ข้อกำหนดและการจำลองการทำงานของโปรโตคอล TFTP โดยใช้เครื่องมือ EDT และ Estelle Graphical Editor

Formal Specification and Simulation of TFTP Protocol Using EDT and Estelle Graphical Editor Tools

อาหมาน หมัดเจริญ

Ahmarn Mudcharoen

วิทยานิพนธ์วิทยาศาสตรมหาบัณฑิต สาขาวิชาวิทยาการคอมพิวเตอร์

มหาวิทยาลัยสงขลานครินทร์

Master of Science Thesis in Computer Science

Prince of Songkla University

2542

ชื่อวิทยานิพนธ์	ข้อกำหนดและการจำลองการทำงานของโปรโตคอล TFTP โดยใช้เครื่องมือ EDT และ Estelle Graphical Editor
ผู้เขียน	นายอาหมาน หมัดเจริญ
สาขาวิชา	วิทยาการคอมพิวเตอร์

คณะกรรมการที่ปรึกษา

อาจิน จิรชีพพัฒนา ...ประธานกรรมการ
(ผู้ช่วยศาสตราจารย์ ดร. อาจิน จิรชีพพัฒนา)

ติดราชการต่างประเทศ กรรมการ
(อาจารย์ ดารารัตน์ แซ่ลี้)

คณะกรรมการสอบ

อาจิน จิรชีพพัฒนา ประธานกรรมการ
(ผู้ช่วยศาสตราจารย์ ดร. อาจิน จิรชีพพัฒนา)

ติดราชการต่างประเทศ กรรมการ
(อาจารย์ ดารารัตน์ แซ่ลี้)

.....กรรมการ
(ผู้ช่วยศาสตราจารย์ อิว ไอยราคาญจนกุล)

.....กรรมการ
(อาจารย์ ปราโมทย์ จูฑาพร)

บัณฑิตวิทยาลัย มหาวิทยาลัยสงขลานครินทร์ อนุมัติให้นับวิทยานิพนธ์ ฉบับนี้เป็นส่วนหนึ่งของการศึกษา ตามหลักสูตรวิทยาศาสตรมหาบัณฑิต สาขาวิชาวิทยาการคอมพิวเตอร์

.....
(รองศาสตราจารย์ ดร.ก้าน จันทร์พรหมมา)
คณบดีบัณฑิตวิทยาลัย

ชื่อวิทยานิพนธ์	ข้อกำหนดและการจำลองการทำงานของโปรโตคอล TFTP โดยใช้เครื่องมือ EDT และ Estelle Graphical Editor
ผู้เขียน	นายอาหมาน หมัดเจริญ
สาขาวิชา	วิทยาการคอมพิวเตอร์
ปีการศึกษา	2542

บทคัดย่อ

การออกแบบโปรโตคอลสำหรับสื่อสารผ่านทางคอมพิวเตอร์นั้นผู้ออกแบบแต่ละรายจะทำการออกแบบขึ้นมาแล้วเขียนรายละเอียดขั้นตอนการทำงานของโปรโตคอลด้วยภาษาและสัญลักษณ์ที่เป็นที่เข้าใจของตนเอง ทำให้เป็นการยากในการที่ผู้อื่นจะนำโปรโตคอลนั้นไปพัฒนาและนำมาใช้งานจริง จึงได้มีการกำหนดมาตรฐานการเขียนข้อกำหนดโปรโตคอลขึ้นเรียกว่า FDT (Formal Description Technipue) ซึ่ง Estelle ก็เป็นหนึ่งในวิธีการเขียนข้อกำหนดตามมาตรฐานนี้

งานวิจัยนี้เป็นการเขียนข้อกำหนดของโปรโตคอล TFTP ตามแบบภาษา Estelle โดยใช้เครื่องมือ Estelle graphical editor ช่วย ขั้นแรกจะเป็นการศึกษาการทำงานของโปรโตคอล TFTP ให้ทราบรายละเอียดลักษณะการรับส่งข้อมูลแล้วทำการเขียนข้อกำหนดโปรโตคอลตามหลักภาษา Estelle โดยศึกษาลักษณะการออกแบบโครงสร้างโปรโตคอลเป็นลำดับชั้นของวัตถุใน Estelle หรือที่เรียกว่า Module แล้วทำการกำหนดรายละเอียดการทำงานของโมดูลซึ่งจะเป็นการกำหนดสถานะและพฤติกรรมต่าง ๆ ของโมดูล

เครื่องมือ Estelle Graphical Editor เป็นเครื่องมือที่ใช้ในช่วยในการเขียนข้อกำหนดโปรโตคอลตามแบบภาษา Estelle โดยเขียนข้อกำหนดในรูปแบบเชิงสัญลักษณ์ที่เรียกว่า Estelle/GR เมื่อเขียนข้อกำหนดในรูปแบบสัญลักษณ์เสร็จเครื่องมือนี้จะสามารถแปลงข้อกำหนดในรูปแบบสัญลักษณ์ให้เป็น Estelle เชิงอักขระได้ เมื่อได้ข้อกำหนดโปรโตคอลในรูปแบบภาษา Estelle เชิงอักขระหรือ Estelle/PR แล้วจึงใช้เครื่องมือ EDT ในการตรวจสอบความถูกต้องและจำลองการทำงานของโปรโตคอล TFTP ผลลัพธ์ที่ได้จากการจำลองการทำงานของโปรโตคอลก็มีการแสดงไว้ในวิทยานิพนธ์ด้วย

Thesis Title	Fomal Specification and Simulation of TFTP Protocol Using EDT and Estelle Graphical Editor Tools
Author	Mr. Ahmarn Mudcharoen
Major Program	Computer Science
Academic Year	1999

Abstract

Protocol design is up to the designers who use their own languages and symbols which lead any implementer with some difficulties. Consequently, standard techniques have been developed to describe protocol behavior so that any ambiguity is excluded. These techniques are called FDT (Formal Description Technique) and Estelle is one of these techniques.

The aim of this thesis is to write a specification of the TFTP protocol according to the Estelle technique by using Estelle Graphical Editor tool and to simulate the specification using the EDT tool. The first step is to study the detail of the TFTP protocol, Estelle syntax and semantic, and then to design the protocol into units of Estelle (i.e. modules). Then each module is described in detail by providing its control states and behavior.

Estelle Graphical Editor is a tool that make writing protocol specification in Estelle more easier by providing graphical representation. Estelle in graphical representation is called "Estelle/GR" . When finishing writing specification in Estelle/GR the tool can convert it into Estelle in phrasal form. Having specification of the TFTP in Estelle language, the EDT tool is used to check the correction of the specification and to simulate the behavior of TFTP protocol. The simulation results are also presented in the thesis.

กิตติกรรมประกาศ

วิทยานิพนธ์นี้สำเร็จได้อย่างสมบูรณ์ ด้วยความช่วยเหลือและสนับสนุนจากบุคคลหลายฝ่ายซึ่งผู้วิจัยรู้สึกทราบซึ้ง และขอกราบขอบพระคุณอย่างสูงมา ณ. โอกาสนี้ คือ

ผศ.ดร. อาจิน จิรชีพพัฒนา อาจารย์ที่ปรึกษา ที่กรุณาให้คำปรึกษา ข้อเสนอแนะทางวิชาการและแก้ไขปัญหาต่าง ๆ ตลอดจนตรวจทานวิทยานิพนธ์ให้แก่ผู้วิจัยมาโดยตลอด

อาจารย์ดารารัตน์ แซ่ลี้ อาจารย์ที่ปรึกษาร่วม ที่ได้กรุณาให้คำปรึกษาตรวจทานวิทยานิพนธ์ให้แก่ผู้วิจัย

คณะกรรมการสอบวิทยานิพนธ์ทุกท่าน ที่กรุณาช่วยตรวจและแก้ไขวิทยานิพนธ์

สำนักงานพัฒนาวิทยาศาสตร์และเทคโนโลยีแห่งชาติ ซึ่งได้สนับสนุนทุนสำหรับการศึกษาและทำวิจัย

บัณฑิตวิทยาลัย ที่ได้สนับสนุนทุนอุดหนุนการวิจัย

เจ้าหน้าที่ภาควิชาคณิตศาสตร์ทุกท่านที่ให้ความช่วยเหลือและอำนวยความสะดวกในการทำวิทยานิพนธ์

คุณพ่อ พี่ชาย พี่สาว และภรรยา ซึ่งได้สนับสนุนช่วยเหลือและให้กำลังใจในการทำวิทยานิพนธ์นี้มาโดยตลอด

อาหมาน หมัดเจริญ

สารบาญ

รายการภาพประกอบ

บทที่ 1

บทนำ

ปัจจุบันการติดต่อสื่อสารทางคอมพิวเตอร์เข้ามามีบทบาทในชีวิตประจำวันของมนุษย์มากยิ่งขึ้น โดยเฉพาะเมื่อมีการใช้เครือข่ายอินเทอร์เน็ตกันอย่างกว้างขวาง ในการติดต่อสื่อสารกันผ่านทางคอมพิวเตอร์นั้นย่อมมีความแตกต่างกันในด้านสถาปัตยกรรมทางฮาร์ดแวร์ที่ผู้ผลิตแต่ละรายผลิตคอมพิวเตอร์ของตนเองออกมาจำหน่าย ทำให้เครื่องคอมพิวเตอร์เหล่านั้นไม่สามารถติดต่อสื่อสารกันเข้าใจได้ จึงจำเป็นที่ต้องมีกฎเกณฑ์ร่วมกันเพื่อใช้ในการสื่อสารที่จะทำให้การสื่อสารตามกฎเกณฑ์นั้นสามารถเป็นที่เข้าใจกันระหว่างผู้ส่งกับผู้รับ กฎเกณฑ์ที่ว่านี้ก็คือโปรโตคอลนั่นเอง

โปรโตคอลที่ใช้สำหรับสื่อสารผ่านทางคอมพิวเตอร์มีมากมาย โดยที่การออกแบบโปรโตคอลสำหรับสื่อสารผ่านทางคอมพิวเตอร์นั้นผู้ออกแบบแต่ละรายก็ออกแบบขึ้นมาแล้วเขียนรายละเอียดขั้นตอนการทำงานของโปรโตคอลด้วยภาษาและสัญลักษณ์ที่เป็นที่เข้าใจของตนเอง ทำให้เป็นการยากในการที่ผู้อื่นจะนำโปรโตคอลนั้นไปพัฒนาและนำมาใช้งานจริง จึงได้มีการกำหนดมาตรฐานการเขียนข้อกำหนดโปรโตคอลขึ้นเรียกว่า FDT (Formal Description Technique) ซึ่ง Estelle (ISO 9074, 1989) ก็เป็นหนึ่งในวิธีการเขียนข้อกำหนดตามมาตรฐานนี้

การที่ Estelle เป็นภาษาที่นิยมใช้ในการเขียนข้อกำหนดโปรโตคอลนั้นเพราะมีผู้สนใจในภาษานี้จำนวนมากจึงมีผู้สร้างเครื่องมือเพื่อช่วยในการเขียน วิเคราะห์และทดสอบการทำงานของโปรโตคอลที่เขียนขึ้นด้วยภาษา Estelle ขึ้นมาซึ่งเครื่องมือดังกล่าวที่เป็นที่นิยมใช้กันอย่างกว้างขวางคือ EDT (Estelle Development Toolset) (Lai and Jichiefpattana, 1998 : 178) และ Estelle Graphical Editor โปรโตคอลต่าง ๆ ที่ใช้เครื่องมือเหล่านี้ในการเขียนข้อกำหนดและทดสอบการทำงานมาแล้วได้แก่โปรโตคอล FTAM, OSI-TP, CMIP, TP0, TP2 และ XTP (Institut National des Telecommunications, 1999 : 6) สำหรับ EDT นั้นมีมหาวิทยาลัยและศูนย์วิจัยต่าง ๆ มากกว่า 40 แห่งซื้อรุ่นสมบูรณ์ไปใช้ (Institut National des Telecommunications, 1999 : 6)

เครื่องมือ EDT เขียนขึ้นด้วยภาษาซี สามารถทำงานได้บนระบบปฏิบัติการยูนิกซ์บนเครื่อง SUN-Sparc (ภายใต้ระบบปฏิบัติการ SunOS 4.1.x และ Solaris 2.x) เครื่อง HP 9000/700 (ภายใต้ระบบปฏิบัติการ HP-UX 10.x) และบนเครื่องคอมพิวเตอร์ส่วนบุคคล (ภายใต้ระบบปฏิบัติการ Linux 2.x)

สำหรับงานวิจัยนี้เป็นการเขียนข้อกำหนดของโปรโตคอล TFTP ตามแบบภาษา Estelle โดยใช้เครื่องมือ Estelle Graphical Editor ช่วยในการเขียนซึ่งทำให้เห็นการทำงานของโปรโตคอลง่ายขึ้นเพราะเป็นการแสดงในแบบรูปภาพแทนที่จะเขียนด้วยหลักภาษาของ Estelle ที่อยู่ในรูปแบบอักขระซึ่งต้องจำรูปแบบคำสั่งและเขียนคำสั่งให้ถูกต้องทุกคำสั่งจึงอาจทำให้เสียนเวลาในการตรวจสอบความถูกต้องของคำสั่งนั้น แต่หากใช้เครื่องมือ Estelle Graphical Editor แล้วจะมีสัญลักษณ์ที่ใช้แทนคำสั่ง ต่าง ๆ ซึ่งจะถูกแปลเป็นคำสั่งของ Estelle ได้อย่างถูกต้องให้โดยอัตโนมัติจึงเป็นการประหยัดเวลาในส่วนของรูปแบบคำสั่งได้มาก หลังจากนั้นจึงแปลงข้อกำหนดจากรูปภาพไปเป็นภาษา Estelle อีกทีหนึ่ง เมื่อได้ข้อกำหนดโปรโตคอลในรูปแบบภาษา Estelle แล้วจึงใช้เครื่องมือ EDT ในการตรวจสอบความถูกต้องและจำลองการทำงานของโปรโตคอล TFTP โดยแสดงให้เห็นขั้นตอนการทำงานของโปรโตคอลในการแสดงพฤติกรรมต่าง ๆ เมื่อมีข่าวสารหรือเหตุการณ์ที่แตกต่างกันข้าวมา

1.1 วัตถุประสงค์

1. เพื่อศึกษาการทำงานของโปรโตคอล TFTP
2. เพื่อศึกษาการเขียนข้อกำหนดโปรโตคอลโดยใช้ภาษา Estelle
3. เพื่อศึกษาการติดตั้งและใช้งาน Estelle Graphical Editor ในการช่วยเขียนข้อกำหนดโปรโตคอลในรูปแบบภาษา Estelle
4. เพื่อศึกษาการติดตั้งและใช้งานเครื่องมือ EDT ในการตรวจสอบและจำลองการทำงานของโปรโตคอล TFTP

1.2 ขอบเขตการดำเนินงาน

1. เขียนข้อกำหนดโปรโตคอล TFTP ด้วยภาษา Estelle ได้อย่างถูกต้อง
2. ใช้เครื่องมือ Estelle Graphical Editor ในการเขียนโปรโตคอลได้
3. ตรวจสอบและจำลองการทำงานของโปรโตคอล TFTP ด้วยเครื่องมือ EDT

1.3 ขั้นตอนและระยะเวลาในการดำเนินงาน

1. ศึกษาขั้นตอนการทำงานของโปรโตคอล TFTP
2. ศึกษาการเขียนข้อกำหนดโปรโตคอลด้วยภาษา Estelle
3. ศึกษาการติดตั้งและใช้งานเครื่องมือ Estelle Graphical Editor บนระบบปฏิบัติการ Solaris 2.x
4. เขียนข้อกำหนดโปรโตคอล TFTP ตามแบบภาษา Estelle โดยใช้ Estelle Graphical Editor
5. ศึกษาการติดตั้งและใช้งานเครื่องมือ EDT บนระบบปฏิบัติการ Solaris 2.x
6. ตรวจสอบความถูกต้องของโปรโตคอล TFTP ด้วย EDT
7. จำลองการทำงานของโปรโตคอลด้วยเครื่องมือ EDT
8. จัดทำรายงานและคู่มือการใช้ระบบ

Gantt chart ของระยะเวลาการดำเนินงาน

ขั้นตอนการดำเนินงาน	ก.ค.	ส.ค.	ก.ย.	ต.ค.	พ.ย.	ธ.ค.	ม.ค.	ก.พ.	มี.ค.	เม.ย.	พ.ค.	มิ.ย.	ก.ค.	ส.ค.	ก.ย.
1	◄►														
2		◄	►												
3				◄►											
4					◄		►								
5							◄	►							
6								◄	►						
7										◄		►			
8													◄		►

1.4 ประโยชน์ที่คาดว่าจะได้รับ

1. ผู้วิจัยสามารถเข้าใจการทำงานของโปรโตคอล TFTP
2. ผู้วิจัยได้ทักษะและประสบการณ์ในการเขียนข้อกำหนดโปรโตคอลด้วยภาษา Estelle
3. ผู้วิจัยได้ทักษะในการใช้เครื่องมือ Estelle Graphical Editor ในการเขียนข้อกำหนดโปรโตคอล

4. ผู้วิจัยเข้าใจการติดตั้งและใช้งานเครื่องมือ EDT เพื่อช่วยตรวจสอบและจำลองการทำงานของโปรโตคอลที่เขียนด้วยภาษา Estelle

1.5 สถานที่และเครื่องมือที่ใช้

<u>สถานที่</u>

1. ห้องปฏิบัติการคอมพิวเตอร์ M105 ภาควิชาวิทยาการคอมพิวเตอร์ มหาวิทยาลัยสงขลานครินทร์ วิทยาเขตหาดใหญ่

<u>เครื่องมือที่ใช้</u>

1. เครื่องคอมพิวเตอร์ SUN SPARC station Model SparcStation 4 หน่วยความจำ 32 MB ฮาร์ดดิสก์ 1.05 GB ระบบปฏิบัติการยูนิกซ์ Solaris 2.5.1
2. โปรแกรม EDT รุ่น 4.2
3. ตัวแปรภาษาซี gcc รุ่น 2.7.1
4. โปรแกรม Estelle Graphical Editor รุ่น 2.1

บทที่ 2

โปรโตคอล TFTP

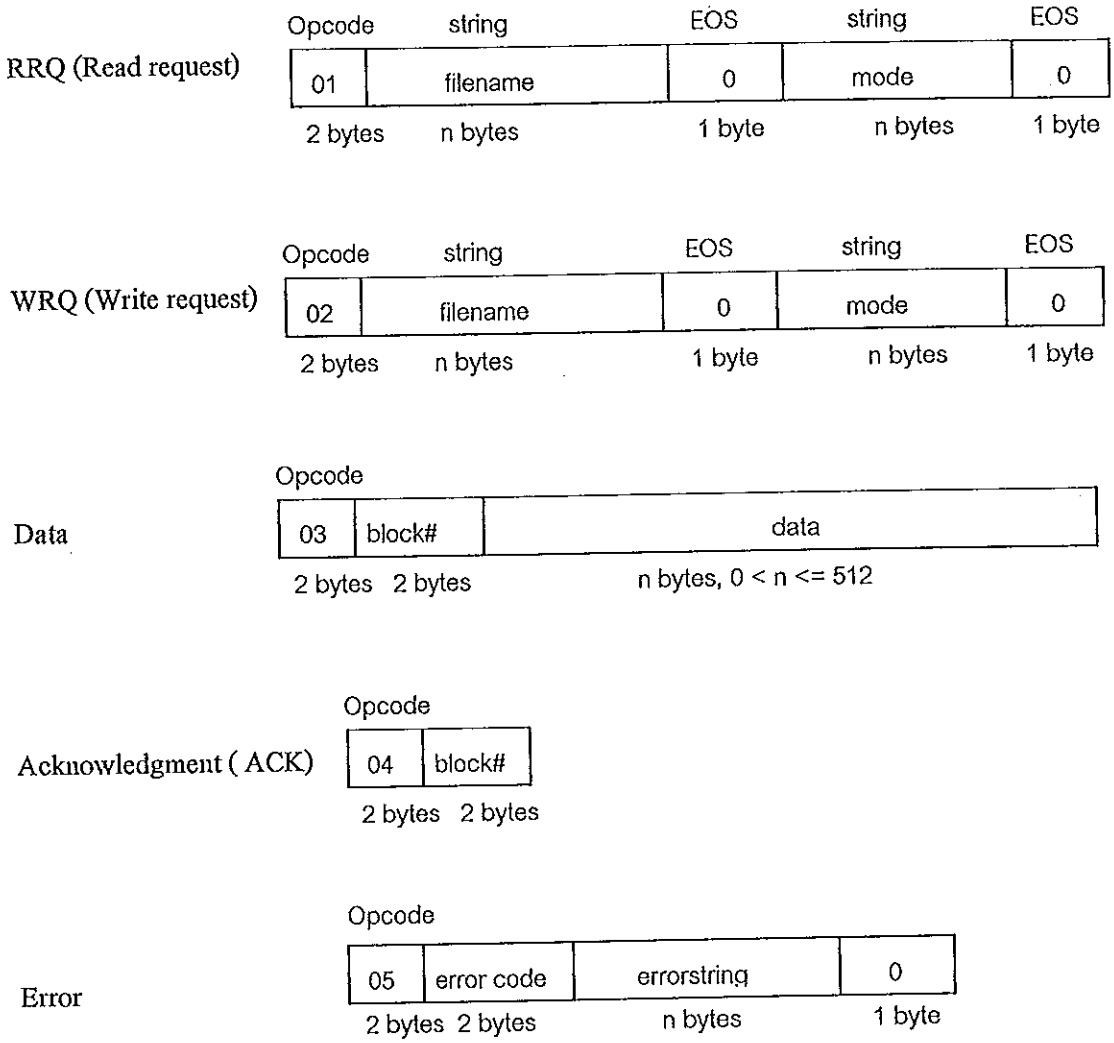
(Trivial File Transfer Protocol)

การส่งหรือถ่ายโอนแฟ้มระหว่างโหนดต่าง ๆ หรือระหว่างเครื่องคอมพิวเตอร์ใด ๆ ถือว่าเป็นส่วนสำคัญอย่างหนึ่งของระบบเครือข่าย แต่เนื่องจากในระบบเครือข่ายต่าง ๆ นั้นมีความแตกต่างกันในเรื่องของระบบเครื่องที่เชื่อมต่อด้วยกันหรือมีความแตกต่างกันในเรื่องของระบบปฏิบัติการที่ใช้อยู่บนเครื่องนั้น ๆ จึงจำเป็นต้องมีกฎเกณฑ์หรือข้อกำหนดเพื่อมาเป็นกรอบในการให้เครื่องต่าง ๆ ที่จะถ่ายโอนแฟ้มข้อมูลระหว่างกันนั้นใช้ร่วมกันอย่างสอดคล้องหรือที่เรียกว่าโปรโตคอลนั่นเอง

TFTP เป็นโปรโตคอลที่ใช้สำหรับถ่ายโอนแฟ้มระหว่างคอมพิวเตอร์สองระบบด้วยกัน โดยรูปแบบของโปรโตคอล TFTP นี้เป็นไปตามข้อกำหนดของ RFC 783 (Sollins, 1981) ซึ่งเป็นโปรโตคอลอย่างง่ายไม่มีความสลับซับซ้อนมากนัก ซึ่งแตกต่างจากโปรโตคอล FTP (File Transfer Protocol) ที่มีหน้าที่การทำงานที่อำนวยความสะดวกมากกว่าเช่น สามารถดูรายการในสาระบบย่อย ตรวจสอบสิทธิการเข้าใช้ระบบของผู้ใช้ (User authentication) เป็นต้น แต่สำหรับโปรโตคอล TFTP นั้นทำหน้าที่อย่างเดียวคือสามารถรับหรือส่งแฟ้มระหว่างผู้ร้องขอและผู้ให้บริการ และเนื่องจากโปรโตคอล TFTP มีขนาดไม่ใหญ่โตมากนักจึงสามารถทำหน้าที่เริ่มต้นทำงานเป็น Bootstrap ให้แก่เครื่องที่เชื่อมต่ออยู่บนเครือข่าย (Workstation) ต่างๆ เพื่อเชื่อมต่อเข้าสู่ระบบได้โดยสามารถติดตั้งลงในหน่วยความจำชนิดอ่านได้อย่างเดียว (Read Only Memory) ได้เลย

2.1 ชนิดของแพคเก็ต

ข้อมูลหรือแพคเก็ต (Packet) ที่จะใช้ในการทำงานข่องโปรโตคอล TFTP มีอยู่ 5 ชนิด แต่ละชนิดมีรูปแบบดังนี้



ภาพประกอบ 2-1 รูปแบบของแพคเก็ตต่าง ๆ ที่ใช้ในโปรโตคอล TFTP

(ที่มา: Stevens, W. Richard, 1991 : 466)

แพคเก็ตทุก ๆ แพคเก็ตจะมีสดมภ์แรกที่มีขนาดสองไบต์ทำหน้าที่เป็นตัวบอกรหัสชนิดของแพคเก็ตซึ่งเรียกว่าอ๊อปโค้ด (Opcode)

แพคเก็ต RRQ และ WRQ ฝ่ายผู้ขอบริการ (Client) จะส่งไปยังผู้ให้บริการ (Server) เพื่อขอรับแฟ้มจากผู้ให้บริการ (RRQ) หรือขอส่งแฟ้ม (WRQ) ไปยังผู้ให้บริการ โดยกำหนดชื่อแฟ้มและโหมด (mode) ของการรับส่ง

ทั้งชื่อแฟ้มและโหมด เป็นข้อมูลอักขระแอสกีที่ไม่จำกัดความยาวตายตัวแต่จะจบข้อความด้วยอักขระเลขศูณย์ซึ่งแสดงให้เห็นในที่นี้เป็น EOS (End of string) ค่าของโหมดเป็นไปได้สองอย่างคือสายอักขระ "netsacii" หรือไม่ก็สายอักขระ "octet"

แพคเก็ตที่เป็นข้อมูล (Data packet) จะประกอบไปด้วยข้อมูลจริง ๆ ที่ทำการรับหรือส่งพร้อมกับมีหมายเลขบล็อค (block#) กำกับไปด้วย ส่วนที่เป็นเนื้อหาของข้อมูลจะมีความยาวอยู่ระหว่าง 1 ถึง 512 ไบต์ ถ้าข้อมูลมีความยาวระหว่าง 0 ถึง 511 ไบต์แสดงว่าข้อมูลนั้นจะเป็นข้อมูลสุดท้าย ส่วนแพคเก็ตที่มีเนื้อหาข้อมูลความยาว 512 ไบต์แสดงว่ายังมีข้อมูลที่จะส่งอีกคือไม่ได้เป็นแพคเก็ตสุดท้าย สำหรับหมายเลขบลอ็คนั้นจะใช้สำหรับอีกฝ่ายหนึ่งในการแจ้งข่าวสาร (Acknowledgment) ว่าแพคเก็ตไหนเป็นเป็นแพคเก็ตที่ถูกต้องล่าสุดที่ได้รับมา

Error packet เป็นแพคเก็ตแสดงข้อผิดพลาดซึ่งจะส่งไปเมื่อมีข้อผิดพลาดเกิดขึ้นในการรับส่งข้อมูลซึ่งประกอบไปด้วยรหัสข้อผิดพลาด (Error code) และข้อความอธิบายข้อผิดพลาด (Error string) ที่อธิบายความหมายของรหัสข้อผิดพลาดนั้นซึ่งส่วนนี้จะมีหรือไม่มีก็ได้ โดย Error string นี้จะมีความยาวไม่แน่นอนและต้องจบด้วยไบต์ที่เป็นอักขระศูนย์ ภาพประกอบ 2.2 เป็นตารางแสดงรหัสข้อผิดพลาดและความหมาย

รหัสผิดพลาด (error code)	ความหมาย	หมายเหตุ
0	Not defined	รหัสนี้ยังไม่กำหนดความหมาย
1	File not found	ไม่เจอแฟ้มที่ต้องการ
2	Access violation	
3	Disk full or allocation exceeded	ที่เก็บข้อมูลเต็ม
4	Illegal TFTP operation	
5	Unknown port number	
6	File already exists	
7	No such user	

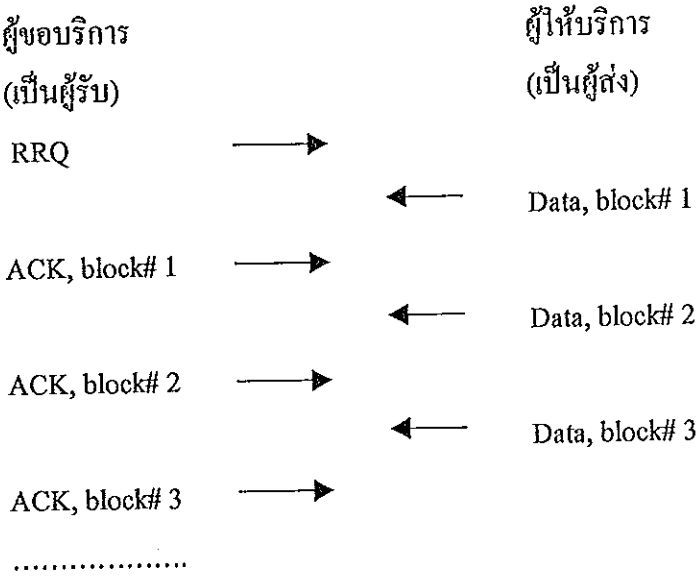
ภาพประกอบ 2-2 ตารางแสดงรหัสข้อผิดพลาดและความหมาย

(ที่มา: Stevens, W. Richard , 1991 : 467)

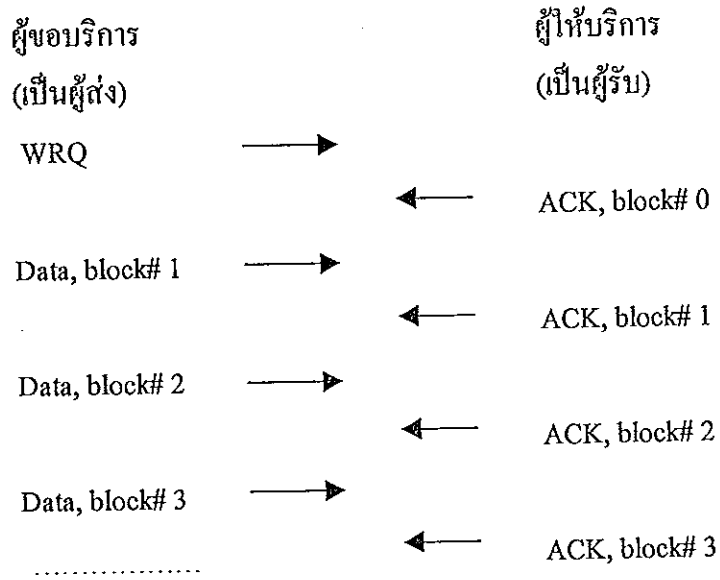
2.2 ลักษณะการทำงานของโปรโตคอล TFTP

ลักษณะการรับส่งข้อมูลระหว่างผู้ให้บริการกับผู้ขอบริการซึ่งมีอยู่สองแบบเป็นดังนี้

1. ผู้ขอบริการร้องขอแฟ้มข้อมูลจากผู้ให้บริการ



2. ผู้ขอบริการส่งแฟ้มข้อมูลไปให้ผู้ให้บริการ



(ที่มา: Stevens, W. Richard, 1991 : 467)

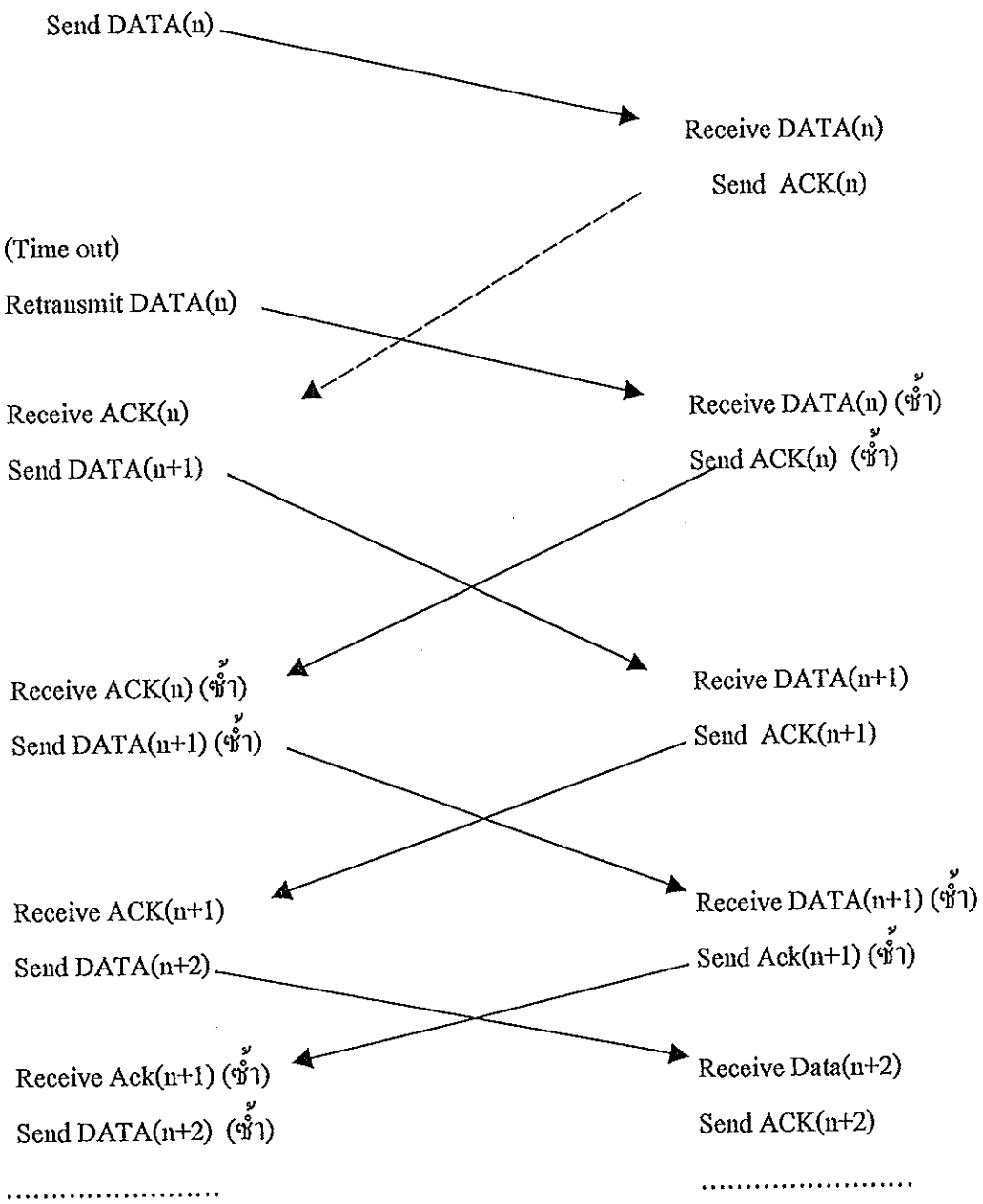
โปรโตคอล TFTP จะทำการควบคุมหรือติดตามดูแพคเก็ตที่สูญหายโดยกำหนดให้ฝ่ายที่ทำหน้าที่ส่งแพคเก็ตข้อมูล (Data packet) ตั้งช่วงเวลา (Time out) เวลาสำหรับจะส่งข้อมูลซ้ำอีกครั้งหนึ่ง โดยหากแพคเก็ตข้อมูลเกิดสูญหายฝ่ายส่งจะรู้ได้เมื่อครบเวลาที่ตั้งไว้แล้วแต่ยังไม่ได้รับแพคเก็ตแจ้งข่าวสาร (Acknowledge packet) กลับมาจึงทำการส่งข้อมูลครั้งล่าสุดไปอีกครั้งหนึ่ง จะสังเกตได้ว่าทั้งผู้ขอบริการและผู้ให้บริการสามารถเป็นผู้ส่งแพคเก็ตข้อมูลได้ทั้งสองฝ่ายขึ้นอยู่กับว่าผู้ขอบริการจะส่งคำร้องขอรับแฟ้มหรือส่งคำร้องขอส่งแฟ้ม แต่หากแพคเก็ตแจ้งข่าวสารเกิดสูญหาย ฝ่ายที่ทำการส่งแพคเก็ตข้อมูลก็จะเกิดการหมดเวลาที่ตั้งไว้เช่นกันและจะทำการส่งแพคเก็ตข้อมูลครั้งล่าสุดใหม่อีกครั้งหนึ่ง ซึ่งในกรณีนี้ฝ่ายที่รับแพคเก็ตข้อมูลจะทราบจากหมายเลขบล็อคข้อมูลว่าเป็นข้อมูลซ้ำ จึงปล่อยข้อมูลนั้นทิ้งไปและทำการส่งแพคเก็ตแสดงข่าวสารเดิมไปใหม่อีกครั้งหนึ่ง จะเห็นได้ว่าหมายเลขบล็อคในแพคเก็ตข้อมูลหรือแพคเก็ตแสดงข่าวสารของโปรโตคอล TFTP นี้จะทำหน้าที่ตรวจสอบการสูญหายหรือความซ้ำซ้อนของแพคเก็ต

ข้อผิดพลาดอื่น ๆ ที่เกิดขึ้นนอกจากเรื่องหมดเวลา (Time out) แล้วจะทำให้การทำงานสิ้นสุดลง เมื่อฝ่ายหนึ่งส่งแพคเก็ตข้อผิดพลาดอีกฝ่ายหนึ่งจะไม่ส่งแพคเก็ตแสดงข่าวสารกลับไปให้และฝ่ายส่งแพคเก็ตข้อผิดพลาดนั้นก็จะไม่ส่งแพคเก็ตข้อผิดพลาดซ้ำใหม่อีกครั้งหนึ่งเช่นกัน

ข้อผิดพลาดส่วนใหญ่ที่เกิดขึ้นมักจะได้แก่ไม่มีแฟ้มข้อมูลในกรณีของการร้องขอรับแฟ้มหรือไม่มีสิทธิ์ในการเขียนแฟ้มในกรณีของการร้องขอส่งแฟ้ม

เมื่อได้มีการร้องขอรับแฟ้มหรือร้องขอส่งแฟ้มจากผู้ขอบริการเกิดขึ้นแล้ว การทำงานส่วนที่เหลือของโปรโตคอลจะทำงานในลักษณะเดียวกันของทั้งผู้ขอบริการและผู้ให้บริการเพราะทั้งสองฝ่ายสามารถเป็นได้ทั้งผู้ส่งแพคเก็ตข้อมูล แพคเก็ตแสดงข่าวสารและแพคเก็ตข้อผิดพลาดเหมือนกันนั่นเอง

อย่างไรก็ตามก็ยังมีปัญหาเล็กน้อยที่อาจเกิดขึ้นได้จากข้อกำหนดที่กล่าวมาแล้ว คือหากทั้งผู้ส่งและผู้รับใช้การตั้งเวลาและการส่งข้อมูลซ้ำอีกครั้งเมื่อหมดเวลาที่ตั้งไว้โดยส่งข้อมูลครั้งสุดท้ายที่ได้ส่งไปอีกนั้นจะเกิดปัญหาที่มีชื่อว่า "sorcerer's apprentice syndrome" ภาพประกอบ 2.3 แสดงลักษณะการสรับส่งข้อมูลที่เกิดปัญหา sorcerer's apprentice syndrome



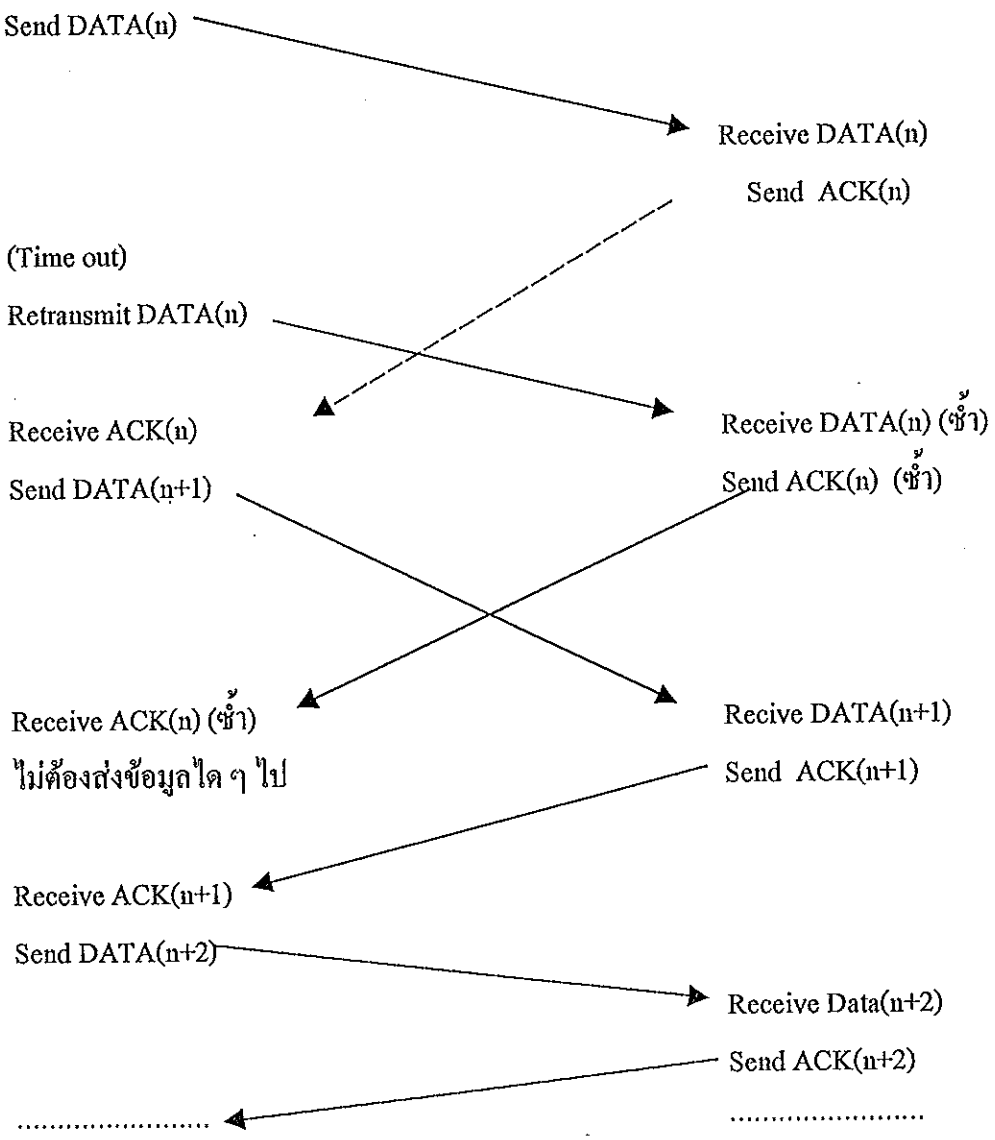
ภาพประกอบ 2-3 แสดงลักษณะการเกิดปัญหา sorcerer's apprentice syndrome

(ที่มา: Stevens, W. Richard, 1991 : 469)

จากภาพประกอบ 2.3 จะเห็นว่าหาก ACK(n) ที่ส่งโดยฝ่ายผู้รับแพคเก็ตข้อมูลเกิดเสียเวลาหรือขัดข้องอย่างใดอย่างหนึ่งในเครือข่ายซึ่งแสดงโดยใช้เส้นประตามรูปนั้น ผลที่เกิดขึ้นคือ ทุก ๆ แพคเก็ตข้อมูลและแพคเก็ตแสดงข่าวสารตั้งแต่จุดที่เริ่มเกิดปัญหาขึ้นจะถูกส่งซ้ำสองครั้งไป

เรื่อย ๆ โดยที่การรับส่งแฟ้มนั้นก็ยังคงดำเนินไปได้โดยไม่มีปัญหาแต่จะเสียเวลาในการทำงานของระบบเท่านั้น

การแก้ไขปัญหานี้ทำได้โดยฝ่ายที่ส่งแพคเก็ตข้อมูลไม่ต้องส่งข้อมูลซ้ำไปใหม่หากได้รับแพคเก็ตแสดงข่าวสารซ้ำของครั้งก่อนที่เพิ่งได้รับมาล่าสุด ซึ่งก็จะได้ลักษณะการทำงานที่ไม่เกิดปัญหา sorcerer's apprentice syndrome ดังแสดงในภาพประกอบ 2.4



ภาพประกอบ 2-4 แสดงลักษณะการขนส่งข้อมูลเมื่อแก้ไขปัญหา sorcerer's apprentice syndrome

(ที่มา: Stevens, W. Richard, 1991 : 470)

ผลที่ได้จากการแก้ปัญหา sorcerer's apprentice syndrome ก็คือฝ่ายที่รับแพคเก็ตข้อมูลไม่จำเป็นต้องมีการตั้งเวลา (Time out) ซึ่งจะทำให้การนำโปรโตคอล TFTP ไปเขียนใช้งานง่ายขึ้น ถ้านำไปใช้เป็นโปรแกรม bootstrap

บทที่ 3

Formal Description Technique: Estelle

การสื่อสารข้อมูลผ่านทางระบบคอมพิวเตอร์ต่าง ๆ ที่เชื่อมต่อเข้าด้วยกันและเป็นการเชื่อมต่อแบบระบบเปิดที่ไม่จำกัด หรือเจาะจงว่าต้องเป็นระบบที่มาจากแหล่งผู้ผลิตรายเดียวกันหรือมีสถาปัตยกรรมเดียวกันนั้นจะมีประสิทธิภาพได้ก็จะต้องมีกระบวนการหรือกฎในการสื่อสารข้อมูลที่เป็นมาตรฐานเดียวกันเพื่อไม่ให้เกิดความสับสน ดังนั้นการกำหนดกฎเกณฑ์ต่าง ๆ สำหรับการสื่อสารผ่านทางคอมพิวเตอร์หรือที่เรียกว่าโปรโตคอลนั้นจึงเป็นสิ่งจำเป็นอย่างยิ่งในสภาพแวดล้อมการทำงานของระบบเครือข่าย การกำหนดโปรโตคอลจึงต้องเกี่ยวข้องกับการเขียนบรรยายลักษณะการทำงานของแต่ละฝ่ายที่ทำการติดต่อกันให้ชัดเจน ไม่คลุมเครือและเป็นที่เข้าใจตรงกันไม่ว่าผู้ใดจะอ่าน ซึ่งการเขียนบรรยายคุณลักณะหรือการทำงานของโปรโตคอลนั้นหากต่างคนต่างเขียนด้วยภาษาของตนเองจะทำให้ผู้อื่นที่ต้องการนำโปรโตคอลนั้นไปสร้างเป็นโปรแกรมสำหรับใช้งานจริงก็อาจเกิดความสับสนได้เพราะหากมีผู้นำโปรโตคอลนั้นไปสร้างขึ้นมาใช้งานโดยตีความหมายการทำงานคุณลักษณะของโปรโตคอลที่เขียนขึ้นนั้นแตกต่างกันจะทำให้โปรแกรมต่าง ๆ ที่ใช้โปรโตคอลนั้นที่เขียนขึ้นจากผู้เขียนคนละรายกันไม่สามารถทำงานได้สอดคล้องกันหรืออาจไม่สามารถทำงานร่วมกันได้เลย ดังนั้นจึงต้องมีภาษาใดภาษาหนึ่งที่ใช้สำหรับบรรยายการทำงานหรือคุณลักษณะของโปรโตคอลที่เมื่อผู้ใดมาอ่านแล้วสามารถเข้าใจได้ตรงกัน ไม่คลุมเครือหรือไม่ทำให้สามารถแปลความหมายได้หลายนัย สำหรับภาษาพูดของแต่ละชาติที่ใช้กันอยู่ในปัจจุบันนั้นอาจไม่เหมาะสมในการที่จะนำมาเขียนข้อกำหนดของโปรโตคอลที่ประกอบไปด้วยการอธิบายพฤติกรรม การบริการ หน้าที่การทำงานย่อย ข้อมูลที่จะแลกเปลี่ยนกันในแต่ละหน่วยการทำงานของระบบ เพราะอาจทำให้ผู้ที่นำเอาข้อกำหนดนั้นไปสร้างโปรแกรมใช้งานจริงเข้าใจคลาดเคลื่อนไม่ตรงกับจุดประสงค์ที่ผู้เขียนข้อกำหนดตั้งไว้เพราะความเข้าใจในภาษาที่เขียนนั้นไม่ดีหรืออาจเข้าใจในรายละเอียดปลีกย่อยผิดความหมายไป ดังนั้นหากมีภาษาที่ตกลงกันสำหรับเขียนข้อกำหนดโปรโตคอลให้เป็นเอกลักษณ์เฉพาะ เป็นภาษาที่ทุกคนเข้าใจตรงกัน ไม่มีลักษณะที่คลุมเครือหรือตีความหมายได้หลายนัย ก็จะทำให้การเขียนข้อกำหนดของโปรโตคอลและนำข้อกำหนดนั้นไปเขียนโปรแกรมใช้งานได้สอดคล้องกันและทำงานร่วมกันได้โดยไม่มีปัญหาถึงแม้ว่าผู้เขียนโปรแกรมขึ้นใช้งานจะเป็นคนละคนกันก็ตามเพราะต่างก็ใช้ข้อกำหนดเดียวกันและเข้าใจในข้อกำหนดนั้นตรงกันนั่นเอง ภาษาในลักษณะดังกล่าวเรียกว่า Formal Description Language ซึ่งก็ได้มีผู้พยามยามคิด

ค้น Formal Description Language ขึ้นมาเพื่อเสนอให้เป็นที่ยอมรับใช้กันมากมาย ภาษาที่เป็น Formal Description นั้นไม่จำเป็นต้องเป็นภาษาที่อยู่ในรูปข้อความ (Textual Language) แต่อาจอยู่ในรูปสัญลักษณ์ต่าง ๆ ก็ได้ โดยที่แต่ละประโยคหรือสัญลักษณ์ในข้อกำหนดต้องมีกฎเฉพาะหรือเรียกว่ามี Formal Syntaxs ซึ่งแต่ละ Syntax ต้องสามารถมีความหมายที่แน่นอนได้อย่างเดียวหรือเรียกว่ามี Formal Semantics

จุดประสงค์หลักของ Formal Description Language คือต้องการมีข้อกำหนดของโปรโตคอลที่ไม่คลุมเครือทำให้ผู้ออกแบบระบบ ผู้นำข้อกำหนดไม่เขียนใช้งานและผู้ใช้งานที่เกี่ยวข้องเข้าใจตรงกันทั้งหมดซึ่งทำให้การวิเคราะห์และตรวจสอบหาข้อผิดพลาดในการทำงานของโปรโตคอลทำได้ง่ายขึ้นตั้งแต่ในขั้นตอนการออกแบบ ทำให้ไม่เสียเวลาต้องมาแก้ไขหลังจากได้นำข้อกำหนดนั้นไปใช้งานจริงแล้ว

3.1 Formal Description Techniques

ในช่วงกลางปี ค.ศ. 1980 ISO ได้ตั้งกลุ่มขึ้นมากลุ่มหนึ่งเพื่อพัฒนา Formal Description Language เพื่อให้เหมาะสมกับ OSI และเรียกภาษานั้นว่า Formal Description Technique (FDTs) โดย FDT ที่ได้พัฒนาขึ้นมามีสองภาษาที่ได้รับเป็นมาตรฐานโดย ISO คือ Estelle กับ Lotos แต่ที่ได้รับความนิยมและมีเครื่องมือมาสนับสนุนมากกว่าก็คือ Estelle โดยมีผู้พัฒนาเครื่องมือขึ้นมาสำหรับตรวจสอบการทำงานของโปรโตคอลที่เขียนขึ้นด้วย Estelle ชื่อว่า EDT (Estelle Development Toolsets) และก็ได้มีผู้คิดโปรแกรมที่ใช้ในการออกแบบโปรโตคอลตามหลักภาษาของ Estelle ให้อยู่ในรูปแบบเชิงสัญลักษณ์เพื่อให้ง่ายแก่การมองภาพโดยรวมของโปรโตคอลขึ้นมาด้วยเช่นกันคือ Estelle Graphical Editor

3.2 Estelle

Estelle (ISO 9074, 1989) เป็น Formal Description Technique อย่างหนึ่งที่มีจุดประสงค์เพื่อเสนอวิธีในการขจัดความคลุมเครือของโปรโตคอลต่าง ๆ ซึ่งโดยทั่วไปแล้วจะเขียนขึ้นโดยใช้การผสมผสานกันระหว่างภาษาธรรมชาติ ตารางแสดงสถานะและอื่น ๆ จึงทำให้เกิดความเข้าใจที่คลุมเครือสำหรับผู้ที่จะนำข้อกำหนดของโปรโตคอลนั้นไปใช้งาน

โดยสรุปแล้ว Estelle ถือเป็นส่วนขยายของภาษาปาสคาลระดับศูนย์ (ISO 7185,1983) สำหรับใช้กำหนดการทำงานของระบบในรูปของโครงสร้างลำดับชั้นของหน่วยสื่อสารข้อมูลต่าง ๆ ที่แต่ละหน่วยสามารถ

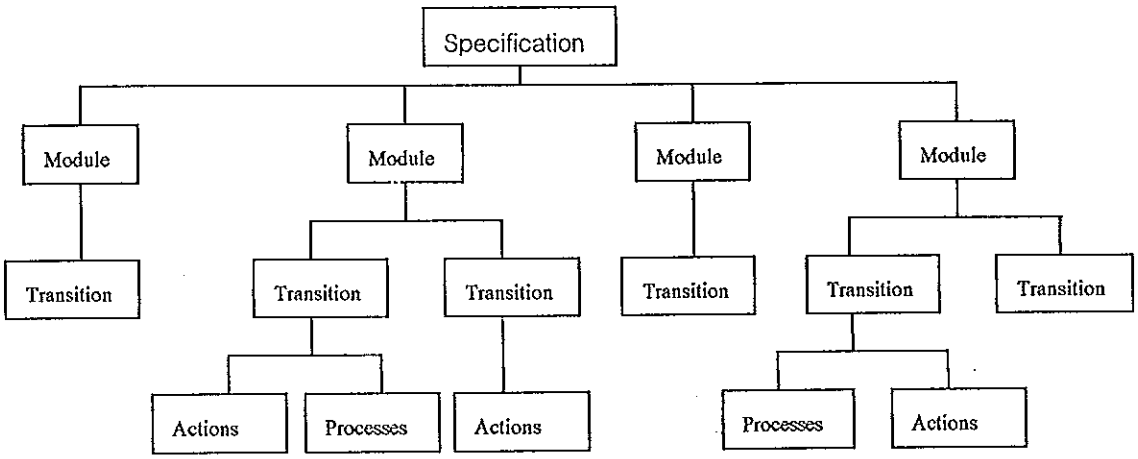
- ทำงานแบบขนาน
- ทำการสื่อสารกันโดยการแลกเปลี่ยนข่าวสารซึ่งกันและกันและโดยการใช้ตัวแปรร่วมกัน

Estelle ใช้วิธีการอธิบายการติดต่อสื่อสารระหว่างส่วนประกอบหน่วยต่าง ๆ ของระบบแยกต่างหากกับการอธิบายพฤติกรรมภายในของแต่ละส่วนประกอบแต่ละหน่วย แต่ละวัตถุหรือส่วนประกอบหน่วยย่อยต่าง ๆ ใน Estelle ต้องมีการประกาศเป็นชนิด (Type) ใด ๆ เหมือนการประกาศชนิดของตัวแปรในภาษาปาสคาล

3.2.1 Estelle Model

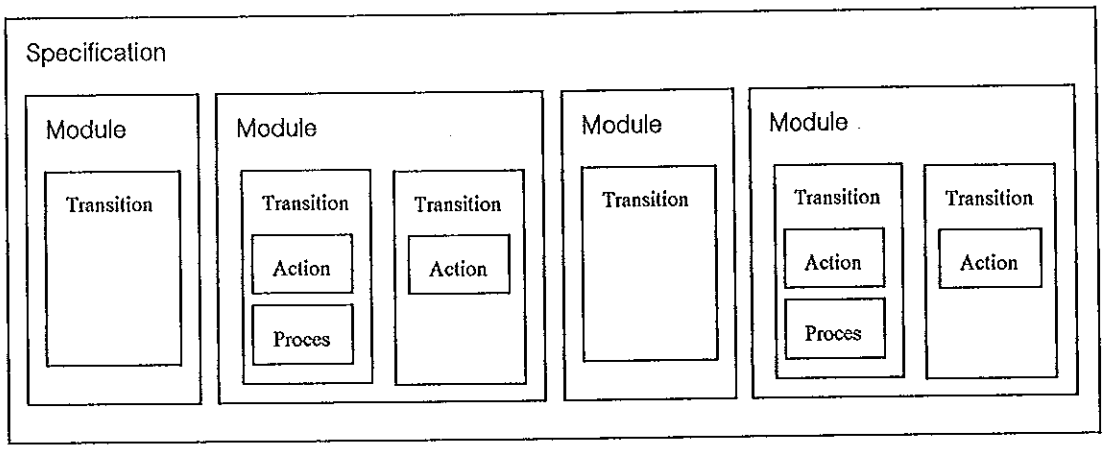
ระบบหรือข้อกำหนดใด ๆ ที่เขียนขึ้นในรูปแบบของ Estelle จะมองได้ว่าระบบนั้นประกอบด้วยกลุ่มของส่วนประกอบหน่วยย่อยที่ทำหน้าที่สื่อสารข้อมูลระหว่างกันเรียกว่าโมดูล (Module) ซึ่งประกอบด้วยตัวแปรที่เกี่ยวข้องในการทำงาน การเปลี่ยนสถานะ (Transition) และอาจมีโมดูลย่อยเป็นลำดับชั้นถัดไปได้อีกเรื่อย ๆ

โครงสร้างการกำหนดโปรโตคอลตามรูปแบบของ Estelle นั้นประกอบไปด้วยส่วนหลัก ๆ 4 ระดับชั้นคือ ระดับ Specification ระดับ Module ระดับ Transition และระดับ actions และ Processes ดังแสดงในภาพประกอบ 3-1 และภาพประกอบ 3-2



ภาพประกอบ 3-1 ส่วนประกอบหลัก 4 ระดับชั้นของตามรูปแบบของ Estelle ในรูปโครงสร้างต้นไม้

(ที่มา: Lai and Jirachiefpattana, 1998 : 40)



ภาพประกอบ 3-2 ส่วนประกอบหลัก 4 ระดับชั้นของตามรูปแบบของ Estelle ในรูปแผนภาพ

(ที่มา: Lai and Jirachiefpattana, 1998 : 40)

3.2.2 โมดูลและตัวแทนของโมดูล (Modules และ Module instances)

ข้อกำหนดโปรโตคอลโดยใช้ Estelle จะเป็นการอธิบายส่วนประกอบหน่วยย่อยต่าง ๆ เรียกว่าโมดูล ที่จะทำหน้าที่สื่อสารข้อมูลกัน โดยที่แต่ละโมดูลที่ทำงานอยู่ในขณะใดขณะหนึ่งเรียกว่าเป็นตัวแทนของโมดูล (Module instance) แต่ในที่นี้จะใช้คำว่าโมดูลในการแทนความหมายของตัวแทนของโมดูลนอกจากในกรณีที่จะเน้นความแตกต่างระหว่างทั้งสองคำนี้เท่านั้น

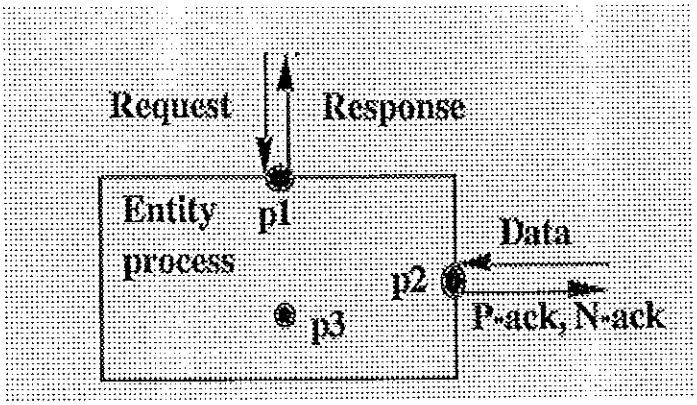
พฤติกรรมและโครงสร้างภายในของโมดูลจะกำหนดโดยใช้ชุดของการเปลี่ยนสถานะการทำงาน (Transitions) ของโมดูลและโดยการนิยามของโมดูลย่อย (Child module) ภายในโมดูลนั้น พร้อมด้วยการอธิบายการติดต่อระหว่างกันของโมดูลต่าง ๆ

โมดูลใด ๆ จะถือว่าเป็นโมดูลที่ทำงาน (Active) ก็ต่อเมื่อมีการกำหนดการเปลี่ยนสถานะการทำงานหรือมีส่วนของ Transition อยู่ในโมดูลนั้นมิเช่นนั้นจะถือว่าโมดูลนั้นเป็นโมดูลที่ไม่ทำงาน (Inactive)

ในส่วนการติดต่อสื่อสารกันระหว่างโมดูลจะมีการกำหนดหรืออธิบายส่วนที่ใช้เป็นทางผ่านหรือที่เรียกว่าอินเตอเฟส (Interface) ของโมดูลซึ่งมีส่วนประกอบสามส่วนดังนี้

- จุดสื่อสาร (Interaction point)
- ช่องทางสื่อสาร (Channel)
- สารหรือข่าวสาร (Interactions)

โมดูลหนึ่ง ๆ อาจมีจุดสื่อสารสำหรับรับหรือส่งข่าวสารไปให้โมดูลอื่น ๆ ได้หลายจุด โดยจุดสื่อสารเหล่านี้มีสองชนิดคือจุดสื่อสารภายนอก (External interaction point) และจุดสื่อสารภายใน (Internal interaction point) โดยแต่ละจุดสื่อสารจะต้องสัมพันธ์กับช่องทางสื่อสารใดช่องทางหนึ่งเสมอซึ่งช่องทางสื่อสารจะทำให้เกิดกลุ่มของข่าวสารสองกลุ่มคือข่าวสารที่ส่งและข่าวสารที่รับผ่านทางจุดสื่อสาร โดยจุดสื่อสารภายนอกใช้สำหรับแลกเปลี่ยนข่าวสารระหว่างโมดูลกับสภาพแวดล้อมภายนอกหรือโมโดูลอื่น ๆ นั่นเองแต่จุดสื่อสารภายในใช้สำหรับแลกเปลี่ยนข่าวสารไปยังโมดูลย่อยหรือโมดูลลูกของโมดูลนั้น ๆ ภาพประกอบ 2-3 แสดงส่วนติดต่อสื่อสารของโมดูล



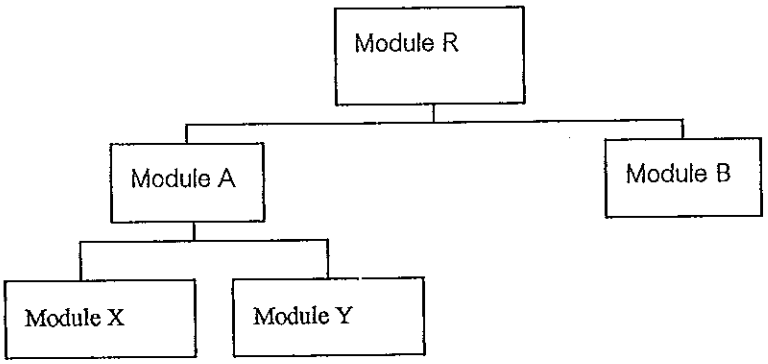
ภาพประกอบ 3-3 ส่วนติดต่อสื่อสารของโมดูล Entity

(ที่มา: ISO 9074, 1989 : 2)

จากภาพประกอบ 3-3 จุดที่อยู่บนขอบของสี่เหลี่ยมที่แทนโมดูลจะหมายถึงจุดสื่อสารภายนอกส่วนจุดสื่อสารภายในจะอยู่ภายในกรอบสี่เหลี่ยมของโมดูล ซึ่งจากรูปมีข่าวสารที่โมดูลนี้รับและส่งคือ Request และ Data เป็นข่าวสารที่รับเข้ามาส่วน Response P-ack และ N-ack เป็นข่าวสารที่ส่งออกไป

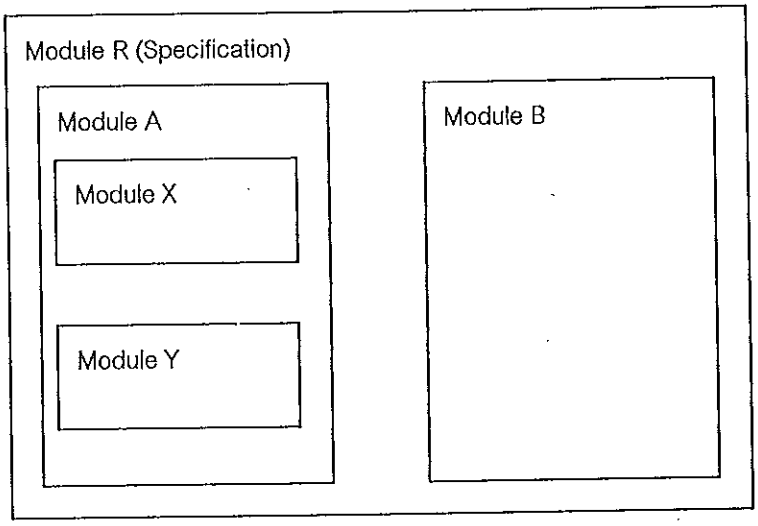
3.2.3 โครงสร้างของโมดูล

การนิยามของโมดูลตามแบบของ Estelle นั้นอาจมีโมดูลย่อยภายในโมดูลเป็นลำดับชั้นลงไปได้เรื่อย ๆ ซึ่งทำให้เกิดโครงสร้างแบบลำดับชั้นของโมดูลขึ้นดังแสดงเป็นแผนภาพโครงสร้างต้นไม้และแผนภาพโครงสร้างแบบสี่เหลี่ยมดังภาพประกอบ 3-4 และภาพประกอบ 3-5 ตามลำดับ



ภาพประกอบ 3-4 แผนภาพโครงสร้างต้นไม้ลำดับชั้นของโมดูล

(ที่มา: ISO 9074, 1989 : 3)



ภาพประกอบ 3-5 แผนภาพโครงสร้างแบบสี่เหลี่ยมลำดับชั้นของโมดูล

(ที่มา: ISO 9074, 1989 : 3)

จากรูปจะใช้สี่เหลี่ยมผืนผ้าแทนโมดูลต่าง ๆ โดยความสัมพันธ์แบบพ่อกับลูกของโมดูลจะใช้เส้นหรือการซ้อนกันของรูปสี่เหลี่ยมซึ่งจากภาพที่ 3.4 และ 3.5 จะได้ว่าโมดูล A และ โมดูล X มีความสัมพันธ์แบบพ่อกับลูกกันโดยโมดูล X เป็นลูกของโมดูล A ส่วนโมดูลที่อยู่ระดับบนสุดหรือโมดูลรากถือเป็นโมดูลหลักหรือเป็นตัว Specification ของโปรโตคอลนั่นเอง โมดูลลูกที่มีโม

ดูลแม่เดียวกันจะเรียกว่า Sibling ซึ่งจากภาพประกอบ 3-4 และ 3-5 จะได้ว่าโมดูล X กับ Y หรือ A กับ B เป็น Sibling กัน

3.2.4 การสื่อสารระหว่างโมดูล

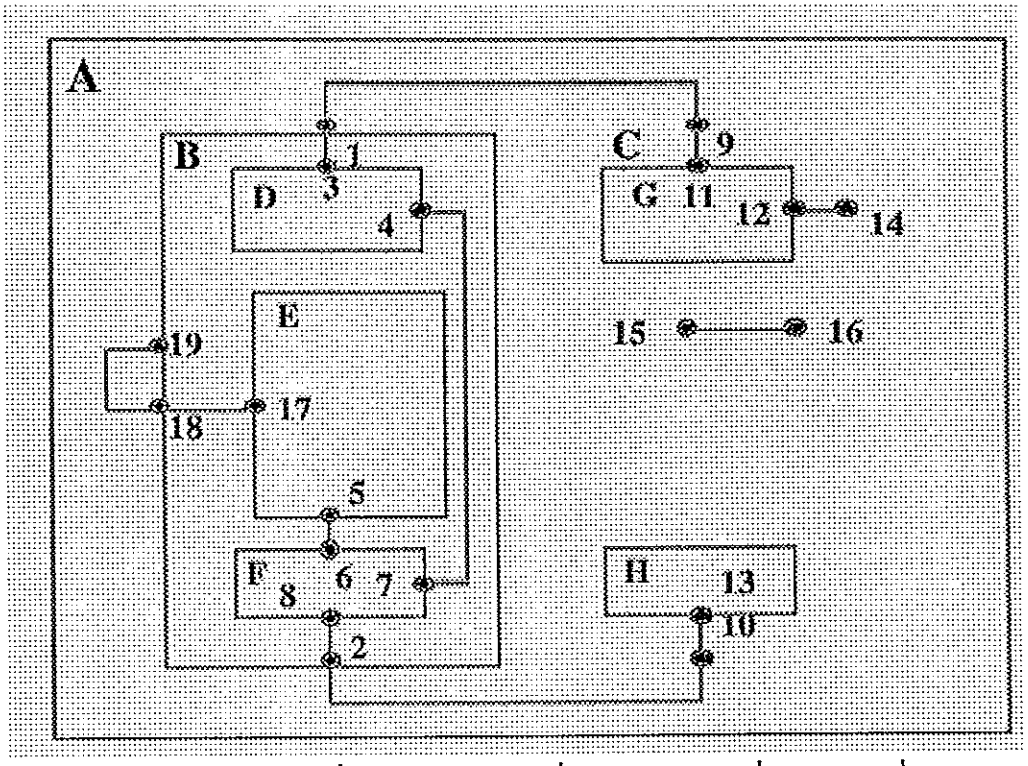
ณ ขณะใด ๆ โมดูลที่อยู่ในลำดับชั้นต่าง ๆ สามารถติดต่อสื่อสารกันได้ 2 วิธีคือ

1. การแลกเปลี่ยนข่าวสาร (Message exchange)
2. ใช้ตัวแปรร่วมกัน

3.2.4.1 การแลกเปลี่ยนข่าวสาร

ข่าวสารหรือข้อความที่โมดูลต่าง ๆ ส่งหรือรับระหว่างกันนั้นเรียกว่า Interactions ซึ่งจะส่งผ่านทางเส้นทางที่เชื่อต่อระหว่างจุดสื่อสารสองจุด ข่าวสารที่โมดูลใด ๆ รับมาผ่านทางจุดสื่อสารจะถูกนำต่อท้ายคิวที่ไม่มีขอบเขตจำกัดและเป็นคิวชนิดมาก่อนได้ก่อน (First in First out หรือ FIFO) โดยที่คิวชนิด FIFO นี้หากเป็นของจุดสื่อสารเดียวก็จะเรียกว่าคิวส่วนตัว (Individual queue) แต่หากเป็นคิวที่จุดสื่อสารอื่น ๆ สามารถมาใช้งานร่วมด้วยก็จะเรียกว่าคิวร่วม (Common queue)

ในการจะกำหนดว่าโมดูลใดสามารถแลกเปลี่ยนข่าวสารได้ต้องทำการเชื่อมเส้นทางสื่อสารระหว่างจุดสื่อสารต่าง ๆ ด้วยคำสั่ง Connect และคำสั่ง Attach ก่อน เส้นทางสื่อสารที่เชื่อมระหว่างจุดสื่อสารสองจุดใด ๆ อาจประกอบไปด้วยเส้นทางย่อยที่ใช้คำสั่ง Connect หนึ่งเส้นทางและเส้นทางย่อยที่ใช้คำสั่ง Attach ศูณย์เส้นทางหรือมากกว่า ภาพประกอบ 3-6 แสดงเส้นทางสื่อสารและเส้นทางย่อยต่าง ๆ ที่ประกอบกันขึ้นเป็นเส้นทางสื่อสารหนึ่งเส้นทาง



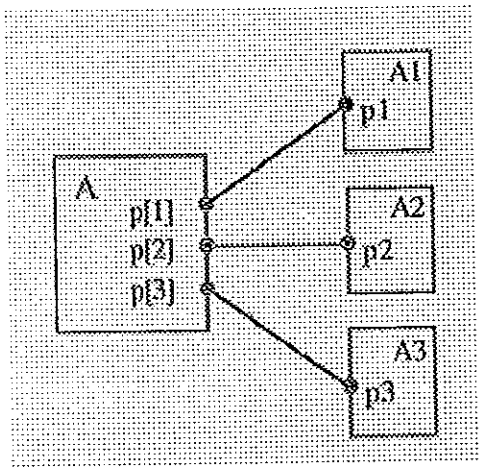
ภาพประกอบ 3-6 เส้นทางสื่อสาระหว่างโมดูลที่ใช้ในการแลกเปลี่ยนข่าวสารซึ่งกันและกัน

(ที่มา: ISO 9074, 1989 : 4)

เมื่อจุดสื่อสารภายนอกของโมดูลใดถูกเชื่อมโยงกับจุดสื่อสารภายนอกของโมดูลที่เป็นโมดูลแม่จะเรียกว่าจุดสื่อสารเหล่านั้นถูกปะติดกัน (Attached) ซึ่งจากภาพประกอบ 2-6 จะได้ว่าคู่ของจุดสื่อสารที่ถูกปะติดกันคือ (1,3) (2,8) (9,11) และ (17,18) แต่ถ้าจุดสื่อสารสองจุดที่เชื่อมโยงกันเป็นจุดสื่อสารภายนอกของโมดูลที่เป็น Sibling กัน เช่นจุด (1,9) (2,10) (4,7) และ (5,6) หรือจุดหนึ่งเป็นจุดสื่อสารภายในส่วนอีกจุดหนึ่งเป็นจุดสื่อสารภายนอกของโมดูลลูกเช่น (12,14) หรือทั้งคู่เป็นจุดสื่อสารภายในหรือจุดสื่อสารภายนอกของโมดูลเดียวกันอย่างเช่น (15,16) และ (18,19) ก็จะเรียกว่าจุดสื่อสารเหล่านี้ถูกเชื่อมต่อกัน (Connected)

เส้นทางสื่อสารจะเป็นตัวกำหนดว่าสองโมดูลที่มีจุดสื่อสารเป็นจุดปลายของเส้นทางสื่อสารสามารถสื่อสารกันได้โดยการแลกเปลี่ยนข่าวสารกันซึ่งกันและกันผ่านทางจุดสื่อสารนั้นทั้งสองทิศทางอย่างเช่น จุดสื่อสาร 3 กับ 11 หรือ จุดสื่อสาร 8 กับ 13 ต่างก็เป็นจุดปลายของเส้นทางสื่อสารระหว่างโมดูล D กับ G และโมดูล F กับ H ตามลำดับ โดยที่จุดสื่อสาร 1 9 2 และ 10 ไม่ใช่จุดปลาย หากโมดูลใดส่งข้อมูลหรือข่าวสารผ่านทางจุดสื่อสารที่เป็นจุดปลายของเส้นทางสื่อสารข้อมูลหรือข่าวสารนั้นก็จะถูกส่งตรงไปยังจุดปลายอีกจุดหนึ่งและถูกเก็บไว้ในคิวชนิด FIFO ของ

โมดูลที่รับข้อมูล หากโมดูลส่งข้อมูลผ่านทางจุดสื่อสารที่ไม่ใช่จุดปลายของเส้นทางสื่อสารก็จะถือว่าข้อมูลนั้นถูกยกเลิกหรือถูกทิ้งไปโดยปริยาย ดังนั้นการสื่อสารระหว่างโมดูลจึงเป็นแบบจุดต่อจุด (End to end communication) เท่านั้น นอกจากนี้ยังมีการสื่อสารระหว่างโมดูลที่เรียกว่าการสื่อสารแบบกระจายข่าว (Multicast communication) โดยวิธีการส่งข้อมูลผ่านทางเส้นทางสื่อสารพร้อมกันหลาย ๆ เส้นทางจากโมดูลหนึ่งผ่านทางจุดสื่อสารหลาย ๆ จุดไปยังจุดสื่อสารของโมดูลที่ต้องการจะส่ง ดังแสดงในภาพประกอบ 3-7



ภาพประกอบ 3-7 การส่งข้อมูลแบบกระจายข่าว

(ที่มา: ISO 9074, 1989 : 6)

จากรูปโมดูล A ส่งข้อมูลพร้อมกันแบบกระจายข่าวผ่านจุดสื่อสาร p[1] p[2] และ p[3] ไปยังโมดูล A1 A2 และ A3 จะสังเกตได้ว่าสามารถประกาศจุดสื่อสารให้เป็นแบบแถวคอย (Array) ใน Estelle ได้

3.2.4.2 การใช้ตัวแปรร่วมกัน

การสื่อสารกันระหว่างโมดูลนอกจากใช้วิธีแลกเปลี่ยนข่าวสารผ่านทางจุดสื่อสารแล้วใน Estelle ยังสามารถใช้วิธีการใช้ตัวแปรร่วมกันอีกแต่เป็นวิธีที่ไม่ยืดหยุ่นเหมือนวิธีแรก โดยสามารถใช้ตัวแปรร่วมกันระหว่างโมดูลแม่กับโมดูลลูกเท่านั้น โดยที่ตัวแปรเหล่านั้นต้องมีการประกาศเป็นแบบ exported โดยโมดูลลูก แต่ทั้งสองโมดูลไม่สามารถเข้าถึงตัวแปรพร้อมกันได้เพราะลำดับความสำคัญของโมดูลแม่จะมีมากกว่าในการทำงาน

3.2.5 คุณลักษณะชั้นของโมดูล (Class attribute)

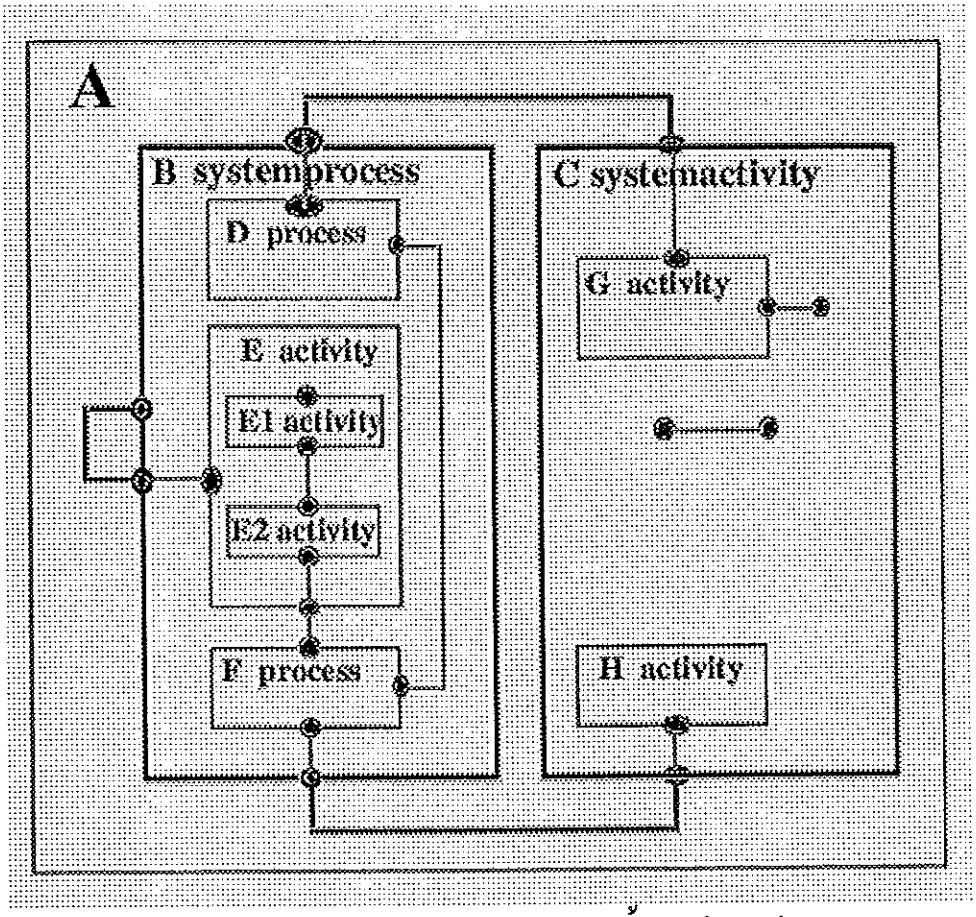
การที่โมดูลจะมีพฤติกรรมเกี่ยวข้องกับโมดูลอื่น ๆ เป็นอย่างไรนั้นขึ้นอยู่กับคุณลักษณะของโมดูลนั้นอยู่ในชั้นใด โมดูลอาจมีคุณลักษณะชั้นหนึ่งใน 4 ลักษณะดังนี้คือ

- systemprocess
- systemactivity
- process
- activity

หรืออาจไม่มีคุณลักษณะชั้นเลยก็ได้และจะเรียกโมดูลประเภทนี้ว่าโมดูลขาดคุณลักษณะชั้น (Non attribute module) ทุกตัวแทนของโมดูลจะมีคุณลักษณะชั้นตามที่ประกาศไว้ในส่วนหัวของโมดูล โมดูลที่มีคุณลักษณะชั้นเป็น systemprocess หรือ systemactivity จะเรียกว่าโมดูลระบบ (System module) การกำหนดคุณลักษณะชั้นของโมดูลจะเป็นไปตามหลักดังต่อไปนี้

- โมดูลปฏิบัติงาน (Active module) ทุก ๆ โมดูลสามารถมีคุณลักษณะชั้น (ไม่เป็นโมดูลที่ขาดคุณลักษณะชั้น)
- โมดูลระบบไม่สามารถเป็นโมดูลย่อยของโมดูลที่มีคุณลักษณะชั้นได้
- โมดูลที่มีคุณลักษณะชั้น process หรือ activity สามารถเป็นโมดูลระดับล่างของโมดูลระบบได้
- โมดูลที่มีคุณลักษณะชั้น process หรือ systemprocess สามารถมีโมดูลย่อยเป็นโมดูลที่มีคุณลักษณะชั้นเป็น process หรือ activity ได้เท่านั้น
- โมดูลที่มีคุณลักษณะชั้น activity หรือ systemactivity สามารถมีโมดูลย่อยเป็นโมดูลที่มีคุณลักษณะชั้นเป็น activity ได้เท่านั้น

คุณลักษณะชั้น systemprocess และ systemactivity ทำหน้าที่เป็นตัวบอกว่าโมดูลนั้นเป็นโมดูลที่ทำหน้าที่แยกต่างหากจากโมดูลอื่นในข้อกำหนดโปรโตคอล ภาพประกอบ 3-8 แสดงคุณลักษณะชั้นของโมดูลต่าง ๆ ตามหลักการกำหนดคุณลักษณะชั้นของโมดูล



ภาพประกอบ 3-8 คุณลักษณะชั้นของโมดูลต่าง ๆ

(ที่มา: ISO 9074, 1989 : 7)

3.2.6 โครงสร้างและหลักภาษาของ Estelle

3.2.6.1 ช่องทางสื่อสาร ข่าวสารและจุดสื่อสาร

ช่องทางสื่อสาร (Channel) ใน Estelle จะทำหน้าที่กำหนดกลุ่มของข่าวสาร (Interactions) การประกาศจุดสื่อสาร (Interaction point) จะอ้างถึงช่องทางสื่อสารด้วยวิธีการเฉพาะ โดยที่การอ้างถึงช่องทางสื่อสารนั้นจะเป็นการกำหนดว่าจุดสื่อสารใด ๆ จะมีข่าวสารอะไรบ้างที่จะส่งออกหรือรับเข้ามาผ่านทางจุดสื่อสารนั้น ตัวอย่างการกำหนดช่องทางสื่อสารใน Estelle เป็นดังนี้

Channel Channel_Id (Role1,Role2);

By Role1: m1;
m2;
...
mK;

Bye Role2: q1;
q2;
...
qM;

By Role1,Role2 r1;
r2;
...
rN;

โดยที่ m1,....mK, q1,...qM, r1,.....rN เป็นการประกาศ interaction หรือข่าวสารที่ต้องการสื่อสารผ่านทางช่องทางการสื่อสารนั้น โดยที่การประกาศข่าวสารประกอบไปด้วยชื่อข่าวสารและอาจมีพารามิเตอร์เพิ่มเติมก็ได้ อย่างเช่น

REQUEST(a:integer;b:boolean)

เป็นการประกาศชนิดข่าวสารชื่อ REQUEST ซึ่งภายหลังสามารถมีข่าวสารชนิดเดียวกันนี้โดยผ่านค่าพารามิเตอร์แทน x และ y อย่างเช่น

REQUEST(1,true) และ REQUEST(5,false)

ต่างก็เป็นข่าวสารชนิดเดียวกัน
ดังนั้นการประกาศจุดสื่อสาร p1 จึงสามารถประกาศได้ดังนี้

p1 : Channel_Id(Role1)

และจุดสื่อสาร p2 ก็สามารถประกาศได้ดังนี้

p2 : Channel_Id(Role2)

ในกรณีแรกกลุ่มของข่าวสารที่สามารถส่งผ่านจุดสื่อสาร p1 ได้นั้นต้องเป็นกลุ่มของข่าวสารที่ประกาศอยู่หลัง "By Role1" และหลัง "Bye Role1,Role2" ในส่วนประกาศของช่องสื่อสารเท่านั้น อย่างเช่นข่าวสาร m1,m2,..,mK และ r1,r2,..,rN ส่วนข่าวสารที่จุดสื่อสาร p1 สามารถรับได้ก็คือข่าวสารที่ประกาศอยู่หลัง "By Role2" และ "By Role1,Role2" ในส่วนประกาศของช่องสื่อสาร จะเรียกจุดสื่อสาร p1 และ p2 ว่ามีบทบาทตรงกันข้าม หากต้องการจำกัดว่าคิวชนิด FIFO ของจุดสื่อสาร p1 จะใช้งานร่วมกับจุดสื่อสารอื่นได้หรือไม่ได้ก็เขียนได้เป็น

p1 : Channel_id(Role1) comm queue หรือ

p1 : Channel_Id(Role1) individual queue

ในกรณีแรกจะได้ว่าคิวชนิด FIFO สามารถมีจุดสื่อสารอื่นที่ประกาศในโมดูลเดียวกันสามารถมาใช้ร่วมด้วยได้

ความแตกต่างระหว่างจุดสื่อสารภายนอกกับจุดสื่อสารภายในอยู่ที่ตำแหน่งในการประกาศ หากประกาศจุดสื่อสารไว้ภายในส่วนหัวของโมดูลก็จะเป็นจุดสื่อสารภายนอกแต่หากประกาศไว้ในส่วนรายละเอียดของโมดูลก็จะเป็นจุดสื่อสารภายใน

3.2.6.2 Modules

การประกาศหรือนิยามโมดูลใน Estelle ประกอบด้วย 2 ส่วนคือ

- การนิยามส่วนหัวของโมดูล (Module header definition)
- การประกาศส่วนเนื้อหาของโมดูล (Module body definition)

ส่วนหัวของโมดูลจะทำหน้าที่กำหนดชนิดของโมดูลโดยกำหนดเป็นคุณลักษณะชั้นพร้อมด้วยจุดสื่อสารและตัวแปรร่วมต่าง ๆ ที่จะใช้ในโมดูล โดยการประกาศส่วนหัวของโมดูลจะเริ่มต้นด้วยคำว่า module ตามด้วยชื่อของโมดูล ส่วนคุณลักษณะชั้นของโมดูลถ้ามีก็จะตามหลังชื่อโมดูลอีกทีหนึ่งและหากมีการรับค่าพารามิเตอร์ก็ให้ประกาศตัวแปรที่จะรับค่าพารามิเตอร์ไว้ในวงเล็บ หลังจากนั้นก็จะเป็นการประกาศจุดสื่อสารและตัวแปรร่วมต่าง ๆ ที่โมดูลจะใช้ในการสื่อสาร

กับโมดูลอื่น ๆ การประกาศจุดสื่อสารจะตามหลังคำสงวน "ip" ส่วนตัวแปรร่วมจะตามหลังคำสงวน "export" และจบการประกาศส่วนหัวของโมดูลด้วย "end" ตัวอย่างการประกาศส่วนหัวของโมดูลหนึ่งเป็นดังนี้

```
module A systemactivity (R:boolean);
        ip   p:  T(S) individual queue;
             p2:  W(K) common queue;
             p3:  U(S) common queue;
        export X,Y : integer; Z:boolean
End;
```

เมื่อประกาศส่วนหัวของโมดูลเรียบร้อยแล้วก็สามารถประกาศส่วนเนื้อหาของโมดูลได้ โดยปกติแล้วจะมีการประกาศส่วนเนื้อหาหนึ่งส่วนต่อหนึ่งส่วนหัวของโมดูล แต่ก็สามารถมีส่วนเนื้อหาได้หลายครั้งสำหรับหนึ่งส่วนหัวของโมดูลหากต้องการให้โมดูลมีพฤติกรรมที่แตกต่างกัน การประกาศส่วนเนื้อหาของโมดูลจะเริ่มต้นด้วยคำสงวน "body" ตามด้วยชื่อส่วนเนื้อหา (Body name) และจบด้วยคำสงวน "end" ภายในส่วนเนื้อหาก่อนคำสงวน "end" ก็จะเป็นการบรรยายรายละเอียดหรือพฤติกรรมของโมดูล ตัวอย่างการประกาศส่วนเนื้อหาของโมดูลของโมดูล A ที่ผ่านมาเป็นดังนี้

```
body B for A;
{ ส่วนรายละเอียดภายใน }
end;
Body C for A;
{ส่วนรายละเอียดภายใน}
end;
```

ส่วนรายละเอียดภายในของส่วนเนื้อหาของโมดูลก็จะมีรายละเอียดแยกย่อยเป็นส่วน ๆ ดังนี้

- ส่วนประกาศตัวแปรและชนิดวัตถุ (Declaration part)
- ส่วนกำหนดค่าเริ่มต้น (Initialisation part)
- ส่วนกำหนดการเปลี่ยนสถานะ (Transition part)

- ส่วนประกาศตัวแปรและชนิดวัตถุ

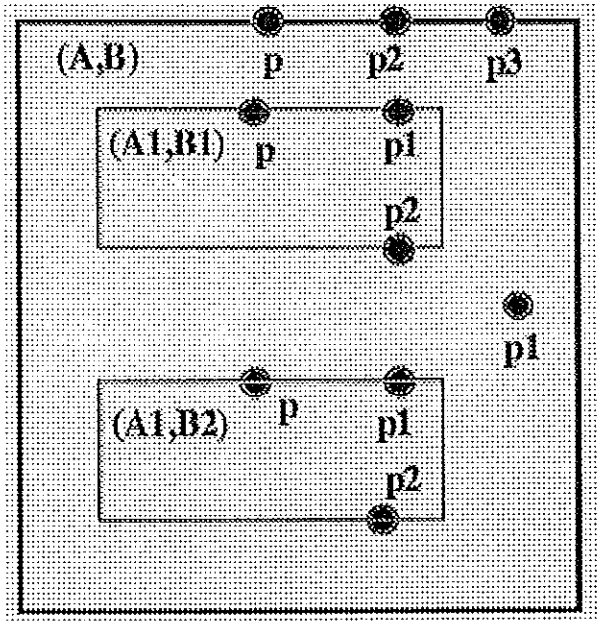
  - ช่องสื่อสาร
  - ส่วนหัวและรายละเอียดของโมดูล
  - ตัวแปรชนิดโมดูล
  - สถานะและชุดของสถานะ
  - จุดสื่อสารภายใน

ส่วนรายละเอียดภายในของส่วนเนื้อหาสามารถมีส่วนหัวและส่วนเนื้อหาของโมดูลอื่นอยู่ภายในได้ทำให้เกิดลำดับชั้นของโมดูลไปเรื่อย ๆ ตัวอย่างต่อไปนี้เป็นการประกาศส่วนเนื้อหาของส่วนหัวโมดูล A ที่ผ่านมาโดยมีชื่อส่วนเนื้อหา (Body name) เป็น B และมีโมดูลย่อยภายในคือ A1 เป็นโมดูลลูก ในโมดูลลูก A1 นี้จะมีส่วนเนื้อหาสองชื่อคือ B1 กับ B2 ในที่นี้จะใช้สัญลักษณ์ (A,B) เป็นตัวแทนของโมดูลที่มีส่วนเนื้อหาชื่อ B ของส่วนหัว A

```
module A (ดูหัวข้อโมดูลที่ผ่านมา) ... end;
body B for A;
        ip p1: T1(R2) common queue;  {internal ip}
        module A1 activity (K:integer);
                ip p1: T1(R1) individual queue;
                   p2: T1(R2) individual queue;
                   p:   T(S) individual queue;
```

```
        end;
        body B1 for A1; {ส่วนรายละเอียดโมดูล} end;
        body B2 for A1; {ส่วนรายละเอียดโมดูล} end;
end;
```

ก็จะได้ลำดับชั้นของโมดูล A ที่มีส่วนเนื้อหา B และโมดูลย่อยภายในดังภาพประกอบ 3-9



ภาพประกอบ 3-9 ส่วนหัวและเนื้อหาของโมดูลและโมดูลย่อย

(ที่มา: ISO 9074, 1989 : 13)

เมื่อมีการประกาศโมดูลครบสมบูรณ์แล้วก็สามารถประกาศตัวแปรขึ้นมาทำงานเป็นตัวแทนของโมดูล การประกาศตัวแปรชนิดโมดูลใช้คำสงวน modvar ในการประกาศเช่น

```
modvar  X,Y,Z  : A1
```

จะหมายถึงตัวแปร X,Y และ Z เป็นตัวแปรชนิดโมดูล A1 การประกาศตัวแปรเป็นชนิดโมดูลจะใช้ส่วนหัวของโมดูลในการอ้างถึงชนิดของโมดูลที่ต้องการ ตัวแทนของโมดูลสามารถสร้างและถูก

ทำลายได้โดยใช้คำสั่ง init release และ terminate ตามด้วยตัวแปรชนิดโมดูลได้ดังจะได้กล่าวในหัวข้อ คำสั่ง Estelle

พฤติกรรมภายในของโมดูลใด ๆ กำหนดได้โดยการแสดงการเปลี่ยนสถานะของโมดูล ซึ่งสถานะต่าง ๆ ของโมดูลที่จะมีได้นั้นเรียกว่าสถานะควบคุม (Control state) โดยสามารถประกาศสถานะควบคุมต่าง ๆ ของโมดูลได้โดยใช้คำว่า state เช่น

state IDLE,WAIT,OPEN,CLOSED

เป็นการประกาศสถานะควบคุมสี่สถานะให้แก่โมดูลซึ่งภายหลังสามารถอ้างถึงสถานะต่างทั้งสี่ได้โดยใช้ชื่อ IDLE WAIT OPEN และ CLOSED หรือจะใช้ตัวเลข 0 1 2 และ 3 แทนได้ตามลำดับ นอกจากนี้ยังสามารถอ้างถึงสถานะหลาย ๆ สถานะภายใต้ชื่อเดียวกันเพื่อความสะดวกโดยใช้คำสงวน stateset เช่น

stateset NOT_IDLE = [BUSY,WAITING]

เป็นการกำหนดชื่อ NOT_IDLE สำหรับอ้างถึงสถานะ BUSY และ WAITING พร้อม ๆ กัน

- ส่วนกำหนดค่าเริ่มต้น (Initialisation part)

ส่วนกำหนดค่าเริ่มต้นของเนื้อหาโมดูลเป็นการำหนดค่าเริ่มต้นแก่ตัวแปร สถานะเริ่มต้นของโมดูล เริ่มเข้าสู่ส่วนของการกำหนดค่าเริ่มต้นด้วยคำสงวน initialse และตามด้วยการกำหนดค่าเริ่มต้นต่าง ๆ โดยหากเป็นการกำหนดค่าเริ่มต้นแก่ตัวแปรต่าง ๆ ก็ใช้หลักภาษาเหมือนภาษาปาสคาลแต่หากเป็นการกำหนดค่าเริ่มต้นให้แก่สถานะควบคุมก็จะใช้คำ to นำหน้าสถานะ เช่น

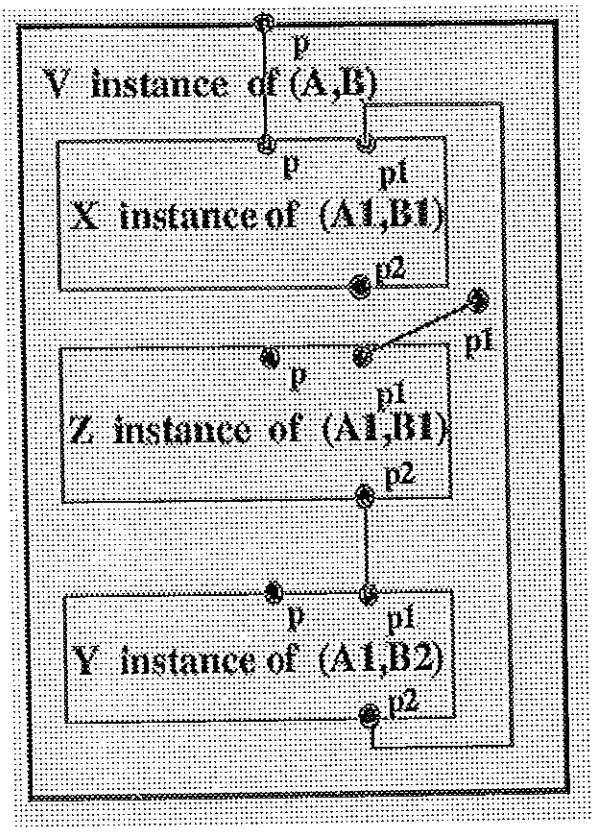
to IDLE

ส่วนการกำหนดค่าเริ่มต้นให้แก่ตัวแปรชนิดโมดูลก็จะเป็นการสร้างตัวแทนของโมดูลชนิดนั้นขึ้นมา ซึ่งการกำหนดค่าเริ่มต้นตัวแปรชนิดโมดูลจะใช้คำสั่ง init นอกจากนี้ยังมีการสร้างเส้นทางสื่อสาร

เชื่อมระหว่างจุดสื่อสารต่าง ๆ ด้วยคำสั่ง connect และ attach อยู่ในส่วนกำหนดค่าเริ่มต้นนี้ด้วย ตัวอย่างส่วนกำหนดค่าเริ่มต้นของโมดูล (A,B) เป็นดังนี้

```
initialise
    to IDLE
        begin
            T:=5
            Init X with B1(0);
            Init Y with B2(1);
            Init Z with B1(4);
            Connect p1 to Z.p1;
            Connect X.p1 to Y.p2;
            Connect Y.p1 to Z.p2;
            Attach p to X.p
        End;
```

ส่วนกำหนดค่าเริ่มต้นตัวอย่างนี้จะเป็นการสร้างตัวแทนโมดูลสามโมดูลซึ่งอ้างถึงได้ด้วยตัวแปร X Y และ Z ตามลำดับ ทุกโมดูลนี้มีคุณลักษณะภายนอกเหมือนกันเพราะเป็นโมดูลประเภท A1 โดยที่โมดูล X กับ Z จะเป็นตัวแทนโมดูลชนิดเดียวกันคือ (A1,B1) ส่วนโมดูลที่อ้างถึงด้วยตัวแปร Y เป็นโมดูล (A1,B2) โดยโมดูล X Y และ Z มีค่าเริ่มต้นที่ส่งผ่านพารามิเตอร์แตกต่างกัน เมื่อมีส่วนกำหนดค่าเริ่มต้นเหมือนตัวอย่างก็จะได้ลักษณะลำดับชั้นของโมดูลดังภาพประกอบ 3-10



ภาพประกอบ 3-10 ลักษณะลำดับชั้นของโมดูลที่ได้หลังประกาศส่วนกำหนดค่าเริ่มต้น

(ที่มา: ISO 9074, 1989 : 16)

- ส่วนกำหนดการเปลี่ยนสถานะ (Transition part)

ส่วนกำหนดการเปลี่ยนสถานะของโมดูลเป็นการอธิบายพฤติกรรมภายในของโมดูล พฤติกรรมของโมดูลจะเกี่ยวกับการเปลี่ยนสถานะจากสถานะควบคุมหนึ่งไปยังอีกสถานะหนึ่ง ส่วนกำหนดการเปลี่ยนสถานะประกอบด้วยชุดของการประกาศการเปลี่ยนสถานะ (Transition declaration) ซึ่งการประกาศการเปลี่ยนสถานะจะเริ่มด้วยคำสงวน trans โดยสามารถเขียนการประกาศการเปลี่ยนสถานะได้สองแบบคือแบบขยายกับแบบรวมซ้อนกัน การประกาศการเปลี่ยนสถานะแบบขยายประกอบด้วยสองส่วนคือ

- ส่วนของ clause-group
- ส่วนของ transition-block

ในแต่ละ clause-group จะใช้คำต่อไปนี้ในการกำหนดเงื่อนไขการเปลี่ยนสถานะของโมดูล

- from-clause (from A1,..,An โดยที่ Ai (I >= 1) เป็นสถานะควบคุมหรือชุดของสถานะควบคุมของโมดูล
- when-clause (when p.m โดยที่ p เป็นจุดสื่อสารและ m เป็นข่าวสารหรือ interaction)
- provided-clause (provided B โดยที่ B เป็นนิพจน์ตรรกศาสตร์ที่ให้ค่าความจริงเป็นจริงหรือเท็จ)
- priority-clause (priority n โดยที่ n เป็นค่าคงที่ที่เป็นจำนวนเต็มบวก)
- delay-clause (delay(e1,e2) โดยที่ e1 และ e2 เป็นนิพจน์ที่ให้ค่าเป็นจำนวนเต็มบวก)

การประกาศการเปลี่ยนสถานะที่มี when-clause จะไม่มี delay-clause หรือในทางกลับกันหากมี delay-clause ก็จะไม่มี when-clause เช่นกัน การเปลี่ยนสถานะที่มี when-clause จะเรียกว่าเป็นการเปลี่ยนสถานนำเข้า (input trainsition)

เมื่อเงื่อนไขในการเปลี่ยนสถานะเป็นจริงโมดูลจะเปลี่ยนสถานะจากเดิมไปเป็นสถานะใหม่หรือจะแสดงพฤติกรรมอย่างไรนั้นขึ้นอยู่กับการกำหนด to-clause หรือ transition-block โดยที่ to-clause กับ transition-block โดยการกำหนด to-clause จะเป็นการกำหนดว่าสถานะถัดไปที่โมดูลจะเปลี่ยนไปเป็นอย่างไรอย่างเช่น to WAIT เป็นกำหนดให้โมดูลเปลี่ยนจากสถานะปัจจุบันที่เป็นอยู่เข้าสู่สถานะ WAIT ส่วน transition-block เป็นการกำหนดชุดของคำสั่งตามแบบภาษาปาสคาลหรือคำสั่งเพิ่มเติมที่มีใน Estelle โดยชุดคำสั่งเหล่านั้นต้องอยู่ระหว่างคำสงวน begin และ end ตัวอย่างส่วนประกาศการเปลี่ยนสถานะที่มีการประกาศการเปลี่ยนสถานะแบบขยายเป็นดังนี้

```
trans
    from IDLE
        to IDLE
            priority medium
                when N.DATA_INDICATION
```

```
                name t1: begin
                            output U.DATA_INDICATION;
                            ak_no := ak_no+1
                         end;
trans
   from IDLE
      to AK_SENT
         provided (ak_no > 0) and (ak_no <=4)
            priority low
               delay(min,max)
                  name t2:    begin
                                 output N.SEND_AK(ak_no)
                              end;
trans
   from IDLE
      to AK_SENT
         provided (ak_no > 4) and (ak_no < 7)
            priority high
               delay(min)

                  name t3:    begin
                                 output N.SEND_AK(ak_no)
                              end;
trans
   from IDLE
      to AK_SENT
         provided ak_no = 7
            priority high
               name t4:    begin
```

```
                output N.SEND_AK(ak_no)
            end;
trans
    from IDLE
        to AK_SENT
            provided ak_no = 0
                priority low
                    delay(inactive_period)
                        name t5:    begin
                                        output N.SEND_AK(ak_no)
                                    end;

trans
    from AK_SENT
        to IDLE
            name t6:    begin
                            ak_no := 0
                        end;
```

หากเปลี่ยนเป็นการเขียนการประกาศแบบรวมซ้อนก็จะได้ดังนี้

```
trans
    from IDLE
        to IDLE
            priority medium
                when N.DATA_INDICATION
                    name t1: begin
                                output U.DATA_INDICATION;
                                ak_no := ak_no+1
```

```
                    end;
to AK_SENT
    provided (ak_no > 0) and (ak_no <=4)
        priority low
            delay(min,max)
                name t2:    begin
                                output N.SEND_AK(ak_no)
                            end;
    provided (ak_no > 4) and (ak_no < 7)
        priority high
            delay(min)
                name t3:    begin
                                output N.SEND_AK(ak_no)
                            end;
to AK_SENT
    provided ak_no = 7
        priority high
            name t4:    begin
                            output N.SEND_AK(ak_no)
                        end;
to AK_SENT
    provided ak_no = 0
        priority low
            delay(inactive_period)
                name t5:    begin
                                output N.SEND_AK(ak_no)
                            end;
```

```
to IDLE
    name t6:    begin
                    ak_no := 0
                end;
```

นอกจาก clause-group ทั้งหมดที่กล่าวมาแล้วยังมี Any-clause เพื่อช่วยในการเขียนการเปลี่ยนสถานะให้ดูกระชับหรือสั้นกว่าเดิมซึ่งมีรูปแบบการใช้ดังนี้

```
any domain do
```

โดยที่ domain เป็นตัวแปรท้องถิ่นที่กำหนดค่าอยู่ในช่วงจำกัดใด ๆ ดังตัวอย่างการเขียนการประกาศการเปลี่ยนสถานะแบบที่ใช้ any-clause กับแบบไม่ใช้ any-clause ดังนี้

```
trans
    from S1 to S2
        any n: 1..2; k: 3..4 do
            when p[n].m
                begin
                    Variable:= k
                end;
```

เมื่อเขียนแบบไม่ใช้ any-clause จะได้ดังนี้

```
trans
    from S1 to S2
        when p[1].m
            begin
                Variable:= 3
```

```
        end;
    when p[1].m
        begin
            Variable: =4
    when p[2].m
        begin
            Variable:= 3
        end;
    when  p[2].m
        begin
            Variable:= 4
        end;
```

3.2.7 พฤติกรรมของโมดูล Specification

ทุก ๆ โมดูลที่กำหนดในข้อกำหนดของโปรโตคอลต้องอยู่ภายใต้โมดูลหลักนั่นก็คือตัว Specification ของโปรโตคอลที่ออกแบบ โมดูลหลัก Specfication มีได้เพียงหนึ่งโมดูลเท่านั้นในแต่ละข้อกำหนดของโปรโตคอล ดังรูปแบบที่กำหนดโดย ISO 9074 ต่อไปนี้

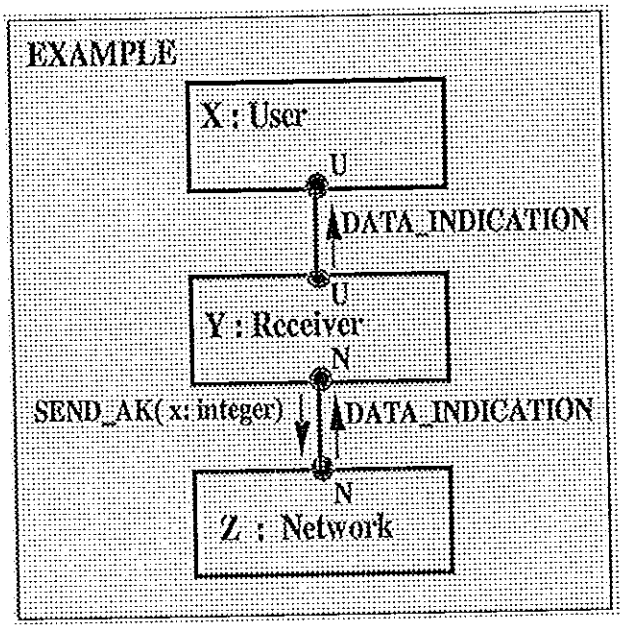
```
Specification specification_name [system_class];
        [default_option];
        [time_option];
        [Type definitions];
        [Channel definitions];
        [Module declarations and definitions];
        Body definition (of the specification);
End. { of the specification }
```

โดยที่วงเล็บก้ามปูหมายถึงข้อกำหนดที่อยู่ภายในวงเล็บเป็นทางเลือกจะมีหรือไม่มีก็ได้ในข้อกำหนดของโปรโตคอล ดังนั้นจะเห็นได้ว่าข้อกำหนดที่น้อยที่สุดจะประกอบด้วยการกำหนดหัว

เรื่อง (Heading definition) และการกำหนดส่วนเนื้อหา (Body definition) ทางเลือก system_class อาจเป็นอย่างใดอย่างหนึ่งดังนี้

- systemprocess
- systemactivity

ทางเลือก time_option เป็นการกำหนดหน่วยของเวลาเพื่อใช้สำหรับอ้างถึงในข้อกำหนดโดยไม่ต้องกำหนดหน่วยเวลาทุกครั้งเมื่อต้องการกล่าวถึงช่วงเวลาโดยใช้คำสงวน timescale ในการกำหนด เช่น หากกำหนด time_option เป็น timecale seconds เมื่อกล่าวถึงช่วงเวลา 10 จะหมายถึง 10 วินาทีนั่นเอง ซึ่งส่วนมากกำหนดช่วงเวลาจะใช้ในประโยค delay เพื่อกำหนดช่วงเวลาที่จะเปลี่ยนจากสถานะหนึ่งไปยังอีกสถานะหนึ่งของโมดูลใด ๆ โมดูล specification จะไม่มีจุดสื่อสารภายนอกและตัวแปรร่วม ซึ่งหมายถึงว่าโมดูลนี้ไม่ได้เป็นโมดูลที่จะติดต่อสื่อสารกับโมดูลภายนอกอื่น ๆ โมดูลหลักนี้เปรียบเสมือนกรอบโครงสร้างสำหรับอธิบายระบบทั้งหมดโดยไม่ต้องอธิบายรายเอียดภายในของแต่ละโมดูลย่อย ตัวอย่างต่อไปนี้สมมติว่าโมดูลอื่น ๆ ถูกเขียนอธิบายไว้เรียบร้อยแล้วโดยใช้คำ external จึงเขียนเฉพาะการทำงานของส่วนโมดูลหลัก specification



ภาพประกอบ 3-11 ตัวอย่างโมดูล Specification

(ที่มา: ISO 9074, 1989 : 34)

```
Specification EXAMPLE;
        Defalult individual queue;
        Timescale second;
        Channel UCH(User,Provider);
            By Provider: DATA_INDICATION;

        Channel NCH(User,Provider);
            By User : DATA_INDICATION;
            By Provider: SEND_AK(x:integer);
        Module USER systemactivity;
            Ip U: UCH(User);
        End;
        Body USER_BODY for USER; external;
        Module RECEIVER systemactivity;
            Ip U: UCH(Provider);
               N: NCH(Provider);
        End;
        Body RECEIVER_BODY for RECEIVER; external;
        Module NETWORK systemactivity;
            Ip N: NCH(User);
        End;
        Body NETWORK_BODY for NETWORK; external;
        Modvar X:USER; Y: RECEIVER; Z:NETWORK;
```

```
Initialize
    Begin
        Init X with USER_BODY;
        Init Y with RECEIVER_BODY;
        Init Z with NETWORK_BODY;
        Connect X.U to Y.U
        Connect Y.N to Z.N
    End;
End.
```

บทที่ 4

Estelle ในรูปแบบภาษาสัญลักษณ์ (Estelle/GR)

จากบทที่ผ่านมาได้กล่าวถึงรูปแบบและหลักภาษาหรือไวยากรณ์ของภาษา Estelle สำหรับใช้ในการเขียนข้อกำหนดโปรโตคอล ซึ่งการใช้ข้อความหรืออักขระต่าง ๆ มาประกอบกันเป็นประโยคนั้นเรียกว่าเป็นภาษาเชิงอักขระ (Textual Language) ซึ่งภาษาคอมพิวเตอร์ส่วนใหญ่หรือภาษาพูดที่ใช้กันอยู่ก็ใช้อักขระหรืออักษรในการแสดงแทนประโยคหรือคำพูดที่ต้องการสื่อให้ผู้อื่นทราบความหมาย ซึ่ง Estelle ที่เขียนขึ้นโดยใช้รูปแบบเชิงอักขระก็จะเรียกว่า Estelle/PR (Estelle/Phrasal Representation) แต่ในบางครั้งการใช้สัญลักษณ์แทนความหมายอาจทำให้ผู้อ่านจำได้ง่ายและตีความหมายได้ดีกว่าการแสดงในเชิงอักขระ อย่างเช่นแผนผังสายงาน (Flow chart) ที่นิยมใช้ในการออกแบบโปรแกรมนั้นจะทำให้มองภาพการทำงานโดยรวมของโปรแกรมหรือส่วนของโปรแกรมได้ง่ายกว่าการอ่านจากส่วนภาษาที่ใช้เขียนโปรแกรมนั้นขึ้นมา

ดังนั้นจึงมีผู้พยายามคิดค้นสัญลักษณ์ขึ้นมาเพื่อใช้ในการเขียนข้อกำหนดโปรโตคอลที่ให้มีความหมายตรงกับประโยคตามไวยากรณ์ของภาษา Estelle เพื่อความสะดวกในการอ่านหรือสามารถแปลเป็น Estelle ในรูปแบบเชิงอักขระได้สอดคล้องกันขึ้นมา โดยรูปแบบที่เป็นที่ยอมรับและคาดว่าจะกลายเป็นมาตรฐานต่อไปในไม่ช้าคือรูปแบบสัญลักษณ์ที่กำหนดขึ้นโดยสถาบันการสื่อสารข้อมูลระหว่างชาติในประเทศฝรั่งเศส (Institute National des Telecommunications) ซึ่ง Estelle ที่เขียนขึ้นโดยใช้ภาษาเชิงสัญลักษณ์ที่กล่าวมาจะเรียกว่า Estelle/GR (Estelle/Graphical representation)

การเขียนข้อกำหนดโปรโตคอลโดยใช้ Estelle/GR จะใช้แผนภาพสามชนิดในการเขียนอธิบายโครงสร้างของระบบและอธิบายพฤติกรรมการทำงานของโมดูลต่างในโปรโตคอลดังนี้

- แผนภาพแสดงโครงสร้างต้นไม้ (Structure tree diagram) ใช้กำหนดลำดับชั้นของโมดูลต่าง ๆ ที่มีอยู่ในระบบและแสดงส่วนหัวและส่วนรายละเอียดของโมดูลด้วย
- แผนภาพแสดงระบบอย่างคร่าว ๆ (Instance block diagram) ใช้กำหนดโครงสร้างของระบบอย่างคร่าว ๆ หรือแสดงการทำงานของระบบในสถานะเริ่มต้นการทำงาน (Initial state)

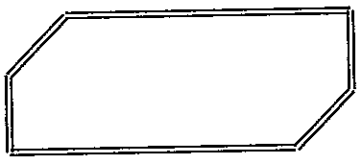
- แผนภาพพฤติกรรมของโมดูล (Module body diagram) ใช้กำหนดการทำงานภายในของโมดูลต่าง ๆ ในระบบ โดยอธิบายการเปลี่ยนสถานะต่าง ๆ ของโมดูลจากสถานะหนึ่งไปยังอีกสถานะหนึ่งเมื่อมีเงื่อนไขใด ๆ เกิดขึ้น

ข้อกำหนดโปรโตคอลใด ๆ โดยใช้ Estelle/GR นั้นต้องประกอบไปด้วยแผนภาพแสดงโครงสร้างต้นไม้หรือแผนภาพแสดงระบบอย่างคร่าว ๆ อย่างใดอย่างหนึ่ง กับแผนภาพพฤติกรรมของโมดูลแต่ละโมดูลที่มีในระบบ

4.1 แผนภาพแสดงระบบอย่างคร่าว ๆ (Instance block diagram)

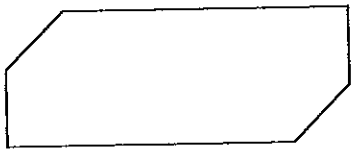
แผนภาพแสดงระบบอย่างคร่าว ๆ จะกำหนดโครงสร้างของข้อกำหนดโดยรวมซึ่งแสดงโมดูลต่าง ๆ ที่อยู่ภายใต้โมดูลหลัก (Specification) พร้อมด้วยช่องทางสื่อสารและจุดสื่อสารของโมดูลเหล่านั้นโดยไม่แสดงรายละเอียดภายในของโมดูลหรือโมดูลย่อยอื่น ๆ ในลำดับถัดไป ช่วยให้มองเห็นภาพโดยรวมของระบบเท่านั้น สัญลักษณ์ที่ใช้ในการกำหนดแผนภาพ Instance block diagram ต่าง ๆ มีดังนี้

1. สัญลักษณ์แสดงโมดูลที่มีคุณลักษณะชั้นเป็น Systemprocess

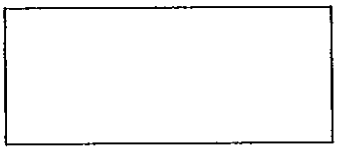


2. สัญลักษณ์แสดงโมดูลที่มีคุณลักษณะชั้นเป็น Systemactivity

3. สัญลักษณ์แสดงโมดูลที่มีคุณลักษณะชั้นเป็น process

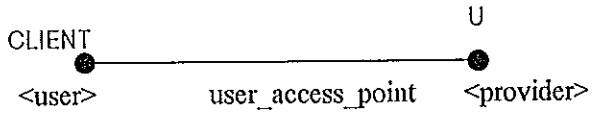


4. สัญลักษณ์แสดงโมดูลที่มีคุณลักษณะชั้นเป็น activity

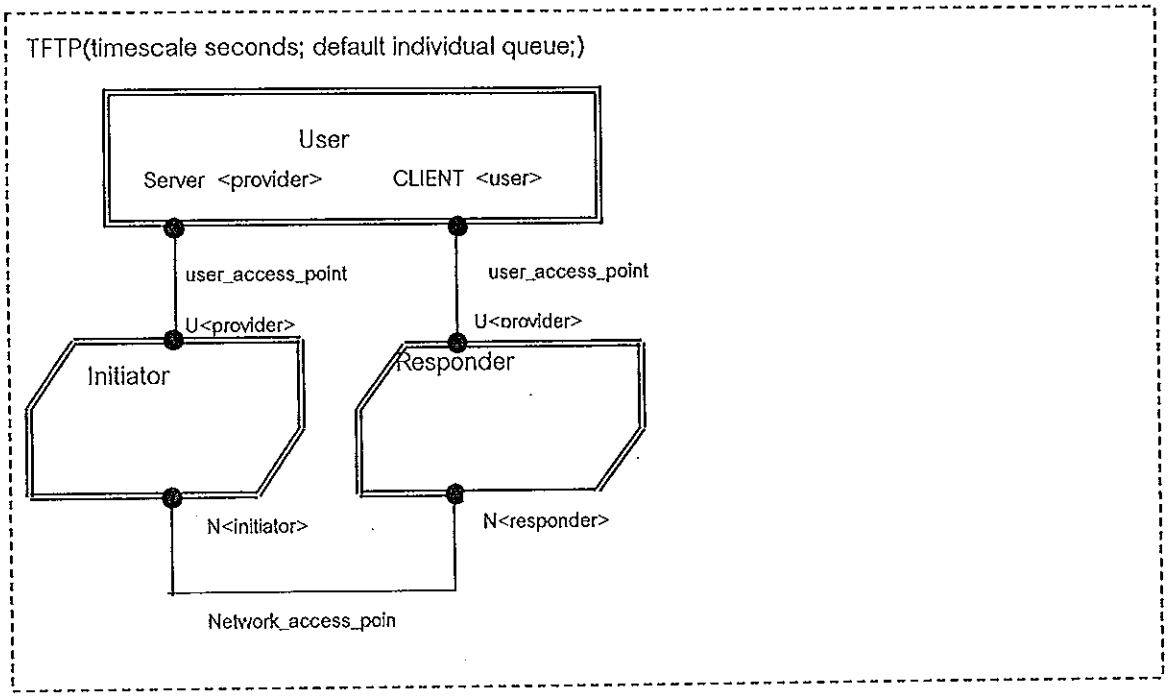


5. สัญลักษณ์แสดงโมดูลที่มีคุณลักษณะชั้น (unattributed

6. สัญลักษณ์แสดงเส้นทางสื่อสารที่เชื่อระหว่างจุดสื่อสารสองจุด



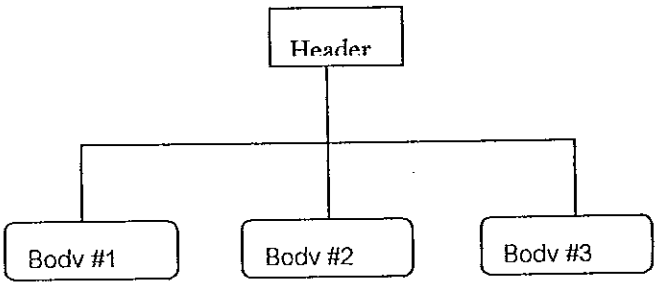
ตัวอย่างการเขียนโครงสร้าง Instance block ของโปรโตคอล TFTP เป็นดังนี้



ภาพประกอบ 4-1 โครงสร้างระบบอย่างคร่าว ๆ

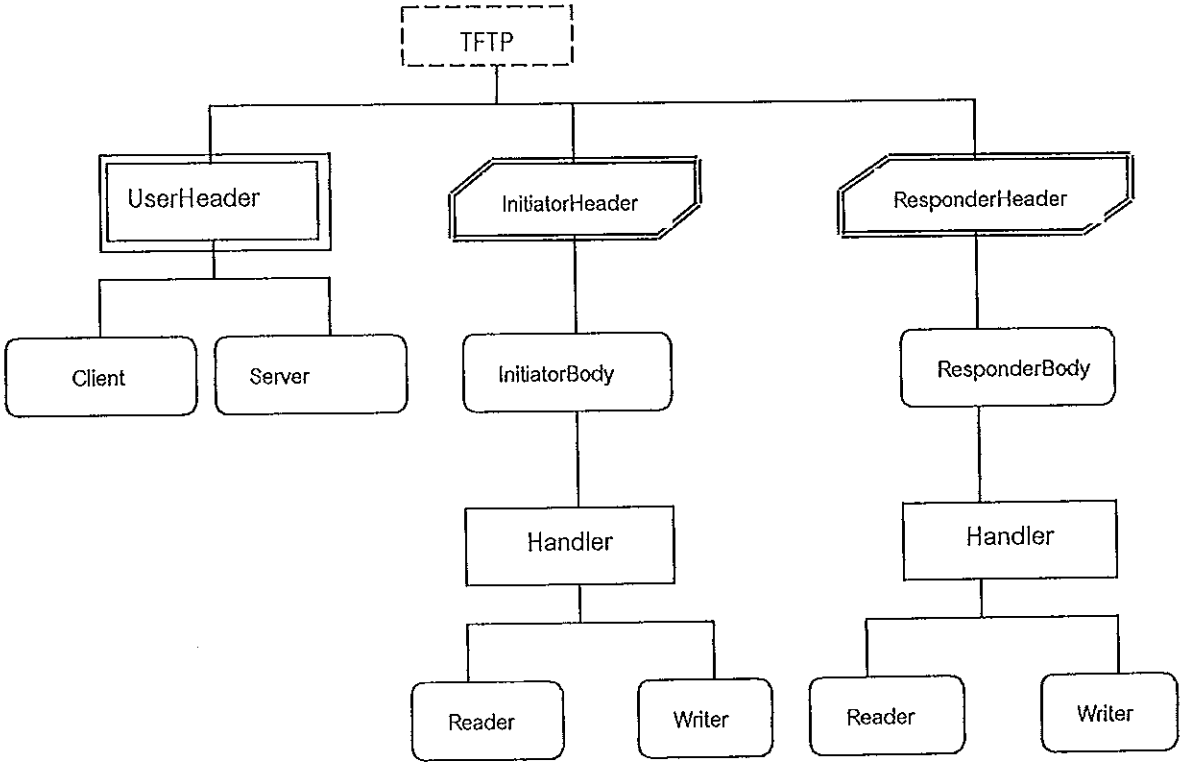
4.2 แผนภาพแสดงโครงสร้างต้นไม้ (Structure tree diagram)

แผนภาพแสดงโครงสร้างต้นไม้ใช้อธิบายโมดูลและโมดูลย่อยต่าง ๆ ที่มีในระบบ โดยแสดงให้เห็นส่วนหัวและส่วนรายละเอียดของโมดูลที่เกี่ยวข้องข้องกับส่วนหัวของโมดูลนั้นอย่างชัดเจนโดยใช้สัญลักษณ์สี่เหลี่ยมมนแทนส่วนรายละเอียดของโมดูลดังนี้



ภาพประกอบ 4-2 สัญลักษณ์รายละเอียดของโมดูลที่เกี่ยวข้องกับส่วนหัวของโมดูล

เมื่อใช้แผนภาพโครงสร้างต้นไม้แสดงโครงสร้างของโปรโตคอล TFTP จะได้ดังภาพประกอบ 3-3

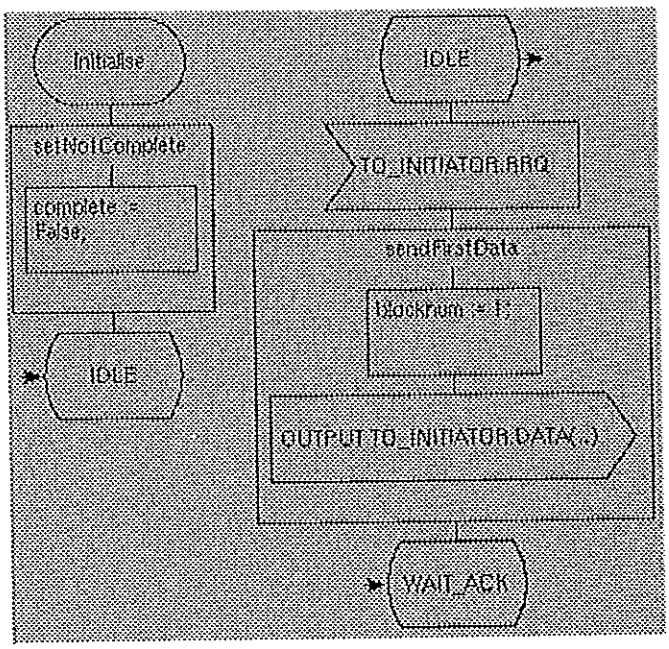


ภาพประกอบ 4-3 แผนภาพโครงสร้างต้นไม้ของโปรโตคอล TFTP ที่ประกอบด้วยโมดูลต่าง ๆ

การแสดงระบบในรูปโครงสร้างต้นไม้ตาม Estelle/GR ไม่สามารถแสดงเส้นทางสื่อสารได้เพราะจะเน้นเฉพาะส่วนของโมดูลทั้งส่วนหัวและส่วนรายละเอียดโมดูลที่เกี่ยวข้องกัน

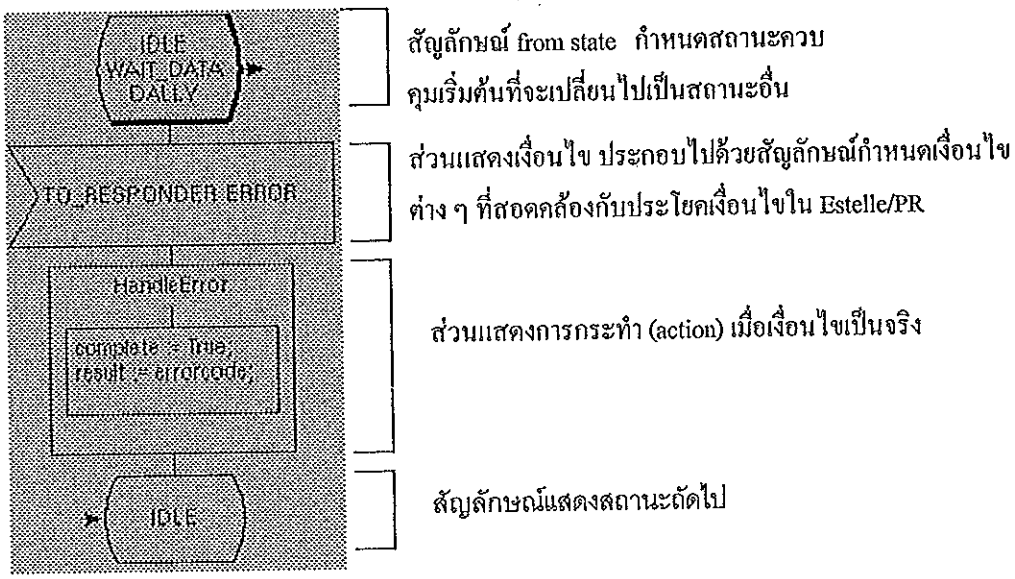
4.3 แผนภาพพฤติกรรมของโมดูล (Module body diagram)

แผนภาพแสดงพฤติกรรมของโมดูลเป็นส่วนอธิบายรายละเอียดการทำงานภายในของโมดูลในส่วนที่เป็นรายละเอียดของโมดูล (Body) โดยการแสดงพฤติกรรมของโมดูลจะเป็นการอธิบายการเปลี่ยนสถานะจากสถานะหนึ่งไปยังอีกสถานะหนึ่ง แต่ละการประกาศการเปลี่ยนสถานะแสดงด้วยต้นไม้ของการเปลี่ยนสถานะ (Transition tree) ดังนั้นในหนึ่งรายละเอียดของโมดูลจะประกอบไปด้วยชุดของต้นไม้การเปลี่ยนสถานะที่สอดคล้องกับ transition part ใน Estelle/PR ภาพประกอบ 4-4 แสดงชุดของต้นไม้การเปลี่ยนสถานะของโมดูลหนึ่งในโปรโตคอล TFTP



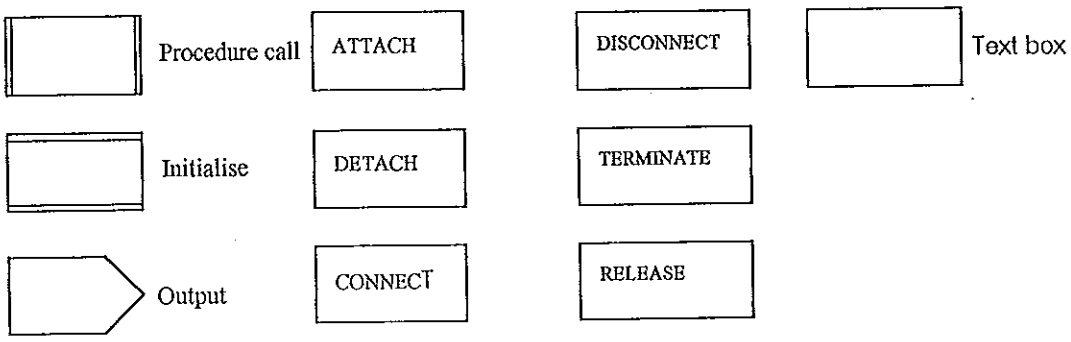
ภาพประกอบ 4-4 ตัวอย่างต้นไม้การเปลี่ยนสถานะของโมดูล

สัญลักษณ์ต่าง ๆ ในต้นไม้การเปลี่ยนสถานะแบ่งเป็นสี่ส่วนใหญ่ ๆ ได้ดังภาพประกอบ 4-5



ภาพประกอบ 4-5 ส่วนต่าง ๆ ของสัญลักษณ์ในต้นไม้การเปลี่ยนสถานะ

ในส่วนของส่วนแสดงการกระทำ อาจเป็นการเรียกใช้โพรซิเยอร์หรือฟังก์ชันรวมถึงการกำหนดค่าตัวแปรต่าง ๆ ตามคำสั่งภาษาปาสคาลส่วนคำสั่งที่เป็นคำสั่งของ Estelle โดยเฉพาะก็จะมีสัญลักษณ์ต่าง ๆ ใช้ดังภาพประกอบ 4-6



ภาพประกอบ 4-6 สัญลักษณ์คำสั่ง Estelle

สำหรับส่วนกำหนดสถานะเริ่มต้นก็ใช้สัญลักษณ์ 

แทนคำสั่ง initialise

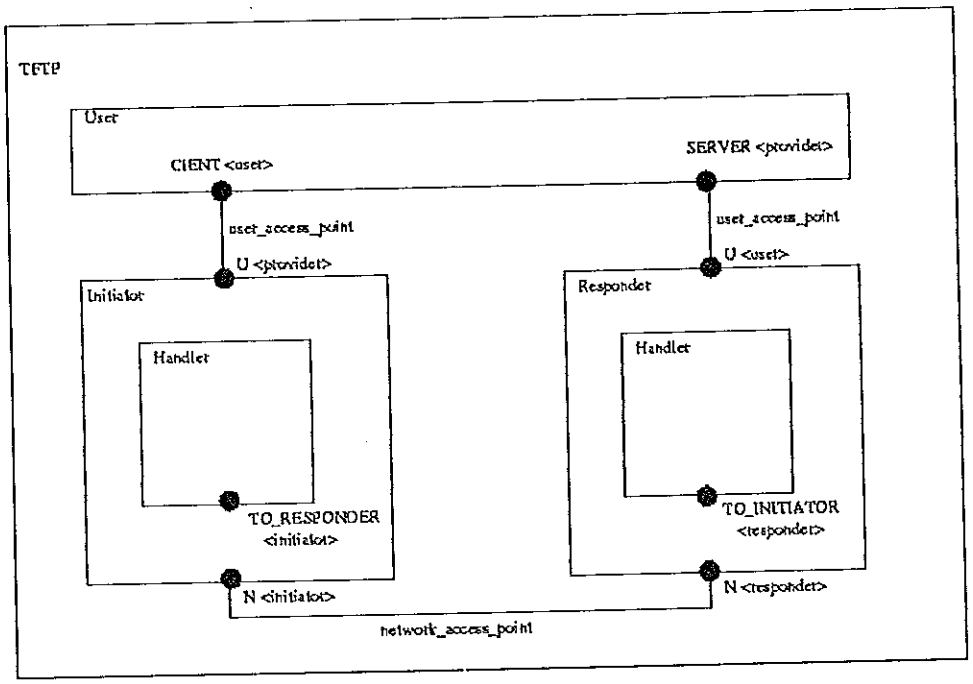
บทที่ 5

ข้อกำหนดโปรโตคอล TFTP โดยใช้เครื่องมือ Estelle Graphical Editor

เมื่อทราบการทำงานของโปรโตคอล TFTP และข้อกำหนด FDT ตามแบบภาษา Estelle แล้วก็สามารถเขียนข้อกำหนดโปรโตคอล TFTP ได้ แต่การเขียนข้อกำหนดด้วยภาษาเชิงอักขระของ Estelle อาจทำให้มองเห็นภาพรวมและพฤติกรรมการทำงานภายในของแต่ละโมดูลของโปรโตคอลไม่ชัดเจนเหมือนกับใช้ภาษา Estelle เชิงสัญลักษณ์หรือที่เรียกว่า Estelle/GR เนื่องจาก Estelle เป็น FDT ที่เป็นที่นิยมจึงมีผู้สร้างโปรแกรมหรือเครื่องมือที่ใช้ช่วยในการเขียนข้อกำหนดตามแบบ Estelle/GR ขึ้นมาเรียกว่า Estelle Graphical Editor ซึ่งเขียนขึ้นโดย Justin George Templemore-Finlayson ขณะทำงานให้กับ Institut National des Telecommunications โดยรุ่นล่าสุดคือรุ่น 2.1 ซึ่งทำงานบนระบบปฏิบัติการยูนิกซ์ Solaris 2.5.1

5.1 โครงสร้างโมดูลของโปรโตคอล TFTP

โปรโตคอล TFTP ที่ผู้ทำการวิจัยออกแบบเพื่อเขียนเป็นข้อกำหนดตามหลัก Estelle สามารถแบ่งเป็นโมดูลต่าง ๆ และมีส่วนประกอบในการติดต่อสื่อสารหว่างกันได้ตามภาพประกอบ 5-1



ภาพประกอบ 5-1 โครงสร้างโปรโตคอล TFTP

5.1.1 โมดูล User

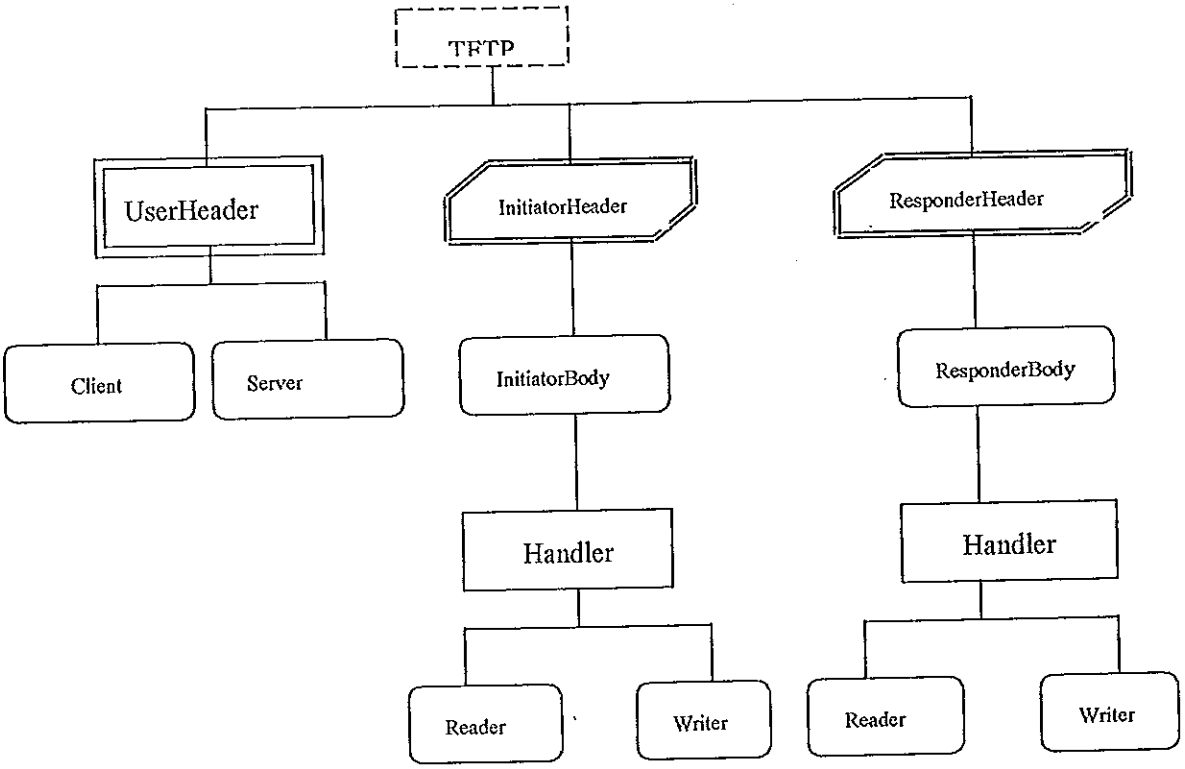
เป็นโมดูลที่ทำหน้าที่ส่งข่าวสาร READ_RQST WRITE_RQST และRESPONSE เพื่อร้องรับแฟ้ม ร้องขอส่งแฟ้มหรือตอบอนุญาตการร้องขอตามลำดับ โมดูลนี้จะมีพฤติกรรมภายในที่ตรงกันข้ามกันคือเป็นผู้ขอบริการฝ่ายหนึ่งกับเป็นผู้ให้บริการอีกฝ่ายหนึ่ง ดังนั้นในการประกาศส่วนรายละเอียด (body) ของโมดูลจะมีส่วนรายละเอียดสองส่วนที่ทำงานตรงข้ามกัน โดยฝ่ายที่ทำหน้าที่เป็นผู้ขอบริการจะส่งข่าวสาร READ_RQST และ WRITE_RQST ผ่านจุดสื่อสาร CLIENT ส่วนฝ่ายผู้ให้บริการจะตอบรับการอนุญาติการร้องขอหรือไม่อนุญาติจะส่งเป็นข่าวสาร RESPONSE ผ่านทางจุดสื่อสาร SERVER

5.1.2 โมดูล Initiator และโมดูล Responder

โมดูล Initiator และโมดูล Responder ถือเป็นโมดูลสำคัญที่ทำหน้าแสดงการทำงานของโปรโตคอล โดยเมื่อมีการร้องขอ READ_RQST หรือ WRITE_RQST ผ่านทางช่องสื่อสาร user_access_point มายังจุดสื่อสาร U ของโมดูล Initiator โมดูลนี้ก็จะทำการเรียกใช้โมดูลลูก Handler อีกทีหนึ่ง โดยโมดูลลูก Handler จะมีพฤติกรรมที่แตกต่างกันขึ้นอยู่กับว่าโมดูล Initiator จะให้แสดงพฤติกรรมใด ซึ่งหากโมดูล Initiator ได้รับ READ_RQST ก็จะสั่งโมดูล Handler แสดงพฤติกรรมเพื่ออ่านหรือรับแฟ้มแต่หากได้รับ WRITE_RQST โมดูล Initiator ก็จะสั่งให้โมดูล Handler แสดงพฤติกรรมเพื่อเขียนหรือส่งแฟ้ม ดังนั้นส่วนแสดงรายละเอียด (Body) ของโมดูล Handler จึงมีสองส่วน สำหรับโมดูล Responder ก็จะทำงานในทำนองเดียวกันขึ้นอยู่กับว่าโมดูล Initiator ส่งผ่านการร้องขอแบบไหนมา หากเป็นการร้องขอ READ_RQST ก็จะรอข่าวสาร REPONSE จากโมดูล User พฤติกรรมเป็นผู้ให้บริการ ผู้ให้บริการตอบรับอนุญาติก็จะเรียกใช้โมดูล Handler ให้ทำพฤติกรรมอ่านหรือขอข้อมูล โดยรายละเอียดพฤติกรรมของโมดูลลูก Handler ทั้งในโมดูล Initiator และโมดูล Responder จะเป็นการควบคุมการรับหรือส่งแพคเก็ตข้อมูลและแพคเก็ตแจ้งข่าวสารให้สอดคล้องกันตามลักษณะการทำงานของโปรโตคอล

5.2 แผนภาพแสดงโครงสร้างต้นไม้ของโปรโตคอล TFTP ตามแบบ Estelle/GR

เมื่อทราบโครงสร้างโมดูลโดยรวมของโปรโตคอลที่จะเขียนข้อกำหนดแล้วก็สามารถนำมาเขียนตามลักษณะภาษา Estelle ในเชิงสัญลักษณ์ได้ ซึ่งต้องประกอบด้วยแผนภาพแสดงโครงสร้างต้นไม้หรือแผนภาพแสดงระบบอย่างคร่าว ๆอย่างใดอย่างหนึ่งกับแผนภาพพฤติกรรมของโมดูล แต่สำหรับเครื่องมือ Estelle graphical editor จะใช้แผนภาพแสดงโครงสร้างต้นไม้กับแผนภาพแสดงพฤติกรรมในการเขียนข้อกำหนดโปรโตคอลซึ่งแผนภาพแสดงโครงสร้างต้นไม้ของโปรโตคอล TFTP เป็นไปตามภาพประกอบ 5-2



ภาพประกอบ 5-2 แผนภาพแสดงโครงสร้างต้นไม้ของโปรโตคอล TFTP

จากภาพประกอบ 5-2 จะได้ว่าข้อกำหนดโปรโตคอล TFTP มีโมดูลหลักอยู่สามโมดูลคือ โมดูล User โมดูล Initiator และโมดูล Responder โดยที่การอ้างถึงหรือกล่าวถึงโมดูลจะใช้ส่วนหัวของโมดูลในการอ้างอิงซึ่งก็จะได้ว่า UserHeader InitatorHeader และ ResponderHeader เป็นส่วนหัวของโมดูลทั้งสามตามลำดับโดยใช้สัญลักษณ์ตามข้อกำหนดของ Estelle/GR ซึ่งจะได้ว่าโมดูล User มีคุณลักษณะชั้นเป็นแบบ Activity ส่วนโมดูล Initiator และ โมดูล Responder มีลักษณะชั้น

เป็นแบบ Systemprocess โดยโมดูล Initiator และโมดูล Responder จะมีโมดูลย่อยอีกคือโมดูล Handler ที่มีคุณลักษณะชั้นแบบ activity ส่วนพฤติกรรมการทำงานของโมดูลต่าง ๆ นั้นจะขึ้นอยู่กับการอธิบายภายในส่วนรายละเอียด (Body) ของแต่ละโมดูล จากรูปจะได้ว่าโมดูล User มีส่วนรายละเอียดแสดงพฤติกรรมสองส่วนซึ่งแทนด้วยชื่อ Client และ Server สำหรับโมดูล Initiator และโมดูล Responder มีส่วนแสดงรายละเอียดเพียงส่วนเดียวคือส่วนที่แทนด้วย InitiatorBody และ ResponderBody แต่โมดูลย่อยของสองโมดูลนี้คือโมดูล Handler มีส่วนรายละเอียดสองส่วนที่แสดงพฤติกรรมเป็น Reader และ Writer

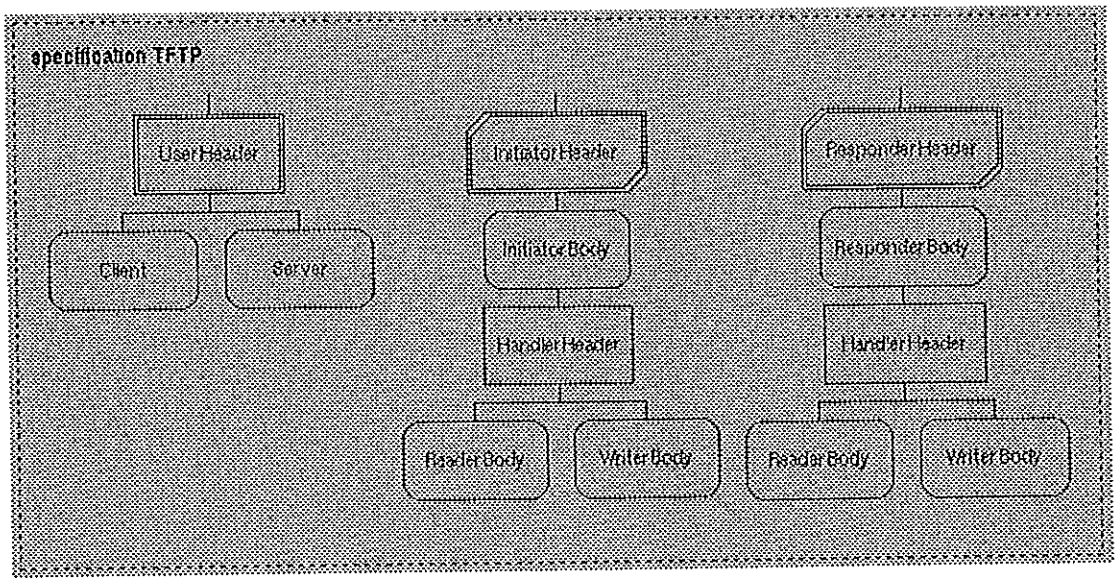
5.3 ข้อกำหนดโปรโตคอล TFTP โดยใช้เครื่องมือ Estelle Graphical Editor

การทำงานของโปรแกรม Estelle Graphcial Editor แบ่งเป็นสองส่วนหลัก ๆ คือ

1. ส่วนกำหนดโครงสร้างข้อกำหนด
2. ส่วนอธิบายพฤติกรรมของโมดูล

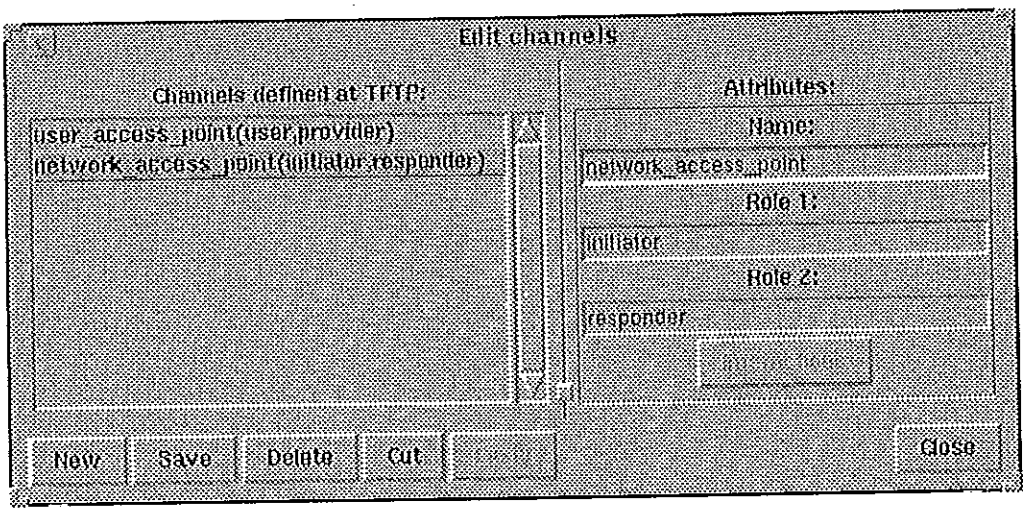
5.3.1 ส่วนกำหนดโครงสร้างข้อกำหนด

ส่วนกำหนดโครงสร้างข้อกำหนดใช้สำหรับเขียนแผนภาพโครงสร้างต้นไม้ตามแบบข้อกำหนด Estelle/GR โดยมีส่วนโมดูลหลัก Specification ล้อมรอบโมดูลอื่น ๆ อยู่ภายในดังภาพประกอบ 5-3



ภาพประกอบ 5-3 โครงสร้างต้นไม้ของข้อกำหนดโปรโตคอล TFTP ใน Estelle Graphical Editor

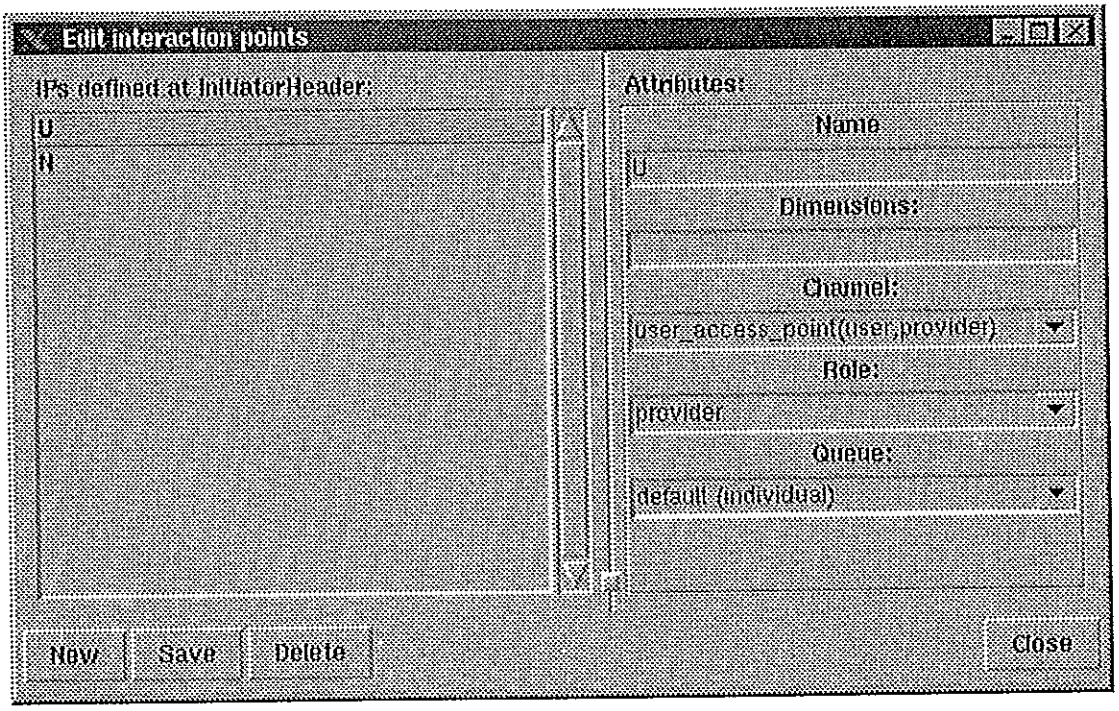
จากรูปไม่มีการแสดงช่องสื่อสารและจุดสื่อสารต่าง ๆ เหมือนภาพประกอบ 4-1 แต่สามารถกำหนดได้โดยคลิกเมาส์ที่พื้นที่ว่างภายในกรอบ Specification แล้วเลือกปุ่มกำหนดช่องสื่อสารก็จะได้หน้าต่างสำหรับกำหนดช่องสื่อสาร จากภาพประกอบ 5-1 ต้องกำหนดช่องสื่อสาร user_access_point และช่องสื่อสาร network_access_point ก็จะได้ดังภาพประกอบ 5-4



ภาพประกอบ 5.4 การกำหนดช่องสื่อสารในโดยใช้ Estelle Graphical Editor

การกำหนดช่องสื่อสารต้องระบุข่าวสารที่จะรับส่งผ่านทางช่องสื่อสารนั้นด้วย

สำหรับจุดสื่อสารของแต่ละโมดูลก็กำหนดได้โดยใช้เมาส์คลิกที่ส่วนหัวของโมดูลนั้นแล้วเลือกปุ่มกำหนดจุดสื่อสาร จากภาพประกอบ 5-5 เป็นการกำหนดจุดสื่อสาร U และ N ให้แก่โมดูล Initiator

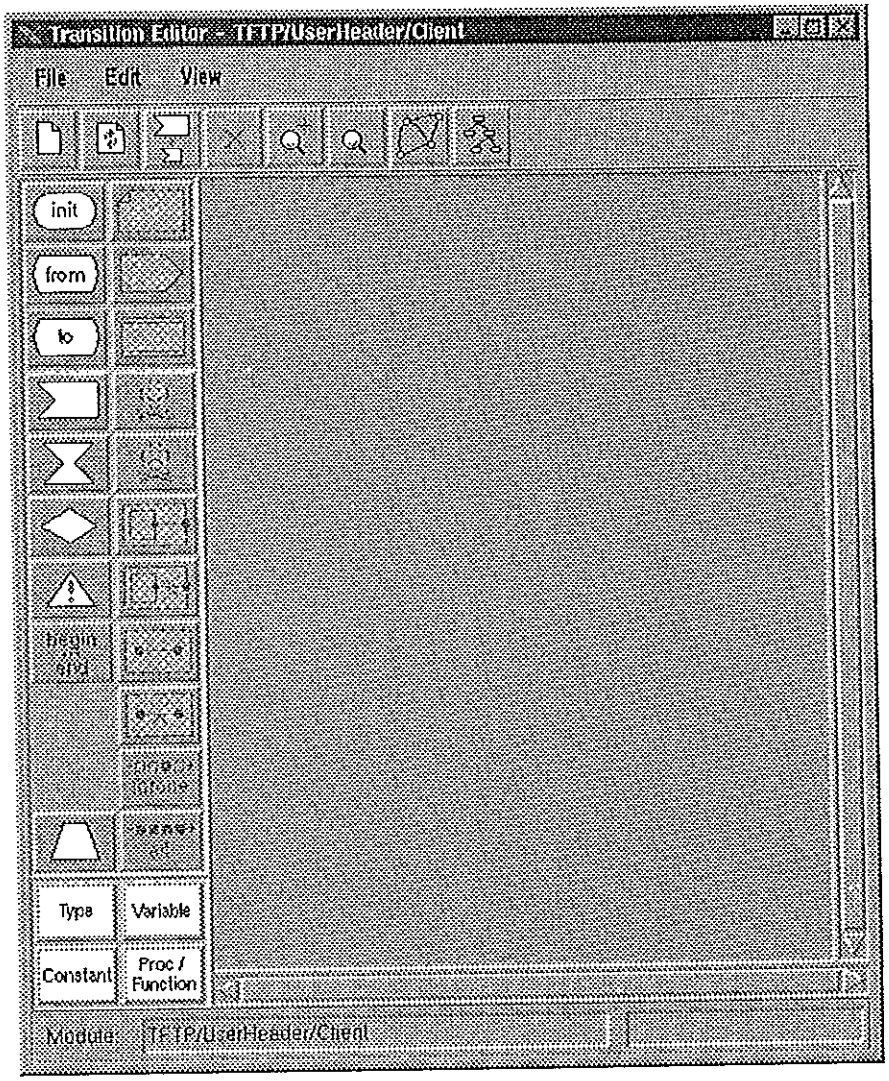


ภาพประกอบ 5-5 การกำหนดจุดสื่อสารให้แก่โมดูล Initiator

สำหรับจุดสื่อสารของโมดูลอื่น ๆ ก็กำหนดได้ในทำนองเดียวกัน

5.3.2 ส่วนอธบายพฤติกรรมของโมดูล

การอธิบายหรือเขียนส่วนแสดงพฤติกรรมของโมดูลโดยใช้ Estelle Graphical Editor นั้นจะเป็นการเปิดหน้าต่างการทำงานของโปรแกรมขึ้นมาอีกหน้าต่างหนึ่งต่างหาก เช่นเมื่อต้องการเขียนส่วนรายละเอียดของโมดูล Initiator ก็คลิกเมาส์ที่ส่วนกำหนดรายละเอียด (Body) InitiatorBody จากภาพประกอบ 5-3 แล้วเลือกปุ่มกำหนดพฤติกรรมก็จะได้หน้าต่างการทำงานขึ้นมาใหม่เพื่อเขียนรายละเอียดพฤติกรรมของโมดูลที่เลือกดังภาพประกอบ 5-6

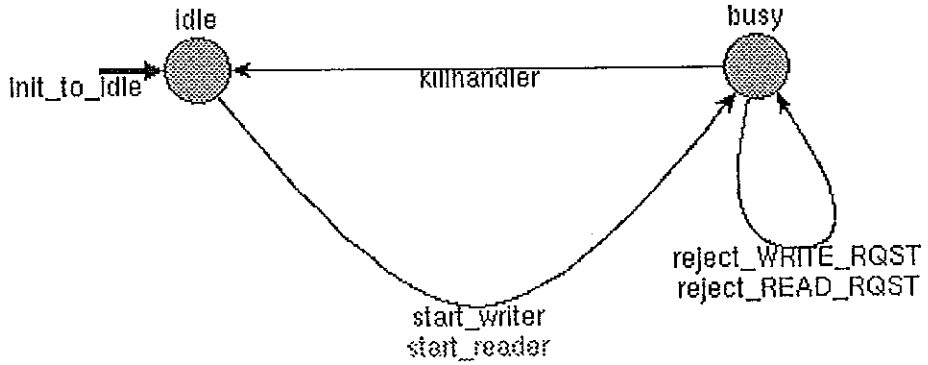


ภาพประกอบ 5-6 หน้าต่างสำหรับกำหนดพฤติกรรมของโมดูล

จากหน้าต่างนี้ก็สามารถใช้ปุ่มต่าง ๆ ทางด้านซ้ายมือเขียนพฤติกรรมของโมดูลในรูปแบบ Estelle/GR

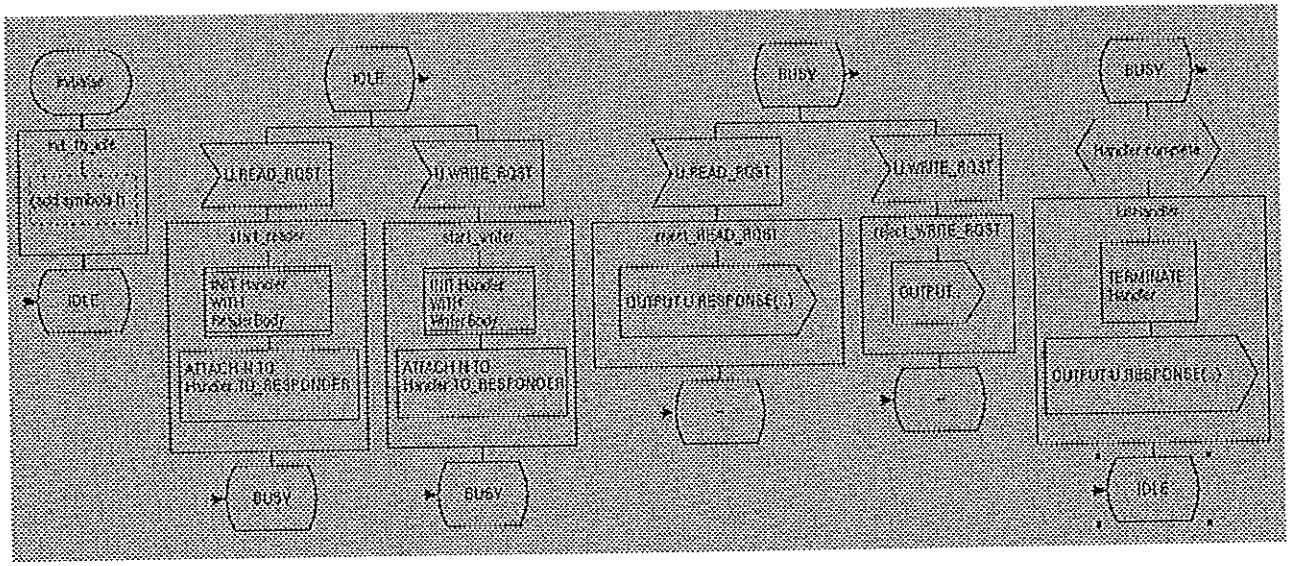
- พฤติกรรมโมดูล Initiator

  โมดูล Initiator จะทำหน้าที่เรียกใช้โมดูลย่อย Handler อีกทีหนึ่งให้สอดคล้องกับข่าวสารที่ได้รับมาเช่นหากเป็น READ_RQST ก็จะเรียกโมดูล Handler ที่แสดงพฤติกรรม Writer แต่หากเป็นข่าวสาร WRITE_RQST ก็จะเรียกโมดูล Handler ที่แสดงพฤติกรรม Reader มาทำงานอีกทีหนึ่ง โดยที่การเปลี่ยนสถานะการทำงานของโมดูล Initiator เป็นดังภาพประกอบ 5-7 ซึ่งมีสถานะการทำงานอยู่ 2 สถานะคือ IDLE กับ BUSY



ภาพประกอบ 5-7 การทำงานของโมดูล Initiator

จากภาพประกอบ 5.7 นำมาเขียนพฤติกรรมของโมดูล Initiator ด้วยโปรแกรม Estelle Graphical Editor ซึ่งประกอบด้วยส่วนกำหนดการเปลี่ยนสถานะสี่ส่วน ส่วนแรกเป็นการกำหนดสถานะเริ่มต้นแก่โมดูลคือส่วนที่มีสัญลักษณ์ initialise และส่วนอื่น ๆ ที่เหลือเป็นการเปลี่ยสถานะจากสถานะควบคุมหนึ่งไปยังอีกสถานะหนึ่ง ดังภาพประกอบ 5-8



ภาพประกอบ 5-8 พฤติกรรมโมดูล Initiator

จากข้อกำหนดตามรูปแบบ Estelle/GR ข้างต้นสามารถเขียนในรูปแบบภาษา Estelle เชิงอักขระ หรือ Estelle/PR ได้ดังนี้

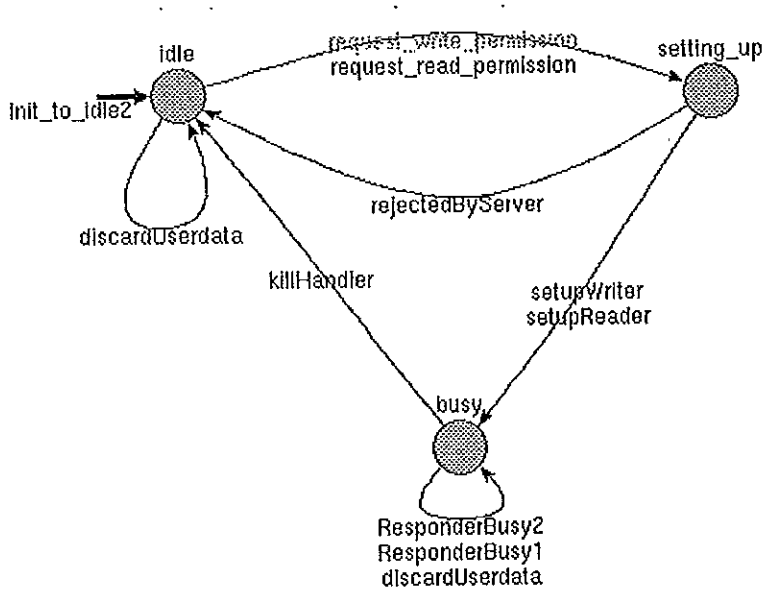
```
initialise
   TO IDLE
        Name init_to_idle : begin
        End;
Trans
   When U.READ_RQST
        FROM IDLE
          TO BUSY
             Name start_reader : begin
               INIT handler WITH ReaderBody;
               ATTACH N TO Handler.TO_RESPONDER;
             End;
        FROM BUSY
          TO same
             Name reject_READ_RQST : begin
```

```
                OUTPUT U.RESPONDER(INITIATOR_BUSY)
            End;
        Begin end;

Trans
    When U.WRITE_RQST
        FROM IDLE
            TO BUSY
                Name start_writer : begin
                    INIT handler WITH WriterBody;
                    ATTACH N TO Handler.TO_RESPONDER;
                End;
        FROM BUSY
            TO same
                Name reject_WRITE_RQST : begin
                    OUTPUT U.RESPONDER(INITIATOR_BUSY)
                End;
        Begin end;
Trans
    PROVIDED Handler.complete
        FROM BUSY
            TO IDLE
                Name killHandler : begin
                    TERMINATE handler;
                    OUTPUT U.RESPONSE(handler.result);
                End;
```

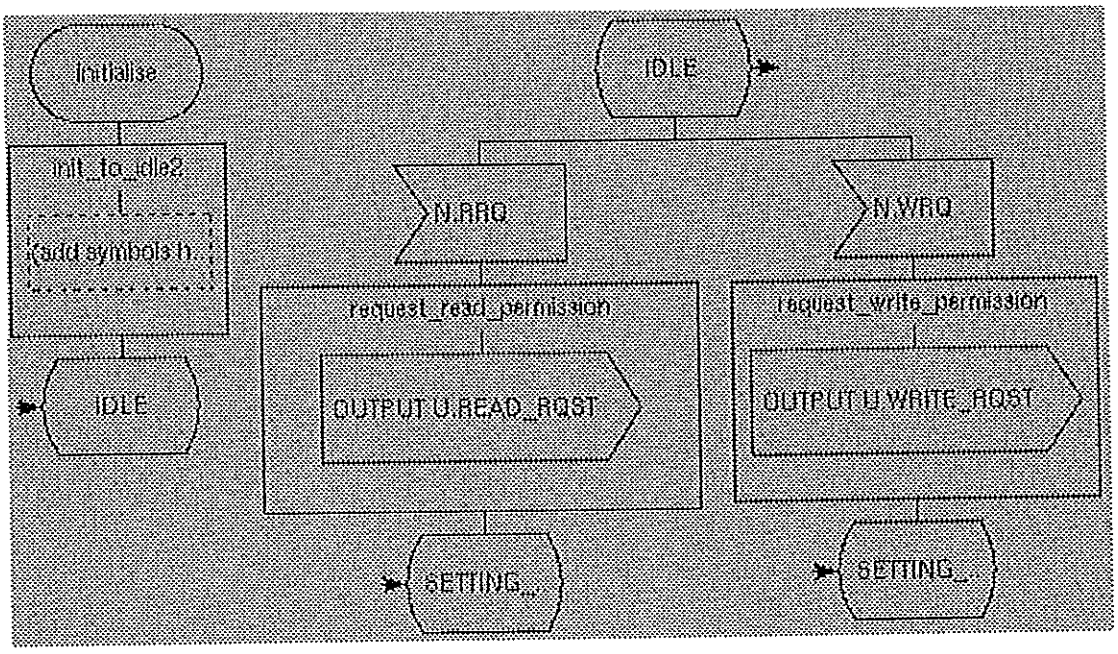
- พฤติกรรมโมดูล Responder

  โมดูล Responder จะทำงานตรงข้ามกับโมดูล Initiator คือจะคอยตอบสนองคำร้องขอจากฝ่ายตรงข้าม (Initiator) โดยเรียกใช้โมดูล Handler ซึ่งเป็นโมดูลลูกของตัวเองเช่นกัน ซึ่งโมดูล Handler ก็มีพฤติกรรมการทำงานสองอย่างคือพฤติกรรม Reader และ Writer เพื่อเรียกใช้ให้เหมาะสมกับข่าวสารที่ได้รับมาว่าเป็นเช่นใดจึงแสดงพฤติกรรมให้สอดคล้องกับข่าวสารนั้น ภาพประกอบ 5-9 แสดงการทำงานของโมดูล Responder

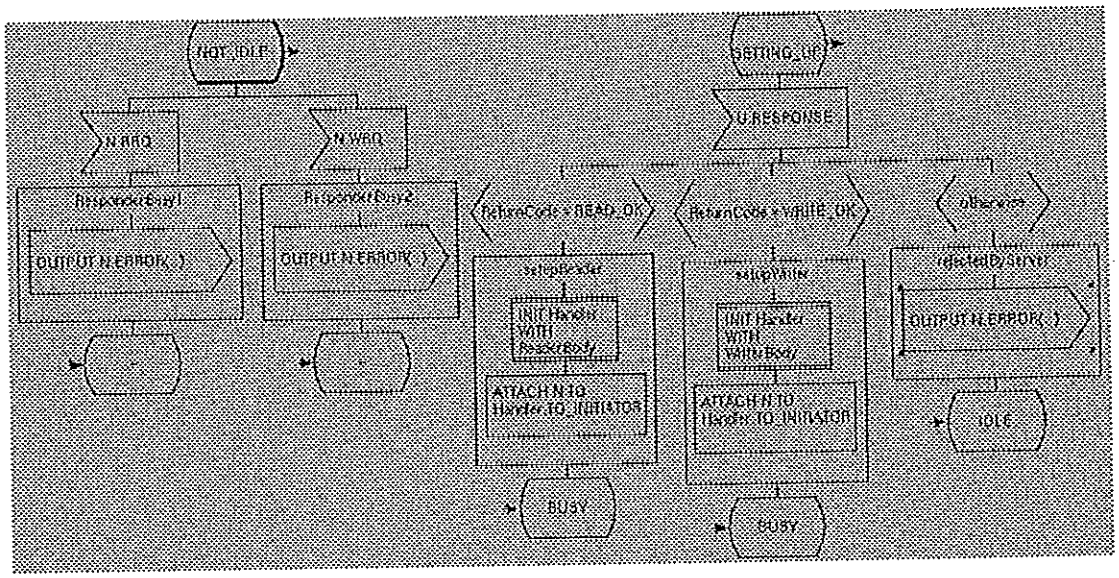


ภาพประกอบ 5-9 การทำงานของโมดูล Responder

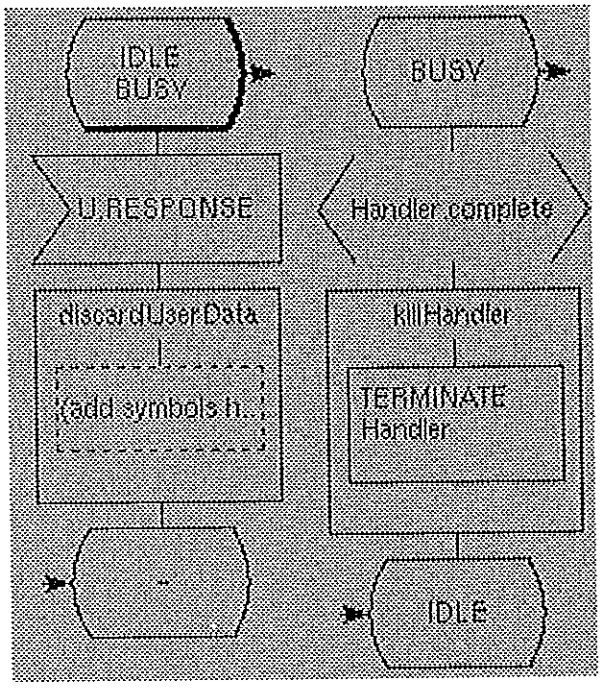
พฤติกรรมของโมดูล Responder นำมาเขียนตามรูป แบบ Estelle/GR ก็จะได้ดังภาพประกอบ 5-10 5-11 และ 5-12



ภาพประกอบ 5-10 พฤติกรรมโมดูล Responder (1)

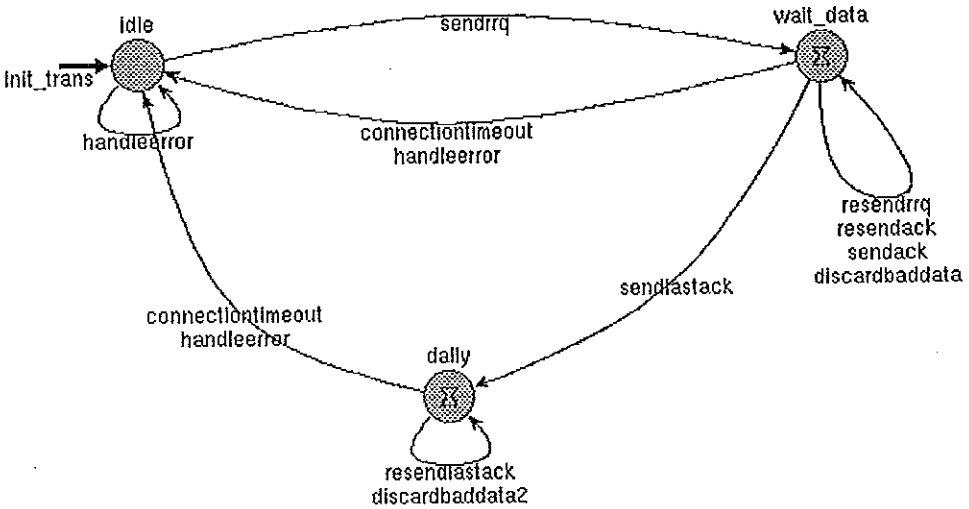


ภาพประกอบ 5-11 พฤติกรรมโมดูล Responder (2)



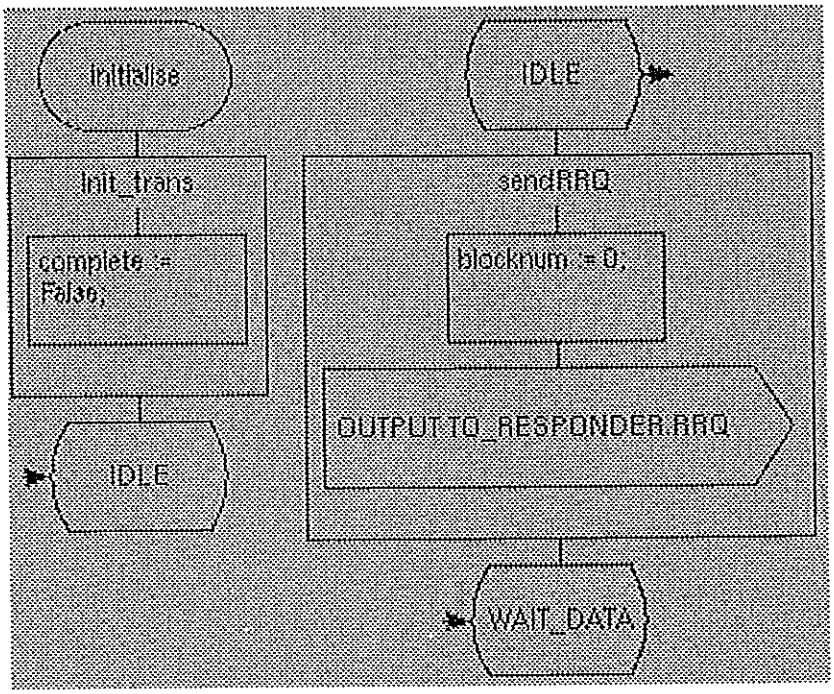
ภาพประกอบ 5-12 พฤติกรรมโมดูล Responder (3)

- พฤติกรรมโมดูลลูก Handler ของโมดูล Initiator ที่แสดงพฤติกรรม Reader โมดูล Handler ที่เป็นโมดูลลูกของโมดูล Initiator แสดงพฤติกรรมสองอย่างคือเป็น Reader และ Writer โดยส่วนที่เป็น Reader มีการทำงานดังภาพประกอบ 5-13

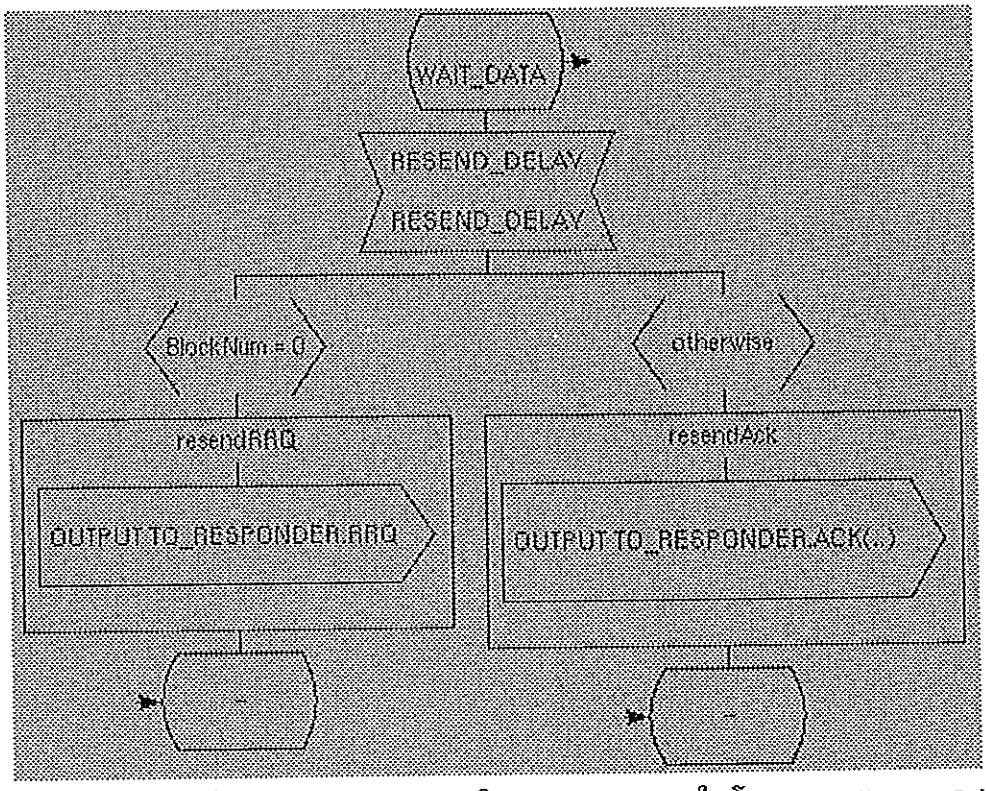


ภาพประกอบ 5-13 การทำงานของโมดูลลูก Handler ของ Initiator ที่เป็น Reader

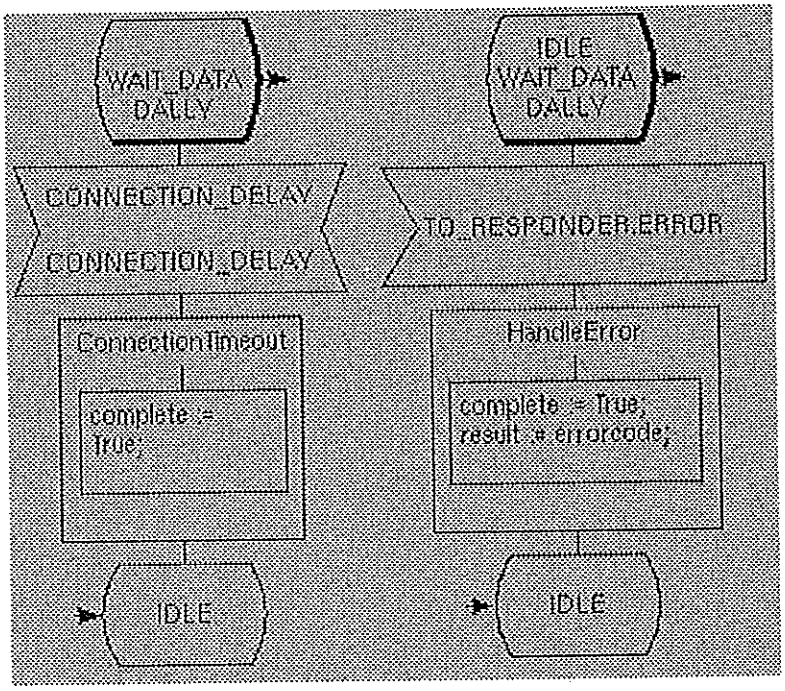
เมื่อนำมาเขียนพฤติกรรมในรูปแบบ Estelle/GR ก็จะได้ดังภาพประกอบ 5-14 5-15 5-16 5-17 และ 5-18



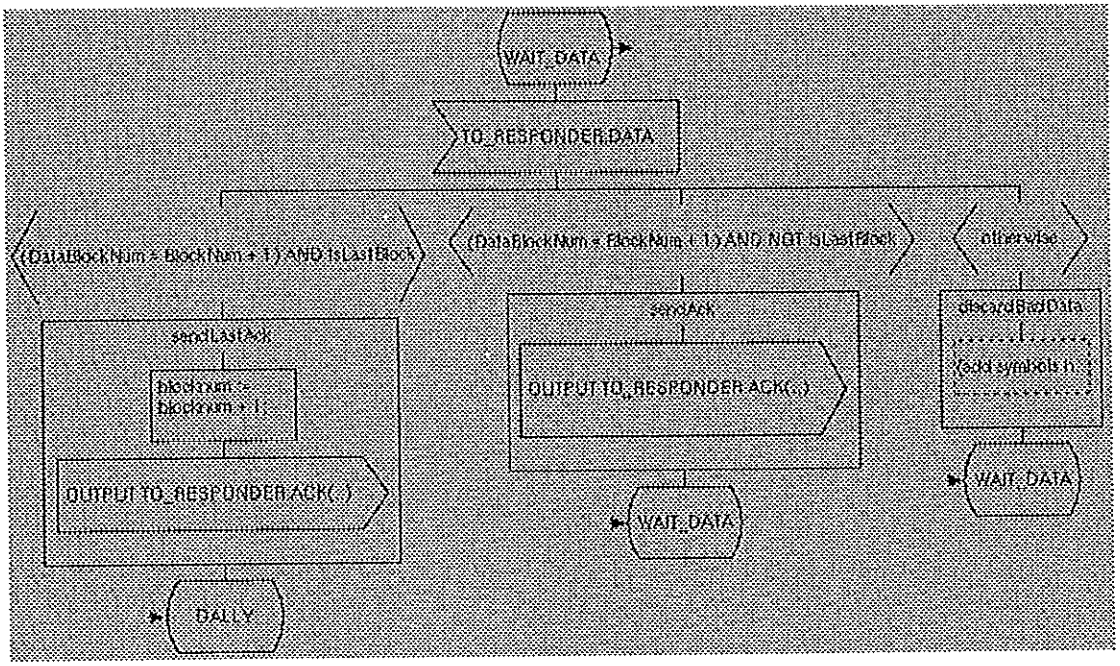
ภาพประกอบ 5-14 พฤติกรรม Reader (1) ในโมดูล Handler ของ Initiator



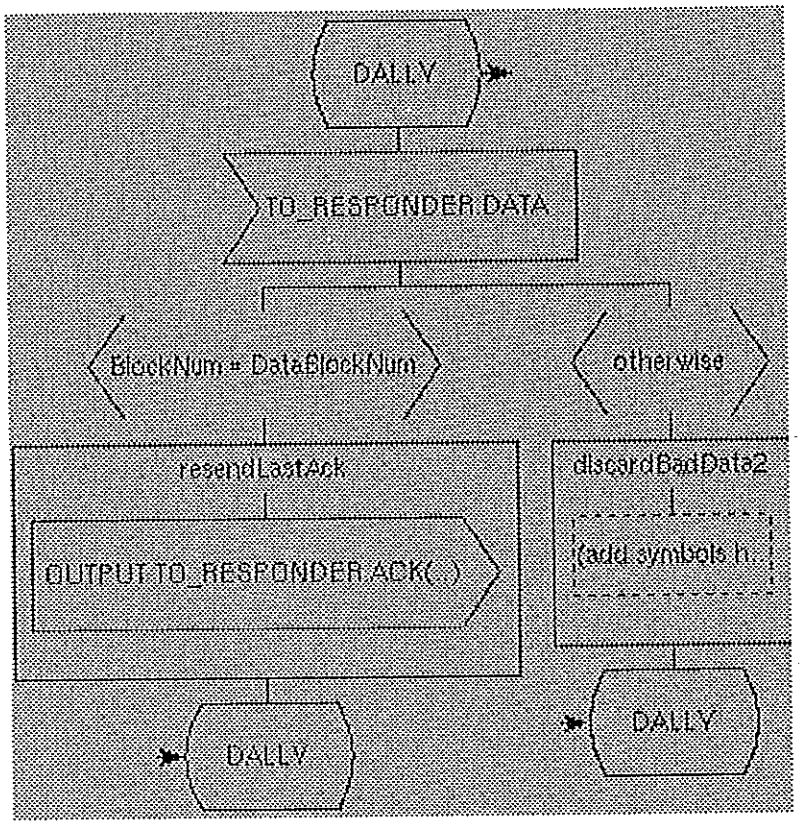
ภาพประกอบ 5-15 แสดงพฤติกรรม Reader (2) ในโมดูล Handler ของ Initiator



ภาพประกอบ 5-16 แสดงพฤติกรรม Reader (3) ในโมดูล Handler ของ Initiator

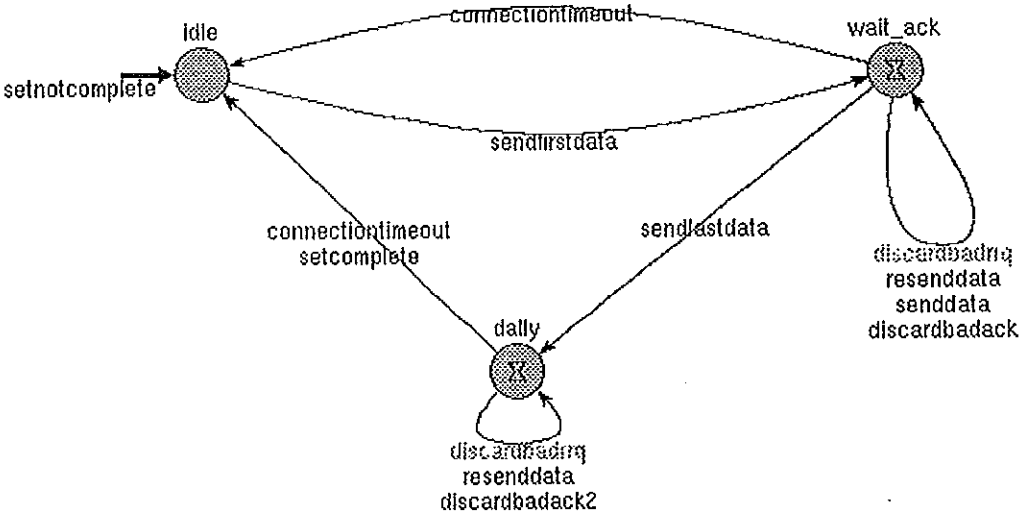


ภาพประกอบ 5-17 แสดงพฤติกรรม Reader (4) ในโมดูล Handler ของ Initiator



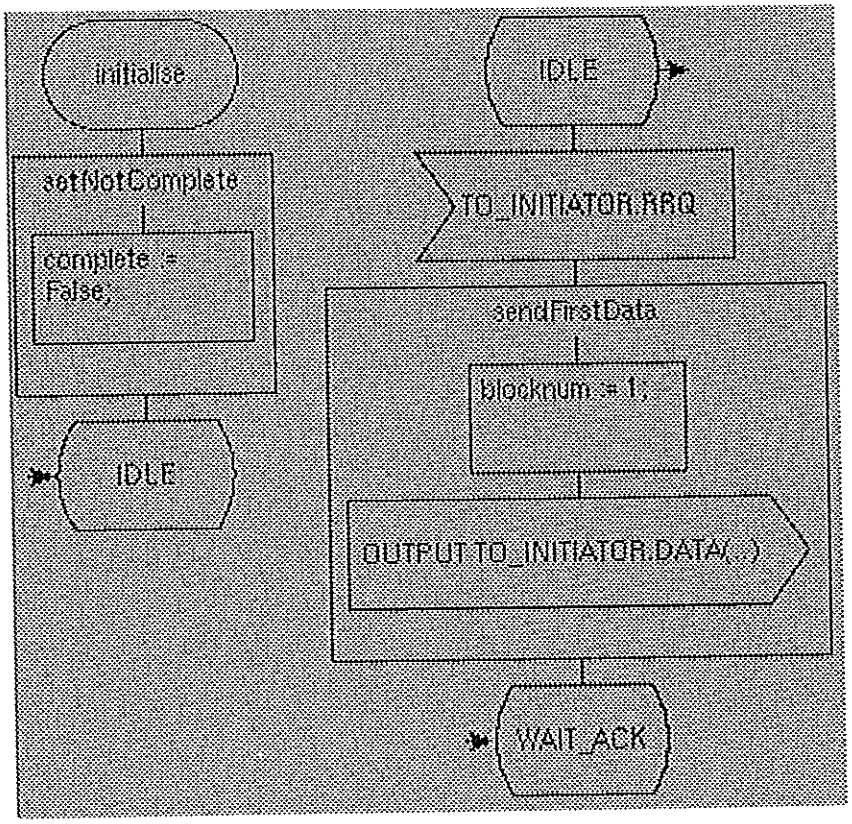
ภาพประกอบ 5-18 แสดงพฤติกรรม Reader (5) ในโมดูล Handler ของ Initiator

- พฤติกรรมโมดูลลูก Handler ของโมดูล Responder ที่แสดงพฤติกรรม Reader เมื่อโมดูล Initiator ร้องขอ READ_RQST โดยให้โมดูลลูก Handler มีพฤติกรรม Reader แล้วทางฝ่าย Responder ก็จะเรียกใช้โมดูลลูก Handler ของตนให้ทำงานในพฤติกรรม Reader เช่นกันแต่ Reader ของ Initiator กับ Reader ของ Responder จะทำงานแตกต่างกันเพราะเมื่อฝ่าย Initiator ส่ง READ_RQST ฝ่าย Responder ก็จะทำหน้าที่ส่งแพคเก็ตข้อมูลและคอยรับแพคเก็ตแสดงข่าวสาร ส่วนฝ่าย Initiator ก็จะคอยรับแพคเก็ตข้อมูลแต่ส่งแพคเก็ตแสดงข่าวสารไป ภาพประกอบ 5-19 แสดงการทำงานของโมดูล Handler ที่เป็น Reader ในโมดูล Responder

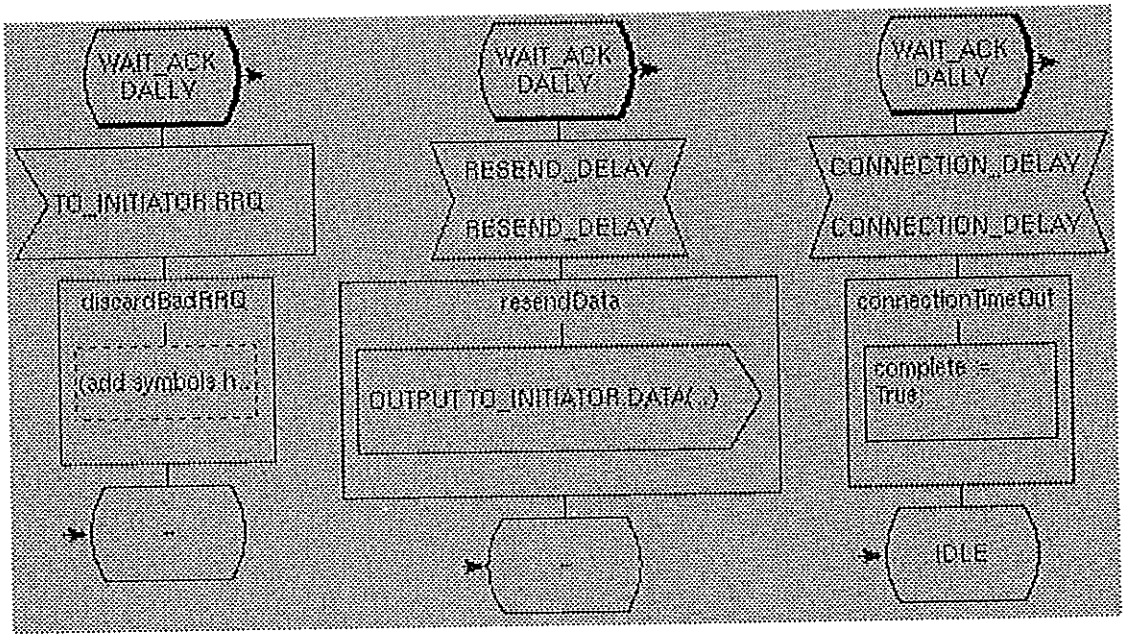


ภาพประกอบ 5-19 การทำงานของโมดูล Handler ที่เป็น Reader ในโมดูล Responder

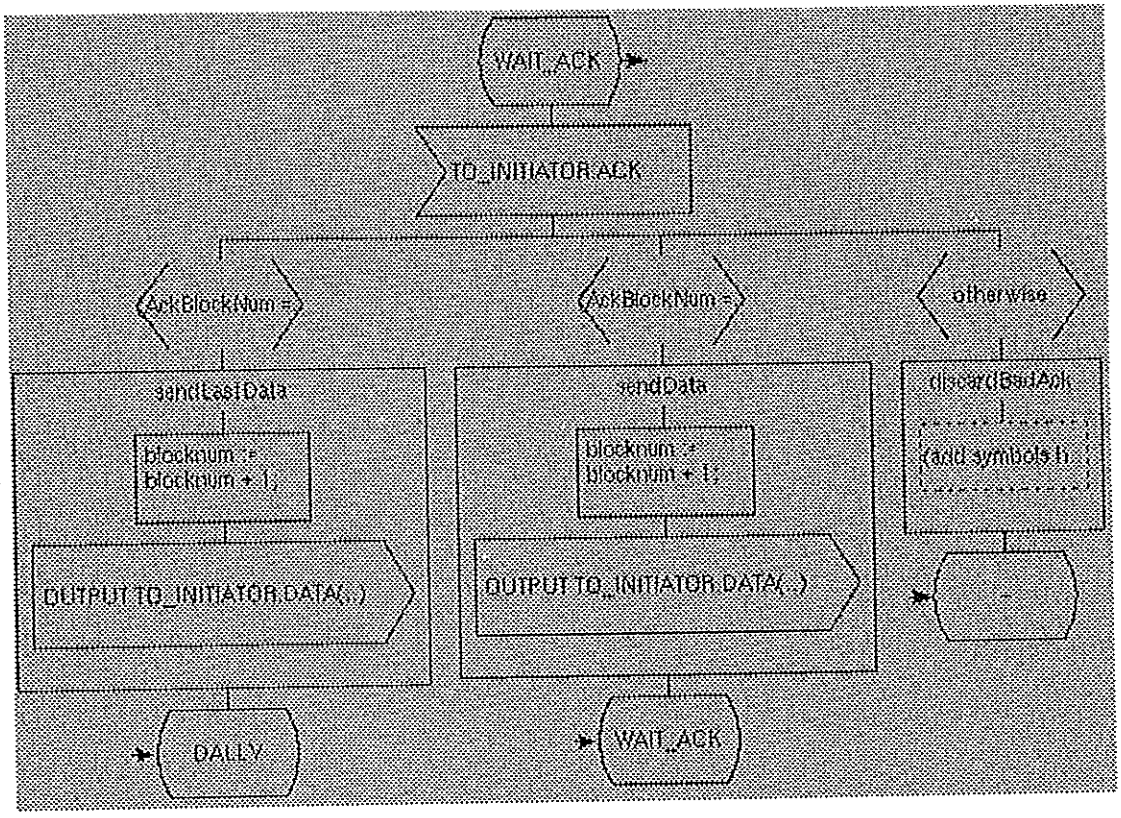
เมื่อนำมาเขียนพฤติกรรมของโมดูลลูก Handler ใน Responder ที่เป็น Reader ตามรูปแบบ Estelle/GR ก็จะเป็นดังภาพประกอบ 5-20 5-21 5-22 และ 5-23



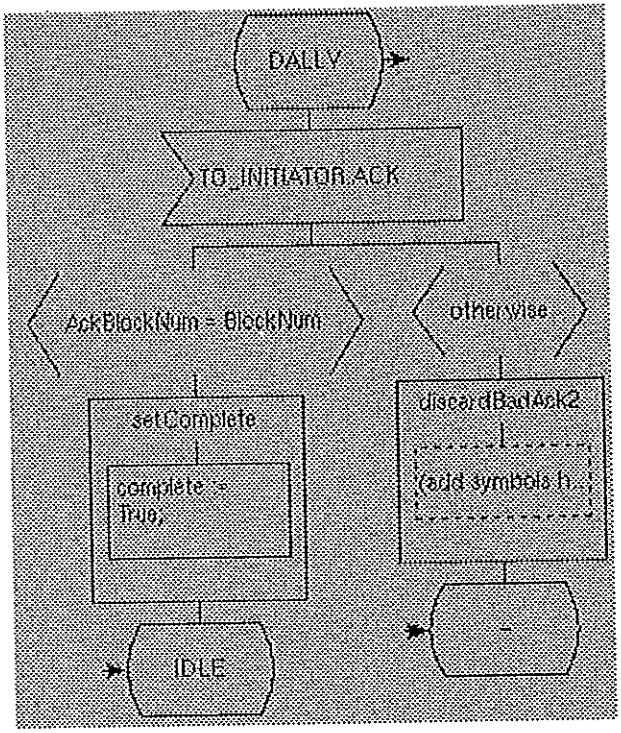
ภาพประกอบ 5-20 พฤติกรรม Reader (1) ในโมดูล Handler ของ Responder



ภาพประกอบ 5-21 พฤติกรรม Reader (2) ในโมดูล Handler ของ Responder

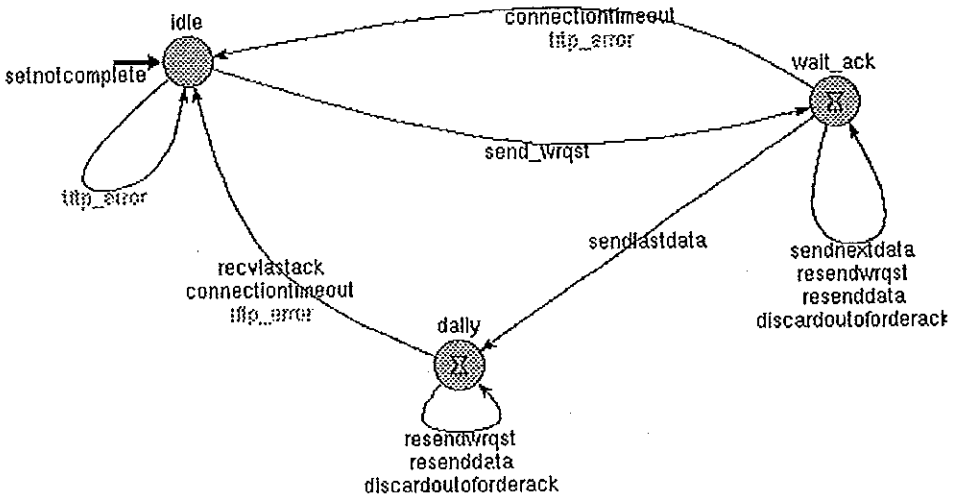


ภาพประกอบ 5-22 พฤติกรรม Reader (3) ในโมดูล Handler ของ Responder



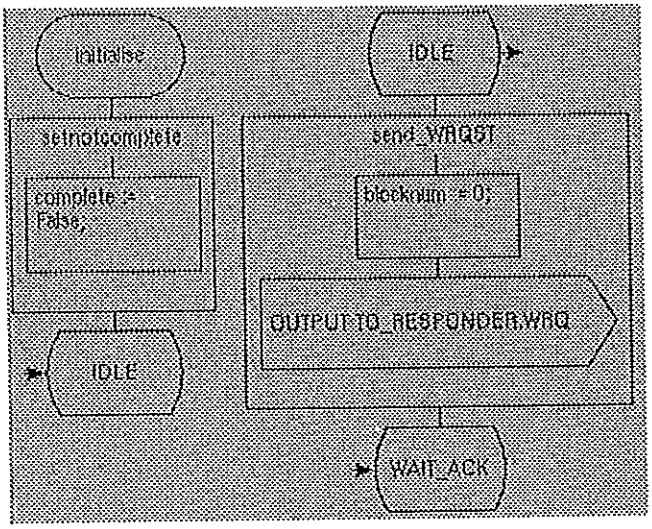
ภาพประกอบ 5-23 พฤติกรรม Reader (4) ในโมดูล Handler ของ Responder

- พฤติกรรมโมดูลลูก Handler ของ Initiator ที่แสดงพฤติกรรมเป็น Writer
  เมื่อโมดูล Initiator เรียกใช้โมดูลลูก Handler ให้ทำงานโดยแสดงพฤติกรรม Writer มีการทำงานดังภาพประกอบ 5-24

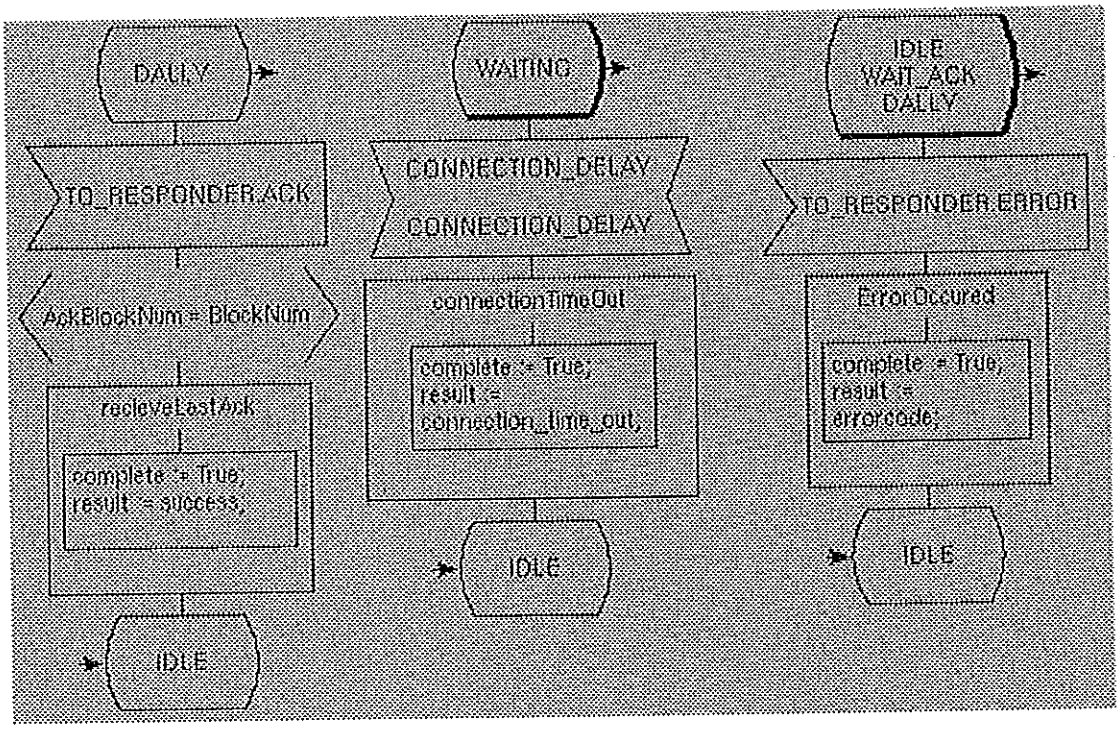


ภาพประกอบ 5-24 การทำงานของโมดูลลูก Handler ในโมดูล Initiator ที่เป็น Writer

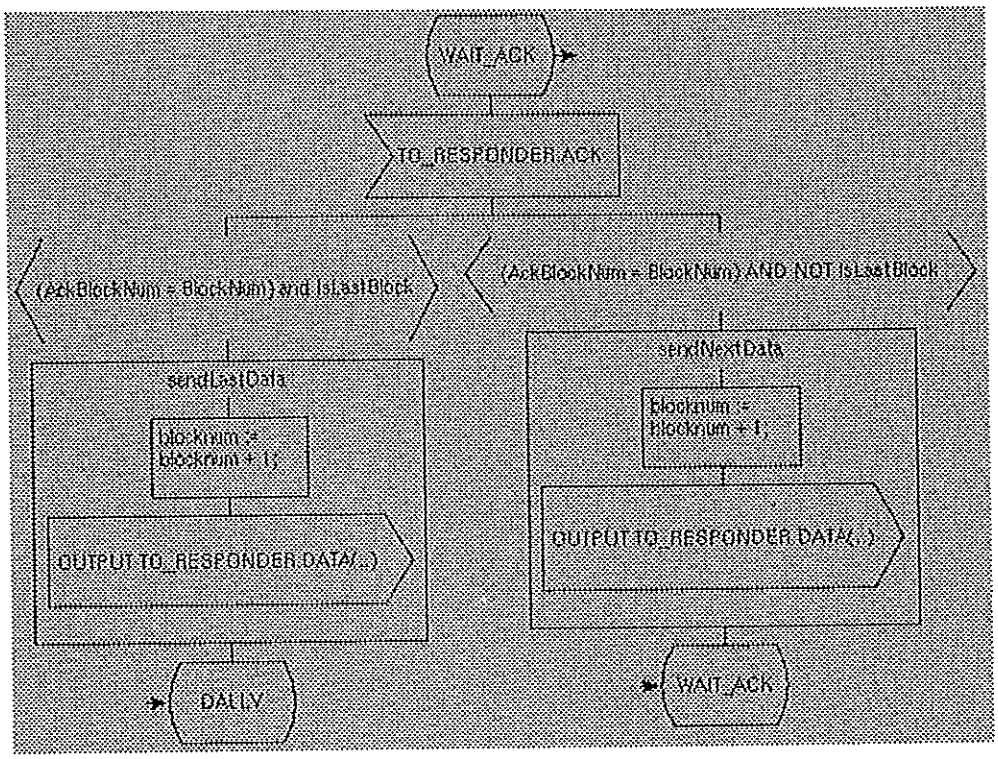
เมื่อนำมาเขียนตามรูปแบบ Estelle/GR ก็จะได้ดังภาพประกอบ 5-25 5-26 5-27 และ 5-28



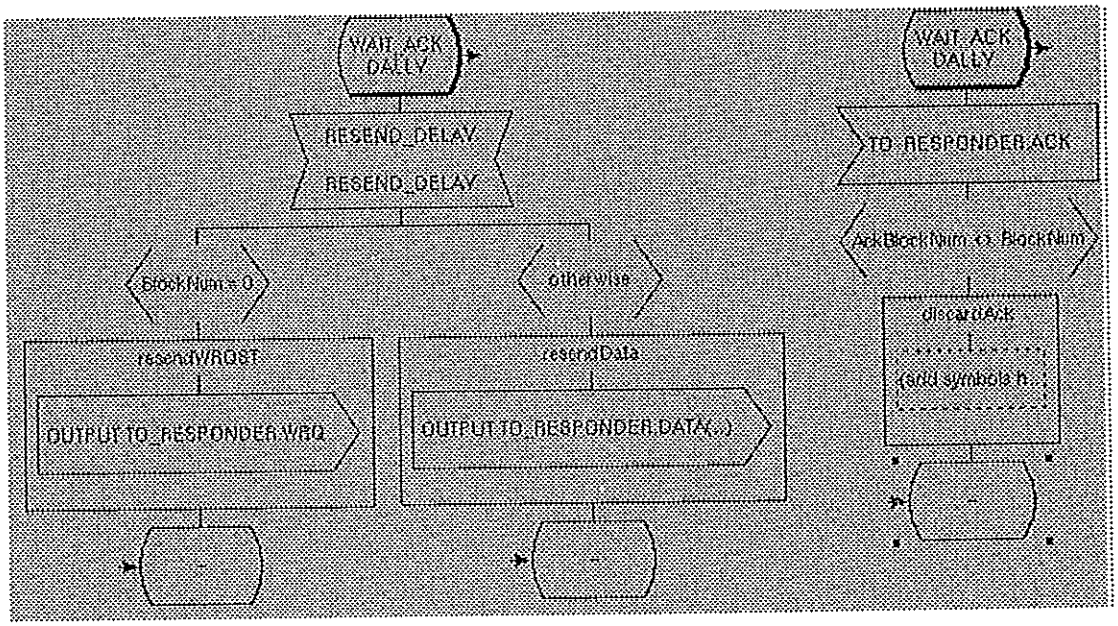
ภาพประกอบ 5-25 พฤติกรรม Writer (1) ในโมดูล Handler ของ Initiator



ภาพประกอบ 5-26 พฤติกรรม Writer (2) ในโมดูล Handler ของ Initiator

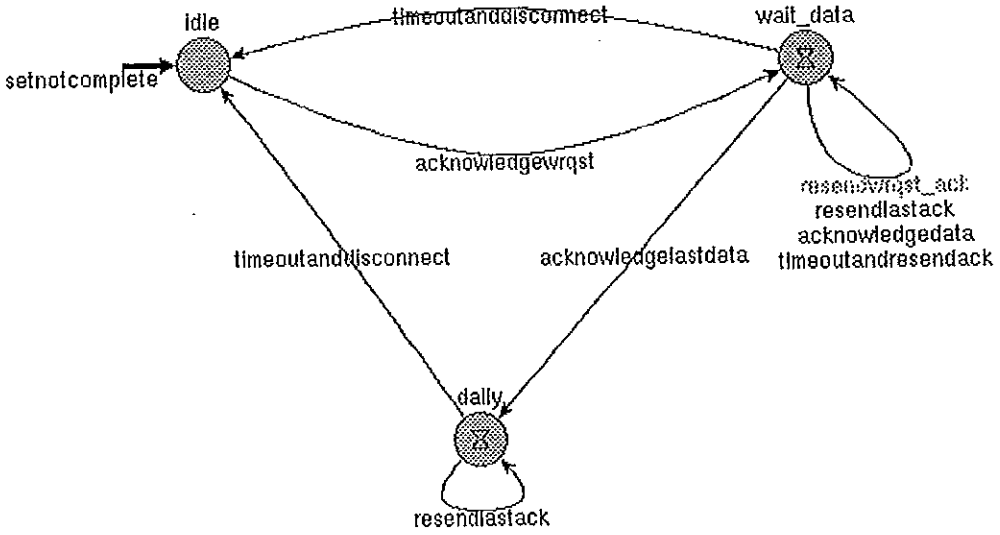


ภาพประกอบ 5-27 พฤติกรรม Writer (3) ในโมดูล Handler ของ Initiator



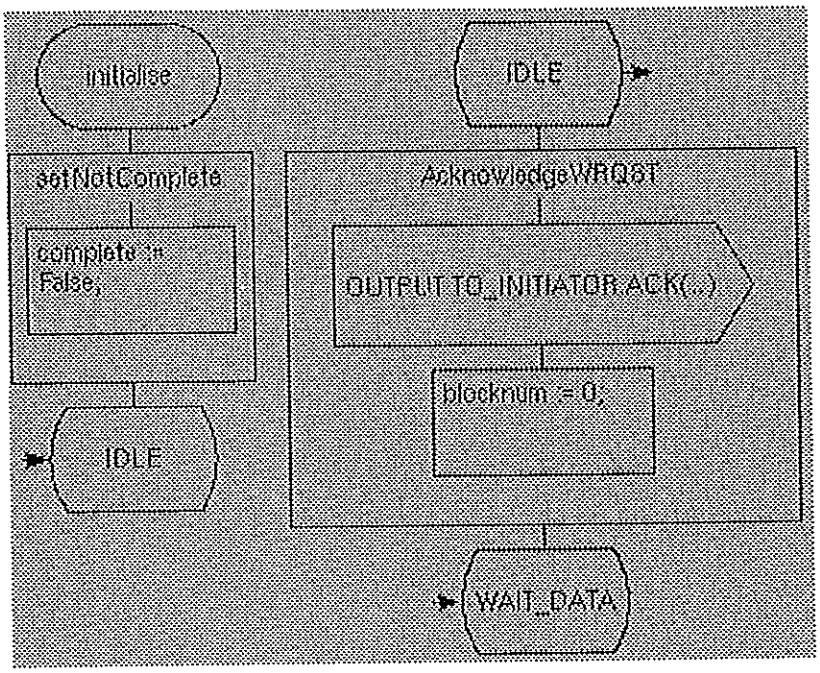
ภาพประกอบ 5-28 พฤติกรรม Writer (4) ในโมดูล Handler ของ Initiator

- พฤติกรรมโมดูลลูก Handler ของ Responder ที่แสดงพฤติกรรมเป็น Writer
  เมื่อโมดูล Responder ได้รับ WRITE_RQST ก็จะเรียกโมดูลลูก Handler ให้ทำงาน โดยแสดงพฤติกรรม Writer ให้สอดคล้องกับการทำงานของโมดูลลูก Handler ของ Initiator ที่แสดงพฤติกรรม Writer เช่นกัน ภาพประกอบ 5-29 แสดงการทำงานของโมดูลลูก Handler ของ Responder ที่เป็น Writer

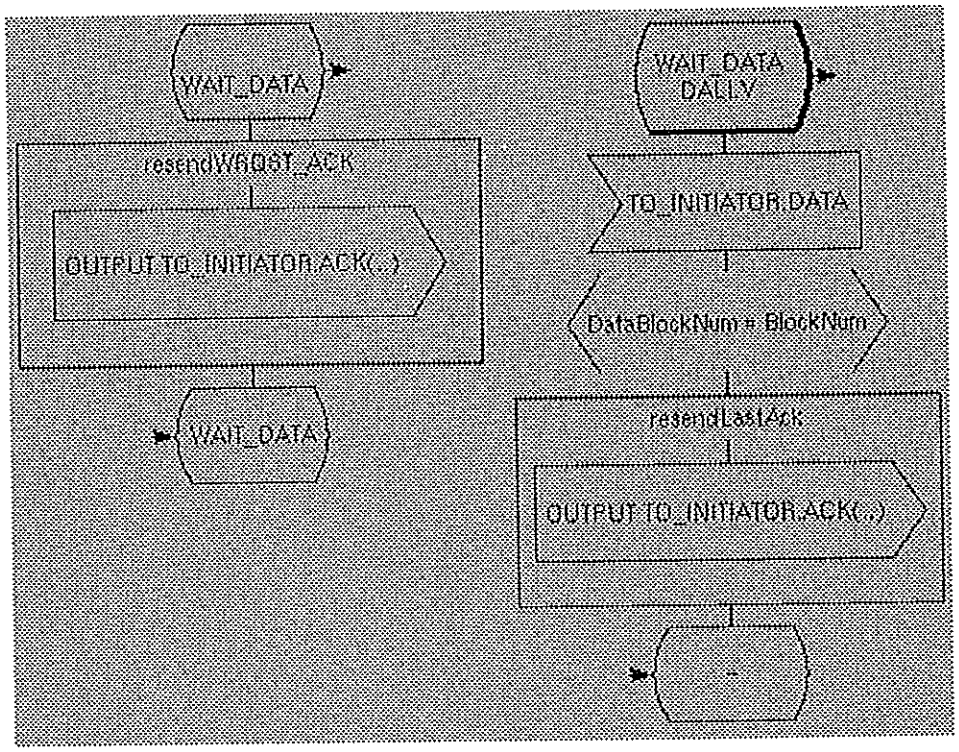


ภาพประกอบ 5-29 การทำงานของโมดูลลูก Hander ของ Responder ที่เป็น Writer

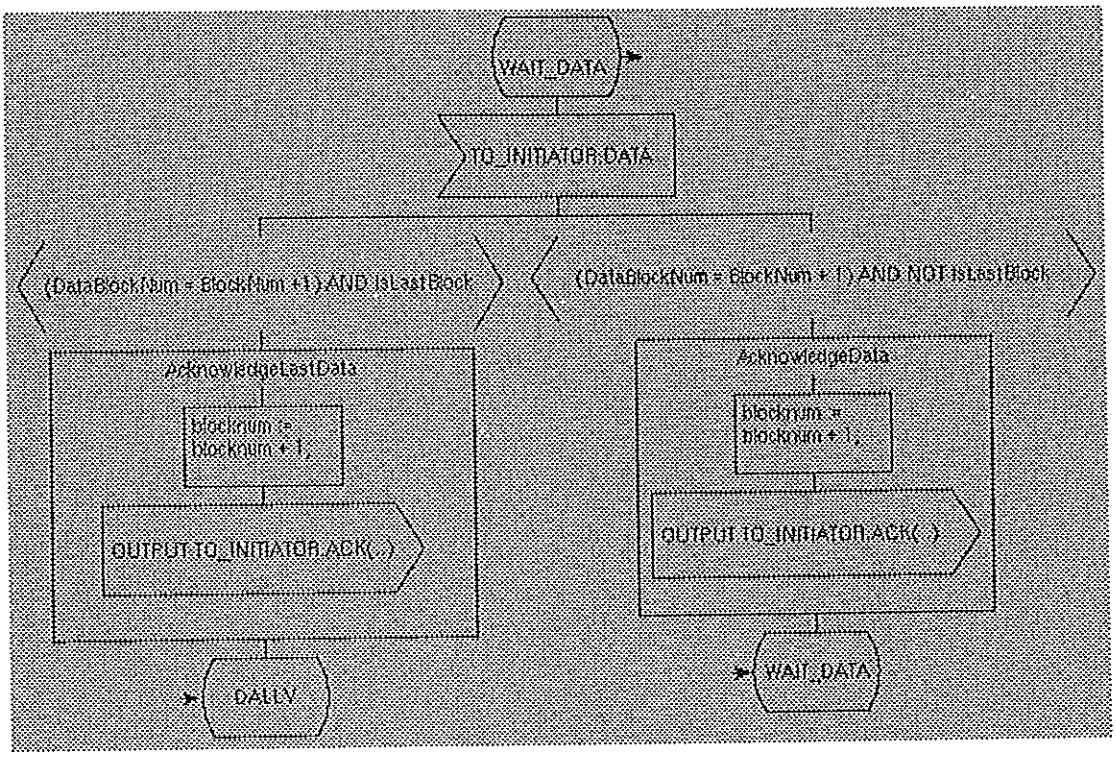
เมื่อนำพฤติกรรมของโมดูล Handler ใน Responder ที่เป็น Writer มาเขียนในรูปแบบ Estelle/GR ก็จะได้ดังภาพประกอบ 5-30 5-31 5-32 และ 5-33



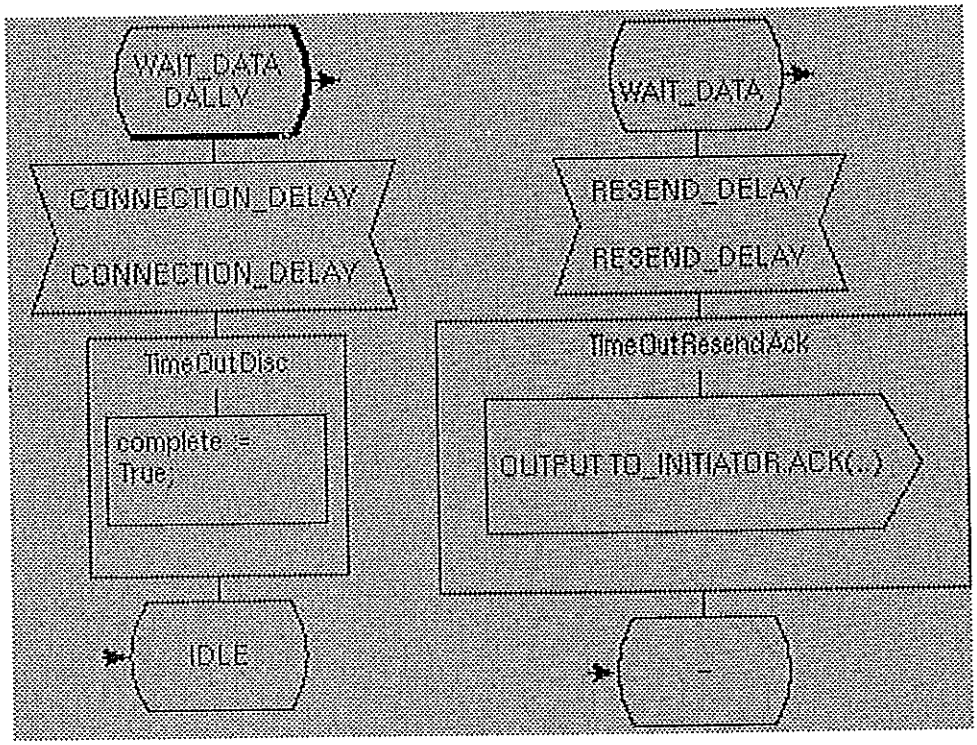
ภาพประกอบ 5-30 พฤติกรรม Writer (1) ในโมดูล Handler ของ Responder



ภาพประกอบ 5-31 พฤติกรรม Writer (2) ในโมดูล Handler ของ Responder



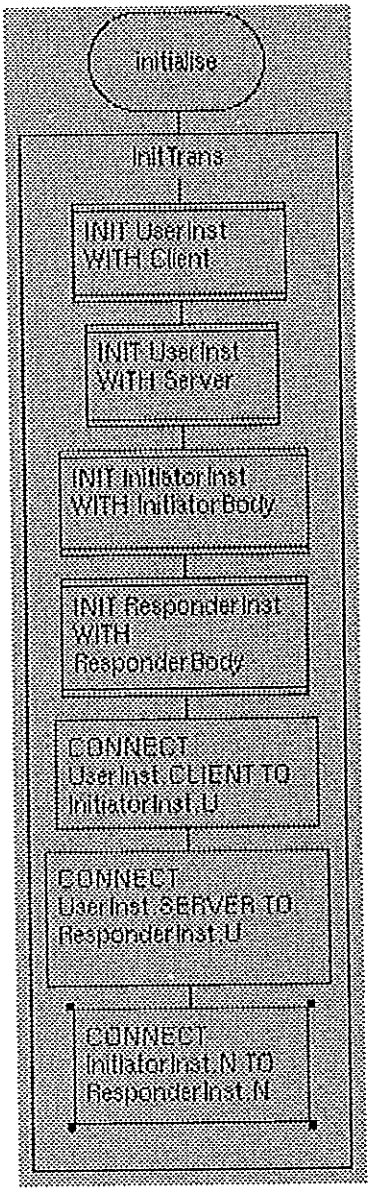
ภาพประกอบ 5-32 พฤติกรรม Writer (3) ในโมดูล Handler ของ Responder



ภาพประกอบ 5-33 พฤติกรรม Writer (4) ในโมดูล Handler ของ Responder

- พฤติกรรมโมดูลหลัก (Specification module)

  พฤติกรรมหรือการทำงานของโมดูลหลักเป็นการประกาศตัวแปรชนิดโมดูลต่าง ๆ และเรียกใช้โมดูลตามความเหมาะสม ทำการเชื่อมโยงจุดสื่อสารต่าง ๆ ระหว่างโมดูลด้วยคำสั่ง Connect หรือ Attach โดยการทำงานเหล่านี้จะกำหนดในส่วนการเปลี่ยนสถานะเริ่มต้น พฤติกรรมของโมดูลหลักของโปรโตคอล TFTP ตามรูปแบบ Estelle/GR เป็นดังภาพประกอบ 5-29



ภาพประกอบ 5-34 แสดงพฤติกรรมโมดูลหลัก TFTP

เมื่อเขียนข้อกำหนดในรูปแบบ Estelle/GR แล้วก็สามารถให้โปรแกรม Estelle Graphical Editor แปลงให้เป็นรูปแบบ Estelle/PR ได้ ซึ่งข้อกำหนดโปรโตคอล TFTP ที่สมบูรณ์ในรูปแบบ Estelle/PR เป็นดังในภาคผนวก ข

บทที่ 6

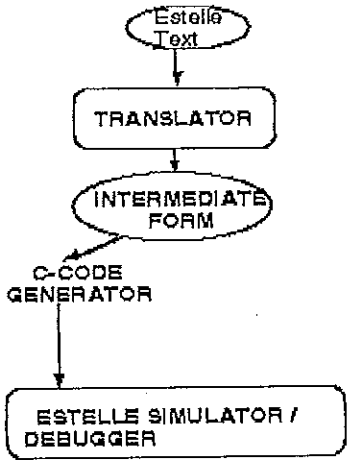
การใช้เครื่องมือ EDT ในการตรวจและจำลองการทำงานของโปรโตคอล TFTP

EDT (Estelle Development Toolsets) เป็นชุดของโปรแกรมที่ใช้สำหรับตรวจสอบข้อกำหนดโปรโตคอลที่เขียนขึ้นตามหลักภาษา Estelle และทำการจำลองการทำงานของโปรโตคอลโดยการตรวจสอบพฤติกรรมของโปรโตคอลที่ขึ้นว่าเป็นไปตามที่กำหนดหรือไม่ ซึ่งจะเป็นการตรวจสอบสถานะของโมดูลต่าง ๆ ที่จะเปลี่ยนไปเมื่อมีเงื่อนไขที่ถูกต้องในการเปลี่ยนสถานะ เครื่องมือที่ EDT ที่จะใช้ในการตรวจสอบและจำลองการทำงานของโปรโตคอลมีดังนี้

6.1 ส่วนประกอบของเครื่องมือ EDT

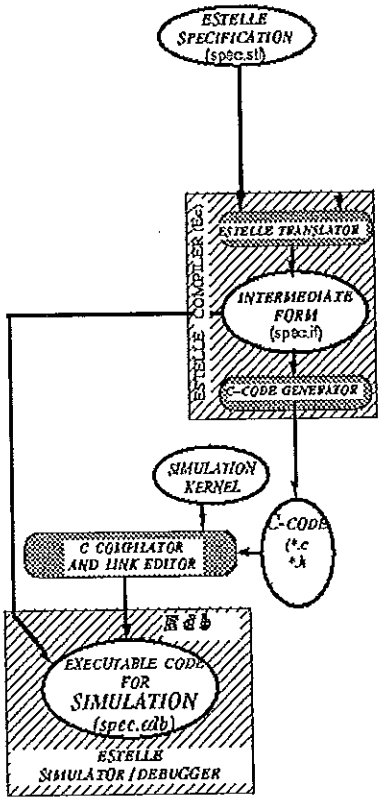
- Estelle Compiler (Ec) ใช้สำหรับตรวจสอบความถูกต้องโปรโตคอลว่าเป็นไปตามหลักภาษาของ Estelle หรือไม่ หากไม่ถูกก็จะแสดงข้อความผิดพลาดแจ้งให้ทราบ ถ้าหากข้อกำหนดนั้นถูกต้องตามหลักภาษาของ Estelle ก็จะทำการแปลข้อกำหนดนั้นให้อยู่ในรูปที่เรียกว่า Intermediate Form โดยหากแฟ้มที่เป็นข้อกำหนดโปรโตคอลชื่อ TFTP.stl ก็จะได้แฟ้มในรูป Intermediate Form ชื่อ TFTP.if ตัว Estelle Compiler เองจะประกอบไปด้วยเครื่องมือสองอย่างคือ ตัวแปร (Translator) กับ ตัวสร้างโค้ดภาษาซี (C-code Generator)
- Estelle Simulator/Debugger (Edb) ใช้สำหรับตรวจสอบการทำงานของโปรโตคอลหรือจำลองการทำงานของโปรโตคอลโดยเน้นถึงการตรวจสอบความถูกต้องของพฤติกรรมของโปรโตคอล ซึ่งหากข้อกำหนดนั้นผ่านการตรวจสอบและแปลเป็น Intermediate Form มาแล้วตัว Edb ก็จะสร้างชุดโค้ดเฉพาะสำหรับทำการจำลองการทำงานของโปรโตคอลในชื่อ TFTP.edb (เมื่อตั้งชื่อข้อกำหนดเป็น TFTP.stl)

ลักษณะการทำงานของเครื่องมือ EDT แสดงได้ดังภาพประกอบ 6-1 และ 6-2



ภาพประกอบ 6-1 แสดงส่วนประกอบของและการทำงานของ EDT

(ที่มา: Institut National des Telecommunications, 1999 : 6)



ภาพประกอบ 6-2 แสดงลักษณะการทำงานของส่วนประกอบต่าง ๆ ของ EDT

(ที่มา: Institut National des Telecommunications, 1999 : 6)

6.2 การใช้ EDT สำหรับตรวจสอบและจำลองการทำงานของโปรโตคอล TFTP

เมื่อได้ข้อกำหนดโปรโตคอล TFTP โดยเก็บไว้ในชื่อ TFTP.stl แล้วก็สามารถใช้เครื่องมือ EDT สำหรับตรวจสอบการทำงานของโปรโตคอลได้โดยเรียกใช้ส่วนประกอบต่าง ๆ ของ EDT โดยตรงจากเครื่องหมายพร้อมท์รับคำสั่งของระบบปฏิบัติการ เช่น ใช้คำสั่ง

```
$ ec TFTP.stl
```

เพื่อให้โปรแกรม ec ทำการตรวจสอบและสร้างแฟ้ม TFTP.if สำหรับใช้ในขั้นตอนต่อไป ซึ่งเมื่อผ่านขั้นตอนการแปลได้แฟ้ม TFTP.if แล้วก็สามารถเรียกใช้คำสั่ง edb เพื่อตรวจสอบการทำงานของโปรโตคอลได้ดังนี้

```
$ edb TFTP.stl
```

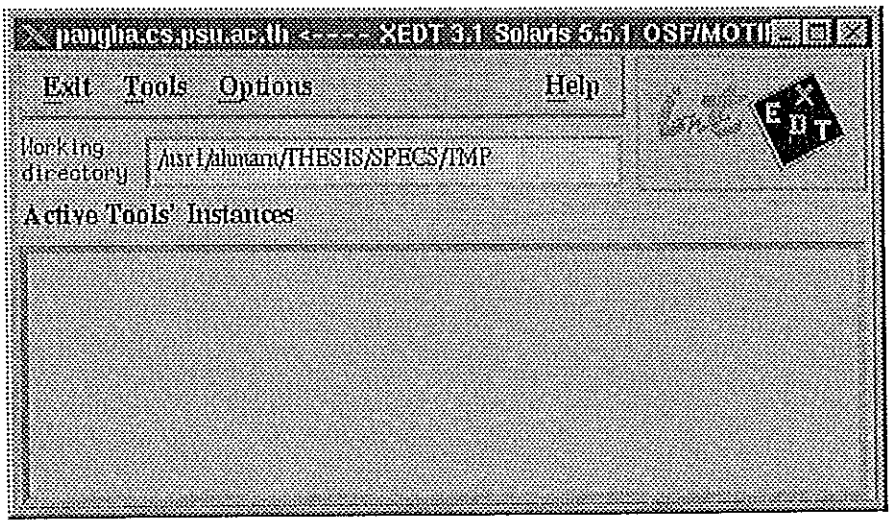
เมื่อผ่านขั้นตอนนี้แล้วก็จะเข้าสู่ส่วนคำสั่งของ edb สำหรับตรวจสอบการทำงานของโปรโตคอล โดยจะได้เครื่องหมายพร็อมท์ของ edb ดังนี้

```
EDB>
```

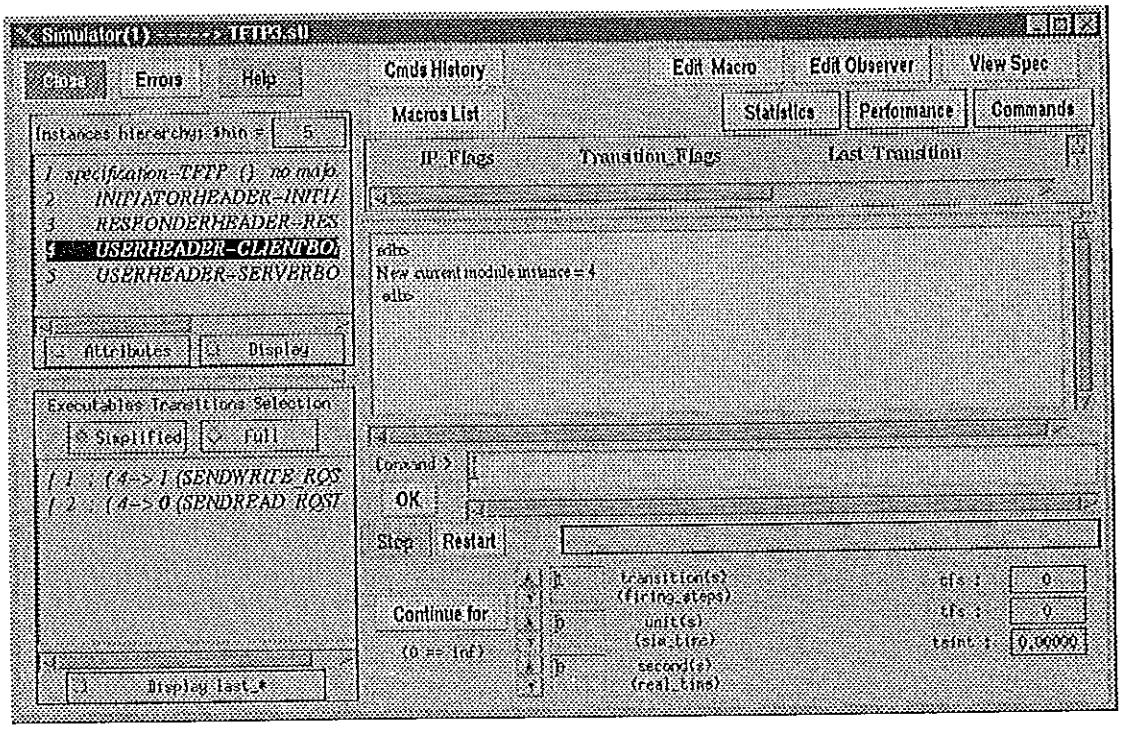
การใช้งานโดยผ่านคำสั่งดังกล่าวอาจไม่สะดวก จึงอาจใช้ส่วนที่เป็น Graphical User Interface ของ EDT สำหรับทำงานแทนการป้อนคำสั่งโดยตรงโดยเรียกใช้โปรแกรม xedt ดังนี้

```
$xedt
```

หลังจากนั้นก็จะได้หน้าต่างทำงานหลักของ EDT ดังภาพประกอบ 6-3 ส่วนหน้าต่างตรวจสอบการทำงานของโปรโตคอลเป็นดังภาพประกอบ 6-4



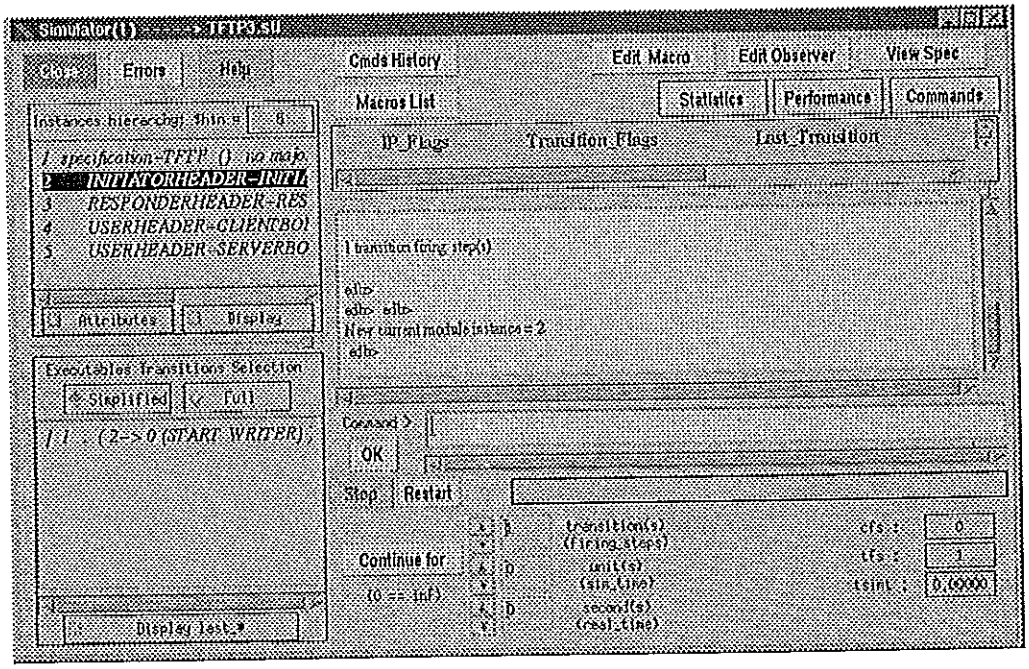
ภาพประกอบ 6-3 ส่วน Graphical User Interface ของ EDT



ภาพประกอบ 6-4 หน้าต่างตรวจสอบการทำงานของโปรโตคอล

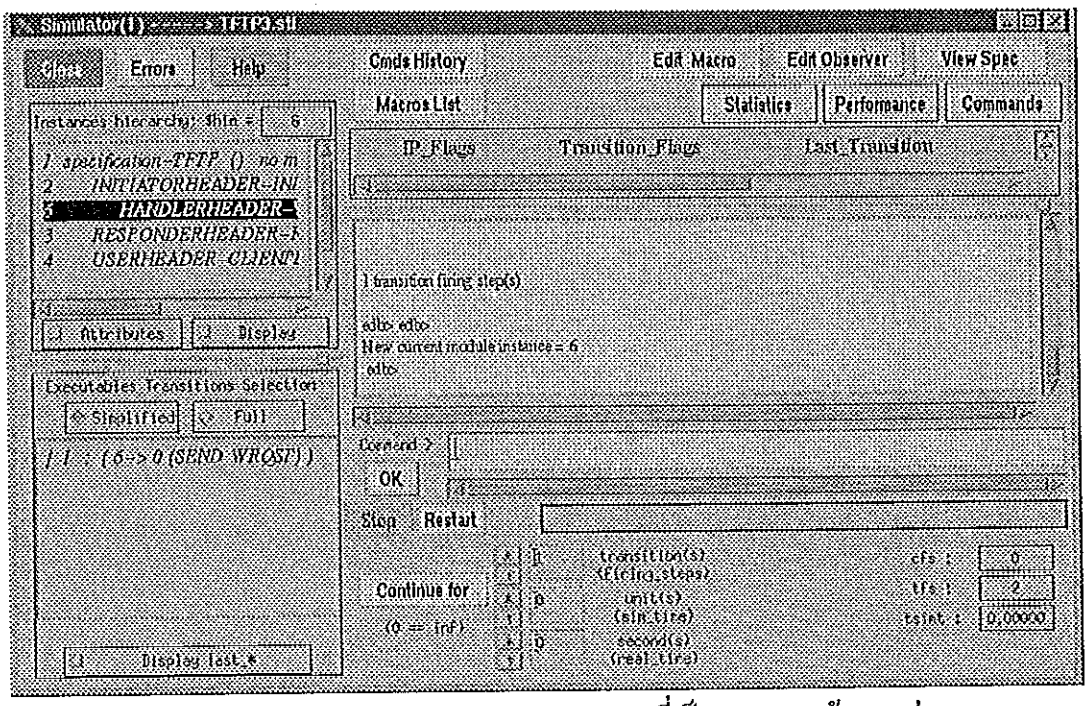
6.3 ผลลัพธ์จากการจำลองการทำงานของโปรโตคอล TFTP

ผลลัพธ์ที่ได้จากการจำลองการทำงานของโปรโตคอล TFTP ที่เขียนขึ้นตามข้อกำหนด Estelle นั้นเป็นไปตามข้อกำหนดที่เขียนขึ้นคือการทำงานของโปรโตคอลถูกต้องตามลักษณะการเปลี่ยนสถานะการทำงานของโมดูลต่าง ๆ ที่อยู่ในข้อกำหนด จากภาพประกอบ 6-4 เป็นผลที่ได้จากส่วนกำหนดค่าเริ่มต้นของโมดูลหลัก Specification จะเห็นว่าโมดูล User ที่เป็น CLENT พร้อมที่จะทำงานคือทำการส่ง Write request หรือ Read request ซึ่งเมื่อเลือก (4->1 SENDWRITE_RQST) จากรายการในกรอบ Executables Transitions Selection เมื่อส่ง Write request ก็จะได้ผลลัพธ์ดังภาพประกอบ 6-5

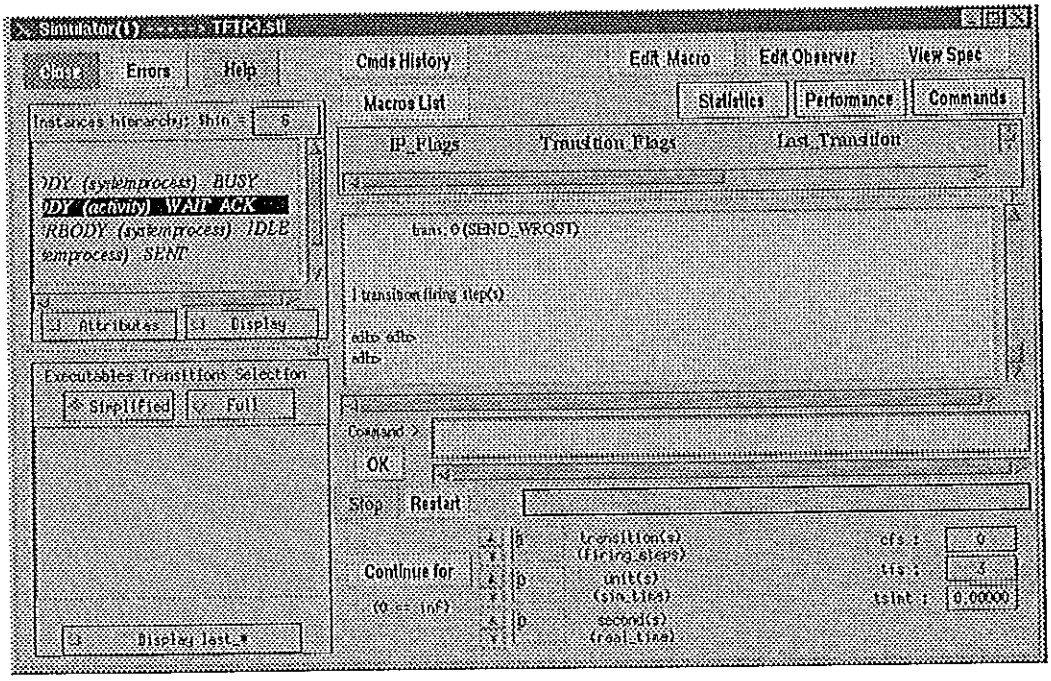


ภาพประกอบ 6-5 หน้าต่างแสดงโมดูล Initiator พร้อมที่จะเรียกโมดูล Handler ให้ทำหน้าที่ Writer

เมื่อเลือก (2->0 (START WRITER) เพื่อให้โมดูล Initiator ทำงานโดยเรียกใช้โมดูลลูก Handler ให้แสดงพฤติกรรม Writer นั้นโมดูล Initiator เองก็จะเปลี่ยนสถานะจาก IDLE สู่สถานะ BUSY และโมดูลลูก Handler ที่ทำหน้าที่เป็น Writer ก็พร้อมจะทำงานโดยส่ง Write request ต่อไปให้ฝ่ายตรงข้ามคือ Responder ภาพประกอบ 6-6 แสดงถึงโมดูลลูก Handler พร้อมจะทำการส่ง WRRQ ให้ฝ่าย Responder

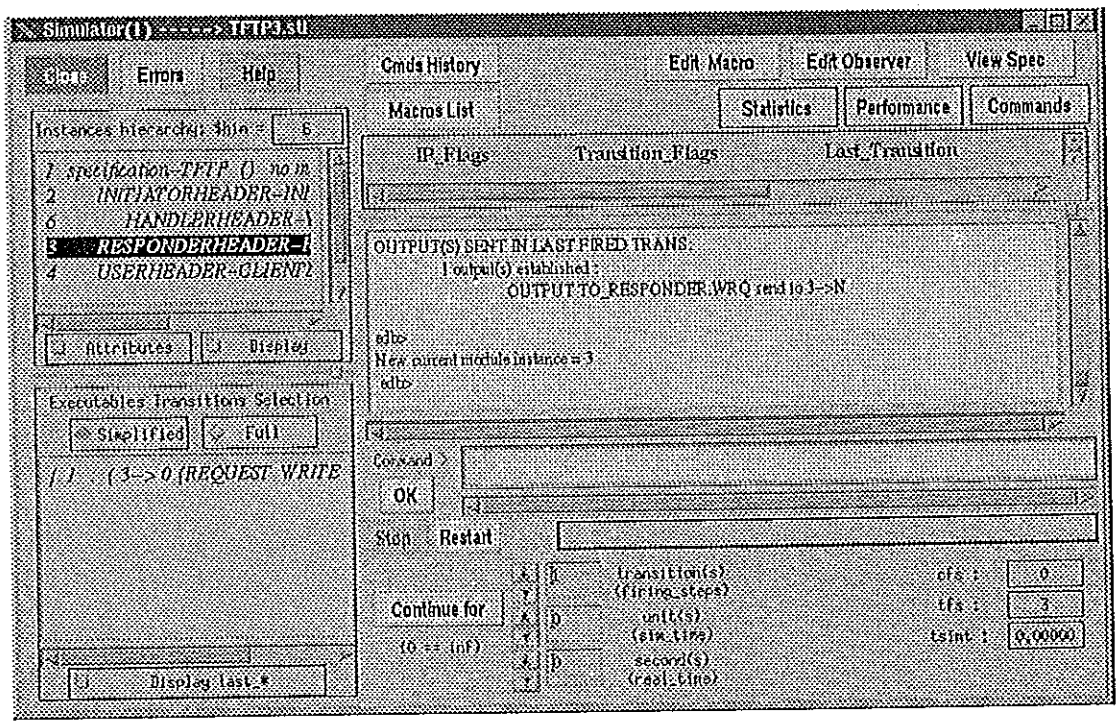


ภาพประกอบ 6-6 โมดูลถูก Handler ของ Initiator ที่เป็น Writer พร้อมจะส่ง WRQST



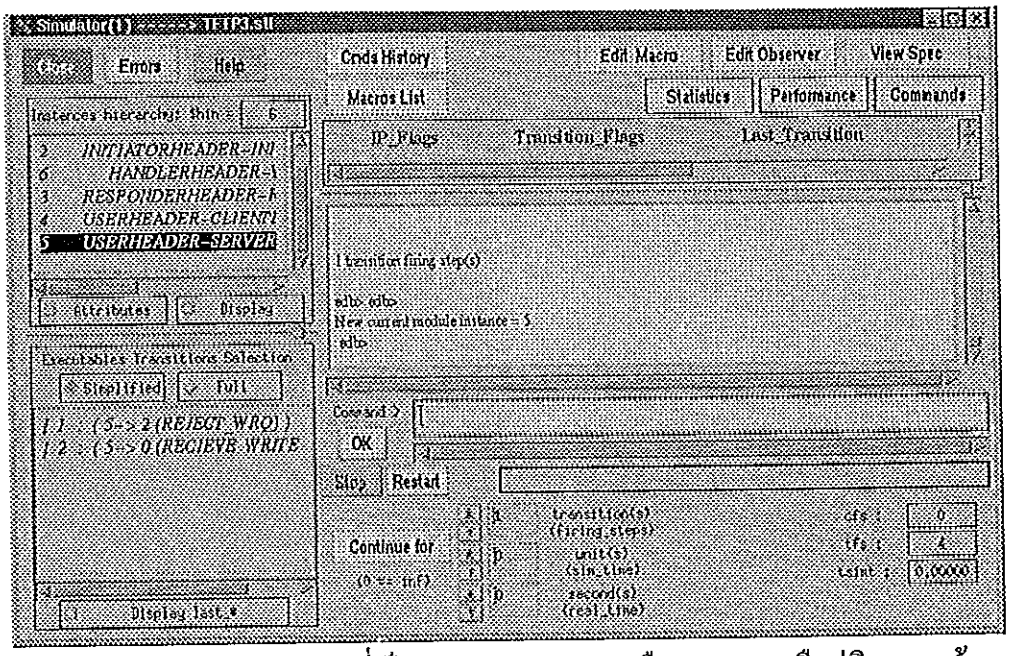
ภาพประกอบ 6-7 โมดูลถูก Handler ของ Initiator เปลี่ยนสถานะเป็น WAIT ACK

จากภาพประกอบ 6-7 จะเห็นได้ว่าโมดูล Handler ที่เป็น Writer เปลี่ยนสถานะเป็น WAIT ACK (จากบรรทัดที่มีแถบระบาย) สำหรับฝ่าย Responder เมื่อได้รับ Write request ก็จะส่งการร้องขออนุญาต Write request ไปให้โมดูล User ที่ทำหน้าที่เป็น Server ตรวจสอบได้โดยเลือก (3->0 (REQUEST WRITE PERMISSION) จากกรอบด้านล่างซ้าย



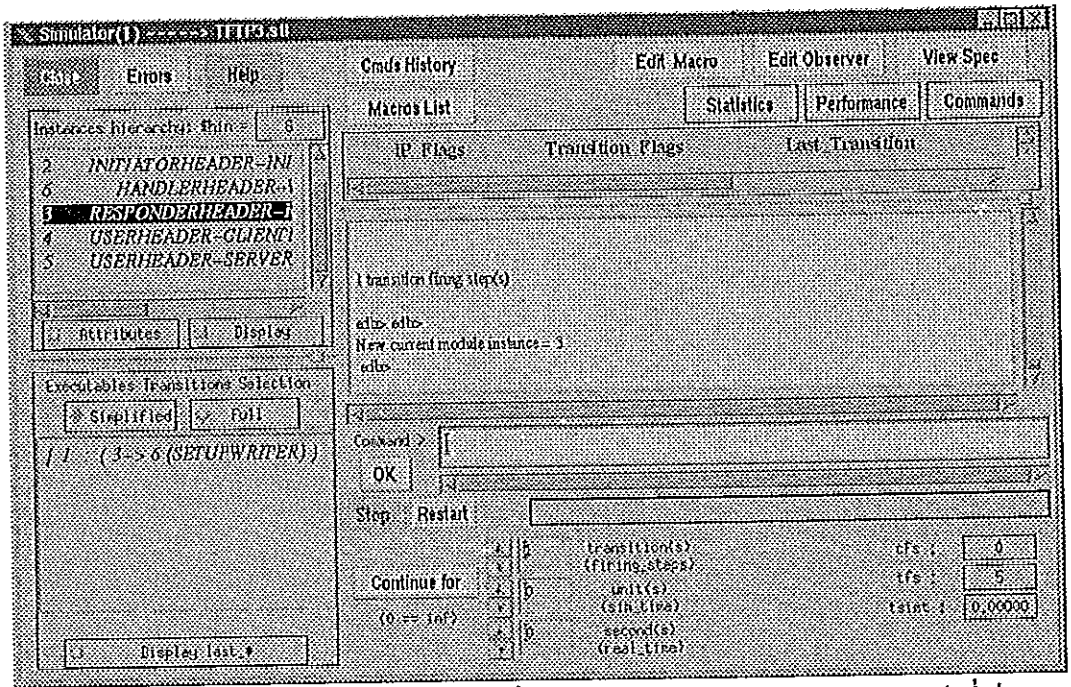
ภาพประกอบ 6-8 โมดูล Responder พร้อมที่จะทำการขออนุญาตจาก Server

จากภาพประกอบ 6-8 เมื่อเลือก (3->0 (REQUEST WRITE PERMISSION) โมดูล User ที่เป็น SERVER ก็พร้อมที่จะอนุญาตหรือปฏิเสธการร้องขอนี้ดังแสดงในภาพประกอบ 6-9



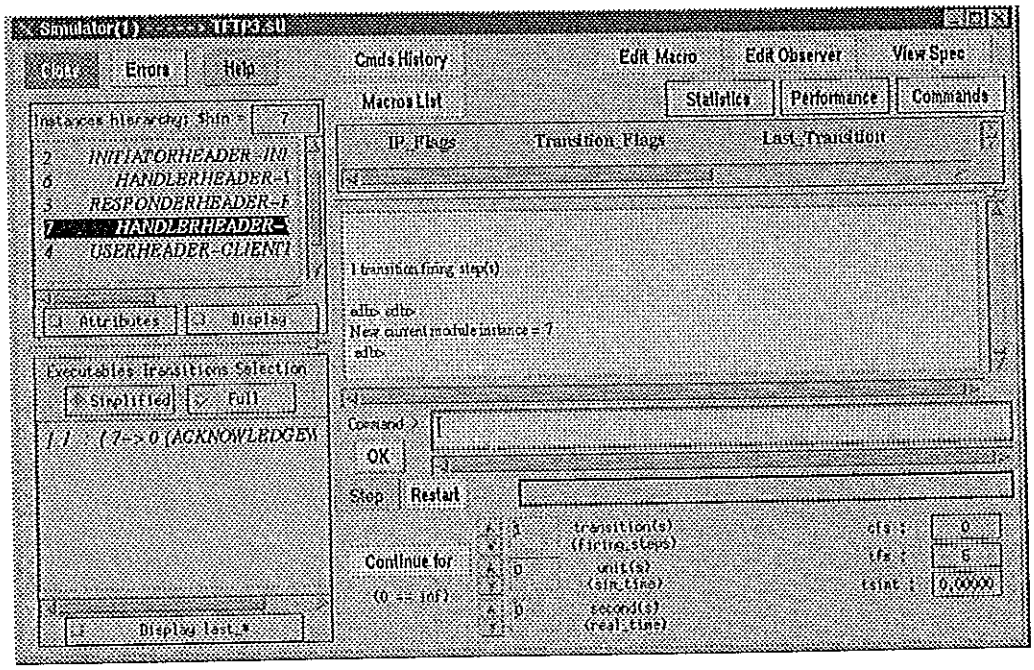
ภาพประกอบ 6-9 โมดูล User ที่เป็น SERVER สามารถเลือกอนุญาตหรือปฏิเสธการร้องขอ

จากภาพประกอบ 6-9 เมื่อเลือก (5-0 (RECEIVE WRITE)) เพื่อตอบรับการร้องขอก็จะได้ว่าโมดูล Responder ก็พร้อมจะเรียกใช้โมดูลลูก Handler ให้ทำหน้าที่รับแฟ้มจากฝ่าย Initiator ดังแสดงใน ภาพประกอบ 6-10

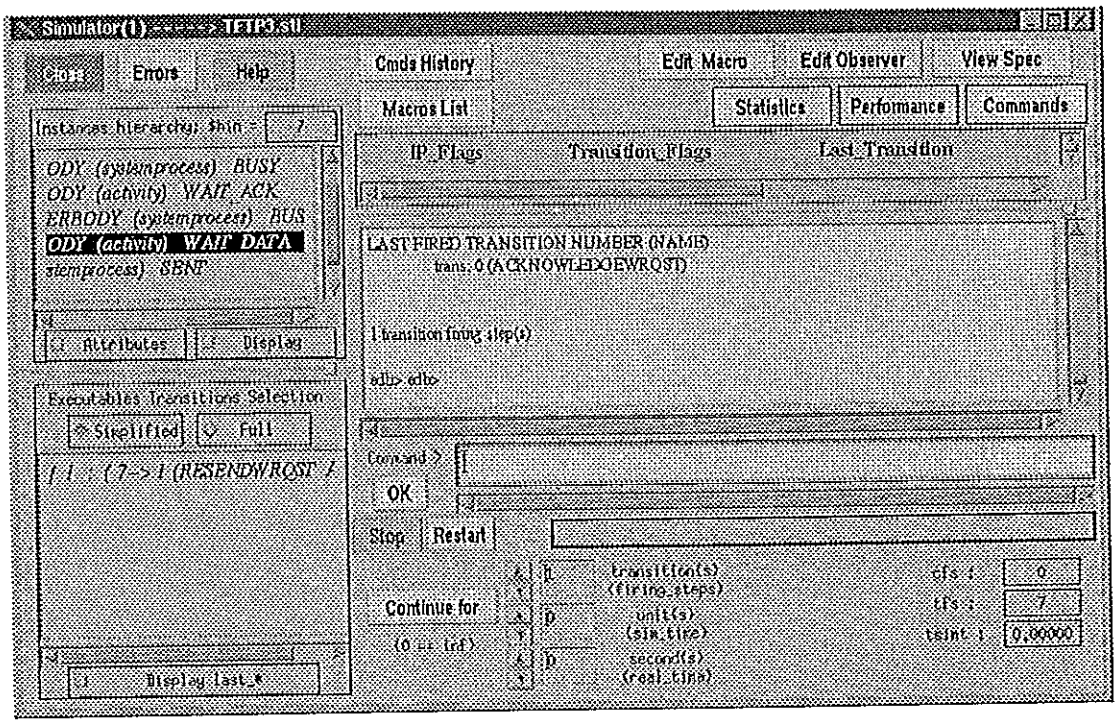


ภาพประกอบ 6-10 โมดูล Responder พร้อมที่จะเรียกใช้โมดูลลูก Handler ให้ทำหน้าที่เป็น Writer

จากภาพประกอบ 6-10 เมื่อเลือก (3->6 (SETUPWRITER)) ก็จะได้โมดูลลูก Handler ที่จะทำหน้าที่เป็น Writer อยู่ภายใต้โมดูล Responder ดังภาพประกอบ 6-11

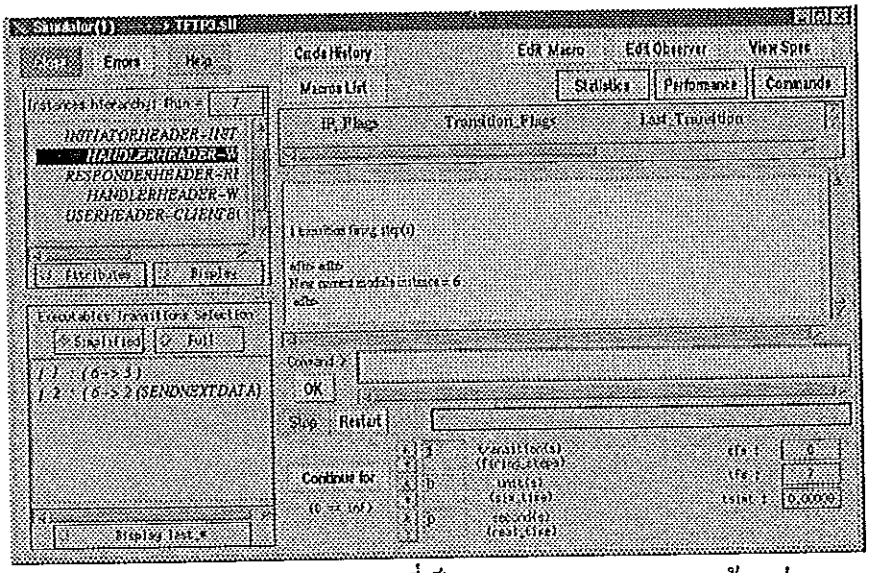


ภาพประกอบ 6-11 โมดูลลูก Handler ถูกสร้างขึ้นมาและพร้อมจะส่ง ACK

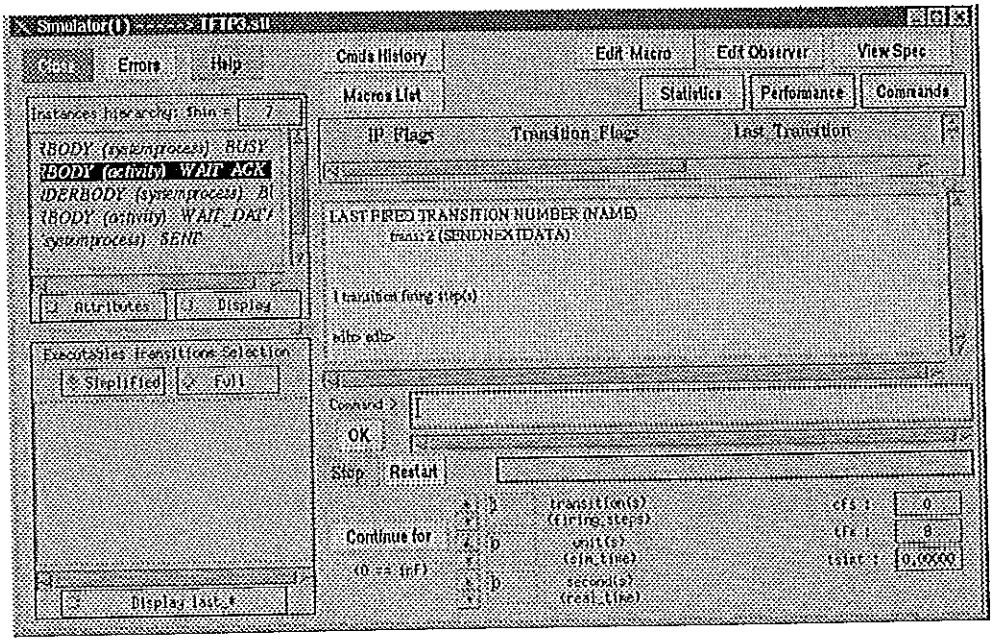


ภาพประกอบ 6-12 โมดูล Handler ที่เป็น Writer เปลี่ยนเป็น WAIT DATA เมื่อส่ง ACK เสร็จ

ทางฝ่าย Writer ของ Initiator เมื่อได้รับ ACK ก็สามารถที่จะส่งแพคเก็ตข้อมูลได้ดังแสดงในภาพประกอบ 6-13

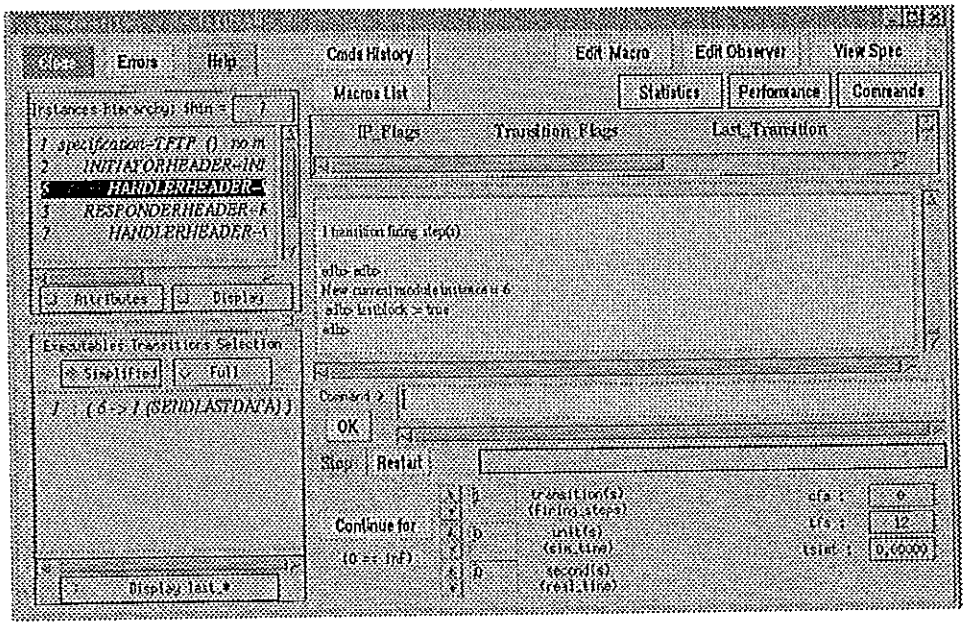


ภาพประกอบ 6-13 โมดูล Handler ที่เป็น Writer ของ Initiator พร้อมส่ง DATA เมื่อได้รับ ACK



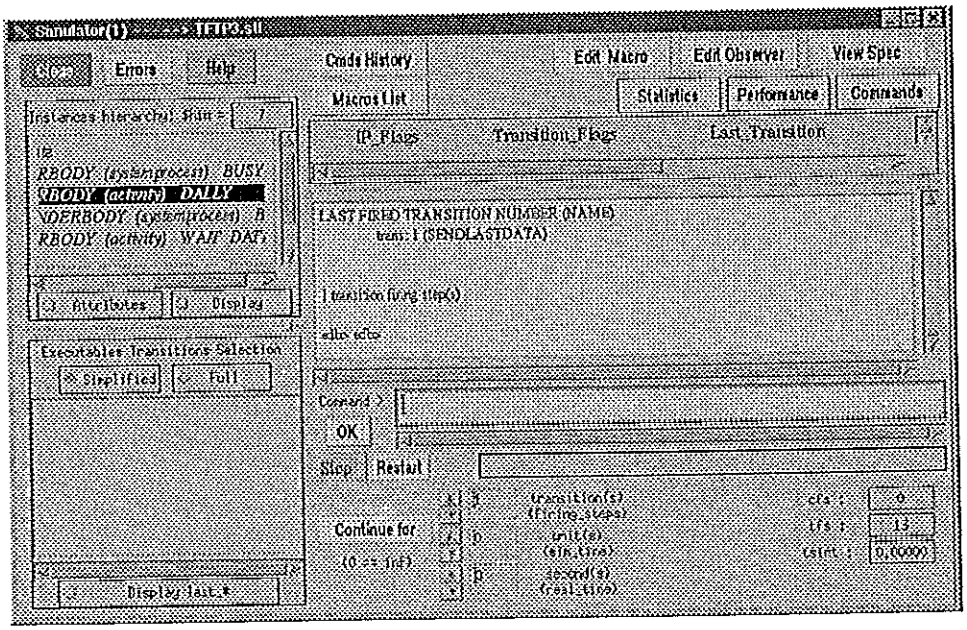
ภาพประกอบ 6-14 โมดูล Handler ของ Initiator เข้าสู่สถานะ WAIT ACK เมื่อส่ง DATA เสร็จ

แต่หากกำหนดให้ข้อมูลที่ส่งเป็นแพคเก็ตสุดท้ายโดยกำหนดให้ตัวแปร lastblock มีค่าเป็นจริงก็จะได้ว่าโมดูล Handler นี้ก็จะทำงานในส่วนของการเปลี่ยนสถานะที่ชื่อ (6->1 (SENDLASTDATA)) ดังภาพประกอบ 6-15

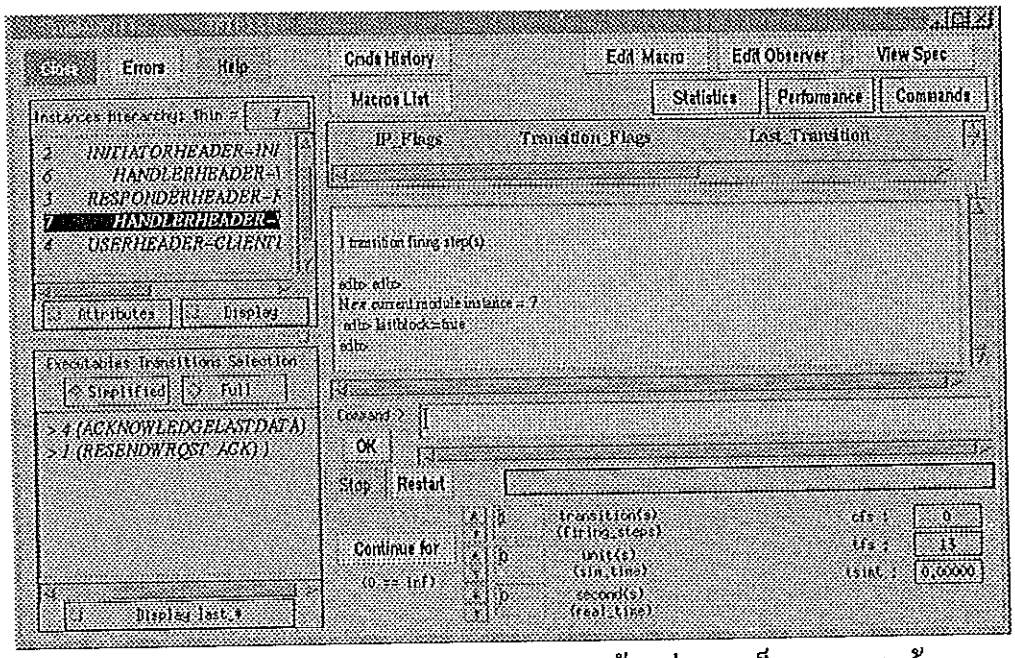


ภาพประกอบ 6-15 โมดูลลูก Handler ของ Initiator พร้อมส่งแพคเก็ตสุดท้าย

จากภาพประกอบ 6-15 เมื่อสั่งให้โมดูลนี้ทำงานโดยส่งแพคเก็ตสุดท้าย (เลือกรายการ (6->1 (SENDLASTDATA))) ก็จะเข้าสู่สถานะ DALLY ซึ่งเป็นสถานะที่แสดงว่าได้ส่งข้อมูลแพคเก็ตสุดท้ายแล้วและรอรับข้อมูล ACK จากฝ่ายตรงข้ามดังแสดงในภาพประกอบ 6-16

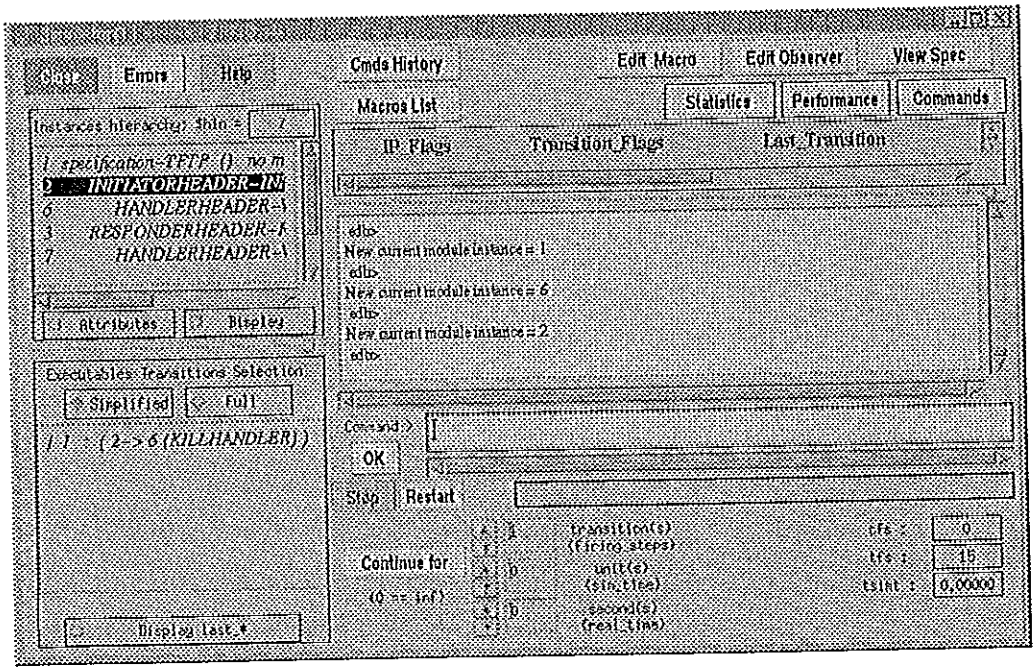


ภาพประกอบ 6-16 โมดูล Handler ของ Initiator เข้าสู่สถานะ DALLY



ภาพประกอบ 6-17 โมดูล Handler ของ Responder พร้อมส่งแพคเก็ต ACK สุดท้าย

จากภาพประกอบ 6-17 เมื่อได้รับแพคเก็ตข้อมูลชิ้นสุดท้ายฝ่าย Responder ก็จะส่งแพคเก็ต ACK สุดท้ายให้ทราบเช่นกัน ซึ่งเมื่ออีกฝ่ายได้รับ ACK สุดท้ายก็พร้อมสำหรับหยุดทำงานดังภาพประกอบ 6-18



ภาพประกอบ 6-18 โมดูล Initiator พร้อมหยุดโมดูลลูก Handler เมื่อได้รับ ACK สุดท้าย

บทที่ 7

สรุป ปัญหา และข้อเสนอแนะ

ในบทนี้ กล่าวถึงข้อสรุปและผลที่ได้จากการดำเนินการวิจัย และกล่าวถึงปัญหาตลอดจนอุปสรรคที่เกิดขึ้นในขณะทำการวิจัย ในหัวข้อสุดท้ายเป็นการให้ข้อเสนอแนะแก่ผู้วิจัยที่สนใจที่จะนำงานวิจัยไปพัฒนาต่อไป

7.1 สรุปผลการวิจัย

งานวิทยานิพนธ์นี้ได้ศึกษาการทำงานของโปรโตคอล TFTP และเขียนข้อกำหนดโปรโตคอลนี้ตามแบบภาษา Estelle โดยใช้เครื่องมือ Estelle Graphical Editor ช่วนในการเขียนข้อกำหนดในรูปแบบเชิงสัญลักษณ์แล้วจึงใช้เครื่องมือนี้แปลงข้อกำหนดจากรูปแบบเชิงสัญลักษณ์ให้เป็นตามแบบภาษา Estelle เชิงอักขระซึ่งจะนำไปใช้ในการตรวจสอบความถูกต้องของข้อกำหนดและจำลองการทำงานของโปรโตคอลด้วยเครื่องมือ EDT อีกทีหนึ่ง โดยสรุปแล้วผลที่ได้จากการทำวิทยานิพนธ์นี้มีดังนี้

1. ได้ข้อกำหนดโปรโตคอล TFTP ในรูปแบบสัญลักษณ์ที่เรียกว่า Estelle/GR โดยใช้เครื่องมือ Estelle Graphical Editor ช่วยในการเขียน ซึ่งทำให้การเขียนข้อกำหนดง่ายขึ้น มองภาพการทำงานโดยรวมของโปรโตคอลง่ายกว่าการอ่านจากข้อกำหนดที่เขียนในเชิงอักขระ
2. ได้ข้อกำหนดโปรโตคอล TFTP ตามแบบภาษา Estelle ในเชิงอักขระที่เรียกว่า Estelle/PR ซึ่งได้จากการใช้เครื่องมือ Estelle Graphical Editor เปลี่ยนจากรูปแบบ Estelle/GR ที่มีอยู่ให้เป็นข้อกำหนดในรูปแบบ Estelle/PR
3. ได้ตรวจสอบความถูกต้องของโปรโตคอล TFTP ในรูปแบบ Estelle/PR ว่าถูกตามหลักภาษา Estelle โดยใช้เครื่องมือ EDT
4. ได้ตรวจสอบและจำลองการทำงานของโปรโตคอล TFTP ที่ได้ในรูปภาษา Estelle โดยใช้เครื่องมือ EDT ซึ่งการทำงานของโปรโตคอล TFTP ที่ได้ก็ทำงานถูกต้องตามพฤติกรรมที่ควรจะเป็น
5. สามารถนำข้อกำหนดที่ได้ไปใช้ประกอบในการเรียนการสอนในเนื้อหาวิชาบางส่วนของวิชาที่เกี่ยวข้องกับโปรโตคอลได้

7.2 ปัญหาและอุปสรรค

1. สิ้นเปลืองเวลาที่ใช้ในการศึกษารูปแบบของ Estelle มาก

2. เครื่องมือหรือโปรแกรม EDT ที่ใช้ต้องทำการขอแฟ้ม edt.dat จากผู้ผลิตทุกครั้งเมื่อครบเวลาหนึ่งเดือน โดยขอได้เพียงครั้งเดียวเท่านั้น ซึ่งในการทำวิจัยครั้งนี้ต้องใช้เวลาหลายเดือนจึงต้องขอหลายครั้งทำให้ผิดกฎที่ฝ่ายเจ้าของโปรแกรมตั้งไว้

3. ติดต่อขอโปรแกรมเครื่องมือ EDT และ Estelle Graphical Editor โดยใช้โปรแกรม ftp และ telnet ผ่านทางเครือข่ายภายในของมหาวิทยาลัยไม่ได้อยู่ช่วงหนึ่งเนื่องจากทางหน่วยคอมพิวเตอร์ของมหาวิทยาลัยไม่อนุญาตให้ติดต่อกับคอมพิวเตอร์นอกเครือข่ายของมหาวิทยาลัยเพื่อรักษาความปลอดภัยของเครือข่าย

4. การออกแบบโปรโตคอลตามหลักภาษา Estelle ให้อยู่ในรูปหน่วยย่อยโมดูลต่าง ๆ เป็นเรื่องยุ่งยากและใช้เวลาเยอะเพราะต้องแสดงรายระเอียดทั้งผู้ให้บริการและผู้ขอบริการ

7.3 ข้อเสนอแนะ

1. การทำงานของโปรแกรม Estelle Graphical Editor ในส่วนของการอ่านแฟ้มที่อยู่ในรูป Estelle/PR มาเป็น Estelle/GR ยังทำไม่ได้ หากผู้ที่สนใจในการเขียนข้อกำหนดโปรโตคอลตามรูปแบบภาษา Estelle ต้องการโปรแกรมนี้และต้องการให้มีความสามารถในส่วนดังกล่าวต้องรอรุ่นใหม่ที่จะออกมาในไม่ช้า

2. ยังมี FDT ที่น่าสนใจอีกอย่างคือ LOTOS หากผู้ใดสนใจอาจศึกษาเทคนิคนี้ดูได้ แต่ยังไม่มีเครื่องมือสนับสนุนเหมือนอย่าง Estelle

3. ควรใช้เครื่องมือ EDT ที่เป็นชุดสมบูรณ์จะได้ไม่ต้องขอแฟ้มลิขสิทธ์ทุก ๆ ครั้งเมื่อหมดอายุการใช้งานและการทำงานก็จะมีความสะดวกมากกว่า

4. การเขียนข้อกำหนดโปรโตคอลแตกต่างจากการเขียนโปรแกรม โดยจะต้องเข้าใจการทำงานของโปรโตคอลให้ดีทั้งฝ่ายส่งและฝ่ายรับ เพราะการที่ผู้พัฒนาโปรแกรมขึ้นใช้งานบนโปรโตคอลนั้นจะทำงานได้สมบูรณ์หรือไม่สมบูรณ์อยู่ที่ว่าโปรโตคอลที่ออกแบบมานั้นทำงานถูกต้องตามหลักเกณฑ์ที่วางไว้หรือตามข้อกำหนดที่เขียนขึ้นหรือไม่

บรรณานุกรม

ISO 7185-1983. 1983. International Organization for Standardization, Specification for Computer Programming Language Pascal.

ISO 9074. 1989. Information processing systems – Open Systems Interconnection – Estelle : A formal description technique based on an extended state transition melel.

Hallinger, Dave. 1998. TFTP Trivial File Transfer Protocol. http://www.cs.rpi.edu/courses/netprog/lectures/ppthtml/tftp/. (20/5/1999).

Halsall, Fred. 1992. DATA COMMUNICATIONS, COMBUTER NETWORKS AND OPEN SYSTEMS. 3rd ed. New York : Addison Wesley.

Institut National des Telecommunications. ESTELLE COMPILER Ec. http://alix.int-evry.fr/~stan/edt.html. (13/6/1999).

Institut National des Telecommunications. ESTELLE GENERAL INFORMATION. http://alix.int-evry.fr/~stan/edt.html. (14/6/1999).

Institut National des Telecommunications. ESTELLE SIMULATOR/DEBUGGER Edb. http://alix.int-evry.fr/~stan/edt.html. (14/6/1999).

Lai, Richard and Jirachiefpattana, Ajin. 1998. Communication Protocol Specification and Verification. Massachusetts : Kluwer Academic Publishers.

Sollins, K. 1981. TFTP Protocol (revision 2). [RFC 783].

Stevens, W. Richard. 1991. UNIX Network Programming. New Jersey : Prentice-Hall.

ภาคผนวก ก

คำสั่งเฉพาะใน Estelle

- คำสั่ง init

คำสั่ง init เป็นคำสั่งที่ใช้ในการสร้างตัวแทนโมดูล โดยที่โมดูลที่ถูกสร้างขึ้นมาจะเป็นลูกของโมดูลที่ใช้คำสั่งนี้ รูปแบบของคำสั่ง init มีสองรูปแบบดังนี้

init module_variable **with** body-identifier;

init module_variable **with** body-identifier(ชุดของพารามิเตอร์)

ความแตกต่างระหว่างสองรูปแบบนั้นคือแบบที่สองอนุญาติให้ผ่านค่าพารามิเตอร์ให้กับโมดูลที่จะถูกสร้างแต่ต้องเป็นโมดูลที่ประกาศรับพารามิเตอร์ในส่วนหัวของโมดูลด้วย โดยที่ module_variable เป็นชื่อตัวแปรที่ประกาศไว้แล้วและเป็นชนิดโมดูล สมมติตัวแปร X ถูกประกาศให้เป็นชนิดโมดูล A ดังนั้นคำสั่ง

intit X with B เป็นการสร้างตัวแทนโมดูล A โดยใช้ตัวแปร X ในการอ้างถึงและ X มีส่วนของเนื้อหาโมดูลแบบ B คือ โมดูล A อาจมีส่วนเนื้อหาได้หลายส่วนนอกจากที่กำหนดไว้ด้วยชื่อ B แล้วก็ได้

- คำสั่ง connect

รูปแบบของคำสั่ง connect เป็นดังนี้

connect internal-ip **to** child-exteral-ip

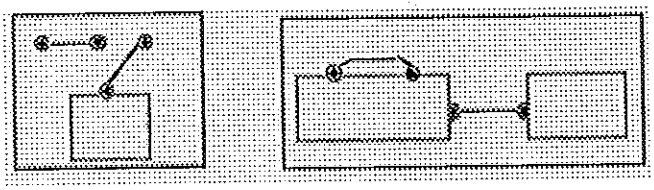
connect child-external-ip **to** internal-ip

connect child-external-ip **to** child-external-ip

connect internal-ip **to** internal-ip

การที่โมดูลใดใช้คำสั่ง connect นั้นก็เพื่อเชื่อมต่อ

- จุดสื่อสารภายในของโมดูลนั้นกับจุดสื่อสารภายนอกของโมดูลที่เป็นลูก (ตามรูปแบบที่ 1 และ 2)
- เชื่อมต่อจุดสื่อสารภายนอกของโมดูลลูกเข้าด้วยกัน (ตามรูปแบบที่ 3)
- เชื่อมต่อจุดสื่อสารภายในของโมดูลเข้าด้วยกัน (ตามรูปแบบที่ 4)



ภาพประกอบ ก-1 แสดงลักษณะของคำสั่ง connect ที่เป็นไปได้

(ที่มา: ISO 9074, 1989 : 23)

จุดสื่อสารที่อ้างถึงในคำสั่ง connect นั้นต้องอยู่ในส่วนประกาศช่องสื่อสารเดียวกันและต้องมีบทบาทตรงข้ามกัน และสังเกตว่าคำสั่ง connect นั้นจะอ้างถึงจุดสื่อสารภายนอกของโมดูลที่ใช้คำสั่งเองไม่ได้

3.2.7.3 คำสั่ง disconnect

คำสั่ง disconnect มีรูปแบบการใช้ดังนี้

disconnect internal-ip;

disconnect child-external-ip;

disconnect module-variable;

คำสั่ง disconnect 2 รูปแบบแรกเป็นการยกเลิกการเชื่อมต่อระหว่างจุดสื่อสารสองจุดถึงแม้ว่าจุดสื่อสารที่อ้างถึงหลังคำสั่งจะมีเพียงจุดเดียวก็ตามเพราะการเชื่อมต่อเป็นการเชื่อมต่อระหว่างจุดสื่อสาร

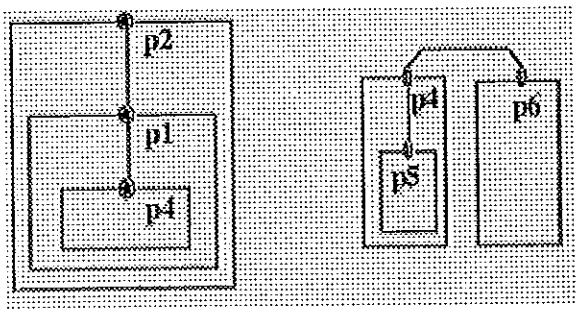
สองจุดจากคำสั่ง connect นั่นเอง ส่วนรูปแบบที่สามใช้ตัวแปรชนิดโมดูลตามหลังคำสั่งก็จะหมายถึงให้ยกเลิกการเชื่อมต่อทุกจุดสื่อสารภายนอกของโมดูลลูกของโมดูลที่ใช้คำสั่งนั้น

3.2.7.4 คำสั่ง attach

รูปแบบของคำสั่งเป็นดังนี้

attach external-ip to child-external-ip

คำสั่ง attach เป็นการเชื่อมจุดสื่อสารภายนอกของโมดูลที่ใช้คำสั่งกับจุดสื่อสารภายนอกของโมดูลลูกเท่านั้น โดยที่จุดสื่อสารแรกที่อยู่หลังคำสั่งต้องเป็นจุดสื่อสารภายนอกของโมดูลที่ใช้คำสั่ง attach และจุดสื่อสารที่สองต้องเป็นจุดสื่อสารภายนอกของโมดูลลูก จุดสื่อสารทั้งสองจุดนี้ต้องใช้ช่องสื่อสารเดียวกันในการประกาศที่ผ่านมาและต้องเป็นจุดสื่อสารที่มีบทบาทเดียวกัน ภาพประกอบ 2.12 แสดงลักษณะการใช้คำสั่ง attach



ภาพประกอบ ก-2 แสดงลักษณะการเชื่อมโยงเส้นทางสื่อสารโดยใช้คำสั่ง attach

(ที่มา: ISO 9074, 1989 : 34)

- **คำสั่ง detach**

คำสั่ง detach เป็นการยกเลิกการเชื่อมโยงเส้นทางสื่อสารที่เชื่อมด้วยคำสั่ง attach ซึ่งมีรูปแบบดังนี้

detach external-ip;

detach child-external-ip;

detach module-variable;

สองรูปแบบแรกเป็นการยกเลิกการเชื่อมโยงจุดสื่อสารสองจุดที่เชื่อมต่อด้วยคำสั่ง attach มาก่อนออกจากกัน ส่วนรูปแบบที่สามเป็นการยกเลิกการเชื่อมโยงจุดสื่อสารภายนอกทั้งหมดของโมดูลที่กำหนดด้วย module-variable

- คำสั่ง release และ terminate

คำสั่งทั้งสองมีรูปแบบดังนี้

release module-variable;

terminate module-variable;

ผลลัพธ์ที่ได้จากการใช้คำสั่ง release X เมื่อ X เป็นตัวแปรชนิดโมดูลและเป็นโมดูลลูกของโมดูลที่ใช้คำสั่งนี้จะเหมือนกับการกระทำสองอย่างนี้

- **disconnect** X และ **detach** X
- ทำลายตัวแทนโมดูล X และโมดูลลูกที่เกิดจากโมดูล X

ส่วนผลลัพธ์ของคำสั่ง terminate จะต่างจากการใช้คำสั่ง release ตรงที่ terminate X จะทำลายโมดูล X และโมดูลลูกทันทีพร้อมด้วยทำลายข่าวสารต่าง ๆ ที่อยู่ในคิวของโมดูลลูก

- คำสั่ง output

รูปแบบดังนี้

output ip-reference.interaction-identifier;

output ip-reference.interaction-identifier(actual-parameter-list);

เมื่อใช้คำสั่ง output แสดงว่าโมดูลนั้นทำการส่งข่าวสารหรือข้อมูลผ่านทางจุดสื่อสารที่กำหนด อย่างเช่น

output p1.REQUEST(3)

เป็นการส่งข่าวสาร REQUEST(3) ผ่านทางจุดสื่อสาร p1 หากจุดสื่อสาร p1 และ p2 เป็นจุดปลายของช่องเส้นทางสื่อสารการใช้คำสั่ง output p1.m จะเป็นการส่งข่าวสาร m เข้าไปในคิวของจุดสื่อสาร p2 นั่นเอง หากจุดสื่อสาร p1 ไม่ได้เป็นจุดปลายของเส้นทางสื่อสารข่าวสารที่ส่งผ่านจุดนี้ก็จะไม่มีผลถือว่าสูญหายไป

- คำสั่ง All

รูปแบบของคำสั่ง all เป็นดังนี้

all domain **do** statement;

โดยที่ statement คือคำสั่งทั่ว ๆ ไปในปาสคาลและใน Estelle ที่เสริมเข้ามาซึ่ง statement อาจเป็นประโยคผสมที่อยู่ระหว่างคำสงวน begin และ end ก็ได้
domain กำหนดกลุ่มของค่าต่าง ๆ ซึ่งอาจเป็นกลุ่มของโมดูลหรือชุดของตัวเลข หากใช้กับโมดูล domain ก็จะเป็นการประกาศชนิดของโมดูลและถือเป็นตัวแปรท้องถิ่นที่ใช้ได้เฉพาะในคำสั่ง all ขณะนั้นเท่านั้น ตัวอย่างเช่น

all M:ModuleType **do** M.Inprogress := false;

เป็นการกำหนดค่าให้ตัวแปรร่วม Inprogress มีค่าเป็น false ของโมดูลลูกทุกโมดูลที่มีชนิดเป็น ModuleType

- คำสั่ง **forone**

คำสั่ง forone อาจถือว่าเป็นคำสั่งตรงข้ามของคำสั่ง all ก็ได้เพราะการทำงานจะเป็นการค้นหาวัตถุเพียงหนึ่งวัตถุที่สอดคล้องกับเงื่อนไขก็จะทำคำสั่งตามที่กำหนดโดยมีรูปแบบดังนี้

forone domain **suchthat** boolean-expression **do** statement;

forone domain **suchthat** boolean-expression **do** statement1 **otherwise** statement2

โดยที่ domain จะเหมือนกับ domain ที่กำหนดในประโยค all ส่วน boolean-expression จะถูกประเมินผลสำหรับทุกวัตถุใน domain จนกว่าจะได้ค่าความจริงเป็นจริงจึงหยุด เมื่อวัตถุใดอยู่ในช่วงเงื่อนไขเป็นจริง statement ที่อยู่หลังคำ do จะถูกทำงานและสิ้นสุดกระบวนการของคำสั่ง forone หาก boolean-expression ให้ค่าความจริงเป็นเท็จสำหรับทุกวัตถุที่อยู่ใน domain statement2 จะถูกประมวลผลในกรณีรูปแบบที่สอง

- คำสั่ง **exist**

คำสั่ง exist เป็นคำสั่งที่ใช้สำหรับตรวจสอบว่ามีวัตถุที่สนใจอยู่หรือไม่โดยจะให้ค่าความจริงเป็นจริงหรือเท็จโดยมีรูปแบบดังนี้

exist domain **suchthat** boolean-expression;

- ข้อแตกต่างระหว่าง Estelle กับ Pascal (ISO 7185, 1983)

เนื่องจาก Estelle เป็นเทคนิคการเขียนข้อกำหนดโปรโตคอลไม่ใช่เป็นภาษาสำหรับเขียนโปรแกรมจึงมีข้อแตกต่างกับภาษาปาสคาลโดยในปาสคาลมีคำสั่งจัดการเกี่ยวกับแฟ้มหลายคำสั่งแต่ไม่มีใน Estelle ดังนี้

- คำสั่ง file
- คำสั่ง text
- คำสั่ง get
- คำสั่ง put
- คำสั่ง read
- คำสั่ง write
- คำสั่ง readln
- คำสั่ง writelon
- คำสั่ง eof
- คำสั่ง eoln

ภาคผนวก ข

ข้อกำหนดโปรโตคอล TFTP ในรูปแบบภาษา Estelle

```
SPECIFICATION TFTP;
default individual queue ;
timescale seconds ;

{ CONSTANT DECLARATION PART TFTP_protocol }
  const
    WRITE_OK = 1;
    READ_OK = 2;
    DENIED = 3;
    RESPONDER_BUSY = 4;
    CONNECTION_DELAY = 5;
    INITIATOR_BUSY = 6;
    SUCCESS = 7;
    RESEND_DELAY = 8;
    CONNECTION_TIMED_OUT = 9;

{ TYPE DECLARATION PART TFTP_protocol }
        type
                RRQ_PACKET = record
                        opcode: integer;
                        filename: array[1..50] of char;
                        eos1: char;
                        mode: array[1..9] of char;
                        eos2: char;
                end;
```

```
WRQ_PACKET = record
                opcode: integer;
                filename: array[1..50] of char;
                eos1: char;
                mode: array[1..9] of char;
                eos2: char;
            end;

            DATA_PACKET = record
                opcode: integer;
                block_no: integer;
                data: packed array[1..512] of char;
            end;

            ACK_PACKET = record
                opcode: integer;
                block_no: integer;
            end;

            ERROR_PACKET = record
                opcode: integer;
                errcode: integer;
                errstring: array[1..100] of char;
                eos: char;
            end;
```

```
{ CHANNEL DECLARATION PART TFTP_protocol }

 channel user_access_point (user,provider);
  by user:
   READ_RQST ;
   WRITE_RQST ;
  by provider:
   RESPONSE (returnCode:integer);

 channel network_access_point (initiator,responder);
  by initiator:
   WRQ ;
   RRQ ;
  by initiator,responder:
   ACK (ackBlockNum:integer);
   DATA (dataBlockNum:integer);
   ERROR (errorCode:integer);

{ MODULE HEADER DECLARATION PART TFTP_protocol }
 module InitiatorHeader systemprocess ;
  ip
   U : user_access_point (provider) ;
   N : network_access_point (initiator) ;
 end; { MODULE HEADER InitiatorHeader }
 module ResponderHeader systemprocess ;
  ip
   N : network_access_point (responder) ;
   U : user_access_point (user) ;
 end; { MODULE HEADER ResponderHeader }
```

```
module UserHeader systemprocess ;
  ip
    SERVER : user_access_point (provider) ;
    CLIENT : user_access_point (user) ;
end; { MODULE HEADER UserHeader }

{ MODULE BODY DECLARATION PART TFTP_protocol }
body InitiatorBody for InitiatorHeader;

{ VARIABLE DECLARATION PART InitiatorBody }
  var
    userResult : integer;

{ STATE DECLARATION PART InitiatorBody }
  state IDLE,BUSY;

{ MODULE HEADER DECLARATION PART InitiatorBody }
  module HandlerHeader activity ;
    ip
      TO_RESPONDER : network_access_point (initiator) ;
    export
      complete : boolean;
      result : integer;
  end; { MODULE HEADER HandlerHeader }
```

```
{ MODULE BODY DECLARATION PART InitiatorBody }
  body WriterBody for HandlerHeader;

  { VARIABLE DECLARATION PART WriterBody }
    var
      blockNum : integer;
      lastblock : boolean;
  { PROCEDURE&FUNCTION DECLARATION PART WriterBody }
function isLastBlock  : boolean;
                begin
                        isLastBlock:= lastblock;
                end;
{ STATE DECLARATION PART WriterBody }
    state IDLE,WAIT_ACK,DALLY;
    stateset
      WAITING = [WAIT_ACK, DALLY];

  { TRANSITION DECLARATION PART }
  initialize
    TO IDLE
      name setNOTcomplete : begin
        complete := false;
      end;
  trans
    FROM IDLE
      TO WAIT_ACK
        name send_WRQST : begin
          blockNum := 0;
          OUTPUT TO_RESPONDER.WRQ ;
```

```
    end;
trans
  FROM WAIT_ACK
    WHEN TO_RESPONDER.ACK
      PROVIDED (ackBlockNum=blockNum) AND isLastBlock
        TO DALLY
          name sendLastDATA : begin
            blockNum := blockNum+1;
            OUTPUT TO_RESPONDER.DATA(blockNum) ;
          end;
      PROVIDED (ackBlockNum=blockNum) AND NOT isLastBlock
        TO WAIT_ACK
          name sendNextDATA : begin
            blockNum := blockNum + 1;
            OUTPUT TO_RESPONDER.DATA(blockNum) ;
          end;
    begin end;
 trans
   FROM DALLY
     WHEN TO_RESPONDER.ACK
       PROVIDED ackBlockNum=blockNum
         TO IDLE
           name recvLastACK : begin
             complete := true;
             result := SUCCESS;
           end;
```

```
trans
  DELAY (CONNECTION_DELAY, CONNECTION_DELAY )
    FROM WAITING
      TO IDLE
        name connectionTimeOut : begin
          complete := true;
          result := CONNECTION_TIMED_OUT;
        end;
trans
  WHEN TO_RESPONDER.ERROR
    FROM IDLE, WAIT_ACK, DALLY
      TO IDLE
        name TFTP_error : begin
          complete := true;
          result := errorCode;
        end;
trans
  DELAY (RESEND_DELAY, RESEND_DELAY )
    FROM WAIT_ACK, DALLY
      TO same
        PROVIDED blockNum=0
          name resendWRQST : begin
            OUTPUT TO_RESPONDER.WRQ ;
          end;
        PROVIDED otherwise
          name resendDATA : begin
            OUTPUT TO_RESPONDER.DATA(blockNum) ;
          end;
    begin end;
```

```
trans
  WHEN TO_RESPONDER.ACK
    PROVIDED ackBlockNum <> blockNum
      FROM WAIT_ACK, DALLY
        TO same
          name discardOutOfOrderACK : begin
          end;
end;  { MODULE BODY WriterBody }
body ReaderBody for HandlerHeader;

{ VARIABLE DECLARATION PART ReaderBody }
  var
    blockNum : integer;
    lastblock : boolean;
{ PROCEDURE&FUNCTION DECLARATION PART ReaderBody  }
function isLastBlock  : boolean;
            begin
                  isLastBlock:= lastblock;
            end;
{ STATE DECLARATION PART ReaderBody }
  state IDLE,WAIT_DATA,DALLY;
{ MODULE VARIABLE DECLARATION PART ReaderBody }

{ TRANSITION DECLARATION PART }
initialize
  TO IDLE
    name init_trans : begin
      complete := true;
    end;
```

```
trans
  FROM IDLE
    TO WAIT_DATA
      name sendRRQ : begin
        blockNum := 0;
        OUTPUT TO_RESPONDER.RRQ ;
      end;

trans
  DELAY (RESEND_DELAY, RESEND_DELAY )
    FROM WAIT_DATA
      TO same
        PROVIDED blockNum=0
          name resendRRQ : begin
            OUTPUT TO_RESPONDER.RRQ ;
          end;
        PROVIDED otherwise
          name resendACK : begin
            OUTPUT TO_RESPONDER.ACK(blockNum) ;
          end;
    begin end;
  trans
    DELAY (CONNECTION_DELAY, CONNECTION_DELAY )
      FROM WAIT_DATA, DALLY
        TO IDLE
          name connectionTimeout : begin
            complete := true;
          end;
```

```
trans
  WHEN TO_RESPONDER.ERROR
    FROM IDLE, WAIT_DATA, DALLY
      TO IDLE
        name handleError : begin
          complete := true;
          result := errorCode;
        end;
trans
  FROM WAIT_DATA
    WHEN TO_RESPONDER.DATA
      PROVIDED isLastBlock AND (dataBlockNum=blockNum+1)
        TO DALLY
          name sendLastACK : begin
            blockNum := blockNum + 1;
            OUTPUT TO_RESPONDER.ACK(blockNum) ;
          end;
      PROVIDED NOT isLastBlock AND (dataBlockNum=blockNum+1)
        TO WAIT_DATA
          name sendACK : begin
            OUTPUT TO_RESPONDER.ACK(blockNum) ;
          end;
      PROVIDED otherwise
        name discardBadDATA : begin
        end;
    begin end;
```

```
trans
  FROM DALLY
    TO DALLY
      WHEN TO_RESPONDER.DATA
        PROVIDED blockNum=dataBlockNum
          name resendLastACK : begin
          end;
        PROVIDED otherwise
          name discardBadDATA2 : begin
          end;
    begin end;
end; { MODULE BODY ReaderBody }

{ MODULE VARIABLE DECLARATION PART InitiatorBody }
  modvar
    handler : HandlerHeader;

{ TRANSITION DECLARATION PART }
initialize
  TO IDLE
    name init_to_idle : begin
    end;
trans
  WHEN U.WRITE_RQST
    FROM IDLE
      TO BUSY
        name start_writer : begin
          INIT handler WITH WriterBody ;
```

```
        ATTACH N TO handler.TO_RESPONDER ;
      end;
  FROM BUSY
    TO same
      name reject_WRQST : begin
        OUTPUT U.RESPONSE(INITIATOR_BUSY) ;
      end;
  begin end;
trans
  WHEN U.READ_RQST
    FROM IDLE
      TO BUSY
        name start_reader : begin
          INIT handler WITH ReaderBody ;
          ATTACH N TO handler.TO_RESPONDER ;
        end;
    FROM BUSY
      TO same
        name reject_RRQST : begin
          OUTPUT U.RESPONSE(INITIATOR_BUSY) ;
        end;
    begin end;
trans
  PROVIDED handler.complete
    FROM BUSY
      TO IDLE
        name killHandler : begin
          TERMINATE handler ;
          OUTPUT U.RESPONSE(handler.result) ;
        end;
```

```
end;  { MODULE BODY InitiatorBody }

body ResponderBody for ResponderHeader;
{ VARIABLE DECLARATION PART ResponderBody }
  var
    oldBusy : boolean;

{ STATE DECLARATION PART ResponderBody }
  state IDLE,BUSY,SETTING_UP;
  stateset
    NOT_IDLE = [BUSY];

{ MODULE HEADER DECLARATION PART ResponderBody }
  module HandlerHeader activity ;
    ip
      TO_INITIATOR : network_access_point (responder)  ;
    export
      complete : boolean;
  end;  { MODULE HEADER HandlerHeader }

{ MODULE BODY DECLARATION PART ResponderBody }
  body WriterBody for HandlerHeader;

  { VARIABLE DECLARATION PART WriterBody }
    var
      blockNum : integer;
      lastblock : boolean;
  { PROCEDURE&FUNCTION DECLARATION PART WriterBody  }
function isLastBlock  : boolean;
                begin
```

```
                    isLastBlock:= lastblock;
                end;

{ STATE DECLARATION PART WriterBody }
  state IDLE,WAIT_DATA,DALLY;
{ TRANSITION DECLARATION PART }
initialize
  TO IDLE
    name setNOTcomplete : begin
      complete := false;
    end;
trans
  FROM IDLE
    TO WAIT_DATA
      name acknowledgeWRQST : begin
        OUTPUT TO_INITIATOR.ACK(0) ;
        blockNum := 0;
      end;
trans
  FROM WAIT_DATA
    TO WAIT_DATA
      name resendWRQST_ACK : begin
        OUTPUT TO_INITIATOR.ACK(0) ;
      end;
trans
  FROM WAIT_DATA, DALLY
    TO same
      WHEN TO_INITIATOR.DATA
        PROVIDED dataBlockNum=blockNum
          name resendLastACK : begin
```

```
            OUTPUT TO_INITIATOR.ACK(blockNum) ;
          end;
trans
  FROM WAIT_DATA
    WHEN TO_INITIATOR.DATA
      PROVIDED (dataBlockNum=blockNum+1) AND isLastBlock
        TO DALLY
          name acknowledgeLastDATA : begin
            blockNum := blockNum+1;
            OUTPUT TO_INITIATOR.ACK(blockNum) ;
          end;
      PROVIDED (dataBlockNum=blockNum+1) AND NOT isLastBlock
        TO WAIT_DATA
          name acknowledgeDATA : begin
            blockNum :=blockNum+1;
            OUTPUT TO_INITIATOR.ACK(blockNum) ;
          end;
    begin end;
trans
  DELAY (CONNECTION_DELAY, CONNECTION_DELAY )
    FROM WAIT_DATA, DALLY
      TO IDLE
        name timeoutAndDisconnect : begin
          complete := true;
        end;
trans
  DELAY (RESEND_DELAY, RESEND_DELAY )
    FROM WAIT_DATA
      TO same
        name timeoutAndResendACK : begin
```

```
            OUTPUT TO_INITIATOR.ACK(blockNum) ;
          end;
    end;  { MODULE BODY WriterBody }
    body ReaderBody for HandlerHeader;

    { VARIABLE DECLARATION PART ReaderBody }
      var
        blockNum : integer;
        lastblock : boolean;
    { PROCEDURE&FUNCTION DECLARATION PART ReaderBody  }
function isLastBlock  : boolean;
                begin
                        isLastBlock:= lastblock;
                end;

    { STATE DECLARATION PART ReaderBody }
      state IDLE,WAIT_ACK,DALLY;

    { TRANSITION DECLARATION PART }
    initialize
      TO IDLE
        name setNOTcomplete : begin
          complete := false;
        end;
    trans
      WHEN TO_INITIATOR.RRQ
        FROM IDLE
          TO WAIT_ACK
            name sendFirstDATA : begin
              blockNum := 1;
```

```
        OUTPUT TO_INITIATOR.DATA(blockNum) ;
      end;
  FROM WAIT_ACK, DALLY
    TO same
      name discardBadRRQ : begin
      end;
  begin end;
trans
  DELAY (RESEND_DELAY, RESEND_DELAY )
    FROM WAIT_ACK, DALLY
      TO same
        name resendDATA : begin
          OUTPUT TO_INITIATOR.DATA(blockNum) ;
        end;
trans
  DELAY (CONNECTION_DELAY, CONNECTION_DELAY )
    FROM WAIT_ACK, DALLY
      TO IDLE
        name connectionTimeOut : begin
          complete := true;
        end;
trans
  FROM WAIT_ACK
    WHEN TO_INITIATOR.ACK
      PROVIDED (ackBlockNum=blockNum) AND isLastBlock
        TO DALLY
          name sendLastDATA : begin
            blockNum := blockNum+1;
            OUTPUT TO_INITIATOR.DATA(blockNum) ;
          end;
```

```
    PROVIDED (blockNum=ackBlockNum) AND NOT isLastBlock
      TO WAIT_ACK
        name sendDATA : begin
          blockNum := blockNum + 1;
          OUTPUT TO_INITIATOR.ACK(blockNum) ;
        end;
      PROVIDED otherwise
        name discardBadACK : begin
        end;
    begin end;
trans
  FROM DALLY
    WHEN TO_INITIATOR.ACK
      PROVIDED ackBlockNum=blockNum
        TO IDLE
          name setComplete : begin
            complete := true;
          end;
      PROVIDED otherwise
        TO same
          name discardBadACK2 : begin
          end;
    begin end;
end; { MODULE BODY ReaderBody }
```

```
{ MODULE VARIABLE DECLARATION PART ResponderBody }
  modvar
    handler : HandlerHeader;

{ TRANSITION DECLARATION PART }
initialize
  TO IDLE
    begin end;
trans
  WHEN N.WRQ
    FROM IDLE
      TO SETTING_UP
        name request_write_permission : begin
          OUTPUT U.WRITE_RQST ;
        end;
    FROM NOT_IDLE
      TO same
        name responderBusy1 : begin
          OUTPUT N.ERROR(RESPONDER_BUSY) ;
        end;
    begin end;
 trans
   WHEN N.RRQ
     FROM IDLE
       TO SETTING_UP
         name request_read_permission : begin
           OUTPUT U.READ_RQST ;
         end;
```

```
    FROM NOT_IDLE
      TO same
        name responderBusy2 : begin
          OUTPUT N.ERROR(RESPONDER_BUSY) ;
        end;
    begin end;
trans
  WHEN U.RESPONSE
    FROM SETTING_UP
      PROVIDED returnCode = WRITE_OK
        TO BUSY
          name setupWriter : begin
            INIT handler WITH WriterBody ;
            ATTACH N TO handler.TO_INITIATOR ;
          end;
      PROVIDED returnCode = READ_OK
        TO BUSY
          name setupReader : begin
          end;
      PROVIDED otherwise
        TO IDLE
          name rejectedByServer : begin
            OUTPUT N.ERROR(DENIED) ;
          end;
      begin end;
    FROM IDLE, BUSY
      TO same
        name discardUserData : begin
```

```
        end;
    begin end;
  trans
    PROVIDED handler.complete
      FROM BUSY
        TO IDLE
          name killHandler : begin
            TERMINATE handler  ;
          end;
  end;  { MODULE BODY ResponderBody }

body ClientBody for UserHeader;
        state SEND,WAIT;

        initialse to SEND
                begin end;

        trans
           from SEND
                to WAIT
                   delay(CONNECTION_DELEY)
                       name sendREAD_RQST :
                                begin
                                   output CLIENT.READ_RQST;
                                end;
        trans
           from SEND
                to WAIT
                   delay(CONNECTION_DELEY)
```

```
                    name sendWRITE_RQST :
                      begin
                          output CLIENT.WRITE_RQST;
                      end;
        end;

  body ServerBody for UserHeader;

state WAIT,RECIEVE;

initialise to WAIT
    begin end;

trans
   from WAIT
        to RECIEVE
           delay(CONNECTION_DELEY)
                name recieve_WRQ :
                  begin
                      output SERVER.RESPONSE(WRITE_OK);
                  end;
trans
    from WAIT
        to RECIEVE
            delay(CONNECTION_DELEY)
                name recieve_RRQ :
                  begin
                      output SERVER.RESPONSE(READ_OK);
```

```
                    end;
        end; { UserBody }
{ MODULE VARIABLE DECLARATION PART TFTP_protocol }
  modvar
    Initiator : InitiatorHeader;
    Responder : ResponderHeader;
    ClientInst : UserHeader;
    ServerInst : UserHeader;

{ TRANSITION DECLARATION PART }
initialize
  name trans1 : begin
    INIT Initiator WITH InitiatorBody ;
    INIT Responder WITH ResponderBody ;
          INIT ClientInst WITH ClientBody;
          INIT ServerInst WITH ServerBody;
          connect ClientInst.CLIENT to Initiator.U;
          connect ServerInst.SERVER to Responder.U;
    CONNECT Initiator.N TO Responder.N ;
  end;
end. { end of specification TFTP_protocol }
```

ภาคผนวก ค

การติดตั้งโปรแกรม Estelle Graphical Editor

โปรแกรม Estelle Graphical Editor ที่ใช้ในงานวิจัยนี้เป็นรุ่น 2.1 ที่ทำงานบนระบบปฏิบัติการ Solaris 2.5.1 ซึ่งสามารถดาวน์โหลดมาใช้ได้โดยแฟ้มที่เก็บโปรแกรมนี้เป็นแฟ้มที่ถูกบีบอัดอยู่ในชื่อ editor-solaris-21.tar.gz เมื่อได้แฟ้มนี้มาแล้วก็ทำการขยายโดยใช้คำสั่งดังนี้

```
gzip –dc editor-solaris-21.tar.gz | tar xvf –
```

เมื่อขยายแล้วก็จะได้แฟ้มต่าง ๆ ภายในไดเรคทอรีปัจจุบันดังนี้

startGE	เป็นแฟ้มสำหรับเรียกใช้โปรแกรม Estelle Graphical Editor
english.data	แฟ้มภาษาที่ใช้สำหรับโปรแกรม ซึ่งที่โปรแกรมใช้เป็นภาษาอังกฤษ
LIMITS.TXT	แฟ้มอธิบายข้อจำกัดต่าง ๆ ของโปรแกรม

เมื่อได้แฟ้มต่าง ๆ แล้วก็สามารถทำการติดตั้งใช้งานได้ดังนี้

สร้างไดเรคทอรีสำหรับทำงานของโปรแกรม ในที่นี้คือ EDITOR

```
mkdir EDITOR
```

คัดลอกแฟ้ม startGE และแฟ้ม english.dat ไปยังไดเรคทอรี EDITOR

```
cp startGE EDITOR/
cp english.data EDITOR/
```

กำหนดตัวแปรเชลล์ GUIPATH ให้ชี้ไปยัง ไดเรคทอรี EDITOR

```
setenv GUIPATH EDITOR  (กรณีที่ใช้ C หรือ Tc shell)
export GUIPATH=EDITOR (กรณีที่ใช้ Bourn shell)
```

เพิ่ม GUIPATH ให้กับตัวแปร PATH

setpath ($path $GUIPATH) (กรณีที่ใช้ Tc หรือ C shell)

export PATH=$PATH:$GUIPATH (กรณีที่ใช้ Bourn shell)

การเรียกใช้โปรแกรมทำได้โดยพิมพ์คำสั่ง

$startGE &

ภาคผนวก ง

การติดตั้ง EDT (Estelle Development Toolsets)

โปรแกรม EDT ที่ใช้ในงานวิจัยนี้เป็นรุ่นทดลองซึ่งต้องดาวน์โหลดแฟ้มลิขสิทธ์ (Liscence file) มาใช้ทุก ๆ หนึ่งเดือน เมื่อครบหนึ่งเดือนต้องขอแฟ้มลิขสิทธ์ใหม่ โดยแฟ้มลิขสิทธ์นี้ต้องเก็บไว้ในชื่อ edt.dat ในสาระบบที่บรรจุโปรแกรม EDT อยู่ ขั้นตอนการติดตั้งโปรแกรม EDT เป็นดังนี้

1. สร้างไดเรคทอรีเพื่อเก็บชุดโปรแกรมที่ใช้ในโปรแกรม EDT ดังนี้

   ```
   pangha$ mkdir EDT
   ```

2. คัดลอกแฟ้ม edtsol.tar.gz ไว้ในไดเรคทอรี EDT แล้วทำการขยาย (unzip) แฟ้มดังกล่าว

   ```
   pangha$ gzip –dc edtsol.tar.gz | tar xvf --
   ```

   ขั้นตอนนี้จะได้ไดเรคทอรี TOOLSET อยู่ภายใต้ไดเรคทอรี EDT พร้อมกับมีไดเรคทอรีย่อยต่าง ๆ ภายใต้ไดเรคทอรีนี้ เช่น TOOLSET/bin TOOLSET/edb

3. สร้างตัวแปรเชลล์ ESTEL เพื่อให้ชี้ไปยังไดเรคทอรีที่เก็บชุดคำสั่งของ EDT ที่ได้จากข้อสอง คือไดเรคทอรี TOOLSET

   ```
   export ESTEL=~/EDT/TOOLSET   (กรณีใช้ Bourn shell)
   setenv ESTEL ~/EDT/TOOLSET  (กรณีใช้ C หรือ Tc shell)
   ```

4. เพิ่มตัวแปร PATH ให้ชี้ไปยัง $ESTEL/bin

   ```
   export PATH=$PATH:$ESTEL/bin   (กรณีใช้ Bourn shell)
   set path=($ESTEL/bin $path)          (กรณีใช้ C หรือ Tc shell)
   ```

5. เพิ่มตัวแปร CPPPATH ให้ชี้ไปยัง absolute path ของ CPP pre-processor ซึ่งเป็นไดเรคทอรีที่ CPP ตามด้วยชื่อแฟ้ม CPP เช่นหาก CPP pre-processor อยู่ในไดเรคทอรี /usr/lib/ ก็จะได้ CPPPATH เป็น /usr/lib/cpp

   export CPPPATH=/usr/lib/cpp (กรณีใช้ Bourn shell)
   setenv CPPPATH /usr/lib/cpp (กรณีใช้ C หรือ Tc shell)

6. เพิ่มตัวแปร LD_LIBRARY_PATH ให้ชี้ไปยังไดเรคทอรีที่เก็บไลบรารีของ Motif (กรณีที่ใช้เครื่องคอมพิวเตอร์ (Sun Sparc)

   export LD_LIBRARY_PATH=/usr/dt (กรณีใช้ Bourn shell)
   setenv LD_LIBRARY_PATH /usr/dt (กรณีใช้ C หรือ Tc shell)

7. คัดลอกแฟ้ม XEDT-color จากไดเรคทอรี $ESTEL/app-defaults ไปยังไดเรคทอรีบ้าน (Home directory) ของผู้ใช้ โดยเปลี่ยนเป็นชื่อ Xedt

   cp $ESTEL/app-deffaults/XEdt-color ~/XEdt

ประวัติผู้เขียน

ชื่อ นายอาหมาน หมัดเจริญ

วัน เดือน ปีเกิด วันจันทร์ที่ 11 เดือน พฤศจิกายน พ.ศ. 2511

วุฒิการศึกษา

วุฒิ	ชื่อสถาบัน	ปีที่สำเร็จการศึกษา
วิทยาศาสตรบัณฑิต (คณิตศาสตร์ประยุกต์)	มหาวิทยาลัยสงขลานครินทร์	2539

ทุนการศึกษา (ที่ได้รับระหว่างการศึกษา)

งานวิจัยนี้ได้รับทุนสนับสนุนจากโครงการทุนบัณฑิตศึกษาในประเทศ สำนักงานพัฒนาวิทยาศาสตร์และเทคโนโลยีแห่งชาติ