต้องมีการแปลงสัญลักษณ์เมต้ากลับเป็นวัตถุเพื่อประมวลผลสาร หลักการดังกล่าวสอดคล้องกับลักษณะการสื่อสารของคนที่จะต้องทำการแปลง

สิ่งที่ต้องการจะสื่อ (Object level) ให้อยู่ในรูปของคำพูด (Meta level) ในการส่งสาร และเมื่อได้รับสาร ก็จะต้องทำความเข้าใจในคำพูดที่ได้ยินนั่นเอง ในงานวิจัยอื่นๆ การแบ่งแยกระหว่าง Object level และ Meta level ยังไม่ชัดเจน

4) *สามารถทำงานพื้นฐานได้เช่นเดียวกับภาษาคอมพิวเตอร์ทั่วไป* โดยมองว่าเป็นการพูดคุยกับ CL ผ่านทาง User

5) *ภาษา CL เป็นภาษาที่อ่านและเข้าใจได้ง่าย* เนื่องจากไวยากรณ์ของภาษาส่วนหนึ่งได้ใช้ไวยากรณ์ของภาษาไพธอนที่มีรูปแบบที่ง่ายอยู่แล้ว อีกทั้งยังมีการแยกส่วนของสคริปต์สื่อสารออกไปอย่างชัดเจน และข้อความสื่อสารที่ใช้ได้กำหนดขึ้นจากกระบวนการสื่อสารของคนที่พบเห็นกันอยู่ทั่วไป จึงไม่ใช่เรื่องยากที่ผู้อ่านจะสามารถเข้าใจความหมายของการโปรแกรมได้

6) *ภาษา CL มีความยืดหยุ่นสูง* เนื่องจาก รูปแบบและโครงสร้างของภาษา CL ถูกพัฒนาขึ้นโดยคำนึงถึงการทำกิจกรรมของมนุษย์เป็นสำคัญ ไม่ว่าจะเป็นการทำงานที่เกี่ยวข้องกับเวลา หรือการทำงานที่ต้องอาศัยการติดต่อสื่อสาร ด้วยความสามารถเหล่านี้ ทำให้ภาษา CL มีความยืดหยุ่นและสามารถนำไปใช้งานได้หลากหลาย

7) *สามารถนำไปพัฒนาเอเจนต์ที่ชาญฉลาดได้ในอนาคต*

9.1.2 ข้อบกพร่องของงานวิจัย

1) *ความเร็วในการประมวลผล* เนื่องจากภาษา CL เป็นภาษาเมต้าของภาษาไพธอนซึ่งเขียนขึ้นในรูปแบบจำลองผ่านการใช้ PLY Tool ทำให้ภาษา CL ที่พัฒนาขึ้นนี้ใช้เวลาในการประมวลผลมากกว่าการประมวลผลในภาษาคอมพิวเตอร์ทั่วไป

2) *ความสมบูรณ์ของภาษา* ผู้วิจัยได้ทำการพัฒนาภาษา CL อยู่ในลักษณะของต้นแบบ (Prototype) ยังขาดเรื่อง Performance และยังมีข้อผิดพลาดอยู่บ้าง เช่น ภาษา CL ไม่มีความสามารถในการกำหนดคลาส

บทที่ 10
สรุปงานวิจัยและข้อเสนอแนะ

ในงานวิจัยนี้ได้นำเสนอการออกแบบและการพัฒนาภาษา CL เพื่อนำไปใช้ในการสร้างเอเจนต์ที่มีความสามารถในการสื่อสาร โดยผู้เขียนโปรแกรมสามารถระบุช่วงเวลาการทำงานให้กับคำสั่งในโปรแกรม และสามารถโปรแกรมให้คอมพิวเตอร์กระทำคำสั่งเพื่อตอบสนองต่อข้อความสื่อสารที่ได้รับจากเอเจนต์หรือโปรแกรมภาษา CL อื่นๆ ในรูปแบบของการใช้บทสนทนา เนื่องด้วยความสามารถในการสื่อสารและการประมวลผลเวลาที่สอดคล้องกับการทำงานของคน ทำให้ CL เป็นภาษาที่มีความยืดหยุ่น และสามารถนำไปใช้ในงานต่างๆ ได้หลากหลาย

ในส่วนการประมวลผลเวลานั้น ภาษา CL ได้ศึกษาถึงการเสนอรูปแบบของเวลา ความสัมพันธ์ของเวลา และความสัมพันธ์ระหว่างเวลาและการทำงานจากนักวิจัยหลายท่าน จนสามารถแจกแจงเวลาได้เป็น 3 ประเภท คือ จุดเวลา (Time Point) ช่วงเวลา (Time Intervel) และระยะเวลา (Duration) และสามารถอธิบายความสัมพันธ์ระหว่างเวลาประเภทต่างๆ ทั้งยังเสนอว่าเวลาเป็นจุดเริ่มต้นของการเกิดกระบวนการใดๆ ดังนั้น ไม่ว่าจะเป็นการทำงานหรือการเกิดเหตุการณ์ล้วนแล้วสัมพันธ์กับเวลาทั้งสิ้น จึงเห็นว่าความสามารถของการระบุช่วงเวลาการทำงานให้กับโปรแกรมของภาษา CL นั้นจะทำให้โปรแกรมที่พัฒนามีประสิทธิภาพมากขึ้น

ในส่วนของการสื่อสาร วิทยานิพนธ์นี้ได้อธิบายถึงนามธรรมของการสื่อสารระหว่างเอเจนต์โดยอาศัยหลักการคิดที่ระดับเมต้า (Meta-reasoning) นำไปสู่การพัฒนา CL ภาษาที่ใช้สร้างเอเจนต์เพื่อการสื่อสารที่มีความยืดหยุ่นสูง โดยผู้เขียนโปรแกรมสามารถสร้างเอเจนต์ให้สื่อสารกันได้ตามบทสนทนา ตามประเภทโปรโตคอล และตามช่วงเวลาที่ต้องการ ผ่านทางสคริปต์ตอบสนองต่อเหตุการณ์ (Event response Scripts) หรือสคริปต์ตอบสนองการสื่อสาร (Communication Script) การสื่อสารของแต่ละเอเจนต์เกิดขึ้นจากการทำงานตามวัฏจักร Observe–Fire Comm. Script–Act

แนวทางที่จะพัฒนาในอนาคต

1) พัฒนาภาษา CL ให้สมบูรณ์มากขึ้น โดยเพิ่มรายละเอียดโครงสร้างและการทำงานในภาษา เช่น การเพิ่มความสามารถในการกำหนดคลาส เป็นต้น
2) ปรับปรุงอินเตอร์พรีตเตอร์ให้ทำงานได้รวดเร็วขึ้น
3) พัฒนาข้อความสื่อสารให้สมบูรณ์ครบถ้วนตามที่ได้เสนอในแนวคิดในงานวิจัย
4) เพิ่มความสามารถในการจัดตารางเวลา การประมวลผลแบบ Scheduling อย่างเช่นที่ใช้ในระบบปฏิบัติการ เพื่อขยายความสามารถและประสิทธิภาพของการใช้ภาษา CL

5) พัฒนาส่วนติดต่อผู้ใช้ (User interface) ให้ดียิ่งขึ้น เพื่อให้สามารถใช้งาน CL ได้สะดวก เช่น การสร้างหน้าต่างปฏิทิน

เอกสารอ้างอิง

[1] Beazley D.M. **Python Essential Reference.** : New Riders. 1999.

[2] Chun W. J. **Core Python Programming.** : Prentice Hall. Upper Saddle River. 2001.

[3] วชิระ ศิริพจนาวรรณ และ สุพัณณดา โชติพันธ์. "**บราวเซอร์ที่โปรแกรมได้**". ปริญญานิพนธ์ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีเจ้าคุณทหารลาดกระบัง ปีการศึกษา 2543.

[4] ธนวัฒน์ แก้วค้า และ บุญทวี สันติศรีวราภรณ์. "**บราวเซอร์ที่โปรแกรมได้โดยใช้ภาษาไจธอน**". ปริญญานิพนธ์ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีเจ้าคุณทหารลาดกระบัง ปีการศึกษา 2544.

[5] กฤษณ์ชัย วันชัยนาวิน และ ณัฐวุฒิ นาไวย. "**อินเตอร์พรีตเตอร์ไพธอนขนาดเล็กสำหรับบราวเซอร์ที่โปรแกรมได้บนโทรศัพท์มือถือ**". ปริญญานิพนธ์ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีเจ้าคุณทหารลาดกระบัง ปีการศึกษา 2545.

[6] วิศิษฏ์ หิรัญกิตติ และคณะ. "บราวเซอร์ที่สามารถโปรแกรมได้". **การประชุมวิทยาการและวิศวกรรมคอมพิวเตอร์แห่งชาติ ครั้งที่** 6 NCSEC2002. 2545.

[7] ชาคริต เฮงสิริกุล และ เชิดเกียรติ แซ่แต้. "**บราวเซอร์ที่โปรแกรมได้บนเครื่อง** Pocket PC". ปริญญานิพนธ์ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีเจ้าคุณทหารลาดกระบัง ปีการศึกษา 2546.

[8] Allen, J. "Maintaining knowledge about temporal intervals" . **Communications of ACM 26(11).** 1983. pp832-843.

[9] Allen, J. "Time and Time Again : The Many Ways to Represent Time". **International Journal of Intelligent Systems.** 6(4). July 1991. pp341-355.

[10] Allen, J. "Towards a General Theory of Action and Time". **Artificial Intelligence.** 23. 1984. pp123-154.

[11] Elton LR. And Messel H. **Time and Man.** : Pergamon Press. 1978.

[12] Kowalski R. and Sergot M. "A Logic-based Calculus of Events". **New Generation Computing.** 4. 1986. pp67-95.

[13] Labrou Y., Finin T., Peng Y. "Agent Communication Languages : The Current Landscape". **IEEE Intelligent systems.** MARCH/APRIL. 1999.

[14] Finin T., Labrou Y. and Mayfield J., "KQML as an Agent Communication Language.", Baltimore, U.S.A., 1995, 1-22

[15] Costantini S., Tocchio A., Verticchio A. **"Communication Architecture in the DALI Logic Programming Agent-Oriented Language"**. [online]. Available : www.cs.unipr.it/CILC04/DownloadArea/ CostantiniTocchio_demo.pdf.

[16] **"PLY(Python Lex-Yacc)"**. [online]. Available : http://systems.cs.uchicago.edu/ply.

[17] Hayes, P. J. "Computation and Deduction". **Proc. Of 2nd Symposium on the Mathematical Foundations of Computer Science**. Czechoslovak. pp. 105-118. 1973.

[18] Kowalski R. A. **Using Metalogic to Reconcile Reactive with Rational Agents**. in *Meta-Logics and Logic Programming*, K. Apt and F. Turini (Gds.). : MIT Press. 1995.

[19] Kowalski R. A. ***Logic for Problem Solving***. Elsevier-North Holland, 1979.

[20] วิศิษฏ์ หิรัญกิตติ และสุพัณณดา โชติพันธ์. "CL: ภาษาสำหรับการสั่งงานคอมพิวเตอร์ด้วยเวลาและเหตุการณ์". **การประชุมวิทยาการและวิศวกรรมคอมพิวเตอร์แห่งชาติ ครั้งที่** 7 NCSEC2003. 2546.

ประวัติผู้เขียน

ชื่อ-นามสกุล	นางสาวสุพัณณดา โชติพันธ์
วัน เดือน ปีเกิด	21 กันยายน 2522 ที่อุบลราชธานี
ที่อยู่	209 ถ.อุปลีสาน ต.ในเมือง อ.เมือง จ.อุบลราชธานี 34000
ประวัติการศึกษา	2544 วิศวกรรมศาสตรบัณฑิต สาขาวิชาวิศวกรรมคอมพิวเตอร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง