โปรแกรมเกตเวย์สำหรับแปลงรูปแบบเอกสาร XML บนเว็บเซิร์ฟเวอร์

XML TRANSLATOR FOR GATEWAY ON WEBSERVER

วิรัตน์ มหาศาลประเสริฐ
อนิรุจ วรกิจรุ่งเรือง
อนิวัชร อารมณ์รัตน์

ปัญหาพิเศษนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิทยาศาสตรบัณฑิต
ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์
คณะวิทยาศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2544

XML TRANSLATOR FOR GATEWAY ON WEBSERVER

WIRAT MAHASALPRASERT

ANIRUT WORAKITRUNGRUANG

ANIWAT ARROMRATANA

A SPECIAL PROJECT SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF BACHELOR OF SCIENCE

DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE

FACULTY OF SCIENCE

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

ACADEMIC YEAR 2001

หัวข้อปัญหาพิเศษ โปรแกรมเกตเวย์สำหรับแปลงรูปแบบเอกสาร XML บนเว็บเซิร์ฟเวอร์
XML TRANSLATOR FOR GATEWAY ON WEBSERVER

ชื่อนักศึกษา นายวิรัตน์ มหาศาลประเสริฐ 41056102
นายอนิรุจ วรกิจรุ่งเรือง 41056138
นายอนิวัชร อารมณ์รัตน์ 41056139

ภาควิชา คณิตศาสตร์และวิทยาการคอมพิวเตอร์
สาขาวิชา วิทยาการคอมพิวเตอร์
อาจารย์ที่ปรึกษา อาจารย์วิสันต์ ตั้งวงษ์เจริญ

ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง อนุมัติให้นับปัญหาพิเศษนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตร วิทยาศาสตรบัณฑิต สาขาวิทยาการคอมพิวเตอร์ ประจำปีการศึกษา 2544

คณะกรรมการสอบ	ลายมือชื่อ
ประธานกรรมการ อาจารย์วีระชัย ตันยะสิทธิ์	
กรรมการ ดร.นันทิกา เบญจเทพานันท์	
กรรมการและอาจารย์ที่ปรึกษา อาจารย์วิสันต์ ตั้งวงษ์เจริญ	

(ผู้ช่วยศาสตราจารย์ไพรบูลย์ พันธรักษ์พงษ์)
หัวหน้าภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์

หัวข้อปัญหาพิเศษ โปรแกรมเกตเวย์สำหรับแปลงรูปแบบเอกสาร XML บนเว็บเซิร์ฟเวอร์

ชื่อนักศึกษา นายวิรัตน์ มหาศาลประเสริฐ 41056102

นายอนิรุจ วรกิจรุ่งเรือง 41056138

นายอนิวัชร อารมณ์รัตน์ 41056139

ปริญญา วิทยาศาสตรบัณฑิต

ภาควิชา คณิตศาสตร์และวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์

สาขาวิชา วิทยาการคอมพิวเตอร์

ปีการศึกษา 2544

อาจารย์ที่ปรึกษา อาจารย์วิสันต์ ตั้งวงษ์เจริญ

บทคัดย่อ

ในปัจจุบัน การเข้าถึงเอกสารที่อยู่บนเครือข่ายอินเทอร์เน็ทสามารถทำได้หลายรูปแบบ อาทิ การเข้าถึงเอกสารในรูปแบบของการใช้เว็บเบราเซอร์บนคอมพิวเตอร์, การเข้าถึงเอกสารในรูปแบบของการใช้ WAP Browser บนโทรศัพท์มือถือ เป็นต้น ซึ่งการเข้าถึงเอกสารในสองรูปแบบนี้จะเข้าถึงเอกสารในรูปแบบ HTML และ WML ตามลำดับ สาเหตุนี้ ทำให้เว็บเซิร์ฟเวอร์นั้นจะต้องมีเอกสารทั้งสองรูปแบบนี้ เพื่อรองรับการร้องขอเอกสารจากผู้ใช้บริการ

โปรแกรมเกตเวย์สำหรับแปลงรูปแบบเอกสาร XML บนเว็บเซิร์ฟเวอร์ ที่สร้างขึ้นด้วยภาษาจาวานี้ จึงเป็นโปรแกรมที่รองรับการร้องขอเอกสารทั้งสองรูปแบบนี้ได้ โดยที่โปรแกรมจะต้องเก็บเอกสารเพียงรูปแบบเดียว แต่สามารถให้บริการการร้องขอเอกสารได้ทั้งสองรูปแบบ กล่าวคือกรณีที่เว็บเซิร์ฟเวอร์ไม่มีเอกสารในรูปแบบนั้น โปรแกรมจะทำการแปลงเอกสารให้เหมาะสมกับรูปแบบการร้องขอ ก่อนจะทำการส่งเอกสารกลับไปยังผู้ร้องขอเอกสาร และกรณีที่เว็บเซิร์ฟเวอร์มีเอกสารรูปแบบนั้น เว็บเซิร์ฟเวอร์จะทำการส่งเอกสารกลับไปยังผู้ร้องขอได้ทันที

Special Project Title	XML Translator for Gateway on Web Server	
Students	Mr. Wirat Mahasalprasert	40056102
	Mr. Anirut Worakitrungruang	40056138
	Mr. Aniwat Arromratana	40056139
Degree	Bachelor's Degree of Science	
Department	Mathematics and Computer Science, Faculty of Science	
Programme	Computer Science	
Academic Year	2001	
Special Project Advisor	Lecturer Wisan Tangwongcharoen	

ABSTRACT

Nowadays, there are several methods for accessing to documents on the internet such as access using web browser on computer, access using WAP browser on mobile phone, etc. These two methods are for accessing to documents of HTML format and WML format, respectively. This forces the web server should have two formats of a document only for responsing user's request.

XML Translator for Gateway on Web Server, developed by Java, is able to response two methods of request. The program needs just any one format of each documents for service to any method of request. It is because the program contains the translator used to translate documents to be in appropriate format before sending the document back to user. This is useful when the server doesn't have the document in suitable format matching with method of request. In the case that the server has the document in appropriate format, the response could be happened immediately.

กิตติกรรมประกาศ

ในการทำปัญหาพิเศษเรื่อง โปรแกรมเกตเวย์สำหรับแปลงรูปแบบเอกสาร XML บนเว็บเซิร์ฟเวอร์ สามารถสำเร็จลุล่วงไปได้ด้วยดี ทางคณะผู้จัดทำต้องขอขอบพระคุณ อาจารย์วิสันต์ ตั้งวงษ์เจริญ ในฐานะที่เป็นอาจารย์ผู้รับผิดชอบปัญหาพิเศษฉบับนี้ ที่กรุณาให้คำแนะนำและเป็นที่ปรึกษาในการแก้ปัญหาต่าง ๆ รวมทั้งเป็นผู้ตรวจสอบความถูกต้องของปัญหาพิเศษฉบับนี้

ท้ายที่สุด ทางคณะผู้จัดทำต้องขอขอบพระคุณ บิดา มารดา ที่ได้ให้ความสนับสนุนทางด้านกำลังใจรวมถึงทุนทรัพย์ในการทำปัญหาพิเศษ รวมถึง เพื่อน พี่และน้อง ผู้คอยให้กำลังใจแก่คณะผู้จัดทำ รวมถึงพี่ ๆ เจ้าหน้าที่ห้องปฏิบัติการคอมพิวเตอร์ 214 ที่ช่วยเปิดห้องปฏิบัติการคอมพิวเตอร์ค่ำ ๆ ทุกวัน จนการทำปัญหาพิเศษนี้สำเร็จด้วยดีไว้ ณ ที่นี้

คณะผู้จัดทำ

มีนาคม 2545

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

ตารางที่ **หน้า**

สารบัญภาพ

ภาพที่ **หน้า**

สารบัญภาพ (ต่อ)

ภาพที่ **หน้า**

สารบัญภาพ (ต่อ)

ภาพที่ **หน้า**

สารบัญภาพ (ต่อ)

ภาพที่ **หน้า**

สารบัญภาพ (ต่อ)

ภาพที่ **หน้า**

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ในปัจจุบันเราสามารถเข้าสู่เครือข่ายอินเทอร์เน็ทได้จากอุปกรณ์ต่าง ๆ เช่น คอมพิวเตอร์, โทรศัพท์มือถือ ฯลฯ และอุปกรณ์เหล่านี้ใช้รูปแบบของเอกสารที่แตกต่างกัน แต่เอกสารเหล่านี้มีโครงสร้างมาจากเอกสาร XML (eXtensible Markup Language) ซึ่งเป็นเอกสารที่จะเน้นทางด้านโครงสร้างของข้อมูลและความถูกต้องของข้อมูล โดยรูปแบบเอกสารที่ใช้ในปัจจุบัน อาทิ เช่น

- HTML (Hypertext Markup Language) แสดงบนเว็บเบราเซอร์ของเครื่องคอมพิวเตอร์
- WML (Wireless Markup Language) แสดงบนโทรศัพท์มือถือระบบ WAP

จึงเป็นเรื่องยากที่จะให้เว็บเซิร์ฟเวอร์ให้บริการรูปแบบเอกสารได้หลายชนิด เพื่อจะอำนวยความสะดวกให้แก่ผู้ที่จะเข้ามาใช้บริการของเว็บเซิร์ฟเวอร์ให้สามารถเข้าถึงเอกสารได้หลายรูปแบบ ทางผู้พัฒนาจึงได้คิดหาวิธีการที่จะให้มีเอกสารเพียงรูปแบบใดรูปแบบหนึ่งบนเว็บเซิร์ฟเวอร์ และสามารถที่จะตอบสนองกับผู้ใช้ที่ร้องขอเอกสารรูปแบบอื่น ๆ ได้ โดยการแปลงรูปแบบเอกสารที่มีอยู่บนเว็บเซิร์ฟเวอร์ให้เป็นรูปแบบเอกสารตามที่ผู้ใช้ร้องขอมา ก่อนส่งกลับไปยังผู้ร้องขอ ซึ่งทำให้ผู้ที่ต้องการร้องขอเอกสารรูปแบบอื่น ๆ นอกเหนือจากที่มีอยู่บนเว็บเซิร์ฟเวอร์สามารถเข้าถึงข้อมูลบนเว็บเซิร์ฟเวอร์ได้ นอกจากนี้ ยังช่วยให้ผู้พัฒนาเว็บเพจทั้งหลายได้รับความสะดวกมากขึ้น โดยการสร้างเว็บเพจเพียงรูปแบบเดียวที่มีความชำนาญ แต่สามารถตอบสนองผู้ที่เข้ามาชมเว็บเพจได้ในวงกว้างมากขึ้น นับว่าเป็นการประหยัดเวลาและเป็นการลดค่าใช้จ่ายในการสร้างเว็บเพจให้กับเจ้าของเว็บไซท์ที่มีการจ้างผู้พัฒนาเว็บเพจได้อีกด้วย

1.2 ความมุ่งหมายและวัตถุประสงค์ของการศึกษา

1.2.1 เพื่อแก้ปัญหาที่ต้องสร้างเอกสารไว้หลายรูปแบบไว้บนเว็บเซิร์ฟเวอร์เดียวกัน

1.2.2 เพื่ออำนวยความสะดวกแก่ผู้ใช้ที่ต้องการเข้าเว็บเซิร์ฟเวอร์จากอุปกรณ์ต่างๆ

1.2.3 เพื่อสร้างตัวแปลงเอกสารโดยใช้สไตล์ชีทเป็นหลักในการสร้างเอกสาร

1.2.4 เพื่อสร้างเว็บเซิร์ฟเวอร์ที่สามารถตอบสนองการร้องขอตามรูปแบบของโพรโตคอล HTTP

1.3 ทฤษฎีหรือแนวคิดที่ใช้ในการศึกษา

เว็บเซิร์ฟเวอร์ที่สร้างขึ้น มีการทำงานบนเครือข่ายอินเทอร์เน็ทภายใต้โพรโตคอล HTTP โดยจะสามารถรับการร้องขอแบบ HTTP จากไคลเอนท์และตอบสนองแบบ HTTP กลับไปยังไคลเอนท์

ภาษาที่ใช้ในการจัดสร้างเอกสารสำหรับให้บริการบนเครือข่ายอินเทอร์เน็ทส่วนใหญ่มีพื้นฐานมาจากภาษา XML แต่อาจจะมีโครงสร้างเอกสารแตกต่างกันไป ตัวอย่างเช่น ภาษา WML, ภาษา XHTML เป็นต้น และเนื่องจากภาษา XSL มีความสามารถในการปรับเปลี่ยนโครงสร้างเอกสารที่เขียนด้วยภาษา XML ดังนั้นภาษา XSL จึงเป็นภาษาที่เหมาะสมสำหรับสร้างตัวแปลงเอกสาร

1.4 ขอบเขตของการศึกษา

1.4.1 สามารถให้บริการเอกสารต่าง ๆ ได้เช่นเดียวกับเว็บเซิร์ฟเวอร์ทั่วไป ไม่ว่าจะเป็นการรับการร้องขอเอกสาร หรือการตอบสนองการร้องขอเอกสาร

1.4.2 สามารถแปลงเอกสารที่อยู่บนเว็บเซิร์ฟเวอร์ที่ไม่เป็นชนิดเดียวกันกับที่ไคลเอนท์ร้องขอให้เป็นเอกสารชนิดเดียวกับเอกสารที่ร้องขอได้ โดยสามารถแปลงรูปแบบเอกสารได้ดังนี้

- จาก เอกสาร HTML ไปเป็น เอกสาร WML
- จาก เอกสาร WML ไปเป็น เอกสาร HTML

1.5 ขั้นตอนการศึกษา

1.5.1 ศึกษาโครงสร้างของเอกสารในรูปแบบต่าง ๆ ที่มีให้บริการบนเครือข่ายอินเทอร์เน็ท รวมทั้งโครงสร้างเอกสาร XSL ซึ่งเป็นตัวกำหนดรูปแบบการแสดงผล

1.5.2 ศึกษาการทำงานของโพรโตคอล HTTP ที่ทำงานอยู่บนเครือข่ายอินเทอร์เน็ท

1.5.3 ศึกษาการทำงานของโปรแกรมเว็บเซิร์ฟเวอร์ ในเรื่องการตอบสนองการร้องขอแบบต่าง ๆ

1.5.4 ศึกษาการทำงานของภาษาจาวา ในส่วนที่เกี่ยวข้องกับการทำงานบนระบบเครือข่ายอินเทอร์เน็ท

1.5.5 ศึกษาการทำงานและการใช้งาน Java Servlet

1.5.6 สร้างโปรแกรมเว็บเซิร์ฟเวอร์ที่มีการทำงานสอดคล้องกับหลักการของโพรโตคอล HTTP โดยใช้ภาษาจาวา พร้อมทดสอบการทำงาน

1.5.7 สร้างตัวแปลงเอกสารระหว่างเอกสาร WML และ เอกสาร HTML โดยใช้ภาษาจาวา พร้อมทดสอบการทำงาน

1.5.8 นำตัวแปลงเอกสารเข้าไปรวมเป็นส่วนหนึ่งของโปรแกรมเว็บเซิร์ฟเวอร์ พร้อมทดสอบการทำงาน

1.5.9 เพิ่มความสามารถในการทำงานของโปรแกรมเว็บเซิร์ฟเวอร์ให้ใช้งานกับ Java Servlet ได้

1.5.10 รวบรวมข้อมูลในการทำปัญหาพิเศษทั้งหมดมาดำเนินการจัดทำเป็นเอกสาร

บทที่ 2
ทฤษฎีที่เกี่ยวข้อง

2.1 ภาษา XML (eXtensible Markup Language)

ภาษา XML เกิดขึ้นในปี 1996 โดยองค์กร W3C ภาษา XML เป็นภาษา Markup ที่ได้รวมเอาความยืดหยุ่นและประสิทธิภาพของภาษา SGML เข้ากับการเป็นที่ยอมรับอย่างกว้างขวางของภาษา HTML

ภาษา XML สร้างขึ้นจากพื้นฐานของภาษา SGML เพื่อใช้อธิบายโครงสร้างของเอกสาร โดยมีการนำเสนอเป็นส่วนแยกอิสระออกมา ทำให้สามารถใช้ข้อมูลชุดเดิมซ้ำได้หลาย ๆ ครั้ง และแสดงออกมาในรูปแบบที่แตกต่างกันได้

2.1.1 โครงสร้างประโยคพื้นฐานในภาษา XML

โครงสร้างของภาษา XML มีลักษณะคล้ายกับภาษา HTML กล่าวคือ จะใช้เครื่องหมาย “<” และ “>“ เพื่อกำหนดแท็ก โดยเริ่มต้นด้วย “<” และจบด้วยเครื่องหมาย “>” นอกจากนี้ ภาษา XML ยังมีคุณสมบัติ Case-sensitive ที่จะพิจารณาว่า คำเดียวกันแต่เขียนตัวเล็กตัวใหญ่ต่างกัน ถือว่าไม่ใช่คำเดียวกัน

Book ≠ BOOK ≠ book ≠ booK

ภาพที่ 2.1 ภาพแสดงคุณสมบัติ Case-sensitive ของภาษา XML

2.1.1.1 ตัวอักษร

เนื่องจากภาษา XML มีจุดมุ่งหมายเพื่อใช้ได้ในขอบข่ายที่กว้างขวางขึ้น ตัวอักษรจึงไม่ได้จำกัดอยู่แค่เพียง 7 บิตตามมาตรฐานแอสกีเท่านั้น โดยภาษา XML กำหนดใช้ตัวอักษรแบบ 16 บิตตามมาตรฐาน Unicode 2.1 ซึ่งถือเป็นมาตรฐานใหม่ ถึงแม้ตัวอักษรส่วนใหญ่จะไม่ได้เป็นแบบ Unicode แต่มาตรฐาน Unicode ก็ถูกออกแบบมาให้สามารถทำงานได้กับตัวอักษรที่มีการเข้ารหัสทุกชนิดที่มีอยู่ในปัจจุบัน การแปลงตัวอักษรจากมาตรฐานแอสกีเป็น Unicode จะทำการแปลงรหัสตัวอักษรจาก 7 บิตเป็น 16 บิต โดยที่ยังคงเก็บค่าของตัวอักษร 7 บิตอยู่ ตัวอักษรในภาษา XML ที่ถูกต้องรวมถึงรหัส C0 ของมาตรฐานแอสกีที่ควบคุมตัวอักษรแอสกีและตัวอักษร Unicode

ตารางที่ 2.1 ตารางแสดงค่าตัวอักษรแบบ Unicode ในรูปแบบเลขฐานสิบหก

Character Value (hexadecimal)	Description
09	Horizontal Tab (HT)
0A	Line-Feed (LF)
0D	Carriage-Return (CR)
20..7E	ASCII Display Characters
80..D7FF	Unicode Characters
E000..F8FF	“Private Use Area”
F900..FFFD	CJK (Chinese-Japanese-Korean) Compatibility

มาตรฐาน Unicode เพิ่มตัวอักษรขึ้นกว่า 137,000 ตัว โดยเรียกว่า “Private Use Area” สำหรับแอพพลิเคชันที่ใช้ตัวอักษรเฉพาะ การแลกเปลี่ยนข้อมูลที่จะใช้ตัวอักษรพิเศษเหล่านี้ต้องมีการตกลงกันเกี่ยวกับความหมายให้ตรงกันก่อนทำการแลกเปลี่ยนข้อมูล

2.1.1.2 ชื่อ

โครงสร้างที่ใช้ในภาษา XML ส่วนใหญ่มักจะถูกตั้งชื่อ โดยชื่อในภาษา XML ต้องขึ้นต้นด้วยตัวอักษร, เครื่องหมายขีดล่าง (underscore, _), หรือเครื่องหมายโคลอน (:) และต่อด้วยตัวอักษรที่ถูกต้อง ประกอบไปด้วย ตัวเลข, ตัวอักษร, เครื่องหมายยัติภังค์ (-), หรือจุด (.) ในกรณีเครื่องหมายโคลอน (:) จะใช้เป็นตัวคั่นระหว่างช่องว่างของชื่อเฉพาะเท่านั้น เนื่องจากตัวอักษรไม่ถูกจำกัดอยู่เพียงตัวอักษรตามมาตรฐานแอสกี ดังนั้นผู้ใช้สามารถใช้ภาษาของตนเองเพื่อทำงานได้ เช่น

Book

BOOK

Wrox:Book_Catalog

สมาคม_ยุโรป_เพื่อ_เอเชีย_(CERN)

ภาพที่ 2.2 ภาพแสดงชื่อที่ถูกต้องในภาษา XML

จากภาพที่ 2.2 สองบรรทัดแรก ชื่อทั้งสองถือว่าเป็นคนละตัวกัน เพราะภาษา XML มีลักษณะเป็น Case-sensitive ส่วนบรรทัดที่สามเป็นการแสดงถึงการใช้เครื่องหมายโคลอน (:) เพื่อแบ่งคั่นช่องว่างตัวอักษร

-Book

42book

AmountIn$

$E=mc^2$

XmlData

XML_on_NeXt_machines

ภาพที่ 2.3 ภาพแสดงชื่อที่ไม่ถูกต้องในภาษา XML

จากภาพที่ 2.3 สองบรรทัดแรกใช้ตัวเริ่มต้นที่ผิด ถึงแม้ว่าจะใช้ตัวอักษรที่ถูกต้องตามกฎคือ "-" และ "4" แต่ไม่มีการอนุญาตให้ใช้ "-" และตัวเลข ขึ้นต้นชื่อ บรรทัดที่สามและสี่ผิดเพราะไม่อนุญาตให้ใช้ "$" หรือตัวยกในการตั้งชื่อ ส่วนสองบรรทัดสุดท้ายผิดเพราะคำว่า "xml" เป็นคำสงวนตามกฎการตั้งชื่อในภาษา XML

2.1.1.3 **อีลิเมนต์**

อีลิเมนต์เป็นขอบเขตพิเศษในภาษา XML ซึ่งอาจจะประกอบด้วยอีลิเมนต์อื่น ๆ, character data, character references, entity reference, PIs, หมายเหตุ และ/หรือ CDATA ทั้งหมดนี้ถือเป็นรายละเอียดของอีลิเมนต์ โดยข้อมูลในภาษา XML ทั้งหมดต้องบรรจุอยู่ในอีลิเมนต์ ยกเว้น หมายเหตุ, PIs และช่องว่าง

อีลิเมนต์จะแบ่งโดยใช้แท็กที่ประกอบไปด้วยชื่ออีลิเมนต์และปิดด้วยเครื่องหมาย "<>" ทุก ๆ อีลิเมนต์ต้องเริ่มต้นด้วยแท็กเริ่มต้นและปิดด้วยแท็กปิดท้าย อาจมีข้อยกเว้นสำหรับกฎข้อนี้อย่างหนึ่งคือ การใช้อีลิเมนต์ว่าง ซึ่งก็คือ อีลิเมนต์ที่เริ่มด้วยแท็กเริ่มต้นและต่อด้วยแท็กปิดท้าย

- แท็กเริ่มต้น ประกอบด้วย เครื่องหมาย "<" ตามด้วยชื่ออีลิเมนต์ แล้วปิดด้วย ">"
- แท็กปิดท้าย ประกอบด้วย เครื่องหมาย "<" ตามด้วย "/" ชื่ออีลิเมนต์ ปิดด้วย ">" แท็กปิดท้ายทุกตัวต้องมีชื่ออีลิเมนต์ที่ตรงกันกับแท็กเริ่มต้น เช่น <some_tag> content goes here </some_tag>
- แท็กอีลิเมนต์ว่าง เริ่มด้วยแท็กเริ่มต้น ต่อด้วยแท็กปิดท้าย มีลักษณะดังนี้ <point1></point1> โดยสามารถใช้รูปย่อได้เป็น <point1/>

2.1.1.4 **แอตทริบิวต์**

ถ้าเปรียบอีลิเมนต์เป็นคำนามในภาษา XML แล้ว แอตทริบิวต์ก็สามารถเปรียบได้เป็นคำคุณศัพท์ที่บ่งบอกถึงคุณสมบัติของคำนามนั่นเอง แอตทริบิวต์จะถูกใช้เมื่อต้องการให้ข้อมูลต่าง ๆ เกี่ยวกับอีลิเมนต์นั้น ๆ โดยจะประกอบไปด้วย ชื่อและค่าของแอตทริบิวต์ ซึ่งจะสามารถใช้ใน 2 รูปแบบ ดังภาพที่ 2.4

ชื่อแอตทริบิวต์=”ค่าแอตทริบิวต์”

ชื่อแอตทริบิวต์=’ค่าแอตทริบิวต์’

ภาพที่ 2.4 ภาพแสดงการใช้ชื่อและค่าของแอตทริบิวต์

ค่าของแอตทริบิวต์ อาจจะประกอบไปด้วย entity references, character references และ/หรือ ตัวอักษร โดยไม่สามารถมีแอตทริบิวต์ที่มีชื่อเหมือนกันอยู่ในแท็กเริ่มต้นหรือแท็กอีลิเมนต์ว่างเดียวกัน

```
<carton number=”0-666-42-1”>
…
</carton>
<carton number=”0-666-42-2”>
…
</carton>
<EOF char=”1A”/>
```

ภาพที่ 2.5 ภาพแสดงการใช้แอตทริบิวต์ที่ถูกต้อง

จากภาพที่ 2.5 ชื่อของแอตทริบิวต์ คือ “number” มีค่าเป็น 0-666-42-1 และ 0-666-42-2 ตามลำดับ เช่นเดียวกับในอีลิเมนต์ว่างชื่อ “EOF” ที่มีแอตทริบิวต์ชื่อ “char” ซึ่งมีค่าเป็น 1A

2.1.1.5 **พื้นที่ว่าง**

พื้นที่ว่าง คือ อักขระที่ถูกใช้โดยจะไม่มีการกำหนดความหมาย แต่มีประโยชน์สำหรับผู้ใช้ในแง่ของการเขียนโปรแกรม โดยตัวอักษรที่ใช้เป็นพื้นที่ว่างในภาษา XML มีอยู่ 4 ตัว

ตารางที่ 2.2 ตารางแสดงค่าตัวอักษร 4 ตัวที่ใช้เป็นพื้นที่ว่างในรูปแบบเลขฐานสิบหก

Character Values (in hexadecimal)	Description
09	Horizontal tab (HT)
0A	Line-feed (LF)
0D	Carriage-return (CR)

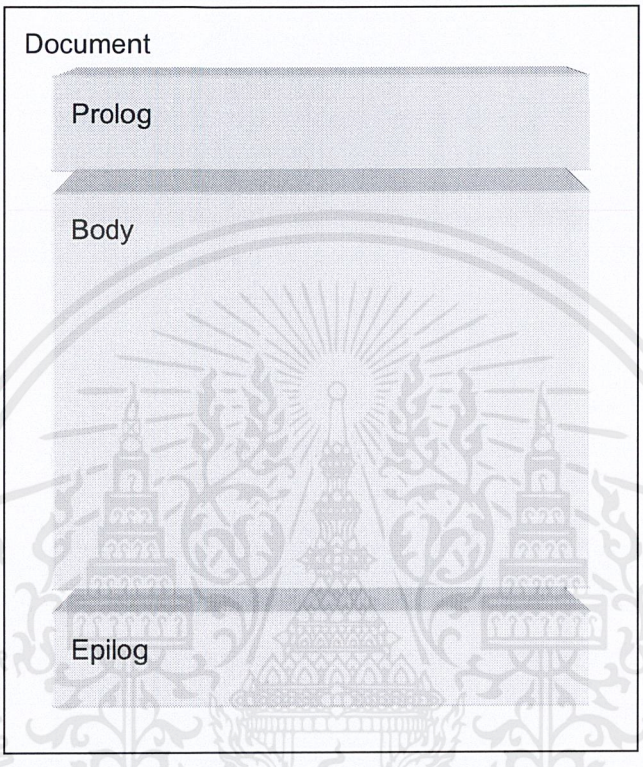
2.1.1.6 **หมายเหตุ**

การสร้างหมายเหตุในเอกสารจะแสดงถึงข้อความพิเศษ เพื่อให้ผู้สร้างสามารถทำความเข้าใจได้และง่ายต่อการสืบค้นภายหลัง หมายเหตุจะปรากฏในตัวเอกสารหรือไม่ก็ได้ ซึ่งจะไม่ถือรวมเป็นส่วนหนึ่งของเอกสารจริงและสามารถใส่ลงที่ส่วนใดของเอกสารก็ได้ โดยข้อความหมายเหตุต้องอยู่ภายใต้เครื่องหมาย “<!- -“ และ “- ->”

2.1.2 โครงสร้างของเอกสาร XML

เอกสาร XML ที่มีรูปแบบถูกต้อง ประกอบด้วย 3 ส่วน ดังนี้

- Prolog
- Body
- Epilog



ภาพที่ 2.6 ภาพแสดงโครงสร้างของเอกสาร XML

Prolog มีหรือไม่ก็ได้ Prolog เป็นส่วนเริ่มของเอกสาร XML ที่ระบุว่าเอกสารนั้นเป็นเอกสาร XML อธิบายถึงวิธีการเข้ารหัสตัวอักษรของเอกสารนั้น และแสดงการกำหนดค่าต่าง ๆ ที่จำเป็นต้องใช้ในการตรวจสอบเอกสาร แล้วส่งต่อให้กับแอพพลิเคชัน โดยประกอบไปด้วย ส่วนประกาศ XML ตามด้วย หมายเหตุ, PIs และ ช่องว่าง

Body ประกอบไปด้วยอีลิเมนต์อย่างน้อยหนึ่งอีลิเมนต์ และอาจจะมี character data ด้วย

Epilog มีหรือไม่ก็ได้ ประกอบด้วย หมายเหตุ, processing instructions (PI), และ/หรือ ช่องว่าง แล้วต่อท้ายด้วยอีลิเมนต์ทรี

2.1.3 ตัวอย่างเอกสาร XML

```
<? xml version="1.0" encoding="utf-8" standalone="yes"?>
<! - - ======= The Wrox Press Book Catalog Application ======= - - >
<Book>
  <Title>Professional XML</Title>
  <Author>
    Mark Birbeck, Steven Livingstone, Didier Matin, Stephen Mohr, Nicola Ozu, et al.
  </Author>
  <Publisher>Wrox Press, Ltd.</Publisher>
  <PubDate>November 1999</PubDate>
  <Abstract>XML 0-500kmh in 3 seconds</Abstract>
  <Pages>750</Pages>
  <ISBN>1-861003-11-0</ISBN>
  <RecSubjCategories>
  <Category>Internet</Category>
  <Category>Web Publisher</Category>
  <Category>XML</Category>
  </RecSubjCategories>
</Book>
```

ภาพที่ 2.7 ภาพแสดงตัวอย่างเอกสาร XML

2.2 ไวยากรณ์ DTD (Document Type Definition)

DTD เป็นไวยากรณ์ประเภทที่ไม่ขึ้นกับเนื้อหา ใช้เพื่อแจ้งรายการโครงสร้างทางตรรกะที่ใช้ในเอกสาร XML

2.2.1 โครงสร้างเอกสาร DTD

เอกสาร DTD ประกอบด้วย อีลิเมนต์, แอตทริบิวต์, เอนติตี, โนเตชัน โดยภายในเอกสารหนึ่ง ๆ จะประกอบด้วยองค์ประกอบทั้ง 4 อย่างหรือน้อยกว่า เรียงต่อกัน

2.2.1.1 **อีลิเมนต์**

ใช้นิยามองค์ประกอบของเอกสารและความสัมพันธ์ระหว่างองค์ประกอบนั้น การแจ้งอีลิเมนต์จะใช้คำสงวน "<! ELEMENT (ชื่ออีลิเมนต์, แบบจำลองเนื้อหา) ตัวบ่งปริมาณ>"

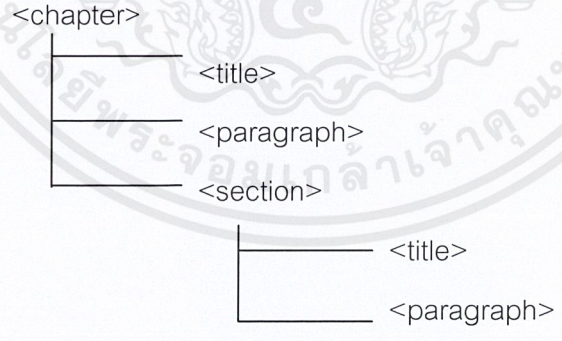
ตารางที่ 2.3 ตารางแสดงตัวบ่งปริมาณและความหมายของตัวบ่งปริมาณ

ตัวบ่งปริมาณ	ชื่อ	ความหมาย
?	Question Mark	มีหรือไม่มีก็ได้ ถ้ามี สามารถมีได้เพียงหนึ่ง
*	Asterisk	มีหรือไม่มีก็ได้ ถ้ามี สามารถมีได้มากกว่าหนึ่ง
+	Plus Sign	มีได้หนึ่งหรือมากกว่าหนึ่ง

```
<?xml encoding="UCS-2"?>
<!ELEMENT chapter (title, (paragraph | section) )>
<!ELEMENT title (#PCDATA)>
<!ELEMENT paragraph (#PCDATA)>
<!ELEMENT section (title,paragraph)>
```

ภาพที่ 2.8 ภาพแสดงการแจ้งอีลิเมนต์

เอกสาร XML ที่มีการนิยามตามภาพที่ 2.8 จะมีโครงสร้างทางตรรกะ เป็นดังภาพที่ 2.9



ภาพที่ 2.9 ภาพแสดงโครงสร้างทางตรรกะของเอกสาร XML ที่นิยามตามภาพที่ 2.8

2.2.1.2 **แอตทริบิวต์**

ใช้นิยามแอตทริบิวต์ต่าง ๆ ของอีลิเมนต์ ประกอบด้วย ชื่อแอตทริบิวต์, ประเภทแอตทริบิวต์ และค่าดีฟอลท์

```
<!ATTLIST    chapter
  author    CDATA    #IMPLIED
>
```

ภาพที่ 2.10 ภาพแสดงการแจ้งรายการแอตทริบิวต์

ภาพที่ 2.10 เป็นการแจ้งแอตทริบิวต์ชื่อ author ซึ่งเป็นแอตทริบิวต์ของอีลิเมนต์ชื่อ chapter โดยแอตทริบิวต์นี้เป็นประเภท CDATA คือ ข้อมูลตัวอักษร และค่าดีฟอลท์ คือ #IMPLIED ซึ่งหมายถึงไม่ได้มีการกำหนดค่าอะไรเป็นพิเศษ ในกรณีที่อีลิเมนต์มีหลายแอตทริบิวต์ ก็ให้ทำเช่นเดียวกับแถวที่ 2 ของภาพที่ 2.10 คือแจ้งชื่อแอตทริบิวต์, ประเภทแอตทริบิวต์ และค่าดีฟอลท์ของแต่ละแอตทริบิวต์เป็นแถว ๆ แล้วปิดท้ายด้วยเครื่องหมาย “>”

ตารางที่ 2.4 ตารางแสดงค่าที่เป็นไปได้ของค่าดีฟอลท์สำหรับใช้ในการแจ้งแอตทริบิวต์

ค่าดีฟอลท์	ความหมาย
#REQUIRED	แอตทริบิวต์ต้องถูกประกาศในเอกสาร
#IMPLIED	สามารถประกาศหรือไม่ก็ได้ ถ้าไม่ประกาศ จะมีค่าเท่ากับค่าที่แอปพลิเคชั่นกำหนดไว้ให้
“defaultValue”	มีค่าที่กำหนดให้เป็นไปตามค่าภายใต้เครื่องหมาย “ “ ในกรณีที่ไม่ได้มีการประกาศค่าไว้
#FIXED ”fixedValue”	กำหนดค่าแอตทริบิวต์นั้นมีค่าคงที่เท่ากับค่าภายใต้เครื่องหมาย “ “

2.2.1.3 **เอนติตี**

ใช้เพื่อกำหนดชื่อให้แก่สารสนเทศ ซึ่งจะทำให้การอ้างอิงสารสนเทศในเอกสารเป็นไปโดยสะดวก โดยสามารถใช้ชื่อที่ให้นั้นในการอ้างอิง

```
<!ENTITY  image1
SYSTEM   “filename.gif”
NDATD    gif
>
```

ภาพที่ 2.11 ภาพแสดงการแจ้งเอนติตีที่อยู่ภายนอกเอกสาร

จากภาพที่ 2.11 “ENTITY” เป็นคำสงวนสำหรับการแจ้งเอนติตี โดยที่ image1 เป็นชื่อที่กำหนดให้แก่สารสนเทศ ซึ่งเป็นภาพที่อยู่ในไฟล์ชื่อ “filename.gif” ส่วน “SYSTEM” เป็นคำสงวนที่ใช้บอกว่าสารสนเทศนั้นอยู่ในระบบภายนอกและไม่ได้เป็นส่วนหนึ่งของเอกสาร XML นี้ “NDATA” เป็นคำสงวนใช้บอกว่าข้อมูลเป็นแบบไบนารีและ gif เป็นชื่อประเภทข้อมูล หลังจากการประกาศตามภาพที่ 2.11 แล้ว เมื่อมีการอ้างอิงชื่อ image1 ในเอกสาร สารสนเทศที่ปรากฏขึ้นมาจริงจะเป็นไฟล์กราฟิก filename.gif

สารสนเทศที่อ้างอิงอาจอยู่ในเอกสารเองก็ได้ ไม่จำเป็นต้องเป็นสิ่งที่อยู่นอกเอกสารเสมอไป

```
<!ENTITY   chap2 “Chapter 2: ทำไม XML จึงจำเป็น ? ”>
```

ภาพที่ 2.12 ภาพแสดงการแจ้งเอนติตีที่อยู่ภายในเอกสาร

2.2.1.4 โนเตชัน

ใช้นิยามความเกี่ยวข้องระหว่างสัญลักษณ์ชื่อกับโปรแกรม หรือการประมวลผลภายนอกที่ไม่ได้อยู่ในส่วนของการประมวลผล XML ซึ่งสามารถถูกเรียกมาใช้งานได้ โดยสัญลักษณ์ชื่อนั้นต้องถูกแจ้งไว้ก่อน

```
<!NOTATION          gif  SYSTEM “/programs/viewgif.exe”>
```

ภาพที่ 2.13 ภาพแสดงการแจ้งโนเตชัน

ภาพที่ 2.13 เป็นการอ้างอิงชื่อ gif กับโปรแกรมการดูภาพกราฟิกชื่อ viewgif.exe ทำให้ระบบประมวลผล XML เรียกโปรแกรม viewgif.exe มาแสดงภาพ image1 ได้อย่างถูกต้อง

2.2.2 ตัวอย่างเอกสาร DTD

```
<!ELEMENT Author (FirstName , MI? , LastName , Biographical , Portrait ) >
<!ATTLIST Author authorCiteID ID #REQUIRED>
<!ELEMENT FirstName (#PCDATA) >
<!ELEMENT MI (#PCDATA) >
<!ELEMENT LastName (#PCDATA) >
<!ELEMENT Biographical (#PCDATA) >
<!ELEMENT Portrial EMPTY >
<!ATTLIST  Protriat piclink CDATA #IMPLIED >
```

ภาพที่ 2.14 ภาพแสดงตัวอย่างเอกสาร DTD

เอกสาร XML ที่ประกาศโครงสร้าง DTD ตามตัวอย่างข้างบน จะมีโครงสร้างทางตรรกะดังแผนภาพด้านล่าง

ภาพที่ 2.15 ภาพแสดงโครงสร้างทางตรรกะของ DTD

อีลิเมนต์ชื่อ “MI” สามารถมีหรือไม่ก็ได้ ถ้ามี จะมีได้เพียงหนึ่งในส่วนของแอตทริบิวต์ อีลิเมนต์ชื่อ Author มีแอตทริบิวต์ชื่อ “authorCiteID” ประเภทแอตทริบิวต์เป็น ID ซึ่งเป็นข้อมูลที่เป็นหนึ่งเดียวในเอกสาร มีค่าดีฟอลท์เป็น #REQUIRED นั่นคือ ต้องมีแอตทริบิวต์นี้ในอีลิเมนต์ที่มีการประกาศ ส่วนอีลิเมนต์ชื่อ “Portrait” มีแอตทริบิวต์ชื่อ “piclink” ประเภทแอตทริบิวต์เป็น CDATA มีค่าดีฟอลท์คือ #IMPLIED หมายความว่า สามารถมีแอตทริบิวต์ตัวนี้หรือไม่ก็ได้

2.3 ภาษา XSL (eXtensible Stylesheet Language)

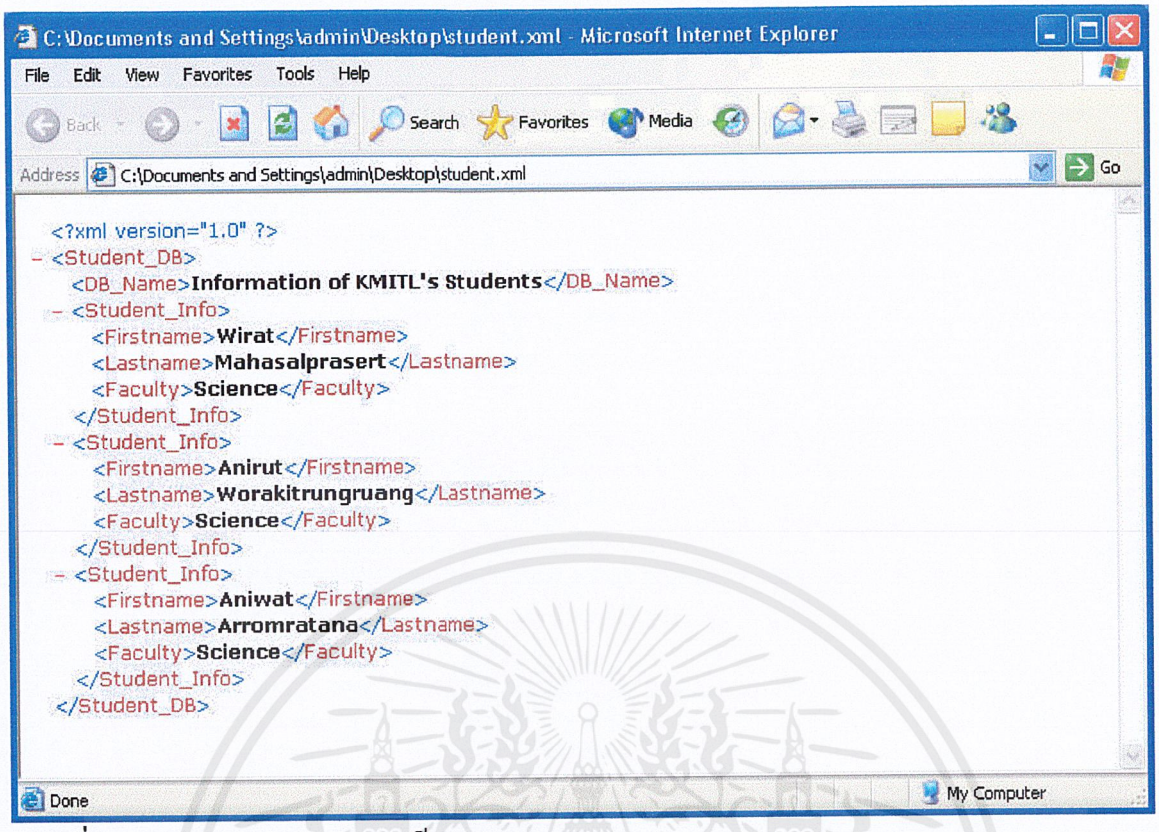
ภาษา XSL เป็นภาษาที่มีพื้นฐานมาจากภาษา XML ถูกออกแบบขึ้นโดยองค์กร W3C เพื่อใช้ในการกำหนดรูปแบบการแสดงผลหรือรูปแบบการนำเสนอข้อมูลให้กับเอกสาร XML ประกอบด้วย 2 ส่วน คือ

- XSLT (XSL Transformation) คือ ส่วนที่ใช้สำหรับการแปลงรูปเอกสาร XML ให้เป็นเอกสารที่มีโครงสร้างแตกต่างออกไป

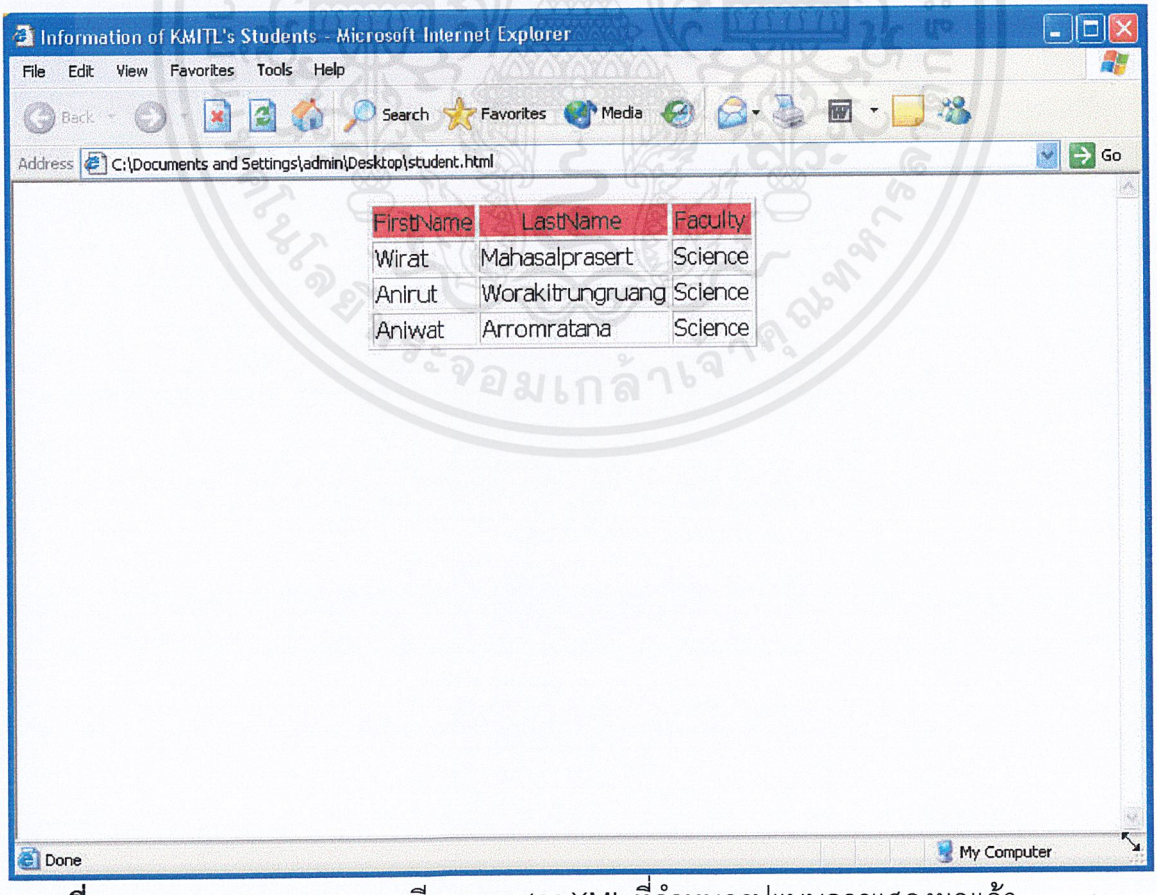
- XSLF (XSL Formatting object) คือ ส่วนที่กำหนดรูปแบบการแสดงผลให้กับเอกสาร XML เช่น การกำหนดขนาดของตัวอักษร, การวางหน้ากระดาษ เป็นต้น

การเรียกดูเอกสาร XML ที่ไม่ได้มีการกำหนดรูปแบบการแสดงผลผ่านทางเบราเซอร์ ผลจากการเรียกดูเอกสารเป็นดังภาพที่ 2.16 ซึ่งแสดงรายละเอียดของเอกสาร XML ออกมาทั้งหมด

เพื่อให้บราวเซอร์แสดงแต่ข้อมูลที่อยู่ภายในเอกสารเมื่อมีการเรียกดู เราสามารถใช้ภาษา XSL กำหนดรูปแบบการแสดงผลให้กับเอกสาร XML ได้ โดยที่รูปแบบการแสดงผลสามารถอยู่ในรูปแบบต่าง ๆ มากมาย เช่น รูปตารางดังภาพที่ 2.17 หรืออยู่ในรูปเอกสารที่สื่อซึ่งใช้เข้าถึงข้อมูลต่าง ๆ สามารถเรียกดูเอกสารได้ ทั้งนี้ ขึ้นอยู่กับการเขียนเอกสารที่ใช้กำหนดรูปแบบการแสดงผลของผู้ให้บริการข้อมูล



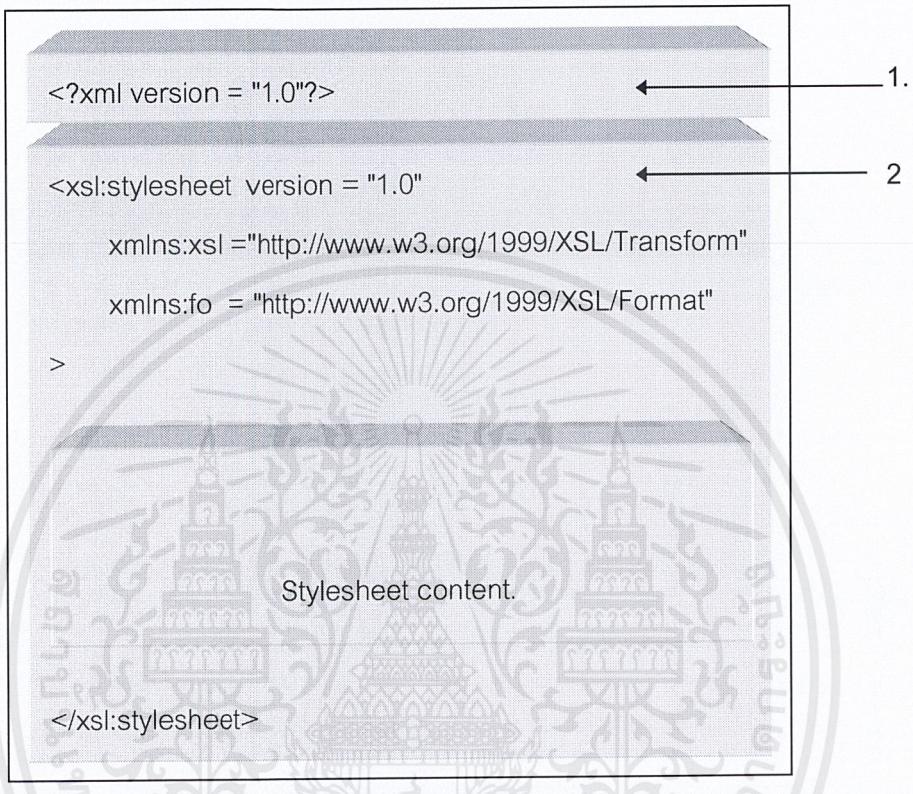
ภาพที่ 2.16 ภาพแสดงผลจากการเรียกดูเอกสาร XML



ภาพที่ 2.17 ภาพแสดงผลจากการเรียกดูเอกสาร XML ที่กำหนดรูปแบบการแสดงผลแล้ว

2.3.1 โครงสร้างเอกสาร XSL

เนื่องจาก XSL เป็นภาษาที่มีพื้นฐานมาจากภาษา XML ดังนั้นโครงสร้างของเอกสาร XSL จึงมีโครงสร้างที่เหมือนกับเอกสาร XML ถ้าไม่พิจารณาในส่วนของ Epilog จะมีส่วนประกอบหลัก 2 ส่วน ดังแสดงในภาพที่ 2.18



ภาพที่ 2.18 ภาพแสดงโครงสร้างของเอกสาร XSL

จากภาพที่ 2.18 จะขอกล่าวถึงรายละเอียดในโครงสร้างแต่ละส่วนดังนี้

ส่วนที่ 1 คือ ส่วนที่ประกาศเพื่อบอกข้อมูลต่าง ๆ ของภาษา XML ที่เอกสารนี้ใช้ในการอ้างอิง

ส่วนที่ 2 คือ ส่วนที่ประกาศว่าเป็นเอกสารที่เขียนด้วยภาษา XSL บอกข้อมูลเกี่ยวกับภาษา XSL ที่ใช้สำหรับเขียนเอกสาร ได้แก่ เวอร์ชัน, XML namespaces ของส่วน XSLT และส่วน XSLF ตามลำดับ XML namespaces ในส่วนของ XSLF นั้น ให้ประกาศเมื่อต้องการใช้แท็กคำสั่งในส่วนของ XSLF เท่านั้น

ส่วน Stylesheet content เป็นรายละเอียดของการกำหนดรูปแบบการแสดงผล รายละเอียดของอีลิเมนต์และความสัมพันธ์ระหว่างอีลิเมนต์ สามารถดูได้จาก DTD ของเอกสาร XSL ในภาคผนวก ง.

เพื่อให้เกิดความเข้าใจมากขึ้น ขอยกตัวอย่างพร้อมคำอธิบายในแต่ละส่วนดังนี้

```
1       <?xml version = "1.0"?>
2       <xsl:stylesheet xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">
3       <xsl:output method = "html" doctype-public = "-//W3C//DTD HTML 4.0
Transitional//EN"/>
4       <xsl:template match="/">
5       <html>
6       <xsl:apply-templates />
7       </html>
8       </xsl:template>
9       <xsl:template match = "body" >
10      <body>
11      <xsl:apply-templates />
12       </body>
13      </xsl:template>
14      <xsl:template match = "b" >
15      <b>
16      <xsl:apply-templates />
17      </b>
18      </xsl:template>
19      </xsl:stylesheet>
```

ภาพที่ 2.19 ภาพแสดงตัวอย่างเอกสาร XSL

จากภาพที่ 2.19 บรรทัดที่ 1 และ 2 เป็นการประกาศในส่วนที่ 1. และ 2. ตามลำดับ บรรทัดที่ 3 เป็นแท็กคำสั่ง <xsl:output> ซึ่งเป็นแท็กคำสั่งในภาษา XSL เพื่อใช้ในการระบุว่าเอกสารที่ได้มาจากการกำหนดรูปแบบการแสดงผลนั้น เป็นเอกสารที่มีโครงสร้างภาษาเป็นอย่างไร จากตัวอย่างนี้ ระบุว่าเอกสารที่ได้เป็นเอกสารที่มีโครงสร้างภาษาเป็นภาษา HTML บรรทัดที่ 4 เป็นแท็กคำสั่ง <xsl:template match = "/"> ที่กำหนดให้ทำตามคำสั่งที่อยู่ระหว่าง <xsl:template match = "/"> กับ </xsl:template> เมื่อพบแท็กซึ่งอยู่ภายใต้เครื่องหมาย " " ในเอกสารที่เขียนขึ้นด้วยภาษา XML ในตัวอย่างนี้ กำหนดให้ทำตามคำสั่งเมื่อเริ่มทำการกำหนดรูปแบบการแสดงผล ส่วนในบรรทัดที่ 6 เป็นแท็กคำสั่ง <xsl:apply-templates/> ที่กำหนดให้ทำการค้นหาแท็กต่อไป เพื่อทำการจับคู่กับแท็กที่กำหนดไว้ในเอกสารที่เขียนขึ้นด้วยภาษา XSL

2.3.2 ตัวอย่างการใช้งานเอกสาร XSL

กำหนดเอกสาร XML ให้มีรายละเอียดดังนี้

```
<?xml version = "1.0" ?>
<Document>
<body> <b>XSL</b> is a language for expressing stylesheet.</body>
</Document>
```

ภาพที่ 2.20 ภาพการกำหนดรายละเอียดของเอกสาร XML

กำหนดรูปแบบการแสดงผลให้เป็นเอกสาร HTML ข้อความที่อยู่ระหว่าง กับ เป็นตัวหนา สามารถเขียนเอกสารเพื่อกำหนดรูปแบบการแสดงผลได้ดังนี้

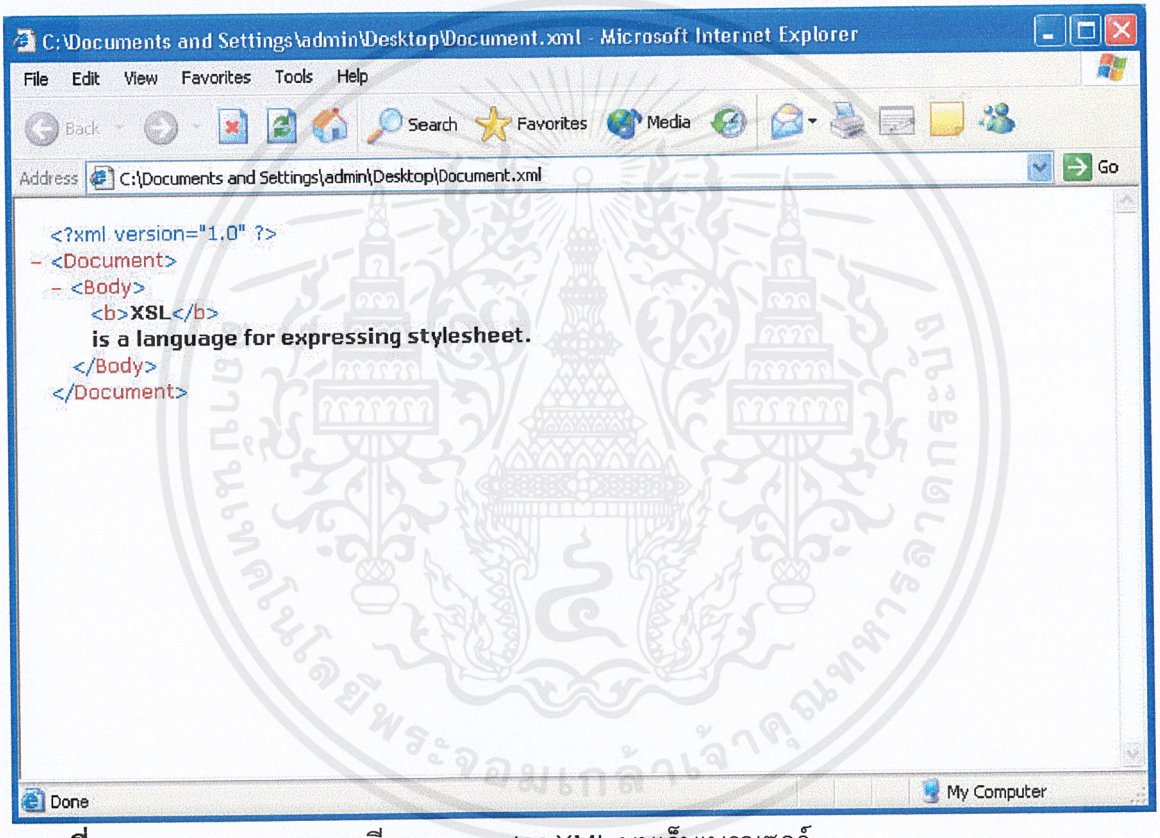
```
<?xml version = "1.0" ?>
<xsl:stylesheet xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" doctype-public="-//W3C//DTD HTML 4.0
Transitional//EN"/>
<xsl:template match="/">
  <html> <xsl:apply-templates /></html>
</xsl:template>
<xsl:template match = "body" >
  <body><xsl:apply-templates /></body>
</xsl:template>
<xsl:template match = "b" >
  <b> <xsl:apply-templates /></b>
</xsl:template>
</xsl:stylesheet>
```

ภาพที่ 2.21 ภาพการกำหนดรายละเอียดของเอกสาร XML ให้แสดงผลเป็นเอกสาร HTML

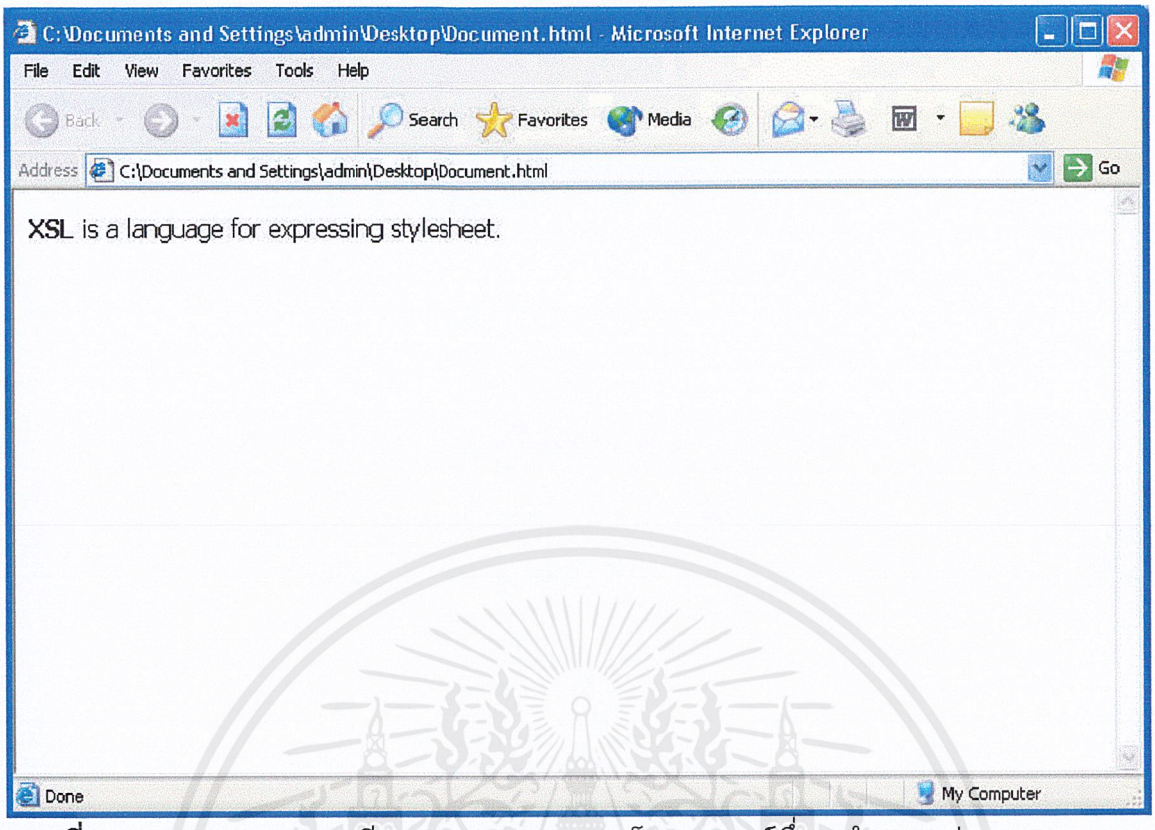
ผลที่ได้จากการใช้งานเอกสาร XSL เป็นดังนี้

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
  <body> <b>XSL</b> is a language for expressing stylesheet.
  </body>
</html>
```

ภาพที่ 2.22 ภาพผลของการใช้งานเอกสาร XSL



ภาพที่ 2.23 ภาพแสดงรายละเอียดของเอกสาร XML บนเว็บเบราเซอร์



ภาพที่ 2.24 ภาพแสดงผลการเรียกดูเอกสาร XML บนเว็บเบราเซอร์ ซึ่งถูกกำหนดรูปแบบการแสดงผลเอาไว้

2.4 โพรโตคอล HTTP (Hypertext Transfer Protocol)

HTTP ย่อมาจากคำว่า Hypertext Transfer Protocol เป็นโพรโตคอลที่ใช้ในการส่งข้อมูลต่าง ๆ ในโลกของ World Wide Web ข้อมูลต่าง ๆ เหล่านี้โดยทั่วไปมักจะถูกเรียกว่า “Resource” โดย Resource เหล่านี้อาจเป็นไฟล์ เช่น ไฟล์ HTML, ไฟล์ภาพ หรือคำสั่งต่าง ๆ เช่น คำสั่งที่ส่งไปที่โปรแกรม CGI หรืออาจเป็น binary stream ในกรณีของการ ดาวน์โหลด/อัพโหลด ไฟล์ หรืออาจจะเป็นสิ่งอื่น ๆ อีกมากมายตามแต่จะกำหนดขึ้น

HTTP เป็นโพรโตคอลที่อยู่ในส่วนของแอพพลิเคชันเลเยอร์ใน protocol stack โดยข้อมูลต่าง ๆ จากเลเยอร์นี้จะถูกส่งผ่านไปยังเลเยอร์อื่น ๆ ที่ต่ำกว่า ซึ่งส่วนหนึ่งในนั้นก็คือโพรโตคอล TCP/IP นั่นเอง

HTTP เป็นโพรโตคอลสำหรับการสื่อสารบนเครือข่าย ที่ใช้หลักการของไคลเอนท์/เซิร์ฟเวอร์สำหรับการติดต่อสื่อสาร ซึ่งหลักการทำงานอย่างคร่าว ๆ มีดังนี้

- ไคลเอนท์จะเชื่อมต่อไปหาเซิร์ฟเวอร์ ซึ่งโดยทั่วไปจะเชื่อมต่อผ่านทางซ็อคเก็ตของ TCP/IP
- หลังจากนั้นไคลเอนท์จะทำการส่งคำร้องขอ ซึ่งอยู่ในรูปของเมสเสจ ไปให้เซิร์ฟเวอร์ เพื่อร้องขอ Resource
- เซิร์ฟเวอร์จะทำการตีความคำร้องขอที่ได้รับและตอบสนองเป็น Resource กลับไปที่ไคลเอนท์ โดยส่งกลับมาเป็นลักษณะของเมสเสจ คล้ายกับคำร้องขอของไคลเอนท์ที่ส่งมาให้เซิร์ฟเวอร์
- หลังจากที่การตอบสนองเสร็จสิ้น เซิร์ฟเวอร์จะทำการปิดการเชื่อมต่อที่มาจากไคลเอนท์
- ในกรณีที่ไคลเอนท์ต้องการ Resource อื่น ๆ ไคลเอนท์จะต้องทำการสร้างการเชื่อมต่อใหม่และส่งคำร้องขอไปยังเซิร์ฟเวอร์อีกครั้ง

จากหลักการข้างต้น จะเห็นว่าการติดต่อสื่อสารระหว่างไคลเอนท์และเซิร์ฟเวอร์ เป็นลักษณะครั้งต่อครั้ง ในด้านเครือข่าย เราเรียกการติดต่อสื่อสารแบบนี้ว่า “Stateless”

อันที่จริงแล้ว เว็บเบราเซอร์ก็คือไคลเอนท์นั่นเอง เหตุผลคือเราใช้เว็บเบราเซอร์ในการส่งคำร้องขอไปยังเว็บเซิร์ฟเวอร์ เพื่อจะรับ Resource ที่ต้องการโดยใช้โพรโตคอล HTTP โดยทั่วไป Resource จะกระจายอยู่ตามโหนดต่าง ๆ ในระบบเครือข่ายทั่วโลก สิ่งหนึ่งที่ขาดไม่ได้ในการอ้างถึง Resource เหล่านี้คือ URL(Universal Resource Locator) URL ใช้ชี้ถึงแหล่งที่อยู่ของ Resource ว่าอยู่ที่ไหน ซึ่งจริง ๆ แล้ว URL สามารถใช้เพื่ออ้างถึง Resource อะไรก็ได้ โดยใช้โพรโตคอลอะไรก็ได้ ยกตัวอย่าง เช่น

http://www.jarticles.com/index.html
เป็น URL ของโพรโตคอล HTTP เพื่อใช้โหลดไฟล์ html
http://www.jarticles.com:80/coffee.jpg
เป็น URL ของโพรโตคอล HTTP เพื่อใช้โหลดไฟล์ภาพ
http://www.jarticles.com/cgi-bin/sendMail.pl
เป็น URL ของโพรโตคอล HTTP เพื่อใช้โหลดค่าจากการประมวลผลของโปรแกรม CGI ชื่อ sendMail.pl
ftp://ftp.jarticles.com/jspTutorial.doc
เป็น URL ของโพรโตคอล FTP เพื่อใช้ดาวน์โหลดไฟล์เอกสาร

ภาพที่ 2.25 ภาพตัวอย่างของ URL สำหรับการร้องขอ Resource ต่าง ๆ

จากตัวอย่างข้างต้น เราจะเห็นว่าในหนึ่ง URL จะประกอบด้วย ชื่อของโพรโตคอล, ชื่อเซิร์ฟเวอร์, พอร์ทที่ใช้และ Resource ที่ต้องการ ดังรูปแบบข้างล่าง

```
http://www.jarticles.com:80/index.html
------ ---------------------- ------------------
  |              |          |       |
Protocol   Hostname    Port  Resource
```

ภาพที่ 2.26 ภาพส่วนประกอบของ URL

2.4.1 โครงสร้างของโพรโตคอล HTTP

จากหลักการทำงานของ HTTP ข้างต้น เราจะเห็นว่าในการติดต่อสื่อสารระหว่างไคลเอนท์กับเซิร์ฟเวอร์ เมสเสจจะเป็นตัวกลางที่ใช้ในการติดต่อสื่อสารเสมอ และเพื่อให้ง่ายต่อการใช้งาน รูปแบบของเมสเสจที่ใช้ในคำร้องขอหรือการตอบสนองจึงมีลักษณะคล้าย ๆ กัน โดยประกอบไปด้วย

- บรรทัดเริ่มต้น (initial line)
- เฮดเดอร์
- บรรทัดว่าง (blank line) ซึ่งก็คือ CRLF หรือการเว้นหนึ่งบรรทัด
- ข้อความ ซึ่งอาจจะใช้บรรจุไฟล์, คำสั่งต่าง ๆ หรืออาจจะเป็นผลลัพธ์ที่มาจากเซิร์ฟเวอร์ โดยส่วนนี้จะมีหรือไม่ก็ได้ ขึ้นอยู่กับจุดประสงค์ของการใช้งาน

รูปแบบโดยทั่วไปของเมสเสจจะเป็น

```
<initial line>
Header1: value1
Header2: value2
Header3: value3
[blank line]
<message body, optional .....>
```

ภาพที่ 2.27 ภาพโครงสร้างของเมสเสจ

GET /images/coffee.jpg	<-- บรรทัดเริ่มต้น
From: soup@jarticles.com	<-- เฮดเดอร์
User-Agent: Mozilla/4.72	<-- เฮดเดอร์
[blank line]	<-- บรรทัดว่าง
หรือ	
POST /servlet/searchEngine HTTP/1.0	<-- บรรทัดเริ่มต้น
From: webmaster@jarticles.com	<-- เฮดเดอร์
User-Agent: Mozilla/4.72	<-- เฮดเดอร์
[blank line]	<-- บรรทัดว่าง
keyword=java&topic=jsp	<-- ข้อความ

ภาพที่ 2.28 ภาพตัวอย่างของเมสเสจที่ไคลเอนท์ใช้

HTTP 1.0 200 OK	<-- บรรทัดเริ่มต้น
Date: Sunday, 23-July-00 04:01:12 GMT	<-- เฮดเดอร์
Server: Apache/1.3.12(Unix) (Red Hat/Linux) PHP/3.0.15 mod_perl/1.21	<-- เฮดเดอร์
MIME-version: 1.0	<-- เฮดเดอร์
Content-type: text/html	<-- เฮดเดอร์
Content-length: 115	<-- เฮดเดอร์
[blank line]	<-- บรรทัดว่าง
<HTML><HEAD><TITLE>HTTTP Tutorial</TITLE></HEAD>	<-- ข้อความ
<BODY>This is a tutorial, but please visit me again..</BODY>	<-- ข้อความ
<HTML>	<-- ข้อความ

ภาพที่ 2.29 ภาพตัวอย่างของเมสเสจที่เซิร์ฟเวอร์ใช้

2.4.2 **บรรทัดเริ่มต้นของคำร้องขอ** (Initial Request Line)

บรรทัดเริ่มต้นของคำร้องขอจะแตกต่างจากบรรทัดเริ่มต้นของการตอบสนองเล็กน้อย โดยมี 3 ส่วน คือ

- ชื่อของเมตธอด เช่น GET, POST, HEAD, TRACE
- ตำแหน่งของ Resource ที่ไคลเอนท์ต้องการ
- เวอร์ชันของ HTTP/x.x ที่ไคลเอนท์ใช้

สามส่วนนี้จะประกอบกันเป็นบรรทัดเริ่มต้น โดยแต่ละส่วนจะถูกแยกออกจากกันด้วยช่องว่าง

GET /path/to/file/index.html HTTP/1.0

ภาพที่ 2.30 ภาพตัวอย่างบรรทัดเริ่มต้นของคำร้องขอ

ตัวอย่างนี้ใช้เมตธอดที่ชื่อ GET เพื่อขอ Resource ซึ่งในที่นี้คือไฟล์ชื่อ path/to/file/index.html โดยโพรโตคอล HTTP เวอร์ชัน 1.0 นอกจากนี้ ยังสังเกตจากตัวอย่างได้ว่า ชื่อของ method จะต้องใช้ตัวใหญ่เสมอ และเวอร์ชันของ HTTP จะอยู่ในรูป HTTP/x.x และจะต้องเป็นตัวใหญ่เสมอ

2.4.3 **บรรทัดเริ่มต้นของการตอบสนอง** (Initial Response Line)

โดยทั่วไป บรรทัดเริ่มต้นของการตอบสนองมักจะเรียกว่า “Status line” ซึ่งประกอบไปด้วย 3 ส่วนย่อย คือ

- เวอร์ชันของ HTTP/x.x ที่เซิร์ฟเวอร์ใช้สำหรับส่งเมสเสจ
- รหัสแสดงสถานะของการตอบสนอง (Response status code) ทำหน้าที่บอกว่าผลการร้องขอที่ไคลเอนท์ส่งมาเป็นอย่างไร
- คำอธิบาย (Reason phase) อธิบายความหมายของรหัสแสดงสถานะของการตอบสนองอีกทีหนึ่ง

HTTP/1.0 200 OK

หรือ

HTTP/1.0 404 Not Found

ภาพที่ 2.31 ภาพตัวอย่างบรรทัดเริ่มต้นของการตอบสนอง

จากตัวอย่าง จะสังเกตได้ว่า เวอร์ชันของ HTTP จะต้องอยู่ในรูปของ HTTP/x.x และจะต้องเป็นตัวใหญ่เสมอ

รหัสแสดงสถานะของการตอบสนองจะอยู่ในลักษณะของเลขสามหลัก โดยหลักแรกจะบอกถึงความหมายทั่ว ๆ ไปของการตอบสนอง

- 1xx เป็นรหัสที่ใช้บอกถึงเหตุการณ์ต่าง ๆ ที่เกิดขึ้นในการสื่อสารระหว่างไคลเอนท์และเซิร์ฟเวอร์
- 2xx เป็นรหัสที่บ่งบอกว่าการร้องขอที่ส่งจากไคลเอนท์ได้รับการตอบสนองอย่างสมบูรณ์แล้ว
- 3xx เป็นรหัสที่บอกให้ไคลเอนท์ทำการเชื่อมต่อไปยัง URL อื่นแทน
- 4xx เป็นรหัสที่บ่งบอกถึงความผิดพลาดที่เกิดขึ้นจากส่วนของไคลเอนท์
- 5xx เป็นรหัสที่บ่งบอกถึงความผิดพลาดที่เกิดขึ้นจากส่วนของเซิร์ฟเวอร์

100 Continue

เซิร์ฟเวอร์ได้รับคำร้องขอบางส่วนจากไคลเอนท์แล้ว ให้ไคลเอนท์ส่งส่วนที่เหลือมาให้

200 OK

คำร้องขอที่มาจากไคลเอนท์ถูกต้อง และ Resource ที่ไคลเอนท์ต้องการอยู่ในข้อความของการตอบสนองนี้

301 Moved Permanently

Resource ที่ไคลเอนท์ต้องการเคยอยู่ที่เซิร์ฟเวอร์นี้ แต่ถูกย้ายไปอยู่ที่อื่นแล้ว

302 Moved Temporarily

Resource ที่ไคลเอนท์ต้องการถูกย้ายไปอยู่ที่เซิร์ฟเวอร์อื่น หรือไม่สามารถเข้าถึงได้ชั่วคราว

400 Bad Request

เซิร์ฟเวอร์ไม่สามารถเข้าใจคำร้องขอที่ไคลเอนท์ส่งมา

403 Forbidden

เซิร์ฟเวอร์เข้าใจคำร้องขอที่ไคลเอนท์ส่งมา แต่ไม่ต้องการที่จะส่ง Resource ที่ไคลเอนท์ต้องการกลับไปให้

404 Not Found

เซิร์ฟเวอร์ไม่มี Resource ที่ไคลเอนท์ต้องการ

500 Server Error

เกิดข้อผิดพลาดขึ้นในส่วนของเซิร์ฟเวอร์

ภาพที่ 2.32 ภาพตัวอย่างรหัสแสดงสถานะของการตอบสนองและคำอธิบายที่พบบ่อย ๆ

2.4.4 เฮดเดอร์

โดยทั่วไปเฮดเดอร์จะเป็นส่วนที่บอกถึงรายละเอียดของคำร้องขอที่กำลังส่งไปให้เซิร์ฟเวอร์ หรือการตอบสนองที่กำลังส่งกลับมายังไคลเอนท์

ไคลเอนท์จะส่งเฮดเดอร์ที่บ่งบอกชนิดและขนาดของข้อมูลที่อยู่ข้างในข้อความในกรณีที่ไคลเอนท์ต้องการจะอัพโหลดไฟล์ไปยังเซิร์ฟเวอร์

เซิร์ฟเวอร์ส่งเฮดเดอร์ที่เกี่ยวกับชนิดและขนาดของ Resource ที่ไคลเอนท์กำลังจะได้รับ โดยใช้ Content-Type และ Content-Length

เฮดเดอร์เป็นข้อความที่ประกอบด้วยตัวอักษรธรรมดา หนึ่งบรรทัดจะถูกใช้สำหรับหนึ่งเฮดเดอร์ ซึ่งจะอยู่ในรูปของ "Header-Name: value" และจบด้วย CRLF อาจมีช่องว่างหรือแท็บระหว่าง ":" ของ Header-Name และ value กว้างเท่าไรก็ได้ ชื่อของเฮดเดอร์สามารถใช้ตัวใหญ่หรือตัวเล็กก็ได้ ใน HTTP เวอร์ชัน 1.0 มีเฮดเดอร์กำหนดไว้ 16 แบบ แต่เวอร์ชัน 1.1 จะมีถึง 46 แบบด้วยกัน

```
From: soup@jarticles.com
User-Agent: Mozilla/4.72
Content-Type: text/html
Content-Length: 250
```

ภาพที่ 2.33 ภาพตัวอย่างของเฮดเดอร์

ในเฮดเดอร์ที่มักพบเห็นทั่วไปในส่วนของไคลเอนท์ “From:” ใช้เก็บอีเมล์แอดเดรสของไคลเอนท์ เช่น From: soup@jarticles.com ส่วน “User-Agent:” ใช้บอกถึงชื่อของโปรแกรมที่ใช้เป็นไคลเอนท์สำหรับการร้องขอ เช่น User-Agent: Mozilla/4.72

ในเฮดเดอร์ที่พบเห็นทั่วไปในส่วนของเซิร์ฟเวอร์ “Server:”คล้ายกับ “User-Agent:” แต่จะบอกถึงชื่อโปรแกรมที่ใช้เป็นเซิร์ฟเวอร์ในการตอบสนอง เช่น Server: Apache/1.93 ขณะที่ “Last-Modified:” เป็นเฮดเดอร์ที่ใช้บอกถึงเวลาล่าสุดเมื่อ Resource ที่ไคลเอนท์ต้องการถูกเปลี่ยนแปลง โดยเวลาที่ใช้จะเทียบจาก GMT (Greenwich Mean Time) ยกตัวอย่างเช่น เวลาของเมืองไทยถือว่าเป็น GMT+7.00 ถ้าตอนนี้เวลาในเมืองไทยคือ Sun, 23 July 2000 20:39:56 เวลาที่เป็น GMT ก็คือ Last-Modified: Sun, 23 July 2000 13:39:56 GMT

เมื่อไรก็ตามที่ไคลเอนท์หรือเซิร์ฟเวอร์ต้องการส่งข้อมูลไปกับเมสเสจ ส่วนที่เป็นข้อความจะเป็นส่วนที่ใช้สำหรับเก็บข้อมูลดังกล่าว ในกรณีของไคลเอนท์ ตัวข้อความอาจใช้สำหรับบรรจุไฟล์ที่ต้องการอัพโหลด หรือใช้สำหรับเก็บข้อมูลที่มาจากอีลิเมนต์ต่าง ๆ ของฟอร์มในภาษา HTML แล้วส่งไปยังเซิร์ฟเวอร์ก็ได้ ในกรณีของเซิร์ฟเวอร์ ข้อความ จะเป็นส่วนที่ใช้เก็บ Resource ที่ไคลเอนท์ร้องขอ

หรืออาจใช้สำหรับเก็บคำอธิบายต่าง ๆ ในกรณีที่มีความผิดพลาดเกิดขึ้นก็ได้ และถ้าเมสเสจมีส่วนของข้อความอยู่ด้วย ไคลเอนท์หรือเซิร์ฟเวอร์มักจะเพิ่มเฮดเดอร์ที่ช่วยบอกถึงรายละเอียดของข้อความดังกล่าวด้วย ยกตัวอย่างเช่น ถ้าเซิร์ฟเวอร์ส่ง Resource ที่เป็นไฟล์ภาพมาให้ไคลเอนท์ เฮดเดอร์ที่ถูกเพิ่มขึ้นมาก็จะเป็นดังภาพ

Content-Type: image/gif

Content-Length: 1026

ภาพที่ 2.34 ภาพแสดงส่วนที่เพิ่มขึ้นมาในเฮดเดอร์เมื่อมีส่วนของข้อความ

"Content-Type:" บอกถึง MIME type ของข้อมูลในส่วนข้อความ ขณะที่ "Content-Length:" บอกถึงขนาดของข้อความในหน่วยไบท์

ถ้าไคลเอนท์ต้องการ Resource จาก http://www.jarticles.com/tutorials/basic/helloworld.html การส่งเมสเสจระหว่างไคลเอนท์และเซิร์ฟเวอร์จะเป็นดังนี้

ขั้นแรก ไคลเอนท์ทำการเรียกใช้ซ็อคเก็ตเพื่อติดต่อไปที่เซิร์ฟเวอร์ชื่อ www.jarticles.com โดยปกติจะติดต่อไปที่พอร์ท 80 ซึ่งเป็นค่าดีฟอลท์ของโพรโตคอล HTTP แต่กรณีที่ต้องการติดต่อไปที่พอร์ทอื่น ก็เพิ่มพอร์ท เช่น http://www.jarticles.com:8080/tutorials/advance/hellocorba.html

หลังจากนั้นไคลเอนท์จะทำการส่งคำร้องขอไปยังเซิร์ฟเวอร์ ดังภาพ

GET /tutorials/basic/helloworld.html HTTP/1.0

From: soup@jarticles.com

User-Agent: Mozilla/4.72

[blank line]

ภาพที่ 2.35 ภาพเมสเสจที่ไคลเอนท์ส่งไปร้องขอไฟล์ helloworld.html

หลังจากที่เซิร์ฟเวอร์ได้รับเมสเสจ เซิร์ฟเวอร์จะตอบสนองด้วยการส่งเมสเสจพร้อม Resource กลับมา

HTTP/1.0 200 OK

Date: Sunday, 23-July-00 04:01:12 GMT

Content-type: text/html

Content-length: 1556

[blank line]

<HTML><HEAD><TITLE>HTTTP Tutorial</TITLE></HEAD>

<BODY>This is HelloWorld tutorial, but please read it for fun :)

(more file contents)

...

...

...

</BODY>

<HTML>

ภาพที่ 2.36 ภาพเมสเสจที่เซิร์ฟเวอร์ตอบสนองกลับมาพร้อมกับไฟล์ helloworld.html

ท้ายสุด เซิร์ฟเวอร์จะทำการปิดการเชื่อมต่อของไคลเอนท์

2.4.5 เมตธอด GET

ในการหาข้อมูลบางอย่างจาก Search Engine อาจจะสังเกตเห็นว่าหลังจากที่ใส่คำที่ต้องการค้นหาลงไป แล้วคลิกปุ่มเพื่อทำการค้นหา URL ที่ปรากฏบนเว็บเบราเซอร์จะเปลี่ยนไป เช่น

http://www.google.com/search.cgi?keyword=jarticles

ภาพที่ 2.37 ภาพตัวอย่าง URL เมื่อมีการค้นหาคำว่า jarticles ที่ www.google.com

ถ้าลองดูส่วนที่อยู่หลังจากส่วนของโฮสท์เนม โดยในที่นี้โฮสท์เนมคือ www.google.com จะเห็นว่า URI ที่ถูกอ้างถึงในการร้องขอ ซึ่งในที่นี้คือ /search.cgi?keyword=jarticles ไม่ได้เป็นไฟล์ html เหมือนทั่ว ๆ ไป แต่กลับเป็นชื่อโปรแกรม search.cgi ตามด้วยเครื่องหมายคำถามและคำที่ใช้ค้นหา

จากที่กล่าวมาแล้วในตอนต้นว่า Resource ก็คือตัวที่ใช้บ่งชี้ถึงข้อมูลที่อยู่ที่เซิร์ฟเวอร์ โดย Resource อาจเป็นไฟล์หรือคำสั่งที่ใช้ส่งไปเรียกโปรแกรมที่เซิร์ฟเวอร์ก็ได้ ในกรณีของตัวอย่าง Search Engine ข้างต้น Resource ก็คือโปรแกรมที่ชื่อ search.cgi นั่นเอง

โดยทั่วไปการตอบสนองที่ได้กลับมาจากเซิร์ฟเวอร์อาจอยู่ในรูปของไฟล์ ถ้า Resource ที่เราทวงถามเป็นไฟล์ ยกตัวอย่างเช่น http://www.jarticles.com/tutorial/http.html หรืออาจจะเป็นผลลัพธ์ที่ได้จากการประมวลผลของโปรแกรมที่รับคำสั่งเข้าไป เช่น ถ้าส่งคำสั่งไปที่ Search Engine ผลที่เราจะได้กลับมาก็คือลิงค์ต่าง ๆ ที่เกี่ยวข้องกับคำที่ใช้ค้นหานั่นเอง

โดยทั่วไปเว็บเบราเซอร์จะใช้ เมตธอด GET ในการส่งคำร้องขอไปที่เซิร์ฟเวอร์ในกรณีที่ Resource ที่ต้องการเป็นไฟล์ อย่างไรก็ตามเมตธอด GET ยังสามารถใช้ในการส่งคำสั่งสั้น ๆ ไปยังโปรแกรมที่ทำงานอยู่ที่เซิร์ฟเวอร์ได้อีกด้วย ยกตัวอย่างเช่น

ถ้าต้องการส่งข้อมูลชื่อ keyword โดยมีค่าเท่ากับ jarticles ไปที่เซิร์ฟเวอร์ชื่อ www.google.com โดยเจาะจงไปที่ Resource ซึ่งเป็นโปรแกรม CGI ชื่อ search.cgi วิธีการทำก็คือ

- กำหนด Resource ลงไปในส่วน URI หลังจากส่วนของโฮสท์เนม จะได้ http://www.google.com/search.cgi
- ใส่เครื่องหมายคำถาม จะได้ http://www.google.com/search.cgi? ซึ่งเป็นการบอกเซิร์ฟเวอร์ว่าส่วนถัดไปจะเป็นส่วนของข้อมูล
- ทำการเข้ารหัสชื่อและค่าของตัวแปรต่าง ๆ ที่เรียกว่า “URL-encoding” โดยในกรณีนี้ ตัวแปรก็คือ keyword จะได้ http://www.google.com/search.cgi?keyword=jarticles

ถ้าพิมพ์ URL ข้างบนเข้าไปในเว็บเบราเซอร์ แล้วกดปุ่ม Enter เว็บเบราเซอร์ก็จะทำการตีความ URL ดังกล่าว ซึ่งผลที่ได้คือการใช้เมตธอด GET ส่งคำร้องขอไปที่เซิร์ฟเวอร์ www.google.com โดยจะแนบคำว่า "jarticles" ไป ซึ่งบรรทัดเริ่มต้นของคำร้องขอจะเป็นดังภาพ

```
GET /search.cgi?keyword=jarticles HTTP/1.0
From: soup@jarticles.com
User-Agent: Mozilla/4.72
[blank line]
```

ภาพที่ 2.38 ภาพเมสเสจที่ไคลเอนท์ร้องขอให้เซิร์ฟเวอร์ประมวลผลไฟล์ search.cgi

หลังจากนั้นเราจะได้ลิงค์ต่าง ๆ ที่เกี่ยวกับ jarticles กลับมา

วิธีการส่งคำสั่งไปกับ URL เหมาะสำหรับคำสั่งที่ไม่ยาวมากนัก โดยทั่วไปความยาวของ URL มักจะจำกัดอยู่ที่ 80 ตัวอักษร ตัวอักษรที่ถัดจากนั้นจะถูกตัดออกไปโดยอัตโนมัติ แต่จริง ๆ แล้วขึ้นอยู่กับเซิร์ฟเวอร์ว่าถูกกำหนดให้ทำงานอย่างไร

2.4.6 **การเข้ารหัส** URL (URL-Encoding)

ในการส่งข้อมูลไปที่เซิร์ฟเวอร์ด้วยวิธีเมตธอด GET หรือเมตธอด POST ซึ่งจะพูดถึงในเรื่องต่อไปนั้น ชื่อและค่าของตัวแปรต่าง ๆ ซึ่งอยู่หลังจากเครื่องหมายคำถามในกรณีของเมตธอด GET หรืออยู่ในส่วนของข้อความในกรณีของเมตธอด POST จะต้องถูกเข้ารหัสด้วยวิธีการที่เรียกว่า “URL-Encoding” เสียก่อน

โดยทั่วไป URL เองจะมีตัวอักษรบางตัวที่ใช้สำหรับความหมายพิเศษ เช่น :, /, ~, &, ? ซึ่งในบางครั้งชื่อและค่าของตัวแปรต่าง ๆ ที่ถูกส่งไปที่เซิร์ฟเวอร์อาจจะมีตัวอักษรเหล่านี้ปะปนอยู่ด้วย เพื่อป้องกันความสับสนในการตีความของเซิร์ฟเวอร์ ตัวอักษรต่าง ๆ ที่ใช้ใน URL จะต้องถูกเข้ารหัสเสียก่อน โดยมีหลักการดังต่อไปนี้

- เปลี่ยนตัวอักษรที่ใช้ใน URL ซึ่งอยู่ในชื่อและค่าของตัวแปรให้กลายเป็น "%xx" โดย "xx" คือค่ารหัส ASCII ของตัวอักษรนั้น ๆ ในแบบเลขฐานสิบหก ซึ่งตัวอักษรเหล่านี้รวมไปถึง =, &, %, +, และตัวอักษรต่าง ๆ ที่ไม่สามารถพิมพ์ได้ หรือแม้กระทั่งตัวอักษรที่ต้องการจะเข้ารหัสเอง
- เปลี่ยนช่องว่างทั้งหมดให้กลายเป็นเครื่องหมายบวก
- จับคู่ชื่อและค่าของตัวแปรต่าง ๆ ด้วยเครื่องหมายเท่ากับ
- ถ้ามีชื่อและค่าของตัวแปรมากกว่าหนึ่งตัว ให้นำชื่อและค่าของตัวแปรแต่ละคู่มาทำการเชื่อมกันด้วยเครื่องหมาย &
- นำชื่อและค่าของตัวแปรที่เชื่อมติดกันทั้งหมดไปใส่ที่ URL โดยมีเครื่องหมาย ? อยู่ข้างหน้าในกรณีของ GET หรือนำไปใส่ที่ส่วนข้อความในกรณีของเมตธอด POST

หลังจากทำการเข้ารหัส จะได้ออกมาเป็น keyword=jarticles&name=Soup+%26+friends

2.4.7 **เมตธอด** POST

บางครั้งเราอาจจะไม่ต้องการส่งข้อมูลไปยังเซิร์ฟเวอร์ด้วยการแนบข้อมูลเหล่านั้นไปกับ URL ด้วยเหตุผลที่ว่าข้อมูลเหล่านั้นอาจเกี่ยวข้องกับข้อมูลส่วนตัว ยกตัวอย่างเช่น ล็อกอินและพาสเวิร์ดที่ใช้เป็นตัวผ่านเข้าไปในเว็บไซท์ ซึ่งในกรณีที่เราใช้เมตธอด GET หลังจากที่ทำการล็อกอินไปแล้ว URL อาจออกมาเป็น http://www.myserver.com/login.cgi?login=me&password=youandme และถ้าบังเอิญว่าระหว่างที่ล็อกอิน มีคนอื่นนั่งอยู่ข้าง ๆ คนคนนั้นก็อาจจะนำเอาข้อมูลไปใช้อย่างผิดได้

อีกกรณีหนึ่งคือ ข้อมูลที่ติดไปกับส่วนของ URI ทำให้ URL มีความยาวมากกว่าความยาวของ URL ที่เซิร์ฟเวอร์กำหนดไว้ ผลกระทบที่อาจเกิดขึ้นคือข้อมูลบางส่วนอาจหายไปเนื่องจากการตัดทอนโดยเซิร์ฟเวอร์

วิธีการที่สามารถใช้เพื่อปิดบังข้อมูลแบบง่าย ๆ หรือเพื่อส่งข้อมูลที่มีความยาวมาก ๆ ก็คือการใช้เมตธอด POST

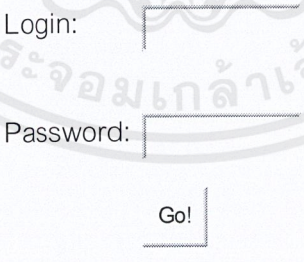
เมตธอด POST ต่างกับเมตธอด GET ตรงที่ว่าเมตธอด POST จะไม่ทำการแนบข้อมูลไปกับ URL แต่จะใส่ข้อมูลเข้าไปในส่วนของข้อความในคำร้องขอแทน ซึ่งในกรณีนี้ URI จะมีปรากฏเพียงชื่อของโปรแกรมที่ต้องการส่งข้อมูลไปให้เท่านั้น

ในการส่งข้อมูลไปในส่วนของข้อความ เพื่อให้ง่ายต่อการตีความโดยเซิร์ฟเวอร์ เมตธอด POST จึงมีการบังคับให้ใส่เฮดเดอร์พิเศษ เพื่อบรรยายรายละเอียดต่าง ๆ ของข้อมูลที่อยู่ในส่วนข้อความนั้นอีกด้วย ซึ่งโดยทั่วไปเฮดเดอร์ที่มักจะถูกเพิ่มเข้าไปก็คือ Content-Type และ Content-Length

การใช้เมตธอด POST ที่พบเห็นทั่วไปคือการใช้ในการส่งข้อมูลที่มาจากฟอร์มในภาษา HTML เช่น

```
<FORM METHOD="POST" NAME="loginform" ACTION="http://www.jarticles.com/cgi-bin/login.cgi">
<INPUT TYPE="text" NAME="username" SIZE="20" MAXLENGTH="20" VALUE="">
<INPUT TYPE="PASSWORD" NAME="password" SIZE="20" MAXLENGTH="20" VALUE="">
<INPUT TYPE="SUBMIT" VALUE="Go!">
</FORM>
```

ภาพที่ 2.39 ภาพตัวอย่างการใช้เมตธอด POST ร่วมกับฟอร์มในภาษา HTML



ภาพที่ 2.40 ภาพฟอร์มที่ปรากฏจากตัวอย่างในภาพที่ 2.39

หลังจากที่คลิกปุ่ม Go! แล้ว เว็บเบราเซอร์จะทำการส่งเมสเสจไปที่เซิร์ฟเวอร์

```
POST /cgi-bin/login.cgi HTTP/1.0
From: soup@jarticles.com
User-Agent: Mozilla/4.72
Content-Type: application/x-www-form-urlencoded
Content-Length: 32
userName=bababa&password=wow
```

ภาพที่ 2.41 ภาพเมสเสจที่จะถูกส่งไปที่เซิร์ฟเวอร์หลังจากคลิกปุ่ม Go!

จากภาพ จะสังเกตได้ว่า ในกรณีของการส่งข้อมูลจากฟอร์มในภาษา HTML ไคลเอนท์จะทำการกำหนดค่า Content-Type เป็น application/x-www-form-urlencoded และ Content-Length จะมีค่าเท่ากับความยาวของชื่อและค่าของอีลิเมนต์ต่าง ๆ ที่อยู่ในฟอร์มรวมกัน ซึ่งจากตัวอย่างก็คืออีลิเมนต์ชื่อ username และ password

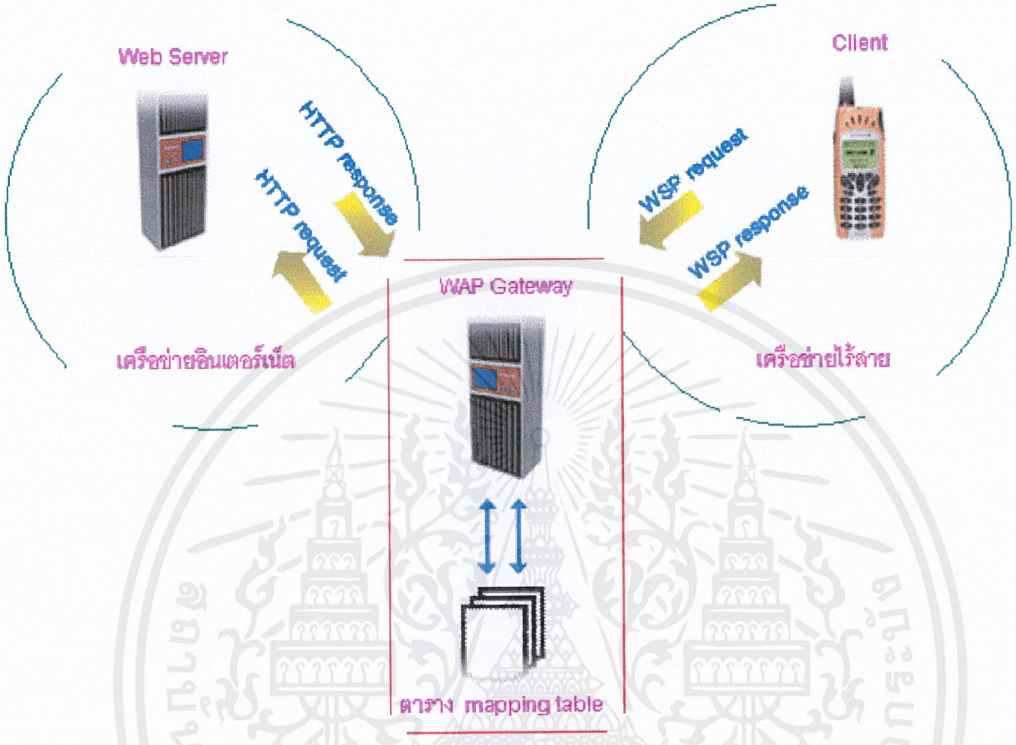
นอกจากนี้ เมตธอด POST ยังสามารถใช้กับการส่งข้อมูลแบบอื่น ๆ ได้อีก โดยขึ้นอยู่กับวิธีที่เว็บเบราเซอร์และ เว็บเซิร์ฟเวอร์ได้ทำการตกลงกันไว้ ยกตัวอย่างเช่น ในกรณีของการอัพโหลดไฟล์ Content-Type ก็จะถูกกำหนดเป็น Multipart/form-date แทน

```
<FORM ACTION="/upload/uploadServlet" EncType="multipart/form-data"
METHOD="POST">
Which file do you want to upload?
<INPUT TYPE="file" NAME="yourfile"><br>
<INPUT TYPE="submit">
</FORM>
```

ภาพที่ 2.42 ภาพตัวอย่างการใช้เมตธอด POST สำหรับการอัพโหลดไฟล์

2.5 รายละเอียดของ WAP Gateway

2.5.1 การทำงานของ WAP Gateway



ภาพที่ 2.43 ภาพแสดงการทำงานของ WAP Gateway

- ผู้ใช้โทรศัพท์มือถือ ซึ่งถือว่าเป็นไคลเอนท์ ส่ง URL ของเอกสารที่ต้องการไปยัง WAP Gateway โดยจะส่งเป็นคำร้องขอตามรูปแบบของโพรโตคอล WSP

- WAP Gateway ถอดรหัสคำร้องขอที่อยู่ในรูปแบบไบนารี เพื่อแปลงให้อยู่ในรูปของคำร้องขอแบบ HTTP โดยอาจอาศัยตาราง mapping table ที่มีอยู่ใน WAP Gateway เป็นตัวช่วย การถอดรหัสจะขึ้นอยู่กับเทคนิคของผู้ผลิตและผู้พัฒนาระบบ WAP Gateway แต่ละราย

- WAP Gateway สร้างการเชื่อมต่อไปยังเว็บเซิร์ฟเวอร์ แล้วส่งคำร้องขอตามไปในรูปแบบโพรโตคอล HTTP

- เว็บเซิร์ฟเวอร์จะประมวลผลคำร้องขอนั้น และตรวจสอบดูว่าเอกสารที่ร้องขอเป็นลักษณะโคด WMLธรรมดาหรือไม่ แต่ถ้าหากเอกสารนั้นเรียกการทำงานของสคริปต์ต่าง ๆ เช่น CGI, ASP ก็จะต้องประมวลผลสคริปต์นั้นก่อน เพื่อให้กลายเป็นเอกสาร WML ธรรมดา ซึ่งประกอบไปด้วยแท็กและข้อความ

- เว็บเซิร์ฟเวอร์ส่งเอกสารกลับมายัง WAP Gateway โดยส่งเป็นคำตอบกลับในรูปแบบโพรโตคอล HTTP

- WAP Gateway ก็จะเข้ารหัสเอกสารไปเป็นรูปแบบไบนารี โดยอาจจะอาศัยตาราง mapping table เป็นตัวช่วยอีกเช่นกัน

- WAP Gateway สร้างการเชื่อมต่อไปยังไคลเอนท์ แล้วส่งข้อมูลไบนารีนั้นเป็นคำตอบกลับไปในรูปแบบโพรโตคอล WSP ไปยังไคลเอนต์ต่อไป

2.5.2 **หน้าที่หลักของ** WAP Gateway

คุณสมบัติของโปรแกรม WAP Gateway แต่ละตัวไม่เหมือนกันเสียทีเดียว เนื่องจากผู้ผลิตแต่ละรายอาจเพิ่มหน้าที่บางอย่างเสริมเข้าไป แต่โดยทั่วไป WAP Gateway มีหน้าที่ดังนี้

- รองรับโพรโตคอล WAP และชุดโพรโตคอลในอินเทอร์เน็ท
- ทำการสลับโพรโตคอล (Protocol Conversion)
- เข้ารหัสเอกสาร WML ให้เป็นข้อมูลรูปแบบไบนารี
- คอมไพล์โคด WMLScript
- เป็น proxy server เพื่อให้บริการข้อมูลที่ถูกเรียกใช้บ่อย ๆ
- ดูแลจัดการด้านการรักษาความปลอดภัยของข้อมูล
- เปลี่ยนเอกสาร HTML ที่ได้รับจากเว็บเซิร์ฟเวอร์ให้เป็นเอกสาร WML

2.5.3 **หลักการของ** Protocol Conversion

โดยปกติ ในการส่งข้อมูลไปมาระหว่างเว็บเบราเซอร์และเว็บเซิร์ฟเวอร์ ซึ่งอาศัยโพรโตคอล HTTP นั้น นอกจากตัวเอกสาร HTML ที่ประกอบไปด้วยแท็กคำสั่งต่าง ๆ ที่รู้จักกันดีแล้ว จะต้องมีข้อมูลอีกส่วนหนึ่งอยู่ที่ช่วงต้นของเอกสารด้วยเสมอ เรียกว่า เฮดเดอร์ ซึ่งเป็นตัวบ่งบอกรายละเอียดเกี่ยวกับเอกสารนั้น ๆ เช่น ประเภทของข้อมูลในเอกสาร, ความยาวหรือขนาดของข้อมูลหรือเอกสาร เป็นต้น

ในกรณีที่เอกสารถูกส่งไปยังเครือข่ายแบบไร้สาย WAP Gateway ก็ต้องเข้ารหัสเฮดเดอร์ให้เป็นแบบไบนารีด้วย เหตุผลที่ต้องเข้ารหัสก็คือเพื่อลดข้อจำกัดทางด้าน bandwidth และ latency ของเครือข่ายแบบไร้สาย

เฮดเดอร์เดิมอยู่ในระบบโพรโตคอล HTTP การเข้ารหัสจึงเป็นการแปลงเฮดเดอร์นั้นให้สามารถส่งต่อไปในระบบโพรโตคอล WSP ฉะนั้นเฮดเดอร์เดิมที่เรียกว่า HTTP header ก็จะกลายเป็น WSP header

ต่อไปนี้จะเป็นการเปรียบเทียบให้เห็นว่า เมื่อเฮดเดอร์เหล่านี้ผ่านการเข้ารหัส/ถอดรหัสที่ WAP Gateway แล้ว จะเปลี่ยนเป็นอย่างไร

Accept-Language: en,sv

เมื่อเทียบกับรหัสแบบไบนารีที่ใช้ส่งจากโทรศัพท์ผ่านเครือข่ายไร้สายไปยัง WAP Gateway จะเป็นดังนี้

0x83 0x99 0x83 0xF0

ภาพที่ 2.44 ภาพแสดงการบ่งบอกภาษาที่ไคลเอนท์ใช้ใน HTTP header ที่ส่งจากไคลเอนท์ไปยังเว็บเซิร์ฟเวอร์

Date : Sat , 13 Aug 2000 02:05:47 GMT

เมื่อเข้ารหัสเป็น WSP header แล้ว จะได้ผลลัพธ์เป็นข้อมูลแบบไบนารีดังนี้

0x92 0x04 0x35 0x3f 0x45 0x11

ภาพที่ 2.45 ภาพแสดงข้อความบ่งบอกเวลาที่ส่งข้อมูลใน HTTP header ที่ส่งจากเว็บเซิร์ฟเวอร์ไปยังไคลเอนท์

WSP header ของคำร้องขอไฟล์ index.wml ใน http://wecocsie.fju.edu.tw ที่ส่งมาจาก Nokia WAP Toolkit :ซึ่งเป็นโปรแกรมที่ใช้พัฒนา WAP Application ของโนเกีย มีลักษณะเป็นข้อมูลไบนารีดังนี้

---

01 40 25 68 74 74 70 3a 2f 2f 77 65 63 6f 2e 63 73 69 95 2e 66 6a
75 2e 65 64 75 2e 74 77 2f 69 6e 64 65 78 2e 77 6d 6c 81 ea 81 84
80 94 80 95 80 al 80 9d a9 4e 6f 6b 69 61 2d 57 41 50 2d 54 6f 6f
6c 6b 69 74 2f 31 2e 33 62 65 74 61 00

ภาพที่ 2.46 ภาพข้อมูลไบนารีของ WSP header ของการร้องขอไฟล์ index.wml จาก http://wecocsie.fju.edu.tw

เมื่อ WAP Gateway ถอดรหัส WSP header ให้เป็น HTTP header แล้ว จะมีลักษณะดังนี้

TID=1

Type=GET

URL=http://weco.csie.fju.edu.tw/index.wml

Accept-Charset:Somali , Amharic

Accept : application/vnd.wap.wmlc ,

Application/vnd.wap.wmlscriptc,image/vnd.wap.wbmp,image/gif

User-Agent:Nokia-WAP-Toolkit/1.3beta

ภาพที่ 2.47 ภาพแสดงการถอดรหัส WSP header เป็น HTTP header โดย WAP Gateway

2.5.4 การเข้ารหัสเอกสาร WML ให้เป็นข้อมูลไบนารี

เอกสาร WML ที่ส่งไปมาระหว่างเบราเซอร์และเว็บเซิร์ฟเวอร์ ประกอบไปด้วยเฮดเดอร์และเนื้อหาเอกสารจริง ๆ ดังนั้นในหัวข้อนี้จะกล่าวถึงการเข้ารหัสเนื้อหาเอกสาร WML ซึ่งประกอบไปด้วยแท็กต่าง ๆ และข้อความ

เอกสาร WML เหล่านี้จะส่งมาจากเว็บเซิร์ฟเวอร์หรือ Application Server ผ่านเครือข่ายอินเทอร์เน็ท โดยมีรูปแบบเป็นข้อความที่สามารถอ่านได้ แต่เมื่อ WAP Gateway ได้รับเอกสารแล้ว ก็จะตรวจสอบความถูกต้องตามหลักไวยากรณ์ของเนื้อหาว่าเป็นไปตามกฎของภาษา XML หรือไม่ ก่อนที่จะเข้ารหัสให้เป็นข้อมูลแบบไบนารี เพื่อส่งต่อไปยังไคลเอนท์ เพราะว่า WML เป็นรูปแบบหนึ่งของภาษา XML และต้องยึดถือกฎเกณฑ์ของ XML ด้วย ถ้าหาก WAP Gateway พบว่าเนื้อความในเอกสาร WML ไม่ถูกต้องตามหลักไวยากรณ์ของ WML สำหรับ WAP Gateway บางตัวก็จะส่งข้อความที่มีเนื้อหาเกี่ยวกับความผิดพลาดมาแสดงที่ไคลเอนท์ แต่บางตัวก็อาจแสดงข้อความที่ไม่ถูกต้องออกมา โดยไม่แจ้งถึงความผิดพลาดที่เกิดขึ้น

หากมีคำถามว่าทำไมต้องตรวจสอบไวยากรณ์ ก็ตอบได้ว่าเพราะภาษา WML อาศัยกฎเกณฑ์ของ XML จึงต้องใช้ภาษาให้ตรงตามรูปแบบ มิฉะนั้นก็จะไม่สามารถแสดงผลที่เบราเซอร์ของไคลเอนท์ได้ถูกต้อง และถ้าหากถามต่ออีกว่า ทำไมไม่ผลักภาระการตรวจสอบนี้ให้กับเครื่องไคลเอนท์ ก็ตอบได้เลยว่าหลักการของ WAP จะพยายามไม่ผลักภาระการประมวลผลใด ๆ ให้กับเครื่องไคลเอนท์หรือโทรศัพท์มือถือ เนื่องจากมีข้อจำกัดต่าง ๆ มากมาย เช่น มีขีดความสามารถในการประมวลผลที่จำกัด,

มีพลังงานจำกัด เพราะใช้แบตเตอรี่ เป็นต้น ดังนั้นหลักการนี้ก็สามารถนำไปใช้การสร้าง Wap Application ได้

เอกสาร WML ที่ส่งมาจาก WAP Application Server หรือเว็บเซิร์ฟเวอร์มีเนื้อหาดังนี้

```
<wml>
<card id="abc" ordered="true">
<p>
        <do type="accept">
                <go href="http://xyz.org/s"/>
        </do>
                X:$(X) <br/>
                Y:$(&#x59;)<br/>
                Enter name : <input type = "text" name="N"/>
</p>
</card>
</wml>
```

ภาพที่ 2.48 ภาพตัวอย่างเอกสาร WML

เมื่อผ่านการเข้ารหัสที่ Wap Gateway แล้ว จะได้ข้อมูลที่มีลักษณะเป็นไบนารี ดังนี้

```
02 08 6A 04 'X' 00 'Y' 00 7F E7 55 03 'a' 'b' 'c' 00
80 01 60 E8 38 01 AB 4B 03 'x' 'y' 'z' 00 88 03
's' 01 01 01 03 ' ' 'x' ' : ' ' ' 00 82 00 26 03 ' ' 'Y'
':' ' ' 00 82 02 26 03 ' ' 'e' 'n' 't' 'e' 'r' ' ' 'n'
'a' 'm' 'e' ':' ' ' 00 AF 48 21 03 'N' 00 01 01 01 01
```

ภาพที่ 2.49 ภาพเอกสาร WML หลังจากที่ถูกเข้ารหัส

2.5.5 **คอมไพล์โคด** WMLScript

โคด WMLScript ที่ส่งมาจากทางเครื่องเว็บเซิร์ฟเวอร์ จำเป็นต้องให้ WAP Gateway คอมไพล์เหมือนกับโปรแกรมที่เขียนด้วยภาษาอื่น ๆ เช่นกัน โดยที่ WAP Gateway ต้องตรวจดูไวยากรณ์ของภาษาก่อน โดยผลของการคอมไพล์จะได้ข้อมูลเป็นไบท์โคด ซึ่งเป็นข้อมูลไบนารีแบบหนึ่ง เมื่อไบท์โคดเหล่านี้ถูกส่งต่อไปยังโทรศัพท์มือถือ ก็จะต้องผ่านกระบวนการแปลและประมวลผลเพื่อให้ทำงานตามโคดที่เขียนเอาไว้

การเข้ารหัสเอกสาร WML กับการคอมไพล์โคดภาษา WMLScript เป็นกระบวนการที่ไม่เหมือนกันเสียทีเดียว สามารถอธิบายได้ดังนี้

- การเข้ารหัสเอกสาร WML อาศัยการเปรียบเทียบแท็ก WML ต่าง ๆ กับรหัสไบนารีที่กำหนดไว้แล้วในตาราง mapping table ซึ่งเรียกว่า WSP specification ดังที่เคยยกตัวอย่างมาแล้ว เช่น คำว่า “Accept-Language” ใน HTTP header แทนด้วยรหัส 0x83 คำว่า “Date” ก็แทนด้วย 0x92 เป็นต้น ส่วนแท็กอื่น ๆ ก็จะมีรหัสไบนารีเฉพาะของแต่ละแท็ก
- การคอมไพล์โคด WMLScript เป็นกระบวนการอีกลักษณะหนึ่ง ซึ่งคล้ายกับการคอมไพล์โปรแกรม คือต้องมีการตรวจสอบไวยากรณ์ของภาษาด้วย ไม่ได้อาศัยการเปรียบเทียบเหมือนกับการเข้ารหัสภาษา WML
- เนื้อหา WML ที่ผ่านการเข้ารหัสเป็นข้อมูลแบบไบนารีมาจาก WAP Gateway แล้ว ก่อนที่จะแสดงผลจริง ๆ โดยเบราเซอร์ที่ไคลเอนท์ได้ ก็จะต้องมีการถอดรหัสกลับคืน โดยอาศัยการเปรียบเทียบในทำนองเดียวกับการเข้ารหัส
- ส่วนในกรณีของ WMLScript นั้น เมื่อไบท์โคดมาถึงไคลเอนท์ ก็จะถูกแปลกลับคืนโดยอาศัยอินเทอร์พรีเตอร์ ซึ่งวิธีนี้ต้องไปยุ่งเกี่ยวกับสถานะของตัวแปร

ก่อนจบหัวข้อนี้ ขอย้อนกลับไปกล่าวถึงหน้าที่ของ WAP Gateway ทางด้านการดูแลจัดการรักษาความปลอดภัยของข้อมูลอีกครั้งหนึ่ง คือ หน้าที่นี้จะอาศัยโพรโตคอลของการรักษาความปลอดภัยในอินเทอร์เน็ทที่ชื่อว่า SSL (Secure Socket Layer) มาดัดแปลงให้เหมาะกับเครือข่ายแบบไร้สาย

อีกหน้าที่หนึ่งที่ขอย้อนกล่าวถึง คือ การเปลี่ยนเอกสาร HTML ที่ส่งมาจากเว็บเซิร์ฟเวอร์ ให้เป็นเอกสาร WML ซึ่งรวมถึงการละเว้นการแสดงข้อมูลทางมัลติมีเดีย เช่น Shockwave, Flash หรือภาพ ความจริงแล้ว WAP Gateway ควรมีความสามารถในการทำหน้าที่นี้ เมื่อไคลเอนท์เรียกดูเอกสารที่เขียนด้วยภาษา HTML นั้น ดังนั้นผู้พัฒนา WAP Application จึงไม่ควรหวังพึ่งพาหน้าที่การทำงานนี้ของ WAP Gateway ซึ่งมีเพียงไม่กี่ตัวที่สนับสนุน แต่ควรใช้ภาษา WML เป็นหลักในการพัฒนาจะดีที่สุด

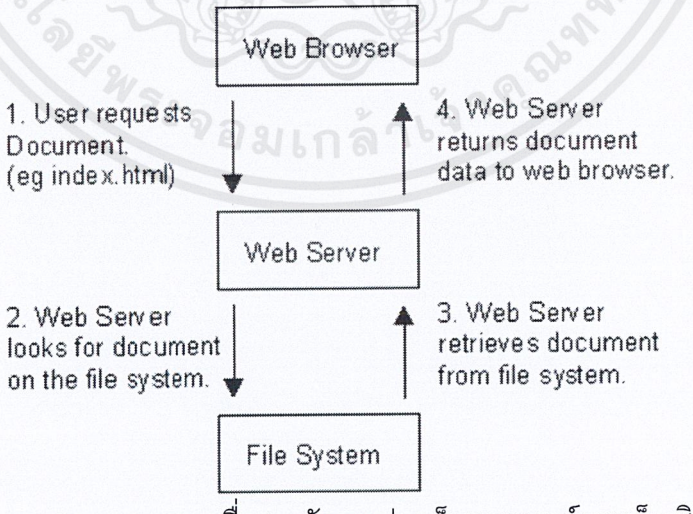
2.6 เว็บเซิร์ฟเวอร์

2.6.1 การทำงาน

เมื่อพูดถึงการทำงานของเว็บเซิร์ฟเวอร์ การอธิบายด้วยไดอะแกรมของแพ็คเก็ตที่ถูกส่งไปมาในเครือข่ายนั้น ยังถือว่าไม่เพียงพอ

เมื่อหลายปีก่อน ตอนที่เว็บเซิร์ฟเวอร์ต้นแบบถูกสร้างขึ้นมาครั้งแรกนั้น เว็บเซิร์ฟเวอร์ให้บริการได้เพียงเอกสาร HTML และรูปภาพ แต่ในปัจจุบันนี้เว็บเซิร์ฟเวอร์ถูกใช้ประโยชน์มากขึ้นกว่าเดิมมาก

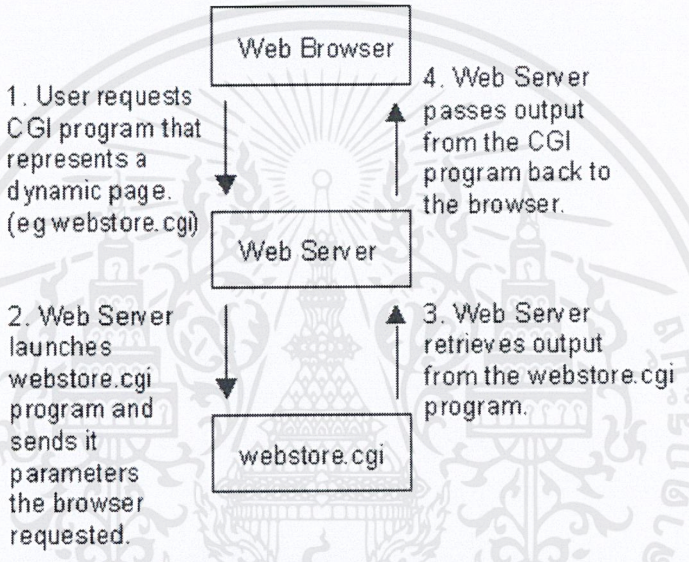
ในตอนแรก เรามองว่าเว็บเซิร์ฟเวอร์เป็น black box และมีคำถามต่าง ๆ เกิดขึ้นมากมาย เช่น เว็บเซิร์ฟเวอร์ทำงานอย่างไร, เว็บเซิร์ฟเวอร์สามารถทำงานด้านใดได้บ้าง มีการตั้งสมมติฐานว่าผู้ใช้อินเทอร์เน็ทส่วนใหญ่เชื่อว่าความสำเร็จหรือความล้มเหลวในการทำงานของเว็บเซิร์ฟเวอร์ เกิดจากข้อมูลหรือหน้าที่การทำงานของเว็บเซิร์ฟเวอร์ มากกว่าที่จะเป็นเพราะการใช้งานเว็บเซิร์ฟเวอร์หนักเกินไป อย่างไรก็ตาม การเข้าใจเกี่ยวกับความสามารถและขีดจำกัดในการทำงานของเว็บเซิร์ฟเวอร์ก็เป็นสิ่งที่สำคัญอย่างหนึ่งด้วยเช่นกัน ดังนั้นถ้าจะถามว่าเว็บเซิร์ฟเวอร์มีหน้าที่อะไร คำตอบก็คือว่าเว็บเซิร์ฟเวอร์มีหน้าที่พื้นฐานในการให้บริการข้อมูลแบบ static แก่เว็บเบราเซอร์ นั่นคือเว็บเซิร์ฟเวอร์ต้องได้รับการร้องขอเว็บเพจ เช่น http://www.webcompare.com/index.html แล้วจึงทำการแปลง URL (Uniform Resource Locator) ที่ได้รับนั้นให้เป็นตำแหน่งของไฟล์ที่อยู่บนโฮสท์เซิร์ฟเวอร์ ดังนั้นในกรณีนี้ ไฟล์ index.html ก็จะอยู่ที่ใดที่หนึ่งบนระบบไฟล์ของโฮสท์ หลังจากนั้นเว็บเซิร์ฟเวอร์จะโหลดไฟล์จากดิสก์และส่งผ่านเครือข่ายกลับไปยังเว็บเบราเซอร์ การแลกเปลี่ยนข้อมูลใช้เว็บเบราเซอร์และเว็บเซิร์ฟเวอร์เป็นสื่อในการติดต่อกันผ่านโพรโตคอล HTTP



ภาพที่ 2.50 ภาพแสดง workflow ของการสื่อสารกันระหว่างเว็บเบราเซอร์และเว็บเซิร์ฟเวอร์

เนื่องจากการทำงานอย่างง่าย ๆ ที่อนุญาตให้ผู้ใช้สามารถรับข้อมูลแบบ static เช่นเอกสาร HTML และไฟล์ภาพได้นั้น เป็นเพียงหลักการเบื้องต้นของ World Wide Web เท่านั้น ข้อดีอีกอย่างหนึ่งก็คือการนำไปสู่การแลกเปลี่ยนข้อมูลที่มีความซับซ้อนมากขึ้นได้

ข้อมูลที่มีความซับซ้อนนั้นก็คือข้อมูลที่มีลักษณะแบบ dynamic เช่น เว็บเพจที่สร้างขึ้นตามลักษณะของข้อมูลที่ผู้ใช้ป้อนเข้ามา ซึ่งจะถูกส่งกลับไปในการตอบสนอง ไม่ว่าจะโดยทางตรงหรือทางอ้อม วิธีการที่เก่าแก่และนิยมใช้กันมากที่สุดสำหรับการทำงานกับข้อมูลแบบ dynamic ก็คือ CGI (Common Gateway Interface) ซึ่งเป็นการทำงานในลักษณะที่เว็บเซิร์ฟเวอร์ประมวลผลโปรแกรมต่าง ๆ และส่งผลลัพธ์ที่ได้กลับไปยังเว็บเบราเซอร์ของผู้ใช้



ภาพที่ 2.51 ภาพแสดงสิ่งที่เกิดขึ้นเมื่อเว็บเบราเซอร์ทำการร้องขอเว็บเพจแบบ Dynamic

ข้อดีอีกอย่างหนึ่งคือความสามารถในการทำงานกับธุรกิจแบบ E-Commerce ซึ่งเป็นการทำงานบน HTTPS (Hyper Text Transmission Protocol, Secure) โดยที่โพรโตคอลนี้ยอมให้มีการสื่อสารที่ต้องการความปลอดภัยระหว่างเว็บเบราเซอร์และเว็บเซิร์ฟเวอร์ นั่นก็หมายความว่าเป็นการปลอดภัยสำหรับผู้ใช้และเว็บเซิร์ฟเวอร์ ในการส่งข้อมูลที่สำคัญระหว่างกันข้ามเครือข่ายที่อาจไม่ปลอดภัย อย่างไรก็ตาม เมื่อข้อมูลมาถึงปลายทางแล้ว การทำอะไรกับข้อมูลนั้นก็นับเป็นสิ่งสำคัญที่ควรพิจารณาร่วมด้วย

บ่อยครั้งที่ความซับซ้อนของเว็บเซิร์ฟเวอร์ นำไปสู่การตัดสินใจในการออกแบบโครงสร้างพื้นฐาน (Infrastructure) ของการทำ Web-hosting การเลือกใช้เทคโนโลยีต่าง ๆ ในการสร้างข้อมูลแบบ Dynamic เป็นสิ่งที่ง่าย เช่น Java, Javascript, Perl, C/C++ และ ASP แต่บางครั้งเทคโนโลยีเหล่านี้ก็กลายเป็นอุปสรรคในการทำงานได้ เนื่องจากเทคโนโลยีแต่ละอย่างต่างก็ทำงานบนระบบปฏิบัติการ

ที่ต่างกันไป พูดอีกอย่างก็คือ การจะอธิบายว่า เว็บเซิร์ฟเวอร์ทำงานอย่างไรนั้น มีประเด็นที่เกี่ยวข้องมากมาย ไม่ใช่แค่เพียงว่าเว็บเซิร์ฟเวอร์ให้บริการเอกสารได้อย่างไร

2.6.2 การให้บริการข้อมูล

การร้องขอข้อมูลจะต้องถูกระบุโดยเว็บเบราเซอร์ เพื่อดาวน์โหลดและแสดงผลข้อมูลนั้นอย่างถูกต้อง วิธีการเบื้องต้นสำหรับการแสดงผลข้อมูลก็คือ MIME type header

MIME (Multipurpose Internet Mail Extension) type บอกเว็บเบราเซอร์ว่าเอกสารที่ถูกส่งมานั้นเป็นเอกสารชนิดใด ชนิดของเอกสารไม่ได้ถูกจำกัดว่าต้องเป็นเพียงกราฟิกอย่างง่าย ๆ หรือ HTML

จริง ๆ แล้วเว็บเซิร์ฟเวอร์อย่าง Apache มี MIME type มี 370 ชนิดที่เป็นดีฟอลท์ในไฟล์สำหรับการกำหนดค่าต่าง ๆ ให้กับเว็บเซิร์ฟเวอร์ (Configuration file) ถึงแม้ว่าจะไม่สามารถแสดงผลข้อมูลได้ทุกชนิดตามที่ระบุไว้ในไฟล์ก็ตาม MIME type ระบุชนิดข้อมูลด้วยการใช้ type/subtype ซึ่งเกี่ยวข้องกับนามสกุลของไฟล์

text/xml	xml
video/mpeg	mpeg mpg mpe
video/quicktime	qt mov

ภาพที่ 2.52 ภาพตัวอย่าง MIME type ในไฟล์ของ Apache

จากภาพ จะเห็นได้ว่า ไฟล์ที่มีข้อมูลเป็น video ชนิด mpeg อาจจะมีนามสกุลเป็น mpeg, mpg, หรือ mpe ก็ได้

2.6.3 การรับการติดต่อจากไคลเอนท์

เว็บเซิร์ฟเวอร์ถูกออกแบบมาเพื่อจุดประสงค์ต่อไปนี้

- รองรับการเชื่อมต่อบนเครือข่ายจากเว็บเบราเซอร์
- อ่านข้อมูลจากดิสก์
- ประมวลผลโปรแกรม CGI
- ส่งข้อมูลกลับไปยังไคลเอนท์
- ทำงานให้เร็วที่สุดเท่าที่จะทำได้

แต่ในความเป็นจริง เว็บเซิร์ฟเวอร์ไม่สามารถบรรลุจุดประสงค์เหล่านี้ได้ทั้งหมด เว็บเซิร์ฟเวอร์ที่มีความสามารถไม่สูงนักจะมีการทำงานดังนี้

- ตอบรับการเชื่อมต่อ

- สร้างข้อมูลแบบ static หรือ dynamic แล้วส่งข้อมูลนั้นกลับไปยังเว็บเบราเซอร์
- ยกเลิกการเชื่อมต่อ
- รับการเชื่อมต่อ
- เริ่มกลับไปสร้างข้อมูลอีกครั้ง

การทำงานในลักษณะนี้เหมาะกับเว็บไซท์ที่ยังไม่ต้องการความสามารถในการทำงานสูงนัก แต่เว็บเซิร์ฟเวอร์จะต้องเจอกับปัญหาทันทีเมื่อไคลเอนท์ที่เข้ามาติดต่อมีจำนวนมากหรือเมื่อเว็บเซิร์ฟเวอร์ใช้เวลาในการสร้างข้อมูลชนิด dynamic นานมากเกินไป เช่น ถ้าโปรแกรม CGI หนึ่งใช้เวลาถึง 30 วินาทีในการสร้างข้อมูล การใช้เวลานานขนาดนี้ไม่ใช่การทำงานที่ดีอย่างแน่นอน และในระหว่างนี้เว็บเซิร์ฟเวอร์จะไม่สามารถให้บริการข้อมูลแก่อื่น ๆ ได้เลย

ดังนั้นแม้ว่าการทำงานในลักษณะนี้จะทำได้ แต่ก็ควรได้รับการออกแบบใหม่เพื่อให้สามารถให้บริการผู้ใช้ได้มากกว่า 1 คนพร้อม ๆ กัน ซึ่งเว็บเซิร์ฟเวอร์จะต้องใช้หลักการที่ต่างกันสองแบบในการให้บริการแก่ผู้ใช้หลาย ๆ คนพร้อมกัน นั่นก็คือ multi-threading และ multi-processing โดยที่เว็บเซิร์ฟเวอร์จะสนับสนุนการทำงานของ inetd module บนระบบ Unix ซึ่งเป็นรูปแบบการทำงานแบบ multi-processing, multi-threading หรือรูปแบบผสมระหว่าง multi-processing และ multi-threading อย่างใดอย่างหนึ่ง

2.6.4 การเลือกใช้ระบบปฏิบัติการ

เดิมนั้นเว็บเซิร์ฟเวอร์ใช้ inetd ในการสร้าง process ต่าง ๆ ที่สามารถจัดการกับการร้องขอจากเว็บเบราเซอร์ เป็นแอพพลิเคชันอย่างง่าย ๆ และไม่มีความสามารถในการรองรับการทำงานหลาย ๆ อย่างพร้อมกันได้ ดังนั้นจึงมีการตัดสินใจออกแบบแอพพลิเคชันขึ้นใหม่

วิธีการที่ง่ายที่สุดที่จะสร้างแอพพลิเคชันของเว็บเซิร์ฟเวอร์สำหรับระบบ Unix ที่สามารถจัดการกับการเชื่อมต่อเข้ามาของไคลเอนท์จำนวนมาก ๆ ก็คือการใช้ข้อดีของ inetd daemon กล่าวคือสามารถจัดการกับการสื่อสารทั้งหมดบนโพรโตคอล TCP/IP ได้

โดยปกติ process ของเว็บเซิร์ฟเวอร์มีหน้าที่ต้องคอยดักจับสัญญาณการเชื่อมต่อที่จะเข้ามาและรับการเชื่อมต่อนั้น จากนั้น process ต้องเลือกที่จะให้บริการแก่การเชื่อมต่อนั้นทันที หรือจะรอให้การเชื่อมต่อที่ได้รับบริการอยู่นั้นทำงานเสร็จเสียก่อน ซึ่ง inetd daemon สามารถทำหน้าที่นี้แทนได้ด้วยการดักจับสัญญาณทางพอร์ท โดยพอร์ทที่เป็นดีฟอลท์คือพอร์ท 80 สำหรับการร้องขอบนโพรโตคอล HTTP และประมวลผล process ของเว็บเซิร์ฟเวอร์

การใช้เมตธอดนี้ ยังทำให้การดูแลการทำงานของเว็บเซิร์ฟเวอร์เป็นไปในลักษณะที่ง่ายขึ้นส่วนมาก บนระบบ Unix จะใช้ inetd ซึ่งเป็น process ที่มีความเสถียรมาก ในทางตรงกันข้าม เว็บเซิร์ฟเวอร์ก็มีความซับซ้อนมากขึ้นและสามารถที่จะเสียหายได้อย่างคาดไม่ถึง แม้ว่าปัญหาเช่นนี้มี

โอกาสที่จะเกิดน้อยลงเมื่อใช้งานแอพพลิเคชันนานขึ้น นอกจากนี้ ถ้า inetd ยังคงทำงานอยู่ นั่นหมายถึงว่า ผู้ดูแลระบบไม่ต้องกังวลเกี่ยวกับการเริ่มและหยุดการทำงานของเว็บเซิร์ฟเวอร์เลย เนื่องจาก inetd จะทำงานเองโดยอัตโนมัติทุกครั้งที่มีการร้องขอเข้ามาทางพอร์ท

แต่ถ้ามองในอีกแง่หนึ่ง การที่มี process ทำงานอัตโนมัติทุกครั้งที่มีการร้องขอเข้ามานั้น ก็เป็นการเพิ่มค่าใช้จ่ายของการดูแลเว็บ และเป็นการทำงานที่ไม่ได้ผลนักสำหรับเว็บไซท์ที่มีผู้เข้าไปเยี่ยมชมจำนวนมาก ในปัจจุบันเว็บไซท์ส่วนมากประมวลผลเว็บเซิร์ฟเวอร์ที่สนับสนุนการทำงานแบบ multi-processing หรือ multi-threading และสามารถจัดการกับการให้บริการแก่ผู้ใช้จำนวนมากได้ดี

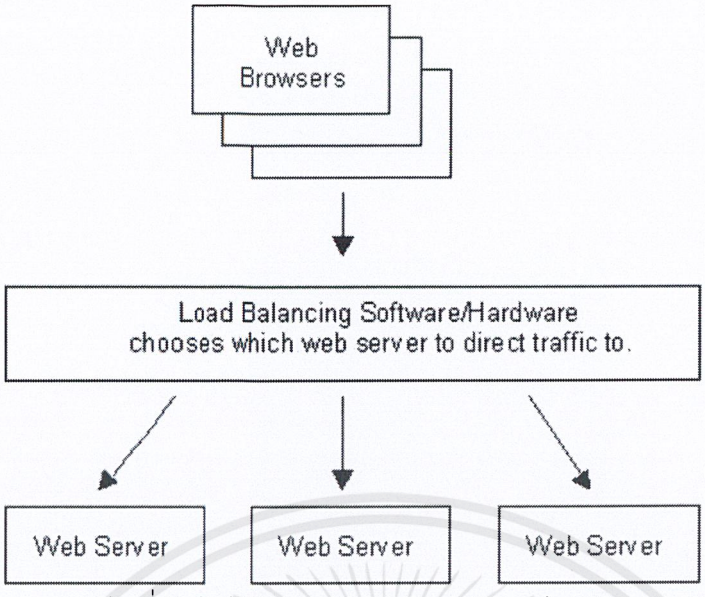
2.6.5 การติดตั้งเว็บเซิร์ฟเวอร์ให้ทำงานอย่างมีประสิทธิภาพ

ข้อจำกัดในเรื่องจำนวนข้อมูลที่จะให้บริการในช่วงเวลาหนึ่ง ๆ นั้นมีอยู่เสมอ ไม่ว่าเว็บเซิร์ฟเวอร์จะดีเพียงใด หรือเครื่องที่ทำหน้าที่เป็นเว็บเซิร์ฟเวอร์มีประสิทธิภาพมากเพียงใดก็ตาม โดยเฉพาะอย่างยิ่งถ้าข้อมูลที่ให้บริการเป็นแบบ dynamic เสียเป็นส่วนใหญ่ ข้อมูลชนิด dynamic มักจะมีการใช้งานฐานข้อมูลหรือมีการประมวลผลโปรแกรมอื่น ๆ ซึ่งต้องใช้ทรัพยากรในการทำงานสูงมาก

อีกปัญหาหนึ่งเกิดขึ้นขณะที่เว็บไซท์มีผู้เข้าชมมากเกินความสามารถในการให้บริการข้อมูล และมากเกินช่องทางในการส่งข้อมูลออกไป

การแก้ปัญหาเกี่ยวกับการส่งข้อมูลในการให้บริการนั้นมีอยู่ วิธีที่ง่ายที่สุดก็คือการแบ่งข้อมูลแต่ละส่วนไปเก็บไว้บนโฮสท์ต่าง ๆ เช่น เก็บไฟล์ HTML ชนิด static ไว้บนโฮสท์หนึ่ง, เก็บไฟล์ภาพทั้งหมดไว้บนอีกโฮสท์หนึ่ง และให้โฮสท์อีกที่หนึ่งทำหน้าที่ประมวลผลโปรแกรม CGI วิธีนี้เป็นวิธีที่ยังไม่ดีนัก และอาจมีผลเสียบางอย่างเกิดขึ้น อีกทั้ง ความสำเร็จในการแก้ปัญหาด้วยวิธีนี้ขึ้นอยู่กับลักษณะข้อมูลด้วย ตัวอย่างเช่น ถ้าโปรแกรม CGI หนึ่งทำให้เกิดปัญหาคอขวด การย้ายไฟล์นี้ไปทำงานบนอีกเว็บเซิร์ฟเวอร์หนึ่งช่วยให้ไฟล์ HTML, ไฟล์ภาพ, และโปรแกรม CGI อื่นทำงานต่อไปได้เท่านั้น ทำให้การทำงานของโปรแกรม CGI จะยังคงเป็นปัญหาคอขวดสำหรับผู้ใช้ในการใช้ทรัพยากรของระบบอยู่ ดังนั้นการแก้ปัญหาเกี่ยวกับ load balance บนเว็บเซิร์ฟเวอร์จึงต้องการวิธีการที่ซับซ้อนมากขึ้น

จำนวนปัจจัยต่าง ๆ ที่มีผลต่อการทำงานของระบบ ต้องได้รับการวิเคราะห์ก่อนที่จะเลือกวิธีการแก้ปัญหาได้อย่างเหมาะสม โดยเฉพาะการตรวจสอบลักษณะการเข้าถึงเว็บไซท์



ภาพที่ 2.53 ภาพแสดงกลไกที่เป็นไปได้ในการกระจายการทำงานไปยังเว็บเซิร์ฟเวอร์ต่าง ๆ

2.6.5.1 **การใช้** DNS (Data Name Server)

DNS balancing เป็นวิธีการที่ง่ายที่สุดที่จะทำให้เว็บไซท์รองรับการเข้าชมจากผู้ใช้ได้ในจำนวนมากขึ้น หลักการก็คือการมีข้อมูลทั้งหมดเหมือนกันบนหลาย ๆ เว็บเซิร์ฟเวอร์ อย่างไรก็ตาม แต่ละเว็บเซิร์ฟเวอร์จะต้องมีลักษณะที่คล้ายกัน

จากนั้น DNS Server จะถูกกำหนดให้สามารถคืนค่าไอพีแอดเดรสของเว็บไซท์นั้นออกมาหลายค่า DNS Server สามารถคืนค่าไอพีแอดเดรสทั้งสำหรับโฮสท์เนมหรือสำหรับแต่ละ DNS request

แม้ว่าเว็บเบราเซอร์จะมองว่าเป็นเว็บไซท์เดียว แต่สิ่งที่เกิดขึ้นก็คือการกระจายการทำงานไปยังเว็บเซิร์ฟเวอร์ต่าง ๆ วิธีการนี้เป็นวิธีการที่ใช้หลักการพื้นฐาน อย่างไรก็ตาม การเลือกไอพีแอดเดรสที่จะให้ไคลเอนท์ติดต่อนั้นทำได้ยาก และไคลเอนท์จะติดต่อกับเว็บเซิร์ฟเวอร์เดียวตลอดการเยี่ยมชมเว็บไซท์ นอกจากนี้ ยังมีความเป็นไปได้ว่าจำนวนผู้ใช้ที่ได้รับไอพีแอดเดรสหนึ่งมีมากกว่าอีกไอพีแอดเดรสหนึ่ง ดังนั้นจึงมีโอกาสที่เว็บเซิร์ฟเวอร์หนึ่งจะถูกใช้งานหนักมากกว่าอีกเว็บเซิร์ฟเวอร์หนึ่งแทนที่จะเป็นการแบ่งงานกันทำระหว่างสองเว็บเซิร์ฟเวอร์

หน่วยความจำแคชของ DNS อาจจะไม่เก็บข้อมูลไว้ตลอดการใช้งานของผู้ใช้ ดังนั้นจึงเป็นไปได้ว่าผู้ใช้อาจจะได้รับไอพีแอดเดรสเปลี่ยนไปเรื่อย ๆ ระหว่างที่เยี่ยมชมเว็บไซท์

เนื่องจากไคลเอนท์หนึ่ง ๆ มีโอกาสที่จะติดต่อกับหลายเว็บเซิร์ฟเวอร์ ดังนั้นข้อมูลที่อยู่บนเว็บเซิร์ฟเวอร์ต่าง ๆ ควรจะเหมือนกันทั้งหมด การควบคุมให้ข้อมูลบนเว็บเซิร์ฟเวอร์ต่าง ๆ เหมือนกันอยู่ตลอดเวลานี้เองที่เป็นปัญหา

2.6.5.2 **การเลือกใช้ ซอฟท์แวร์/ฮาร์ดแวร์**

วิธีการนี้คล้ายกับวิธีการที่ใช้ DNS แต่จะมีเพียงไอพีแอดเดรสเดียวที่ผู้ใช้รู้ แทนที่จะให้ไคลเอนท์เข้าถึงหลายเว็บเซิร์ฟเวอร์ได้จากหลายไอพีแอดเดรส กล่าวคือเครื่องที่มีไอพีแอดเดรสตามที่กำหนดไว้จะรับการร้องขอต่าง ๆ ไว้ แล้วกระจายงานไปยังเว็บเซิร์ฟเวอร์อื่น ๆ ที่มีข้อมูลของเว็บไซท์อยู่ โดยปกติการกระจายงานจะเกิดขึ้นในลักษณะของการหาเส้นทางบนโพรโตคอล TCP/IP ซึ่งจะแปลงไอพีแอดเดรสหนึ่งเป็นอีกไอพีแอดเดรสหนึ่งที่ต้องการ

การทำงานทั้งหมดนี้สามารถใช้ฮาร์ดแวร์หรือซอฟท์แวร์ช่วยได้ การใช้ฮาร์ดแวร์จะทำให้การทำงานดีกว่า แต่ค่าใช้จ่ายก็จะสูงกว่าอย่างแน่นอน วิธีการนี้เป็นวิธีที่ดีกว่าวิธีที่ใช้ DNS มาก เนื่องจากฮาร์ดแวร์ หรือซอฟท์แวร์ที่ทำหน้าที่ในการกระจายงานนั้น สามารถกระจายงานไปได้อย่างทั่วถึงมากกว่า

นอกจากนี้ ฮาร์ดแวร์หรือซอฟท์แวร์ที่ทำหน้าที่ในการกระจายงาน ก็ยังสามารถตรวจสอบได้ว่ามีเว็บเซิร์ฟเวอร์ใดที่มีปัญหาไม่สามารถทำงานได้ และจะทำการส่งงานของเว็บเซิร์ฟเวอร์นั้นไปให้กับอีกเว็บเซิร์ฟเวอร์หนึ่งทำงานแทน ในขณะที่การใช้ DNS ทำให้ไคลเอนท์ที่ติดต่อกับเว็บเซิร์ฟเวอร์ที่มีปัญหาไม่ได้รับบริการจากเว็บเซิร์ฟเวอร์แทนจนกว่าเว็บเบราเซอร์จะสามารถติดต่อกับไอพีแอดเดรสอื่นได้เอง

2.6.5.3 **การใช้** Proxy Server

อีกวิธีหนึ่งที่ทำงานได้ดีสำหรับการลดการส่งข้อมูลของเว็บไซท์คือการใช้ Reverse Proxy ซึ่งรับการร้องขอจากไคลเอนท์ แล้วส่งคำร้องขอเหล่านั้นไปยังเว็บเซิร์ฟเวอร์แทน โดยใช้หน่วยความจำแคชในการส่งข้อมูลที่มีกลับไปยังไคลเอนท์

วิธีนี้ใช้งานได้ดีสำหรับข้อมูลชนิด Static เนื่องจากทำให้ Proxy ไม่ต้องไปติดต่อขอข้อมูลจากเว็บเซิร์ฟเวอร์ทุกครั้ง แต่ก็ยังสามารถให้บริการด้วยข้อมูลที่ Proxy มีอยู่ จึงนับว่าเป็นการลดงานในการส่งข้อมูลจากเว็บเซิร์ฟเวอร์ และยังเป็นผลดีต่อการให้บริการข้อมูลชนิด Dynamic ด้วย ทั้งนี้เนื่องจากเว็บเซิร์ฟเวอร์มีงานในการส่งข้อมูลชนิด Static น้อยลงไป เพราะข้อมูลอยู่ที่ Proxy แล้ว จึงสามารถให้บริการแก่ข้อมูลชนิด Dynamic ได้มากขึ้น และในบางครั้งข้อมูลชนิด Dynamic นี้อาจถูกเก็บไว้ใช้งานด้วยหน่วยความจำแคชในเวลาสั้น ๆ ได้ การใช้ Reverse Proxy นี้จึงช่วยให้การให้บริการข้อมูลมีความเร็วเพิ่มขึ้นสูงมาก

วิธี Reverse Proxy สามารถนำไปใช้ร่วมกับวิธีการที่กล่าวมาแล้วก็ได้ กล่าวคือ ขณะที่ข้อมูลทั้งชนิด Static และ Dynamic ถูกแบ่งไปเก็บไว้บนเว็บเซิร์ฟเวอร์ต่าง ๆ Proxy ก็จะถูกใช้สำหรับการให้บริการข้อมูลชนิด Static เท่านั้น

2.6.5.4 การกระจายข้อมูล

เมื่อไม่นานมานี้ได้มีวิธีการจัดการเกี่ยวกับ Load Balance ที่สามารถจัดการเกี่ยวกับการส่งข้อมูลจำนวนมากเกิดขึ้น เช่น รูปภาพ นั่นคือผู้ให้บริการด้านข้อมูลโดยเฉพาะ ผู้ให้บริการเหล่านี้ได้เช่าหน่วยความจำที่ ISP ที่เปิดให้บริการแก่ผู้ใช้ทั่วโลก

จากนั้นผู้ให้บริการเหล่านี้ก็จะใช้วิธีการที่จัดการเกี่ยวกับปัญหา load balance ในการกระจายงานในด้านการส่งไฟล์ข้อมูลต่าง ๆ ไปทั่วโลก โดยทั่ว ๆ ไปจะใช้วิธี DNS และวิธีซอฟท์แวร์/ฮาร์ดแวร์ร่วมกัน ซึ่งจะสามารถบอกตำแหน่งทางภูมิศาสตร์ได้ด้วยว่าผู้ใช้แต่ละคนอยู่ที่ใดบนโลก เพื่อจะได้ส่งข้อมูลต่าง ๆ จากเว็บเซิร์ฟเวอร์ที่อยู่ใกล้ที่สุดไปยังผู้ใช้

2.6.6 การกำหนดค่าเพื่อให้เว็บเซิร์ฟเวอร์ทำงานได้อย่างมีประสิทธิภาพ

วิธีที่เพิ่มความเร็วในการทำงานของเว็บเซิร์ฟเวอร์อย่างชัดเจนที่สุดคือการทำให้ทรัพยากรต่าง ๆ ของระบบพร้อมต่อการถูกใช้งาน ทรัพยากรเหล่านี้รวมไปถึงความเร็วของดิสก์, หน่วยความจำและประสิทธิภาพ CPU

การมี CPU ที่มีประสิทธิภาพดีนั้นเป็นสิ่งสำคัญสำหรับการให้บริการข้อมูล ถึงกระนั้นก็ตาม การทำให้ CPU มีความพร้อมที่จะถูกใช้งานก็ไม่ได้เป็นผลนัก ถ้าข้อมูลที่ให้บริการเป็นข้อมูลชนิด dynamic เนื่องจากข้อมูลถูกสร้างขึ้นโดยโปรแกรม ดังนั้นเว็บไซท์ส่วนใหญ่ที่มีข้อมูลจำนวนมากจะใช้เทคนิคอื่นในการเพิ่มประสิทธิภาพในการทำงานของเว็บเซิร์ฟเวอร์

โดยทั่วไป ขั้นแรกในการให้บริการข้อมูลคือการอ่านข้อมูลมาจากดิสก์ ดังนั้นการเพิ่มความเร็วในการเข้าถึงข้อมูลที่อยู่ในดิสก์จึงเป็นเทคนิคหนึ่ง ซึ่งการเพิ่มความเร็วของ I/O นั้นก็มีอยู่หลายวิธี

ถึงแม้การปรับแต่งรูปแบบการทำงานของดิสก์ช่วยได้มาก แต่ในที่สุดความเร็วในการอ่านข้อมูลก็ยังมีขีดจำกัดที่ทำให้ไม่สามารถเพิ่มได้อีก ดังนั้นเว็บไซท์ส่วนใหญ่จะใช้การเพิ่มหน่วยความจำให้กับเว็บเซิร์ฟเวอร์แทน เนื่องจากเว็บเซิร์ฟเวอร์ที่มีหน่วยความจำมากก็จะยิ่งสามารถเก็บข้อมูลในหน่วยความจำแคชได้มากขึ้นเท่านั้น แทนที่จะต้องไปอ่านข้อมูลจากดิสก์หลาย ๆ ครั้ง

อีกเทคนิคหนึ่งคือการเก็บข้อมูลภาพและข้อมูลอื่น ๆ ไว้ในหน่วยความจำแคชของ proxy

ในที่สุด E-commerce Server ก็มีปัญหาคอขวดในการเข้ารหัส SSL transaction ต่าง ๆ ปัญหาคือการเชื่อมต่อกับ SSL ต้องมีค่าใช้จ่ายสูงมากสำหรับ CPU ที่มีประสิทธิภาพ นี่คืออีกกรณีหนึ่งที่แสดงให้เห็นว่าการมี CPU ที่ดีจะช่วยแก้ปัญหาได้ อย่างไรก็ดี ยังมีวิธีแก้ปัญหาที่ดีกว่าวิธีนี้อีก อย่างเช่น การเพิ่มความเร็วของ SSL แทนที่จะใช้งบประมาณไปกับการซื้อ CPU ที่มีประสิทธิภาพสูง ๆ องค์กรต่าง ๆ ก็หันไปซื้อ SSL acceleration card แทน เนื่องจาก SSL acceleration card เป็นหน่วยประมวลผลที่ยังมีราคาไม่สูง แต่สามารถทำงานในการเข้ารหัสได้อย่างรวดเร็วมาก การ์ดเหล่านี้มักจะมีข้อดีสำหรับเว็บเซิร์ฟเวอร์อีกข้อหนึ่ง คือ SSL key สามารถเก็บไว้ในการ์ดนี้ได้ โดยสามารถ

กำหนดให้บุคคลอื่นไม่สามารถอ่าน SSL key ได้ ข้อดีนี้มีความสำคัญเพราะถ้าเว็บไซท์ที่มีการใช้ SSL ถูกลักลอบเข้ามา SSL key อาจถูกขโมยไปได้ แต่ถ้าใช้ฮาร์ดแวร์ในการเข้ารหัส SSL key นี้ ก็จะไม่มีทางที่ SSL key จะถูกขโมยไปได้

2.6.7 การทำงานของระบบรักษาความปลอดภัย

การรักษาความปลอดภัยสำหรับเว็บเซิร์ฟเวอร์มีอยู่ 2 ระดับ หนึ่งคือการรักษาความปลอดภัยเกี่ยวกับ Data Stream เพื่อให้ข้อมูลไม่ถูกเปลี่ยนแปลง อีกระดับหนึ่งคือการรักษาความปลอดภัยเกี่ยวกับข้อมูล คือ Authentication และ Authorization

URL ที่ขึ้นต้นด้วย "https" จะถูกจัดการด้วยการใช้ SSL Algorithm ซึ่งทำงานโดยการกำหนดความปลอดภัยและเชื่อมโยงเว็บเบราเซอร์ และเว็บเซิร์ฟเวอร์เข้าด้วยกัน

SSL มีหน้าที่ในการปกป้องข้อมูลที่กำลังถูกส่งไปยังเว็บเซิร์ฟเวอร์ หรือปกป้องการอ่านข้อมูลลับบางอย่างจากเว็บเซิร์ฟเวอร์

ตัวอย่างหนึ่งของการส่งข้อมูลส่วนตัวไปยังเว็บเซิร์ฟเวอร์ก็คือ Web Store Application ซึ่งผู้ใช้จะต้องกรอกข้อมูลเกี่ยวกับบัตรเครดิตเพื่อสั่งซื้อสินค้า แม้ว่าเว็บเซิร์ฟเวอร์อาจจะไม่ส่งข้อมูลของผู้ใช้กลับมาเพื่อเป็นการยืนยัน แต่ว่าการส่งข้อมูลจริง ๆ จะต้องเป็นความลับ

จากนั้นก็จะต้องมีการดูแลข้อมูลที่อยู่บนเว็บเซิร์ฟเวอร์ที่ให้บริการ อาทิเช่น การประมูลสินค้าบนเว็บไซท์ ต้องการปกป้องราคาประมูลสำหรับสินค้าแต่ละชิ้น เพื่อไม่ให้ผู้ใช้คนหนึ่งเห็นราคาประมูลของอีกคนหนึ่ง กรณีนี้ การเข้ารหัสนั้นยังถือว่าไม่เพียงพอ เว็บเซิร์ฟเวอร์จะต้องสามารถที่จะระบุตัวผู้ใช้และข้อมูลที่ผู้ใช้คนนั้น ๆ สามารถเข้าถึงได้ด้วย กระบวนการทั้งสองอย่างนี้ก็คือ Authentication และ Authorization ตามลำดับนั่นเอง

เว็บเซิร์ฟเวอร์สนับสนุนการทำ Authentication ด้วยการใช้เทคนิคที่เรียกว่า Basic Authorization โดยที่หลักการของเทคนิคนี้คือ เว็บเซิร์ฟเวอร์จะส่งเฮดเดอร์พิเศษไปยังเว็บเบราเซอร์ของผู้ใช้เพื่อถาม username และ password

เว็บเซิร์ฟเวอร์ส่วนมากยอมให้ผู้ใช้อ่านข้อมูลต่าง ๆ ที่อยู่ภายในไดเร็กทอรีที่กำหนดไว้สำหรับผู้ใช้เท่านั้น แต่ถ้ามีการกำหนดการให้สิทธิ์ที่ละเอียดมากขึ้นก็จะต้องมีการเขียนโปรแกรมขึ้นมา

2.6.8 การประมวลผลเว็บแอพพลิเคชัน

เทคโนโลยีต่าง ๆ ทำให้เว็บเซิร์ฟเวอร์สามารถให้บริการข้อมูลจากแอพพลิเคชันได้ด้วย เช่น XML ไม่ได้ให้บริการเพียงแค่เอกสาร HTML

แม้เทคโนโลยีเหล่านี้มีลักษณะใกล้เคียงกับ Application Server แต่ในความจริงแล้ว เทคโนโลยีของเว็บแอพพลิเคชันเน้นที่การส่งเอกสาร HTML ชนิด Dynamic ที่สร้างขึ้นไปยังเว็บเบราเซอร์ของผู้ใช้

ขณะที่ Application Server นั้นไม่ได้มีหน้าที่จัดรูปแบบข้อมูลที่จะส่งไปยังผู้ใช้ แต่จะเป็น engine ที่ประมวลผลข้อมูลให้กับอีกโปรแกรม

2.7 แนะนำ Servlet

Servlet มีบทบาทสำคัญในการพัฒนาเว็บแอพพลิเคชันโดยใช้ Java เนื่องจาก Servlets มีความสามารถในการเขียน Server Side Code เพื่อติดต่อกับไคลเอนท์ ดังนั้น Servlet จึงเป็นหัวใจสำคัญในการทำ Server Side Programming

Servlet API มีรูปแบบทั่วไปของคลาสที่สามารถทำงานในการให้บริการได้ ซึ่งการให้บริการก็เป็นหน้าที่สำคัญของเซิร์ฟเวอร์อย่างหนึ่ง แต่ถ้าดูความหมายตามชื่อ Servlet ก็คือเซิร์ฟเวอร์ที่มีขนาดเล็กและมีลักษณะเฉพาะ

Servlet Container ช่วยให้ผู้เขียนโปรแกรม Servlet ไม่ต้องกังวลเกี่ยวกับรายละเอียดในการติดต่อกับเครือข่าย, การตรวจจับการร้องขอจากไคลเอนท์, การตอบสนองที่มีรูปแบบที่ถูกต้อง เนื่องจากสิ่งเหล่านี้เป็นหน้าที่ของ Servlet Container เอง หรือที่รู้จักกันว่า Servlet Engine โดย Servlet Container จะทำการแปลงคำร้องขอที่ส่งมาตามโพรโตคอลต่าง ๆ ให้เป็นอ็อบเจ็คท์ที่ Servlet เข้าใจได้ และจะส่งอ็อบเจ็คท์ไปยัง Servlet ซึ่ง Servlet จะนำไปใช้ในการตอบสนองต่อไป นอกจากนี้ Servlet Container ยังมีหน้าที่รับผิดชอบในการจัดการวัฏจักรการทำงานของ Servlet ด้วย

ทั้ง Servlet API Library และ Servlet Engine เป็นส่วนหนึ่งใน Java Servlet Development Kit (JSDK) โดยสามารถดาวน์โหลดได้จาก http://java.sun.com/products/servlet

2.7.1 ความหมายของ Servlet Container

Servlet Container มีหน้าที่ในการจัดการกับการร้องขอจากไคลเอนท์ด้วยการส่งคำร้องขอไปยัง Servlet และตอบสนองการร้องขอกลับไปยังไคลเอนท์นั้น การทำงานของ Servlet Container จะเปลี่ยนไปตามโปรแกรมต่าง ๆ แต่การทำงานระหว่าง Servlet Container และ Servlet ต่าง ๆ จะถูกระบุโดย Servlet API

ในการทำงานระหว่าง Servlet Container จะมีการกำหนดเมตธอดต่าง ๆ ขึ้น แล้ว Servlet Container จะเรียกให้ Servlet ทำงานตามเมตธอดนั้น ๆ ต่อไป นอกจากนี้ยังกำหนดคลาสต่าง ๆ ของอ็อบเจ็คท์ที่ Servlet Container จะส่งไปยัง Servlet ต่อไปด้วย

วัฏจักรการทำงานของ Servlet หนึ่ง ๆ มีลักษณะดังต่อไปนี้

- Servlet Container สร้าง instance ของ Servlet
- Servlet Container เรียกเมตธอดชื่อ init() ของ instance นั้น

- ถ้า Servlet Container ต้องการใช้ Servlet ในการให้บริการ Servlet Container ก็จะเรียกเมตธอด service() ของ instance นั้น
- ก่อนจะลบ instance ทิ้งไป Servlet Container จะเรียกเมตธอดชื่อ destroy() ของ instance นั้น
- instance นั้นจะถูกลบและถูกกำหนดให้กระบวนการ garbage collection มาจัดการต่อไป

การทำงานระหว่าง Servlet Container การันตีว่าก่อนที่เมตธอดชื่อ service() จะถูกเรียกนั้น เมตธอดชื่อ init() จะต้องทำงานเสร็จสมบูรณ์เสียก่อน และก่อนที่ instance ของ Servlet จะถูกลบไปนั้น เมตธอดชื่อ destroy() จะถูกเรียกให้ทำงานเสียก่อน ในทางปฏิบัติ มองว่า Servlet Container สร้าง instance ของ Servlet เมื่อ Servlet เริ่มทำงานหรือถูกเรียกให้ทำงานในครั้งแรก และถือว่า Servlet Container จะเก็บ instance ไว้ในหน่วยความจำ เพื่อให้บริการสำหรับคำร้องขออื่น ๆ ต่อไป Servlet Container อาจตัดสินใจที่จะลบ instance ออกไปจากหน่วยความจำเมื่อใดก็ได้ เช่น เมื่อ Servlet นั้นไม่ถูกเรียกใช้งานเลยในช่วงเวลาที่กำหนดไว้ หรือเมื่อ Servlet Container หยุดการทำงาน

ดังนั้น Servlet Container จะสร้างเพียงหนึ่ง instance สำหรับแต่ละ Servlet ถ้าเมตธอดชื่อ service() กำลังทำงานอยู่และมีอีกการร้องขอเข้ามา Servlet Container สามารถรอจนกระทั่งเมตธอดชื่อ service() นั้นทำงานจนเสร็จสมบูรณ์ ก่อนจะทำการเรียกเมตธอดชื่อ service() มาทำงานอีกครั้งหนึ่ง หรือสร้างอีก thread เพื่อให้บริการ โดยการเรียกเมตธอดชื่อ service() เช่นกัน เนื่องจากไม่มีข้อกำหนดว่ามีเพียง thread เดียวที่สามารถเรียกใช้เมตธอดชื่อ service() ในเวลาหนึ่ง ๆ

ในทางปฏิบัติ Servlet Container จะไม่สร้าง thread ใหม่ทุก ๆ ครั้งที่รับคำร้องขอ แต่ Servlet Container จะใช้ pool ของ thread ในการจัดการกับการร้องขอที่เข้ามาพร้อม ๆ กันแทน แต่ผลกระทบที่มีต่อ Servlet ของทั้งสองวิธีก็มีมากเท่า ๆ กัน

เนื่องจากเว็บเซิร์ฟเวอร์ส่วนใหญ่ถูกสร้างขึ้นด้วยภาษาอื่น เช่น ภาษา C หรือ C++ ดังนั้นขณะที่เว็บเซิร์ฟเวอร์ที่สร้างด้วย Java อาทิ Sun's Java Web Server และ ServletRunner ที่มีอยู่ใน JSDK มี Servlet Container เป็นของตัวเอง เว็บเซิร์ฟเวอร์อื่น ๆ อาทิ Apache และ Microsoft IIS จึงต้องการโปรแกรมบางอย่างเพิ่มเติมเพื่อควบคุมการทำงานของ Servlet นอกจากนั้น ยังต้องการ plug-in หรือโมดูลในการจัดการการติดต่อกันระหว่างเว็บเซิร์ฟเวอร์กับ Servlet Container ยกตัวอย่างเช่น Apache Jserv ที่มีโพรโตคอล internet พิเศษที่ชื่อว่า AJPv1.1 ในการจัดการเกี่ยวกับการติดต่อกันนี้ โดยเว็บเซิร์ฟเวอร์จะส่งคำร้องขอไปยังไฟล์ Servlet และ Servlet Container จะส่งผลลัพธ์กลับไปยังเว็บเซิร์ฟเวอร์ ซึ่งจะเห็นว่าในกรณีนี้ Servlet Container และเว็บเซิร์ฟเวอร์ไม่จำเป็นต้องทำงานอยู่บนเครื่องเดียวกัน

เนื่องจาก Servlet มีการทำงานใกล้ชิดกับเว็บเซิร์ฟเวอร์มาก ดังนั้น Servlet จึงสามารถช่วยเพิ่มความสามารถของเว็บเซิร์ฟเวอร์ได้เป็นอย่างดี Servlet มีความสามารถกระทั่งการทำงานพื้นฐานส่วนใหญ่ กล่าวคือ Servlet สามารถจัดการกับ process มาตรฐานของเว็บเซิร์ฟเวอร์ได้ เช่น การแปลง

URL ที่ร้องขอมาให้เป็นตำแหน่งของไฟล์ (filepath), การส่งไฟล์ HTML คืนกลับไป และยังรวมไปถึงการประมวลผล CGI, การทำงานของ Java Server Pages (JSP), เท็มเพลทของ HTML และ Server Side Includes (SSI) นั่นแสดงว่า Servlet สามารถทำงานได้ทุกอย่างบนเว็บเซิร์ฟเวอร์

2.7.2 Servlet API ที่สำคัญ

คลาสและอินเทอร์เฟสต่าง ๆ อยู่ใน 2 แพ็คเกจ ได้แก่ javax.servlet และ javax.servlet.http โดยที่แพ็คเกจแรกมีอินเทอร์เฟสพื้นฐาน และและแพ็คเกจหลังมีคลาสต่าง ๆ ที่ได้รับมาจากอินเทอร์เฟสชื่อ GenericServlet ซึ่งมีเครื่องมือพิเศษที่ใช้ในการให้บริการการร้องขอบนโพรโตคอล HTTP

2.7.2.1 อินเทอร์เฟส Servlet

วัฏจักรการทำงานของ Servlet ถูกกำหนดไว้ในอินเทอร์เฟสชื่อ javax.servlet.Servlet เมื่อสร้าง Servlet จะต้องเพิ่มคำสั่ง “implements” ให้กับอินเทอร์เฟสนี้ด้วย ไม่ว่าจะโดยทางตรงหรือทางอ้อม ส่วนมากจะทำทางอ้อมด้วยการใช้คำสั่ง “extends” กับคลาส javax.servlet.GenericServlet หรือคลาส javax.servlet.http.HttpServlet โดยขณะที่มีการใช้คำสั่ง “implements” กับอินเทอร์เฟส Servlet ห้าเมตธอดต่อไปนี้จะต้องถูก implement ไปด้วย

- init()

public void init(Servletconfig config) throws ServletException

เมื่อ Servlet ถูกสร้างขึ้น Servlet Container จะเรียกเมตธอดชื่อ init() โดย Servlet Container จะส่งอ็อบเจ็คท์ชนิด ServletConfig ไปยังเมตธอด init() เพื่อใช้เก็บข้อมูลที่เกี่ยวกับการกำหนดค่าของ Servlet Engine ซึ่งจะถูกนำไปใช้ต่อไปในภายหลัง เมตธอด init() จะส่งเหตุการณ์ที่เกิดจาก ServletException เนื่องจากมีการประกาศ throws ServletException เมื่อเกิดเหตุการณ์เช่นนี้ Servlet นั้นจะไม่สามารถให้บริการได้ แล้วจะเรียกไปยัง Servlet ที่ทำให้ Servlet นั้นถูกเรียกขึ้นมาใช้งานอีกครั้ง โดย Servlet Container และเมตธอดชื่อ init() ก็จะถูกเรียกทำงานอีกครั้งหนึ่ง อินเทอร์เฟส Servlet การันตีว่าเมตธอด init() จะถูกเรียกเพียงครั้งเดียว และการันตีว่าเมตธอด init() จะสามารถทำงานได้เสร็จสมบูรณ์โดยไม่มีการส่งเหตุการณ์ที่เกิดจาก ServletException ก่อนที่จะมีการร้องขอใด ๆ ถูกส่งไปยัง Servlet

- service()

public void service(ServletRequest req, ServletResponse res) throws ServletException, IOException

เมตธอด Service() จะถูกเรียกก็ต่อเมื่อ Servlet พร้อมสำหรับการให้บริการการร้องขอที่เข้ามาแล้ว เมตธอด Service() รับพารามิเตอร์ 2 ตัว ได้แก่อ็อบเจ็คท์ชื่อ ServletRequest และ ServletResponse โดย ServletRequest มีเมตธอดที่จะเข้าถึงข้อมูลเกี่ยวกับการร้องขอ ส่วน ServletResponse มีเมตธอดที่ Servlet ใช้สร้าง Response

- destroy()

public void destroy()

ณ เวลาใด ๆ ที่กำหนดไว้ Servlet Container อาจตัดสินใจที่จะลบ Servlet ออกไป ซึ่งอาจเกิดขึ้นเมื่อต้องการหน่วยความจำเพิ่มหรือเมื่อเว็บเซิร์ฟเวอร์ถูกปิดการทำงาน ก่อนที่ Servlet Container จะเรียกเมตธอดนี้ Servlet Container จะรอให้ thread ที่ทำหน้าที่ในการให้บริการได้ทำงานให้เสร็จเสียก่อน ดังนั้นอินเทอร์เฟส Servlet การันตีว่าเมตธอด destroy() จะไม่ทำงานขณะที่เมตธอด service() กำลังทำงานอยู่

- getServletConfig()

public ServletConfig getServletConfig()

ระหว่างที่ Servlet กำลังเริ่มต้นทำงาน อ็อบเจ็คท์ ServletConfig ที่ถูกส่งจาก Servlet Engine จะเก็บ instance ของ Servlet ไว้ โดย ServletConfig สามารถเข้าถึงพารามิเตอร์ Init และอ็อบเจ็คท์ ServletContext ได้ พารามิเตอร์ Init มักจะถูกระบุไว้ในไฟล์ของ Servlet Container โดยมีจุดประสงค์เพื่อให้ Servlet ได้รับข้อมูลที่จำเป็นต้องใช้ขณะ runtime ขณะที่ ServletContext ช่วยให้ Servlet มีความสามารถในการค้นหาข้อมูลที่เกี่ยวกับ Servlet Container อนึ่ง การเรียกใช้อ็อบเจ็คท์ทั้งสองโดยเมตธอด getServletConfig นี้ จะทำเมื่อใดก็ได้

- getServletInfo()

public String getServletInfo()

เมตธอดนี้คืนค่าเป็นอ็อบเจ็คท์ชนิดสตริงที่เก็บข้อมูลเกี่ยวกับ Servlet เช่น ชื่อผู้สร้าง Servlet, วันที่สร้าง Servlet, คำบรรยาย เป็นต้น Servlet Container ก็สามารถใช้เมตธอดนี้ได้เช่นกัน โดยอาจเก็บข้อมูลเป็นลิสท์ของ Servlet ต่าง ๆ ที่อยู่ด้วยกันพร้อมคำอธิบาย

2.7.2.2 อินเทอร์เฟส GenericServlet

คลาส GenericServlet สามารถใช้คำสั่ง “implements” ร่วมกับอินเทอร์เฟส Servlet ได้ จะสังเกตเห็นได้ว่าคลาส GenericServlet ถูกประกาศไว้เป็น abstract เนื่องจากว่าเมตธอด service() ถูกประกาศเป็น abstract ซึ่งหมายความว่าเมื่อเราใช้คำสั่ง “extends” กับคลาสนี้ เราต้อง implement เมตธอด service() ด้วย ส่วนเมตธอดอื่นจะถูก implement ดังนี้

เมตธอด init(ServletConfig conf) เก็บอ็อบเจ็คท์ชนิด ServletConfig ไว้ใน instance variable แบบ private transient หรือที่เรียกกันว่า config ซึ่งเมตธอด getServletConfig() สามารถเข้าถึงและคืนค่า reference ที่ใช้อ้างถึงอ็อบเจ็คท์นั้นได้ นั่นหมายความว่าถ้าเรา override เมตธอด getServletConfig() โดยไม่ระวัง เราจะไม่สามารถใช้เมตธอดนี้ในการเรียกใช้อ็อบเจ็คท์ชนิด ServletConfig ได้ ด้วยเหตุผลนี้ เราจะต้องทำการเรียก super.config(conf) ด้วย

API เวอร์ชัน 2.1 ของ Java Servlet มีเมตธอด init() ที่ไม่มีอาร์กิวเมนท์อยู่ในคลาส GenericServlet เพื่อให้ override ได้ การ override จะทำให้เราสามารถเขียนโคดใน Servlet นั้นได้โดยไม่ต้องเรียกใช้ super.init(conf)

ความสามารถที่เพิ่มขึ้นมาใน API เวอร์ชัน 2.1 คือ คลาส GenericServlet สามารถ implement อินเทอร์เฟส ServletConfig ได้ ทำให้ผู้พัฒนา Servlet สามารถเรียกใช้เมตธอด ServletConfig() ได้โดยตรง โดยไม่ต้องเรียกผ่าน อ็อบเจ็คท์ชนิด ServletConfig โดยเมตธอดเหล่านี้ได้แก่ getInitParameter(), getInitParameterNames(), getServletContext() ซึ่งแต่ละเมตธอดจะไปเรียกเมตธอดที่เกี่ยวข้องที่อยู่ในอ็อบเจ็คท์ชนิด ServletConfig

คสาส GenericServlet ยังมีอีกสองเมตธอดที่ใช้สร้าง Servlet Log ซึ่งในความเป็นจริงจะเรียกเมตธอดที่เกี่ยวข้องของ ServletContext มาช่วยทำงาน โดยเมตธอดแรกก็คือ log(String msg) จะเขียนชื่อของ Servlet และอาร์กิวเมนท์ msg ลงใน log ของ Servlet Container อีกเมตธอดหนึ่งคือ log(String msg, Throwable cause) ซึ่งมี exception เพิ่มเข้าไปใน log นอกเหนือจากชื่อของ Servlet และอาร์กิวเมนท์ชื่อ msg

2.7.2.3 คลาส HttpServlet

คลาส HttpServlet extend คลาส GenericServlet และมีเมตธอดที่เกี่ยวกับโพรโตคอล HTTP อยู่ในอินเทอร์เฟส ชื่อ Servlet ให้เรียกใช้ คลาสนี้เป็นคลาสที่เรามักจะเรียกใช้โดย extend ด้วย Servlet ของเรา การ extend คลาสชื่อ GenericServlet ทำให้เราสามารถมองเห็นขอบเขตการทำงานที่เกี่ยวกับการให้บริการบนเครือข่าย แต่ HTTP เป็นโพรโตคอลที่นิยมใช้กับ Servlet มากที่สุด

- service()

protected void service (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException

public void service(ServletRequest req, ServletResponse res) throws ServletException, IOException

เมตธอด service() จะถูก implement โดย HttpServlet ที่ทำหน้าที่ในการจัดการกับการร้องขอบนโพรโตคอล HTTP ดังนั้นจึงไม่ควร override เมตธอดนี้ เมื่อมีการร้องขอ เมตธอด service() จะตรวจสอบชนิดของการร้องขอนั้น ซึ่งได้แก่ GET, POST, HEAD, OPTIONS, DELETE, PUT และ TRACE และจัดการกับการร้องขอนั้นด้วยเมตธอดที่ทำงานสำหรับการร้องขอชนิดนั้น ๆ อันได้แก่ doGet(), doPost(), doHead(), doOptions(), doDelete(), doPut() และ doTrace() การจะเขียน Servlet ที่โต้ตอบกับการร้องขอตามชนิดที่เจาะจง เราทำได้ด้วยการ override เมตธอด doXxx() ของการร้องขอชนิดนั้น ๆ แต่ถ้ามีการร้องขอชนิดใดที่ไม่ได้ override ไว้ Servlet จะคืนค่าความผิดพลาดที่เป็นมาตรฐานที่บอกว่าเมตธอดนั้นไม่สามารถใช้งานได้

- getLastModified()

protected long getLastModified(httpServletRequest req)

เมตธอดนี้จะคืนค่ามาเป็นเวลาที่ Servlet ถูกแก้ไขล่าสุดในรูปแบบ 00:00:00 ตามมาตรฐาน GMT แต่ถ้าไม่รู้เวลาในขณะที่เกิดการแก้ไข ค่าดีฟอลท์ที่ส่งคืนมาจะเป็นตัวเลขที่เป็นค่าลบ เมื่อมีการให้บริการแก่การร้องขอที่ใช้เมตธอด GET นี้ จะสามารถบอกเซิร์ฟเวอร์ได้ว่า Servlet ที่ให้บริการนั้นถูกแก้ไขล่าสุดเมื่อใด

2.7.2.4 **อินเทอร์เฟส** HttpServletRequest

เพื่อให้เข้าใจอินเทอร์เฟสนี้ได้ง่าย เราจะต้องรู้ด้วยว่าโพรโตคอล HTTP ยอมให้ข้อมูลส่งผ่านไปยังเว็บเซิร์ฟเวอร์ได้อย่างไร โพรโตคอล HTTP ยอมให้เราส่งพารามิเตอร์ต่าง ๆ ไปพร้อม ๆ กับการร้องขอได้ ในการร้องขอแบบ GET นั้น พารามิเตอร์เหล่านี้ถูกเพิ่มเข้าไปตามหลัง URL ของคำร้องขอในฟอร์มของคำสั่งในการสืบค้นข้อมูล แต่สำหรับการร้องขอแบบ POST พารามิเตอร์เหล่านี้จะอยู่ในส่วนของข้อความของการร้องขอ ในทั้งสองกรณี พารามิเตอร์จะอยู่ในรูปคู่ของชื่อพารามิเตอร์กับค่าของพารามิเตอร์ อย่างไรก็ตาม โพรโตคอล HTTP ไม่ได้กำหนดว่าชื่อพารามิเตอร์จะต้องไม่ซ้ำกัน ดังนั้นพารามิเตอร์บางตัวจึงสามารถมีค่าเป็นชุด ๆ ได้ ซึ่งอาจเกิดขึ้นเนื่องจากฟอร์มในภาษา HTML มีลักษณะแบบนี้

อ็อบเจ็คท์ที่ implement อินเทอร์เฟส HttpServletRequest เช่น อ็อบเจ็คท์ของการร้องขอบนโพรโตคอล HTTP ที่ถูกส่งจาก Servlet Engine จะช่วยให้ไฟล์ Servlet สามารถเข้าถึงข้อมูลที่เกี่ยวกับการร้องขอไปจนถึงเมตธอดต่าง ๆ

ต่อไปนี้เป็นเมตธอดพื้นฐานที่จะต้องใช้เพื่อรับข้อมูลจากฟอร์มในภาษา HTML

- getParameter()

public String getParameter(String key)

เมตธอดนี้จะพยายามหาพารามิเตอร์ตามชื่อที่ส่งมากับคำสั่ง แล้วจึงคืนค่าของพารามิเตอร์นั้น ๆ กลับมา ถ้าค่าของพารามิเตอร์นั้น ๆ มีหลายค่า ค่าที่ถูกส่งกลับมาจะเป็นค่าแรก

- getParameterValues()

public String[] getParameterValues(String key)

ถ้าพารามิเตอร์หนึ่งมีหลายค่าที่สามารถคืนกลับมาได้ เช่น เซ็ตของเช็คบ็อกซ์ เราควรใช้เมตธอดนี้ในการรับค่าทั้งหมดของพารามิเตอร์นั้น

- getParameterNames()

public Enumeration getParameterNames()

เมตธอดนี้จะคืนค่าเป็นอ็อบเจ็คท์ชนิด Enumeration และชื่อของพารามิเตอร์ทั้งหมด

2.7.2.5 **อินเทอร์เฟส** HttpServletResponse

Servlet Engine มีอ็อบเจ็คท์ที่ implement อินเทอร์เฟสนี้ และส่งไปยัง Servlet หรือเมตธอดที่ให้บริการ โดยที่ Servlet ที่ให้บริการนั้น สามารถดัดแปลงเฮดเดอร์ของการตอบสนอง และส่งผลลัพธ์คืนไปโดยใช้อ็อบเจ็คท์ชนิด HttpServletResponse และเมตธอดของอ็อบเจ็คท์นี้

ต่อไปนี้เป็นเมตธอดพื้นฐานที่ควรรู้

- setContentType()

public void setContentType(String type)

ก่อนจะมีการตอบสนองกลับไป เมตธอดนี้จะต้องถูกเรียกก่อน เพื่อจะกำหนด MIME type ของการตอบสนอง โดยจะเป็น MIME type ใดก็ได้ แต่ถ้าจะสร้างเอกสาร HTML เพื่อทำการส่งกลับไปยังเว็บเบราเซอร์ MIME type ก็ควรจะเป็น text/html

- getWriter()

public PrintWriter getWriter() throws IOException

เมตธอดนี้จะคืนค่ากลับเป็นอ็อบเจ็คท์ชนิด PrintWriter ที่สามารถใช้ในการส่งเมสเสจสำหรับการตอบสนองรูปแบบที่เป็นตัวอักษรทั้งหมด เพื่อตอบกลับไปยังผู้เรียก PrintWriter แปลงตัวอักษรชนิด Java's internal Unicode ให้ถูกต้องอย่างอัตโนมัติ เพื่อให้ไคลเอนท์สามารถอ่านข้อความนั้นได้ อนึ่ง การใช้อ็อบเจ็คท์ชนิด PrintWriter เราจะเขียนข้อมูลลงไปยังอ็อบเจ็คท์ชนิด esponse โดยใช้เมตธอดชื่อ println(String txt)

- getOutputStream()

public ServletOutputStream getOutputStream() throws IOException

คืนค่าเป็น ServletOutputStream ซึ่งเป็นสับคลาสของ java.io.OutputStream อ็อบเจ็คท์นี้สามารถใช้ในการส่งข้อมูลชนิดไบนารีกลับไปยังไคลเอนท์

- setHeader()

public void setHeader(String name, String value)

เมตธอดนี้สามารถใช้ในการกำหนดเฮดเดอร์ที่จะส่งกลับไปยังไคลเอนท์ ถึงแม้จะมีหลายเมตธอดที่ใช้กำหนดเฮดเดอร์ แต่ในบางสถานการณ์เราอาจจะต้องการใช้เมตธอดนี้แทน

2.7.3 การใช้งาน Servlet

การคอมไพล์โคดต่อไปนี้ จะต้องลง JSDK ถ้ากำลังใช้ Java2 ก็เพียงแต่ย้าย servlet.jar จาก JSDK\LIB ไปยัง JDK\JRE\LIB\EXT แต่ถ้าใช้ JDK 1.1 ก็เพิ่ม JSDK\LIB\SERVLET.JAR ลงไปในตัวแปร CLASSPATH

```
//Import Servlet Libraries
import javax.servlet.*;
import javax.servlet.http.*;
Import Java Libraries
import java.io.*;
public class HelloWorld extends HttpServlet{
        public void doGet(HttpServletRequest req, HttpServletResponse res) throws
ServletException, IOException{
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        out.println("<HTML>");
        out.println("<HEAD>");
        out.println("<TITLE>Hello World Sample Servlet</TITLE>");
        out.println("</HEAD>");
        out.println("<BODY>");
        out.println("<CENTER><H1>Hello World!</H1></CENTER>");
        out.println("</BODY>");
        out.println("</HTML>");
        out.close();
        }
}
```

ภาพที่ 2.54 ภาพตัวอย่างไฟล์ Servlet

บทที่ 3
วิธีดำเนินการวิจัย

วิธีดำเนินการวิจัยปัญหาพิเศษ เรื่องโปรแกรมเกตเวย์สำหรับแปลงรูปแบบเอกสาร XML บนเว็บเซิร์ฟเวอร์ มีขั้นตอนการดำเนินการ ซึ่งสามารถแบ่งออกเป็น 3 ส่วน ดังนี้

1. ขั้นตอนการศึกษาโครงสร้างภาษา HTML และ ภาษา WML เพื่อหาความสัมพันธ์ระหว่างภาษา HTML กับ ภาษา WML ว่ามีแท็กใดบ้างที่เหมือนกัน เพื่อใช้เป็นหลักในการสร้างเอกสาร XSL สำหรับใช้ในการแปลงเอกสาร

2. ขั้นตอนการสร้างเอกสาร XSL สำหรับใช้ในการแปลงเอกสาร

- HTML ให้เป็นภาษา WML และในทางกลับกัน
- WML ให้เป็นภาษา HTML และในทางกลับกัน

และรูปแบบเอกสารอื่น ๆ ในอนาคต เพื่อใช้สำหรับเป็นอินพุตให้กับตัวแปลงภาษารูปแบบ XSL ที่ทำงานบนเว็บเซิร์ฟเวอร์ สำหรับการแปลงเอกสารให้เหมาะสมกับเอกสารที่ไคลเอนท์ร้องขอ

3. ขั้นตอนการออกแบบและสร้างเว็บเซิร์ฟเวอร์ที่ทำงานภายใต้โพรโตคอล HTTP เพื่อรอรับการร้องขอเอกสารที่ไคลเอนท์ร้องขอเข้ามา และตอบสนองการร้องขอเอกสารนั้นกลับไปยังไคลเอนท์

4. ขั้นตอนการออกแบบสร้างตัวแปลงภาษารูปแบบ XSL ที่ทำงานบนเว็บเซิร์ฟเวอร์ เพื่อใช้ในการแปลงเอกสารที่มีอยู่ในเว็บเซิร์ฟเวอร์ ให้เป็นเอกสารในรูปแบบที่ไคลเอนท์ร้องขอเข้ามา ก่อนส่งให้เว็บเซิร์ฟเวอร์ส่งกลับไปยังไคลเอนท์

3.1 ขั้นตอนการศึกษาโครงสร้างภาษา HTML และภาษา WML

การศึกษาโครงสร้างจากภาษา HTML และ WML นั้น จะศึกษา DTD (Document Type Definition) เพื่อให้เห็นว่าแต่ละแท็กในภาษาต่าง ๆ มีแอตทริบิวต์อะไรบ้าง และนำมาเปรียบเทียบว่าแท็กต่าง ๆ ของภาษานี้ทำงานเช่นเดียวกับแท็กของภาษาอะไร เพื่อที่จะนำมาสร้างเป็นเอกสาร XSL ใช้ในการทำงานร่วมกับตัวแปลงเอกสารแบบ XSL ต่อไป

ตารางที่ 3.1 ตารางเปรียบเทียบแท็กของภาษา HTML และภาษา WML

TAG	WML	HTML
1. wml	<wml> </wml> xml:lang = nmtoken	<html> </html>
2. card	<card> </card> title = vdata newcontext = boolean ordered = boolean onenterforward = href onenterbackward = href ontimer = href xml:lang id class	
3. template	<template> </template> onenterforward = href onenterbackward = href ontimer = href id class	
4. head	<head> </head>	<head> </head>
5. access	<access> domain = cdata path = cdata id class	

TAG	WML	HTML
6. meta	<meta/> content = cdata name = cdata http-equiv = cdata forua = boolean scheme = cdata id class	<meta> content = cdata name = name http-equiv = name scheme = cdata
7. do	<do> </do> type = cdata label = vdata name = nmtoken optional = boolean xml:lang id , class	
8. onevent	<onevent> </onevent> type = cdata id , class	
9. postfield	<postfield/> name = vdata value = vdata Id class	
10. go	<go> </go> href = href sendreferer = boolean method = (post | get) accept-charset = cdata class	

TAG	WML	HTML
11. prev	<prev/> id class	
12. refresh	<refresh> </refresh> id class	
13. noop	<noop/> id class	
14. setvar	<setvar/> name = vdata value = vdata id class	
15. input	<input/> name = nmtoken value = vdata type = (text | password) format = cdata emptyok = boolean size = number maxlength = number title = vdata tabindex = number xml:lang id class	<input> name = cdata value = cdata type = (text | password | checkbox | radio | submit | reset | file | hidden | image | button) size = cdata maxlength = number title tabindex lang id class checked src = uri

TAG	WML	HTML
		style alt align accept readonly disabled accesskey Usemap onfocus onblur onselect onchange onclick ondblclick
16. select	<select> </select> multiple = boolean name = nmtoken value = vdata iname = nmtoken ivalue = vdata title = vdata tabindex = number xml:lang id class	<select> </select> multiple name = cdata title tabindex lang id class size = number style disabled onclick ondblclick onmousedown onmouseup onmouseover

TAG	WML	HTML
		onmousemove onmouseout onkeypress onkeydown onkeyup
17. option	<option> </option> value = vdata title = vdata onpick = href xml:lang Id class	<option> </option> value = cdata title lang id class label = text name = cdata size = number multiple style disabled tabindex onclick ondblclick onmousedown onmouseup onmouseover
18. optgroup	<optgroup> </optgroup> title = vdata xml:lang id class	<optgroup> </optgroup> title lang id class label = text

TAG	WML	HTML
19. fieldset	<fieldset> </fieldset> title = vdata xml:lang id class	
20. anchor	<anchor> </anchor> title = vdata xml:lang id class	
21. a	<a> title = vdata href xml:lang id class	<a> href name rel rev tabindex target accesskey
22. img	<img/> alt = vdata src = href localsrc = vdata vspace = length hspace = length align = (top | middle | bottom) height = length width = length xml:lang id , class	<img> alt src = uri longdesc = uri border vspace hspace align height width lang id , class

TAG	WML	HTML
		title style usemap onclick ondblclick onmousedown onmouseup onmouseover onmousemove onmouseout onkeypress onkeydown onkeyup
23. timer	<timer/> name = nmtoken value = vdata id class	
24. em		
25. strong		
26. i		
27. b		
28. u	<u> </u>	<u> </u>
29. big	<big> </big>	<big> </big>
30. small		
31. br	 xml:lang id , class	 id , class clear title , style

TAG	WML	HTML
32. p	 align = (left | right | center) mode = (wrap | nowrap) xml:lang id class	 align lang id class title style onclick ondblclick onmousedown onmouseup onmouseover onmousemove onmouseout onkeypress onkeydown onkeyup
33. table	<table> </table> title = vdata align = cdata columns = number xml:lang id class	<table> </table> title align = left | center | right lang id class summary = text width = length style onclick ondblclick onmousedown onmouseup

TAG	WML	HTML
		onmouseover onmousemove onmouseout onkeypress onkeydown onkeyup bgcolor frame border cellspacing cellpadding
34. tr	<tr> </tr> id class	<tr> </tr> id class lang title style onclick ondblclick onmousedown onmouseup onmouseover onmousemove onmouseout onkeypress onkeydown onkeyup align char , charoff valign

TAG	WML	HTML
35. td	<td> </td> xml:lang id class	<td> </td> lang id class headers = idrefs scope = scope-name abbr = text axis = cdata rowspan = number colspan = number width = pixels height = pixels title style onclick ondblclick onmousedown onmouseup onmouseover onmousemove onmouseout onkeypress onkeydown onkeyup bgcolor align char charoff valign

3.2 ขั้นตอนการสร้างเอกสาร XSL สำหรับใช้ในการแปลงเอกสาร

```
<xsl:stylesheet  xmlns:xsl = "http://www.w3.org/1999/XSL/Transform" >
 <xsl:output  method = "xml"  doctype-public   = "-//WAPFORUM//DTD WML 1.1//EN"
        doctype-system =  "http://www.wapforum.org/DTD/wml_1.1.xml" />
 <!-- match root element -->
 <xsl:template match="/">
  <wml>
     <card title ="{//title}">
        <xsl:apply-templates select = ".//body"/>
      </card>
  </wml>
 </xsl:template>
 <!-- meta part -->
 <!-- body part -->
 <xsl:template match="body">
   <p><xsl:apply-templates /></p>
 </xsl:template>
 <xsl:template match = "br"> <br/>
 </xsl:template>
 <xsl:template match = "b|strong">
   <b><xsl:apply-templates /></b>
 </xsl:template>
 <xsl:template match="img">
   <img src = "{@src}" alt = "{@alt}" width = "{@width}" height = "{@height}"/>
 </xsl:template>
 </xsl:stylesheet>
```

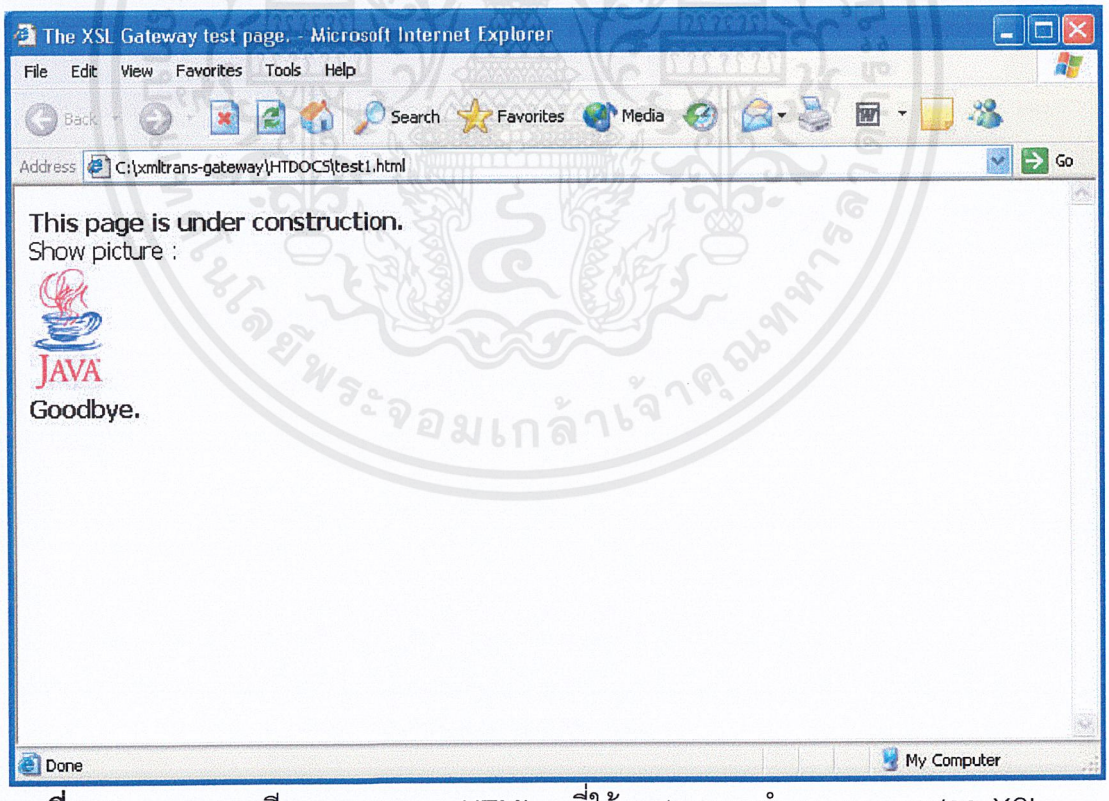
ภาพที่ 3.1 ภาพแสดงเอกสาร XSL สำหรับการแปลงเอกสาร HTML เป็น WML

การสร้างเอกสาร XSL สำหรับใช้ในการแปลงเอกสาร อาศัยแท็กพื้นฐานที่เหมือนกันของภาษาในแต่ละคู่ ในการแปลงจากภาษาหนึ่งไปเป็นอีกภาษาหนึ่งเป็นหลักในการสร้าง

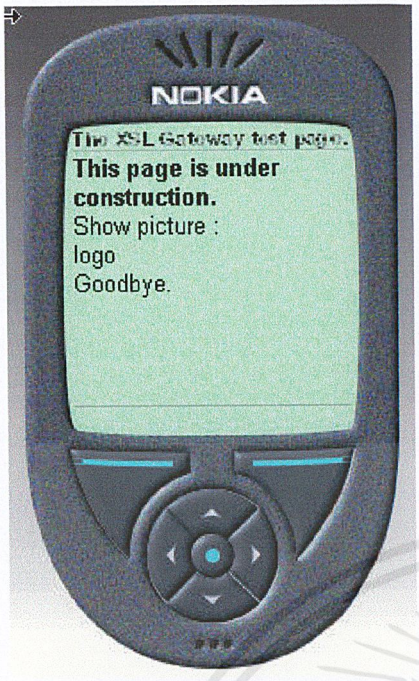
เราสามารถใช้เอกสาร XSL ที่ได้จากภาพที่ 3.1 มาใช้งานโดยการแปลงเอกสาร HTML ที่มีรายละเอียดตามภาพที่ 3.2 ให้เป็นเอกสาร WML และการแสดงผลเป็นไปดังภาพที่ 3.3

```
<html>
<head>
  <title>The XSL Gateway Test page.</title>
</head>
<body>
<b>This page is under construction.</b><br/>
Show picture : <br/><img src="/images/javalogo.gif" alt="logo" /><br/>
<strong>Goodbye.</strong>
</body>
</html>
```

ภาพที่ 3.2 ภาพแสดงรายละเอียดของเอกสาร HTML สำหรับทดสอบการทำงานของเอกสาร XSL



ภาพที่ 3.3 ภาพผลการเรียกดูเอกสาร HTML ที่ใช้ทดสอบการทำงานของเอกสาร XSL บนเว็บเบราเซอร์

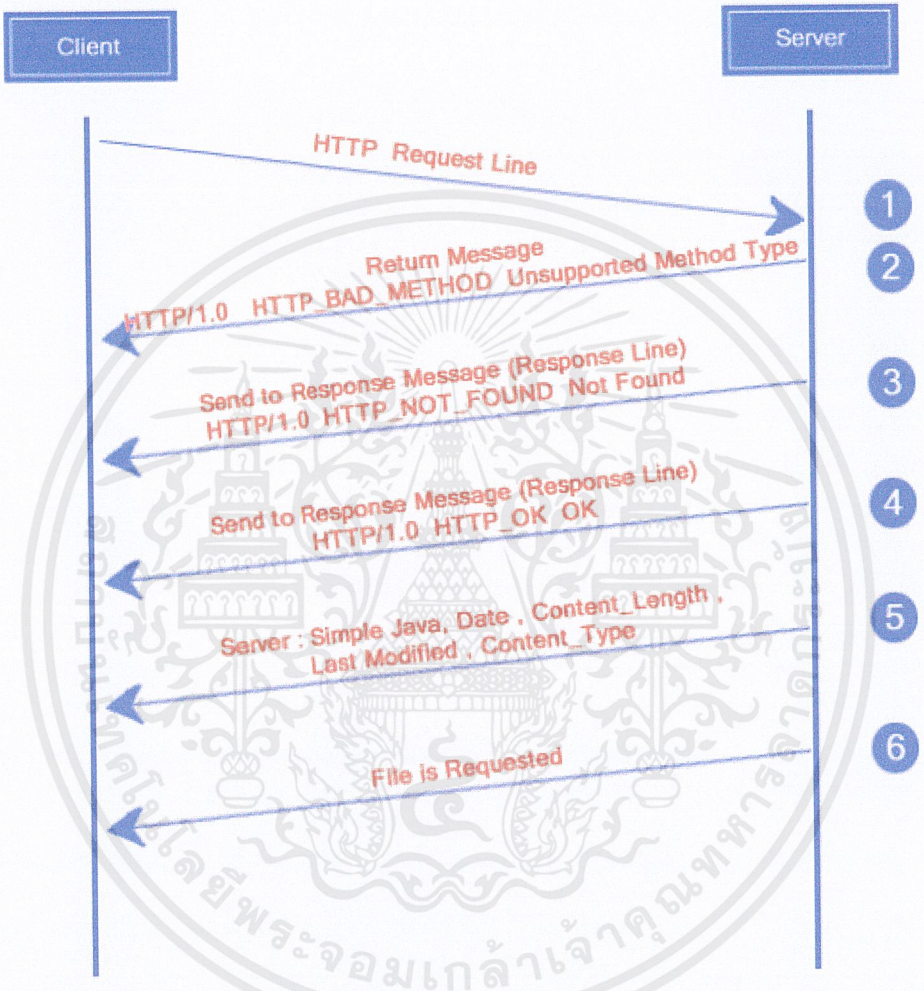



```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE wml PUBLIC "-//WAPFORM//DTD WML 1.1//EN" "http://www.wapforum.org/DTD/wml_1.1.xml">
<wml><card title="The XSL test page."><p>
<b>This page is under construction.</b><br/>
Show picture : <img src="./img/logo.gif" alt="logo" height="10" width="10"/><br/>
<b>Goodbye.</b>
</p></card></wml>
```

ภาพที่ 3.4 ภาพแสดงเอกสาร WML ที่ได้จากการกำหนดของเอกสาร XSL และผลการเรียกดูเอกสาร

3.3 ขั้นตอนการออกแบบและสร้างเว็บเซิร์ฟเวอร์ที่ทำงานภายใต้โพรโตคอล HTTP

การสร้างเว็บเซิร์ฟเวอร์เพื่อให้บริการเอกสารรูปแบบรูปแบบต่างๆ โดยสามารถรับการร้องขอจากไคลเอนท์ที่ร้องขอเอกสารเข้าและตอบสนองเอกสารที่ร้องขอกลับไปยังไคลเอนท์โดยมีขั้นตอนการทำงาน ดังภาพที่ 3.5



ภาพที่ 3.5 ภาพแสดงการทำงานของเมตธอดต่าง ๆ ของเว็บเซิร์ฟเวอร์เมื่อได้รับการร้องขอจากไคลเอนท์

จากรูปการทำงานจะเป็นดังขั้นตอนนี้

1. หลังจากมีการร้องขอส่งมายังเว็บเซิร์ฟเวอร์ เมตธอด handleClient โดยเมตธอดจะทำงานดังนี้

is = new BufferedInputStream (s.getInputStream())

ps = new PrintStream (s.getOutputStream())

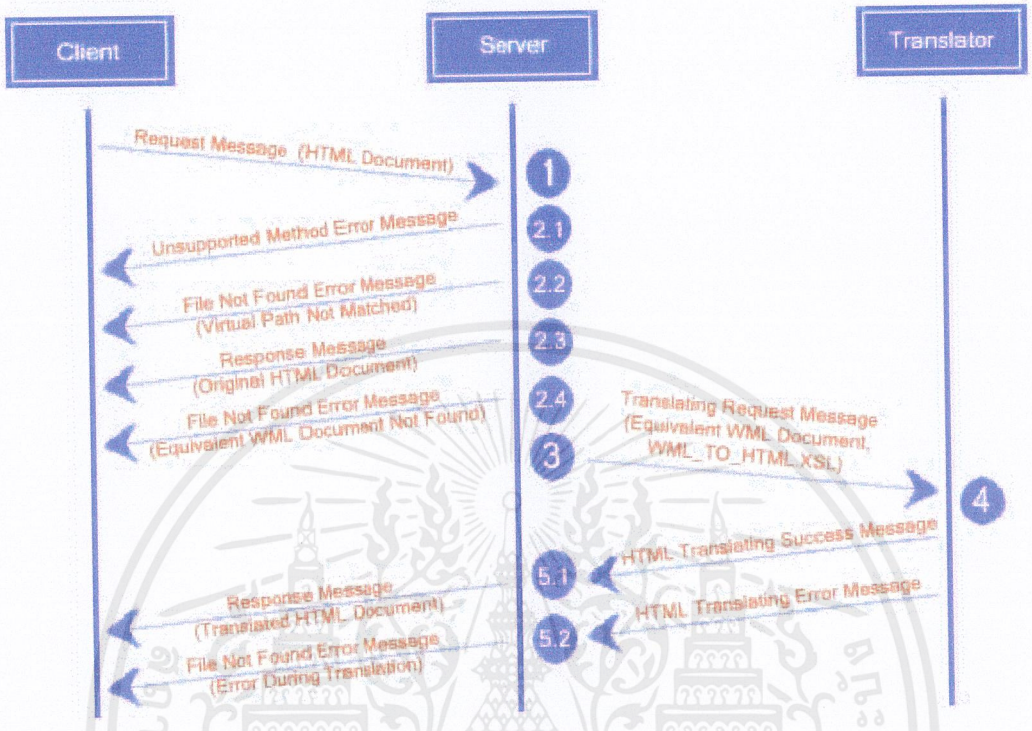
ตรวจข้อมูลในบัฟเฟอร์ว่าเป็นเมตธอด GET/HEAD หรือไม่

2. ถ้าไม่ใช่ จะส่งเมสเสจกลับไปทางอ็อบเจ็คท์ PrintStream ซึ่งจะเป็นการส่งเมสเสจกลับไปยังไคลเอนท์ว่าเว็บเซิร์ฟเวอร์ไม่รองรับเมตธอดที่ไคลเอนท์ร้องขอ

ถ้าใช่ จะนำข้อมูลจากบัฟเฟอร์ที่เป็นชื่อไฟล์ที่ต้องการมาเก็บไว้ในสตริงชื่อ fname แล้วใส่เครื่องหมาย / ซึ่งเป็นตัวคั่นชื่อไฟล์ (File Separator) ของ Virtual Path ให้ถูกต้องสำหรับแต่ละระบบปฏิบัติการ แล้วตัดตัวคั่นชื่อไฟล์ที่ตำแหน่งที่ 0 ในสตริงทิ้ง (ถ้ามี) หลังจากนั้นจะสร้างอ็อบเจ็คท์ไฟล์ขึ้น โดยให้ไฟล์ที่ชี้ไปเป็นตำแหน่งของ Physical Path ซึ่งเกิดจากการนำตัวแปรที่เก็บตำแหน่ง Root ปัจจุบันของเว็บเซิร์ฟเวอร์มาทำการ Concat กับ fname ด้วย Constructor ดังนี้ File targ = new File (WebServer.root, fname); และใช้เมตธอด isDirectory เพื่อตรวจสอบว่าเป็นไดเร็กทอรีหรือไม่ ซึ่งถ้าใช่ ก็จะเปลี่ยนตำแหน่งของ targ ไปที่ไดเร็กทอรีนั้น แล้ว Concat กับ index.html หลังจากนั้นจะทำการเรียกเมตธอด printHeaders เพื่อตรวจสอบว่ามีไฟล์นั้นอยู่จริงหรือไม่

3. ถ้าไม่มี จะส่งเมสเสจว่า HTTP/1.0 HTTP_NOT_FOUND กลับไปยังไคลเอนท์
4. ถ้ามี จะส่งเมสเสจว่า HTTP/1.0 HTTP_OK OK บันทึกข้อความลงไฟล์ Log
5. ส่งเมสเสจตอบสนองกลับไปยังไคลเอนท์
6. เรียกเมตธอด sendFile ทำงาน เพื่อส่งข้อมูลของไฟล์นั้นกลับไปยังไคลเอนท์

ภาพด้านล่างเป็นไดอะแกรมแสดงการทำงานของเมตธอดต่าง ๆ ของเว็บเซิร์ฟเวอร์เมื่อได้รับการร้องขอเอกสารแบบ HTML จากไคลเอนท์



ภาพที่ 3.6 ภาพแสดงการทำงานของเมตธอดต่าง ๆ ของเว็บเซิร์ฟเวอร์เมื่อได้รับการร้องขอเอกสารแบบ HTML

จากภาพ การทำงานจะเป็นดังขั้นตอนนี้

1. เมื่อไคลเอนท์ส่งคำร้องขอมายังเว็บเซิร์ฟเวอร์ โปรแกรมก็จะทำการสร้าง Thread ขึ้นมา 1 Thread เพื่อรองรับการร้องขอนั้น ๆ ซึ่งการใช้ Thread ก็เพื่อให้สามารถทำงานตอบสนองการร้องขอได้หลายครั้งพร้อมกัน

2. จากนั้นโปรแกรมจะทำการประมวลผลการร้องขอ ดังนี้

- ตรวจสอบว่าการร้องขอที่เข้ามาเป็นเมตธอดอะไร ซึ่งถ้าเว็บเซิร์ฟเวอร์ไม่รองรับการทำงานกับเมตธอดที่ร้องขอเข้ามา ก็จะตอบกลับไปยังไคลเอนท์ด้วยข้อความแบบ Unsupport Method Error

- ตรวจสอบว่าคำร้องขอที่เข้ามานั้นมี Virtual Path ที่ตรงกับ Physical Path ใด ถ้าไม่ตรง ก็ตอบกลับไปยัง ไคลเอนท์ด้วยข้อความแบบ File Not Found Error (Virtual Path Not Matched)

- โปรแกรมจะทำการตรวจสอบไฟล์ที่ร้องขอนั้น ว่ามีอยู่ใน Physical Path หรือไม่ ถ้ามีก็จะทำการนำ Original HTML Document ที่มีและ HTTP Code 200 OK มาใส่ในส่วนข้อความของเมสเสจในการตอบสนอง ซึ่งหมายถึงการดำเนินงานตามคำร้องขอสำเร็จ

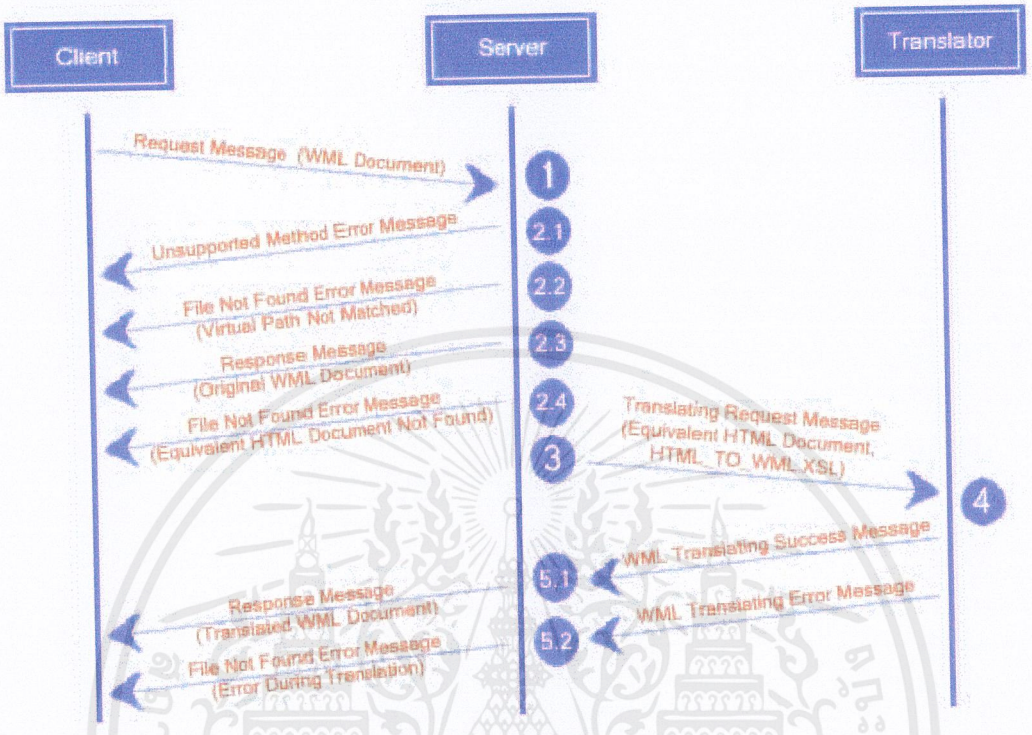
- ในกรณีที่ไม่มีเอกสาร HTML ที่ร้องขอ โปรแกรมจะตรวจสอบไฟล์เอกสารที่มีชื่อเดียวกับเอกสารที่ถูกร้องขอ แต่เป็นเอกสารประเภท WML ซึ่งถ้าพบว่าไม่มีเอกสารนั้นอยู่ใน Physical Path ก็จะตอบกลับไปยังไคลเอนท์ด้วยเมสเสจแบบ File Not Found Error ซึ่งมีรหัสเป็น 404 เนื่องจากไม่มี Equivalent WML Document ทำให้ไม่สามารถนำเอกสารไปแปลงให้เป็นรูปแบบเอกสารที่ร้องขอได้

3. โปรแกรมจะทำการส่งไฟล์เอกสาร WML พร้อมกับไฟล์ Stylesheet ที่ชื่อ WML_TO_HTML.XSL ไปให้กับตัว Translator เพื่อทำการแปลงเอกสารที่ได้นี้ให้เป็นไฟล์เอกสาร HTML ที่ต้องการ โดยการส่งเมสเสจตามภาพว่า Translating Request Message (Equivalent WML Document, WML_TO_HTML.XSL)

4. จากนั้นจะทำการส่งเมสเสจกลับไปยังเซิร์ฟเวอร์ว่า ได้ทำการแปลงเอกสารสำเร็จแล้ว โดยส่งเมสเสจแบบ HTML Translating Success แต่ถ้าการแปลงเอกสารไม่สำเร็จก็จะส่งเมสเสจไปยังเซิร์ฟเวอร์แบบ HTML Translating Error

5. เซิร์ฟเวอร์จะทำการตรวจสอบว่าการแปลงเอกสารสำเร็จหรือไม่ ถ้าสำเร็จ ก็จะนำ Translated HTML Document ที่แปลงได้ใส่ลงไปในส่วนข้อความพร้อมกับ HTTP Code 200 OK แต่ถ้าไม่สำเร็จ ก็จะส่งเมสเสจแบบ File Not Found Error Message ซึ่งมีรหัสเป็น 404 กลับไปยังไคลเอนท์ เนื่องจากการแปลงไม่สำเร็จ

ภาพต่อไปนี้คือไดอะแกรมแสดงการทำงานของเมตธอดต่าง ๆ ของเว็บเซิร์ฟเวอร์เมื่อได้รับการร้องขอเอกสารแบบ WML จากไคลเอนท์



ภาพที่ 3.7 ภาพแสดงการทำงานของเมตธอดต่าง ๆ ของเว็บเซิร์ฟเวอร์เมื่อได้รับการร้องขอเอกสารแบบ WML

จากภาพ การทำงานจะเป็นดังขั้นตอนต่อไปนี้

1. เมื่อไคลเอนท์ส่งคำร้องขอมายังเว็บเซิร์ฟเวอร์ โปรแกรมก็จะทำการสร้าง Thread ขึ้นมา 1 Thread เพื่อรองรับการร้องขอนั้น ๆ ซึ่งการใช้ Thread ก็เพื่อให้สามารถทำงานตอบสนองการร้องขอได้หลายครั้งพร้อมกัน

2. จากนั้นโปรแกรมจะทำการประมวลผลการร้องขอ ดังนี้

- ตรวจสอบว่าการร้องขอที่เข้ามาเป็นเมตธอดอะไร ซึ่งถ้าเว็บเซิร์ฟเวอร์ไม่รองรับการทำงานกับเมตธอดที่ร้องขอเข้ามา ก็จะตอบกลับไปยังไคลเอนท์ด้วยข้อความแบบ Unsupport Method Error

- ตรวจสอบว่าคำร้องขอที่เข้ามานั้นมี Virtual Path ที่ตรงกับ Physical Path ใด ถ้าไม่ตรง ก็ตอบกลับไปยัง ไคลเอนท์ด้วยข้อความแบบ File Not Found Error (Virtual Path Not Matched)

- โปรแกรมจะทำการตรวจสอบไฟล์ที่ร้องขอนั้น ว่ามีอยู่ใน Physical Path หรือไม่ ถ้ามีก็จะทำการนำ Original WML Document ที่มีและ HTTP Code 200 OK มาใส่ในส่วนข้อความของเมสเสจในการตอบสนอง ซึ่งหมายถึงการดำเนินงานตามคำร้องขอสำเร็จ

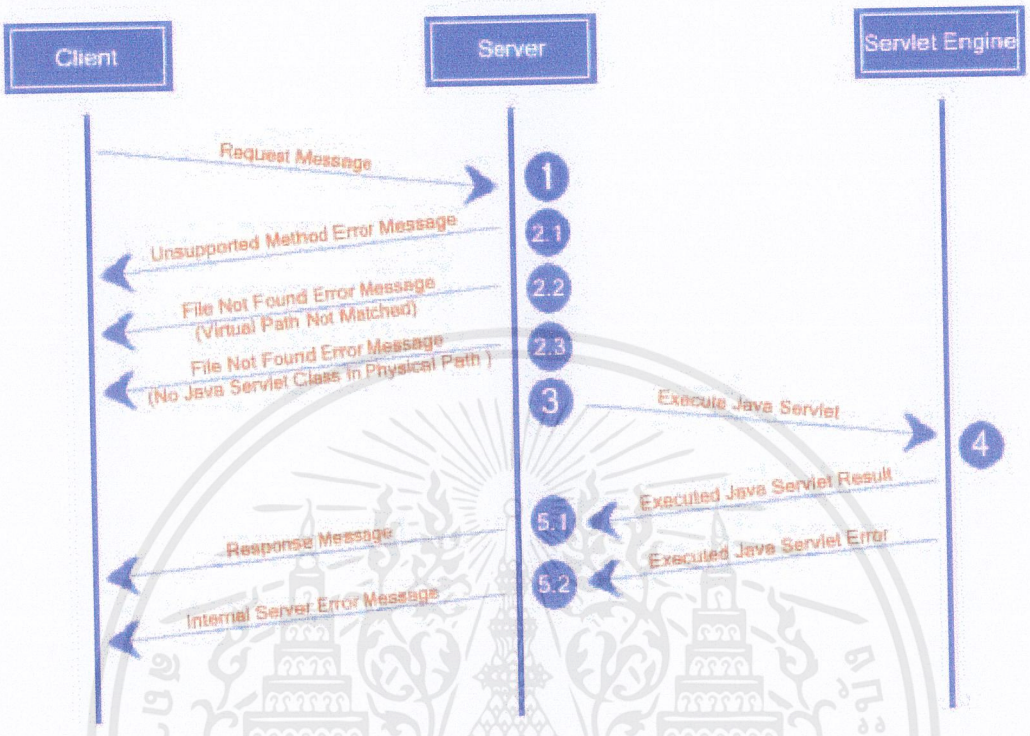
- ในกรณีที่ไม่มีเอกสาร WML ที่ร้องขอ โปรแกรมจะตรวจสอบไฟล์เอกสารที่มีชื่อเดียวกับเอกสารที่ถูกร้องขอ แต่เป็นเอกสารประเภท HTML ถ้าพบว่าไม่มีเอกสารนั้นอยู่ใน Physical Path ก็จะตอบกลับไปยังไคลเอนท์ด้วยเมสเสจแบบ File Not Found Error ซึ่งมีรหัสเป็น 404 เนื่องจากไม่มี Equivalent HTML Document ทำให้ไม่สามารถนำเอกสารไปแปลงให้เป็นรูปแบบเอกสารที่ร้องขอได้

3. โปรแกรมจะทำการส่งไฟล์เอกสาร WML พร้อมกับไฟล์ Stylesheet ที่ชื่อ HTML_TO_WML.XSL ไปให้กับตัว Translator เพื่อทำการแปลงเอกสารที่ได้นี้ให้เป็นไฟล์เอกสาร WML ที่ต้องการ โดยการส่งเมสเสจตามภาพว่า Translating Request Message (Equivalent HTML Document, HTML_TO_WML.XSL)

4. จากนั้นจะทำการส่งเมสเสจกลับไปยังเซิร์ฟเวอร์ว่า ได้ทำการแปลงเอกสารสำเร็จแล้ว โดยส่งเมสเสจแบบ WML Translating Success แต่ถ้าการแปลงเอกสารไม่สำเร็จก็จะส่งเมสเสจไปยังเซิร์ฟเวอร์แบบ WML Translating Error

5. เซิร์ฟเวอร์จะทำการตรวจสอบว่าการแปลงเอกสารสำเร็จหรือไม่ ถ้าสำเร็จ ก็จะนำ Translated WML Document ที่แปลงได้ใส่ลงไปในส่วนข้อความพร้อมกับ HTTP Code 200 OK แต่ถ้าไม่สำเร็จ ก็จะส่งเมสเสจแบบ File Not Found Error Message ซึ่งมีรหัสเป็น 404 กลับไปยังไคลเอนท์ เนื่องจากการแปลงไม่สำเร็จ

ภาพต่อไปเป็นไดอะแกรมแสดงการทำงานของเมตธอดต่าง ๆ ของเว็บเซิร์ฟเวอร์ เมื่อได้รับการร้องขอให้ประมวลผลไฟล์ Java Servlet จากไคลเอนท์



ภาพที่ 3.8 ภาพแสดงการทำงานของเมตธอดต่าง ๆ ของเว็บเซิร์ฟเวอร์เมื่อได้รับการร้องขอไฟล์ Java Servlet

จากภาพ การทำงานจะเป็นดังนี้

1. เมื่อไคลเอนท์ส่งคำร้องขอมายังเว็บเซิร์ฟเวอร์ โปรแกรมก็จะทำการสร้าง Thread ขึ้นมา 1 Thread เพื่อรองรับการร้องขอนั้น ๆ ซึ่งการใช้ Thread ก็เพื่อให้สามารถทำงานได้หลาย Request พร้อม ๆ กัน

2. จากนั้นโปรแกรมจะทำการประมวลผลคำร้องขอดังนี้

- ตรวจสอบว่าคำร้องขอที่เข้ามาว่าเป็นเมตธอดอะไร ซึ่งถ้าเซิร์ฟเวอร์ไม่รองรับการทำงานกับเมตธอดนั้น ก็จะตอบกลับไปยังไคลเอนท์ด้วยเมสเสจแบบ Unsupport Method Error

- ตรวจสอบว่าคำร้องขอที่เข้ามานั้นมี Virtual Path ที่ตรงกับ Physical Path ใด ถ้าไม่ตรง ก็ตอบกลับไปยังไคลเอนท์ด้วยเมสเสจแบบ File Not Found Error (Virtual Path Not Matched)

- โปรแกรมจะทำการตรวจสอบไฟล์ Java Servlet ที่ร้องขอนั้นว่ามีอยู่ใน Physical Path หรือไม่ ถ้าไม่มี โปรแกรมจะตอบกลับไปยังไคลเอนท์ด้วยเมสเสจแบบ File Not Found Error

3. โปรแกรมจะทำการส่งไฟล์ Java Servlet นั้นไปประมวลผลด้วย Servlet Engine

4. Servlet Engine จะทำการประมวลผลคลาส Java Servlet ที่เว็บเซิร์ฟเวอร์ส่งมา พร้อมส่งผลกลับมายังเว็บเซิร์ฟเวอร์ ในกรณีที่การประมวลผลเกิดความผิดพลาดขึ้น Servlet ก็จะส่งเมสเสจกลับมายังเว็บเซิร์ฟเวอร์ว่าการประมวลผลไฟล์ Java Servlet เกิดปัญหาขึ้น

5. เว็บเซิร์ฟเวอร์จะนำผลการประมวลผลไฟล์ Java Servlet มาใส่ลงในเมสเสจในส่วนของข้อความในการตอบสนอง พร้อมกับ HTTP Code 200 OK ที่ส่วนเฮดเดอร์ ก่อนส่งกลับไปยังไคลเอนท์ แต่เกิดข้อผิดพลาดในการประมวลผล โปรแกรมจะส่งเมสเสจที่ระบุว่า HTTP Code 500 Internal Server Error กลับไปยังไคลเอนท์

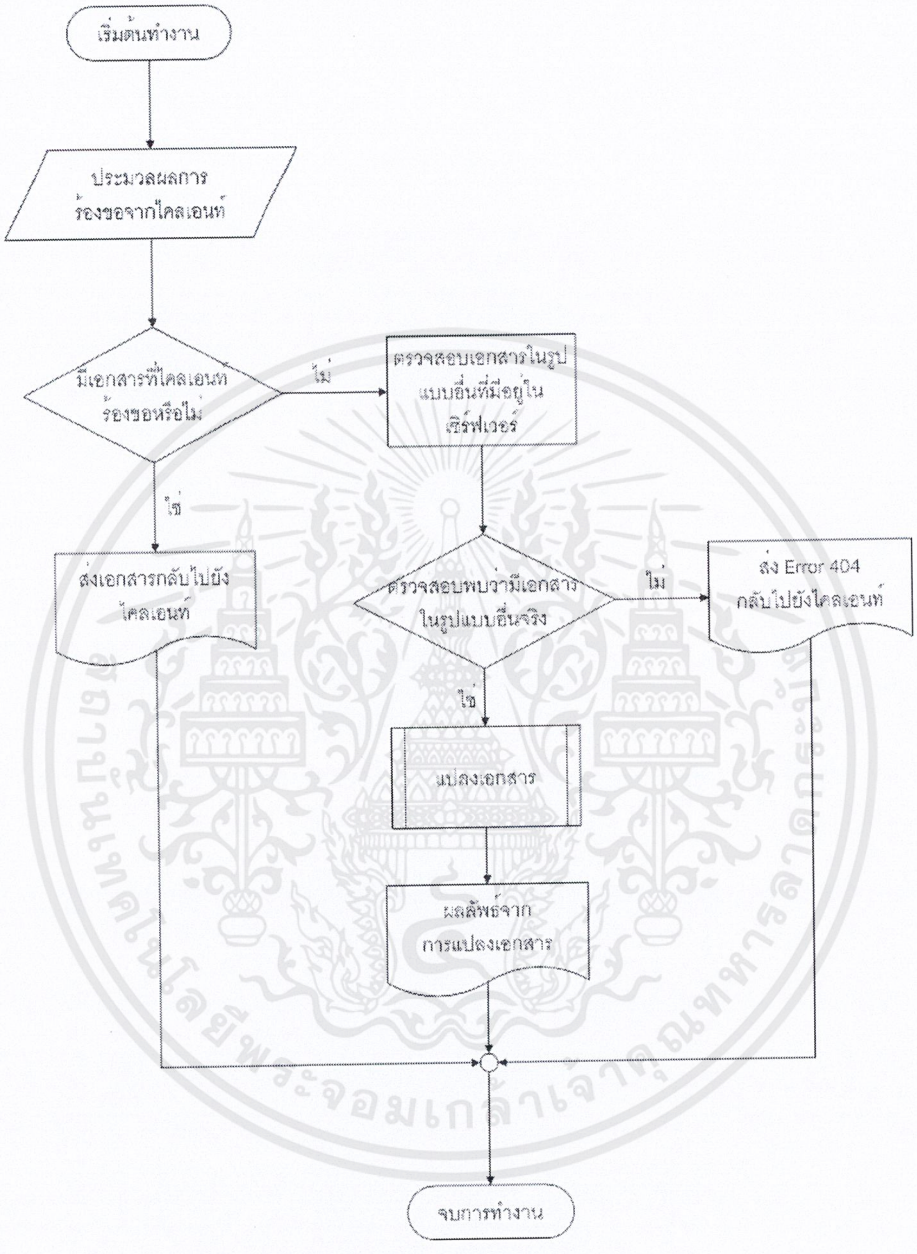
3.4 ขั้นตอนการออกแบบสร้างตัวแปลงภาษารูปแบบ XSL ที่ทำงานบนเว็บเซิร์ฟเวอร์

เว็บเซิร์ฟเวอร์นอกจากจะให้บริการเอกสารในรูปแบบต่าง ๆ แล้ว ยังสามารถทำการแปลงเอกสารในรูปแบบหนึ่งให้เป็นเอกสารในอีกรูปแบบหนึ่งได้ ในกรณีรูปแบบเอกสารที่ถูกร้องขอนั้นไม่มีอยู่ในเว็บเซิร์ฟเวอร์ เช่น ในกรณีที่มีการร้องขอเอกสารในรูปแบบ WML จากไคลเอนท์ แต่ในเว็บเซิร์ฟเวอร์นั้นมีไม่มีเอกสารในรูปแบบ WML ที่ร้องขออยู่ มีเพียงแต่เอกสารที่ถูกร้องขอในรูปแบบ HTML อยู่บนเว็บเซิร์ฟเวอร์นั้น ในกรณีนี้ตัวแปลงเอกสาร XSL จะทำการแปลงเอกสารให้อยู่ในรูปแบบที่ตรงกับการร้องขอ ก่อนส่งกลับไปยังไคลเอนท์ โดยตัวแปลงภาษารูปแบบ XSL มีข้อมูลเข้าเป็นเอกสารที่ต้องการซึ่งมีอยู่บนเว็บเซิร์ฟเวอร์ที่ต้องทำการแปลง และผลลัพธ์ที่ได้จากตัวแปลงภาษารูปแบบ XSL คือ เอกสารที่ผ่านการดำเนินการแปลงแล้วให้อยู่ในรูปแบบที่ไคลเอนท์ร้องขอดังแสดงในภาพที่ 3.7

ภาพที่ 3.9 ภาพแสดงการรับเอกสารเข้าและเอกสารออกของตัวแปลงภาษารูปแบบ XSL

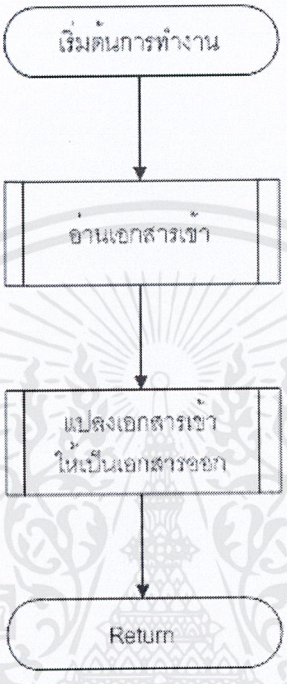
เมื่อมีการร้องขอเอกสารจากไคลเอนท์ เว็บเซิร์ฟเวอร์จะตรวจสอบว่ามีเอกสารนั้นอยู่บนเว็บเซิร์ฟเวอร์หรือไม่ ถ้ามี เว็บเซิร์ฟเวอร์จะทำการส่งเอกสารที่ร้องขอกลับไปยังไคลเอนท์ ถ้าไม่มี เว็บเซิร์ฟเวอร์จะดำเนินการดังนี้

- ตรวจสอบว่าในเว็บเซิร์ฟเวอร์มีเอกสารในรูปแบบอื่นที่สามารถแปลงให้เป็นเอกสารในรูปแบบที่ร้องขอมาได้หรือไม่
- ถ้าไม่มี จะทำการส่งเมสเสจแสดงความผิดพลาดด้วยรหัส 404 File Not Found กลับไปยังไคลเอนท์
- ถ้ามี จะทำการส่งเอกสารในรูปแบบอื่นนั้น ไปเป็นเอกสารเข้าให้กับตัวแปลงเอกสารรูปแบบ XSL
- หลังจากตัวแปลงเอกสารรูปแบบ XSL แปลงเอกสารเข้าให้เป็นเอกสารออกในรูปแบบที่ไคลเอนท์ร้องขอ แล้วเว็บเซิร์ฟเวอร์จะส่งเอกสารออกนั้นกลับไปยังไคลเอนท์

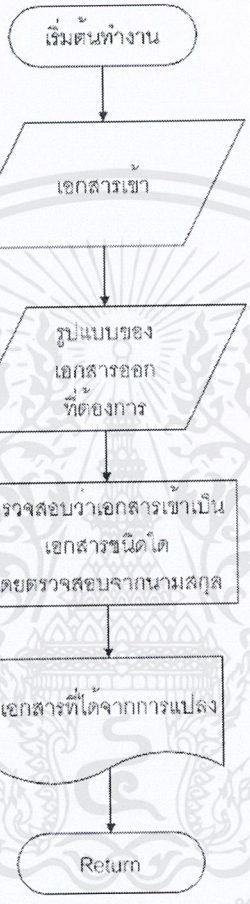


ภาพที่ 3.10 ภาพ Flow Chart แสดงขั้นตอนการทำงานของเว็บเซิร์ฟเวอร์

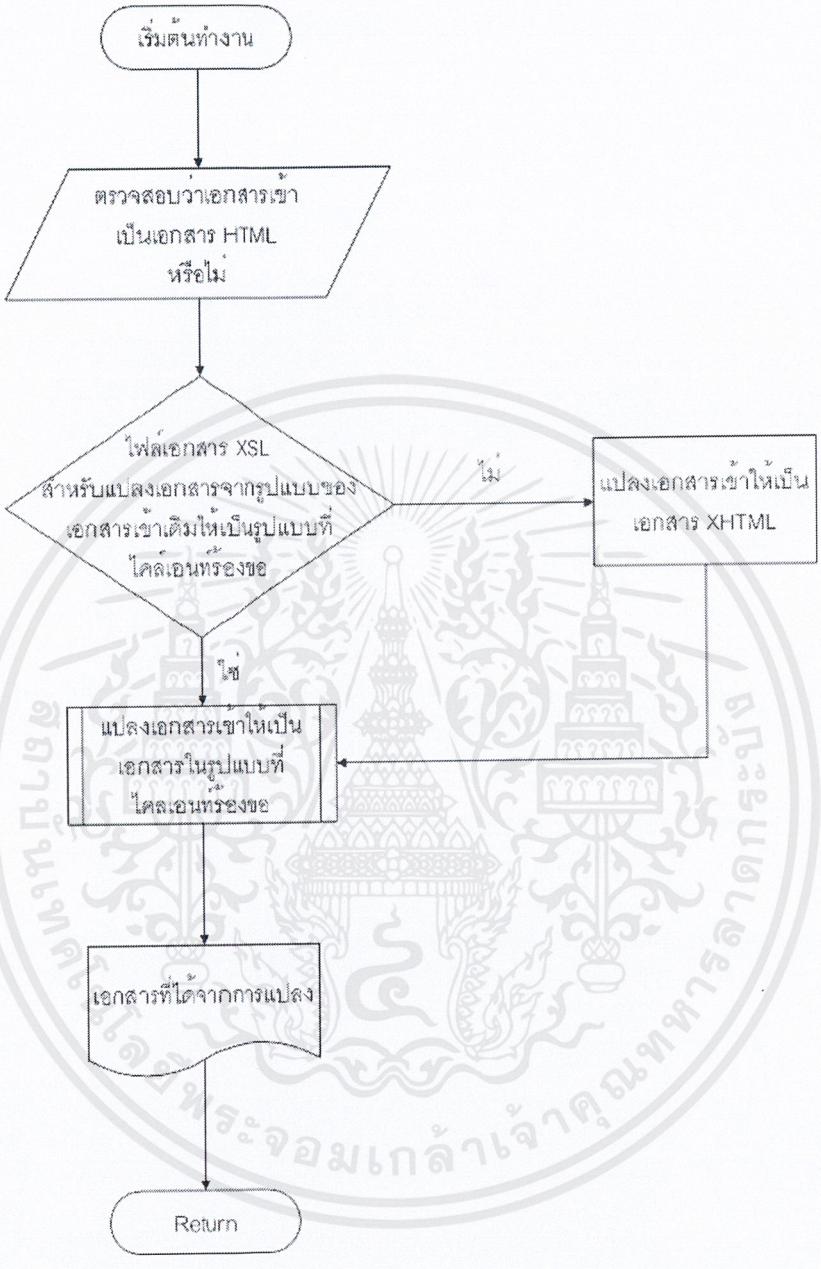
ในการดำเนินการของตัวแปลงภาษารูปแบบ XSL อาศัยเอกสารเข้าร่วมกับเอกสาร XSL สำหรับกำหนดกฎในการแปลงเอกสาร ซึ่งอยู่ภายในตัวแปลงภาษารูปแบบ XSL (เป็นเอกสารที่อยู่ภายใน ซึ่งผู้ดูแลเว็บเซิร์ฟเวอร์สามารถติดตั้งไฟล์เอกสาร XSL นี้ลงไปได้) ซึ่งการดำเนินการแปลงเอกสารมีขั้นตอนดังแสดงในภาพที่ 3.11



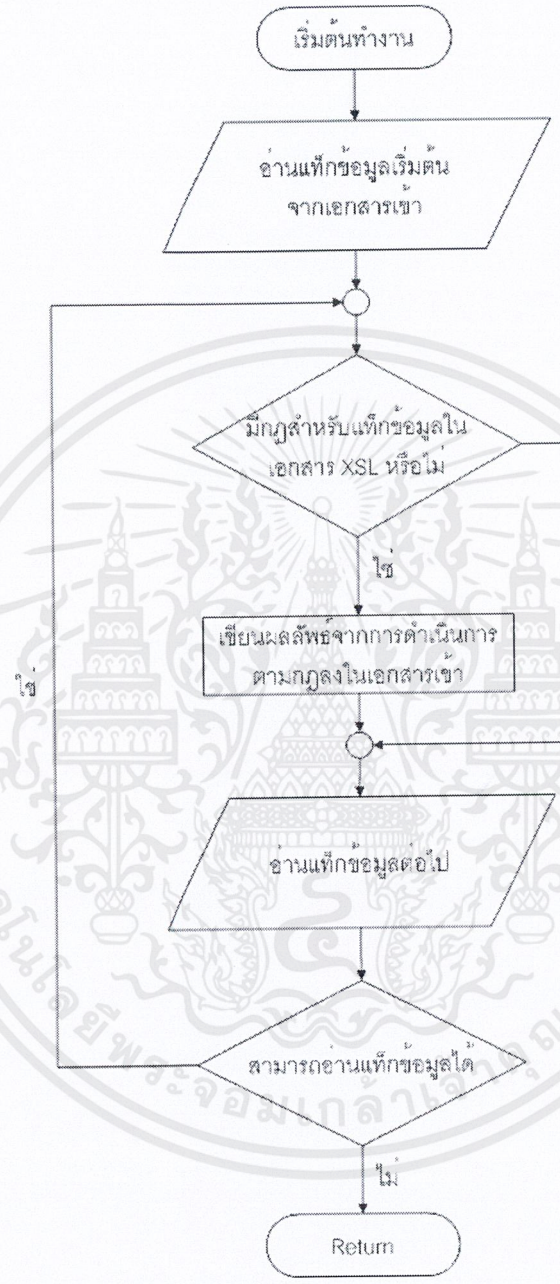
ภาพที่ 3.11 ภาพ Flow Chart แสดงการทำงานของฟังก์ชันย่อยการแปลงเอกสาร



ภาพที่ 3.12 ภาพ Flow Chart แสดงการทำงานของฟังก์ชันย่อยการอ่านเอกสารเข้า



ภาพที่ 3.13 ภาพ Flow Chart แสดงการทำงานของฟังก์ชันย่อยการแปลงเอกสารเข้าให้เป็นเอกสารออก



ภาพที่ 3.14 ภาพ Flow Chart แสดงการทำงานของฟังก์ชันย่อยการแปลงเอกสารเข้า

เพื่อให้เข้าใจในขั้นตอนการแปลงเอกสารเข้าให้เป็นเอกสารออกมากขึ้น จึงขอยกตัวอย่างประกอบ แล้วแสดงการสร้างเอกสารออกตามขั้นตอนทีละขั้น โดยมีเอกสารเข้าดังภาพที่ 3.15 และมีเอกสาร XSL ดังภาพที่ 3.16

```
<html>                                                    ----------1
<body>                                                    ----------2
  Hello World! <b>this is me.</b><br/>                    ----------3
  <u>underline.</u>                                       ----------4
</body>                                                   ----------5
</html>                                                   ----------6
```

ภาพที่ 3.15 ภาพเอกสารเข้า HTML สำหรับแสดงขั้นตอนการแปลงเอกสารเข้าให้เป็นเอกสารออก

```
<xsl:stylesheet  xmlns:xsl = "http://www.w3.org/1999/XSL/Transform" >   --------1
    <xsl:template match="/">                                            --------2
     <wml>                                                              --------3
       <card title =”Untitle.">                                         --------4
         <xsl:apply-templates select = ".//body"/>                      --------5
       </card>                                                          --------6
     </wml>                                                             --------7
    </xsl:template>                                                     --------8
    <xsl:template match="body">                                         --------9
      <p><xsl:apply-templates /></p>                                    -------10
    </xsl:template>                                                     -------11
    <xsl:template match = "br"> <br/>                                   -------12
    </xsl:template>                                                     -------13
    <xsl:template match = "b|strong">                                   -------14
      <b><xsl:apply-templates /></b>                                    -------15
    </xsl:template>                                                     -------16
    </xsl:stylesheet>                                                   -------17
```

ภาพที่ 3.16 ภาพเอกสาร XSL สำหรับแสดงขั้นตอนการแปลงเอกสารเข้าให้เป็นเอกสารออก

ขั้นตอนการแปลงเอกสารเข้าให้เป็นเอกสารออก เริ่มจากการอ่านแท็กข้อมูลเริ่มต้นในเอกสารเข้า จากตัวอย่างแท็ก ข้อมูลเริ่มต้นของเอกสารเข้าคือแท็ก <html> จากนั้นจึงทำค้นหากฎที่กำหนดไว้ในเอกสาร XSL พบว่ามีกฎสำหรับการปรับเปลี่ยนโครงสร้างตรงกับบรรทัดที่ 2 ในภาพที่ 3.15 จึงดำเนินการตามกฎที่กำหนดไว้ คือ เขียนข้อมูล <wml> <card title ="Untitle."> ลงเอกสารออก ทำการอ่านแท็กข้อมูลต่อไป ได้แท็ก <body> ทำการค้นหากฎและดำเนินการตามกฎที่เขียนในเอกสาร XSL เมื่อดำเนินการกับแท็ก <body> เสร็จ จึงดำเนินการเขียนข้อมูล </wml> ลงในเอกสารออก

เมื่อทำการค้นหากฎสำหรับการแปลงเอกสารในเอกสาร XSL พบว่า มีกฎการแปลงเอกสารสำหรับแท็ก <body> ในบรรทัดที่ 9 ในภาพที่ 3.15 ทำการเขียนข้อมูล และทำการอ่านแท็กข้อมูลต่อไป แล้วดำเนินการค้นหาและทำตามกฎไปจนไม่มีแท็กข้อมูล เมื่อดำเนินการเสร็จแล้วจึงทำการเขียนข้อมูล ลงในเอกสารออก

ผลจากการดำเนินการ จะได้เอกสารออก ดังภาพที่ 3.17

```
<wml>
  <card title ="Untitle.">
    <p>
      Hello World! <b>this is me.</b><br/>
    </p>
  </card>
</wml>
```

ภาพที่ 3.17 ภาพเอกสารออก WML สำหรับแสดงขั้นตอนการแปลงเอกสารเข้าให้เป็นเอกสารออก

บทที่ 4
ผลการดำเนินการ

จากการศึกษาถึงโปรแกรมเกตเวย์สำหรับแปลงรูปแบบเอกสาร XML บนเว็บเซิร์ฟเวอร์ ทำให้ผู้จัดทำได้ทราบถึงผลที่ได้จากการดำเนินการของตัวแปลงเอกสารรูปแบบ XSL ใน 4 ลักษณะ ดังนี้

4.1 การตอบสนองการร้องขอเอกสารจากเครื่องไคลเอนท์

4.2 ผลจากการแปลงเอกสาร HTML บนเว็บเซิร์ฟเวอร์ให้เป็นเอกสาร WML

4.3 ผลจากการแปลงเอกสาร WML บนเว็บเซิร์ฟเวอร์ให้เป็นเอกสาร HTML

4.4 ผลการประมวลผลไฟล์ Java Servlet ที่เครื่องเซิร์ฟเวอร์

4.1 การตอบสนองการร้องขอเอกสารจากเครื่องไคลเอนท์

4.1.1 การร้องขอเอกสาร HTML

เว็บเซิร์ฟเวอร์จะตอบสนองโดยการค้นหาว่ามีเอกสาร HTML นั้น ๆ อยู่หรือไม่ ถ้ามี เว็บเซิร์ฟเวอร์ก็จะทำการส่งเอกสารกลับไปยังไคลเอ็นท์ทันที แต่ถ้าไม่มี เว็บเซิร์ฟเวอร์ก็จะทำการค้นหาเอกสาร WML ที่มีชื่อตรงตามที่ไคลเอนท์ร้องขอมา ซึ่งถ้าหาพบ เว็บเซิร์ฟเวอร์ก็จะทำการแปลงเอกสาร WML นั้นให้เป็นเอกสาร HTML แล้วจึงส่งเอกสารที่ได้จากการแปลงกลับไปยังไคลเอนท์ แต่ถ้ายังหาไม่พบอีก เว็บเซิร์ฟเวอร์ก็จะส่งเอกสาร HTML ที่มีเนื้อหาอธิบายถึงความผิดพลาดที่เกิดขึ้นจากการที่เว็บเซิร์ฟเวอร์ไม่สามารถหาเอกสารที่ไคลเอนท์ร้องขอมาได้

4.1.2 การร้องขอเอกสาร WML

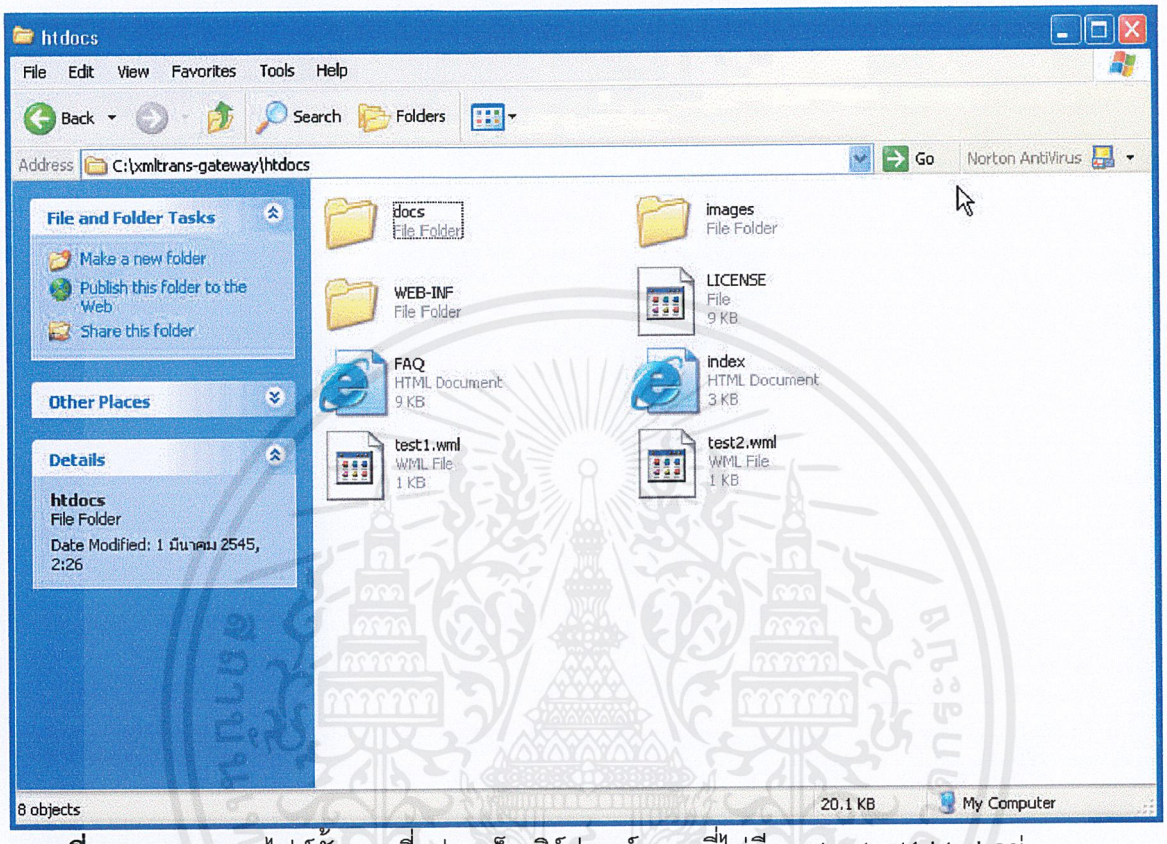
เว็บเซิร์ฟเวอร์จะตอบสนองโดยการค้นหาว่ามีเอกสาร WML นั้น ๆ อยู่หรือไม่ ถ้ามี เว็บเซิร์ฟเวอร์ก็จะทำการส่งเอกสารกลับไปยังไคลเอ็นท์ทันที แต่ถ้าไม่มี เว็บเซิร์ฟเวอร์ก็จะทำการค้นหาเอกสาร HTML ที่มีชื่อตรงตามที่ไคลเอนท์ร้องขอมา ซึ่งถ้าหาพบ เว็บเซิร์ฟเวอร์ก็จะทำการแปลงเอกสาร HTML นั้นให้เป็นเอกสาร WML แล้วจึงส่งเอกสารที่ได้จากการแปลงกลับไปยังไคลเอนท์ แต่ถ้ายังหาไม่พบอีก เว็บเซิร์ฟเวอร์ก็จะส่งเอกสาร WML ที่มีเนื้อหาอธิบายถึงความผิดพลาดที่เกิดขึ้นจากการที่เว็บเซิร์ฟเวอร์ไม่สามารถหาเอกสารที่ไคลเอนท์ร้องขอมาได้

4.1.3 การประมวลผลไฟล์ Java Servlet

เว็บเซิร์ฟเวอร์สามารถตอบสนองการร้องขอการประมวลผล Java Servlet ที่เครื่องเซิร์ฟเวอร์ โดยเว็บเซิร์ฟเวอร์จะต้องมีไฟล์คลาสของ Java Servlet อยู่ เมื่อผู้ใช้ร้องขอไฟล์คลาสนั้น ๆ เว็บเซิร์ฟเวอร์จะทำการค้นหาไฟล์คลาสที่ไคลเอนท์ร้องขอมา แล้วทำการประมวลผลไฟล์คลาสนั้น ก่อนจะส่งผลที่ได้จากการประมวลผลกลับมายังไคลเอนท์ ทำให้โปรแกรมสามารถสร้างและตอบสนองเอกสารแบบ Dynamic บนเว็บเซิร์ฟเวอร์ได้

4.2 ผลจากการแปลงเอกสาร WML ให้เป็นเอกสาร HTML บนเว็บเซิร์ฟเวอร์

การทำงานเริ่มจาก การตรวจสอบไฟล์ทั้งหมดบนเว็บเซิร์ฟเวอร์ โดยที่สมมติว่าไคลเอนท์ทำการร้องขอเอกสาร HTML ชื่อ test1.html แต่จากรูปจะพบว่าไม่มีเอกสารชื่อ test1.html อยู่ ดังภาพ



ภาพที่ 4.1 ภาพแสดงไฟล์ทั้งหมดที่อยู่บนเว็บเซิร์ฟเวอร์ ขณะที่ไม่มีเอกสาร test1.html อยู่

แต่บนเว็บเซิร์ฟเวอร์มีเอกสาร WML ชื่อ test1.wml อยู่ ดังนั้นเว็บเซิร์ฟเวอร์จะทำการแปลงเอกสารนี้เป็นเอกสาร HTML ชื่อ test1.html แทน เนื่องจากเอกสาร WML ที่มีอยู่นี้ มีชื่อตรงกับเอกสารที่ไคลเอนท์ร้องขอมา เพียงแต่เป็นเอกสารต่างชนิดกัน

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN" "file:///C:/xmltrans-
        gateway/dtd/wml_1.1.xml">
<wml><card title="Untitled."><p>
<strong>Translation is Success !!</strong></p><p>
<b>test1.wml --> test1.html</b><br/>
</p></card></wml>
```

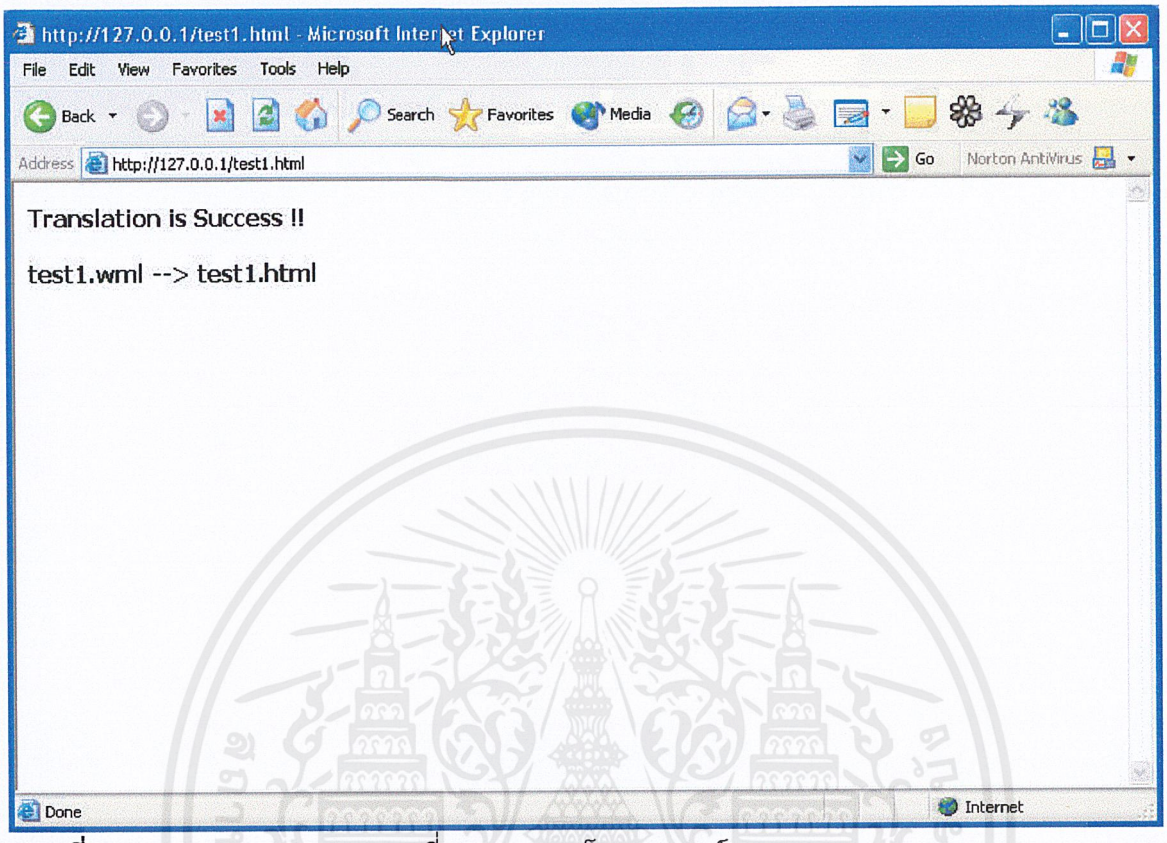
ภาพที่ 4.2 ภาพรายละเอียดเอกสาร test1.wml ที่จะนำไปใช้แปลงเป็นเอกสาร HTML

หลังจากที่ใช้ตัวแปลงเอกสารรูปแบบ XSL ของโปรแกรม จะได้ผลลัพธ์ที่เป็นเอกสาร HTML

```
<html>
        <head>
        <title></title>
        </head>
        <body>
                <p align="left"><strong>Translation is Success !!</strong></p>
                <p align="left"><b>test1.wml --&gt; test1.html</b><br /></p>
        </body>
</html>
```

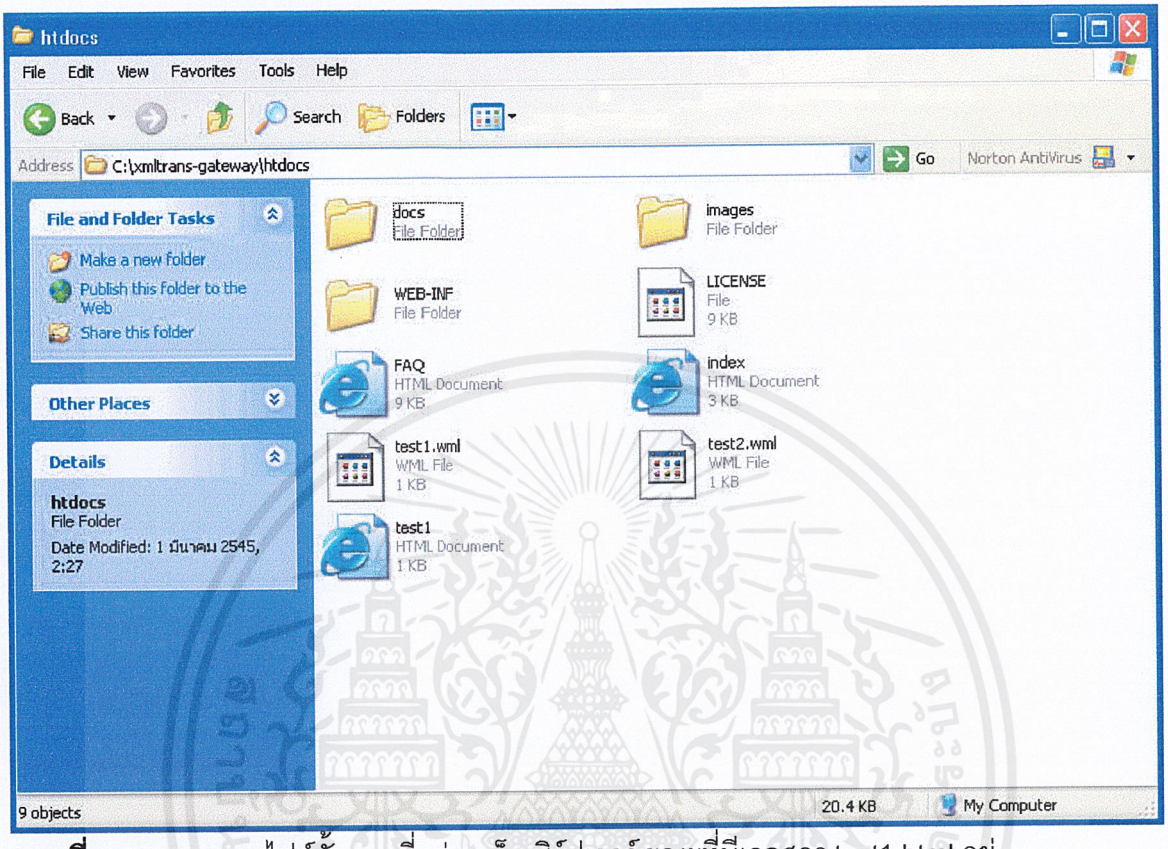
ภาพที่ 4.3 ภาพรายละเอียดเอกสาร test1.html ที่ได้จากการแปลงมาจากเอกสาร WML

เอกสาร HTML ที่ได้จากการแปลงนั้น สามารถแสดงผลลัพธ์ได้ดังภาพ



ภาพที่ 4.4 ภาพเอกสาร test1.html ที่แสดงผลบนเว็บเบราเซอร์

เมื่อตรวจสอบไฟล์ทั้งหมดบนเว็บเซิร์ฟเวอร์อีกครั้ง จะพบว่าจะมีเอกสาร HTML ชื่อ test1.html แล้ว ดังภาพ

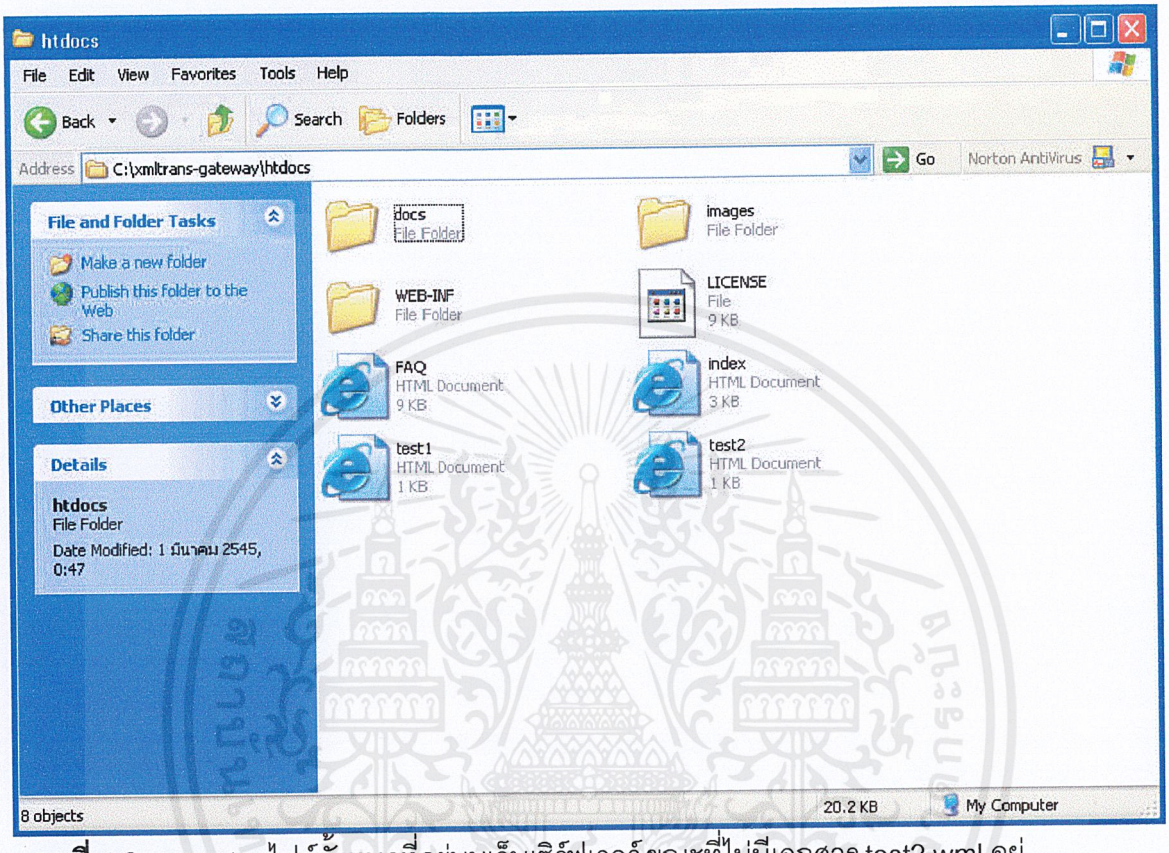


ภาพที่ 4.5 ภาพแสดงไฟล์ทั้งหมดที่อยู่บนเว็บเซิร์ฟเวอร์ ขณะที่มีเอกสาร test1.html อยู่

อนึ่ง เนื่องจากตัวอย่างการแปลงเอกสาร WML ให้เป็นเอกสาร HTML นั้น ทางผู้จัดทำยังไม่พบปัญหาเกี่ยวกับการแสดงผลใด ๆ ของเอกสาร HTML ที่ต่างไปจากเอกสาร WML ต้นแบบ

4.3 ผลจากการแปลงเอกสาร HTML ให้เป็นเอกสาร WML บนเว็บเซิร์ฟเวอร์

การทำงานเริ่มจาก การตรวจสอบไฟล์ทั้งหมดบนเว็บเซิร์ฟเวอร์ โดยที่สมมติว่าไคลเอนท์ทำการร้องขอเอกสาร WML ชื่อ test2.wml แต่จากรูปจะพบว่าไม่มีเอกสารชื่อ test2.wml อยู่ ดังภาพ



ภาพที่ 4.6 ภาพแสดงไฟล์ทั้งหมดที่อยู่บนเว็บเซิร์ฟเวอร์ ขณะที่ไม่มีเอกสาร test2.wml อยู่

แต่บนเว็บเซิร์ฟเวอร์มีเอกสาร HTML ชื่อ test2.html อยู่ ดังนั้นเว็บเซิร์ฟเวอร์จะทำการแปลงเอกสารนี้เป็นเอกสาร WML ชื่อ test2.wml แทน เนื่องจากเอกสาร HTML ที่มีอยู่นี้ มีชื่อตรงกับเอกสารที่ไคลเอนท์ร้องขอมา เพียงแต่เป็นเอกสารต่างชนิดกัน

```
<html>
<head>          <title>XSLTranslator.</title> </head>
<body>
<font size="5"><b>XML Gateway</b></font><br /> <i>Students</i><br />
    <table>
      <tr><td>Name</td><td>Lastname</td></tr>
      <tr><td>Wirat</td><td>Mahasalprasert</td></tr>
      <tr><td>Anirut</td><td>Worakitrungruang</td></tr>
      <tr><td>Aniwat</td><td>Arromratna</td></tr>
    </table>
  </body>
</html>
```

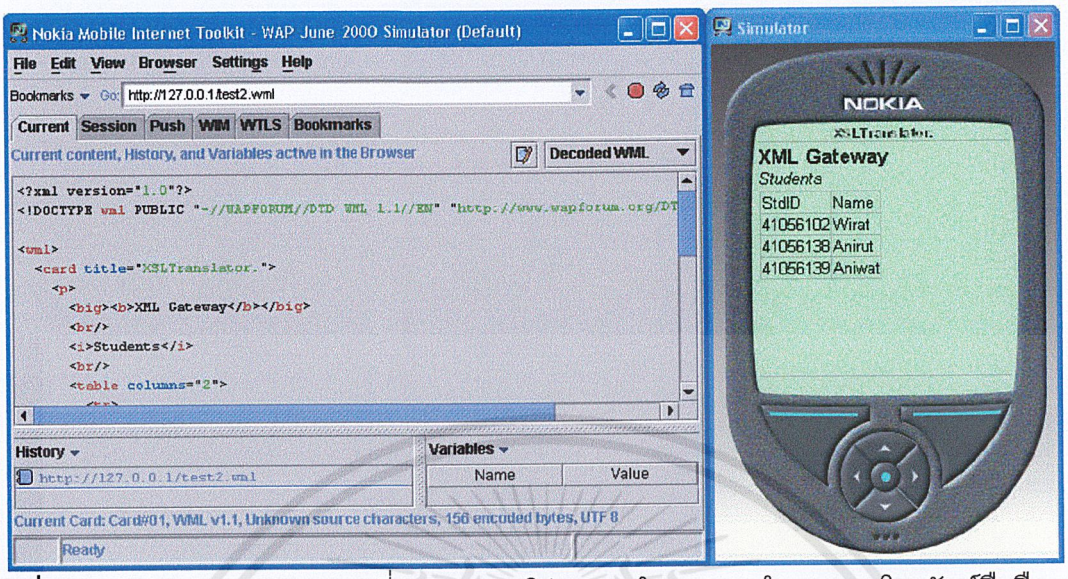
ภาพที่ 4.7 ภาพรายละเอียดเอกสาร test2.html ที่จะนำไปใช้แปลงเป็นเอกสาร WML

หลังจากที่ใช้ตัวแปลงเอกสารรูปแบบ XSL ของโปรแกรม จะได้ผลลัพธ์ที่เป็นเอกสาร WML

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
        "http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
 <card title="XSLTranslator.">
  <p>
   <big><b>XML Gateway</b></big><br/>
   <i>Students</i><br/>
   <table columns="2">
      <tr>  <td>StdID</td>      <td>Name</td>  </tr>
      <tr>  <td>41056102</td>  <td>Wirat</td>  </tr>
      <tr>  <td>41056138</td>  <td>Anirut</td>  </tr>
      <tr>  <td>41056139</td>  <td>Aniwat</td>  </tr>
   </table>
  </p>
 </card>
</wml>
```

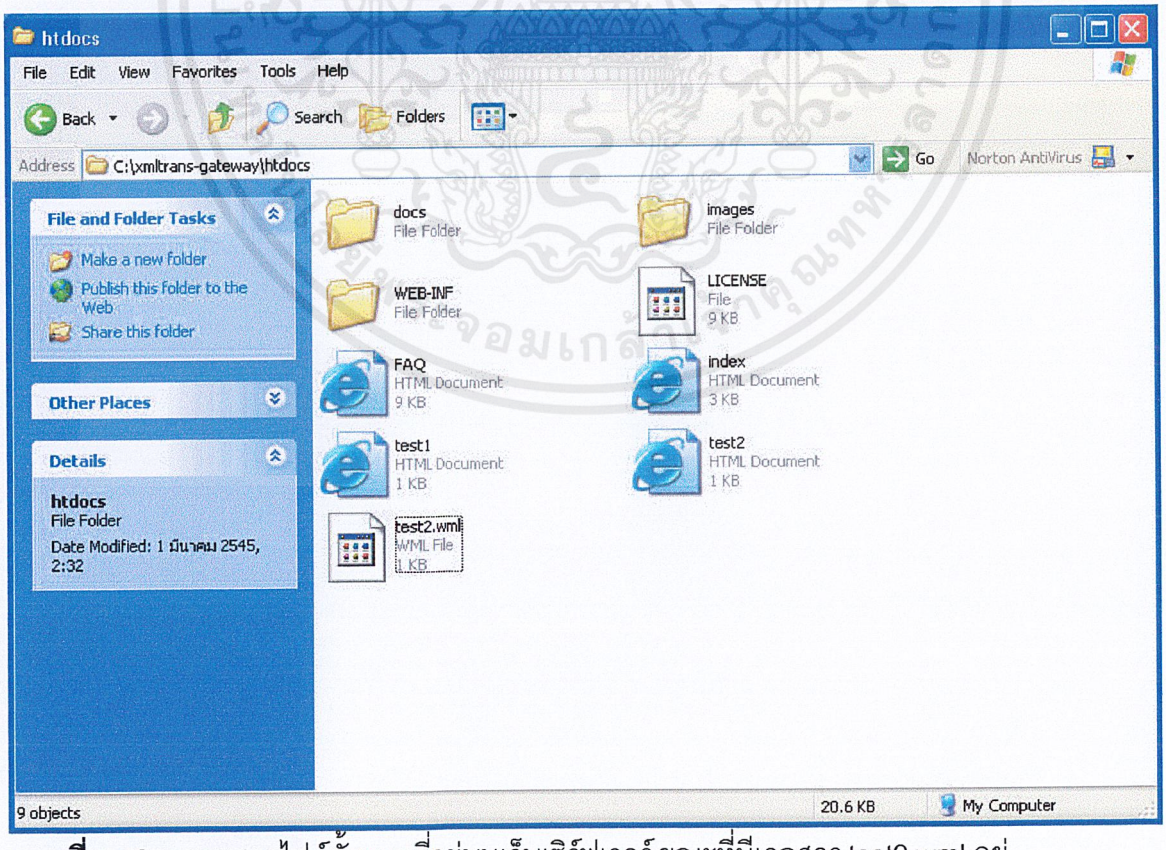
ภาพที่ 4.8 ภาพรายละเอียดเอกสาร test2.wml ที่ได้จากการแปลงมาจากเอกสาร HTML

เอกสาร WML ที่ได้จากการแปลงนั้น สามารถแสดงผลลัพธ์ได้ดังภาพ



ภาพที่ 4.9 ภาพเอกสาร test2.wml ที่แสดงผลบนโปรแกรมจำลองการทำงานของโทรศัพท์มือถือ

เมื่อตรวจสอบไฟล์ทั้งหมดบนเว็บเซิร์ฟเวอร์อีกครั้ง จะพบว่าจะมีเอกสาร WML ชื่อ test2.wml แล้ว ดังภาพ



ภาพที่ 4.10 ภาพแสดงไฟล์ทั้งหมดที่อยู่บนเว็บเซิร์ฟเวอร์ ขณะที่มีเอกสาร test2.wml อยู่

อนึ่ง เอกสาร WML ที่แปลงได้จากการแปลงเอกสาร HTML ข้างต้น สามารถแสดงผลได้เหมือนกับเอกสาร WML ทุกส่วน ดังนั้นเอกสาร WML ที่ได้จึงมีรายละเอียดเหมือนกับเอกสาร HTML ต้นแบบ แต่จากการดำเนินการ ผู้จัดทำพบว่ายังมีเอกสาร HTML อีกหลายลักษณะที่ไม่สามารถแสดงผลได้เหมือนในรูปของเอกสาร WML

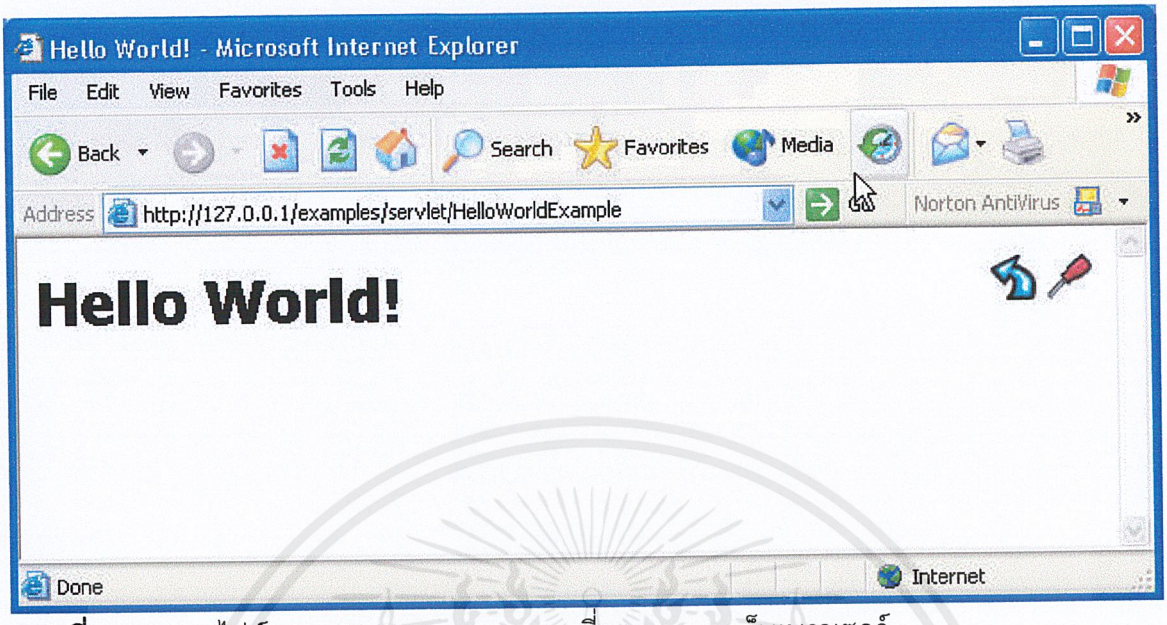
4.4 ผลการประมวลผลไฟล์ Java Servlet ที่เครื่องเซิร์ฟเวอร์

เมื่อมีการร้องขอการประมวลผลไฟล์คลาสของ Java Servlet โปรแกรมก็ทำการตอบสนองได้

```
import java.io.*;
import java.text.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloWorldExample extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException {
         response.setContentType("text/html");
         PrintWriter out = response.getWriter();
         out.println("<html>");
         out.println("<body bgcolor=\"white\">");
         out.println("<head>");
         String title = "Hello World!";
         out.println("<title>" + title + "</title>");
         out.println("</head>");
         out.println("<body>");
         out.println("<a href=\"../servlets/helloworld.html\">");
         out.println("<img src=\"../images/code.gif\" height=24 " + "width=24
    align=right border=0   alt=\"view code\"></a>");
         out.println("<a href=\"../servlets/index.html\">");
         out.println("<img src=\"../images/return.gif\" height=24 " + "width=24
    align=right border=0   alt=\"return\"></a>");
         out.println("<h1>" + title + "</h1>");
         out.println("</body>");
         out.println("</html>");
        }
}
```

ภาพที่ 4.11 ภาพไฟล์ HelloWorldExample.java ที่จะนำมาทดสอบการประมวลผลไฟล์ Java Servlet

ผลที่ได้จากการประมวลผลไฟล์คลาสของ Java Servlet จะแสดงผลบนเว็บเบราเซอร์ได้ดังภาพ



ภาพที่ 4.12 ภาพไฟล์ HelloWorldExample.java ที่แสดงผลบนเว็บเบราเซอร์

บทที่ 5
การอภิปรายผลการดำเนินการ

จากการดำเนินการของโปรแกรมเกตเวย์สำหรับแปลงรูปแบบเอกสาร XML บนเว็บเซิร์ฟเวอร์ ผู้จัดทำขอสรุปถึงความสามารถที่โปรแกรมสามารถทำงานได้มี 3 ส่วนใหญ่ ๆ ดังนี้

5.1 ลักษณะของการติดต่อระหว่างผู้ร้องขอเอกสารกับตัวโปรแกรม

5.2 ลักษณะของตัวแปลงเอกสารรูปแบบ XSL

5.3 ลักษณะของโปรแกรมเว็บเซิร์ฟเวอร์ที่สามารถประมวลผลที่เครื่องเซิร์ฟเวอร์

5.1 ลักษณะของการติดต่อระหว่างผู้ร้องขอเอกสารกับตัวโปรแกรม

เมื่อไคล์เอนท์ร้องขอเอกสาร ตัวโปรแกรมสามารถที่จะตอบสนองการร้องขอเอกสารได้ โดยจะส่งเอกสารกลับไปให้ไคล์เอนท์ใน 2 รูปแบบเท่านั้น ได้แก่ เอกสาร HTML, เอกสาร WML ซึ่งอาจจะเป็นเอกสารที่ไคลเอนท์ร้องขอมาจริง หรือเป็นเอกสารที่มีเนื้อหาเกี่ยวกับความผิดพลาดที่เกิดจากการที่ไคลเอนท์ร้องขอเอกสารที่ไม่มีอยู่จริงก็ได้

5.2 ลักษณะของตัวแปลงเอกสารรูปแบบ XSL

ดังที่ได้กล่าวถึงผลจากการดำเนินการของตัวแปลงเอกสารรูปแบบ XSL ผู้จัดทำจึงแก้ปัญหาที่เกิดขึ้นจากการแปลงเอกสารจาก HTML เป็น WML โดยการตัดบางส่วนของเอกสาร HTML ที่ไม่สามารถแสดงผลได้ด้วยภาษา WML ออกไปสำหรับบางแท็ก หรือปรับเปลี่ยนลักษณะของเอกสารบางส่วนให้อยู่ในรูปแบบที่เหมาะสมกับการแสดงผลด้วยภาษา WML ทั้งนี้ การทำงานในส่วนนี้เป็นหน้าที่ของตัวแปลงเอกสารรูปแบบ XSL ที่แปลงเอกสาร HTML ให้เป็น WML นั่นเอง

5.3 ลักษณะของโปรแกรมเว็บเซิร์ฟเวอร์ที่สามารถประมวลผลที่เครื่องเซิร์ฟเวอร์

เนื่องจากโปรแกรมสามารถประมวลผล Java Servlet ได้ กล่าวคือสามารถตอบสนองคำร้องขอจากไคลเอนท์ได้ แต่ยังไม่สามารถที่จะทำการตอบสนองการร้องขอ Java Servlet ให้ตรงตาม Content Type ที่ร้องขอได้ เพราะในการสร้างไฟล์คลาส Servlet นั้น ผู้สร้างจะต้องทำการระบุ Content Type เอง จึงทำให้โปรแกรมยังไม่สามารถใช้ความสามารถในการประมวลผลที่เครื่องเซิร์ฟเวอร์ร่วมกับตัวแปลงเอกสารรูปแบบ XSL ได้

บทที่ 6
สรุปผลการวิจัยและข้อเสนอแนะ

6.1 สรุปผลการวิจัย

จากการวิจัยโครงงานนี้สรุปได้ว่า โครงงานนี้เป็นเว็บเซิร์ฟเวอร์ที่สามารถตอบสนองเอกสารได้ทั้งแบบ Static และแบบ Dynamic โดยการตอบสนองเอกสารที่เป็น Static นั้นมีส่วนพิเศษที่ต่างจากเว็บเซิร์ฟเวอร์ทั่วไป คือ โปรแกรมจะเก็บรูปแบบเอกสารรูปแบบใดรูปแบบหนึ่งเพียงรูปแบบเดียวบนเว็บเซิร์ฟเวอร์ แต่สามารถตอบสนองการร้องขอเอกสารได้ถึงสองรูปแบบ นั่นคือ HTML และ WML นักพัฒนาเว็บเพจที่ต้องการให้เว็บไซท์ของตนสามารถเข้าถึงได้ทั้งจากคอมพิวเตอร์และจากโทรศัพท์มือถือจึงสามารถใช้ประโยชน์จากจุดนี้ได้ โดยการเลือกที่จะพัฒนาเว็บเพจด้วยภาษาหนึ่งที่ตนมีความถนัดมากกว่า ส่วนการตอบสนองการร้องขอเอกสารแบบ Dynamic นั้น โปรแกรมสามารถทำการประมวลผลไฟล์ Java Servlet บนเครื่องเซิร์ฟเวอร์ แล้วตอบสนองผลที่ได้กลับมายังไคลเอนท์

อีกทั้งโปรแกรมยังสามารถแปลงรูปภาพฟอร์แมท jpeg หรือ gif ไปเป็นฟอร์แมท wbmp ได้อีกด้วย

6.2 ข้อจำกัดของโปรแกรม

6.2.1 โปรแกรมนี้สามารถแปลงเอกสารได้เพียง 2 รูปแบบเท่านั้น คือ การแปลงรูปแบบเอกสารจาก HTML ไปเป็นเอกสาร WML และการแปลงรูปแบบเอกสารจาก WML ไปเป็นเอกสาร HTML

6.2.2 ในกรณีที่ผู้ใช้ต้องการเพิ่มความสามารถในการแปลงเอกสารรูปแบบอื่น ๆ ผู้ใช้ต้องไปทำการแก้ไขตัวโปรแกรม ซึ่งเป็นการยุ่งยาก

6.2.3 โปรแกรมนี้ยังไม่สามารถแปลงแท็กหรือแอตทริบิวต์บางอย่างของเอกสาร HTML ไปอยู่ในรูปของเอกสาร WML ได้ เนื่องจากภาษา HTML มีความหลากหลายในตัวภาษามากกว่า

6.2.4 โปรแกรมไม่สามารถแปลงรูปแบบเอกสารที่อยู่ในรูปแบบที่มีตารางซ้อนตารางได้

6.2.5 โปรแกรมไม่สามารถแปลง Client Side Script ที่ใช้บนเอกสารรูปแบบหนึ่งไปเป็น Client Side Script บนเอกสารอีกรูปแบบหนึ่งได้ ตัวอย่างเช่น ในเอกสาร HTML ซึ่งใช้ Java Script หรือ VB Script โปรแกรมจะไม่สามารถแปลงไปเป็น WMLScript ซึ่งบนเอกสาร WML

6.2.6 โปรแกรมสามารถทำการแปลงเอกสาร HTML ที่มีขนาดเล็กและมีจำนวนรูปภาพน้อยได้เท่านั้น เพราะถ้าเอกสาร HTML มีขนาดใหญ่ และมีจำนวนรูปภาพมาก เมื่อแปลงไปเป็นเป็นเอกสาร WML จะเกิดปัญหาในการแปลงเอกสาร และการแสดงผลเอกสาร WML บนโทรศัพท์มือถือ

6.2.7 การแปลงรูปภาพจากฟอร์แมทที่ใช้แสดงผลกับเว็บเบราเซอร์ เช่น gif หรือ jpeg ไปเป็นรูปภาพฟอร์แมทที่สามารถแสดงผลบนโทรศัพท์มือถือได้ อย่าง wbmp จะเกิดปัญหาขึ้นคือ ในกรณีที่

รูปภาพนั้นมีขนาดใหญ่ แต่การนำรูปภาพไปแสดงผลบนโทรศัพท์มือถือ ซึ่งมีเนื้อที่บนหน้าจอขนาดเล็กนั้น ก็อาจทำให้เกิดปัญหาได้

6.2.8 การแปลงรูปภาพจากฟอร์แมท wbmp ไปเป็นฟอร์แมท gif หรือ jpeg ยังไม่สามารถทำได้ เนื่องจากยังไม่มี Java API ที่จะนำมาใช้ในการแปลงได้

6.2.9 ในกรณีที่เอกสาร HTML มีการอ้างถึงข้อมูลที่เป็นลักษณะมัลติมีเดียต่าง ๆ เช่น ไฟล์เสียง หรือไฟล์วีดีโอ เอกสาร WML ที่ได้จากการแปลงจะไม่สามารถแสดงผลข้อมูลเหล่านี้ได้ เนื่องจากข้อจำกัดในการแสดงผลของโทรศัพท์มือถือที่ยังมีอยู่

6.2.10 ในการทดสอบโปรแกรมร่วมกับโทรศัพท์มือถือจริง โปรแกรมก็สามารถทำงานได้กับโทรศัพท์มือถือระบบ GSM Advance และ DTAC เท่านั้น โดยรุ่นโทรศัพท์มือถือที่นำมาทดสอบคือ

- Nokia 7110
- Nokia 6210
- Siemens C35
- Siemens S45

6.2.11 การทดสอบกับ Simulator นั้นใช้โปรแกรม Nokia Toolkit 3.0 ในการทดสอบ

6.3 ข้อเสนอแนะ

6.3.1 ถ้าต้องการเพิ่มความสามารถในการแปลงรูปแบบเอกสารรูปแบบอื่น ๆ นอกจากสองรูปแบบที่มีอยู่นี้ โปรแกรมจะยอมให้ผู้ใช้สามารถใส่คอมโพเนนท์ชนิด plug-in ใหม่ ๆ เพิ่มเติมลงไปในโปรแกรมเว็บเซิร์ฟเวอร์ได้เอง เพื่อให้ง่ายต่อผู้ใช้ในการแปลงเอกสารได้หลายรูปแบบขึ้น โดยที่ไม่ต้องยุ่งยากกับการแก้ไขโปรแกรม

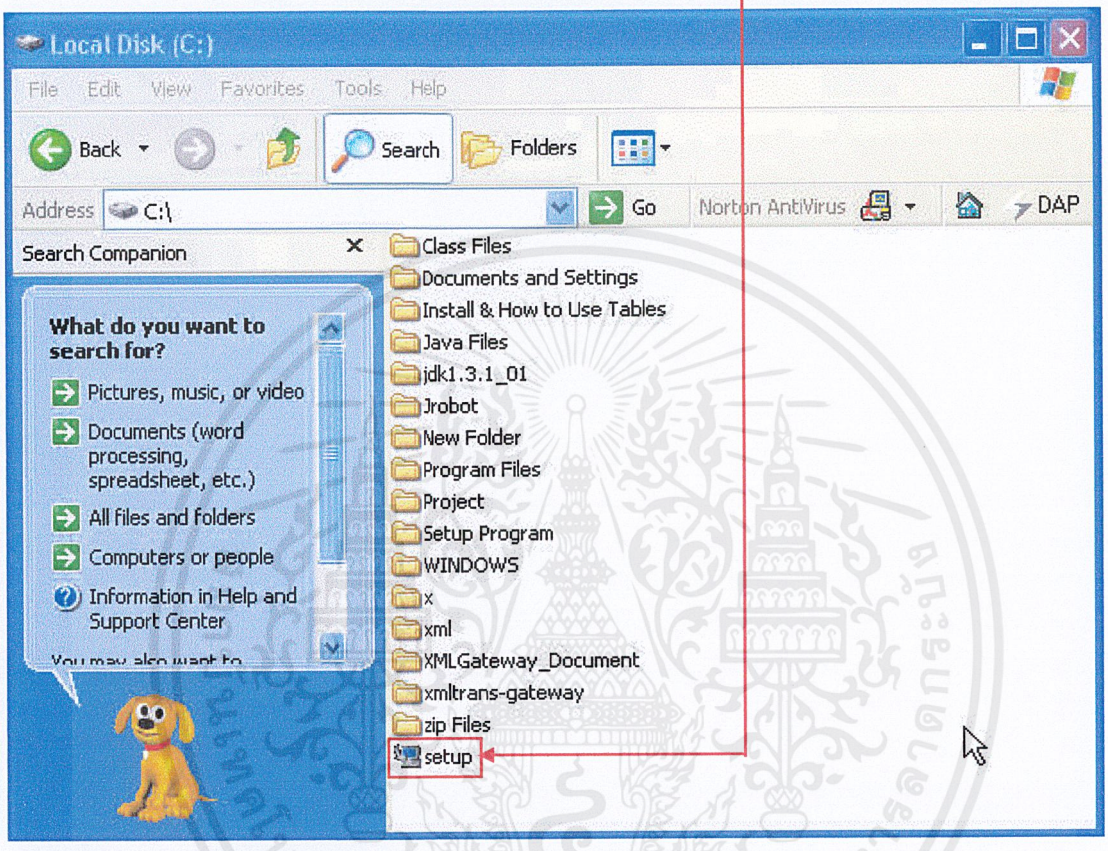
6.3.2 แนวทางการแก้ไขปัญหาเรื่องขนาดรูปภาพหลังทำการแปลง คือ การเพิ่มความสามารถในการทำงาน ให้โปรแกรมเว็บเซิร์ฟเวอร์สามารถตรวจสอบความละเอียดในการแสดงผลของหน้าจอโทรศัพท์ที่ติดต่อเข้ามาได้ เพื่อจะได้นำค่านี้มาใช้ในการคำนวณหาขนาดของภาพที่เหมาะสมสำหรับโทรศัพท์ได้

6.3.3 แนวทางการแก้ไขปัญหาเรื่องของการแสดงภาพที่สมจริง เพื่อให้สามารถสื่อความหมายได้ดีกว่าภาพที่มีลักษณะเป็นภาพขาวดำ ปัญหาอยู่ที่โทรศัพท์มือถือของผู้ใช้เอง ผู้จัดทำเองก็หวังว่าบริษัทผู้ผลิตโทรศัพท์มือถือจะสามารถพัฒนาโทรศัพท์มือถือรุ่นใหม่ ๆ ที่สามารถแสดงผลรูปภาพที่มีหลากหลายสีมากขึ้นได้ในอนาคตอันใกล้นี้

ภาคผนวก

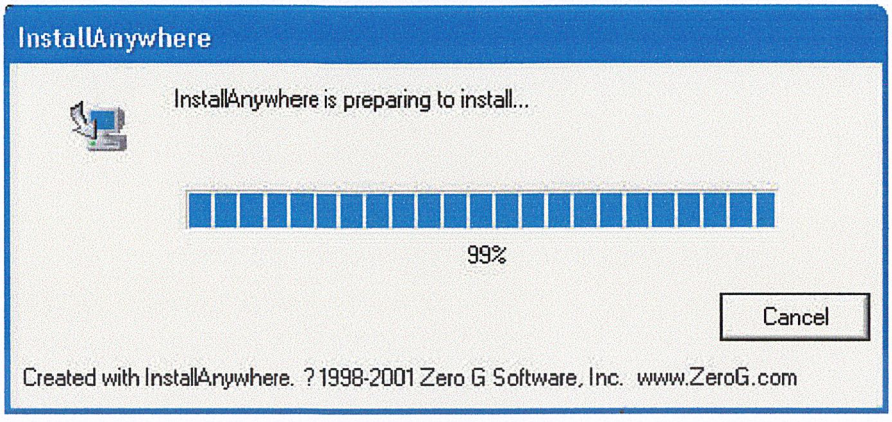
ภาคผนวก ก.
วิธีการติดตั้งโปรแกรม

ขั้นที่ 1 ดับเบิลคลิกที่ไอคอนชื่อ setup



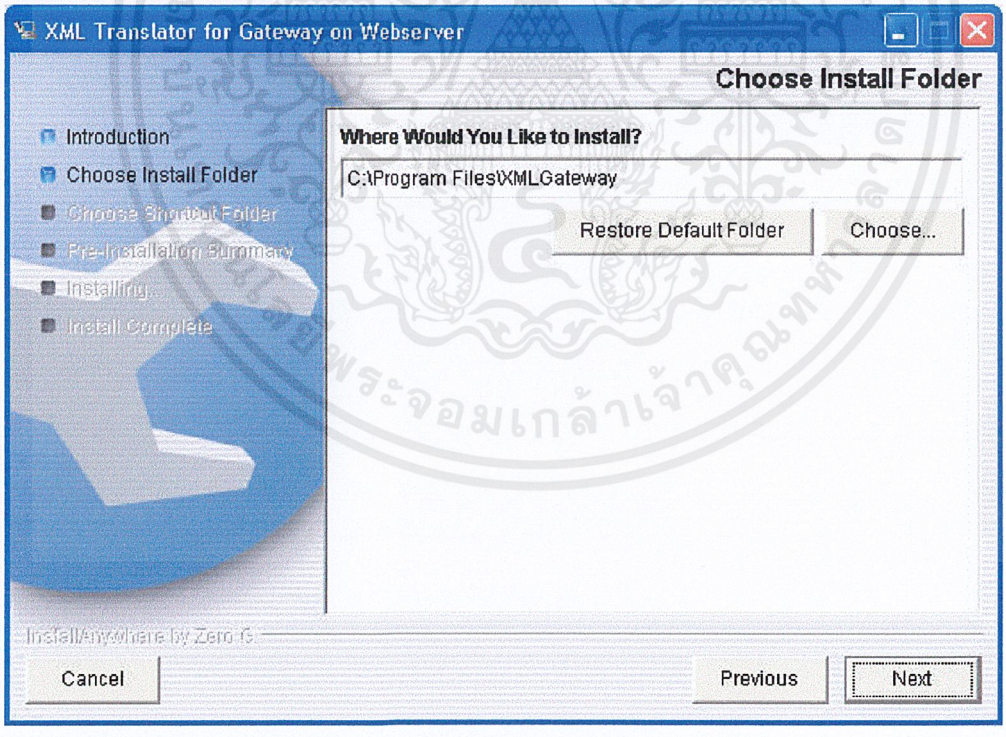
ภาพที่ ก-1 ภาพแสดงรูปของไฟล์ setup

ขั้นที่ 2 จากนั้นโปรแกรมจะแสดงหน้าจอให้เห็นว่ากำลังเตรียมทำการติดตั้ง



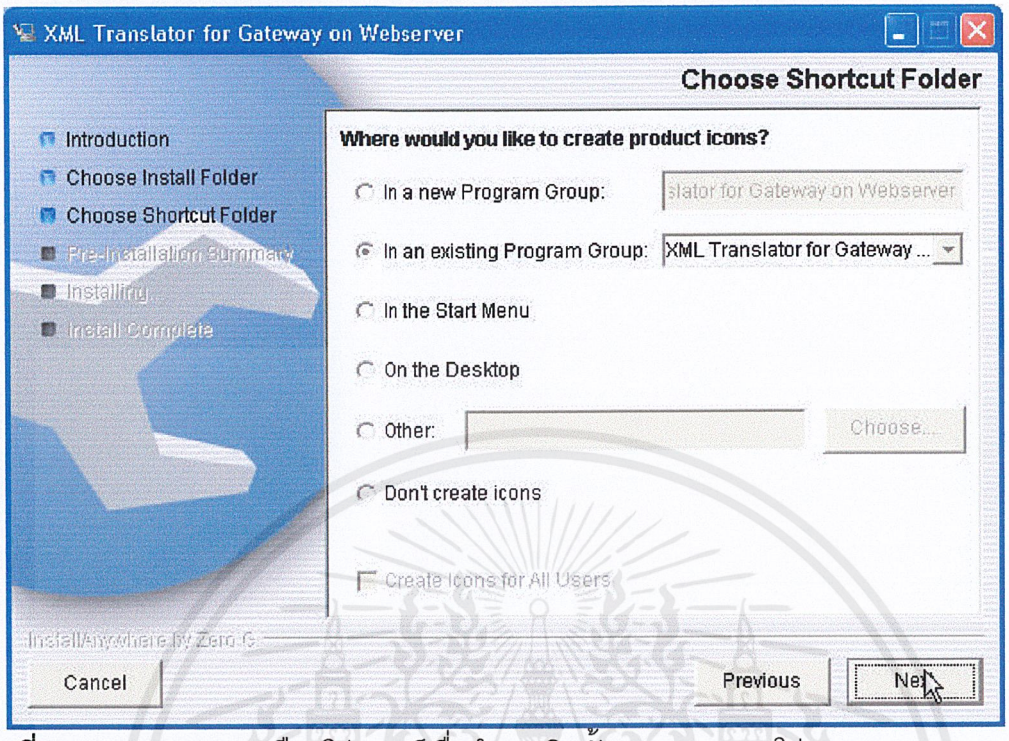
ภาพที่ ก-2 ภาพแสดงการเตรียมการติดตั้งของโปรแกรม

ขั้นที่ 3 โปรแกรมจะเริ่มเข้าสู่การติดตั้ง โดยผู้ใช้สามารถเลือกโฟลเดอร์ที่ต้องการทำการติดตั้งโปรแกรมลงไป จากนั้นคลิกปุ่ม Next เพื่อเข้าสู่การติดตั้งในขั้นต่อไป แต่ถ้าไม่ต้องการติดตั้งก็สามารถยกเลิกได้ด้วยการคลิกปุ่ม Cancel



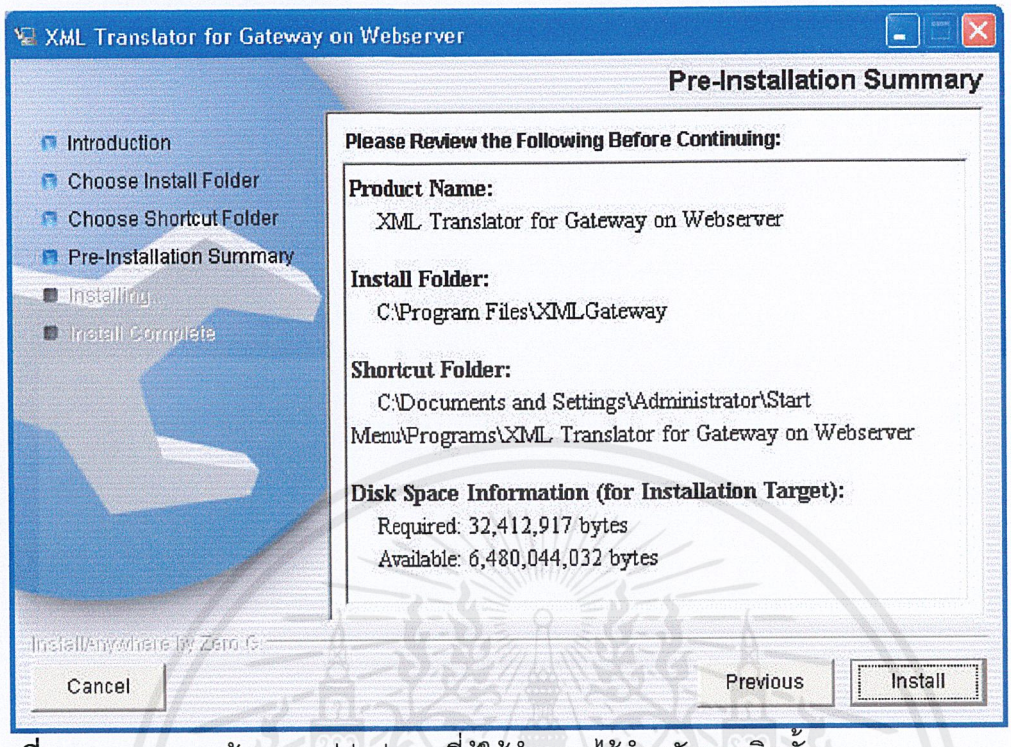
ภาพที่ ก-3 ภาพแสดงการเลือกโฟลเดอร์เพื่อทำการติดตั้งโปรแกรม

ขั้นที่ 4 ตอนนี้ผู้ใช้สามารถเลือกโฟลเดอร์ที่ทำการติดตั้ง shortcut แล้วจึงคลิกปุ่ม Next



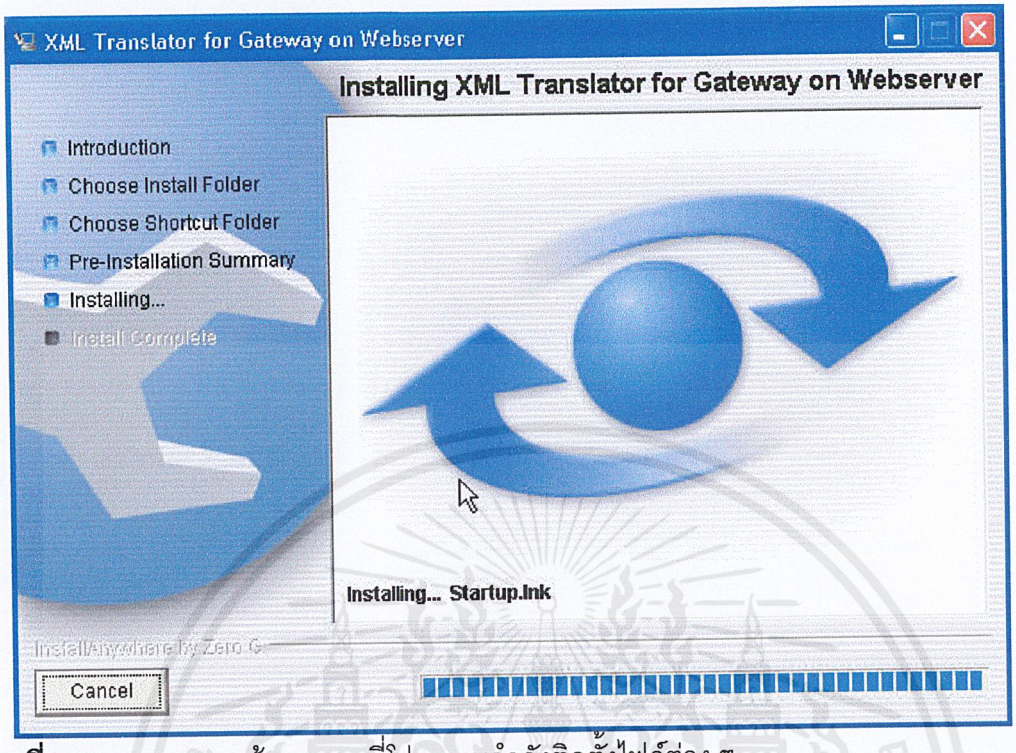
ภาพที่ ก-4 ภาพแสดงการเลือกโฟลเดอร์เพื่อทำการติดตั้ง shortcut ของโปรแกรม

ขั้นที่ 5 โปรแกรมจะแสดงข้อมูลสรุปสิ่งที่เราได้กำหนดไว้ข้างต้นบนหน้าจอ ให้กดปุ่ม Install



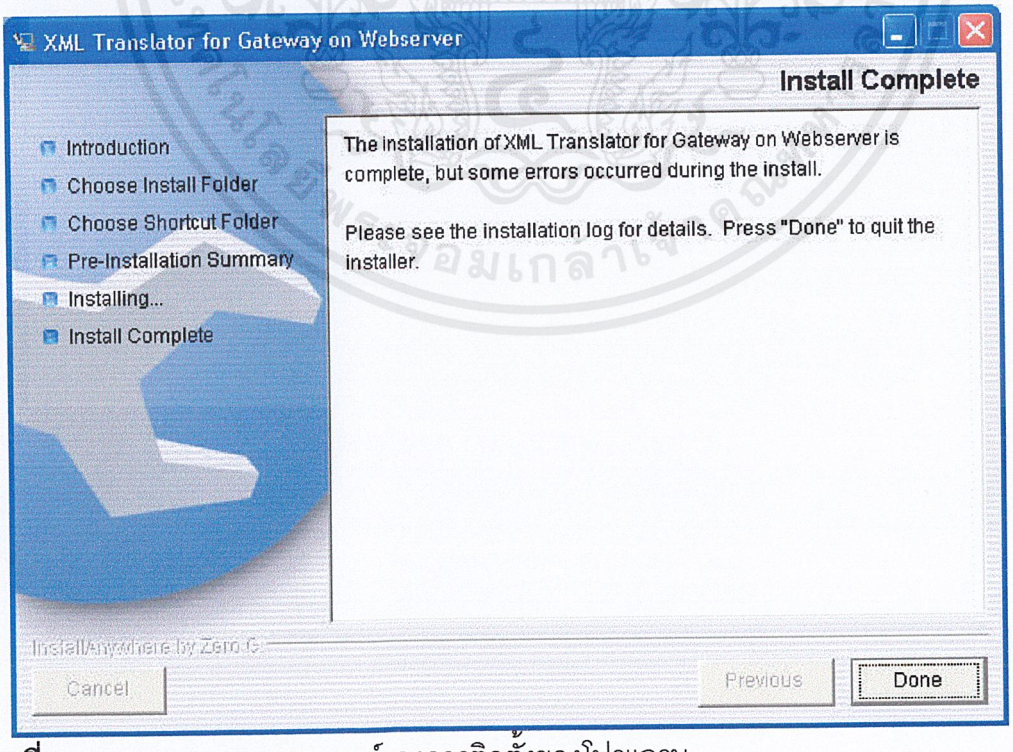
ภาพที่ ก-5 ภาพแสดงข้อมูลสรุปค่าต่าง ๆ ที่ผู้ใช้กำหนดไว้สำหรับการติดตั้ง

ขั้นที่ 6 โปรแกรมทำการติดตั้งไฟล์ต่าง ๆ ลงไป



ภาพที่ ก-6 ภาพแสดงหน้าจอขณะที่โปรแกรมกำลังติดตั้งไฟล์ต่าง ๆ

ขั้นที่ 7 เมื่อโปรแกรมได้ถูกติดตั้งเสร็จสมบูรณ์แล้ว ให้กดปุ่ม Done ก็เป็นอันเสร็จสิ้นการติดตั้ง



ภาพที่ ก-7 ภาพแสดงความสมบูรณ์ของการติดตั้งของโปรแกรม

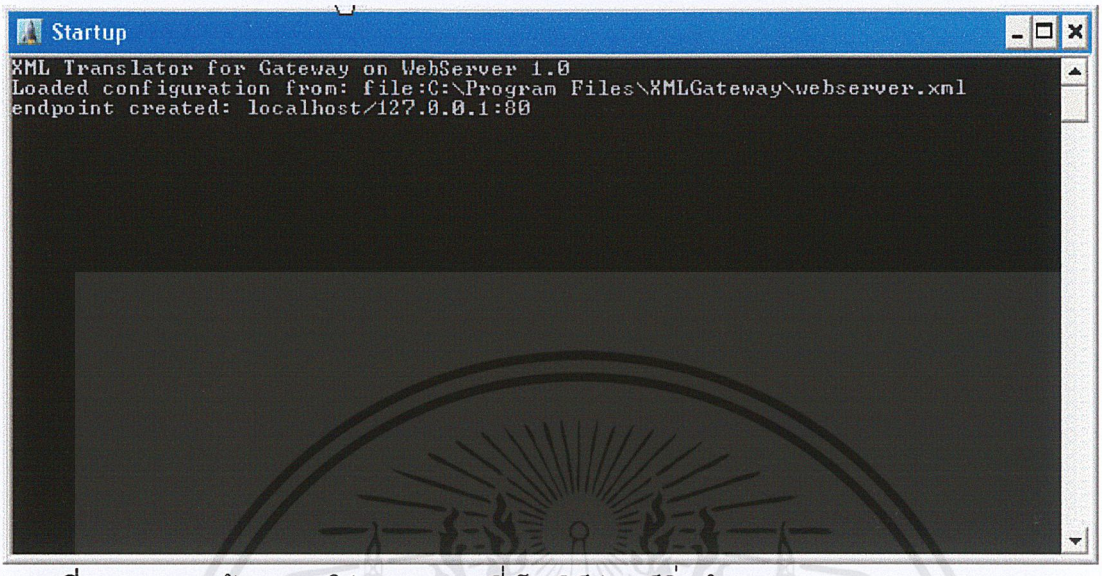
ภาคผนวก ข.
วิธีการใช้งาน

การเรียกใช้โปรแกรมจากเมนู Start ของ Windows

1. เมื่อทำการติดตั้งโปรแกรมเสร็จสิ้นแล้วสามารถจะเรียกให้โปรแกรมทำงานได้โดยเลือก Start->Program->XML Translator for Gateway on Webserver->Startup ดังนี้

ภาพที่ ข-1 ภาพการเรียกโปรแกรมจากเมนู Start

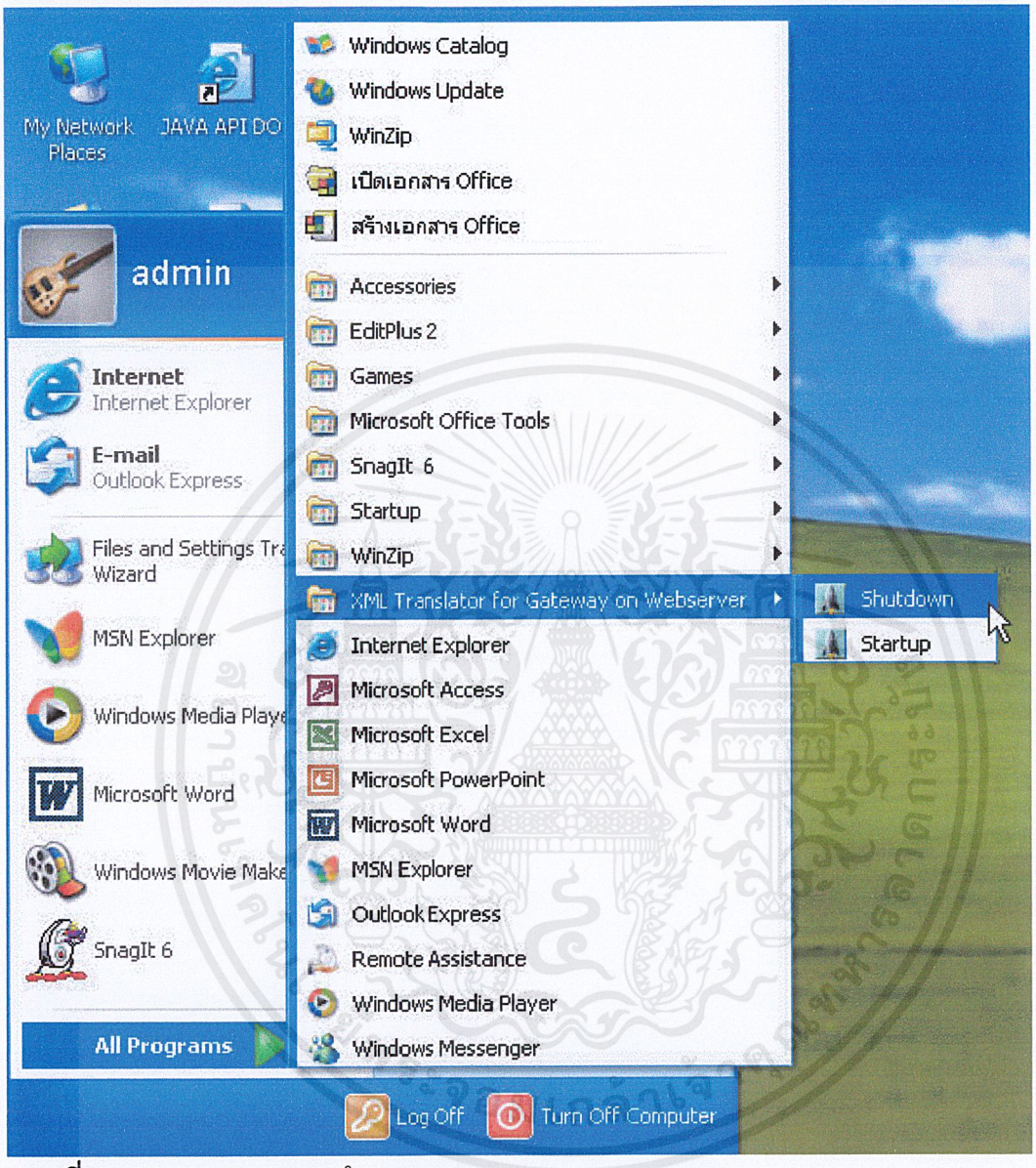
2. เมื่อทำการ Startup เพื่อทำการเปิดเว็บเซิร์ฟเวอร์ หลังจากนั้นโปรแกรมสามารถรอรับการร้องขอเอกสารจากไคลเอนท์ได้ โดยที่หน้าจอของเว็บเซิร์ฟเวอร์ เป็นดังนี้



ภาพที่ ข-2 ภาพหน้าจอของโปรแกรมขณะที่เว็บเซิร์ฟเวอร์เริ่มทำงาน

3. การหยุดการทำงานของเว็บเซิร์ฟเวอร์ทำได้โดยเลือก

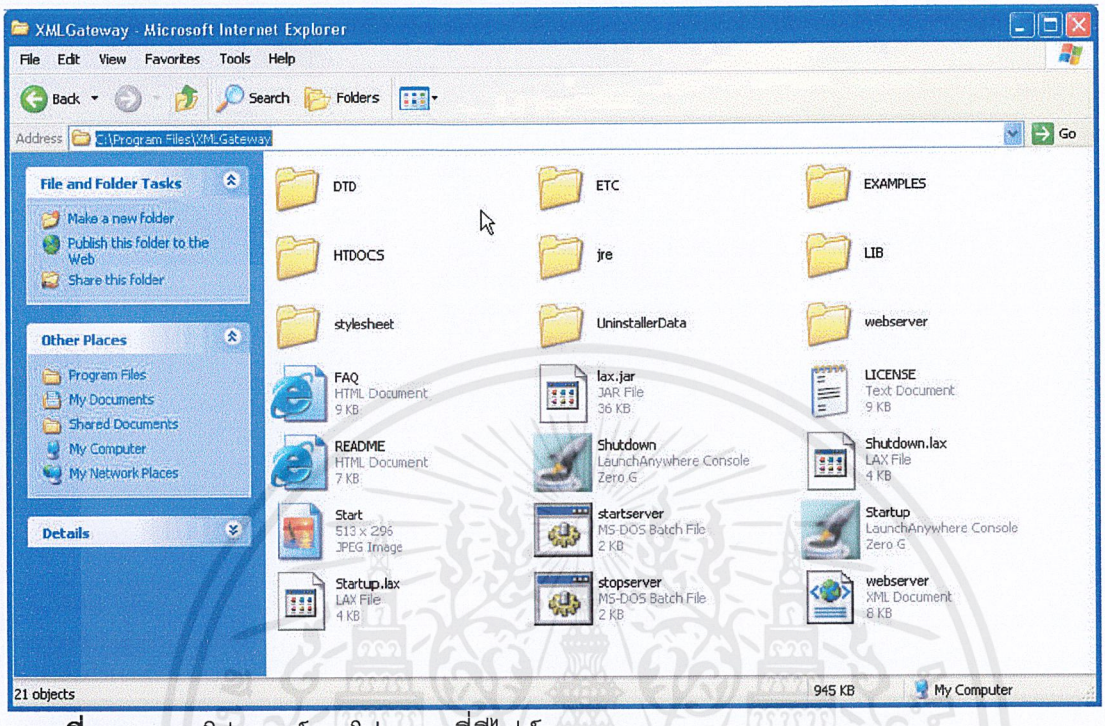
Start->Program->XML Translator for Gateway on Webserver->Shutdown



ภาพที่ ข-3 ภาพการหยุดการทำงานจากเมนู Start

การเรียกใช้โปรแกรมจากไฟล์ batch โดยตรง

1. เข้าสู่โฟลเดอร์ของโปรแกรม โดยปกติแล้วโฟลเดอร์ที่เป็นดีฟอลท์ของการติดตั้งโปรแกรม ก็คือ C:\Program Files\XMLGateway

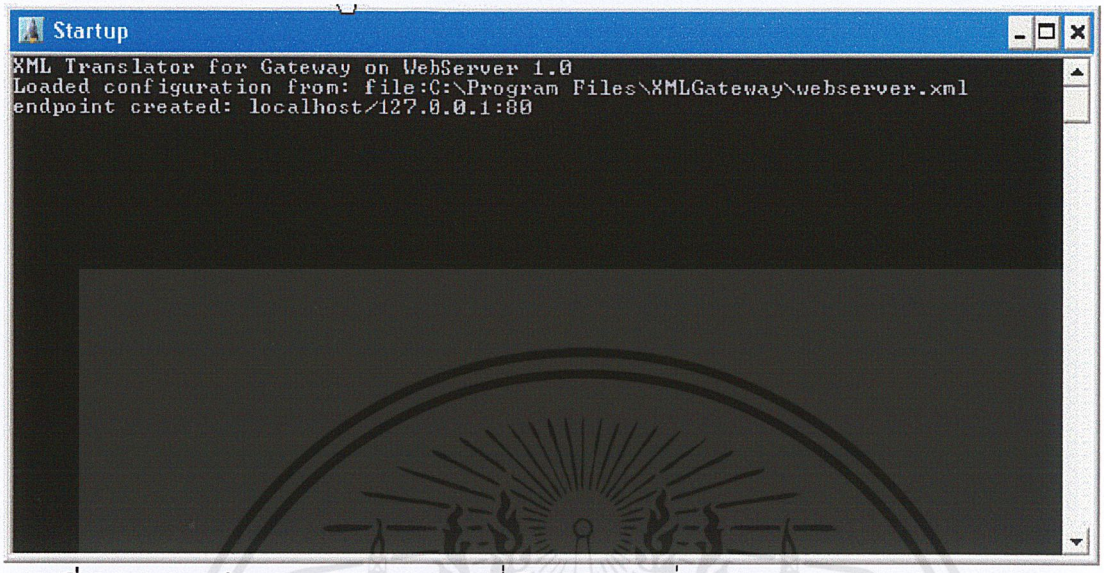


ภาพที่ ข-4 ภาพโฟลเดอร์ของโปรแกรมที่มีไฟล์ batch

2. เมื่อต้องการเข้าเรียกให้เว็บเซิร์ฟเวอร์ทำงาน ให้เรียกไฟล์ startserver.bat

ภาพที่ ข-5 ภาพไฟล์ startserver.bat

3. หลังจากนั้นโปรแกรมก็จะสามารถรอรับการร้องขอเอกสารจากไคลเอนท์ได้ หน้าจอของเว็บเซิร์ฟเวอร์เป็นดังนี้



ภาพที่ ข-6 ภาพหน้าจอของโปรแกรมขณะที่เว็บเซิร์ฟเวอร์เริ่มทำงาน

4. การหยุดการทำงานของเว็บเซิร์ฟเวอร์ทำโดยทำการเรียกไฟล์ stopserver.bat

ภาพที่ ข-7 ภาพไฟล์ stopserver.bat

ภาคผนวก ค.
การกำหนดค่าในโปรแกรม

การกำหนดค่าต่าง ๆ ใน XML Translator for Gateway on Webserver ทำได้ทำการแก้ไขไฟล์ XML ที่มีชื่อว่า webserver.xml ซึ่งอยู่ใน Folder หลักของโปรแกรมหลังจาก ทำการติดตั้งโปรแกรม ซึ่งรายละเอียดการ Configuration ต่างๆได้รวมเข้าไว้กับไฟล์ webserver.xml แล้ว

รายละเอียดของไฟล์ webserver.xml มีดังนี้

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!--

This file is the default configuration file for the
XML Translator for Gateway on Webserver. Following is a brief overview of the
XML Translator for Gateway on Webserver configuration options.

=============================================================
webserver dtd and xml:
Element
   Attribute(s)
   Element(s)
=============================================================

WebServer - A collection of web services managed by a
        single HTTP Web Server instance.

   id - A Unique Web Server id.
   adminPort - The Web Server administration port
           number which is used as an external
           Web Server hook to invoke administrative
           tasks such as gracefully shutting down
```

the web server (note: these services
are presently not specified and as such
are subject to change).

Service - A managed web service.

Service - A distinct web resource which is associated
with a fully qualified URI.

id - A unique Service id.
port - The port number with which the Service is
registered.
hostName - The system host name in which the Service
is hosted.
inet - The system ip address in which the Service is
hosted.
docBase - The Service document base.
workDir - The Service work directory.
workDirIsPersistent - Indicator as to whether or not
the Web Server should return
the associated work directory
to the host system upon shut
down.

WebApplication - A managed association of web resources.

WebApplication - A collection of associated web resources
which correspond to a distinct fully
qualified URI.

id - A unique Web Application id.
mapping - The URI prefix with which this Web Application

is associated with relative to the hosted
Service.

docBase - The Web Application document base.

maxInactiveInterval - The maximum session timeout
period.

==========

command line options:

-help This Message
-config [file|url] Read config from URL
-noconfig Do not read config
-adminport [int] Administration Port
-serviceid [str] Service Id
-port[:id|:*] [int] Listen on int [for Service id]
-inet[:id|:*] [inetaddr] Bind server to inet [for Service id]
-hostname[:id|:*] [name] Use name as hostname [for Service id]
-docbase[:id|:*] [name] Use URL as the content base [for Service id]
-workdir[:id|:*] [name] Use scratch file [for Service id]

==========

configuration attribute details:

All "id" values must be unique with a collection
of like elements.

Most configuration values can be declared in one
of following three means respectively:

command line
xml declaration
default as specified by the dtd and/or
application

Many fields have default falues which are either
specified in the associated dtd or within XML Translator for Gateway on
Webserver.
These default values can be changed by modifying
the included webserver.xml and fully qualifying
the appropriate element attributes.

A XML Translator for Gateway on Webserver can be started on any platform
with no changes to the provided default
webserver.xml configuration.

The XML Translator for Gateway on Webserver will create a default
configuration file upon initialization if it does
not find one either by looking in the XML Translator for Gateway on Webserver
install directory for the file "webserver.xml or
the "-config [file | url]" command line option.

If an explicit WebServer.adminPort value is not
specified then a series of 5 attempts will be
made to bind to an available port number randomly
chosen between the range of 2048 and 8192. Upon
success, the chosen administration port is logged
in the "webserverlog.txt" file. If an administration
port cannot be determined, the web server will
not be started.

The WebServer.id field is likely to be removed
in a future release.

The Service.id field is required as the key for
the command line arguments. Duplicate Server.port
entries will cause the subsequent duplicate Services
to be disregarded.

The Server.hostName is opional and a value of
"localhost" will be used if it remains
unspecified.

A Server.port field must be unique within a
collection WebServer configuration. Subsequent
Service instances specified with duplicate port
numbers will fail initialization.

The Server.docBase is the file system location
which is accessed by the Web Server to route
inbound http requests to a specific Service instance
and not picked up by an associated Web Application.
A document base can be specified as relative or
absolute and need not reside within the XML Translator for Gateway on Webserver
install directory. It should be possible to specify URI
addresses as well effectively turning this WebServer
Service instance into a proxy server although this is
experimental with this release.

The Server.workDir specfies the local file system
directory available to the Web Server as needed to
use as a cache, archive object persistence among

other tasks. The work directory can be specified as relative or absolute and need not reside within the XML Translator for Gateway on Webserver install directory.

The Server.workDirIsPersistent is an indicator to the WebServer to either save or return to the host system the associated work directory. This field is likely to be renamed to "isWorkDirPersistent" in a future release.

The WebApplication.id field is likely to be removed in a future release.

The WebApplication.mapping is used to specify the URI prefix with which this Web Application instance is to be associated with. The specified value of this field must be unique amongst a collection managed by a single Service instance. This field is likely to be renamed to "path" in a future release.

The WebApplication.docBase is the file system location which is accessed by the Web Server to service inbound http requests routed to this specific Web Application instance. This field shares many of the attributes specified in the Server.docBase description above.

The WebApplication.maxInactiveInterval is not utilized at this time and will likely specify the "session time out in minutes" threshold in

```
    a future release.

============================================================

Miscellany:

    Any number of Service and/or Web Application
    instances can be readily added to an existing Web
    Server configuration by adding the appropriate and
    valid (as specified by the dtd and associated rules)
    xml details.

============================================================

-->

<!DOCTYPE WebServer [

<!ELEMENT WebServer (Service+)>
<!ATTLIST WebServer
    id ID #REQUIRED
    adminPort NMTOKEN "8080">


<!ELEMENT Service (WebApplication*)>
<!ATTLIST Service
    id ID #REQUIRED
    port NMTOKEN "80"
    hostName NMTOKEN "xmlgateway"
    inet NMTOKEN ""
    docBase CDATA "htdocs"
```

```
    workDir CDATA "work"
    workDirIsPersistent (false | true) "false">

<!ELEMENT WebApplication EMPTY>
<!ATTLIST WebApplication
    id ID #REQUIRED
    mapping CDATA #REQUIRED
    docBase CDATA #REQUIRED
    maxInactiveInterval NMTOKEN "30">
]>

<WebServer id="webServer">
    <Service id="service0">
        <WebApplication id="examples" mapping="/examples" docBase="examples"/>
    </Service>
</WebServer>
```

ภาพที่ ค-1 ภาพรายละเอียดของไฟล์ webserver.xml

ภาคผนวก ง.

ไวยากรณ์ Document Type Definition

Document Type Definition ในภาคผนวก ง. นี้ ประกอบไปด้วย Document Type Definition ของ 3 ภาษา ดังนี้

- Document Type Definition ของภาษา XSL ในส่วนของ XSLT สามารถอ่านรายละเอียดได้จาก [CD-ROM DRIVE]:\DTD\XSL.DTD

- Document Type Definition ของภาษา WML สามารถอ่านรายละเอียดได้จาก [CD-ROM DRIVE]:\DTD\WML_1.1.XML

- Document Type Definition ของภาษา XHTML สามารถอ่านรายละเอียดได้จาก [CD-ROM DRIVE]:\DTD\XHTML1-TRANSITIONAL.DTD

บรรณานุกรม

สราวุธ อ้อยศรีสกุล. "**เปิดมิติ Mobile Internet ด้วย WAP**" วิตตี กรุป. 2544

Anderson, R. et. al. Professional XML. Birmingham : Wrox. 2000.

Eric Armstrong. "**A Quick Introduction to XML**" [Online]. Available : http://java.sun.com/sml/jaxp-docs-1.0.1/docs/tutorial/overview/1_xml.html. 2000

Eric Armstrong. "**Creating a Document Type Definition (DTD)**" [Online]. Available : http://java.sun.com/sml/jaxp-docs-1.0.1/docs/tutorial/sax/5a_dtd.html. 2000

Eric Armstrong. "**Defining Attributes and Entities in DTD**" [Online]. Available : http://java.sun.com/sml/jaxp-docs-1.0.1/docs/tutorial/sax/5c_dtd.html. 2000

Sun Microsystems, Inc. "**Java™2 SDK, Standard Edition Documentation version 1.2.2**" [Online]. Available : http://java.sun.com/products/jdk/1.2/docs/index.html.

Wireless Application Protocol Forum. "**Wireless Markup Language Specification Version 1.1**" [Online]. Available : http://www1.wapforum.org/tech/documents/SPEC-WML-19990616.pdf. 1999

World Wide Web Consortium. "**eXtensible Stylesheet Language (XSL) Version 1.0 W3C Working Draft**" [Online]. Available : http://www.w3.org/TR/2000/WD-xsl-20000327. 2000

World Wide Web Consortium. "**XSL Transformations (XSLT) Version 1.0 W3C Recommendation**" [Online]. Available : http://www.w3.org/TR/1999/REC-xslt-19991116. 1999

World Wide Web Consortium. "**HTML 4.0 Specification W3C Recommendation, revised on 24-Apr-1998, World Wide Web Consortium**" [Online]. Available : http://www.w3.org/TR/1998/REC-html40-19980424. 1998

World Wide Web Consortium. "**Hypertext Transfer Protocol – HTTP/1.1**" [Online]. Available : http://www.w3.org/Protocols/rf2616.html. 1999