ระบบส่งข้อความเสียงอัตโนมัติ

AUTOMATIC VOICE SENDING SYSTEM

โดย

นายภาณุ ลีลาลัคนา

นายวัชรพล พฤกษสุขเกษม

อาจารย์ที่ปรึกษา

ดร. ชุติเมษฎ์ ศรีนิลทา

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2543

ปริญญานิพนธ์ปีการศึกษา 2543

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง ระบบส่งข้อความเสียงอัตโนมัติ

AUTOMATIC VOICE SENDING SYSTEM

ผู้จัดทำ

1. นาย ภาณุ ลีลาลัคนา รหัสประจำตัว 39014391
2. นาย วัชรพล พฤกษสุขเกษม รหัสประจำตัว 39014470

ชุติเมษฎ์ ศรีนิลทา อาจารย์ที่ปรึกษา

(ดร. ชุติเมษฎ์ ศรีนิลทา)

ระบบส่งข้อความเสียงอัตโนมัติ

นายภาณุ ลีลาลัคนา 39014391
นายวัชรพล พฤกษสุขเกษม 39014470
ดร. ชุติเมษฎ์ ศรีนิลทา อาจารย์ที่ปรึกษา
ปีการศึกษา 2543

บทคัดย่อ

โครงงานนี้เป็นการสร้างระบบส่งข้อความเสียงอัตโนมัติขึ้นมา ซึ่งเป็นการผสมผสานของเทคโนโลยีเครือข่ายไอพีและ PSTN (Public Switched Telephone Network) เข้าด้วยกัน ทำให้ลดค่าใช้จ่ายในการติดต่อสื่อสารได้มาก ระบบส่งข้อความเสียงอัตโนมัติจะประกอบไปด้วยแอพพลิเคชัน 2 ส่วน คือ แอพพลิเคชันที่ทำงานอยู่บนเครื่องไคลเอนต์ และแอพพลิเคชันที่ทำงานอยู่บนเซิร์ฟเวอร์ โดยมีการจัดเก็บลงบนฐานข้อมูล ซึ่งทางฝั่งไคลเอนต์จะบันทึกข้อมูลเสียงที่ต้องการเพื่อส่งไปยังฝั่งเซิร์ฟเวอร์ เพื่อเซิร์ฟเวอร์จะได้ใช้วอยซ์โมเด็มทำการโทรศัพท์ไปยังเลขหมายปลายทางตามที่ได้ตั้งเวลา พร้อมทั้งเล่นไฟล์เสียงที่บันทึกไว้ รวมทั้งระบบนี้สามารถคิดค่าบริการตามอัตราโทรที่ได้กำหนดด้วย

Automatic Voice Sending System

Panu Leelaluckana

Watcharaphol Pruksasukasem

Dr. Chutimet Srinilta Advisor

ABSTRACT

This project is about building an Automatic Voice Sending System, which merge technology of IP network and PSTN (Public Switched Telephone Network). It can make a lot of cost reduction in telecommunication. This Automatic Voice Sending System contains 2 applications, one is working on client side and another one is working on server side, which connect to a database. The client-side application functions are to record the voice as needed and send that voice to server. After receiving the voice, the server will make a phone call and play the recorded voice to the end user by using voice modem on scheduled. And this system can collect the service charge by call rate as well.

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้คงไม่อาจเสร็จได้ด้วยดี หากไม่ได้รับความช่วยเหลือ และร่วมมือจากหลาย ๆ ฝ่ายด้วยกัน บุคคลแรกที่ต้องกล่าวถึงเพราะเป็นส่วนสำคัญที่ทำให้วิทยานิพนธ์นี้เสร็จลงได้ก็คือ อาจารย์ ชุติเมษฎ์ ศรีนิลทา ซึ่งเป็นอาจารย์ที่ปรึกษาวิทยานิพนธ์ ที่ให้ความเอาใจใส่ แนะนำและช่วยเหลือเสมอมา ซึ่งต้องขอขอบพระคุณเป็นอย่างมาก รวมถึงเพื่อนๆ ห้อง OLALA ที่ช่วยแนะนำในส่วนที่ติดขัด ทำให้สามารถทำงานลุล่วงไปด้วยดี และยังมีพี่ ExitSub ใน ICQ ที่ช่วยเหลือด้านภาษา C แก่ข้าพเจ้า

และต้องขอขอบพระคุณบุคคลสำคัญที่สุดที่ทำให้ข้าพเจ้ามีวันนี้ ก็คือ บิดาและมารดา อันเป็นที่เคารพรักยิ่ง ซึ่งได้เลี้ยงดูผู้เขียนมาเป็นอย่างดี พร้อมทั้งให้โอกาสในการศึกษาอย่างเต็มที่ และยังให้กำลังใจ เอาใจใส่เสมอมาในทุกๆ ด้านอันหาที่เปรียบมิได้ ข้าพเจ้าขอระลึกในพระคุณอันสุดประมาณ และขอ กราบขอบพระคุณมา ณ ที่นี้

ภาณุ ลีลาลัคนา

วัชรพล พฤกษสุขเกษม

สารบัญ

หน้าที่

สารบัญตาราง

หน้าที่

สารบัญรูปภาพ

หน้าที่

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มา

ในปัจจุบันการติดต่อสื่อสารถึงกันและกันทำได้อย่างง่ายขึ้น เนื่องจากเทคโนโลยีมีความก้าวหน้าไปอย่างมาก ดังจะเห็นได้จากจำนวนการใช้โทรศัพท์มือถือในการติดต่อถึงกันมีเพิ่มสูงขึ้น มีการใช้อี-เมล์แทนการส่งจดหมายถึงกัน ทำให้มีความรวดเร็วซึ่งสามารถถึงผู้รับเพียงไม่กี่ชั่วโมงเท่านั้น การใช้อินเทอร์เนตมีอัตราที่เพิ่มสูงขึ้น ดังนั้นการที่จะนำระบบอินเทอร์เนตและระบบโทรศัพท์ที่มีอยู่แล้วมาอิมพลีเมนต์ร่วมกัน ทำให้การติดต่อสื่อสารให้เข้าถึงบุคคลเป็นไปได้อย่างง่ายขึ้น

เนื่องจากการพัฒนาที่ต่อเนื่องของเทคโนโลยีในการบีบข้อมูลทำให้คุณภาพเสียงดีขึ้น และความสามารถที่เพิ่มขึ้นของเครื่องพีซีในปัจจุบันทำให้สามารถทำงานที่เจาะจงได้โดยไม่ต้องพึ่งฮาร์ดแวร์ที่ทำงานเฉพาะ รวมถึงจำนวนผู้ใช้อินเทอร์เน็ตที่เพิ่มขึ้นอย่างรวดเร็ว เป็นผลให้อินเทอร์เน็ตเทเลโฟนนี่สามารถนำมาประยุกต์ใช้งานได้อย่างมีประสิทธิภาพ

1.2 วัตถุประสงค์ของโครงงาน

1.2.1 ศึกษาเทคโนโลยีที่เกี่ยวกับ IP Telephony โดยเน้นไปที่ TAPI 3.0 ของไมโครซอฟท์ ว่ามีสถาปัตยกรรมอย่างไร มีอินเทอร์เฟซที่ใช้อะไรบ้าง และสามารถนำไปประยุกต์ใช้งานได้อย่างไรบ้าง

1.2.2 เพื่อนำเทคโนโลยีต่างๆ ที่ได้ศึกษามาผสมผสานกันเพื่อให้ใช้งานได้อย่างหลากหลาย ทำให้ต้นทุนในการใช้งานถูกลง เนื่องจากเป็นการพัฒนาทางด้านซอฟต์แวร์แทนผลิตภัณฑ์ฮาร์ดแวร์ที่มีราคาแพง และสามารถลดค่าใช้จ่ายในติดต่อในระยะทางไกลได้

1.3 ขอบเขตของโครงงาน

โครงงานนี้จะสร้างระบบส่งข้อความเสียงอัตโนมัติขึ้นมา ซึ่งจะประกอบไปด้วยแอพพลิเคชัน 2 ส่วน คือ แอพพลิเคชันที่ทำงานอยู่บนเครื่องไคลเอนต์ที่ใช้ภาษา Visual Basic และแอพพลิเคชันที่ทำงานอยู่บนเซิร์ฟเวอร์ที่ใช้ภาษา C++ โดยมีการติดต่อกับฐานข้อมูลบน Microsoft Access 2000 เพื่อใช้ในการบันทึกข้อมูลที่เกี่ยวกับผู้ใช้และการโทรศัพท์ออกไป ซึ่งระบบนี้มีความสามารถในบันทึกข้อมูลเสียงแล้วทำการโทรศัพท์ไปยังเลขหมายปลายทางตามที่ได้ตั้งเวลาเอาไว้ รวมทั้งสามารถใช้ Text-to-Speech ในการแปลงข้อความที่เป็นเท็กซ์เป็นเสียงแทนการบันทึกเสียงได้

อย่างไรก็ตาม โครงงานนี้ไม่สนับสนุนผู้ใช้หลายคนในเวลาเดียวกัน และเนื่องจากข้อจำกัดของ TAPI 3.0 ทำให้ไม่สามารถตรวจสอบสถานะของโทรศัพท์ปลายทางได้ นั่นก็หมายความว่าการเล่นไฟล์ข้อความเสียงไปยังโทรศัพท์ปลายทางนั้น จะกระทำทันทีโดยไม่สนใจว่าผู้ใช้ปลายทางจะรับโทรศัพท์หรือสายไม่ว่างก็ตาม

นอกจากนั้นในโครงงานนี้ยังถือว่าเป็นโครงการที่ทดลองสร้าง เพื่อศึกษาความเป็นไปได้ในการใช้งานของคอมพิวเตอร์ร่วมกับโทรศัพท์ ดังนั้นจึงมีข้อจำกัดของข้อมูลบางอย่าง เช่น อาจจะมีการจำกัดปริมาณของผู้ใช้และรายการที่ใช้ในการโทรศัพท์ออก ซึ่งไม่สามารถรองรับข้อมูลปริมาณมาก ๆ ได้ แต่ถึงอย่างไรก็ยังเพียงพอต่อการทดสอบอย่างแน่นอน

1.4 วิธีการดำเนินงาน

โครงงานนี้จะเริ่มด้วยการศึกษาทฤษฎีพื้นฐานต่าง ๆ ที่เกี่ยวข้อง ซึ่งก็มีเรื่องหลัก ๆ อยู่ 5 เรื่องด้วยกัน คือ TAPI 3.0, DirectShow, SAPI 5.0, Winsock 2 และ FTP ซึ่งมีรายละเอียดดังในบทที่ 4, 5, 6 และ 7 จากนั้นก็จะนำเอาความรู้ที่ได้ศึกษาทั้งหมดมาออกแบบสถาปัตยกรรมของระบบ ซึ่งมีรายละเอียดในบทที่ 8 และออกแบบฐานข้อมูล จากนั้นก็จะเริ่มเข้าสู่ขั้นตอนของการพัฒนาโปรแกรม ซึ่งกล่าวถึงองค์ประกอบโดยรวมของระบบที่พัฒนาขึ้นมาทั้งหมด และยังอธิบายไปถึงรูปแบบการติดต่อกับผู้ใช้ และการประมวลผลต่าง ๆ จากนั้นจะอธิบายแต่ละฟังก์ชันการทำงาน โดยเริ่มจากการเข้าลงทะเบียนเป็นผู้ใช้รายใหม่ การเข้าใช้งานของผู้ใช้เดิม การเพิ่มเครดิตในการโทร การเรียกดูรายการโทรของผู้ใช้ และการเพิ่มรายการในการโทร

สำหรับบทที่ 9 ซึ่งเป็นบทสุดท้าย จะเป็นการทำการทดลองของระบบโดยรวมทั้งหมด และเป็นการสรุปการทำงาน ผลที่ได้รับจากโครงงานนี้ และแนวทางในการพัฒนาโครงงานนี้เพิ่มเติม และแนวทางในการนำไปประยุกต์ใช้

บทที่ 2
ความรู้พื้นฐานของ Component Object Model (COM)

2.1 บทนำ

COM เกิดขึ้นมาใน ค.ศ.1988 เมื่อทีมของไมโครซอฟท์เริ่มที่จะสร้างโครงสร้างพื้นฐานของออบเจ็กต์-โอเรียนเต็ดที่สามารถนำคอมโพเน็นต์กลับมาใช้งานใหม่ได้ เมื่อกลุ่มของวิศวกรได้รวมตัวขึ้นมาจากทีมอื่นๆ เพื่อช่วยในโครงการของ OLE2 นั้น ก็ได้ตัดสินใจในการร่างความต้องการในระดับสูงสำหรับสถาปัตยกรรมใหม่ที่จะสร้างขึ้นมา

ในที่สุดวิศวกรของไมโครซอฟท์เหล่านี้ก็กำหนดความต้องการในระดับสูงอยู่ซึ่งมีด้วยกัน 4 ความต้องการสำหรับสถาปัตยกรรมคอมโพเน็นต์ใหม่นี้ อันแรกสถาปัตยกรรมจะเป็นคอมโพเน็นต์เพราะการบำรุงรักษาและปรับปรุงโค้ดในแอพพลิเคชันที่มีขนาดใหญ่จะกระทำได้ยาก ระบบที่อยู่บนพื้นฐานของคอมโพเน็นต์ที่เป็นไบนารีจะง่ายกว่ามากในการประกอบ บำรุงรักษาและขยายระบบ

อย่างที่สองคือ สถาปัตยกรรมจะอยู่บนพื้นฐานของ

การเรียนรู้ของ COM ในบทนี้จะเป็นแนวทางในการทำความเข้าใจในเรื่องต่างๆ ในบทถัดไปได้ดียิ่งขึ้น เนื่องจากเทคโนโลยีในที่ได้นำมาใช้ในโครงงานนี้ ส่วนใหญ่เป็นเทคโนโลยีที่เป็นรูปแบบของ COM ทำให้สามารถนำไปใช้งานได้กับหลายภาษาของการเขียนโปรแกรม แต่ก็ยังมีข้อจำกัดที่ว่าบางภาษาไม่สามารถสนับสนุนรูปแบบของ COM ได้อย่างเต็มที่ ทำให้การนำเทคโนโลยีเหล่านั้นมาใช้งานจริงยังต้องอิงอยู่เฉพาะบางภาษาเท่านั้น ซึ่งภาษานั้นก็คือ C++

2.2 ความหมายของ COM

COM เป็นระบบที่ไม่ขึ้นกับระบบปฏิบัติการใดๆ และเป็นระบบออบเจ็กต์-โอเรียนเต็ดแบบกระจายสำหรับการสร้างคอมโพเน็นต์ของซอฟต์แวร์ที่สามารถมีผลกระทบต่อกันได้ COM ถือเป็นเทคโนโลยีสำหรับ OLE (compound documents) ของไมโครซอฟท์และเทคโนโลยี ActiveX®

เพื่อที่จะให้เข้าใจในเทคโนโลยี COM จึงควรจะเข้าใจว่า COM ไม่ใช่เป็นภาษาออบเจ็กต์-โอเรียนเต็ด แต่เป็นมาตรฐานอย่างหนึ่ง และ COM ก็ไม่ใช่เป็นการนิยามว่าแอพพลิเคชันควรจะถูกสร้างอย่างไร ซึ่งรายละเอียดของภาษา โครงสร้างและการอิมพลีเมนต์ต้องให้โปรแกรมเมอร์ของแอพพลิเคชันนั้นเป็นคนจัดการ COM ถือเป็นตัวระบุโมเดลของออบเจ็กต์และกำหนดความต้องการของการเขียนโปรแกรมเพื่อทำให้ออบเจ็กต์ COM (อาจจะเรียกว่า คอมโพเน็นต์ COM หรือบางทีเรียกสั้นๆ ว่า ออบเจ็กต์) สามารถติดต่อกับออบเจ็กต์อื่นๆ ได้ ออบเจ็กต์ COM เหล่านี้สามารถอยู่ภายในโพรเซสเดียวกัน ต่างโพรเซส หรือแม้แต่อยู่ที่เครื่องรีโมตก็ได้ ออบเจ็กต์ของ COM สามารถเขียนขึ้นได้จากหลายภาษาซึ่งอาจจะมีโครงสร้างที่ไม่เหมือนกัน COM จึงถูกเรียกได้ว่าเป็น มาตรฐานไบนารี ซึ่งหมายถึง มาตรฐานที่ประยุกต์ใช้หลังจากที่โปรแกรมได้ถูกแปลงเป็นโค้ดไบนารีของภาษาเครื่อง (binary machine code)

ความต้องการทางด้านภาษาของการเขียนโปรแกรมของ COM มีเพียงอยู่ว่าโค้ดที่ถูกสร้างขึ้นมาในภาษาหนึ่งต้องมีความสามารถสร้างโครงสร้างของพอยน์เตอร์และสามารถเรียกฟังก์ชันแบบชัดแจ้ง (explicit) หรือแบบโดยนัย (implicit) โดยผ่านพอยน์เตอร์เหล่านั้นได้ ภาษาที่เป็นออบเจ็กต์-โอเรียนเต็ดอย่างเช่น C++ และ Smalltalk ทำให้กลไกเขียนโปรแกรมในการอิมพลีเมนต์ออบเจ็กต์ COM เป็นไปได้โดยง่าย แต่โดยพื้นเพของภาษาอย่างเช่น C, Pascal, Ada, Java และแม้แต่ภาษา BASIC ก็สามารถสร้างและใช้ออบเจ็กต์ COM ได้เช่นกัน ซึ่งโดยหลักแล้ว COM ถูกออกแบบมาใช้กับภาษา C++ และ Visual Basic

2.3 ออบเจ็กต์และอินเทอร์เฟซของ COM

โดยทั่วไปแล้ว ออบเจ็กต์ของซอฟต์แวร์หนึ่งถูกสร้างขึ้นจากเซตของข้อมูลและฟังก์ชันที่จัดการกับข้อมูลเหล่านั้น โดยออบเจ็กต์ของ COM สามารถเข้าถึงข้อมูลของออบเจ็กต์โดยผ่านทางกลุ่มของฟังก์ชันที่เกี่ยวข้อง กลุ่มของฟังก์ชันเหล่านี้ถูกเรียกว่า อินเทอร์เฟซ (interfaces) และฟังก์ชันของอินเทอร์เฟซถูกเรียกว่า เมธอด (methods) และวิธีเดียวที่จะสามารถเข้าถึงเมธอดของอินเทอร์เฟซได้คือต้องผ่านทางพอยน์เตอร์ของอินเทอร์เฟซนั้น

นอกเหนือจากการที่เป็นตัวกำหนดมาตรฐานของออบเจ็กต์ไบนารีแบบพื้นฐานแล้ว COM ยังกำหนดอินเทอร์เฟซพื้นฐานที่มีฟังก์ชันซึ่งได้ใช้กับเทคโนโลยีที่อ้างอิงถึง COM ได้ทั้งหมด และยังมีฟังก์ชัน API ที่คอมโพเน็นต์ทุกตัวต้องการให้ได้เรียกใช้อีกด้วย ในปัจจุบัน COM ได้ขยายขีดความสามารถให้ออบเจ็กต์สามารถทำงานร่วมกันผ่านทางสภาพแวดล้อมที่เป็นแบบกระจายได้ และเพิ่มความสามารถทางด้านความปลอดภัยเพื่อให้แน่ใจในความถูกต้องของตัวระบบและคอมโพเน็นต์

2.3.1 อินเทอร์เฟซและการอิมพลีเมนต์ของอินเทอร์เฟซ

COM ทำการแยกแยะโดยพื้นฐานระหว่างนิยามของอินเทอร์เฟซและการอิมพลีเมนต์ของมัน อินเทอร์เฟซโดยแท้จริงแล้วเป็นการทำสัญญาที่ประกอบไปด้วยกลุ่มของโพรโตไทป์ (prototype) ของฟังก์ชันที่เกี่ยวข้องซึ่งได้นิยามถึงวิธีการใช้แต่ไม่ได้นิยามถึงการอิมพลีเมนต์ โพรโตไทป์ของฟังก์ชันเหล่านี้เทียบได้กับคลาสพื้นฐานของการเขียนโปรแกรมใน C++ นิยามของอินเทอร์เฟซได้ระบุถึงสมาชิกฟังก์ชันของอินเทอร์เฟซหรือที่เรียกว่า เมธอด ชนิดของค่าที่คืนกลับ จำนวนและชนิดของพารามิเตอร์ของมัน และสิ่งที่มันต้องทำ ดังนั้นการอิมพลีเมนต์จึงไม่เกี่ยวข้องกับอินเทอร์เฟซเลย

การอิมพลีเมนต์อินเทอร์เฟซถือเป็นโค้ดที่โปรแกรมเมอร์ได้สร้างการทำงานภายในที่ระบุไว้ตามนิยามของอินเทอร์เฟซนั้น การอิมพลีเมนต์ของหลายๆ อินเทอร์เฟซที่โปรแกรมเมอร์สามารถใช้งานในแอพพลิเคชันแบบออบเจ็กต์-โอเรียนเต็ดได้ถูกรวมไว้ในไลบรารีของ COM แล้ว โปรแกรมเมอร์สามารถที่จะเลือกไม่ใช้อินเทอร์เฟซเหล่านี้แล้วเขียนเองขึ้นมาใหม่ได้ การอิมพลีเมนต์อินเทอร์เฟซจะเริ่มเกี่ยวข้องกับออบเจ็กต์เมื่อมีการสร้างอินสแทนซ์ของออบเจ็กต์นั้นขึ้น และทำการจัดหาบริการต่างๆ ตามที่ออบเจ็กต์ได้เสนอมา

COM ใช้คำว่าอินเทอร์เฟซในแง่ที่แตกต่างจากที่ใช้ในการเขียนโปรแกรม C++ ทั่วไป อินเทอร์เฟซของ C++ จะหมายถึงฟังก์ชันทั้งหมดที่คลาสสนับสนุนและด้านไคลเอนต์ของออบเจ็กต์สามารถเรียกใช้งานได้ ส่วนอินเทอร์เฟซของ COM หมายถึงกลุ่มที่ได้นิยามไว้แล้วของฟังก์ชันที่เกี่ยวข้องที่มีคลาสของ COM เป็นตัวอิมพลีเมนต์ แต่ไม่จำเป็นที่จะต้องหมายถึงฟังก์ชันทั้งหมดที่คลาสนั้นสนับสนุน ซึ่งภาษาจาวาจะเหมือนกับอินเทอร์เฟซของ COM เพราะจาวาทำการนิยามอินเทอร์เฟซในลักษณะเดียวกันนี้

2.3.2 อินเทอร์เฟซของพอยน์เตอร์และอินเทอร์เฟซต่างๆ

อินสแทนซ์ของการอิมพลีเมนต์อินเทอร์เฟซเป็นพอยน์เตอร์ตัวหนึ่งที่ชี้ไปยังอาร์เรย์ของพอยน์เตอร์ของเมธอดต่างๆ (ซึ่งเป็นตารางของฟังก์ชันที่อ้างอิงถึงการอิมพลีเมนต์เมธอดทั้งหมดที่ถูกระบุในอินเทอร์เฟซ) ออบเจ็กต์ที่มีหลายอินเทอร์เฟซสามารถมีพอยน์เตอร์ที่ชี้ไปที่ตารางของฟังก์ชันได้มากกว่าหนึ่งตาราง โค้ดใดก็ตามที่มีพอยน์เตอร์ที่สามารถเข้าถึงตารางนั้นได้ก็จะสามารถเรียกเมธอดในอินเทอร์เฟซนั้นได้

พอยน์เตอร์ที่ชี้ไปยังตารางฟังก์ชันของอินเทอร์เฟซที่ออบเจ็กต์ตัวอื่นต้องทำการเรียกใช้เมธอดของมันนั้นจะเรียกได้ว่าเป็น พอยน์เตอร์ของอินเทอร์เฟซ (interface pointer) เราสามารถที่จะสร้างตารางของฟังก์ชันในแอพพลิเคชันของภาษา C หรือเกือบจะเป็นแบบอัตโนมัติในภาษา C++ (หรือจะเป็นภาษาออบเจ็กต์-โอเรียนเต็ดอื่นๆ ที่สนับสนุน COM)

ด้านไคลเอนต์ที่มีคอมไพเลอร์สนับสนุนได้อย่างเหมาะสมสามารถเรียกใช้เมธอดของอินเทอร์เฟซโดยผ่านชื่อของมัน ซึ่งไม่ใช่ตำแหน่งในอาร์เรย์นั้น เนื่องจากอินเทอร์เฟซเป็นรูปแบบ การให้ชื่อของเมธอดทำให้คอมไพเลอร์สามารถเช็คชนิดของพารามิเตอร์และค่าที่คืนกลับของแต่ละการเรียกใช้เมธอดของอินเทอร์เฟซได้ ในทางกลับกัน การเช็คชนิดต่างๆ จะไม่มีแม้แต่ในภาษา C หรือ C++ หากไคลเอนต์ใช้วิธีการเรียกใช้แบบตำแหน่ง (position-based calling scheme)

อินเทอร์เฟซแต่ละตัวถูกอ้างอิงในขณะรันไทม์ด้วยชื่อเฉพาะที่เรียกว่า ตัวบ่งชี้อินเทอร์เฟซ หรือ IID (interface identifier) ตัว IID นี้เป็นอินสแทนซ์เฉพาะตัวของ GUID (Globally Unique Identifier) ซึ่ง IID จะอนุญาตให้ไคลเอนต์สามารถสอบถามถึงออบเจ็กต์ได้หากมันสนับสนุนความหมายของคำ (semantic) ของอินเทอร์เฟซโดยที่ไม่มีโอเวอร์เฮด (overhead) ที่ไม่จำเป็นและปราศจากความสับสนที่เกิดขึ้นในระบบจากการที่มีอินเทอร์เฟซตัวเดียวกันชื่อเดียวกันซึ่งมีหลายเวอร์ชัน

เพื่อให้เข้าใจในความหมายของอินเทอร์เฟซ สามารถสรุปได้ดังนี้

- อินเทอร์เฟซไม่ได้เป็นตัวเดียวกันกับคลาสของ C++
- อินเทอร์เฟซไม่ใช่ออบเจ็กต์
- อินเทอร์เฟซเป็นชนิด strongly typed ซึ่งแต่ละอินเทอร์เฟซจะมี GUID เป็นของตนเอง
- อินเทอร์เฟซไม่สามารถเปลี่ยนรูปได้ (Immutable)

2.3.3 IUnknown และการสืบทอดของอินเทอร์เฟซ

การสืบทอด (Inheritance) ใน COM ไม่ได้หมายความว่าเป็นการนำโค้ดกลับมาใช้ใหม่ เพราะว่าไม่ได้มีการอิมพลีเมนต์ที่เกี่ยวข้องกับอินเทอร์เฟซเลย ซึ่งการสืบทอดของอินเทอร์เฟซก็ไม่ได้หมายความว่าเป็นการสืบทอดของโค้ดด้วยเช่นกัน แต่มันหมายถึงสัญญาที่เกี่ยวข้องกับอินเทอร์เฟซนั้นได้ถูกทำการสืบทอดใน C++ โดยการเพิ่มเมธอดใหม่เข้าไปหรือจากการคัดเลือกการใช้เมธอดที่สามารถให้ใช้ได้ ซึ่งจะไม่สามารถเลือกทำการสืบทอดใน COM ได้ ถ้าอินเทอร์เฟซหนึ่งทำการสืบทอดมาจากอีกอินเทอร์เฟซหนึ่ง อินเทอร์เฟซตัวใหม่นั้นจะมีเมธอดทั้งหมดที่อินเทอร์เฟซตัวเก่านั้นนิยามไว้ด้วย

การสืบทอดถูกใช้กับอินเทอร์เฟซที่ได้นิยามไว้แล้วของ COM ซึ่งเป็นส่วนน้อย อินเทอร์เฟซที่ได้นิยามไว้แล้วนี้ (รวมถึงอินเทอร์เฟซที่ได้นิยามขึ้นมาภายหลัง) สืบทอดนิยามของมันเองจากอินเทอร์เฟซที่สำคัญที่เรียกว่า IUnknown ซึ่งประกอบไปด้วย 3 เมธอดคือ QueryInterface, AddRef และ Release ออบเจ็กต์ COM ทุกตัวจะต้องทำการอิมพลีเมนต์อินเทอร์เฟซ IUnknown นี้ เพราะว่ามันทำให้สามารถเคลื่อนไหวไปมาได้อย่างอิสระระหว่างอินเทอร์เฟซที่แตกต่างกันซึ่งออบเจ็กต์สนับสนุนด้วยเมธอดทั้ง 3 ตัวนั้น

ในการสร้างออบเจ็กต์ที่สนับสนุนการทำ Aggregation จะต้องอิมพลีเมนต์กลุ่มฟังก์ชันของ IUnknown สำหรับทุกอินเทอร์เฟซรวมถึงอินเทอร์เฟซ IUknown ที่เป็น stand-alone ด้วย และตัวอิมพลีเมนต์ออบเจ็กต์จะทำการอิมพลีเมนต์เมธอด IUnknown ในทุกๆ กรณี

เมื่อมีอินเทอร์เฟซบางตัวที่สืบทอดคุณสมบัติของมันจากอินเทอร์เฟซอีกตัวหนึ่ง โดยส่วนมากจะเป็นเมธอดของตัวอินเทอร์เฟซ IUnknown และเมธอดที่ได้นิยามในอินเทอร์เฟซนั้น ทำให้อินเทอร์เฟซส่วนใหญ่มีขนาดกระทัดรัดและง่ายต่อการห่อหุ้มไว้ (encapsulate)

บทที่ 3
IP Telephony

3.1 บทนำ

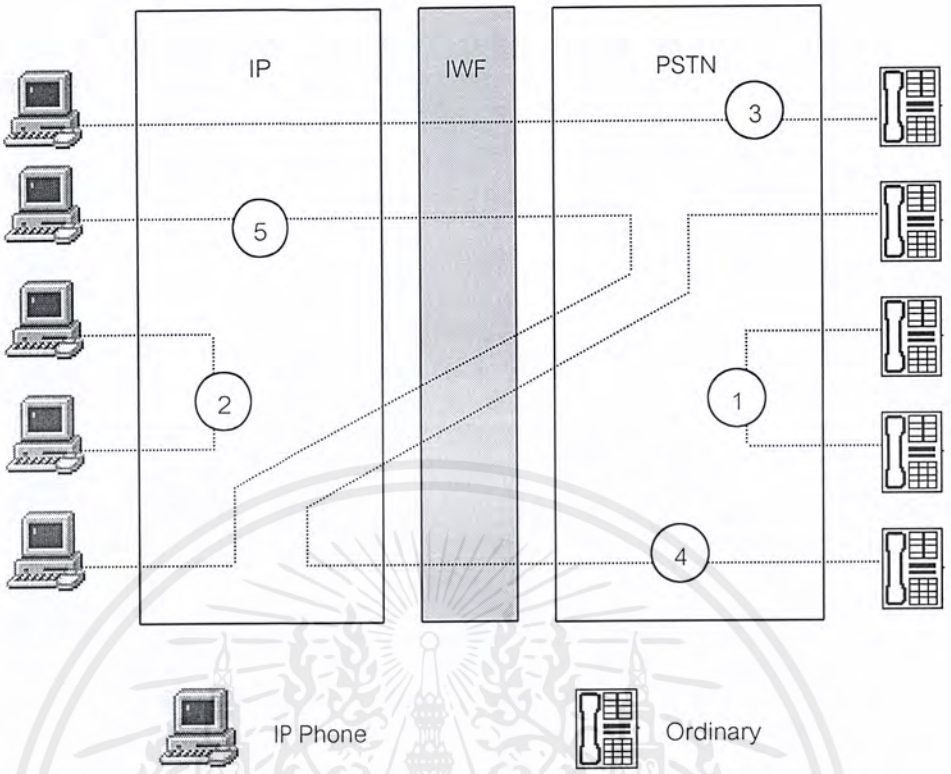
Internet Engineering Task Force (TETF) เป็นองค์กรที่ทำงานด้านการให้บริการไอพีเทเลโฟนนี่ โดยนำมาผนวกเข้ากับระบบ PSTN ในภายหลัง โดยร่วมกับองค์กร International Telecommunication Union – Telecommunication Standardization Sector (ITU-T) ที่ควบคุมและกำหนดมาตรฐานทางด้านระบบเครือข่ายโทรศัพท์ PSTN อยู่ ได้ร่วมกันกำหนดมาตรฐานการติดต่อระหว่างระบบไอพีและระบบโทรศัพท์ขึ้น ต่อมาสมาคมที่เกี่ยวข้องกับทางด้านนี้เช่น International Multimedia Teleconferencing Consortium (IMTC) ผ่านทางกลุ่มของ VoIP (Voice over IP) ได้สนับสนุนให้ควรมีความสามารถในการติดต่อใช้งานระหว่างผู้ผลิตหลายรายได้ด้วย

Vocaltec ถือเป็นบริษัทเป็นแห่งแรกที่ได้เริ่มพัฒนาซอฟต์แวร์ด้านอินเทอร์เน็ตเทเลโฟนนี่เมื่อต้นปีค.ศ. 1995 ทำให้ผู้ใช้ที่มีเครื่องพีซี (Personal Computer) พร้อมกับมีระบบมัลติมีเดียสามารถพูดสื่อสารกันผ่านไมโครโฟนและได้ยินเสียงผ่านทางลำโพงได้ ในปีต่อๆ มา จากที่ผู้ใช้เริ่มนิยมใช้งานเพิ่มมากขึ้น ทำให้บริษัทอื่นๆ เริ่มเห็นความสำคัญและได้พัฒนาซอฟต์แวร์ให้มีความสามารถในการใช้เทคโนโลยีอินเทอร์เน็ตเทเลโฟนนี่ด้วย

ไมโครซอฟท์ถือเป็นบริษัทยักษ์ใหญ่ในวงการคอมพิวเตอร์ที่ได้มีส่วนร่วมในการพัฒนาซอฟต์แวร์ประเภทนี้ด้วยเช่นกัน ดังที่ได้เห็นจากโปรแกรม NetMeeting ซึ่งได้ประยุกต์ใช้ TAPI (Telephony Application Programming Interface) หลังจากนั้นบริษัทไมโครซอฟท์ได้ออก Microsoft Windows 2000 มาพร้อมกับเทคโนโลยีทางด้านเทเลโฟนนี่ด้วย ซึ่งปัจจุบันได้พัฒนาถึง TAPI เวอร์ชัน 3.0 แล้ว

3.2 รูปแบบการสื่อสารที่ใช้ไอพีเทเลโฟนนี่

รูปแบบในการสื่อสารด้วยเสียงผ่านระบบเครือข่ายไอพีและระบบเครือข่ายโทรศัพท์มีทั้งหมด 5 แบบดังรูปที่ 3-1 ซึ่งจะมีเฉพาะแบบที่ 3, 4 และ 5 เท่านั้นที่มีการผสมผสานระหว่างสองระบบเครือข่าย หรือที่เรียกว่า Hybrid Voice Services ซึ่งจะต้องใช้ InterWorking Function (IWF) ซึ่งมีหน้าที่แปลงโพรโตคอลทั้งหมดและดัดแปลงข้อมูลให้เหมาะสมกับโพรโตคอลนั้นๆ โดยมีเกตเวย์ (gateway) เป็นอุปกรณ์ที่บรรจุ IWF ไว้



รูปที่ 3-1 แสดง scenarios ต่างๆ ของการติดต่อสื่อสารด้วยเสียง

แบบที่ 1 เป็นการสื่อสารระหว่างโทรศัพท์ 2 เครื่องโดยผ่านระบบ PSTN ซึ่งเป็นเครือข่ายแบบ circuit-switched ที่มีความสามารถในการส่งข้อมูลด้วยความเร็ว 64 กิโลบิตต่อวินาที เป็นโทรศัพท์พื้นฐานที่ใช้กันตามบ้านเรือนและบริษัททั่วไป

แบบที่ 2 หรือที่เรียกกันว่า PC-to-PC เป็นการใช้เทคโนโลยี Voice over IP ระหว่างเครื่องคอมพิวเตอร์สองเครื่องผ่านระบบเครือข่ายที่เป็นไอพี เช่น อินเทอร์เน็ต ในการประมวลผลของสัญญาณและข้อมูลของมีเดียจะถูกกระทำที่เครื่องคอมพิวเตอร์ปลายทาง ซึ่งบางส่วนของแอปพลิเคชั่นจะอยู่บนชั้นของ TCP หรือ UDP ก็ได้ ลำดับของแพ็กเก็ตที่ส่งในเครือข่ายอาจจะไม่ได้รับเรียงตามลำดับที่ปลายทาง เนื่องจากข้อมูลเป็นสัญญาณเสียง ทำให้มีการนำ RTP (Real-time Protocol) มาใช้ เพื่อที่จะเรียงลำดับตามหมายเลขที่กำหนดขึ้นในแพ็กเก็ตทางด้านผู้รับ

แบบที่ 3 หรือที่เรียกกันว่า PC-to-Phone ซึ่งปลายทางทั้งสองด้านใช้โพรโตคอลในการติดต่อกับเครือข่ายที่ต่างกัน ดังนั้นจึงต้องมีการแปลงสัญญาณข้อมูลระหว่างเครือข่ายไอพีและ PSTN โดยจะต้องแมบทั้งข้อมูล ช่องทางการควบคุมข้อมูล รวมทั้งโพรโตคอลของสัญญาณที่เชื่อมต่อกัน

ส่วนในแบบที่ 4 (Phone-to-Phone) และแบบที่ 5 นั้น มีลักษณะการใช้โพรโตคอลในส่วนของอินเทอร์เฟซที่เหมือนกัน แต่มีโพรโตคอลหลักต่างกัน

3.3 มาตรฐานที่ใช้ในไอพีเทเลโฟนนี่

เนื่องจากแต่ละเครือข่ายมีรูปแบบและมาตรฐานที่แตกต่างกัน ทำให้เรื่องของการติดต่อข้ามเครือข่าย (Internetworking) ได้ถูกนำมาพิจารณาและทำให้ง่ายขึ้นโดยการแนะนำมาตรฐานสากลสำหรับเทเลโฟนนี่ ซึ่งโดยปกติแล้ว International Telecommunication Union (ITU) จะเป็นผู้ประกาศออกมา ตัวของมาตรฐานเองจะทำหน้าที่สำคัญสองประการคือ ช่วยจำกัดจำนวนของระบบโทรศัพท์ที่ต่างประเภทกัน และอีกประการหนึ่งคือ เพื่อทำการนิยามลักษณะสำคัญสำหรับระบบโทรศัพท์ที่ใช้ในการติดต่อ ส่วนระบบไอพีเทเลโฟนนี่จะขยายความจากที่ ITU ครอบคลุมไว้และเพิ่มความซับซ้อนเข้าไปอีก ระบบไอพีเทเลโฟนนี่ใดๆ จะต้องถูกเชื่อมต่อเข้ากับเครือข่ายโทรศัพท์พื้นฐานหรือที่เป็นมาตรฐานทั่วไปอย่างเช่น PSTN เป็นต้น

ITU's Telecommunications Subcommittee (ITU-T) ได้กำหนดรูปแบบของมาตรฐานสำหรับฟังก์ชันต่างๆ ในไอพีเทเลโฟนนี่ ตั้งแต่เรื่องของการเข้ารหัสสัญญาณเสียง การเชื่อมการติดต่อ การเราท์ (routing) ของเส้นทาง และแม้แต่การขยายการเชื่อมต่อไปยังเครือข่าย PSTN ทั่วๆ ไป

3.3.1 มาตรฐาน H.323

H.323 เป็นมาตรฐานที่ ITU รับรองในการส่งข้อมูลมัลติมีเดียบนระบบเครือข่ายที่ไม่มีการรับรองของ QoS (Quality of Service) ดังเช่นในระบบเครือข่ายไอพีและอินเทอร์เน็ต H.323 จะจัดการในเรื่องของการควบคุมการ call การจัดการมัลติมีเดีย และการจัดการแบนด์วิดธ์ของเครือข่ายให้ ซึ่งเป็นมาตรฐานที่ไม่ขึ้นกับระบบเครือข่าย แพลตฟอร์มและแอปพลิเคชั่นใดๆ

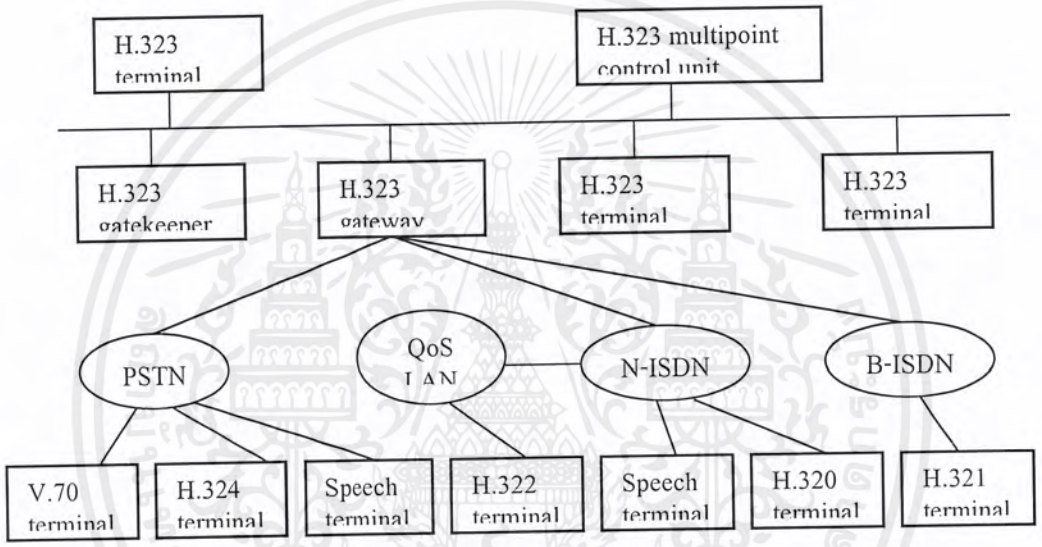
มาตรฐาน H.323 นี้ได้ถูกกำหนดขึ้นครั้งแรกในช่วงต้นปี ค.ศ. 1996 และครั้งสุดท้ายที่ออกมาในเดือนกุมภาพันธ์ ปี ค.ศ. 1998 เพื่อกำหนดลักษณะและคุณสมบัติในการหาหนทางติดต่อระหว่างระบบเครือข่ายต่างๆ ที่ไม่เพียงแต่รับและส่งสัญญาณเสียงให้กันและกันเท่านั้น แต่ยังรวมไปถึงการส่งข้อมูลมัลติมีเดียซึ่งเป็นทั้งภาพและเสียงที่เคลื่อนไหวด้วย แต่ในความเป็นจริงแล้วมาตรฐานนี้ได้ถูกใช้โดยเน้นทางด้านเสียงมากกว่าซึ่งทำให้ VoIP สามารถเป็นจริงขึ้นมาได้ H.323 จะใช้ Real-time Transport Protocol (RTP) ที่องค์กรมาตรฐาน IETF กำหนดขึ้นเพื่อรองรับความต้องการของการส่งเสียงและวิดีโอแบบเรียวไทม์อย่างต่อเนื่องบนเครือข่ายอินเทอร์เน็ต คอมโพเน็นต์ที่จำเป็นในระบบการสื่อสารที่ใช้มาตรฐาน H.323 ประกอบด้วย 4 อย่างดังนี้

1. เทอร์มินอล (Terminals) เป็นจุดปลายทางของไคลเอนต์บนเครือข่าย ซึ่งจะต้องสนับสนุนการติดต่อสื่อสารด้วยเสียงได้
2. เกตเวย์ (Gateway) เป็นส่วนเพิ่มเติมที่ใช้สำหรับเชื่อมต่อเครือข่ายที่สนับสนุน H.323 กับเครือข่าย โพรโตคอล และรูปแบบของมัลติมีเดียอื่นๆ หากไม่มีการเชื่อมต่อข้ามเครือข่ายต่างชนิดก็ไม่ต้องใช้เกตเวย์

3. เกตคีบเปอร์ (Gatekeeper) ทำหน้าที่สำคัญสองประการคือ ช่วยในการแปลงแอดเดรสและจัดการกับแบนด์วิดธ์ ซึ่งตัวเกตคีบเปอร์ไม่จำเป็นต้องมีก็ได้ หากไม่มีแล้วอุปกรณ์ทุกตัวจะต้องสามารถสร้างสัญญาณโดยตรงเองได้
4. Multipoint Control Units (MCU) สนับสนุนการประชุมที่มีตั้งแต่ 3 จุดเป็นต้นไป

3.3.1.1 สถาปัตยกรรมของ H.323

เนื่องจาก VoIP ได้ใช้บางส่วนของสถาปัตยกรรมของ H.323 เราจึงควรทราบระบบของ H.323 ทั้งหมดอย่างคร่าวๆ เสียก่อน ซึ่งสถาปัตยกรรมของ H.323 ทั้งหมดแสดงได้ดังในรูปที่ 3-2



รูปที่ 3-2 สถาปัตยกรรมของ H.323

จากรูปจะเห็นว่าสามารถใช้ H.323 บน LAN (Local Area Network) หรือ WAN (Wide Area Network) ได้ ด้านบนของรูปแสดงเครือข่ายแลนที่ประกอบไปด้วยอุปกรณ์ H.323 ทั้ง 4 ตัว ผู้ใช้จะมีเทอร์มินอล H.323 ซึ่งก็คือเครื่องพีซีมัลติมีเดียธรรมดาที่สามารถใช้ประโยชน์จากคุณสมบัติของ H.323 ได้ ซึ่งรวมถึงการประชุมทางไกลหลายๆ แห่ง (multipoint video conference) การสื่อสารกันหลายๆ จุดจะใช้ MCU เป็นตัวควบคุม H.323 มีความสามารถที่จะใช้ในเครือข่าย WAN ได้ด้วยตราบใดที่การเชื่อมต่อกันเป็นอุปกรณ์ที่สนับสนุน H.323 ทั้งหมด ซึ่งนี่ก็คือหน้าที่หลักของเกตคีบเปอร์

โดยทางเทคนิคแล้ว อะไรก็ตามที่อยู่หลังจากเกตเวย์ของ H.323 จะไม่ถืออยู่ในส่วนของมาตรฐาน H.323 แต่ตัวเกตเวย์ของ H.323 สามารถทำการติดต่อกับอุปกรณ์ชนิดต่างๆ บนเครือข่ายชนิดต่างๆ กันได้ H.323 สามารถใช้กับ PSTN, narrowband ISDN (N-ISDN), broadband ISDN (B-ISDN) และแม้แต่เทอร์มินอลที่เป็นโทรศัพท์หรือเทอร์มินอลที่เป็น Speech ก็สามารถใช้ H.323 ได้ แต่เป็นเพียงแค่สัญญาณเสียงเท่านั้น โดยสรุปแล้ว H.320 เป็นตัวกำหนดชนิดของเทอร์มินอลที่แตกต่างกันทั้ง 4 ชนิดคือ เทอร์มินอล H.321 สำหรับ B-ISDN และ ATM เทอร์มินอล H.322 สำหรับ QoS LAN เทอร์มินอล H.323

สำหรับการคอนเฟอเรนซ์ (conference) และในส่วนอื่นๆ อีก ส่วนเทอร์มินอล H.324 ใช้สำหรับหมุนเบอร์โทรติดต่อที่ความเร็ว 33.6 kb/s

3.3.1.2 มาตรฐาน H.323 ในไอพีเทเลโฟนนี่

โครงสร้างสถาปัตยกรรมทั้งหมดของ H.323 ใหญ่เกินไปสำหรับ VoIP หรือแม้แต่เครือข่ายไอพีเทเลโฟนนี่ ซึ่งเป็นเพียงบางส่วนของ H.323 เท่านั้นที่ถูกใช้ในเครือข่ายไอพี ดังนั้นเมื่อใช้กับไอพีเทเลโฟนนี่จะมีเพียงบางคอมโพเน็นต์ของ H.323 ที่ถูกใช้ จากสแต็กทั้งหมดของโพรโตคอล H.323 ดังรูปที่ 3-3 ในส่วนที่เป็นแรเงา

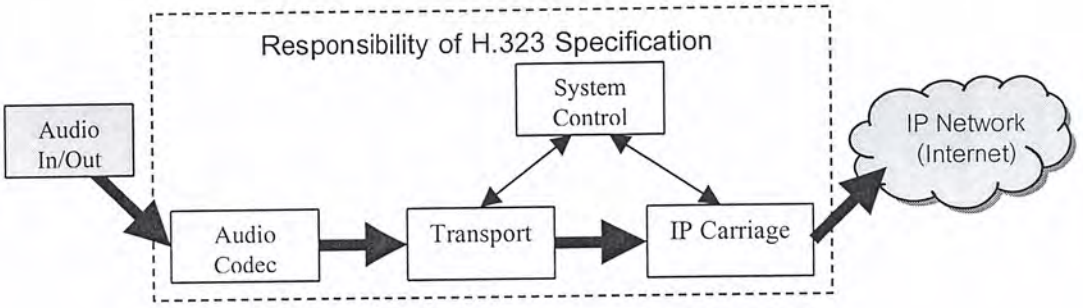
<table>
<tr><td colspan="2">Video</td><td colspan="2">Audio</td><td colspan="3">Control</td><td>Data</td></tr>
<tr><td colspan="2">H.261
H.263
(video coding)</td><td colspan="2">G.711
G.722
G.723
G.728
G.729</td><td>H.225

Terminal to gatekeeper signaling</td><td>H.225
Call signaling</td><td>H.245</td><td>T.120
(Multipoint data transfer)</td></tr>
<tr><td>RTP</td><td>RTCP</td><td>RTP</td><td>RTCP</td><td colspan="3"></td><td></td></tr>
<tr><td colspan="4">Unreliable transport (UDP)</td><td colspan="3">Reliable transport (TCP)</td><td></td></tr>
</table>

รูปที่ 3-3 โพรโตคอล H.323 สำหรับไอพีเทเลโฟนนี่

จะเห็นได้ว่าเป็นเพียงแค่ในส่วนของสัญญาณเสียงและการควบคุมที่ใช้ในไอพีเทเลโฟนนี่ สแต็กของสัญญาณเสียงจะจัดการในเรื่องของฟังก์ชันใน VoIP ทั้งหมด และสัญญาณเสียงที่ต้องการใช้สำหรับ H.323 ก็คือ G.711 (64 kb/s) เท่านั้น แต่ในรูปแบบ VoIP โดยส่วนใหญ่จะมีการอิมพลีเมนต์สัญญาณเสียงดิจิตอลที่ความเร็วต่ำด้วย อาจจะเป็น G.728 (16 kb/s), G.723 (5.3 หรือ 6.4 kb/s) หรือ G.729 (8 kb/s) เนื่องจากมีความล่าช้าที่เกิดขึ้นจากการตรวจสอบความผิดพลาดและอาจจะมีการส่งข้อมูลอีกรอบของ TCP (Transmission Control Protocol) จึงได้มีการนำเอา UDP (User Datagram Protocol) มาใช้ โดย H.323 ได้กำหนดให้เพิ่มเฮดเดอร์ของ RTP (Real Transport Protocol) ไว้ในข้อมูลของ UDP ด้วย ซึ่งบางอิมพลีเมนต์จะใช้ RTCP (Real-Time Control Protocol) ในการจัดการคุณภาพของเสียงบนเครือข่ายในทำนองเดียวกับ RTP

จากรูปที่ 3-4 แสดงให้เห็นว่า H.323 ไม่ได้ไปข้องเกี่ยวกับกลไกการทำงานของสัญญาณเสียงที่เข้าไปยังระบบ VoIP ที่สนับสนุน H.323 หรือไม่ได้มีหน้าที่จัดหาในการทำให้ได้ยินข้อมูลเสียงเมื่อข้อมูลมาถึงปลายทางแล้ว หรือแม้แต่ไม่ได้เป็นตัวกำหนดคุณสมบัติต่างๆ ของเครือข่ายที่แพ็กเก็ตเสียงไหลผ่านหรือที่ตัวอินเทอร์เฟซของเครือข่ายระหว่างเทอร์มินอลเสียงกับเครือข่ายนั้น แต่สิ่งที่ H.323 และอุปกรณ์ VoIP ส่วนใหญ่ต้องทำนั้นได้แสดงไว้ที่รูปในกรอบสี่เหลี่ยมเส้นประแล้ว



รูปที่ 3-4 ฟังก์ชันต่างๆ ของโพรโตคอลในระบบ VoIP

ภายใต้หน้าที่และความรับผิดชอบอื่นๆ H.323 กำหนดบทบาทของคอมโพเน็นต์ของระบบ VoIP ชนิดต่างๆ ซึ่งเทอร์มินอลและเกตเวย์ถือได้ว่าเป็นส่วนที่สำคัญที่สุดเรียกได้ว่าเป็นจุดปลายทาง (end points) อุปกรณ์สองชิ้นนี้มีหน้าที่สำคัญที่เหมือนกันคือ ทำการจดจำจุดเริ่มต้นและจุดสิ้นสุดของส่วนที่เป็นไอพีของการโทรด้วยเสียง เมื่อผู้ใช้เครื่องพีซีมัลติมีเดียสองคนโทรคุยกันด้วยแอพพลิเคชั่นซอฟต์แวร์ที่มีอยู่บนเครื่องของตนโดยใช้ไมโครโฟนและลำโพงที่มากับพีซี การโทรทั้งระบบสามารถกระทำบนไอพีระหว่างเทอร์มินอล H.323 สองตัวได้ ในกรณีที่มีการโทรทางไกลจากโทรศัพท์เครื่องหนึ่งไปยังอีกโทรศัพท์หนึ่งโดยมีตัวกลางที่ใช้ VoIP interexchange carrier เสียงจะผ่านจากเครือข่าย PSTN ไปยังเครือข่ายแพ็กเก็ตแล้วกลับมาที่เครือข่าย PSTN อีกครั้ง ซึ่งการแปลงจาก PSTN ไปยังเครือข่ายไอพีนั้นกระทำที่เทอร์มินอล H.323 โดยเทอร์มินอล H.323 จะสร้างจุดปลายทางของส่วนที่เป็น VoIP ของการโทร ทั้งสองกรณีนี้ฟังก์ชันหลักของอุปกรณ์ยังคงเหมือนเดิมคือ แต่ละตัวจะอธิบายถึงจุดที่เสียงได้เข้าหรือออกจากชุดของแพ็กเก็ตไอพี ซึ่ง H.323 ทำการเรียกชุดของแพ็กเก็ตไอพีเหล่านั้นว่าสตรีมข้อมูล (information stream)

ส่วนประกอบหลักของสตรีมข้อมูลใน VoIP ก็คือเสียง และจากมาตรฐานของ H.323 อนุญาตให้มันสามารถถูกสร้างได้จากหลายวิธี เพื่อที่จะเป็นไปตามข้อตกลงของมาตรฐาน H.323 นั้น เทอร์มินอล H.323 และเกตเวย์ทั้งหมดจะต้องมีความสามารถในการเข้ารหัสสัญญาณเสียงโดยใช้ G.711 codec (coder-decoder) ของ ITU-T ซึ่งเป็นวิธีที่เก่าแก่ในการแปลงสัญญาณเสียงที่เป็นอะนาล็อกให้เป็นข้อมูลดิจิตอล สรุปโดยย่อแล้ว G.711 จะใช้เทคนิคของ PCM (Pulse Code Modulation) ในการดิจิไทซ์ (digitize) สัญญาณเสียงและสร้างสตรีมข้อมูลที่มีขนาด 64 kb/s ซึ่งเป็นเทคนิคเดียวกันที่ใช้ในบริษัทโทรศัพท์ต่างๆ กับสายที่เป็นดิจิตอล

นอกเหนือจาก G.711 แล้ว H.323 ยังกำหนดเทคนิคที่ใช้ในการเข้ารหัสอื่นๆ อีก ดังนี้

- G.728 เป็นข้อกำหนดของ ITU-T สำหรับการเข้ารหัสสัญญาณที่ 16 kb/s โดยใช้อัลกอริทึม LD-ACELP (Low-Delay Algebraic-Code-Excited Linear Prediction)
- G.729 และ G.729a เป็นข้อกำหนดของ ITU-T สำหรับการเข้ารหัสสัญญาณที่ 8 kb/s โดยใช้อัลกอริทึมของการบีบอัดคำพูดของเสียงที่เรียกว่า CS-ACELP (Conjugate-Structure Algebraic-Code-Excited Linear Prediction) ส่วน G.729a เป็นตัวที่ลดความซับซ้อนของ G.729 มีผลทำให้ทำงานได้

เร็วขึ้น ซึ่ง G.729a ได้รับความสนใจจากผู้ผลิตเกตเวย์เพราะมีความล่าช้าของตัวอัลกอริทึมที่ลดลงโดยอยู่ที่เพียงเป็นจำนวนเท่าของ 15 มิลลิวินาทีเท่านั้น

- G.723.1 มีด้วยกันอยู่สองอัตราความเร็วคือ 5.3 และ 6.4 kb/s ซึ่งทั้งคู่จะต้องถูกสนับสนุนโดยตัวเข้ารหัสที่ใช้ทั้งหมดเพราะโพรโตคอลต้องการความสามารถในการสวิตช์ระหว่างอัตราความเร็วต่างๆ ทุกๆ 30 มิลลิวินาที นอกเหนือจากอัตราความเร็วสองตัวที่ว่ามานี้ซึ่งมีขนาดของแพ็กเก็ต 20 และ 24 ไบต์ตามลำดับแล้ว G.723.1 ยังได้กำหนดแพ็กเก็ตขนาด 4 ไบต์สำหรับการระงับความเงียบ (silence suppression) และการส่งข้อมูลที่เป็น "comfort noise" เมื่อทางด้านผู้ส่งเกิดเงียบขึ้นมา และเมื่อเปรียบเทียบกับตัวเข้ารหัส G.729 แล้ว G.723.1 สามารถดักสัญญาณ DTMF ได้ดีกว่า

3.4 Quality of Service (QOS)

Quality of Service หรือ QOS เป็นการริเริ่มในวงกว้างทางด้านอุตสาหกรรมเพื่อที่จะทำให้การใช้เน็ตเวิร์กมีประสิทธิภาพมากขึ้นโดยการแยกแยะความแตกต่างระหว่างกลุ่มของข้อมูล เป้าหมายของ QOS คือการดูแลชุดของข้อมูลชนิดใดชนิดหนึ่งทำให้ข้อมูลเหล่านั้นผ่านไปมาในเครือข่ายอินเทอร์เน็ตหรืออินทราเน็ตด้วยบริการในการสื่อสารที่มีคุณภาพมากขึ้น สำหรับ QOS ในระบบปฏิบัติการวินโดว์ส 2000 เป็นกลุ่มของคอมโพเน็นต์ที่สามารถเลือก จัดการ และดูแลข้อมูลที่มีคุณภาพสูงกว่าในการรับส่งข้ามเครือข่าย โดยคอมโพเน็นต์เหล่านี้พบได้ในฟังก์ชันของ DLLs โพรโตคอล บริการ และอุปกรณ์เน็ตเวิร์กต่างๆ

Internet Engineering Task Force (IETF) เป็นองค์กรที่มีบทบาทที่สำคัญในการที่ให้แน่ใจได้ว่ามาตรฐาน QOS นั้นสามารถใช้กับอุปกรณ์ทางด้านเครือข่ายในการติดต่อกันโดยมี QOS ด้วย ซึ่ง QOS จะทำให้แอพพลิเคชันสามารถคาดเดาและจัดการกับทรัพยากรบนเครือข่ายดังเช่น แบนด์วิดธ์และ latency ทั้งทางด้านเครื่องโลคอลและอุปกรณ์ที่ใช้ติดต่อกับเครือข่าย ทั้งนี้ฟังก์ชันของ QOS จะต้องได้รับความร่วมมือจากโหนดปลายทาง สวิตช์ เราเตอร์ และลิงค์ของ WAN ที่ข้อมูลไหลผ่าน หากไม่มีความร่วมมือจากอุปกรณ์ใดอุปกรณ์หนึ่งแล้ว คุณภาพในการส่งข้อมูลอาจจะไม่ดีนัก เพราะเหมือนกับว่าปล่อยให้อุปกรณ์เหล่านั้นทำการตัดสินใจเองในการส่งข้อมูล ทำให้การดูแลข้อมูลต่างๆ เหมือนกันหมดเป็นในลักษณะของ FIFO (first come-first served) ถึงแม้ว่าอาจจะเป็นการดีที่ทำให้อุปกรณ์เครือข่ายหรือมีเดียที่จะทำการส่งไม่ถูกรับภาระมากเกินไป แต่เมื่อมีการคับคั่งของข้อมูลมากๆ บริการเหล่านั้นสามารถทำให้ข้อมูลทั้งหมดเกิดความล่าช้าขึ้นได้ ดังนั้นจึงมีการนิยาม QOS เพิ่มเติมว่า กลุ่มของข้อมูลชนิดที่เจาะจงจะได้รับการปฏิบัติที่ดีกว่าจากอุปกรณ์ในเครือข่ายนั้นๆ

3.4.1 พารามิเตอร์ของ QOS

โดยคอนเซ็ปต์แล้ว QOS ถือเป็นทรัพยากรหนึ่งของเน็ตเวิร์ก หรือเป็นกลุ่มของทรัพยากรของเน็ตเวิร์ก ซึ่งสามารถแบ่งค่าต่างๆ ในรายละเอียดออกได้เป็น 6 พารามิเตอร์ ดังในตารางที่ 3-1

QoS Parameter	Example Values
Bandwidth (minimum)	64 kb/s, 1.5 Mb/s, 45 Mb/s
Delay (maximum)	50-ms round-trip delay, 150-ms roundtrip delay
Jitter (delay variation)	10% of maximum delay, 5 ms variation
Information loss (error effects)	1 in 1000 packets undelivered
Availability (reliability)	99.99% network availability
Security	Encryption and authentication required on all traffic streams

ตารางที่ 3-1 ค่าพารามิเตอร์ทั้ง 6 ตัวของ QOS

3.4.1.1 Bandwidth

แบนด์วิดธ์ถือได้ว่าเป็นพารามิเตอร์ที่เด่นกว่าตัวอื่นได้อย่างชัดเจน เป็นการวัดของจำนวนบิตต่อวินาทีที่สามารถใช้ได้บนเครือข่ายสำหรับแอพพลิเคชันใดๆ เมื่อใช้เป็นพารามิเตอร์ของ QOS แบนด์วิดธ์จะต้องเป็นค่าอย่างต่ำสุดที่แอพพลิเคชันต้องการในการใช้งาน ตัวอย่างเช่น เสียง PCM 64 kb/s ต้องการแบนด์วิดธ์ที่ 64 kb/s ถึงแม้ว่าแบ็กโบนของเน็ตเวิร์กจะมีความเร็วถึง 45 Mb/s ก็ไม่ต้องการแบนด์วิดธ์ที่สูงกว่านั้นอีกแล้ว แต่ถ้าหากการเข้าถึงเครือข่ายด้วยโมเด็ม V.34 ซึ่งสนับสนุนความเร็วเพียง 33.6 kb/s แบ็กโบนเน็ตเวิร์ก 45 Mb/s มาใช้กับเสียง 64 kb/s ก็ทำให้เสียงที่ได้มีคุณภาพไม่ดี แบนด์วิดธ์อย่างต่ำที่สุดของ QOS จะต้องมีให้ใช้ทุกๆ จุดของการเชื่อมต่อระหว่างผู้ใช้ด้วยกัน

3.4.1.2 Delay

ค่าของดีเลย์ (Delay) มีความสัมพันธ์กันกับค่าของแบนด์วิดธ์ หากมีแบนด์วิดธ์มากค่าของดีเลย์ก็จะน้อยลง เมื่อคำนึงถึงค่าของดีเลย์เป็นหลัก เช่นพวกเสียง PCM 64 kb/s ค่าพารามิเตอร์ของดีเลย์ใน QOS จะเป็นตัวกำหนดค่าดีเลย์ที่มากที่สุดที่จะรับได้ในการให้บิตของข้อมูลไหลผ่านเครือข่าย

3.4.1.3 Jitter

พารามิเตอร์ Jitter ใน QOS นี้เป็นการเซตขอบเขตของค่าดีเลย์ที่เปลี่ยนแปลง (delay variation) ในแอพพลิเคชันบนเครือข่าย คำว่า Jitter มีความหมายมากกว่าเป็นเพียงขอบเขตของค่าดีเลย์ที่เปลี่ยนแปลง แต่ยังถูกใช้ในความหมายทางเน็ตเวิร์กที่หมายถึงความแตกต่างของเวลาในระดับต่ำ (low-level) ของเทคนิค line coding อย่างไรก็ตามก็ยังใช้ในความหมายถึงขอบเขตของค่าดีเลย์ที่เปลี่ยนแปลงเป็นส่วนใหญ่ ค่าของ Jitter สามารถกำหนดเป็นค่าความสัมพันธ์ (relative) หรือค่าคงที่ (absolute) ในค่าพารามิเตอร์ของ QOS ได้ เช่น หากค่าดีเลย์ของเน็ตเวิร์กสำหรับแอพพลิเคชันเป็น 100 มิลลิวินาที Jitter อาจกำหนดเป็น ± 10% ของค่าดีเลย์นี้ ดังนั้นหากเน็ตเวิร์กดีเลย์อยู่ระหว่าง 90 และ 110 ก็ยังอยู่ในช่วงที่ Jitter ได้กำหนดไว้

หรืออีกทางหนึ่งคือ กำหนดเป็นค่าเป็น ±10 มิลลิวินาทีของค่าดีเลย์ ค่าของ Jitter มีผลมากต่อแอพพลิเคชันแบบเรียวไทม์เช่นพวกเสียงและวีดีโอ

3.4.1.4 Information Loss (Error Effects)

การสูญหายของข้อมูล หรือ Information Loss เป็นค่าพารามิเตอร์ QOS ที่ไม่ได้กล่าวถึงบ่อยเหมือนกับแบนด์วิดธ์หรือดีเลย์ ซึ่งอาจจะแตกต่างกันไปตามแอพพลิเคชัน การควบคุมความผิดพลาดที่เกิดขึ้นที่ทำให้เกิดการสูญหายของข้อมูลนั้น กระทำได้โดยมีการตรวจสอบความผิดพลาด (error detection) และกู้ข้อมูลที่ผิดพลาดนั้น (error recovery) ซึ่งเป็นวิธีที่ง่ายเหมือนกับการให้ผู้ส่งทำการส่งข้อมูลที่ผิดพลาดนั้นกลับมาอีกรอบ เน็ตเวิร์กโดยส่วนมากโดยเฉพาะเครือข่ายไอพีจะปล่อยให้แอพพลิเคชันเป็นตัวตรวจสอบและแก้ไขความผิดพลาดแทน

3.4.1.5 Availability (Reliability)

การเกิดการล่มของระบบเครือข่ายเป็นปัญหาหนึ่งที่ทำให้การส่งข้อมูลผ่านเน็ตเวิร์กได้ไม่ราบรื่น ระบบเครือข่ายบางระบบไม่สามารถให้บริการได้ใช้ตลอดเวลา เนื่องจากอาจจะมีการขัดข้องทางเทคนิค อุปกรณ์ต่างๆ เสียหายหรือเกิดอุบัติเหตุอื่นๆ โดยในหนึ่งปีมีทั้งหมด 60 x 60 x 24 x 365 หรือ 31,536,000 วินาที ซึ่งควรจะเกิดระบบล่มไม่ควรเกิน 0.01% ซึ่งหมายถึงระบบล่มเพียง 50 นาทีในหนึ่งปี (หรือเท่ากับ 99.99% ของสภาพที่พร้อมใช้งานได้) ปกติแล้วเครือข่ายในปัจจุบันมีสภาพที่พร้อมใช้งานหรือค่าของ Availability ได้ถึง 99.995%

3.4.1.6 Security

ระบบความปลอดภัยถือเป็นตัวใหม่ที่อยู่ในพารามิเตอร์ของ QOS แต่อาจถือได้ว่าเป็นตัวที่สำคัญรองจากแบนด์วิดธ์ สถาปัตยกรรมต่างๆ พยายามรองรับและเพิ่มเติมในเรื่องของความเป็นส่วนตัว (privacy) และการพิสูจน์ตน (Authentication) ในอินเทอร์เนต โพรโตคอลไอพีที่มีความปลอดภัยหรือที่เรียกว่า IPSec กลายเป็นกุญแจที่สำคัญในสถาปัตยกรรมที่จะมาสนับสนุนในเรื่องอีคอมเมิร์สบนอินเทอร์เนตและการป้องกันการฉ้อฉล (fraud) ในสภาพแวดล้อมของ VoIP พารามิเตอร์ของระบบความปลอดภัยใน QOS อาจจะเป็นการเข้ารหัสข้อมูลและความต้องการในการพิสูจน์ตนสำหรับข้อมูลทุกรูปแบบ หรือเป็นเพียงแค่เข้ารหัสข้อมูลสำหรับการถ่ายโอนข้อมูลและการติดต่อของอินเทอร์เน็ตเทเลโฟนนี่ก็ใช้แค่การพิสูจน์ตน

บทที่ 4

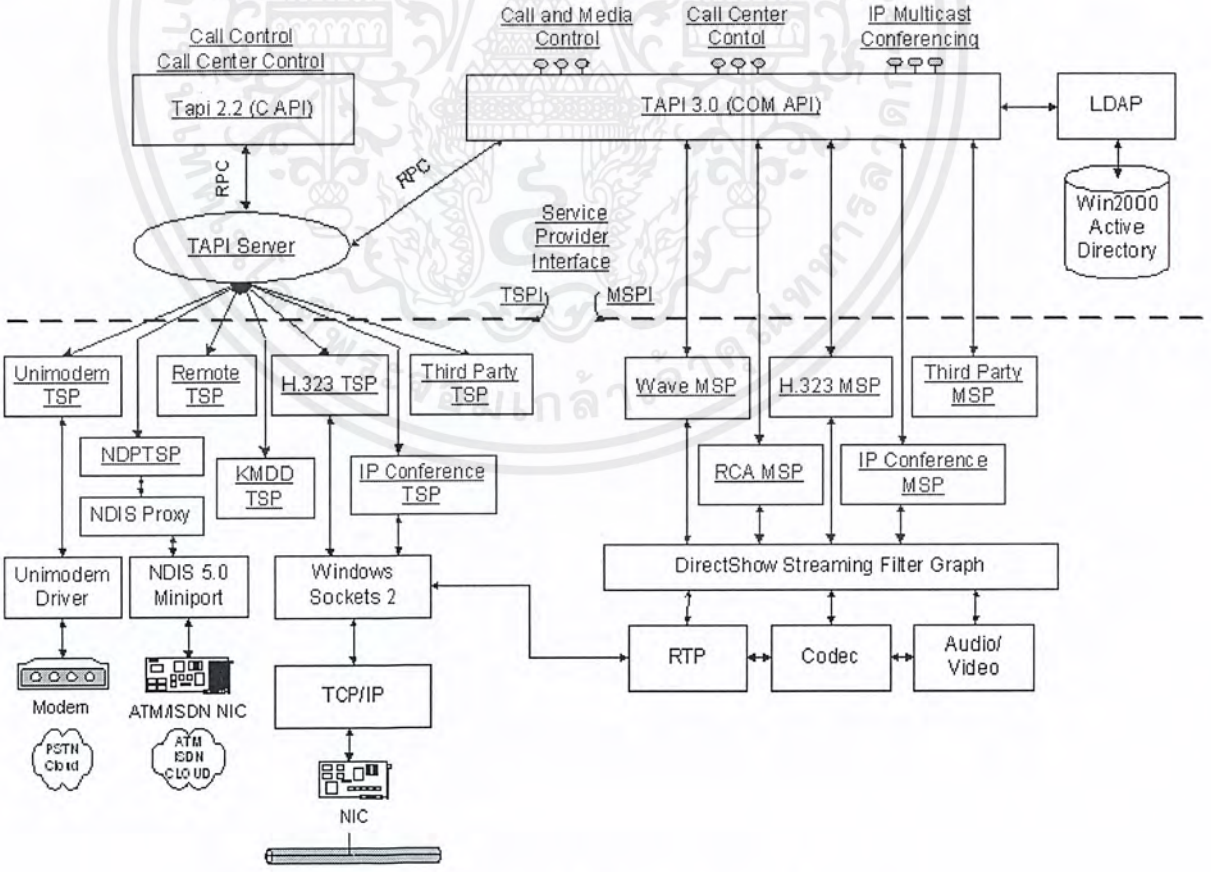
TAPI 3.0 (Telephony Application Programming Interface)

4.1 บทนำ TAPI

TAPI (อ่านว่า แทป-ปี้) เป็น API ที่เกี่ยวข้องกับการโทรศัพท์ซึ่งไม่จำเป็นจะต้องผ่านเครือข่าย PSTN เท่านั้น แต่สามารถที่จะใช้กับเครือข่ายอินเทอร์เนตหรือเครือข่ายไอพีก็ได้ การใช้ TAPI ทำให้กำจัดความยุ่งยากในการพัฒนาระดับรายละเอียดในแอพพลิเคชันต่างๆ ลงได้

4.2 ภาพรวมเทเลโฟนนี่ของไมโครซอฟท์

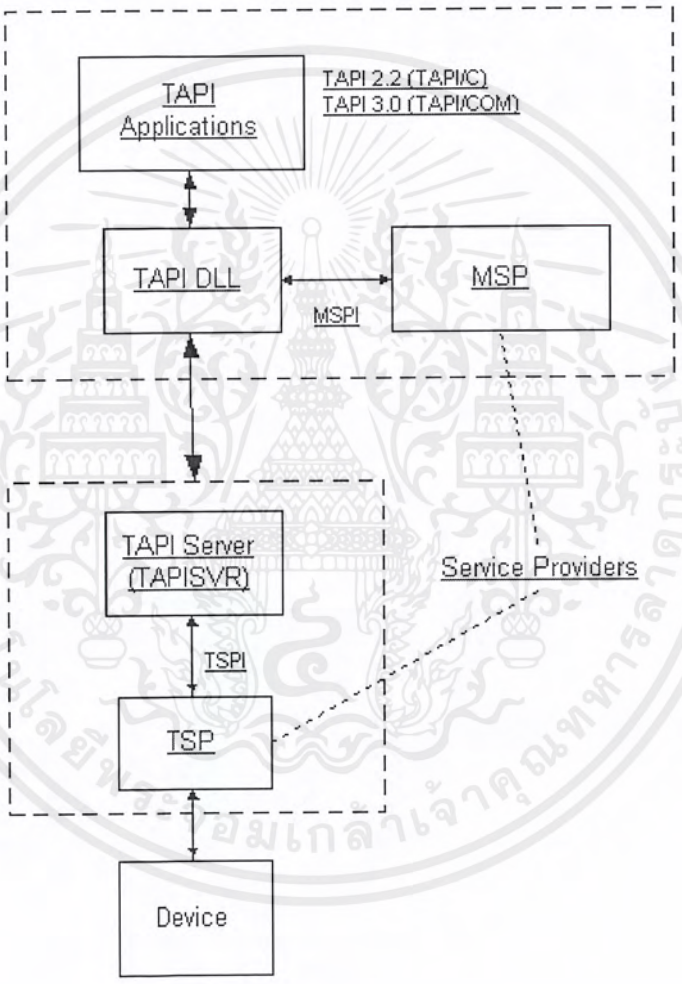
Telephony Application Programming Interface (TAPI), Telephony Service Provider Interface (TSPI) และ Media Service Provider (MSPI) ของไมโครซอฟท์สนับสนุนการพัฒนาแอพพลิเคชันของการติดต่อสื่อสารที่ทำงานบนระบบปฏิบัติการที่สนับสนุน Microsoft Win32 API เช่น Microsoft Windows 98 และ Microsoft Windows 2000 จากรูปที่ 4-1 แสดงถึงสถาปัตยกรรมของเทเลโฟนนี่ของไมโครซอฟท์ ซึ่ง TAPI สามารถนำมาใช้กับระบบ PSTN, ISDN และการติดต่อแบบ TCP/IP อีกทั้งยังสนับสนุนมาตรฐาน H.323 ด้วย



รูปที่ 4-1 สถาปัตยกรรมของไมโครซอฟท์เทเลโฟนนี่

4.3 โมเดลการเขียนโปรแกรมเทเลโฟนนีของไมโครซอฟท์

รูปแบบโมเดลการเขียนโปรแกรมโดยใช้เทคโนโลยีของไมโครซอฟท์ที่เรียกว่า Microsoft Telephony Programming Model ดังในรูปที่ 4-2 เป็นการจำลองการควบคุมการติดต่อจากส่วนที่ควบคุมอุปกรณ์และแอพพลิเคชันทางฝั่งของผู้ใช้ จากการใช้โมเดลนี้ทำให้แอพพลิเคชันไม่จำเป็นจะต้องรู้ถึงข้อมูลในระดับรายละเอียดในการควบคุมอุปกรณ์ และอุปกรณ์ไม่จำเป็นที่จะต้องถูกจัดการโดยแอพพลิเคชันโดยตรง ทำให้สามารถเปลี่ยนแปลงตัวแอพพลิเคชันหรืออุปกรณ์สื่อสารได้โดยไม่ต้องเปลี่ยนแปลงอีกส่วนหนึ่ง



รูปที่ 4-2 โครงสร้างของ Microsoft Telephony Programming Model

โดยจากในรูปที่ 4-2 สามารถแบ่งส่วนประกอบต่างๆ ได้เป็น TAPI Applications, TAPI DLL, TAPI Server, TSP และ MSP ออกมาเป็นรายละเอียดดังนี้

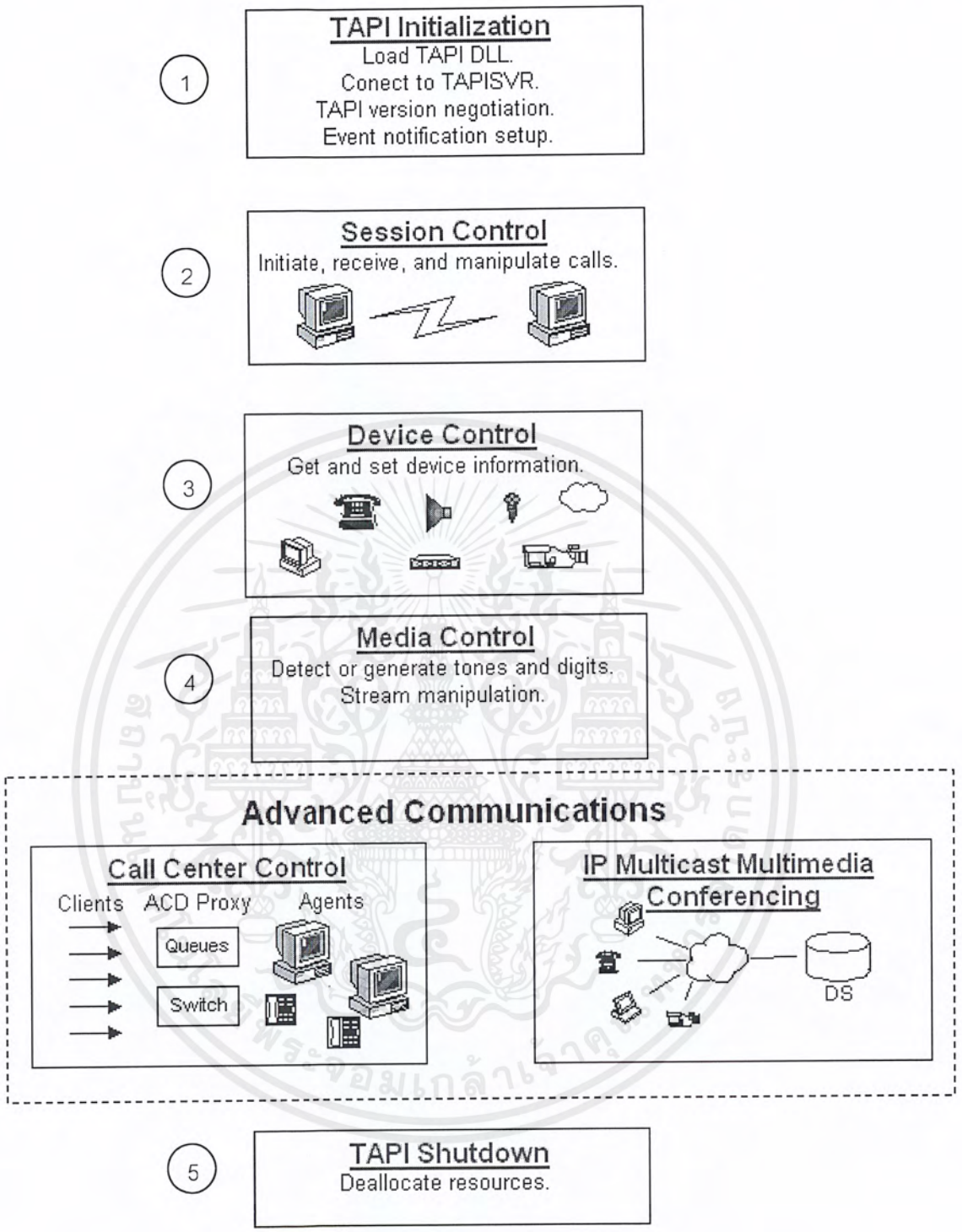
4.3.1 TAPI Applications

TAPI Applications เป็นแอพพลิเคชันที่จะคำนึงถึงเพียงความต้องการของผู้ใช้ ขณะเดียวกันส่วนของ TAPI DLL และ TAPI Server (TAPISVR) จะจัดการในการติดต่อระหว่างไคลเอนต์และเซิร์ฟเวอร์ และส่วนของตัวให้บริการ (Service Providers) จะทราบถึงรายละเอียดในการควบคุมอุปกรณ์ต่างๆ โดยการทำงานมีดังนี้

- แอพพลิเคชันจะโหลด TAPI DLL เข้าสู่หน่วยความจำของโพรเซสและใช้ TAPI เพื่อที่จะทำการติดต่อ
- TAPI จะสร้างการติดต่อแบบ RPC (Remote Procedure Call) กับ TAPI Server
- สำหรับ TAPI เวอร์ชัน 3 จะสร้างออบเจ็กต์ MSP และติดต่อกับออบเจ็กต์นั้นโดยการใช้กลุ่มของคำสั่งที่เรียกว่า Media Service Provider Interface (MSPI)
- เมื่อแอพพลิเคชันเรียกการทำงานของ TAPI แล้ว ส่วนของ TAPI DLL จะทำการตรวจสอบและจัดเรียงพารามิเตอร์ (Marshaling) เพื่อส่งต่อไปยัง TAPISVR
- TAPISVR จะหาแหล่งของอุปกรณ์ในการติดต่อสื่อสารที่มีอยู่ในเครื่องนั้น พร้อมทั้งติดต่อกับ Telephony Service Providers (TSPs) โดยใช้ Telephony Service Provider Interface (TSPI)
- การติดต่อระหว่าง TSP และ MSP เกิดขึ้นโดยใช้การเชื่อมต่อเสมือนผ่านทาง TAPI DLL และ TAPISVR
- ส่วนของ TSP และ MSP จะให้ข้อมูลของสถานะและความสามารถของอุปกรณ์เหล่านั้น และทำการอิมพลีเมนต์คำสั่งที่ต้องการเพื่อที่จะได้การตอบสนองที่ต้องการ

ผลลัพธ์จากการใช้โปรแกรมมิ่งโมเดลนี้คือแอพพลิเคชันอาจต้องการจัดการเมื่อมีการเปลี่ยนแปลงของอุปกรณ์หรือไม่ก็ได้ รวมทั้งการเพิ่มอุปกรณ์ใหม่สามารถใช้ได้โดยไม่ต้องรอเปลี่ยนโค้ดของโปรแกรม

ในส่วนของการอิมพลีเมนต์ สิ่งแรกที่ผู้เขียนโปรแกรมต้องคำนึงถึงในการใช้ TAPI คือปริมาณและรูปแบบของบริการที่ต้องการใช้ (level of service) เช่น ถ้าแอพพลิเคชันต้องการเพียงหมุนโทรศัพท์ การทำแอพพลิเคชันโดยใช้ความสามารถทั้งหมดของ TAPI ก็ไม่จำเป็น สิ่งต่อมาคือการตัดสินใจในการใช้ระหว่าง TAPI เวอร์ชัน 2 ซึ่งเป็น API ในรูปแบบของภาษาซีกับ TAPI เวอร์ชัน 3 ซึ่งเป็น COM (Component Object Model)



รูปที่ 4-3 ส่วนพื้นฐานต่างๆ ของ TAPI Applications

จากรูปที่ 4-3 แสดงส่วนประกอบต่างๆ ของ TAPI Applications ที่อยู่ในสถาปัตยกรรมเทเลโฟนนี่ของไมโครซอฟท์ ซึ่งมีดังนี้

4.3.1.1 TAPI Initialization

ฟังก์ชันของคอมโพเน็นต์ TAPI จำเป็นต้องมีการติดตั้งการติดต่อกับสิ่งแวดล้อมบนเครื่องคอมพิวเตอร์ดังนี้

- Installation จะทำเมื่อมีการเพิ่มซอฟต์แวร์และฮาร์ดแวร์ครั้งแรกให้กับเครื่องคอมพิวเตอร์ การจัดการในรายละเอียดขึ้นกับระบบปฏิบัติการและตัวซอฟต์แวร์
- Primary initialization เป็นการสร้างออบเจ็กต์และเส้นทางการติดต่อ
- Version negotiation เป็นส่วนที่ทำให้แน่ใจว่าคอมโพเน็นต์ TAPI สามารถแลกเปลี่ยนข้อมูลกันได้
- Resource inventory จัดเก็บข้อมูลที่เกี่ยวกับอุปกรณ์ที่มีทั้งหมดในการใช้งานของแอพพลิเคชัน TAPI
- Event notification กำหนดวิธีที่ TAPI และ Service Providers ส่งผลการทำงานแบบอะซิงโครนัสและข้อมูลของการเปลี่ยนสถานะให้กับแอพพลิเคชัน

4.3.1.2 Session Control

เซสชัน (Session) หรือการ Call เป็นการติดต่อตั้งแต่สองแอดเดรสขึ้นไป การเชื่อมต่อนี้เป็นไดนามิคและออบเจ็กต์ที่เกี่ยวข้องจะถูกสร้าง นำมาใช้ และทำการยกเลิกตามความเหมาะสม เช่นในกรณีทั่วไปซึ่งหมายถึงการใช้และวางหูโทรศัพท์ ในกรณีชั้นสูง Session Control อาจหมายถึงการจัดการการประชุมแบบมัลติมีเดียบนระบบเครือข่ายไอพี จนไปถึงในระดับผู้เข้าร่วมประชุมแต่ละคน

Session Control มีความเกี่ยวข้องกับ 2 ส่วนดังนี้

- Session Operations คือการควบคุมการสร้าง รักษา และยกเลิกเซสชันในการติดต่อ
- Session Information อธิบายถึงข้อมูลที่เกี่ยวกับเซสชันของการติดต่อ

4.3.1.3 Device Control

เป็นส่วนของการควบคุมอุปกรณ์บนแอพพลิเคชันทั้งฝั่งไคลเอนต์ผู้ใช้และเซิร์ฟเวอร์ ซึ่งต้องการเพียงส่วนของข้อมูลพื้นฐานเท่านั้น ในส่วนของ Service Providers จะจัดการกับการควบคุมอุปกรณ์ในรายละเอียดเอง ซึ่งจะส่งข้อมูลเกี่ยวกับอุปกรณ์ที่จำเป็นไปให้แก่แอพพลิเคชันโดยผ่านทาง TAPI

ประเภทของอุปกรณ์ที่สำคัญ ได้แก่

- Network: เป็นชั้นของ transport layer สำหรับการติดต่อ จากมุมมองของแอพพลิเคชัน ข้อมูลเกี่ยวกับเน็ตเวิร์คจะถูกฝังอยู่ในชนิดของแอดเดรส เช่น LINEADDRESSTYPE_PHONENUMBER
- Line: เป็นส่วนที่เชื่อมต่อเข้ากับเครือข่าย ซึ่งจะใช้มากกับ TAPI 2.2
- Channel: เป็นส่วนย่อยของ Line ซึ่งโดยปกติ Service Providers จะเป็นตัวจัดการให้
- Address: คือตำแหน่งที่อยู่ของเน็ตเวิร์ค
- Terminal: แหล่งกำเนิดหรือปลายทางของแอดเดรสและชนิดของข้อมูล

Service Providers มีหน้าที่ในการรายงานคุณลักษณะของอุปกรณ์ต่างๆ ให้กับ TAPI เพื่อใช้ในการตอบสนองกับการร้องขอของแอพพลิเคชัน และเป็นตัวเริ่ม (initialize) การรายงานความเปลี่ยนแปลง

ของสถานะของอุปกรณ์ด้วย การเปลี่ยนแปลงเหล่านี้จะถูกรายงานต่อไปยังแอพพลิเคชันตามความต้องการที่กำหนดในระหว่างการทำ initialization คุณลักษณะพื้นฐานของอุปกรณ์เช่น Device Class, Device Identifier, Address Type, Address Identifier, Device Events, Media Type และ Terminal Type

4.3.1.4 Media Control

มีเดียเป็นรูปแบบของข้อมูลที่ถูกส่ง การควบคุมมีเดีย (Media Control) ยอมให้แอพพลิเคชันรองรับชนิดของมีเดียได้หลายรูปแบบและทำการปรับลักษณะของการไหลของข้อมูลได้ เช่น ปริมาณของข้อมูลเสียงในการรับส่งแต่ละครั้ง การควบคุมมีเดียนั้นขึ้นอยู่กับชนิดของแอพพลิเคชัน TAPI, การสนับสนุนของ Service Provider และเอ็นไวรอนเมนต์ของการติดต่อแบบโลคอล

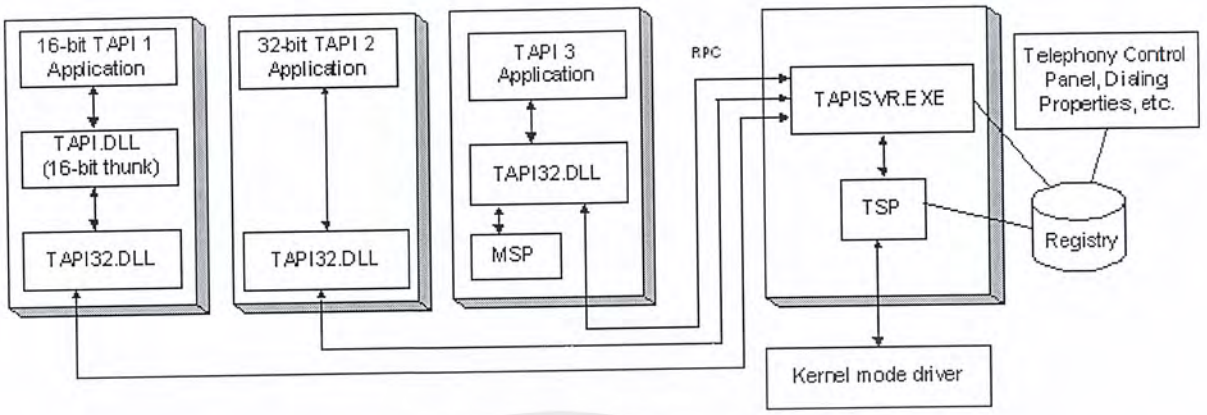
ในอดีตแอพพลิเคชันได้มีส่วนในการควบคุมมีเดียเพียงเล็กน้อยเท่านั้นเมื่อมีการติดต่อสื่อสารเกิดขึ้น ต่อมา เป็นแอพพลิเคชันที่ใช้ TAPI 2.0 สามารถใช้บางฟังก์ชันในการควบคุมมีเดีย แต่ยังไม่สามารถที่จะเข้าถึงข้อมูลของมีเดียได้ สำหรับ TAPI ในเวอร์ชัน 3.0 ได้มีส่วนของ Media Service Providers ซึ่งเพิ่มรายละเอียดและการควบคุมบนมีเดียและเซสชันได้มากขึ้น ทำให้แอพพลิเคชันที่ใช้ TAPI 3 สามารถเข้าถึงข้อมูลของมีเดียได้โดยตรง ข้อมูลได้ถูกสร้างขึ้นสำหรับแต่ละชนิดของมีเดียที่เกี่ยวข้องกับเซสชันเช่น เสียงและวีดีโอ

4.3.1.5 TAPI Shutdown

การปิดตัวของเซสชันและแอพพลิเคชันที่ถูกต้อง จำเป็นในการป้องกันไม่ให้มีการเชื่อมต่อหรือแอพพลิเคชันหลงเหลืออยู่ในการจองทรัพยากรของระบบโดยสูญเปล่า TAPI มีฟังก์ชันและเมธอดที่ช่วยในการจัดการ ซึ่งแอพพลิเคชันต้องรับผิดชอบในการปล่อยหน่วยความจำที่ใช้งานอยู่ ไม่ว่าแอพพลิเคชันจะอยู่ในรูปแบบของโครงสร้างข้อมูล (data structure) หรือ COM ก็ตาม

4.3.2 TAPI DLL

TAPI DLL ใช้ร่วมกับ TAPI Server (Tapisrv.exe) เป็นส่วนสำคัญที่แบ่งแอพพลิเคชันของผู้ใช้ปลายทางและเซิร์ฟเวอร์ออกจากส่วนของ Service Providers ซึ่งแอพพลิเคชัน TAPI จะโหลด DLL ที่จำเป็นไว้ในหน่วยความจำของโพรเซสระหว่างการทำ initialization TAPI จะสร้างการติดต่อแบบ RPC กับ Tapisrv.exe ซึ่งมีไฟล์ DLLs ทั้งหมด 3 ไฟล์ที่เกี่ยวข้องกับ TAPI ได้แก่ Tapi.dll, Tapi32.dll และ Tapi3.dll



รูปที่ 4-4 บทบาทของ TAPI DLL ในเทเลโฟนนี่ของไมโครซอฟท์

จากรูปที่ 4-4 แอพพลิเคชัน 16 บิตจะติดต่อกับ Tapi.dll ใน Windows 3.1 และ Windows 95 ซึ่ง DLL ตัวนี้เป็นแก่นของเทเลโฟนนี่ในวินโดว์สภายใต้ระบบปฏิบัติการ Windows 98 และ Windows NT/Windows 2000 Tapi.dll จะทำหน้าที่เป็น thunk layer เพื่อที่จะแปลงแอดเดรส 16 บิตเป็นแอดเดรส 32 บิต และส่งการร้องขอต่อไปยังตัว Tapi32.dll สำหรับแอพพลิเคชัน 32 บิตจะติดต่อกับ Tapi32.dll เลย สำหรับใน Windows 95 Tapi32.dll นี้จะทำหน้าที่เป็น thunk layer ของ Tapi.dll ใน Windows 98 และ Windows NT/2000 โดย Tapi32.dll เป็นชั้นที่บาง (thin layer) ซึ่งจะส่งการร้องขอไปยัง TAPI Server (Tapisrv.exe) และจะโหลด Media Service Provider DLLs ในกรณีที่จำเป็นลงในโพรเซส และสำหรับแอพพลิเคชันที่ใช้ TAPI 3 จะติดต่อกับ Tapi3.dll เท่านั้น

4.3.3 TAPI Server

TAPI Server (Tapisrv.exe) เป็นศูนย์กลางที่เก็บของข้อมูลเกี่ยวกับเทเลโฟนนี่ของเครื่องผู้ใช้ โพรเซสของบริการนี้จะจัดการกับทรัพยากรของเทเลโฟนนี่ทั้งแบบโลคอลและรีโมต และจัดการให้แอพพลิเคชันลงทะเบียนเพื่อที่จะรองรับการร้องขอและระหว่างการรอคอยฟังก์ชันแบบอะซิงโครนัส รวมทั้งสร้างอินเทอร์เฟซกับ TSP ตามรูปที่ 4-2 ของ Microsoft Telephony Programming Model

บนระบบปฏิบัติการ Windows 95, Windows 98 และ Windows NT ตัว Tapisrv.exe จะทำงานเป็นแต่ละโพรเซสที่แยกออกไป แต่บนระบบปฏิบัติการ Windows 2000 จะทำงานภายในส่วนของ Svchost.exe เมื่อแอพพลิเคชันโหลด TAPI DLL เข้าสู่หน่วยความจำของโพรเซสและทำการ initialization แล้ว DLL จะติดต่อกับ TAPI Server โดยใช้ RPC จากนั้น TAPI Server จะโหลด TSP เข้าไปในหน่วยความจำของโพรเซส โดยไม่ว่าจะมีจำนวนแอพพลิเคชันที่ทำการเข้าถึงอุปกรณ์เท่าใดก็ตาม ก็จะมีเพียง TSP เดียวเท่านั้น

4.3.4 Service Providers

Service Providers จะอิมพลีเมนต์การควบคุมอุปกรณ์ในส่วนของรายละเอียดของเทเลโฟนนี่ โดยแบ่งเป็น Telephony Service Provider (TSP) และ Media Service Provider (MSP) ซึ่ง TSP จะทำหน้าที่ในการบริการของการควบคุมในการ Call และ MSP จะเป็นตัวจัดการกับข้อมูลของมีเดีย โดยทุก TSP จะถูกประมวลผลในโพรเซสของ Tapisrv.exe ซึ่งสามารถสร้างเธรดใน TAPISRV เพื่อใช้ในการทำงานตามที่ต้องการ และมั่นใจได้ว่าไม่มีทรัพยากรที่ถูกสร้างขึ้นจะสามารถถูกทำลายโดยการออกของแอพพลิเคชันใดๆ TAPI Server จะแปลคำสั่งของแอพพลิเคชันเป็นคำสั่งที่รู้จักในนามของ Telephony Service Provider Interface (TSPI) สำหรับ MSP จะทำงานในพื้นที่โพรเซสของแอพพลิเคชัน ซึ่งทำให้มีการตอบรับที่รวดเร็วซึ่งบางครั้งจำเป็นในการควบคุมตัวมีเดียด้วย ตัว TAPI DLL จะทำงานร่วมกับ Media Service Provider Interface (MSPI)

ตัวให้บริการของ TSP DLL สามารถใช้ฟังก์ชันใดๆ ของระบบและคอมโพเน็นต์ต่างๆ ได้ รวมทั้งสามารถเข้าถึงบริการที่เกี่ยวกับเน็ตเวิร์กได้ด้วยเช่นการใช้ RPC และซ็อกเก็ตของวินโดว์สสำหรับเทเลโฟนนี่แบบไคลเอนต์/เซิร์ฟเวอร์

TSP DLL ในส่วนของการอินเทอร์เฟซกับผู้ใช้ (user interface: UI) จะถูกโหลดโดย TAPI เข้าไปในโพรเซสของแอพพลิเคชัน ซึ่งจะเรียกฟังก์ชันของ Service Provider ใดๆ ที่สามารถแสดงไดอะล็อกบ็อกซ์ (dialog box) ได้ และตัว Service Provider จะสามารถทำให้ UI DLL เกิดการถูกโหลดและทำงานในโพรเซสของแอพพลิเคชันถ้าหาก Service Provider ต้องการที่จะแสดงส่วนของการอินเทอร์เฟซกับผู้ใช้ในเวลาที่ไม่คาดคิดมาก่อน เช่น การแสดงไดอะล็อกบ็อกซ์ของการคุยและวางสายโดย Universal Modem Driver (UNIMODEM) ขณะที่โมเด็มแบบข้อมูลถูกใช้ในการโทรศัพท์แบบ interactive voice call โดยใช้ฟังก์ชัน TSPI_lineMakeCall UNIMODEM driver ของ Service Provider สำหรับการควบคุมโมเด็มจะมีให้ใช้บนระบบปฏิบัติการ Windows NT และ Windows 2000

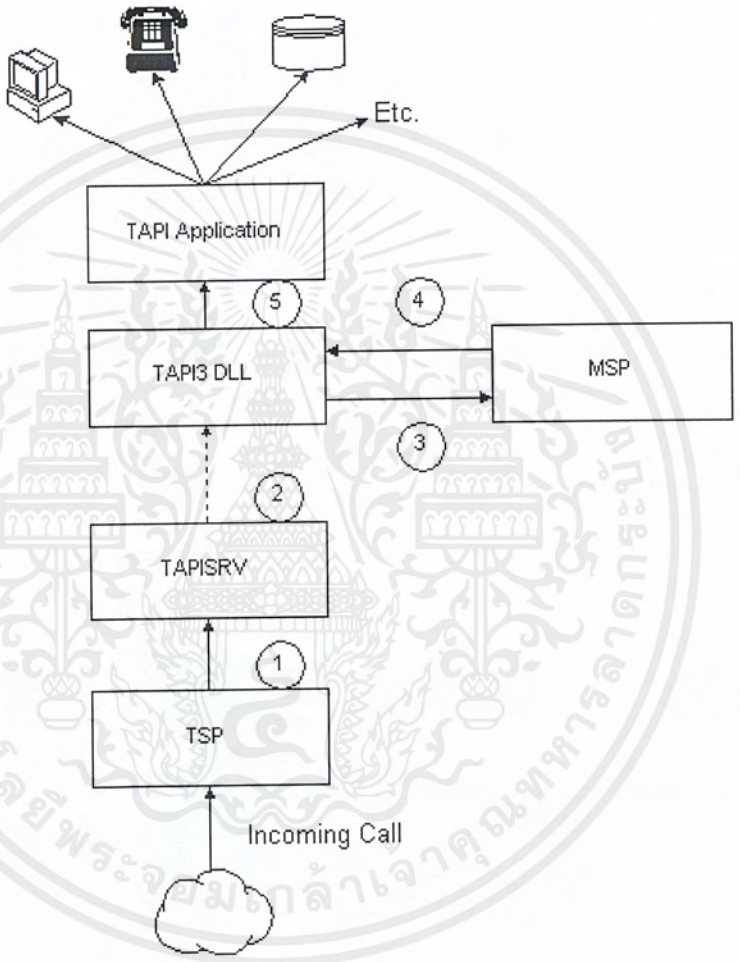
ในการจัดการส่งข้อมูลอย่างต่อเนื่อง TAPI 3 จะใช้ DirectShow™ API เป็นตัวหลักในการจัดการกับข้อมูลนั้น ซึ่ง MSP จะเป็นตัวอิมพลีเมนต์อินเทอร์เฟซของ DirectShow สำหรับ TSP ที่ต้องการ หัวใจสำคัญของ DirectShow อยู่ที่ระบบที่ประกอบไปด้วยคอมโพเน็นต์ต่างๆ ที่ถูกกำหนดขึ้นเรียกว่า filter graph

QoS ใน TAPI 3 จะถูกจัดการผ่านทาง RTP (Real-time Transport Protocol) filter graph ของ DirectShow ซึ่งมีหน้าที่ร้องขอแบนด์วิดธ์ที่ codecs ของ DirectShow ที่ต้องการตามชนิดของข้อมูลในเครือข่าย จากนั้น RTP จะใช้อินเทอร์เฟซ ของ Winsock 2.0 ส่งความต้องการนั้นผ่านไปยัง Winsock2 QoS Service Provider (QoS SP) เพื่อที่จะหาวิธีที่เหมาะสมให้เป็นไปตาม QoS ซึ่งมีดังนี้

- Resource Reservation Protocol (RSVP): เป็นการจองทรัพยากรอย่างเช่น แบนด์วิดธ์ ของเครือข่ายผ่านทางเราเตอร์ตามมาตรฐานของ IETF
- Local Traffic Control: เป็นการควบคุมการไหลของข้อมูล
- IP Type of Service: โดยแต่ละแพ็กเก็ตจะมีฟิลด์ขนาด 3 บิตที่แสดงถึงระดับความสำคัญของแพ็กเก็ตนั้นๆ ซึ่งฟิลด์ที่เพิ่มเหล่านี้จะเป็นตัวบอกถึงความล่าช้า ทรูพุต หรือความเชื่อถือได้ของเครือข่าย

4.4 การจำลองเหตุการณ์เมื่อมีการ Call เข้ามา

การจำลองเหตุการณ์เมื่อมีการ Call เข้ามาจะมีขั้นตอนการตั้งต้น (initialization) ก่อนการรับส่งข้อมูล ตามรูปที่ 4-5 ดังต่อไปนี้



รูปที่ 4-5 เหตุการณ์การทำงานของ TSP และ MSP เมื่อมีการ Call เข้ามา

1. TSP ส่งแมสเซส LINE_NEWCALL ไปยัง TAPISRV โดยสถานะการ Call จะเป็น LINECALLSTATE_OFFERING
2. TAPISRV ส่งสัญญาณบอกให้ไคลเอนต์ทราบถึงการ Call ที่เข้ามา

3. TAPI3 DLL สร้างออบเจ็กต์ของ TAPI Call แล้วเรียก ITMSPAddress::CreateMSPCall ซึ่งถูกอิมพลีเมนต์โดย MSP
4. MSP สร้างออบเจ็กต์ของ MSP Call และปรับลักษณะของข้อมูลให้เป็นไปตามชนิดของมีเดียที่ต้องการสำหรับการ Call นั้น แล้วจึงส่งค่าพอยน์เตอร์ของ IUnknown ให้แก่ออบเจ็กต์ของ MSP Call นั้น
5. TAPI3 ก็รวมออบเจ็กต์ MSP Call เข้าไปยังออบเจ็กต์ของ TAPI Call เพื่อทำการสร้างอินเทอร์เฟซ เช่น ITStreamControl ให้แก่แอพพลิเคชัน แล้วจึงแจ้งให้แอพพลิเคชันทราบว่ามีการ Call เข้ามาใหม่

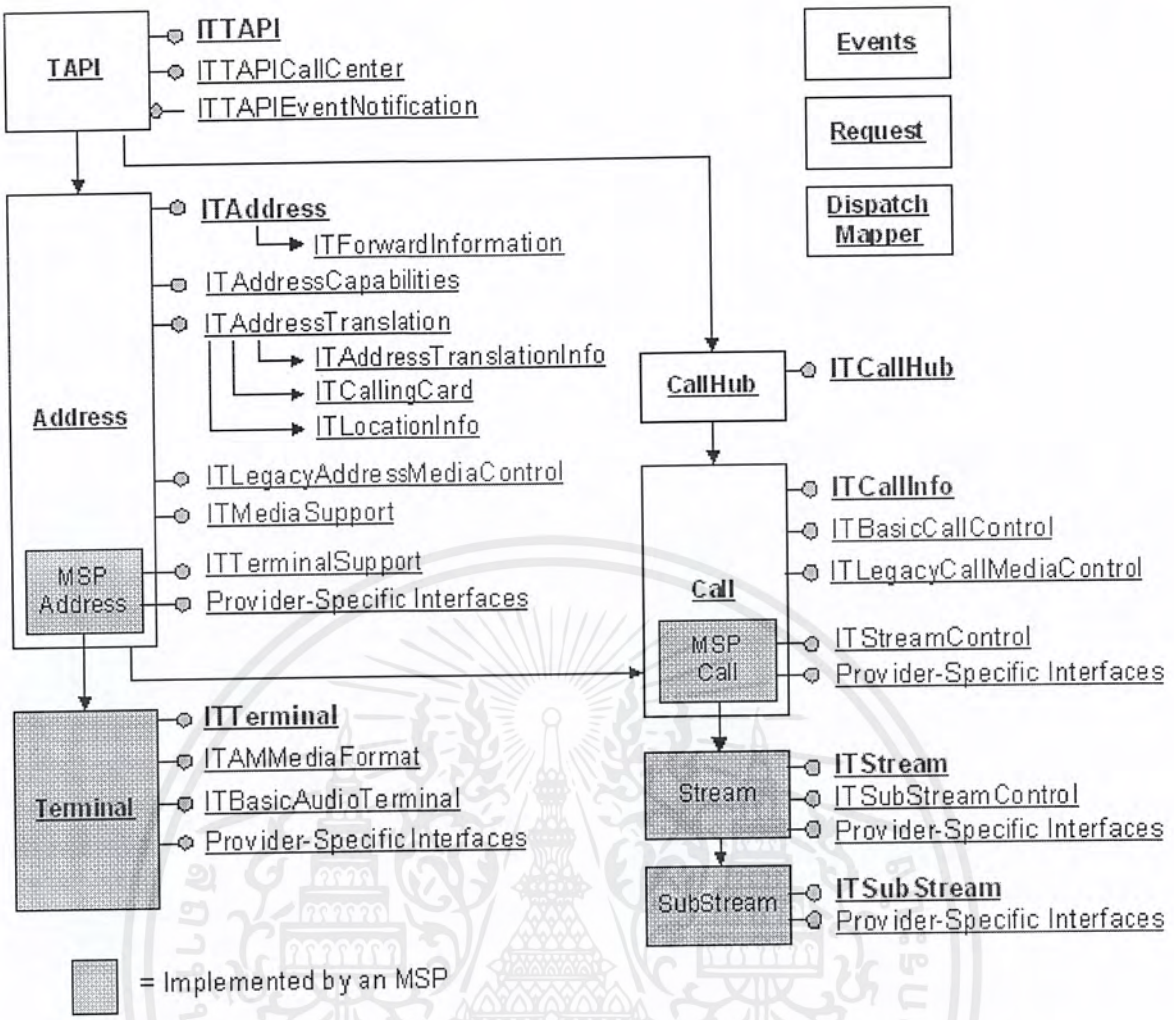
ITStreamControl ที่มีสำหรับแอพพลิเคชัน จะเป็นตัวบอกให้แอพพลิเคชันทราบถึงการเข้ามาของการ Call จากนั้นแอพพลิเคชันสามารถเลือกใช้เมธอดที่เหมาะสมในการเตรียมพร้อมสำหรับการทำงานในขั้นต่อไป

หลังจากนั้นแอพพลิเคชันจึงเรียก ITBasicCallControl::Answer ส่วน TAPI3 เรียก lineAnswer และสำหรับ TAPISERV เรียก TSPI_lineAnswer ในขั้นสุดท้าย TSP จึงเริ่มตั้งต้นการรับส่งข้อมูลของการ Call นั้น โดยทั่วไป TSP จะส่งสัญญาณไปยัง MSP ที่เกี่ยวข้อง แล้ว MSP จึงค่อยเริ่มรับส่งข้อมูล แต่สำหรับการอิมพลีเมนต์ของ TSP และ MSP ในบางครั้ง TSP จะเป็นตัวเริ่มรับส่งข้อมูลเอง

4.5 การควบคุมมีเดียและการ Call ใน TAPI 3.0

มีเดียและการ Call ของ TAPI 3.0 เป็นกลุ่มของ COM ออบเจ็กต์ อินเทอร์เฟซ และเมธอดสำหรับทำการ Call ระหว่างเครื่องตั้งแต่ 2 เครื่องขึ้นไป การ Call ของ TAPI 3.0 ไม่ได้หมายถึงเพียงแต่การส่งเสียงผ่านเครือข่ายโทรศัพท์เท่านั้น แต่หมายถึงตัวกลางและวิธีในการส่งทุกๆ ชนิดที่ผู้ให้บริการสามารถสนับสนุนได้

ออบเจ็กต์หลัก 5 ตัวในสถาปัตยกรรมของการควบคุมมีเดียและการ Call สำหรับ TAPI 3.0 คือ TAPI, Address, Terminal, Call และ CallHub ในรูปจะแสดงออบเจ็กต์เหล่านี้และอินเทอร์เฟซต่างๆ ที่สัมพันธ์กัน



รูปที่ 4-6 ออบเจ็กต์และอินเทอร์เฟซหลักที่ใช้ในการควบคุมมีเดียและการ Call

4.5.1 ออบเจ็กต์ TAPI

ออบเจ็กต์ TAPI แสดงถึงทรัพยากรของเทเลโฟนนี่ทั้งหมดที่คอมพิวเตอร์สามารถเข้าถึงได้ ทำให้แอพพลิเคชันสามารถระบุแอดเดรสโลคอลและรีโมตได้ทั้งหมด แอพพลิเคชัน TAPI 3.0 ต้องสร้างอินแสตนท์ของออบเจ็กต์ TAPI และทำการ initialize มันด้วย

4.5.2 ออบเจ็กต์ Address

ออบเจ็กต์ Address หมายถึงสิ่งที่แสดงว่าสามารถที่จะทำและรับการ Call ได้ ออบเจ็กต์นี้มีอินเทอร์เฟซและเมธอดที่อนุญาตให้แอพพลิเคชันกระทำสิ่งต่างๆ ได้ เช่น

- สามารถแสดงว่าแอดเดรสที่กำหนดให้สามารถสนับสนุนชนิดของมีเดียที่ต้องการได้หรือไม่
- ระบุการ Call ปัจจุบันที่สัมพันธ์กับแอดเดรส
- สร้างหรือทำการส่งต่อการ Call ได้
- แสดงชื่อของผู้ให้บริการที่เกี่ยวข้องได้

- ถ้ามี MSP อยู่ จะรับพอยน์เตอร์ของอินเทอร์เฟซที่อนุญาตกระทำกับเทอร์มินอลในระดับรายละเอียดได้
- รับและเซตรายละเอียดอื่นๆ เช่น มีข้อความรออยู่หรือไม่

แอดเดรสส่วนใหญ่จะสัมพันธ์กับเทอร์มินอลด้วยแต่ก็ไม่จำเป็นเสมอไป เช่น ตัวรีโมต TSP ที่เข้าถึงอุปกรณ์บนเซิร์ฟเวอร์จะมีแอดเดรสบนเครื่องโลคอลแต่ไม่ใช่เป็นเทอร์มินอล เป็นต้น ถ้าหากแอดเดรสไม่ได้สัมพันธ์กับเทอร์มินอลแล้ว อินเทอร์เฟซ ITTerminalSupport ก็จะไม่เปิดให้กับออบเจ็กต์

4.5.3 ออบเจ็กต์ Terminal

ออบเจ็กต์ Terminal แสดงถึงแหล่งกำเนิด (source) หรือตัวแปลสัญญาณ (renderer) เช่น ไมโครโฟนหรือลำโพง เป็นต้น แอพพลิเคชันจะเลือกเทอร์มินอลที่มีขึ้นอยู่กับทิศทางของมีเดียและชนิดของเซสชั่นในการติดต่อสื่อสาร ข้อมูลแต่ละมีเดียที่เกี่ยวข้องก็จะถูกเลือกให้เข้ากับเทอร์มินอลที่เหมาะสมเพื่อที่จะเริ่มต้นการทำสตรีมมิ่ง

โดยปกติเทอร์มินอลจะถูกอิมพลีเมนต์โดย MSP และออบเจ็กต์ Terminal จะไม่สามารถใช้ได้ถ้าหากไม่มี MSP ที่เกี่ยวข้องในการติดต่อสื่อสารนั้น เทอร์มินอลแต่ละตัวจะมาจากคลาสเทอร์มินอล ซึ่งคลาสเทอร์มินอลแสดงถึงความสามารถของกลุ่มของแหล่งกำเนิดหรือตัวแปลสัญญาณ ตัวอย่างเช่น เทอร์มินอลที่แมบเข้ากับลำโพงจะแสดงเป็น CLSID_SpeakersTerminal และตัวให้บริการจะทำการอิมพลีเมนต์การควบคุมความดังของเสียง (volume control) เป็นต้น TAPI 3.0 ได้กำหนดกลุ่มของคลาสเทอร์มินอล ตัว MSP สามารถที่จะกำหนดคลาสเพิ่มเติมได้ และแอพพลิเคชันสามารถลงทะเบียนคลาสเทอร์มินอลใหม่ได้ จากจุดมองของแอพพลิเคชัน ตัวของเทอร์มินอลจะถูกอธิบายโดยชนิดของเทอร์มินอลและทิศทางของมัน ซึ่งชนิดของเทอร์มินอลอาจจะเป็นแบบสถิตหรือไดนามิก เทอร์มินอลแบบสถิตจะแมบไปยังฮาร์ดแวร์ เช่น โทรศัพท์หรือไมโครโฟน และเทอร์มินอลแบบไดนามิกจะแมบไปยังออบเจ็กต์ชั่วคราว เช่น ไฟล์หรือหน้าต่างวีดีโอ ส่วนทิศทางจะอธิบายว่าเทอร์มินอลนั้นเป็นแหล่งกำเนิดหรือตัวแปลสัญญาณ

4.5.4 ออบเจ็กต์ Call

ออบเจ็กต์ Call แสดงถึงการติดต่อของแอดเดรสระหว่างแอดเดรสโลคอลกับแอดเดรสอื่นๆ การควบคุมการ Call ทั้งหมดจะถูกทำผ่านออบเจ็กต์ Call ซึ่ง ITBasicCallControl และ ITCallInfo เป็นอินเทอร์เฟซที่ใช้บ่อยที่สุดของออบเจ็กต์ Call อินเทอร์เฟซเหล่านี้อิมพลีเมนต์การทำงานและการร้องขอต่างๆ เช่น ขอพอยน์เตอร์ของอินเทอร์เฟซสำหรับข้อมูลของมีเดีย

4.5.5 ออบเจ็กต์ CallHub

ออบเจ็กต์ CallHub แสดงถึงลักษณะการมองของเธิร์ดพาร์ตี้ของการ Call จากผู้ผลิตหลายราย อินเทอร์เฟซและเมธอดที่เกี่ยวข้องด้วยจะรับและเซตข้อมูลที่เกี่ยวข้องกับศูนย์กลาง (hub) เช่นมันกำลังทำงานอยู่หรือไม่ การใช้ออบเจ็กต์ CallHub ผู้ใช้ที่ต้องการความปลอดภัยของข้อมูลสามารถพบและควบ

คุมผู้สนทนาคนอื่นๆ ในการ Call ได้ ออบเจ็กต์ CallHub ไม่สามารถถูกสร้างขึ้นโดยตรงได้โดยแอพพลิเคชัน แต่จะถูกสร้างทางอ้อมเมื่อมีการ Call เข้ามาแล้วถูกรับผ่านทาง TAPI 3.0 ซึ่ง TAPI 3.0 อาศัยผู้ให้บริการ TAPI เพื่อจัดหาข้อมูลที่จำเป็นเกี่ยวกับ call ต่างๆ เพื่อที่จะนำมาอิมพลีเมนต์โดยใช้ออบเจ็กต์ CallHub เนื่องจากไม่ใช่ผู้ให้บริการทั้งหมดที่จะจัดหาข้อมูลเหล่านี้ให้และไม่ใช่ฮาร์ดแวร์ทั้งหมดที่จะสามารถติดตาม call hub ได้ ข้อมูลที่มีเกี่ยวกับผู้ใช้คนอื่นๆ ในการติดต่ออาจจะถูกจำกัดหรือไม่มีอยู่เลย

4.5.6 ออบเจ็กต์ Stream

ออบเจ็กต์ Stream เป็นแอ็บสแทรกต์ของข้อมูลมีเดียหรือข้อมูลอื่นๆ ที่เกี่ยวข้องกับเซสชันการ Call อินเทอร์เฟซและเมธอดต่างๆ ที่ถูกเปิดใช้ให้กับออบเจ็กต์สตรีมและสตรีมย่อย (substream) อนุญาตให้แอพพลิเคชันสามารถเข้าถึงการควบคุมในรายละเอียดได้ เช่น การหยุดสตรีมชั่วคราว เพิ่มชนิดของมีเดียให้กับเซสชันในการติดต่อสื่อสาร หรือปรับระดับเสียงของผู้ร่วมสนทนา

สตรีมมี 2 ชนิดหลักคือตัวสตรีมเองและสตรีมย่อย อินเทอร์เฟซและเมธอดของการอิมพลีเมนต์พื้นฐานจะเหมือนกัน แต่สตรีมย่อยอนุญาตให้ควบคุมในระดับที่ต่ำกว่า MSP ทั้งหมดต้องอิมพลีเมนต์อินเทอร์เฟซในการควบคุมสตรีมพื้นฐาน แต่การสนับสนุนสำหรับสตรีมย่อยเป็นแค่เพียงส่วนเพิ่มเติม

MSP และ TAPI เป็นตัวสร้างออบเจ็กต์สตรีมสำหรับการ Call ในระหว่างการทำการเริ่มต้นของเซสชันที่เข้ามาหรือเซสชันที่ออกไป แอพพลิเคชันมีหน้าที่รับผิดชอบในการระบุเทอร์มินอลที่เหมาะสมสำหรับสตรีมเหล่านี้และเลือกเทอร์มินอลให้กับสตรีมด้วย

4.5.7 อีเวนต์

อีเวนต์เป็นส่วนที่สำคัญในการควบคุมการ Call ภายใต้ TAPI 3.0 ซึ่งประกอบด้วย 4 ขั้นตอน โดยในการลงทะเบียนและรองรับการทำงานของอีเวนต์จะต้องทำดังนี้

1. ทำการอิมพลีเมนต์เมธอด ITTAPIEventNotification::Event (TAPI จะ call เมธอดนี้เมื่อมีอีเวนต์เกิดขึ้น) โดยปกติแล้วการอิมพลีเมนต์นี้ไม่ได้ทำไปมากกว่าการทำ AddRef กับพอยน์เตอร์อินเทอร์เฟซของ IDispatch แล้วจึงส่งให้กับตัวสร้างเมสเซจของแอพพลิเคชัน
2. ลงทะเบียนอินเทอร์เฟซ ITTAPIEventNotification ที่ออกไปโดยใช้มาตรฐาน COM ของอินเทอร์เฟซ IConnectionPointContainer และ IConnectionPoint แล้วทำการส่งพอยน์เตอร์ของเมธอด IConnectionPointContainer::Advise ไปยัง ITTAPIEventNotification::Event
3. ทำการเรียกเมธอด ITTAPI::put_EventFilter เพื่อที่จะบอก TAPI ว่าอีเวนต์ไหนบ้างที่แอพพลิเคชันจะรองรับ ตัวกรองอีเวนต์ (event filter) จะประกอบไปด้วยสมาชิก ORed ของตัวระบุ TAPI_EVENT ซึ่งจะต้องทำการเรียกเมธอด ITTAPI::put_EventFilter เพื่อที่จะเซตตัวกรองอีเวนต์และรองรับการทำงานของอีเวนต์ ถ้าหากไม่ทำการเรียก ITTAPI::put_EventFilter แล้ว ตัวแอพพลิเคชันจะไม่สามารถรับอีเวนต์ใดๆ ได้เลย

และจะต้องทำการเรียกเมธอด ITTAPI::RegisterCallNotifications ด้วย สำหรับแต่ละแอดเดรสของออบเจ็กต์ซึ่งแอพพลิเคชันจะเป็นตัวจัดการการ Call

4.5.8 ออบเจ็กต์ Request

ออบเจ็กต์ Request ถูกสร้างโดยใช้ CoCreateInstance ของ COM และมีอินเทอร์เฟซตัวเดียวคือ ITRequest อินเทอร์เฟซนี้จะมีเมธอด MakeCall ซึ่งอนุญาตให้แอพพลิเคชัน TAPI 3.0 สามารถใช้ตัวช่วยเทเลโฟนนี่ (Assisted Telephony) ได้ ตัวช่วยเทเลโฟนนี่จะจัดหาแอพพลิเคชันที่มีความสามารถของเทเลโฟนนี่โดยใช้กลไกที่ไม่สลับซับซ้อนในการโทรศัพท์ ทำให้ผู้พัฒนาไม่จำเป็นต้องรู้เทเลโฟนนี่มากนัก ตัวอย่างเช่น แอพพลิเคชันจำพวกสเปรดชีตสามารถเพิ่มปุ่ม Make Call โดยใช้ตัวช่วยเทเลโฟนนี่ได้โดยง่าย ซึ่งไม่จำเป็นต้องทำการอิมพลีเมนต์ตัวควบคุมที่มีในแอพพลิเคชัน TAPI ทั้งหมด

4.5.9 Dispatch Mapper

Dispatch Mapper ถูกสร้างโดยใช้ CoCreateInstance ของ COM และมีอินเทอร์เฟซตัวหนึ่งมาด้วยคือ ITDispatchMapper อินเทอร์เฟซนี้อนุญาตให้แอพพลิเคชันรับดิสแพตช์พอยน์เตอร์ (dispatch pointer) ของอินเทอร์เฟซอีกตัวบนออบเจ็กต์หนึ่งมาให้กับอินเทอร์เฟซนั้นและ GUID ของอีกอินเทอร์เฟซหนึ่ง อินเทอร์เฟซนี้ถูกสร้างขึ้นเพื่อช่วยโปรแกรมเมอร์ใช้ในการเขียนแอพพลิเคชัน ซึ่งไม่ได้ช่วยในการดึงอินเทอร์เฟซของออบเจ็กต์มาใช้ Dispatch Mapper นี้ใช้อินเทอร์เฟซ IObjectSafety ของออบเจ็กต์เพื่อให้แน่ใจว่าออบเจ็กต์นั้นมีความปลอดภัยสำหรับการเขียนอินเทอร์เฟซที่ต้องการ ถ้าออบเจ็กต์ไม่ทำการอิมพลีเมนต์ IObjectSafety หรือถ้าออบเจ็กต์นั้นไม่ปลอดภัยกับอินเทอร์เฟซนี้ การ Call ก็จะล้มเหลว

4.6 ตัวอย่างโค้ดของการทำ Initialize TAPI

จะแสดงถึงการสร้างออบเจ็กต์ TAPI ดังนี้

ภาษา C++

```
// Initialize COM
CoInitializeEx(
    NULL,
    COINIT_MULTITHREADED
    );

// Create a TAPI entry point object.
CoCreateInstance(
    CLSID_TAPI,
    NULL,
    CLSCTX_INPROC_SERVER,
    IID_ITTAPI,
    (LPVOID *)&gpTapi
    );

// Initialize TAPI
gpTapi->Initialize();
```

ภาษา Visual Basic

```
'Usually declared globally
Dim gobjTapi As TAPI

Dim glRegistrationToken As Long
```

```
Dim gobjReceivedCallInfo As ITCallInfo
Dim gbSupportedCall As Boolean

'Usually performed during form load
'Create the tapi object.
Set gobjTapi = New TAPI

'Call Initialize before calling any other TAPI function.
Call gobjTapi.Initialize
```

4.7 ตัวอย่างโค้ดของการเลือกแอดเดรส

แสดงการใช้ออบเจ็กต์ TAPI ทำการตรวจสอบทรัพยากรเทเลโฟนนี่ที่มีอยู่สำหรับแอดเดรสที่สามารถรองรับความต้องการของชนิดของมีเดียที่ต้องการเหล่านั้นได้ ซึ่งโค้ดที่แสดงนี้ตัวออดิโอและวีดีโอเป็นมีเดียที่ต้องการ และก่อนที่จะใช้โค้ดนี้ได้ จะต้องทำการ Initialize TAPI ก่อน

ภาษา C++

```
// Use the TAPI object to
// enumerate available addresses.
gpTapi->EnumerateAddresses( &pIEnumAddress );

// Locate an address that can support the
// media type the application needs.
while ( S_OK == pEnumAddresses->Next(1, &pAddress, NULL) ) )
{
    // Determine the media support.
    pAddress->QueryInterface(
        IID_ITMediaSupport,
        (void **)&pMediaSupport
        );

    // In this snippet, the required media type is already known.
    // The application can also use the address object to
    // enumerate the media supported, then choose from there.
    pMediaSupport->QueryMediaType(
        TAPIMEDIATYPE_AUDIO|TAPIMEDIATYPE_VIDEO,
        &bSupport
        );
    if (bSupport)
    {
        break;
    }
}
// pAddress is now a usable address.
```

ภาษา Visual Basic

```
'pick up the collection of Address objects
Dim gobjAddress As ITAddress
Dim objCollAddresses As ITCollection
Set objCollAddresses = gobjTapi.Addresses

'find address that supports the desired type, nSelectedType
bFound = False
For indexAddr = 1 To objCollAddresses.Count
    Set objCrtAddress = objCollAddresses.Item(indexAddr)
    Set objMediaSupport = objCrtAddress
    Set objAddressCapabilities = objCrtAddress
```

```
    If objMediaSupport.QueryMediaType( nSelectedType )
        bFound = True
    End If

    Set objAddressCapabilities = Nothing
    Set objMediaSupport = Nothing
    Set objCrtAddress = Nothing

    If bFound = True Then Exit For
Next indexAddr

Set gobjAddress = objcollAddress.Item(indexAddr)

'If no errors occurred, gobjAddress is now a usable address.
```

4.8 ตัวอย่างโค้ดของการลงทะเบียนอีเวนต์

แสดงถึงการอิมพลีเมนต์ตัวจัดการอีเวนต์อย่างง่าย ทำการลงทะเบียนอินเทอร์เฟซอีเวนต์หลักของ TAPI ทำการเซตตัวกรองอีเวนต์ และลงทะเบียนสำหรับการแจ้งการ Call ที่มี โดยก่อนที่จะสามารถใช้โค้ดการลงทะเบียนอีเวนต์ได้นั้น จะต้องทำการ Initialize TAPI และเลือกแอดเดรสก่อน

ภาษา C++

```
//
// Snippet 1: Implement a simple event handler.
//
HRESULT STDMETHODCALLTYPE
CTAPIEventNotification::Event(
    TAPI_EVENT  TapiEvent,
    IDispatch * pEvent
    )
{
    // AddRef the event so it doesn't go away.
    pEvent->AddRef();

    // Post a message to our own UI thread.
    PostMessage(
        ghDlg,
        WM_PRIVATETAPIEVENT,
        (WPARAM) TapiEvent,
        (LPARAM) pEvent
        );

    return S_OK;
}

//
// Snippet 2: Register the event interface.
//
ULONG                           gulAdvise;  // Globally declared
IConnectionPointContainer    * pCPC;
IConnectionPoint             * pCP;
ITTAPIEventNotification      * gpTAPIEventNotification;

gpTapi->QueryInterface(
     IID_IConnectionPointContainer,
     (void **)&pCPC
     );
```

```
pCPC->FindConnectionPoint(
     IID_ITTAPIEventNotification,
     &pCP
     );

pCPC->Release();

// Retain gulAdvise for future use
pCP->Advise(
     gpTAPIEventNotification,
     &gulAdvise
     );

pCP->Release();

//
// Snippet 3: Set the event filter.
//
// Assume we are interested
// only in call-related events.
gpTapi->put_EventFilter(
   TE_CALLNOTIFICATION | TE_CALLSTATE | TE_CALLMEDIA
   );

//
// Snippet 4: Register an address with TAPI
// for call notifications. Assume we are interested
// in video and audio calls, and that pAddress
// is a pointer to the ITAddress interface of
// an address that can handle both media types.

long glRegister; // Globally declared

gpTapi->RegisterCallNotifications(
    pAddress,
    VARIANT_TRUE,    // monitor privileges
    VARIANT_TRUE,    // owner privileges
    TAPIMEDIATYPE_AUDIO|TAPIMEDIATYPE_VIDEO,
    gulAdvise,       // As returned by Advise
    &glRegister
    );
```

ภาษา Visual Basic

```
'Usually declared globally
Dim WithEvents gobjTapiWithEvents As TAPI
Attribute gobjTapiWithEvents.VB_VarHelpID = -1
Dim glRegistrationToken As Long

'Usually performed at the same time
'as TAPI initialization
Const TAPI3_CALL_EVENTS = _
    TE_CALLMEDIA Or _
    TE_CALLNOTIFICATION Or _
    TE_CALLSTATE

'Set the EventFilter to accept all defined TAPI events.
gobjTapi.EventFilter = TAPI3_CALL_EVENTS

'Register the outgoing interface (the one that will actually
```

```
'receive and process the events).
Set gobjTapiWithEvents = gobjTapi
Dim fOwner As Boolean, fMonitor As Boolean
Dim lMediaTypes As Long, lCallbackInstance As Long

'fOwner = True ensures that the application receives incoming calls
'and their call state events.
fOwner = True
fMonitor = False
lMediaTypes = TAPIMEDIATYPE_AUDIO
lCallbackInstance = 1

glRegistrationToken = gobjTapi.RegisterCallNotifications( _
    gobjAddress, _
    fMonitor, _
    fOwner, _
    lMediaTypes, _
    lCallbackInstance
    )
```

4.9 ตัวอย่างโค้ดของการเลือกเทอร์มินอล

แสดงการเลือกเทอร์มินอลสำหรับสตรีมของข้อมูลที่เกี่ยวข้องกับการ Call โดยก่อนที่จะสามารถใช้โค้ดนี้ได้ จะต้องทำการ Initialize TAPI และเลือกแอดเดรสเสียก่อน และแอพพลิเคชันจะต้องมีพอยน์เตอร์ชี้ไปยังอินเทอร์เฟซ ITBasicCallControl ไม่ว่าจะเป็นการ Call ที่เข้ามาหรือออกไปก็ตาม

ภาษา C++

```
// pAddress is an ITAddress interface pointer.
// pBasicCall is an ITBasicCallControl interface pointer.

// Get the ITStreamControl interface.
ITStreamControl * pStreamControl;
pBasicCall->QueryInterface(
    IID_ITStreamControl,
    (void **) &pStreamControl
    );

// Enumerate the streams and select
// terminals onto them.
IEnumStream * pEnumStreams;
ITStream    * pStream;
pStreamControl->EnumerateStreams(&pEnumStreams);
while ( S_OK == pEnumStreams->Next(1, &pStream, NULL) )
    {
        // Get the media type and direction of this stream.
        long                lMediaType;
        TERMINAL_DIRECTION  dir;
        pStream->get_MediaType( &lMediaType );
        pStream->get_Direction( &dir );

        // Create the default terminal for this media type
        // and direction.
        //   If lMediaType == TAPIMEDIATYPE_VIDEO and
        //   dir == TD_RENDER, a dynamic video render terminal
        //   is required. Please see Incoming.cpp in
        //   the samples section of the SDK.
        // For all other terminals, get the default
        // static terminal.
```

```
        ITTerminal * pTerminal;
        ITTerminalSupport * pTerminalSupport;
        pAddress->QueryInterface(
            IID_ITTerminalSupport,
            (void **)&pTerminalSupport
            );
        pTerminalSupport->GetDefaultStaticTerminal(
            lMediaType,
            dir,
            pTerminal
            );
        // Select the terminal on the stream.
        pStream->SelectTerminal(pTerminal);
    }
```

ภาษา Visual Basic

```
'query for ITBasicCallControl, the call control interface
Dim objCallControl As ITBasicCallControl
Set objCallControl = gobjReceivedCallInfo

'query ITTerminalSupport from Address object
Dim objTerminalSupport As ITTerminalSupport
Set objTerminalSupport = gobjAddress

Dim objTerminal As ITTerminal
Set objTerminal = objTerminalSupport.GetDefaultStaticTerminal _
    lMediaType, dir)

'release not needed objects
Set objTerminalSupport = Nothing

'Select the terminal before answering
'we'll need the ITStreamControl interface for doing this.

Dim objStreamControl As ITStreamControl
Set objStreamControl = objCallControl

If Not (objStreamControl Is Nothing) Then
    Dim objITCollStreams As ITCollection

    Set objITCollStreams = objStreamControl.Streams

    Dim nIndex As Long, objCrtStream As ITStream

    For nIndex = 1 To objITCollStreams.Count
        Set objCrtStream = objITCollStreams.Item(nIndex)
        If objCrtStream.MediaType = lMediaType  Then
            If objCrtStream.Direction = dir Then
                Call objCrtStream.SelectTerminal(objTerminal)
            End If
        End If
        Set objCrtStream = Nothing
    Next nIndex

    Set objITCollStreams = Nothing
    Set objStreamControl = Nothing
End If
```

4.10 ตัวอย่างโค้ดของการทำการ Call

แสดงการสร้างออบเจ็กต์การ Call ทำการค้นหาสตรีมที่เกี่ยวข้องกับการ Call ทำการเลือกและสร้างเทอร์มินอลที่เหมาะสม ทำการเลือกเทอร์มินอลที่ใช้สำหรับสตรีมนั้นๆ และจัดการในการติดต่อกัน โดยก่อนที่จะสามารถใช้โค้ดนี้ได้ จะต้องทำการ Initialize TAPI และเลือกแอดเดรสเสียก่อน

ภาษา C++

```
// Specify the destination address.
//
// szAddressToCall and
// dwAddressType have been
// retrieved from a user interface.
ITBasicCallControl * pBasicCall
bstrAddressToCall = SysAllocString( szAddressToCall );
pAddress->CreateCall(
    bstrAddressToCall,
    dwAddressType,
    &pBasicCall
 );

// Create the required terminals for this call.
{
    // See the Select a Terminal code snippet.
}

// Make the connection.
pBasicCall->Connect( TRUE );
```

ภาษา Visual Basic

```
Set gobjCall = gobjOrigAddress.CreateCall(_
    strDestAddress, nSelectedType, lMediaTypes)

'Create the required terminals for this call.
{
    'See the Select a Terminal code snippet.
}

'Connect the call; False means that the call is made asynchronously.
gobjCall.Connect (False)
```

4.11 ตัวอย่างโค้ดของการรับการ Call

แสดงถึงการจัดการแจ้งการ Call ใหม่ที่มีเข้ามา เช่น ค้นหาหรือสร้างเทอร์มินอลที่เหมาะสมในการเรนเดอร์ (render) มีเดีย โค้ดนี้เป็นส่วนหนึ่งของส่วนในการทำสวิตช์เท่านั้นซึ่งแอพพลิเคชันต้องอิมพลีเมนต์สำหรับการจัดการอีเวนต์ โดยตัวโค้ดอาจจะถูกใส่ไว้ในการอิมพลีเมนต์ ITTAPIEventNotification::Event หรือเมธอดอีเวนต์อาจจะส่งเมสเซจไปยังเธรดที่ทำงานซึ่งประกอบไปด้วยสวิตช์ ก่อนที่จะสามารถใช้โค้ดนี้ได้นั้นจะต้องทำการ Initialize TAPI เลือกแอดเดรสและทำการลงทะเบียนอีเวนต์เสียก่อน รวมถึงจะต้องทำการเลือกเทอร์มินอลที่มีการรับพอยน์เตอร์อินเทอร์เฟซของ ITBasicCallControl และ ITAddress ด้วย

ภาษา C++

```
// pEvent is an IDispatch pointer for the
```

```
// ITCallNotificationEvent interface of the
// call object created by TAPI, and is passed
// into the event handler by TAPI.

Case TE_CALLNOTIFICATION:
{
    // Get the ITCallNotification interface.
    ITCallNotificationEvent * pNotify;
    pEvent->QueryInterface(
        IID_ITCallNotificationEvent,
        (void **)&pNotify
        );

    // Get the ITCallInfo interface.
    ITCallInfo * pCallInfo;
    pNotify->get_Call( &pCallInfo);

    // Get the ITBasicCallControl interface.
    ITBasicCallControl *    pBasicCall;
    pCallInfo->QueryInterface(
        IID_ITBasicCallControl,
        (void**)&pBasicCall
        );

    // Get the ITAddress interface.
    ITAddress * pAddress;
    pCallInfo->get_Address( &pAddress );

    // Create the required terminals for this call.
    {
        // See the Select a Terminal code snippet.
    }
    // Complete incoming call processing.
    pBasicCall->Answer();
}
```

ภาษา Visual Basic

```
' pEvent is an IDispatch pointer for the
' ITCallNotificationEvent interface of the
' call object created by TAPI, and is passed
' into the event handler by TAPI.
If TapiEvent = TE_CALLNOTIFICATION Then
    ' Get the ITCallNotification interface.
    Dim objCallNotificationEvent As ITCallNotificationEvent
    Set objCallNotificationEvent = pEvent

   'query ITCallInfo interface for the new call, and store it
    Dim gobjReceivedCallInfo As ITCallInfo
    Set gobjReceivedCallInfo = objCallNotificationEvent.Call

    ' Get the ITBasicCallControl interface.
    Dim objCallControl As ITBasicCallControl
    Set objCallControl = gobjReceivedCallInfo

    ' Create the required terminals for this call.
    ' See the Select a Terminal code snippet.

    'Answer
    objCallControl.Answer
End If
```

4.12 ความแตกต่างของ TAPI 3.0 กับเวอร์ชันอื่นๆ

TAPI 3.0 เป็นเวอร์ชันที่พัฒนามาจากเวอร์ชัน 2.x ซึ่งเป็นลักษณะของ COM โดยสามารถใช้กับวินโดวส์ 2000 เท่านั้น ไม่สามารถนำแอพพลิเคชันมาใช้ในวินโดว์สรุ่นก่อนๆ ได้อย่าง Windows 9x/NT ทำให้การนำมาใช้ยังไม่ค่อยมีความยืดหยุ่น แต่เนื่องจากการเป็นลักษณะของ COM ทำให้สามารถพัฒนาแอพพลิเคชันได้ในหลายภาษา แต่ถ้าเป็นเวอร์ชัน 2.x จะสามารถใช้ในภาษา C เท่านั้น หากต้องการใช้กับภาษาอื่นจะต้องทำเป็นคอมโพเน็นต์ก่อน

บทที่ 5
DirectShow

5.1 บทนำ

Microsoft® DirectShow® เป็นสถาปัตยกรรมสำหรับการทำสตรีมมิ่งของมีเดียต่างๆ บนแพลตฟอร์มของ Microsoft® Windows® ตัว DirectShow สามารถทำการเก็บสตรีมของข้อมูลมัลติมีเดียที่มีคุณภาพสูงและสามารถเล่นสตรีมเหล่านั้นออกไปได้ด้วย ซึ่งสนับสนุนรูปแบบต่างๆ ของข้อมูลมากมาย เช่น Motion Picture Experts Group (MPEG), Audio-Video Interleaved (AVI), MPEG Audio Layer-3 (MP3) และไฟล์ WAV รวมทั้งสนับสนุนการเก็บข้อมูลโดยใช้ Windows Driver Model (WDM) หรือเป็นตัวที่เก่ากว่า DirectShow ถูกรวมเข้ากับเทคโนโลยี DirectX อื่นๆ มันจะทำการตรวจสอบและใช้ฮาร์ดแวร์ของวีดีโอและเสียงโดยอัตโนมัติ

DirectShow ทำเรื่องการเล่นมีเดียต่างๆ การแปลงชนิดของมีเดีย และการเก็บข้อมูลมีเดียให้เป็นเรื่องที่ง่าย ในขณะเดียวกันก็จะสามารถเข้าถึงสถาปัตยกรรมของการควบคุมของสตรีมสำหรับแอพพลิเคชันต่างๆ ที่ต้องการทางเลือกของตน และสามารถสร้างคอมโพเน็นต์ของ DirectShow ขึ้นมาเองได้เพื่อสนับสนุนรูปแบบและเสียงประกอบใหม่ๆ

DirectShow อยู่บนเทคโนโลยีของ Component Object Model (COM) ซึ่งหากต้องการที่จะเขียนแอพพลิเคชันที่เกี่ยวกับ DirectShow จะต้องเข้าใจถึงการเขียน COM แบบไคลเอนต์ด้วย สำหรับแอพพลิเคชันโดยส่วนใหญ่แล้ว จะไม่ต้องทำการอิมพลีเมนต์ออบเจ็กต์ COM ของตัวเอง ซึ่ง DirectShow จะมีคอมโพเน็นต์มาให้อยู่แล้ว นอกจากเสียว่าต้องการที่จะขยายความสามารถของ DirectShow โดยเขียนคอมโพเน็นต์ของตัวเองขึ้นมา ก็จะต้องอิมพลีเมนต์ให้เป็นออบเจ็กต์ของ COM ด้วย

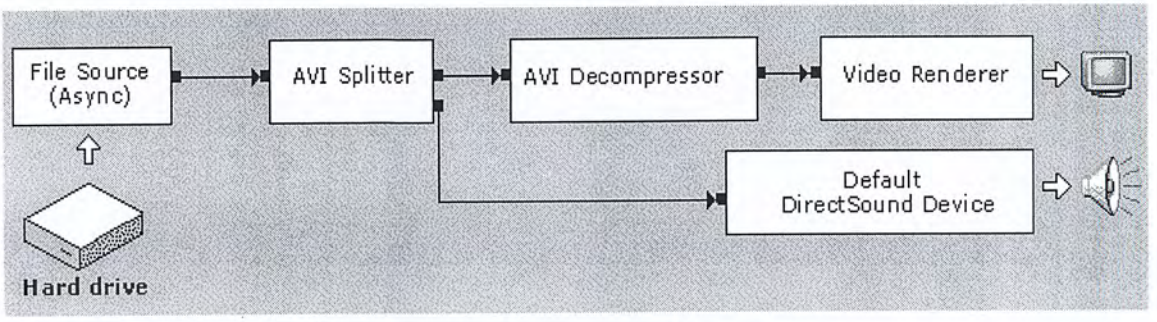
5.2 ฟิลเตอร์กราฟ (Filter Graphs)

บล็อกพื้นฐานที่สร้างขึ้นมาของ DirectShow เป็นคอมโพเน็นต์ของซอฟต์แวร์เรียกว่า ฟิลเตอร์ (filter) โดยทั่วไปตัวของฟิลเตอร์ตัวหนึ่งจะทำขั้นตอนเพียงขั้นตอนเดียวกับสตรีมของข้อมูลมัลติมีเดีย ตัวอย่างเช่น มีฟิลเตอร์ของ DirectShow ที่ทำการ

- อ่านไฟล์ต่างๆ
- รับข้อมูลวีดีโอจากอุปกรณ์วีดีโอแคปเจอร์
- ทำการถอดรหัสสัญญาณชนิดของสตรีม เช่น วีดีโอ MPEG-1
- ทำการส่งข้อมูลไปยังหน่วยแสดงผลหรือการ์ดเสียง

ฟิลเตอร์รับค่าอินพุตและสร้างค่าเอาต์พุตออกมา ตัวอย่างเช่น ถ้าฟิลเตอร์ทำการถอดรหัสสัญญาณวีดีโอ MPEG-1 ตัวอินพุตจะเป็นสตรีมที่เข้ารหัสสัญญาณ MPEG และเอาต์พุตจะเป็นสตรีมวีดีโอ RGB ที่ไม่ได้ทำการบีบอัด ในการทำงานนั้น แอพพลิเคชันจะต่อฟิลเตอร์ต่างๆ เข้าด้วยกันทำให้เอาต์พุต

จากฟิลเตอร์หนึ่งกลายเป็นอินพุตสำหรับอีกฟิลเตอร์หนึ่ง กลุ่มของฟิลเตอร์ที่เชื่อมต่อกันเรียกว่า ฟิลเตอร์กราฟ (filter graph) ซึ่งสามารถแสดงได้ดังในรูปที่ 5-1



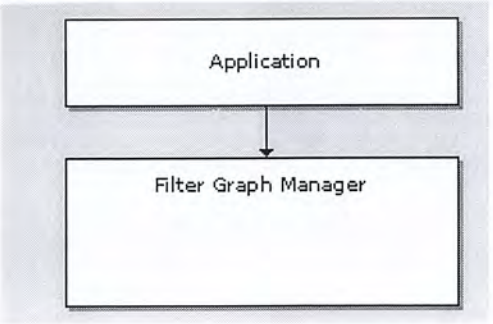
รูปที่ 5-1 ฟิลเตอร์กราฟสำหรับการเล่นไฟล์นามสกุล AVI

แอพพลิเคชันที่สร้างขึ้นไม่จำเป็นต้องมาจัดการฟิลเตอร์เดี่ยวๆ ในฟิลเตอร์กราฟเอง แต่ DirectShow มีคอมโพเน็นต์ในระดับสูงไว้ให้ที่เรียกว่า ตัวจัดการฟิลเตอร์กราฟ (Filter Graph Manager) ซึ่งจะทำการควบคุมการไหลของข้อมูลผ่านกราฟ แอพพลิเคชันจะทำการคอลของ API ในระดับสูง เช่น "Run" ในการทำการเคลื่อนย้ายข้อมูลผ่านกราฟ หรือ "Stop" เพื่อหยุดการไหลของข้อมูล ถ้าหากต้องการที่จะควบคุมสตรีมโดยตรง ก็สามารถเข้าถึงฟิลเตอร์ได้โดยตรงผ่านอินเทอร์เฟซของ COM และตัวจัดการฟิลเตอร์กราฟทำการส่งผ่านการประกาศอีเวนต์ต่างๆ (event notifications) ให้กับแอพพลิเคชันด้วย ทำให้แอพพลิเคชันสามารถตอบสนองกับอีเวนต์นั้นๆ ได้ เช่น การสิ้นสุดของสตรีม นอกจากนั้นตัวจัดการฟิลเตอร์กราฟยังทำให้การสร้างฟิลเตอร์กราฟเป็นไปอย่างง่าย เช่น เพียงแต่ระบุชื่อไฟล์ ตัวจัดการฟิลเตอร์กราฟก็จะสร้างกราฟขึ้นในการเล่นไฟล์นั้น

5.3 การเขียนแอพพลิเคชัน DirectShow

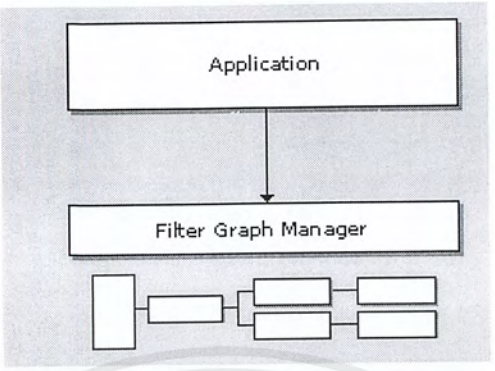
โดยทั่วไปแอพพลิเคชันของ DirectShow จะปฏิบัติตาม 3 ขั้นตอนพื้นฐาน ดังที่แสดงในรูปภาพไดอะแกรมดังต่อไปนี้

1. สร้างอินสแทนซ์ของตัวจัดการฟิลเตอร์กราฟโดยใช้ฟังก์ชัน CoCreateInstance ดังรูปที่ 5-2



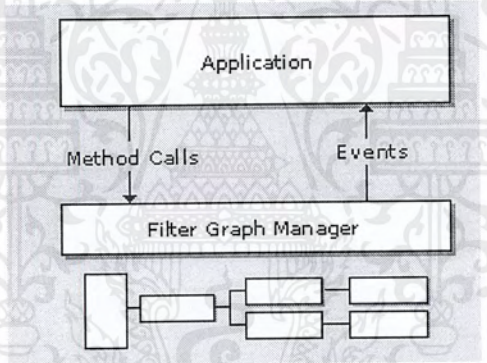
รูปที่ 5-2 การสร้างอินสแทนซ์ตัวจัดการฟิลเตอร์กราฟ

2. ใช้ตัวจัดการฟิลเตอร์กราฟทำการสร้างฟิลเตอร์กราฟ (อาจจะใช้คอมโพเน็นต์ตัวช่วยของ DirectShow ตัวอื่นก็ได้) ดังรูปที่ 5-3



รูปที่ 5-3 ตัวจัดการฟิลเตอร์กราฟทำการสร้างฟิลเตอร์กราฟขึ้นมา

3. ควบคุมฟิลเตอร์กราฟและตอบสนองต่ออีเวนต์ต่างๆ ดังรูปที่ 5-4



รูปที่ 5-4 การทำงานของตัวจัดการฟิลเตอร์กราฟ

5.4 วิธีในการเล่นไฟล์

ตัวอย่างในการเล่นไฟล์มีเดียสำหรับ Microsoft® DirectShow® ซึ่งเป็นภาษา C++ เพียงไม่กี่บรรทัด การเล่นไฟล์ประกอบไปด้วย 3 ขั้นตอนคือ

1. สร้างอินสแทนซ์ของตัวจัดการฟิลเตอร์กราฟ
2. ใช้ตัวจัดการฟิลเตอร์กราฟทำการสร้างฟิลเตอร์กราฟ
3. ใช้ตัวจัดการฟิลเตอร์กราฟทำการรันฟิลเตอร์กราฟ

ซึ่งจะต้องใช้ 2 อินเทอร์เฟซดังนี้

- IGraphBuilder: มีหน้าที่สร้างฟิลเตอร์กราฟ
- IMediaControl: มีหน้าที่จัดการกับการสตรีมมิ่งของมีเดียในฟิลเตอร์กราฟ

ตัวอย่างโค้ดในภาษา C++ มีดังนี้

```
#include <streams.h>

void __cdecl main(void)
{
    IGraphBuilder *pGraph;
    IMediaControl *pMediaControl;
    CoInitialize(NULL);

    // Create the filter graph manager.
    CoCreateInstance(CLSID_FilterGraph, NULL, CLSCTX_INPROC,
                     IID_IGraphBuilder, (void **)&pGraph);
    pGraph->QueryInterface(IID_IMediaControl, (void
**)&pMediaControl);

    // Build the graph. (IMPORTANT: Change string to a file on your
system.)
    pGraph->RenderFile(L"C:\\Hello_World.avi", NULL);

    // Run the graph.
    pMediaControl->Run();

    // Block until the user clicks the OK button.
    // The filter graph runs on a separate thread.
    MessageBox(NULL, "Click me to end playback.", "DirectShow",
MB_OK);

    // Clean up.
    pMediaControl->Release();
    pGraph->Release();
    CoUninitialize();
}
```

เมธอด IGraphBuilder::RenderFile สร้างฟิลเตอร์กราฟเพื่อทำการเล่นไฟล์ที่ระบุเอาไว้ พารามิเตอร์แรกเป็นชื่อไฟล์ซึ่งจะต้องเป็นชนิดของสตริงที่เป็น Unicode แบบ wide character (ขนาด 2 ไบต์) โดยในตัวอย่างจะเป็นสตริงของตัวอักษรและ ตัวนำหน้า "L" จะทำการแปลงสตริงที่เป็นแอสกีให้เป็นสตริงแบบ wide character ส่วนพารามิเตอร์ที่สองได้สงวนเอาไว้และต้องเป็น NULL

หลังจากที่ตัวจัดการฟิลเตอร์กราฟได้สร้างฟิลเตอร์กราฟแล้วก็พร้อมที่จะทำการเล่นไฟล์ เมธอด IMediaControl::Run ทำการย้ายกราฟไปอยู่ในโหมดของการรัน เมื่อแอพพลิเคชันเรียกเมธอดนี้ ข้อมูลมีเดียก็จะเริ่มเคลื่อนที่ผ่านฟิลเตอร์กราฟและแปลงสัญญาณเป็นสัญญาณวีดีโอ เสียง หรือทั้งสองอย่าง หากเมธอดการรันเป็นคำสั่งสุดท้ายของโปรแกรมแล้ว ตัวโปรแกรมจะหยุดการเล่นทันที ซึ่งในตัวแอพพลิเคชันจริงๆ จะต้องรออีเวนต์ของการจบของสตรีมด้วย เพื่อที่จะให้แซมเปิ้ลสั้นที่สุดเท่าที่จะเป็นไปได้ จึงต้องแสดง message box ซึ่งจะขัดจังหวะการทำงานของการรันของโปรแกรม การเล่นจะทำไปเรื่อยๆ โดยแยกออกเป็นเธรดต่างหากจนกระทั่งผู้ใช้คลิ๊กปุ่ม OK สุดท้ายควรที่จะทำการเคลียร์ออบเจ็กต์ต่างๆ ทิ้งด้วย

มิเช่นนั้นอาจจะเกิดข้อผิดพลาดได้ และในการสร้างแอพพลิเคชันจริงๆ ควรจะตรวจสอบค่าที่คืนให้มาของ RenderFile เพื่อที่จะยืนยันได้ว่าได้สร้างฟิลเตอร์กราฟสำเร็จแล้ว

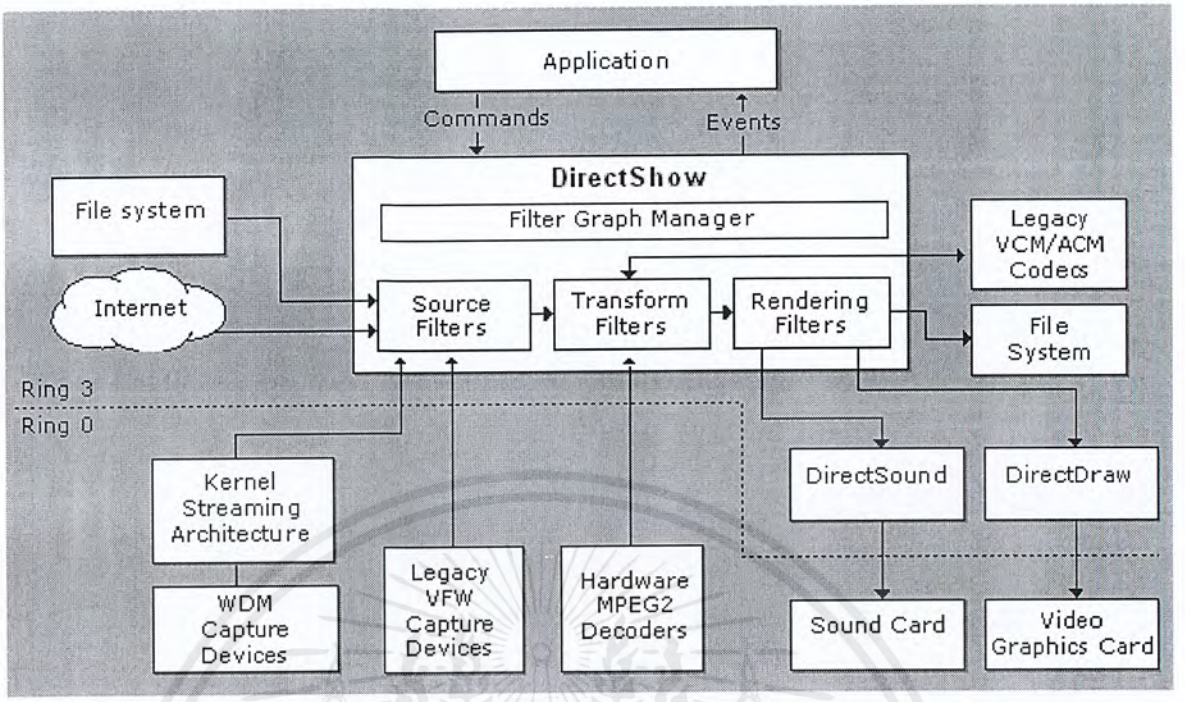
5.5 ภาพรวมของระบบ DirectShow

การทำงานกับข้อมูลมัลติมีเดียเผชิญกับปัญหาต่างๆ ดังนี้

- สตรีมของข้อมูลมัลติมีเดียประกอบไปด้วยข้อมูลขนาดใหญ่มากมายซึ่งจะต้องประมวลผลอย่างเร็วมาก
- สตรีมเสียง วีดีโอ และสตรีมอื่นๆ ต้องเริ่มและหยุดในเวลาเดียวกัน รวมถึงต้องเล่นในอัตราความเร็วที่เท่ากันด้วย
- สตรีมสามารถมาจากหลายแหล่ง ซึ่งรวมถึงไฟล์ที่อยู่ในเครื่อง บนระบบเครือข่าย กล้องวีดีโอ และอุปกรณ์อื่นๆ
- สตรีมมาจากหลายประเภท เช่น Audio-Video Interleaved (AVI), Motion Picture Experts Group (MPEG), Digital Video (DV) และ Motion JPEG (MJPEG)
- แอพพลิเคชันไม่มีทางที่จะทราบได้ว่าอุปกรณ์ฮาร์ดแวร์ตัวไหนจะมีอยู่ในระบบของผู้ใช้

DirectShow ได้ถูกออกแบบมาเพื่อรองรับกับปัญหาต่างๆ เหล่านี้ ซึ่งเป้าหมายหลักในการออกแบบอยู่ที่การทำให้การสร้างแอพพลิเคชันมัลติมีเดียบนแพลตฟอร์มวินโดว์สเป็นไปได้โดยง่าย DirectShow ใช้ DirectDraw® และ DirectSound® ในการแปลงข้อมูลได้อย่างมีประสิทธิภาพตามการ์ดจอและการ์ดเสียงของระบบที่ใช้ เพื่อให้ได้ทรูพุตตามที่จำเป็นสำหรับการทำสตรีมมิ่งข้อมูลวีดีโอและเสียง DirectShow ได้ใช้สถาปัตยกรรมที่เป็นโมดุลในการทำงานกับคอมโพเน็นต์ของระบบที่เรียกว่า ฟิลเตอร์ ซึ่งสามารถผสมและจับคู่ให้เหมาะสมได้หลายรูปแบบ ทำให้สามารถควบคุมแหล่งของมีเดีย รูปแบบและอุปกรณ์ทางด้านฮาร์ดแวร์ต่างๆ ได้

DirectShow มีฟิลเตอร์ที่สนับสนุนการแคปเจอร์สัญญาณข้อมูลมัลติมีเดียและอุปกรณ์การปรับสัญญาณ ซึ่งอยู่บนพื้นฐานของ Windows Driver Model (WDM) รวมถึงฟิลเตอร์ที่สนับสนุนการ์ดแคปเจอร์ legacy Video for Windows (VfW) และตัวเข้ารหัสสัญญาณอื่นๆ ที่ถูกเขียนขึ้นสำหรับอินเทอร์เฟซของ Audio Compression Manager (ACM) และ Video Compression Manager (VCM) ในรูปที่ 5-2 แสดงไดอะแกรมของความสัมพันธ์ระหว่างแอพพลิเคชัน DirectShow บริการของระบบปฏิบัติการอื่นๆ และอุปกรณ์ทางด้านฮาร์ดแวร์



รูปที่ 5-5 ความสัมพันธ์ระหว่างแอพพลิเคชัน คอมโพเน็นต์ของ DirectShow รวมถึงซอฟต์แวร์และฮาร์ดแวร์ที่ DirectShow สนับสนุน

จากในรูปที่ 5-5 แสดงนั้น DirectShow ทำให้แอพพลิเคชันสามารถเล่นไฟล์และสตรีมจากหลายๆ แหล่งได้ รวมทั้งไฟล์ที่อยู่ในฮาร์ดดิสก์ ในไดรฟ์ซีดีรอม ไดรฟ์ DVD ไฟล์ที่อยู่บนเครือข่าย การ์ด TV-tuner และการ์ดแคปเจอร์วีดีโอ ซึ่งอยู่บนพื้นฐานของ Windows Driver Model (WDM) และการ์ดแคปเจอร์วีดีโอ Video For Windows® ตัว DirectShow มีตัวบีบอัดและตัวขยายสำหรับบางชนิดของไฟล์รวมถึงตัวถอดรหัสสัญญาณที่เป็นฮาร์ดแวร์และซอฟต์แวร์ของเธิร์ดพาร์ตี้ซึ่งคอมแพตทิเบิลกับ DirectShow นอกจากนี้ DirectShow ยังสนับสนุนตัวเข้ารหัสสัญญาณ VfW ที่อยู่บนอินเทอร์เฟซ Video Compression Manager (VCW) และ Audio Compression Manager (ACM) การเล่นทำให้การใช้ความสามารถของ DirectDraw และ DirectSound เมื่อมีฮาร์ดแวร์ที่สนับสนุน

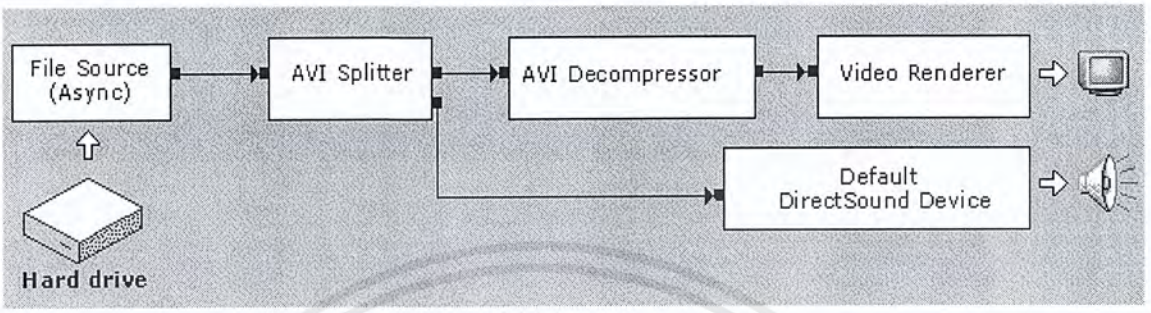
5.6 ฟิลเตอร์กราฟและคอมโพเน็นต์ต่างๆ ของมัน

แอพพลิเคชันไม่จำเป็นต้องจัดการกับการทำงานภายในของฟิลเตอร์กราฟของ DirectShow ในสถานการณ์ส่วนใหญ่ รายละเอียดเช่นการเพิ่มฟิลเตอร์และการเชื่อมต่อของพิน (pins) จะถูกจัดการอย่างอัตโนมัติโดยตัวจัดการฟิลเตอร์กราฟรวมถึงฟิลเตอร์และพินของตัวมันเอง

5.6.1 ฟิลเตอร์กราฟ

เมื่อใดก็ตามที่ไฟล์มีเดียหรือสตรีมถูกเล่น อัด แคปเจอร์ กระจายสัญญาณ หรือถูกประมวลผลใดๆ มันจะทำโดยการเชื่อมต่อกันของฟิลเตอร์ตั้งแต่หนึ่งตัวขึ้นไปอย่างเป็นระบบที่เรียกว่า ฟิลเตอร์กราฟ

ขบวนการสร้างกราฟสามารถถูกทำโดยแอพพลิเคชันเองหรือถูกสร้างโดยอัตโนมัติจากตัวสร้างกราฟ (Graph Builder) หรือตัวจัดการฟิลเตอร์กราฟก็ได้ ในทั้งสองกรณีนี้ปกติการประมวลผลเริ่มด้วยฟิลเตอร์ต้นทาง (source filter) และขึ้นอยู่กับปัจจัยอยู่ 2 อย่างเสมอ คือ จำนวนของสตรีมและชนิดของมีเดียเหล่านั้น ซึ่งฟิลเตอร์เป็นอินพุต ส่วนจำนวนของสตรีมและชนิดของมีเดียเหล่านั้นถือเป็นเอาท์พุต



รูปที่ 5-6 ฟิลเตอร์กราฟอย่างง่ายในการเล่นไฟล์ AVI ที่มีการบีบอัดสัญญาณวีดีโอ

จากในรูปที่ 5-6 ลูกศรในกราฟแสดงทิศทางการไหลของข้อมูล ฟิลเตอร์ต้นทางในที่นี้เป็นแหล่งที่มาจากไฟล์แบบอะซิงโครนัส ฟิลเตอร์นี้ทำการอ่านข้อมูลเป็นไบต์จากไฟล์และส่งต่อไปยังฟิลเตอร์ที่เป็นตัวแยก AVI (AVI Splitter) ซึ่งรู้ชนิดของข้อมูลที่เป็น AVI ซึ่งรวมถึงการบีบอัดสัญญาณวีดีโอด้วย ตัวแยก AVI นี้จะกระจายข้อมูลเป็นแซมเปิ้ล (sample) ของมีเดียและส่งวีดีโอแซมเปิ้ลไปยังตัวขยายการบีบอัดสัญญาณ AVI วีดีโอแซมเปิ้ลแต่ละตัวประกอบไปด้วยหนึ่งเฟรมของวีดีโอ ตัวขยายการบีบอีดสัญญาณ AVI หาตัวถอดสัญญาณที่เหมาะสมเพื่อทำการขยายการบีบอัดสัญญาณแซมเปิ้ลและส่งเฟรมของวีดีโอที่ถูกขยายแล้วไปยังฟิลเตอร์ของตัวเรนเดอร์สัญญาณวีดีโอ (video renderer) ซึ่งแสดงสัญญาณวีดีโอบนหน้าจอคอมพิวเตอร์ เนื่องจากแซมเปิ้ลของเสียงไม่ได้มีการบีบอัด ตัวแยก AVI สามารถเชื่อมต่อได้โดยตรงกับตัวเรนเดอร์สัญญาณเสียง DirectSound

5.6.2 ตัวฟิลเตอร์

ฟิลเตอร์เป็นบล็อกการสร้างพื้นฐานของ DirectShow ซึ่ง DirectShow แยกการประมวลผลของข้อมูลมัลติมีเดียออกเป็นขั้นๆ โดยหนึ่งฟิลเตอร์แสดงถึงการประมวลผลหนึ่งขั้นตอน (หรืออาจจะหลายขั้นในบางครั้ง) ทำให้แอพพลิเคชันสามารถจับคู่และทำการผสมกันของฟิลเตอร์เพื่อที่จะปฏิบัติการได้หลากหลายรูปแบบกับหลายชนิดของมีเดียโดยใช้อุปกรณ์ฮาร์ดแวร์และซอฟต์แวร์ที่แตกต่างกันไป ถึงแม้ว่าแต่ละฟิลเตอร์กระทำบางอย่างที่ไม่เหมือนกันอยู่ภายในตัว แต่จากจุดมองของแอพพลิเคชัน ฟิลเตอร์แต่ละตัวนั้นก็เป็นเพียงฟิลเตอร์ของ DirectShow ที่มีคุณสมบัติมาตรฐานที่แน่นอนคือ ต้องสนับสนุนอินเทอร์เฟซ IbaseFilter และพินอินพุตและ/หรือพินเอาท์พุต ซึ่งแสดงถึงการเชื่อมต่อไปยังฟิลเตอร์อื่นๆ ฟิลเตอร์ทั้งหมดของ DirectShow จะแบ่งออกเป็น 3 จำพวก คือ ฟิลเตอร์ต้นทาง ฟิลเตอร์การแปลง (transform filters) และฟิลเตอร์เรนเดอร์ (renderer filters)

5.6.3 ฟิลเตอร์ต้นทาง (Source Filters)

ฟิลเตอร์ต้นทางแสดงถึงข้อมูลดิบของมัลติมีเดียที่เตรียมสำหรับการประมวลผล ซึ่งอาจจะนำมาจากไฟล์ในฮาร์ดดิสก์ หรือจากซีดี ดีวีดีไดร์ฟ หรืออาจจะมาจากการถ่ายทอดสด เช่น การ์ดตัวรับทีวีหรือการ์ดแคปเจอร์ที่ต่อเข้ากับกล้องดิจิตอล (ฟิลเตอร์ต้นทางเหล่านี้ได้ถูกรวมเข้าไว้ใน DirectShow และออกมาพร้อมกับระบบปฏิบัติการวินโดว์ส 9x และ 2000 ด้วย) ฟิลเตอร์ต้นทางบางตัวเพียงแค่ผ่านข้อมูลดิบเข้าไปยังฟิลเตอร์ตัวแยกหรือตัวกระจาย ขณะที่ฟิลเตอร์ต้นทางบางตัวต้องทำขั้นตอนการกระจายโดยตัวมันเองด้วย

5.6.4 ฟิลเตอร์การแปลง (Transform Filters)

ฟิลเตอร์การแปลงรับข้อมูลดิบหรือข้อมูลที่ถูกประมวลผลบางส่วนแล้วจึงทำการประมวลผลก่อนที่จะส่งต่อไป ฟิลเตอร์การแปลงมีหลายประเภทรวมถึงตัวกระจายซึ่งแยกสตรีมข้อมูลเป็นไบต์เป็นแซมเปิ้ลหรือเฟรม ตัวบีบอัดสัญญาณและตัวขยายสัญญาณ และตัวแปลงชนิดของข้อมูล จากรูปที่ 5-6 ตัวแยก AVI และตัวขยายสัญญาณ AVI เป็นฟิลเตอร์การแปลง

5.6.5 ฟิลเตอร์เรนเดอร์ (Renderer Filters)

ฟิลเตอร์เรนเดอร์โดยปกติจะรับข้อมูลที่ถูกการประมวลผลมาแล้ว และนำมาเล่นออกทางจอของระบบหรือผ่านออกไปยังลำโพงหรืออุปกรณ์ภายนอก ฟิลเตอร์ที่อยู่ในประเภทนี้ด้วยคือ ตัวเขียนไฟล์ (file-writer) ซึ่งทำการเซฟข้อมูลลงดิสก์หรือหน่วยความจำถาวรอื่นๆ ฟิลเตอร์เรนเดอร์สัญญาณวีดีโอที่มาพร้อมกับ DirectShow ใช้ DirectDraw ในการแสดงสัญญาณวีดีโอและฟิลเตอร์เรนเดอร์สัญญาณเสียงใช้ DirectSound ในการเล่นเสียง ฟิลเตอร์ที่ใช้สำหรับการเล่น การแปลง การแคปเจอร์มีเดียชนิดต่างๆ ได้ถูกรวมเข้ากับ DirectShow และออกมาพร้อมกับระบบปฏิบัติการวินโดว์ส 9x และ 2000 ซึ่งนักพัฒนาสามารถสร้างฟิลเตอร์ของตนเองในการจัดการกับรูปแบบข้อมูลมาตรฐานหรือรูปแบบที่สร้างขึ้นเองก็ได้

5.6.6 พิน (Pins)

พินเป็นตัวจัดการกับรายละเอียดในระดับต่ำของข้อมูลที่ถูกทรานส์เฟอร์ผ่านระหว่างฟิลเตอร์ พินเป็นออบเจ็กต์ COM ที่สนับสนุนอินเทอร์เฟซชื่อ IPin มีทิศทางและสัมพันธ์กับฟิลเตอร์เฉพาะในกราฟ พินถือได้ว่าเป็นจุดของการเชื่อมต่อกับฟิลเตอร์อื่น พินเอาท์พุตด้านฟิลเตอร์ของสตรีมต้นทางต่อกับพินอินพุตด้านปลายฟิลเตอร์ตัวต่อไป พินทราบชนิดของมีเดียที่สนับสนุนแล้วจึงเจรจากับชนิดของมีเดียเมื่อสองฟิลเตอร์เริ่มเชื่อมต่อกัน เมื่อตกลงถึงชนิดของมีเดียได้แล้ว พินจึงเจรจาเข้าไปถึงรายละเอียดว่าจะทำการทรานส์เฟอร์ข้อมูลอย่างไรเมื่อฟิลเตอร์กราฟเริ่มทำงาน

5.6.7 มีเดียแซมเปิ้ล (Media Samples)

หลังจากข้อมูลไบต์ดิบได้ถูกดึงไม่ว่าจากไฟล์หรือการ์ดแคปเจอร์หรือจากแหล่งอื่นเข้าไปในกราฟแล้ว ข้อมูลที่เป็นไบต์ต้องถูกกระจายออกเป็นหน่วยที่มีความสำคัญเรียกว่า มีเดียแซมเปิ้ล ในบางครั้ง

ฟิลเตอร์ต้นทางทำการกระจายข้อมูลเอง แต่บางครั้งฟิลเตอร์ที่สำหรับทำการกระจายโดยเฉพาะก็จะทำงานนี้เอง ใน DirectShow มีเดียแซมเปิ้ลถูกห่อหุ้มเข้าไว้ในออบเจ็กต์ COM ซึ่งอิมพลีเมนต์ IMediaSample2 นอกจากมีข้อมูลมัลติมีเดียจริงๆ แล้ว ออบเจ็กต์ยังประกอบไปด้วยรายละเอียดที่เกี่ยวกับชนิดของมีเดียนั้นๆ และเวลาของการซิงโครไนซ์ ออบเจ็กต์ของมีเดียแซมเปิ้ลประกอบด้วยสัญญาณวีดีโอที่มีข้อมูลของวีดีโอหนึ่งเฟรม สำหรับข้อมูลเสียงออบเจ็กต์ของมีเดียแซมเปิ้ลจะมีข้อมูลแซมเปิ้ลเสียงอยู่หลายตัว ซึ่งเมื่อข้อมูลทั้งสองแบบนี้วิ่งผ่านกราฟจากฟิลเตอร์หนึ่งไปยังอีกฟิลเตอร์หนึ่ง มันก็จะอยู่ในรูปของออบเจ็กต์ที่เป็นมีเดียแซมเปิ้ล

5.6.8 Allocators

เมื่อสองฟิลเตอร์เชื่อมต่อกัน พินของมันต้องตกลงถึงรายละเอียดว่าออบเจ็กต์ของมีเดียแซมเปิ้ลจะถูกนำจากฟิลเตอร์ของสตรีมต้นทางไปยังฟิลเตอร์สตรีมปลายทางอย่างไร ในการเชื่อมต่อกันหมายถึงจะต้องคำนึงถึงขนาด ตำแหน่ง และจำนวนแซมเปิ้ลที่จะใช้ ขนาดของแซมเปิ้ลจะขึ้นอยู่กับชนิดและรูปแบบของมีเดียและตำแหน่งของบัฟเฟอร์ที่อาจจะอยู่ในหน่วยความจำหลักหรืออุปกรณ์ฮาร์ดแวร์อื่นๆ เช่น การ์ดแคปเจอร์วีดีโอ การสร้างและการจัดการแซมเปิ้ลนี้ถูกกระทำโดย allocator ซึ่งเป็นออบเจ็กต์ COM ที่สร้างตัวมันเองขึ้นโดยพินอินพุตของทางด้านฟิลเตอร์สตรีมปลายทาง

5.6.9 สัญญาณนาฬิกา (Clocks)

ในการกระทำใดๆ ที่เกี่ยวข้องกับมัลติมีเดีย จะต้องทำการซิงโครไนซ์แซมเปิ้ลเพื่อให้เฟรมต่างๆ ของวีดีโอสามารถแสดงได้ในอัตราความเร็วที่เหมาะสม และไม่ให้สตรีมของเสียงล้ำหน้าไปก่อนสตรีมของวีดีโอ และอื่นๆ ดังนั้นฟิลเตอร์กราฟของ DirectShow จะประกอบด้วยหนึ่งสัญญาณนาฬิกาเสมอซึ่งฟิลเตอร์ทั้งหมดจะใช้เพื่อประทับเวลา (Timestamp) ประมวลผลหรือทำการเรนเดอร์มีเดียแซมเปิ้ล ตัวจัดการฟิลเตอร์กราฟเป็นตัวเลือกสัญญาณนาฬิกา (หรืออาจจะสร้างขึ้นเองหากต้องการ) แล้วชี้นำให้ฟิลเตอร์ทั้งหมดใช้สัญญาณนาฬิกานั้นเสมือนเป็นแหล่งในการซิงโครไนซ์

5.7 สถาปัตยกรรมของการสตรีมมิ่งมัลติมีเดีย

ในหัวข้อนี้จะพูดถึงสถาปัตยกรรมของการสตรีมมิ่งมัลติมีเดีย (Multimedia Streaming) และวิธีการใช้สตรีมกับทูลและแอพพลิเคชันต่างๆ ของนักพัฒนาซอฟต์แวร์ ซึ่งรวมไปถึงประโยชน์ที่ได้รับของการสตรีมมิ่งที่เป็นพื้นฐานของการส่งผ่านข้อมูล โปรแกรมเมอร์ควรจะมีพื้นฐานของการเขียนโปรแกรมทางด้าน COM (Component Object Model) มาก่อน หากต้องการที่จะทำสตรีมมิ่งมัลติมีเดีย

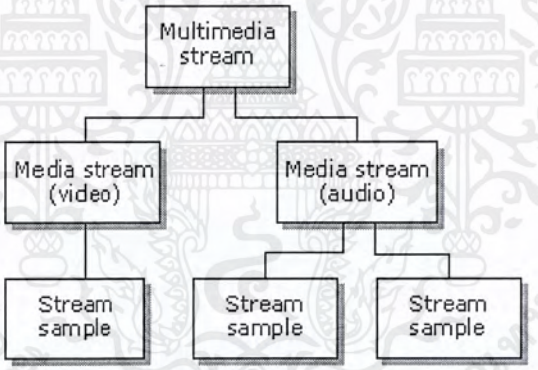
5.7.1 ข้อดีของการทำสตรีมมิ่งมัลติมีเดีย

เมื่อนักพัฒนาใช้การทำสตรีมมิ่งมัลติมีเดียในแอพพลิเคชันที่ต้องการพัฒนาจะทำให้ลดเวลาในการเขียนโปรแกรมของรูปแบบเฉพาะที่ต้องการ โดยทั่วไปแอพพลิเคชันที่ต้องรับข้อมูลของมีเดียจากไฟล์หรืออุปกรณ์ฮาร์ดแวร์จะต้องรู้ทุกอย่างเกี่ยวกับรูปแบบข้อมูลและอุปกรณ์ฮาร์ดแวร์นั้นๆ แอพพลิเคชัน

ต้องจัดการกับการเชื่อมต่อ การทรานส์เฟอร์ข้อมูล การแปลงข้อมูลใดๆ และการเรนเดอร์ข้อมูลที่แท้จริง เนื่องจากอุปกรณ์และรูปแบบข้อมูลแตกต่างกัน ขั้นตอนการประมวลผลนี้ค่อนข้างยุ่งยากและซับซ้อน ในอีกด้านหนึ่งในการทำสตรีมมิ่งมัลติมีเดียจะเจรจาการทรานส์เฟอร์และการแปลงข้อมูลจากแหล่งต้นทางให้กับแอพพลิเคชันโดยอัตโนมัติ อินเทอร์เฟซการสตรีมมิ่งมีขั้นตอนที่มีรูปแบบเดียวกันในการควบคุมและเข้าถึงข้อมูล ซึ่งทำให้ง่ายสำหรับแอพพลิเคชันที่จะเล่นข้อมูลนั้นตามชนิดของข้อมูล ขั้นตอนในการอิมพลีเมนต์การสตรีมมิ่งจากอุปกรณ์ฮาร์ดแวร์ไปจนถึงการเล่นข้อมูลนั้นมีดังนี้

1. แหล่งของข้อมูลสัญญาณวีดีโอ เช่น จาก Microsoft® DirectShow® เปิดอินเทอร์เฟซของการสตรีมมิ่ง
2. นักพัฒนาแอพพลิเคชันใช้อินเทอร์เฟซการสตรีมมิ่งมัลติมีเดียในการจัดการกับการแปลงรูปแบบของข้อมูล
3. นักพัฒนาแอพพลิเคชันใช้อินเทอร์เฟซของ Microsoft DirectDraw® ในการเรนเดอร์ข้อมูลที่ได้ออกมา

5.7.2 แผนผังลำดับชั้นของออบเจ็กต์ (Object Hierarchy)



รูปที่ 5-7 แผนผังลำดับชั้นของออบเจ็กต์ที่ใช้ในการทำสตรีมมิ่งมัลติมีเดีย

จากรูปที่ 5-7 เป็นการแสดงลำดับออบเจ็กต์ต่างๆ ของการทำสตรีมมิ่งโดยสามารถแยกออบเจ็กต์ออกเป็น 3 จำพวกที่ได้กำหนดไว้ในสถาปัตยกรรมของการสตรีมมิ่งมัลติมีเดีย ดังนี้

4. สตรีมของมัลติมีเดีย ซึ่งสนับสนุนอินเทอร์เฟซ IMultiMediaStream
5. สตรีมของมีเดียสนับสนุนอินเทอร์เฟซ IMediaStream และเป็นข้อมูลเฉพาะ โดยทุกๆ สตรีมของ

5.7.3 การสร้างสตรีมของมัลติมีเดียและแซมเปิ้ลของสตรีม

ออบเจ็กต์ที่สนับสนุนอินเทอร์เฟซ IMultiMediaStream นั้นจะเป็นที่บรรจุของข้อมูลที่เป็นสตรีมมัลติมีเดีย อินเทอร์เฟซ IMultiMediaStream รวมไปถึงเมธอดซึ่งระบุถึงข้อมูลสตรีมต่างๆ ของออบเจ็กต์ โดยธรรมดาแล้วสตรีมเหล่านี้คือข้อมูลวีดีโอและเสียง แต่สามารถเป็นข้อมูลประเภทใดก็ได้ เช่น ตัวอักษรล้วนๆ หรือ SMPTE timecode อินเทอร์เฟซ IMultiMediaStream เป็นที่บรรจุของข้อมูลทั่วๆ ไป อย่างไรก็ตาม นักพัฒนาสามารถสร้างอินเทอร์เฟซที่นับสนุนข้อมูลที่มีรูปแบบเฉพาะได้ด้วย อย่างออบเจ็กต์ที่อิมพลีเมนต์อินเทอร์เฟซ IAMMultiMediaStream สามารถระบุและควบคุมสตรีมของข้อมูลรูปแบบใดๆ ใน DirectShow เนื่องจากโดยส่วนตัวแล้วข้อมูลสตรีมจะเป็นข้อมูลประเภทที่เจาะจงไว้ ทำให้มันสามารถสนับสนุนอย่างน้อย 2 อินเทอร์เฟซที่แตกต่างกัน คือ ทั่วๆ ไปและประเภทที่เจาะจงนั้น ทุกๆ สตรีมสนับสนุนอินเทอร์เฟซ IMediaStream ซึ่งมีเมธอดที่จะรับรูปแบบของมันเองและพอยน์เตอร์ที่ชี้ไปที่สตรีมของตัวมันเอง ในขณะที่อินเทอร์เฟซ IDirectDrawMediaStream มีเมธอดที่จัดการเฉพาะกับการเรนเดอร์ข้อมูลวีดีโอ อินเทอร์เฟซใดๆ ที่ได้มาจาก IMultiMediaStream จะสนับสนุนการสร้างแซมเปิ้ลของสตรีมด้วยซึ่งเป็นหน่วยพื้นฐานของการทำสตรีมมิ่งข้อมูล

แซมเปิ้ลของมัลติมีเดียใช้เป็นตัวอ้างอิงถึงออบเจ็กต์ที่ประกอบไปด้วยข้อมูลมีเดีย สำหรับรูปภาพของวีดีโอมันจะเป็นพื้นผิวของ DirectDraw รายละเอียดส่วนประกอบของแซมเปิ้ลนั้นมีหลากหลายขึ้นกับชนิดของมีเดีย เนื่องจากแซมเปิ้ลเป็นทางที่ใช้อ้างอิงไปยังออบเจ็กต์ของข้อมูล จำนวนแซมเปิ้ลของสตรีมใดๆ ก็สามารถใช้อ้างถึงออบเจ็กต์เดียวกันได้ อินเทอร์เฟซ IStreamSample มีเมธอดที่สามารถรับและเซตค่าของแซมเปิ้ลได้ เช่น เวลาเริ่มต้นและเวลาหยุด สถานะ และส่วนที่เกี่ยวข้องกับสตรีม เมธอด IStreamSample::Update ทำการรีเฟรชข้อมูลของแซมเปิ้ลในกรณีที่เป็นสตรีมที่สามารถอ่านได้ สำหรับสตรีมที่สามารถเขียนได้นั้น มันจะเขียนข้อมูลของแซมเปิ้ลไปที่สตรีม โดยทั่วไปแล้วควรใช้เมธอด Update ในลูปของการเรนเดอร์ การทรานส์เฟอร์ หรือช่วงของการเก็บข้อมูลสตรีมมิ่ง

5.7.4 การใช้สตรีมมัลตีมีเดียในแอพพลิเคชัน

อินเทอร์เฟซของการทำสตรีมมิ่งมัลติมีเดียทำให้การเข้าถึงและกระทำกับข้อมูลมัลติมีเดียได้ง่าย โดยการนำคุณสมบัติที่ขึ้นกับชนิดของฮาร์ดแวร์และซอฟต์แวร์เฉพาะออกและทำการสนับสนุนรูปแบบของมีเดียใน Microsoft DirectX® ทั้งหมด สตรีมวางข้อมูลไว้ในระดับที่สูงมาก ทำให้แอพพลิเคชันสามารถที่จะเคลื่อนย้ายข้อมูลเหล่านั้นจากสตรีมหนึ่งไปอีกสตรีมหนึ่งโดยไม่ต้องทราบเกี่ยวกับรูปแบบของข้อมูลนั้นเลย ขั้นตอนในการสร้างสตรีมมัลติมีเดียมีดังนี้

1. สร้างสตรีมมัลติมีเดีย โดยมีเมธอดของการสร้างและเริ่มต้นของสตรีมซึ่งมีสถาปัตยกรรมที่เจาะจง DirectShow สนับสนุนอินเทอร์เฟซ IAMMultiMediaStream ซึ่งใช้ในการทำการเริ่มต้นของสตรีม ส่วนการอิมพลีเมนต์ IAMMultiMediaStream ของเซิร์ฟเวอร์อินโพรเซส (in-process) อื่นๆ จะถูกสร้างและเริ่มต้นโดยใช้กลไกที่แตกต่างกันออกไป

2. หลังจากออบเจ็กต์ของสตรีมมัลติมีเดียได้ถูกทำการเริ่มต้นแล้ว แอพพลิเคชันจะใช้ QueryInterface เพื่อรับอินเทอร์เฟซ IMultiMediaStream สำหรับออบเจ็กต์นั้นมา แล้วใช้อินเทอร์เฟซนี้ในการตรวจสอบคุณสมบัติของสตรีมและทำการระบุถึงสตรีมต่างๆ ซึ่งสามารถใช้เมธอด IMultiMediaStream::GetMediaStream ในการเรียกใช้สตรีมที่ต้องการด้วยการระบุถึง purpose ID ที่เจาะจง โดย MSPID_PrimaryVideo และ MSPID_PrimaryAudio ถือเป็น purpose ID ที่ใช้กันมากที่สุด ซึ่งก็คือวีดีโอและสตรีมของเสียงตามลำดับนั่นเอง
3. เรียกใช้ IUnknown::QueryInterface สำหรับอินเทอร์เฟซที่เจาะจงกับประเภทมีเดียของสตรีม ตัวอย่างเช่น หากต้องการเรนเดอร์สตรีมวีดีโอ ให้รับอินเทอร์เฟซ IDirectDrawMediaStream มาอินเทอร์เฟซสำหรับมีเดียที่เจาะจงได้กำหนดเมธอดเพิ่มเติมที่จำเป็นสำหรับการได้รับประโยชน์ของรูปแบบนั้นได้เต็มที่
4. ทำการสร้างแซมเปิ้ลหนึ่งหรือมากกว่านั้นสำหรับข้อมูลของสตรีม โดยทุกสตรีมมีเดียสนับสนุนเมธอด IMediaStream::CreateSharedSample สำหรับการสร้างแซมเปิ้ล แซมเปิ้ลที่ได้มาจะสนับสนุนอินเทอร์เฟซ IStreamSample ซึ่งทำการควบคุมแซมเปิ้ลและคุณสมบัติของมัน โดยทั่วไปแล้วสตรีมมีเดียสนับสนุนเมธอดที่เจาะจงรูปแบบเฉพาะของการสร้างแซมเปิ้ลซึ่งมีความสามารถมากกว่าเมธอด IStreamSample ที่ได้กล่าวมาแล้ว ตัวอย่างเช่น IDirectDrawMediaStream สามารถสร้างแซมเปิ้ลผูกติดกับพื้นผิว DirectShow และสี่เหลี่ยมผืนผ้าที่ทำการ clipping ที่ต้องการได้ อย่างไรก็ตามในบางสถานการณ์จะต้องจัดการกับข้อมูลโดยไม่ต้องรู้ถึงรูปแบบของข้อมูล ถ้าต้องการข้อมูลสตรีมที่ไม่ขึ้นกับรูปแบบของข้อมูล ให้ใช้เมธอด IMediaStream::CreateSharedSample ในการสร้างแซมเปิ้ลของข้อมูล
5. หลังจากการสร้างแซมเปิ้ลของสตรีมที่ต้องการได้ทั้งหมดแล้ว ให้เริ่มต้นสตรีมโดยเรียกเมธอด IMultiMediaStream::SetState และส่งค่าแฟล็ก STREAMSTATE_RUN เข้าไปเป็นพารามิเตอร์
6. ให้เรียก IStreamSample::Update ทำการอัพเดตแซมเปิ้ลของสตรีม เมื่อออกจากเมธอด IStreamSample::Update แล้ว ทำให้สามารถเข้าถึงข้อมูลของแซมเปิ้ลได้

5.8 อินเทอร์เฟซพื้นฐานของการสตรีมมิ่งมัลติมีเดีย

อินเทอร์เฟซพื้นฐานของการสตรีมมิ่งมัลติมีเดียทำให้สามารถเข้าถึงสตรีมของมัลติมีเดียได้ อย่างไรก็ตามการใช้อินเทอร์เฟซพื้นฐานเข้าถึงชนิดของข้อมูลที่เจาะจงนี้ทำให้จำกัดจำนวนของการควบคุมข้อมูล ดังนั้นนักพัฒนามีเดียควรสร้างเวอร์ชันใหม่ของอินเทอร์เฟซเหล่านี้ซึ่งทำให้สามารถควบคุมความสามารถของชนิดของมีเดียเหล่านั้นได้มากกว่า

IMultiMediaStream

นิยามถึงวิธีการเข้าถึงออบเจ็กต์ของสตรีมมัลติมีเดียในระดับสูงสุด ออบเจ็กต์นี้ทำให้เข้าถึงออบเจ็กต์ของสตรีมอื่นๆ ได้ IMultiMediaStream มีเมธอดที่ระบุหรือรับสตรีมที่ต้องการ รวมทั้งการตรวจสอบระยะเวลาทั้งหมดของสตรีมและการค้นหาภายในสตรีมด้วย

IMediaStream

นิยามถึงออบเจ็กต์ของสตรีมทั่วไป ใช้เมธอดต่างๆ ของตัวนี้ในการรับค่าพอยน์เตอร์ที่ชี้ไปยังสตรีมเพื่อรับรายละเอียดต่างๆ ที่เกี่ยวกับสตรีม และสร้างแซมเปิ้ลจากข้อมูลของสตรีม รวมทั้งสามารถสร้างแซมเปิ้ลของสตรีมร่วมที่หลายๆ สตรีมสามารถเข้าถึงข้อมูลโดยไม่มีการซ้ซ้อนของข้อมูลแซมเปิ้ล

IStreamSample

ควบคุมพฤติกรรมของแซมเปิ้ลสตรีมที่เจาะจง ทำให้สามารถรับสตรีมที่สร้างแซมเปิ้ล ตรวจสอบเวลาเริ่มต้น เวลาหยุดและสถานะที่เสร็จสิ้นของแซมเปิ้ล และสามารถกำหนดฟังก์ชันเองกับแซมเปิ้ลได้ (โดยผ่านทางเมธอด Update) โดยทั่วไปแล้วเมธอด Update ประมวลผลข้อมูลแซมเปิ้ลในลำดับที่เหมาะสม เช่น การเรนเดอร์ข้อมูลของวีดีโอ หรือการเล่นข้อมูลของเสียง

5.9 อินเทอร์เฟซของการสตรีมมิ่งใน DirectDraw

ถ้าใช้รูปแบบวีดีโอที่สนับสนุน Microsoft® DirectDraw® กับสตรีมแล้ว อินเทอร์เฟซต่อไปนี้ทำให้สามารถควบคุมข้อมูลได้มากกว่าอินเทอร์เฟซพื้นฐานทั่วไป

IDirectDrawMediaStream

กำหนดและรับรูปแบบของสตรีมและออบเจ็กต์ DirectDraw ที่สัมพันธ์กับสตรีมของมีเดีย และสามารถใช้ในการสร้างแซมเปิ้ลของวีดีโอได้ด้วย อินเทอร์เฟซนี้ทำการสืบทอดมากจาก IMediaStream

IDirectDrawStreamSample

ทำให้สามารถผูกติดแซมเปิ้ลของวีดีโอเข้ากับพื้นผิวของ DirectDraw ได้ อินเทอร์เฟซนี้สืบทอดมาจากอินเทอร์เฟซ IStreamSample แต่ละพื้นผิวที่ผูกติดจะรวมไปถึงสี่เหลี่ยมผืนผ้าที่ทำการคลิปปิ้งแล้ว ทำให้การทำเรนเดอร์ได้ง่ายขึ้น

5.10 อินเทอร์เฟซของการสตรีมมิ่งเสียง

IAudioMediaStream

ทำการควบคุมสตรีมของมีเดียที่เป็นเสียง อินเทอร์เฟซนี้สืบทอดมาจากอินเทอร์เฟซ IMediaStream และใช้ในการสร้างออบเจ็กต์ IAudioStreamSample รวมไปถึงสามารถใช้ในการกำหนดและรับค่ารูปแบบของข้อมูลของสตรีมในขณะนั้น

IAudioStreamSample

รับค่ารายละเอียดต่างๆ จากข้อมูลออบเจ็กต์ IAudioData

IMemoryData

ประกอบไปด้วยเมธอดที่กำหนดและรับข้อมูลในหน่วยความจำของออบเจ็กต์ที่เป็นข้อมูลเสียง ออบเจ็กต์ที่เป็นข้อมูลเสียงมีข้อมูลที่แซมเปิ้ลของสตรีมทำการเข้าถึง อินเทอร์เฟซนี้สามารถทำ

การเริ่มต้นบัฟเฟอร์ในหน่วยความจำและกำหนดจำนวนข้อมูลเสียงที่แท้จริงในออบเจ็กต์ นอกจากนี้เมธอด IMemoryData::GetInfo สามารถใช้ในการรับข้อมูลเสียงในหน่วยความจำได้

IAudioData

มีเมธอดต่างๆ สำหรับให้แอพพลิเคชันกำหนดและรับค่าของข้อมูลเสียงที่สตรีมของเสียงจะอ้างอิงไปถึง รูปแบบของข้อมูลเสียงจะกำหนดอยู่ในโครงสร้าง WAVEFORMATX

5.11 อินเทอร์เฟซของการสตรีมมิ่งมัลติมีเดียใน DirectShow

อินเทอร์เฟซต่อไปนี้ทำให้มีฟังก์ชันของ Microsoft® DirectShow® ให้กับนักพัฒนาสตรีมมัลติมีเดียได้ใช้ นักพัฒนาแอพพลิเคชันควรอิมพลีเมนต์เฉพาะอินเทอร์เฟซ IAMMultiMediaStream ในแอพพลิเคชันของตนเท่านั้น ส่วน DirectShow จะใช้อินเทอร์เฟซที่เหลือเอง

IAMMultiMediaStream

ควบคุมการเชื่อมต่อระหว่างสตรีมและฟิลเตอร์กราฟของ DirectShow ซึ่งมีเมธอดที่สร้างออบเจ็กต์ฟิลเตอร์กราฟ เปิดไฟล์มีเดียที่สนับสนุน DirectShow และสร้างฟิลเตอร์กราฟที่เหมาะสมสำหรับการเล่นไฟล์ ตัวชี้ของคลาส CLSID_ActiveMovieMMStream สนับสนุนอินเทอร์เฟซนี้

IAMMediaStream

จัดการการเจรจาภายในของการติดต่อระหว่างสตรีมของมัลติมีเดียและฟิลเตอร์กราฟของ DirectShow อินเทอร์เฟซนี้เป็นตัวที่สืบทอดโดยเฉพาะของ DirectShow มาจากอินเทอร์เฟซ IMediaStream นักพัฒนาแอพพลิเคชันไม่ควรใช้หรือทำการอิมพลีเมนต์อินเทอร์เฟซนี้

5.12 ออบเจ็กต์ของคอมโพเน็นต์การสตรีมมิ่งมัลติมีเดีย

ในหัวข้อนี้จะมีตารางซึ่งอธิบายถึงคอมโพเน็นต์ต่างๆ ที่ Microsoft® DirectShow® สนับสนุน ออบเจ็กต์เหล่านี้สนับสนุนอินเทอร์เฟซการสตรีมมิ่งมัลติมีเดีย หัวข้อนี้ยังรวมไปถึงไดอะแกรมที่แสดงลำดับชั้นของออบเจ็กต์คอมโพเน็นต์

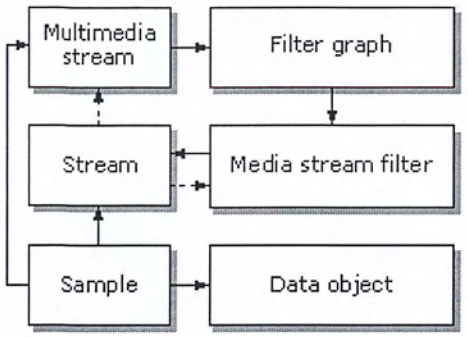
5.12.1 ตารางออบเจ็กต์ต่างๆ ของคอมโพเน็นต์

ชื่อออบเจ็กต์	รายละเอียด	อินเทอร์เฟซที่สนับสนุน
CLSID_AMMultiMediaStream	การอิมพลีเมนต์ DirectShow ของสตรีมมัลติมีเดีย	IAMMultiMediaStream, ImultiMediaStream
CLSID_MediaStreamFilter	จัดหาฟังก์ชันการสตรีมมิ่งมัลติมีเดีย สำหรับออบเจ็กต์ CLSID_AMMultiMediaStream ผ่านทางอินเทอร์เฟซ IAMMultiMediaStream	IbaseFilter
CLSID_AMDirectDrawStream	สตรีมมีเดียของ Microsoft® DirectDraw® ซึ่งสามารถถูกเพิ่มเข้าไปในสตรีมมัลติมีเดียของ DirectShow	IAMMediaStream, ImediaStream, IdirectDrawMediaStream, IPin, ImemInputPin
Samples (สร้างโดยสตรีมของ DirectShow)	สร้างโดยสตรีมของ DirectShow	IstreamSample, IdirectDrawStreamSample, ImediaSample
CLSID_ AMMediaTypeStream	สามารถสร้างแซมเปิ้ลของมีเดียสำหรับชนิดของข้อมูลใดๆ ที่ DirectShow สนับสนุน	IAMMediaStream, ImediaStream, IPin, ImemInputPin
Samples (สร้างโดยออบเจ็กต์ CLSID_ AMMediaTypeStream)	สร้างโดยออบเจ็กต์ CLSID_ AMMediaTypeStream	IstreamSample, ImediaSample, IMediaSample2
CLSID_AMAudioData	การอิมพลีเมนต์ของตัวบรรจุออบเจ็กต์เสียง IAudioData	IAudioData

ตารางที่ 5-1 ออบเจ็กต์ต่างๆ ของคอมโพเน็นต์การสตรีมมิ่งมัลติมีเดีย

5.12.2 ไดอะแกรมการอ้างอิงของออบเจ็กต์

ลำดับชั้นของออบเจ็กต์ทำให้สร้างบางสิ่งบางอย่างที่น่าสนใจในการอ้างอิงระหว่างออบเจ็กต์ DirectShow ด้วยกัน ในรูปที่ 5-8 เป็นไดอะแกรมแสดงออบเจ็กต์และการอ้างอิงทั้งหมด การอ้างอิงของออบเจ็กต์ที่เพิ่มเติมจากออบเจ็กต์ที่ถูกอ้างอิงจะแสดงด้วยเส้นลูกศรทึบ ส่วนการอ้างอิงที่ไม่ต้องใช้ AddRef กับออบเจ็กต์ที่ถูกอ้างอิงจะแสดงด้วยเส้นลูกศรประ



รูปที่ 5-8 ไดอะแกรมของการอ้างอิงของออบเจ็กต์

5.13 ตัวอย่างการสตรีมมิ่งเสียง

ในหัวข้อนี้เป็นตัวอย่างโค้ดภาษา C++ ที่ทำการอิมพลีเมนต์อินเทอร์เฟซต่างๆ ของการสตรีมมิ่งเสียง โดยใช้อินเทอร์เฟซ IAudioMediaStream, IMemoryData และ IAudioData

```
#include <windows.h>
#include <mmsystem.h>
#include <amstream.h>

/*****************************************************************
   Trivial wave player stuff
*****************************************************************/

class CWaveBuffer;

class CWaveBuffer {
    public:
        CWaveBuffer();
        ~CWaveBuffer();
        BOOL Init(HWAVEOUT hWave, int Size);
        void Done();
        BOOL Write(PBYTE pData, int nBytes, int& BytesWritten);
        void Flush();

    private:
        WAVEHDR      m_Hdr;
        HWAVEOUT     m_hWave;
        int          m_nBytes;
};

class CWaveOut {
    public:
        CWaveOut(LPCWAVEFORMATEX Format, int nBuffers, int
BufferSize);
        ~CWaveOut();
        void Write(PBYTE Data, int nBytes);
        void Flush();
        void Wait();
        void Reset();
    private:
        const HANDLE   m_hSem;
```

```
        const int       m_nBuffers;
        int             m_CurrentBuffer;
        BOOL            m_NoBuffer;
        CWaveBuffer    *m_Hdrs;
        HWAVEOUT        m_hWave;
};

/*
    CWaveBuffer
*/

CWaveBuffer::CWaveBuffer()
{
}

BOOL CWaveBuffer::Init(HWAVEOUT hWave, int Size)
{
    m_hWave  = hWave;
    m_nBytes = 0;

    /*  Allocate a buffer and initialize the header. */
    m_Hdr.lpData = (LPSTR)LocalAlloc(LMEM_FIXED, Size);
    if (m_Hdr.lpData == NULL) {
        return FALSE;
    }
    m_Hdr.dwBufferLength  = Size;
    m_Hdr.dwBytesRecorded = 0;
    m_Hdr.dwUser = 0;
    m_Hdr.dwFlags = 0;
    m_Hdr.dwLoops = 0;
    m_Hdr.lpNext = 0;
    m_Hdr.reserved = 0;

    /*  Prepare it. */
    waveOutPrepareHeader(hWave, &m_Hdr, sizeof(WAVEHDR));

    return TRUE;
}

CWaveBuffer::~CWaveBuffer() {
    if (m_Hdr.lpData) {
        waveOutUnprepareHeader(m_hWave, &m_Hdr, sizeof(WAVEHDR));
        LocalFree(m_Hdr.lpData);
    }
}

void CWaveBuffer::Flush()
{
    // ASSERT(m_nBytes != 0);
    m_nBytes = 0;
    waveOutWrite(m_hWave, &m_Hdr, sizeof(WAVEHDR));
}

BOOL CWaveBuffer::Write(PBYTE pData, int nBytes, int& BytesWritten)
{
    // ASSERT((DWORD)m_nBytes != m_Hdr.dwBufferLength);
    BytesWritten = min((int)m_Hdr.dwBufferLength - m_nBytes, nBytes);
    CopyMemory((PVOID)(m_Hdr.lpData + m_nBytes), (PVOID)pData,
BytesWritten);
    m_nBytes += BytesWritten;
    if (m_nBytes == (int)m_Hdr.dwBufferLength) {
```

```
        /*  Write it! */
        m_nBytes = 0;
        waveOutWrite(m_hWave, &m_Hdr, sizeof(WAVEHDR));
        return TRUE;
    }
    return FALSE;
}

void CALLBACK WaveCallback(HWAVEOUT hWave, UINT uMsg, DWORD dwUser,
                           DWORD dw1, DWORD dw2)
{
    if (uMsg == WOM_DONE) {
        ReleaseSemaphore((HANDLE)dwUser, 1, NULL);
    }
}

/*
    CWaveOut
*/

CWaveOut::CWaveOut(LPCWAVEFORMATEX Format, int nBuffers, int
BufferSize) :
    m_nBuffers(nBuffers),
    m_CurrentBuffer(0),
    m_NoBuffer(TRUE),
    m_hSem(CreateSemaphore(NULL, nBuffers, nBuffers, NULL)),
    m_Hdrs(new CWaveBuffer[nBuffers]),
    m_hWave(NULL)
{
    /*  Create wave device. */
    waveOutOpen(&m_hWave,
                WAVE_MAPPER,
                Format,
                (DWORD)WaveCallback,
                (DWORD)m_hSem,
                CALLBACK_FUNCTION);

    /*  Initialize the wave buffers. */
    for (int i = 0; i < nBuffers; i++) {
        m_Hdrs[i].Init(m_hWave, BufferSize);
    }
}

CWaveOut::~CWaveOut()
{
    /*  First, get the buffers back. */
    waveOutReset(m_hWave);

    /*  Free the buffers. */
    delete [] m_Hdrs;

    /*  Close the wave device. */
    waveOutClose(m_hWave);

    /*  Free the semaphore. */
    CloseHandle(m_hSem);
}

void CWaveOut::Flush()
{
    if (!m_NoBuffer) {
```

```
            m_Hdrs[m_CurrentBuffer].Flush();
            m_NoBuffer = TRUE;
            m_CurrentBuffer = (m_CurrentBuffer + 1) % m_nBuffers;
        }
    }

    void CWaveOut::Reset()
    {
        waveOutReset(m_hWave);
    }


    void CWaveOut::Write(PBYTE pData, int nBytes)
    {
        while (nBytes != 0) {
            /*  Get a buffer if necessary. */
            if (m_NoBuffer) {
                WaitForSingleObject(m_hSem, INFINITE);
                m_NoBuffer = FALSE;
            }

            /*  Write into a buffer. */
            int nWritten;
            if (m_Hdrs[m_CurrentBuffer].Write(pData, nBytes, nWritten)) {
                m_NoBuffer = TRUE;
                m_CurrentBuffer = (m_CurrentBuffer + 1) % m_nBuffers;
                nBytes -= nWritten;
                pData += nWritten;
            } else {
                // ASSERT(nWritten == nBytes);
                break;
            }
        }
    }

    void CWaveOut::Wait()
    {
        /*  Send any remaining buffers. */
        Flush();

        /*  Wait for the buffers back. */
        for (int i = 0; i < m_nBuffers; i++) {
            WaitForSingleObject(m_hSem, INFINITE);
        }
        LONG lPrevCount;
        ReleaseSemaphore(m_hSem, m_nBuffers, &lPrevCount);
    }

    /*************************************************************
      End of wave player stuff
     *************************************************************/


    HRESULT RenderStreamToDevice(IMultiMediaStream *pMMStream)
    {
        WAVEFORMATEX wfx;
        #define DATA_SIZE 5000
        PBYTE pBuffer = (PBYTE)LocalAlloc(LMEM_FIXED, DATA_SIZE);

        IMediaStream *pStream;
        IAudioStreamSample *pSample;
```

```
    IAudioMediaStream *pAudioStream;
    IAudioData *pAudioData;

    pMMStream->GetMediaStream(MSPID_PrimaryAudio, &pStream);
    pStream->QueryInterface(IID_IAudioMediaStream, (void
**)&pAudioStream);
    pAudioStream->GetFormat(&wfx);
    CoCreateInstance(CLSID_AMAudioData, NULL, CLSCTX_INPROC_SERVER,
                                    IID_IAudioData, (void
**)&pAudioData);
    pAudioData->SetBuffer(DATA_SIZE, pBuffer, 0);
    pAudioData->SetFormat(&wfx);
    pAudioStream->CreateSample(pAudioData, 0, &pSample);
    HANDLE hEvent = CreateEvent(FALSE, NULL, NULL, FALSE);
    CWaveOut WaveOut(&wfx, 4, 2048);
    int iTimes;
    for (iTimes = 0; iTimes < 3; iTimes++) {
        DWORD dwStart = timeGetTime();
        for (; ; ) {
            HRESULT hr = pSample->Update(0, hEvent, NULL, 0);
            if (FAILED(hr) || MS_S_ENDOFSTREAM == hr) {
                break;
            }
            WaitForSingleObject(hEvent, INFINITE);
            DWORD dwTimeDiff = timeGetTime() - dwStart;
            // Limit to 10 seconds
            if (dwTimeDiff > 10000) {
                break;
            }
            DWORD dwLength;
            pAudioData->GetInfo(NULL, NULL, &dwLength);
            WaveOut.Write(pBuffer, dwLength);
        }
        pMMStream->Seek(0);
    }

    pAudioData->Release();
    pSample->Release();
    pStream->Release();
    pAudioStream->Release();
    LocalFree((HLOCAL)pBuffer);

    return S_OK;
}

HRESULT RenderFileToMMStream(
    WCHAR * pszFileName, IMultiMediaStream **ppMMStream)
{
    IAMMultiMediaStream *pAMStream;
    CoCreateInstance(CLSID_AMMultiMediaStream, NULL,
CLSCTX_INPROC_SERVER,
            IID_IAMMultiMediaStream, (void **)&pAMStream);
    pAMStream->Initialize(STREAMTYPE_READ, AMMSF_NOGRAPHTHREAD,
NULL);
    pAMStream->AddMediaStream(NULL, &MSPID_PrimaryAudio, 0, NULL);
    pAMStream->OpenFile(pszFileName, AMMSF_RUN);
    *ppMMStream = pAMStream;
    return S_OK;
}

int _CRTAPI1 main(int argc, char *argv[])
```

```
{
    IMultiMediaStream *pMMStream;
    CoInitialize(NULL);
    WCHAR wszName[1000];
    MultiByteToWideChar(CP_ACP, 0, argv[1], -1, wszName,
        sizeof(wszName) / sizeof(wszName[0]));
    RenderFileToMMStream(wszName, &pMMStream);
    RenderStreamToDevice(pMMStream);
    pMMStream->Release();
    CoUninitialize();
    return 0;
}
```

บทที่ 6

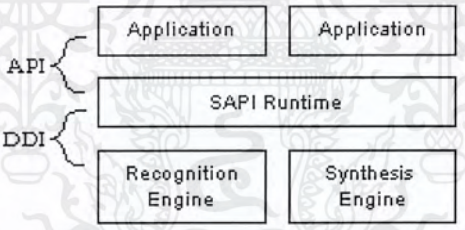
SAPI 5.0 (Speech Application Programming Interface)

6.1 บทนำ SAPI

SAPI เป็น API ที่เกี่ยวข้องกับเสียงพูด ซึ่งเกี่ยวข้องใน 2 ด้าน คือ การสังเคราะห์เสียงจากประโยคหรือจากข้อความธรรมดา (Speech Synthesis) และการจดจำเสียง (Speech Recognition) SAPI เวอร์ชัน 5.0 ประกอบไปด้วย 2 อินเทอร์เฟซ คือ API (Application Programming Interface) และ DDI (Device Driver Interface) การใช้ API ของ SAPI 5.0 ทำให้ลดความยุ่งยากในการสังเคราะห์และจดจำเสียง และมีความน่าใช้ในการพัฒนามากขึ้น

6.2 ภาพรวมของ API และ DDI

DDI และ API ของ SAPI 5.0 ลดรายละเอียดของการอิมพลีเมนต์อย่างเช่น การทำมัลติเธรดและการจัดการกับอุปกรณ์ที่เกี่ยวกับเสียง ทำให้เอนจิ้น (engine) และแอพพลิเคชันของการสังเคราะห์และการจดจำเสียงมีความสะดวกและน่าใช้มากขึ้น อินเทอร์เฟซของ SAPI 5.0 เป็นตัวใหม่หมดซึ่งไม่ได้ใช้อินเทอร์เฟซของ SAPI 4.0 เลย ซึ่งการออกแบบโมเดลของ SAPI 5.0 ใหม่นี้สามารถดูได้จากในรูปที่ 6-1



รูปที่ 6-1 โมเดลของ SAPI 5.0

6.3 อีเวนต์ของ API

ทั้งการสังเคราะห์และการจดจำเสียงนั้นแอพพลิเคชันจะได้รับอีเวนต์มาด้วย ตัวอย่างเช่น เมื่อประโยคของข้อความได้ถูกพูดออกมาหรือน้ำเสียงของคำพูดได้ถูกจดจำ คอมโพเน็นต์ SAPI ที่สร้างอีเวนต์จะทำการอิมพลีเมนต์อินเทอร์เฟซ ISpNotifySource โดยเฉพาะอย่างยิ่งตัว SAPI จะใช้ SetNotifySink โดยแอพพลิเคชันจะทำการส่งพอยน์เตอร์ของอินเทอร์เฟซ ISpNotifySink ไปยัง IspNotifySource::SetNotifySink ซึ่งจะได้รับการแจ้งการเรียกไปยัง IspNotifySink::Notify เมื่อมีหนึ่งอีเวนต์หรือหลายอีเวนต์ที่แอพพลิเคชันสามารถรับข้อมูลได้ โดยปกติแล้วตัวแอพพลิเคชันจะไม่ทำการอิมพลีเมนต์ ISpNotifySink เพราะมันจะต้องเป็นเธรดที่เป็นอิสระ แต่แอพพลิเคชันสามารถใช้ CoCreateInstance (ใช้กับ COM) ในการสร้างออบเจ็กต์พร็อกซี่ ISpNotifySink ซึ่งถูกอิมพลีเมนต์โดยคอม

โพเน็นต์ CLSID_SpNotify แทน ออบเจ็กต์พร็อกซี่นี้จะมีอินเทอร์เฟซ ISpNotifyControl ทำให้มันสามารถสร้างกลไกการแจ้งเหตุการณ์ (notification) ต่างๆ ได้หนึ่งในสามแบบตามตารางที่ 6-1

Method	Function
IspNotifyTranslator ::InitWindowMessage	ทำให้มีฟังก์ชันของ window callback ในการรับแจ้งเหตุการณ์ต่างๆ เหมือนเป็นเมสเซจของวินโดว์สเอง
IspNotifyTranslator ::InitCallback	ทำให้มีฟังก์ชัน callback ในการรับแจ้งเหตุการณ์ต่างๆ
IspNotifyTranslator ::InitWin32Event	ทำให้มีการแจ้งเหตุการณ์ต่างๆ ผ่านอีเวนต์ ของ Win32

ตารางที่ 6-1 เมธอดในการแจ้งเหตุการณ์ของอินเทอร์เฟซ ISpNotifyControl

อินเทอร์เฟซ ISpNotifySource และ ISpNotifySink มีเพียงแค่กลไกสำหรับการแจ้งเหตุการณ์ต่างๆ เท่านั้น แต่ไม่มีรายละเอียดของอีเวนต์ที่เกิดขึ้น ซึ่งการใช้ออบเจ็กต์ ISpEventSource จะทำให้แอพพลิเคชันสามารถรับรายละเอียดเกี่ยวกับอีเวนต์นั้นได้ ISpEventSource ยังมีกลไกในการทำฟิลเตอร์และการจัดคิวของอีเวนต์ได้อีกด้วย โดยปกติแล้วแอพพลิเคชัน (ที่จริงคือ ISpNotifySink) จะไม่ได้รับการแจ้งเหตุการณ์ใดๆ จาก ISpEventSource จนกว่า ISpEventSource::SetInterest จะถูกเรียกใช้เพื่อจะทำการเจาะจงว่าอีเวนต์ใดบ้างที่จะต้องการให้ถูกแจ้ง และสำหรับอีเวนต์ใดที่เกิดการซ้ำจะให้ทำการคิวไว้บ้าง อีเวนต์ถูกชี้โดยใช้แฟล็ก SPEVENTENUM

เมื่อแอพพลิเคชันได้รับแจ้งเหตุการณ์ อาจจะมีรายละเอียดของหลายอีเวนต์อยู่ ซึ่งในการเรียกใช้ ISpEventSource::GetInfo สมาชิก ulCount ในโครงสร้าง SPEVENTSOURCEINFO ที่คืนค่ามานั้นจะสามารถชี้ได้ว่าอีเวนต์ใดบ้างที่มีรายละเอียดอยู่กับอีเวนต์นั้น การใช้ ISpEventSource::GetEvents ทำให้แอพพลิเคชันสามารถขยายจำนวนของโครงสร้าง SPEVENT ที่ต้องการได้ซึ่งแต่ละตัวจะมีรายละเอียดที่เกี่ยวกับอีเวนต์นั้นด้วย

6.4 API สำหรับการสังเคราะห์เสียง

ตัวแอพพลิเคชันต้องใช้อินเทอร์เฟซ ISpVoice สำหรับพื้นฐานของการสังเคราะห์เสียง แอพพลิเคชันจะเรียก CoCreateInstance ของ COM สำหรับคอมโพเน็นต์ CLSID_SpVoice เพื่อรับค่าพอยน์เตอร์ที่ชี้ไปยังอินเทอร์เฟซ ISpVoice ของออบเจ็กต์เสียง แล้วแอพพลิเคชันจึงสามารถเริ่มการเรียกใช้เมธอดของ ISpVoice เช่น ISpVoice::Speak ซึ่งจะทำการสังเคราะห์เสียงสำหรับประโยคที่ส่งเข้าไปเป็นพารามิเตอร์ โดยปกติแล้วออบเจ็กต์เสียงได้ถูกเซตเป็นเสียงตามค่าดีฟอลต์ นอกเสียจากว่าจะเรียกใช้ ISpVoice::SetVoice น้ำเสียงจึงจะเปลี่ยนไป และสามารถเรียก ISpVoice::SetOutput โดยใช้ IUnknown และรูปแบบของเสียงใน SPSTREAMFORMAT ที่เหมาะสมเพื่อกำหนดลักษณะของข้อมูลเสียงปลายทางได้

นอกจากการสร้างเสียงจากเท็กซ์ไฟล์ธรรมดาแล้ว แอพพลิเคชันยังสามารถใส่ภาษาที่เป็น XML ไว้ในข้อความที่ส่งเข้าไปใน ISpVoice::Speak ได้ด้วย แอพพลิเคชันสามารถเล่นข้อมูลเสียงได้โดยใช้ ISpVoice::SpeakStream เท็กซ์ไฟล์ธรรมดาก็สามารถทำให้อ่านออกเสียงโดยใช้ ISpVoice::SpeakStream ได้เช่นกัน รูปแบบของข้อความ (Unicode หรือ ANSI) จะถูกตรวจสอบโดยอัตโนมัติ ฟังก์ชันทั้งสองตัวนี้มีแฟล็กที่ควบคุมว่าควรจะเคลียร์เสียงที่ยังไม่พร้อมจะเล่น รวมทั้งมีแฟล็กว่าต้องการเรียกใช้เป็นแบบอะซิงโครนัสหรือไม่

เนื่องจากออบเจ็กต์ ISpVoice เป็นออบเจ็กต์ ISpEventSource ด้วย แอพพลิเคชันจึงสามารถเริ่มรับรายละเอียดของอีเวนต์โดยการเรียกใช้ ISpNotifySource::SetNotifySink ผ่านเข้าไปในออบเจ็กต์ ISpNotifySink ได้ การเรียกใช้ที่เกี่ยวข้องกับ ISpVoice::Speak หรือ ISpVoice::SpeakStream ตัว ISpVoice จะมีอีเวนต์ SPEI_START_INPUT_STREAM ก่อนที่เอนจิ้นจะเริ่มทำการสังเคราะห์เสียงและมีอีเวนต์ SPEI_END_INPUT_STREAM เมื่อเอนจิ้นเสร็จสิ้นจากการสังเคราะห์เสียงแล้ว การเรียกใช้ ISpVoice::Speak หรือ ISpVoice::SpeakStream ในแต่ละครั้งจะทำให้จำนวนสตรีมเพิ่มขึ้น (ใน pulStreamNum) โดยแต่ละโครงสร้างของ SPEVENT ที่บอกรายละเอียดถึงอีเวนต์จะมีสมาชิก ulStreamNum ที่ชี้ถึงจำนวนสตรีมที่เกี่ยวข้องกับการเรียกใช้ ISpVoice::Speak หรือ ISpVoice::SpeakStream ซึ่งตัว ISpVoice จะสร้างอีเวนต์ต่างๆ ของชนิด SPEVENTENUM โดยดูได้จากตารางที่ 6-2 และแอพพลิเคชันสามารถเรียกใช้ ISpVoice::GetStatus ในการรับสถานะปัจจุบันของการสังเคราะห์เสียงได้

Event	Function
SPEI_VOICE_CHANGE	ชนิดหรือ ID ของเสียงถูกเปลี่ยนขณะที่มีการสังเคราะห์เสียงอยู่
SPEI_TTS_BOOKMARK	bookmark ถูกเจอขณะกำลังทำการสังเคราะห์เสียง
SPEI_WORD_BOUNDARY	เริ่มพูดของต้นคำ
SPEI_SENTENCE_BOUNDARY	เริ่มพูดของต้นประโยค
SPEI_PHONEME	พูดส่วนที่น้อยที่สุดของเสียง (phoneme)
SPEI_VISEME	พูด Viseme

ตารางที่ 6-2 อีเวนต์ต่างๆ ของ ISpVoice

6.5 DDI สำหรับการสังเคราะห์เสียง

เอนจิ้นการสังเคราะห์เสียงของ SAPI ทำการอิมพลีเมนต์อินเทอร์เฟซ ISpTTSEngine เมธอดในขั้นแรกที่ถูกเรียกใช้โดย SAPI เพื่อทำเรนเดอร์เสียงก็คือ ISpTTSEngine::Speak SAPI จะกระจายคำในไวยากรณ์โดยใช้ XML กับสตรีมของข้อความอินพุต เมธอด Speak เป็นตัวเชื่อมข้อความที่กระจายนั้นเป็น link list ที่มีค่าของสถานะ XML ที่เกี่ยวข้อง เมธอด Speak ยังทำการรับพอยน์เตอร์ที่ชี้ไปยังอินเทอร์เฟซ

ISpTTSEngineSite ของออบเจ็กต์ SpVoice ด้วย เอนจิ้นของ TTS (Text-To-Speech) ใช้อินเทอร์เฟซนี้ในการคิวอีเวนต์และเขียนข้อมูลของเอาท์พุต

แม้ว่า SAPI จะเป็นสถาปัตยกรรมที่เป็นเธรดอิสระ แต่อินสแทนซ์ของเอนจิ้น TTS จะยังคงถูกเรียกโดย SAPI โดยเป็นเธรดเดียวเสมอ เอนจิ้น TTS จะไม่ถูกเรียกโดยตรงจากแอพพลิเคชัน SAPI จะตรวจให้แน่ใจว่าพารามิเตอร์ทุกตัวมีค่าที่ใช้ได้และการซิงโครนัสของเธรดได้ทำงานเป็นปกติก่อนที่จะเรียกเอนจิ้น TTS

บทที่ 7
Windows Sockets และ FTP

7.1 Sockets

ซ็อกเก็ต (Socket) หมายถึงเซตหนึ่งของฟังก์ชันที่ใช้ทำการติดต่อแบบสองทางโดยใช้โพรโตคอล TCP/IP หรือโพรโตคอลอื่นๆ ซ็อกเก็ตมีอยู่ 2 ชนิดคือ ซ็อกเก็ตสตรีม (stream socket) และซ็อกเก็ตดาต้าแกรม โดยทั้งสองชนิดใช้กับโพรโตคอลหลักสองประเภทของ TCP/IP คือ TCP (Transmission Control Protocol) และ UDP (User Datagram Protocol) ซึ่งโพรโตคอล TCP มีกระบวนการคล้ายกับการที่เครื่องสองเครื่องสร้างการติดต่อหนึ่งช่องการติดต่อ จากนั้นข้อมูลจะถูกแลกเปลี่ยนผ่านช่องการติดต่อนี้ เมื่อเสร็จก็ทำการปิดช่องการติดต่อนั้น ในทางตรงกันข้าม โพรโตคอล UDP จะไม่เกี่ยวข้องกับการสร้างการติดต่อเครื่องหนึ่งส่งข้อมูลให้เครื่องอื่นโดยไม่สร้างช่องทางการติดต่อ ซ็อกเก็ตสตรีมใช้สร้างโพรโตคอล TCP และถูกใช้กับงานที่จะต้องแลกเปลี่ยนข้อมูลจำนวนมากๆ หรืองานที่เน้นลำดับความสำคัญของข้อมูลที่เข้ามา ตัวอย่างของการใช้งาน TCP ที่เก่าแก่ซึ่งใช้ซ็อกเก็ตสตรีมนั่นคือ FTP (File Transfer Protocol) แต่ซ็อกเก็ตดาต้าแกรมใช้สร้างโพรโตคอล UDP และถูกใช้กับงานที่มีการแลกเปลี่ยนข้อมูลน้อยๆ หรืองานที่ไม่สนใจลำดับความสำคัญของข้อมูล ตัวอย่างของการใช้งาน UDP ที่เก่าแก่เห็นจะเป็นตัวอัพเดตสัญญาณนาฬิกา (clock-updater) ซึ่งจะทำการกระจายข้อมูลของเวลาของระบบไปให้เครื่องอื่นๆ

7.2 Windows Sockets 2

ส่วนวินโดว์สซ็อกเก็ต (Windows Sockets) หรือที่เรียกว่า Winsock จะหมายความถึง API มาตรฐานที่ถูกพัฒนาโดยกลุ่มบริษัทกว่า 20 บริษัทและถูกใช้เป็นมาตรฐานการเปิดเครือข่ายเพื่อจัดการการติดต่อโดยใช้โพรโตคอลต่างๆ วินโดว์สซ็อกเก็ตทำให้โปรแกรมเมอร์ทางด้านแอพพลิเคชันมีความสามารถในการสร้างแอพพลิเคชันที่เกี่ยวกับอินเทอร์เนต อินทราเน็ต และแอพพลิเคชันทางเน็ตเวิร์กอื่นๆ ในการส่งข้อมูลผ่านสายของเครือข่าย โดยที่ไม่ขึ้นกับโพรโตคอลของเน็ตเวิร์กที่ใช้ ในการใช้วินโดว์สซ็อกเก็ต 2 ทำให้โปรแกรมเมอร์ทางด้านแอพพลิเคชันสามารถเข้าถึงความสามารถทางด้านเน็ตเวิร์กของวินโดว์สอย่างเช่น Multicast และ Quality of Service (QoS) ได้ ซึ่งวินโดว์สซ็อกเก็ต 2 มีความคอมแพตทิเบิลกับแอพพลิเคชันของวินโดว์สซ็อกเก็ต 1.1 ด้วย

วินโดว์สซ็อกเก็ต 2 สามารถใช้บนระบบปฏิบัติการวินโดว์สทุกชนิด โดยใช้ต้นแบบซ็อกเก็ตที่ได้รับความนิยมที่มาจาก Berkeley Software Distribution (BSD) ของระบบปฏิบัติการยูนิกซ์ ภายหลังได้ถูกปรับปรุงโดยไมโครซอฟท์วินโดว์สในวินโดว์สซ็อกเก็ต 1.1 ซึ่งวินโดว์สซ็อกเก็ต 2 เป็นอินเทอร์เฟซไม่ใช้โพรโตคอล ทำให้สามารถค้นพบและใช้ความสามารถของการติดต่อสื่อสารของโพรโตคอลบนชั้นทรานสปอร์ตใดๆ ได้ วินโดว์สซ็อกเก็ต 2 ได้ถูกออกแบบมาเพื่อใช้โดยโปรแกรมเมอร์ภาษา C และ C++ ซึ่งควรจะมีความรู้เกี่ยวกับเน็ตเวิร์กของวินโดว์สด้วยพอสมควร

7.3 ฟังก์ชันต่างๆ ของซ็อกเก็ต

รายละเอียดฟังก์ชันของวินโดว์สซ็อกเก็ต 2 แบ่งเป็น 2 กลุ่ม คือ ฟังก์ชันในแบบของเบิร์กเลย์ (Berkeley) และ ฟังก์ชันที่เพิ่มเติมโดยไมโครซอฟท์ จากในตารางที่ 7-1 จะเป็นซ็อกเก็ตแบบเบิร์กเลย์ ซึ่งเป็นส่วนหนึ่งของ API ในวินโดว์สซ็อกเก็ต 1.1

Routine	Meaning
Accept1	An incoming connection is acknowledged and associated with an immediately created socket. The original socket is returned to the listening state.
Bind	Assigns a local name to an unnamed socket.
Closesocket	Removes a socket from the per-process object reference table. Only blocks if SO_LINGER is set with a nonzero time-out on a blocking socket.
connect1	Initiates a connection on the specified socket.
Getpeername	Retrieves the name of the peer connected to the specified socket.
Getsockname	Retrieves the local address to which the specified socket is bound.
Getsockopt	Retrieves options associated with the specified socket.
htonl2	Converts a 32-bit quantity from host-byte order to network-byte order.
htons2	Converts a 16-bit quantity from host-byte order to network-byte order.
inet_addr2	Converts a character string representing a number in the Internet standard "." notation to an Internet address value.
inet_ntoa2	Converts an Internet address value to an ASCII string in "." notation that is, "a.b.c.d".
ioctlsocket	Provides control for sockets.
listen	Listens for incoming connections on a specified socket.
ntohl2	Converts a 32-bit quantity from network-byte order to host-byte order.
ntohs2	Converts a 16-bit quantity from network byte order to host byte order.
recv1	Receives data from a connected or unconnected socket.
recvfrom1	Receives data from either a connected or unconnected socket.
select1	Performs synchronous I/O multiplexing.
send1	Sends data to a connected socket.
sendto1	Sends data to either a connected or unconnected socket.
setsockopt	Stores options associated with the specified socket.
shutdown	Shuts down part of a full-duplex connection.

socket	Creates an endpoint for communication and returns a socket descriptor.
1 The routine can block if acting on a blocking socket. 2 The routine is retained for backward compatibility with Windows Sockets 1.1, and should only be used for sockets created with AF_INET address family.	

ตารางที่ 7-1 รายละเอียดของฟังก์ชันซ็อกเก็ตในแบบของเบิร์กเลย์

ส่วนตารางที่ 7-2 จะเป็นรายละเอียดของฟังก์ชันที่ใช้กับซ็อกเก็ตซึ่งไมโครซอฟท์ได้ทำเพิ่มเติมจากเซตรูทีนมาตรฐานของ Berkeley ตามหลักแล้วฟังก์ชันที่เพิ่มเติมนี้ทำให้แมสเซจหรือฟังก์ชันสามารถเข้าถึงอีเวนต์ของเน็ตเวิร์กแบบอะซิงโครนัส รวมทั้งทำให้มีการทับซ้อนกันของ I/O (overlapped I/O) ได้ด้วย

Routine	Meaning
WSAAccept1	An extended version of **accept**, which allows for conditional acceptance.
WSAAsyncGetHostByAddr2 3 WSAAsyncGetHostByName2 3 WSAAsyncGetProtoByName2 3 WSAAsyncGetProtoByNumber2 3 WSAAsyncGetServByName2 3 WSAAsyncGetServByPort2 3	A set of functions that provide asynchronous versions of the standard Berkeley getXbyY functions. For example, the **WSAAsyncGetHostByName** function provides an asynchronous, message-based implementation of the standard Berkeley **gethostbyname** function.
WSAAsyncSelect3	Performs asynchronous version of **select**.
WSACancelAsyncRequest2 3	Cancels an outstanding instance of a **WSAAsyncGetXByY** function.
WSACleanup	Signs off from the underlying Windows Sockets .dll.
WSACloseEvent	Destroys an event object.
WSAConnect1	An extended version of connect which allows for exchange of connect data and QOS specification.
WSACreateEvent	Creates an event object.
WSADuplicateSocket	Allows an underlying socket to be shared by creating a virtual socket.
WSAEnumNetworkEvents	Discovers occurrences of network events.
WSAEnumProtocols	Retrieves information about each available protocol.
WSAEventSelect	Associates network events with an event object.
WSAGetLastError3	Obtains details of last Windows Sockets error.

WSAGetOverlappedResult	Gets completion status of overlapped operation.
WSAGetQOSByName	Supplies QOS parameters based on a well-known service name.
WSAHtonl	Extended version of **htonl**.
WSAHtons	Extended version of **htons**.
WSAIoctl1	Overlapped-capable version of IOCTL.
WSAJoinLeaf1	Adds a multipoint leaf to a multipoint session.
WSANtohl	Extended version of **ntohl**.
WSANtohs	Extended version of ntohs.
WSAProviderConfigChange	Receives notifications of service providers being installed/removed.
WSARecv1	An extended version of **recv** which accommodates scatter/gather I/O, overlapped sockets and provides the *flags* parameter as in, out.
WSARecvFrom1	An extended version of recvfrom which accommodates scatter/gather I/O, overlapped sockets and provides the *flags* parameter as in, out.
WSAResetEvent	Resets an event object.
WSASend1	An extended version of **send** which accommodates scatter/gather I/O and overlapped sockets.
WSASendTo1	An extended version of **sendto** which accommodates scatter/gather I/O and overlapped sockets.
WSASetEvent	Sets an event object.
WSASetLastError3	Sets the error to be returned by a subsequent WSAGetLastError.
WSASocket	An extended version of socket which takes a **WSAPROTOCOL_INFO** structure as input and allows overlapped sockets to be created.
WSAStartup3	Initializes the underlying Windows Sockets .dll.
WSAWaitForMultipleEvents1	Blocks on multiple event objects.
1 The routine can block if acting on a blocking socket. 2 The routine is always realized by the name resolution provider associated with the default TCP/IP service provider, if any. 3 The routine was originally a Windows Sockets 1.1 function	

ตารางที่ 7-2 รายละเอียดของฟังก์ชันซ็อกเก็ตที่เพิ่มเติมของไมโครซอฟท์

ส่วนฟังก์ชันอื่นๆ ก็มีฟังก์ชันที่เกี่ยวกับการใช้และเข้าถึงบริการการให้ชื่อของเน็ตเวิร์กต่างๆ (network naming services) เมื่อมีการเรียกใช้ฟังก์ชันเหล่านี้ แอพพลิเคชันของวินโดว์สซ็อกเก็ต 2 ไม่จำเป็นต้องแยกแยะความแตกต่างของโพรโตคอลที่เกี่ยวข้องกับบริการการให้ชื่อ อย่างเช่น DNS, NIS, X.500, SAP ฯลฯ ฟังก์ชันในการเข้าถึงฐานข้อมูลที่มีอยู่ทั้งหมดของ getXbyY และ WSAAsyncGetXbyY ยังคงสนับสนุนอยู่ เพื่อให้สามารถคอมแพตทิเบิลกับวินโดว์สซ็อกเก็ต 1.1 ได้ วินโดว์สซ็อกเก็ต 1.1 ยังมีฟังก์ชันที่ยังคงใช้ในการหาชื่อในเครือข่าย TCP/IP ซึ่งเรียกว่า getXbyY ซึ่งมีดังต่อไปนี้

gethostname

gethostbyaddr

gethostbyname

getprotobyname

getprotobynumber

getservbyname

getservbyport

ส่วนเวอร์ชันของฟังก์ชันเหล่านี้ที่เป็นอะซิงโครนัส ก็มีดังนี้

WSAAsyncGetHostByAddr

WSAAsyncGetHostByName

WSAAsyncGetProtoByName

WSAAsyncGetProtoByNumber

WSAAsyncGetServByName

WSAAsyncGetServByPort

ฟังก์ชันต่อไปนี้เป็นฟังก์ชันที่ใช้ในการแปลงแอดเดรสอินเทอร์เน็ตของ Ipv4 กลับไปมาระหว่างรูปแบบสตริงและไบนารี ซึ่งมี 2 ฟังก์ชัน (มีการอิมพลีเมนต์ใน Winsock2.dll) ดังนี้

inet_addr

inet_ntoa

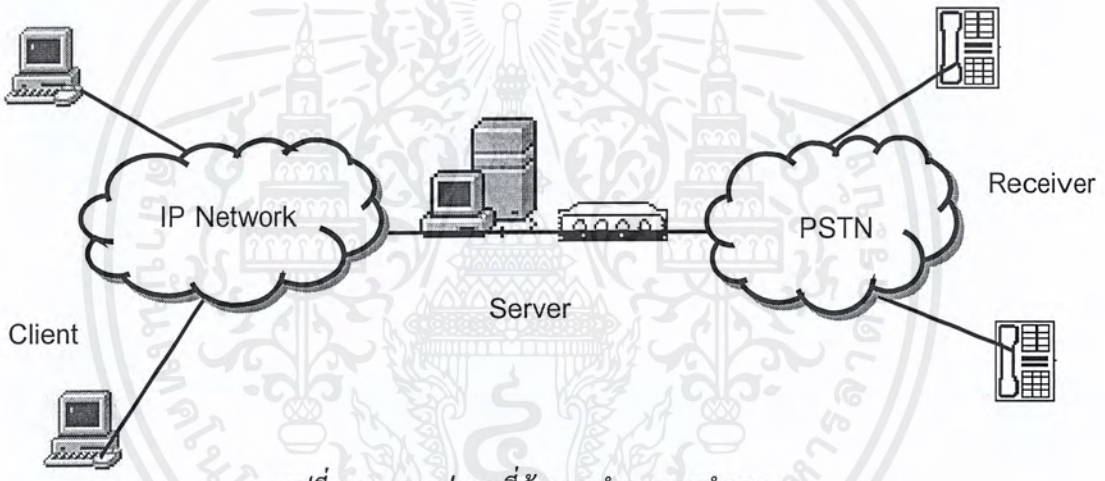
7.4 FTP (File Transfer Protocol)

โพรโตคอลที่เก่าแก่ที่สุดตัวหนึ่งที่ยังคงอยู่ในปัจจุบันก็คือ File Transfer Protocol หรือที่รู้จักกันดีในนามของ FTP ซึ่งเริ่มพัฒนาขึ้นในปี ค.ศ. 1971 FTP ได้ออกแบบมาใช้ในการย้ายไฟล์จากคอมพิวเตอร์เครื่องหนึ่งไปยังอีกเครื่องหนึ่งและรับมือกับปัญหาของการแปลงรูปแบบของไฟล์เมื่อมีการติดต่อกันของต่างชนิดของคอมพิวเตอร์ ในการเข้าเซิร์ฟเวอร์ FTP จะต้องมีชื่อและรหัสผ่าน ข้อมูลนี้จะใช้ในการเทียบระดับของการเข้าถึงในเซิร์ฟเวอร์

บทที่ 8
ระบบส่งข้อความเสียงอัตโนมัติ

8.1 แนวคิดในการออกแบบ

ระบบส่งข้อความเสียงอัตโนมัตินี้เป็นการส่งข้อความเสียงไปยังผู้รับโทรศัพท์ปลายทาง โดยสามารถตั้งเวลาที่ต้องการในการส่งข้อความเสียงได้ ระบบนี้จะแบ่งการออกแบบเป็น 2 ส่วน คือ ส่วนของไคลเอนต์ซึ่งเป็นแอพพลิเคชันสำหรับผู้ที่ต้องการใช้งาน ทำการส่งข้อความเสียงออกไป และอีกส่วนคือส่วนของเซิร์ฟเวอร์ซึ่งเป็นแอพพลิเคชันเช่นกัน จะเป็นตัวรับข้อความเสียงและเวลาที่ต้องการโทรออกจากฝั่งของไคลเอนต์ และทำการโทรศัพท์เพื่อเล่นไฟล์เสียงที่รับฝากข้อความไว้นั้นผ่านโมเด็มออกไปยังเลขหมายโทรศัพท์ปลายทางดังรูปที่ 8-1 ซึ่งทางฝั่งไคลเอนต์จะติดต่อกับฝั่งเซิร์ฟเวอร์โดยใช้เครือข่ายไอพีหรือเครือข่ายอินเทอร์เน็ต



รูปที่ 8-1 แสดงรูปแบบที่ต้องการจำลองการทำงาน

8.2 ขั้นตอนในการพัฒนา

8.2.1 ฮาร์ดแวร์และซอฟต์แวร์ที่ต้องใช้

เนื่องจากเทคโนโลยีของซอฟต์แวร์ที่นำมาใช้ในการอิมพลีเมนต์โครงงานนี้เป็นเรื่องที่ยังใหม่อยู่ ทำให้ระบบปฏิบัติการที่ใช้จะต้องเป็นวินโดว์ส 2000 (หรือสูงกว่า) เท่านั้น เพราะจะต้องใช้กับ TAPI 3.0 ซึ่งจะมี Unimodem driver เป็น Unimodem5 โดยสามารถดูตารางเปรียบเทียบการสนับสนุนของระบบปฏิบัติการต่างๆ ได้จากในตารางที่ 8-1 ซึ่งจากตารางจะสรุปได้ดังนี้

Unimodem สนับสนุนเฉพาะ TAPI เวอร์ชัน 1.4 รวมทั้งมีความสามารถของ Datamodem และ InteractiveVoice แต่ไม่สามารถทำการเล่นและอัดเสียง wavไฟล์ได้

Unimodem/V สนับสนุนเฉพาะ TAPI เวอร์ชัน 1.4 เหมือนกับ Unimodem แต่เพิ่มความสามารถของ InteractiveVoice, CallerID, การดักจับสัญญาณ DTMF และการเล่นและอัดเสียง wav ไฟล์ได้

Unimodem5 มีคุณสมบัติเหมือนกับ Unimodem/V และสนับสนุน TAPI 3.0 ด้วย

Operating System	Unimodem	TAPI Versions
Windows 95 (original)	Unimodem	1.4
Windows 95 (with upgrade)	Unimodem/V	2.1
Windows 98	Unimodem/V	2.1
Windows ME	Unimodem/V	2.1
Windows NT4 (Service Pack 3)	Unimodem	2.0
Windows NT4 (Service Pack 4+)	Unimodem	2.1
Windows 2000	Unimodem5	2.2/3.0

ตารางที่ 8-1 Unimodem และเวอร์ชันของ TAPI ที่สนับสนุนในระบบปฏิบัติการต่างๆ

ซอฟต์แวร์ที่ต้องใช้ในการพัฒนา

- Microsoft Windows 2000 เป็นระบบปฏิบัติการที่สามารถรองรับการทำงานของ TAPI 3.0 ได้
- Microsoft Visual Studio 98 (Visual C++ และ Visual Basic) ใช้ในการพัฒนาแอพพลิเคชันไคลเอนต์และเซิร์ฟเวอร์บนระบบปฏิบัติการวินโดว์ส 2000
- Microsoft Access 2000 ใช้ในการเปิดและติดต่อกับฐานข้อมูลของระบบ
- Microsoft Platform SDK เป็นทูลที่ช่วยในการพัฒนาแอพพลิเคชันทางด้าน TAPI
- Microsoft Speech SDK เป็นทูลที่ช่วยในการพัฒนาแอพพลิเคชันทางด้าน SAPI

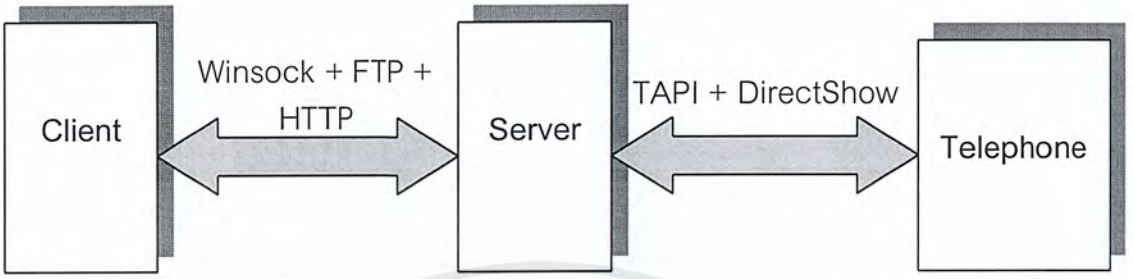
Platform SDK (Platform Software Development Kit) ของไมโครซอฟท์เป็นทูลที่ช่วยในการพัฒนาซอฟต์แวร์บนระบบปฏิบัติการ Microsoft Windows Millennium, Microsoft Windows 2000, ผลิตภัณฑ์ของ Microsoft BackOffice® Server และบนซอฟต์แวร์ Microsoft Internet Explorer เวอร์ชัน 5.5 สำหรับตัว Platform SDK นั้นสามารถติดตั้งลงบนเครื่องที่มีระบบปฏิบัติการ Windows 2000, Windows NT 4.0, Windows 98 หรือ Windows 95 โดยต้องใช้ควบคู่กับตัวคอมไพเลอร์ที่สร้างโปรแกรมแบบ 32 บิต อย่างเช่น Microsoft Visual C++® เวอร์ชัน 6

เครื่องเซิร์ฟเวอร์ที่ใช้ทดลองมีรายละเอียดดังนี้

- CPU Pentium III 550e MHz
- RAM 128 MB
- D-Link Voice Modem
- Sound Blaster Live Value

8.2.2 วิธีในการติดต่อ

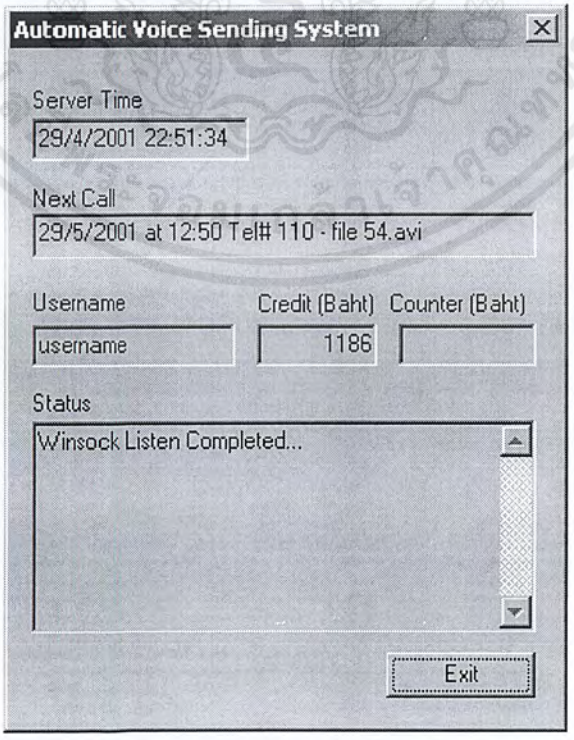
การติดต่อระหว่างไคลเอนต์และเซิร์ฟเวอร์จะใช้ Winsock 2.0 เป็นหลัก และใช้ FTP ในการส่งไฟล์เสียง และ HTTP เพื่อใช้ในการเข้าชมเวบไซต์ในการดาว์นโหลดโปรแกรมของทางฝั่งไคลเอนต์ ส่วนเซิร์ฟเวอร์จะใช้ TAPI 3.0 และ DirectShow ในการโทรและเล่นไฟล์เสียงออกผ่านทางโมเด็ม ดังรูปที่ 8-2



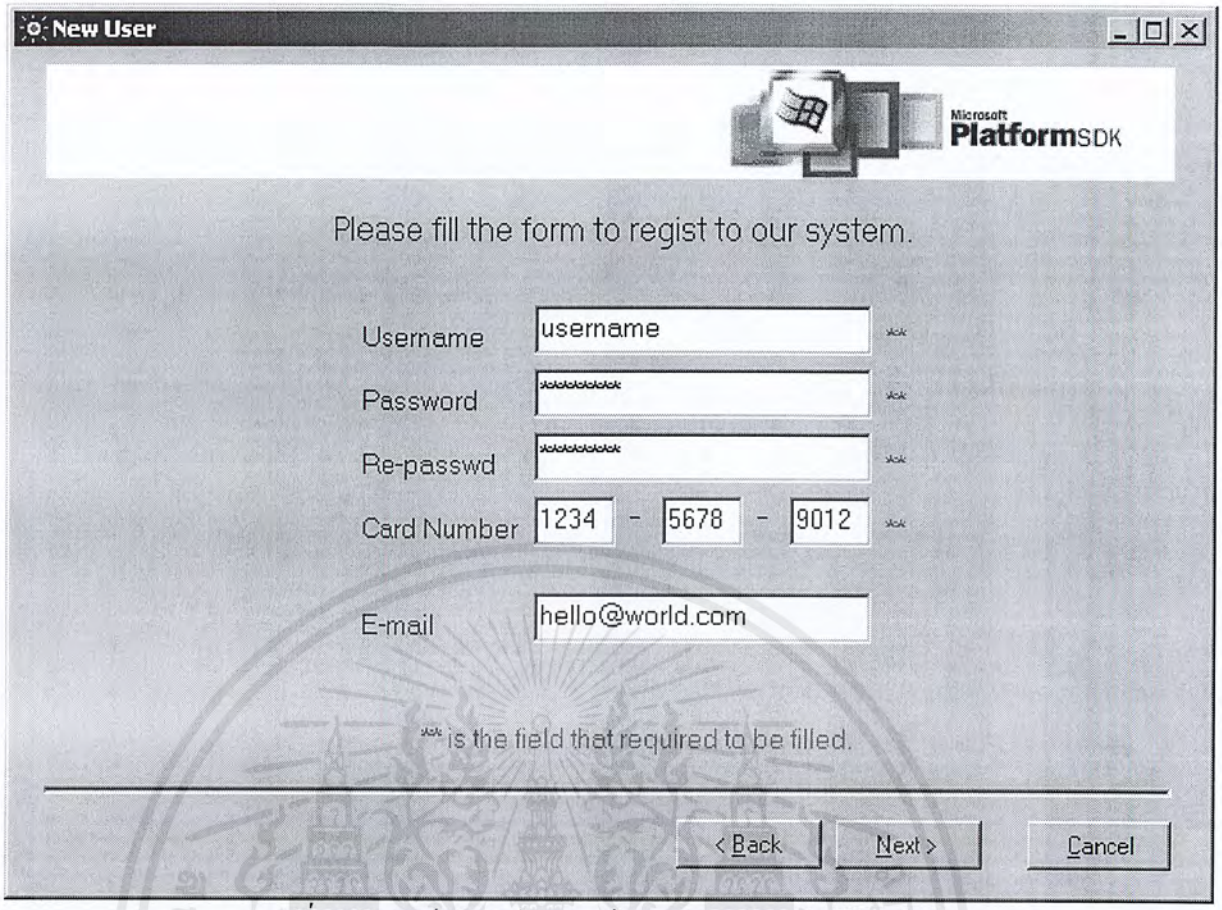
รูปที่ 8-2 รูปแบบการติดต่อของระบบส่งข้อความเสียงอัตโนมัติ

8.2.3 อินเทอร์เฟซของระบบ

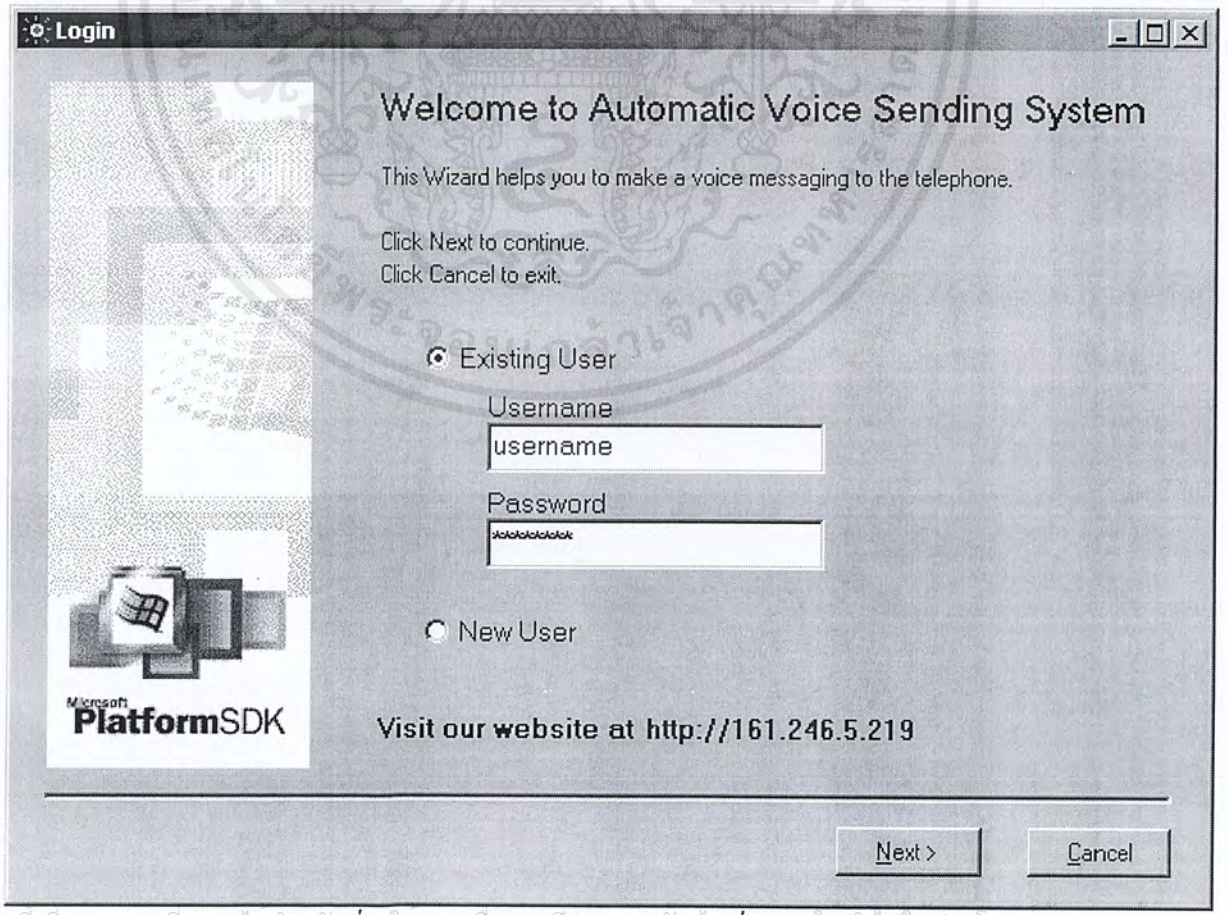
จากรูปที่ 8-3 แสดงถึงหน้าตาอินเทอร์เฟซของเซิร์ฟเวอร์เมื่อทำการรันแอพพลิเคชันขึ้นมา โดยแสดงสถานะเวลาปัจจุบันของเซิร์ฟเวอร์ มีสถานะบอกถึงรายการที่จะทำการโทรออกในครั้งต่อไปเมื่อเวลามาถึง ส่วนรูปที่ 8-4 แสดงหน้าตาอินเทอร์เฟซของไคลเอนต์ในการลงทะเบียนเป็นผู้ใช้ใหม่ รูปที่ 8-5 แสดงหน้าตาอินเทอร์เฟซของไคลเอนต์ในการเข้าสู่ระบบของผู้ใช้ รูปที่ 8-6 แสดงหน้าตาอินเทอร์เฟซของไคลเอนต์ในการเพิ่มรายการที่ต้องการให้เซิร์ฟเวอร์โทรออก หลังจากนั้นเมื่อกด Next ก็จะไปยังรูปที่ 8-7 ซึ่งแสดงหน้าตาอินเทอร์เฟซของไคลเอนต์ในการเลือกบันทึกเสียง หรือสร้างเสียงโดยใช้ข้อความเท็กซ์



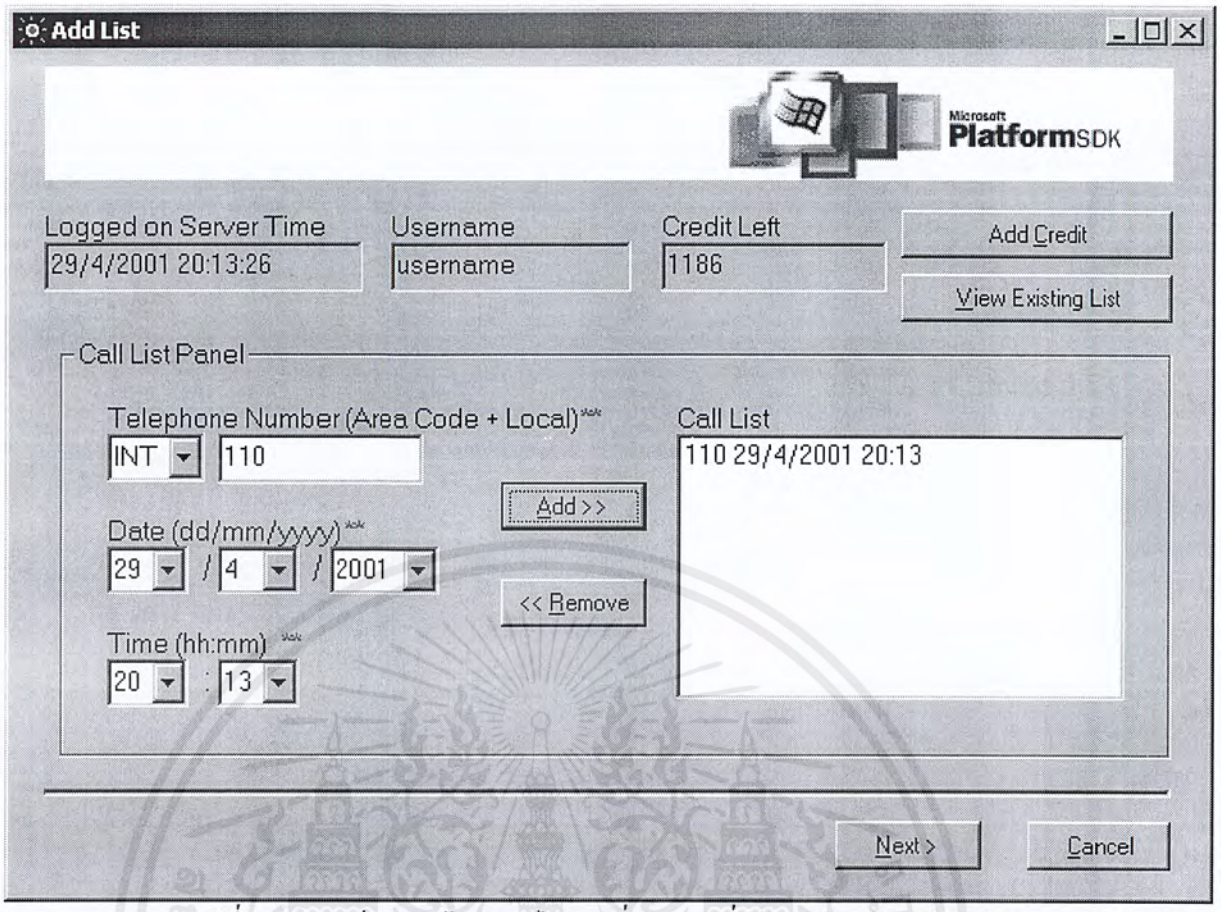
รูปที่ 8-3 อินเทอร์เฟซของเซิร์ฟเวอร์



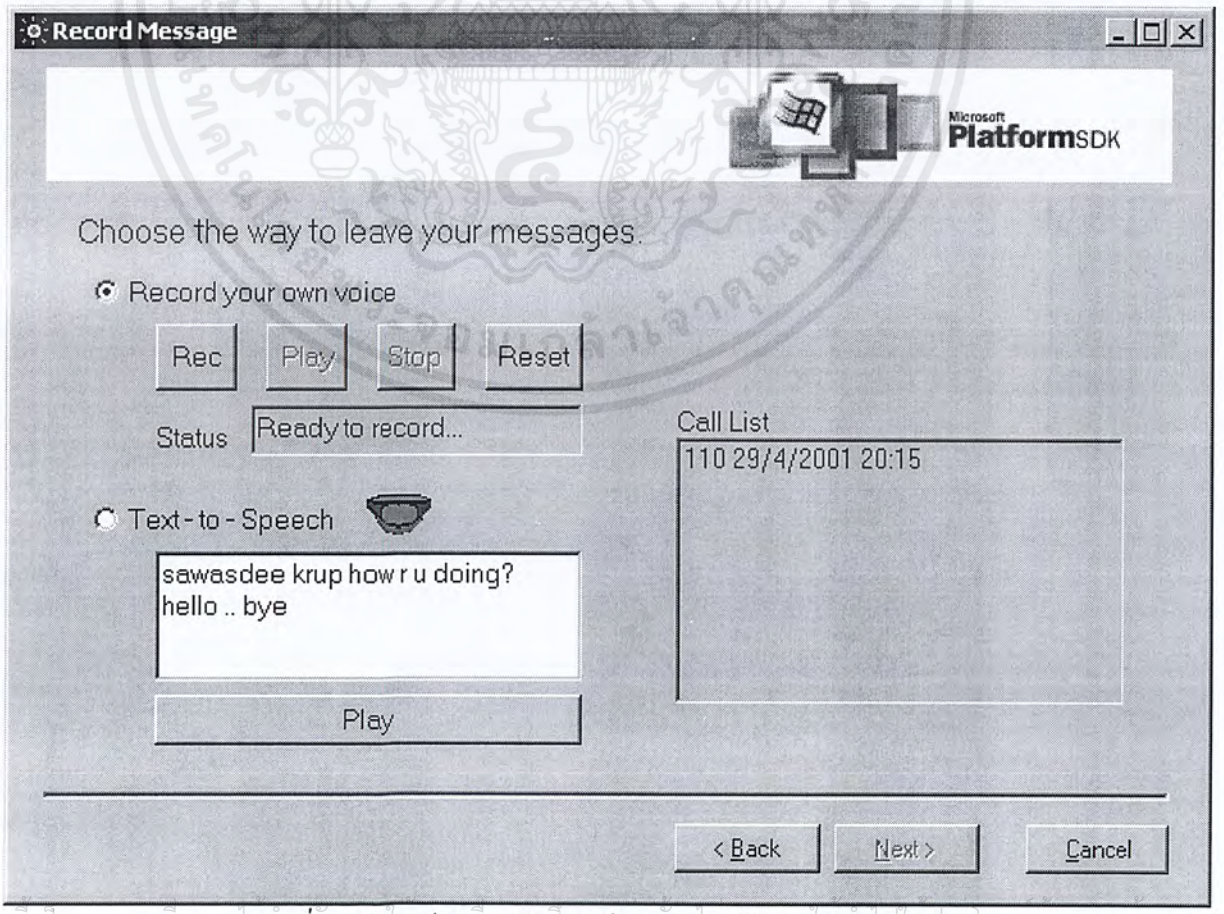
รูปที่ 8-4 อินเทอร์เฟซของไคลเอนต์ในการลงทะเบียนเป็นผู้ใช้คนใหม่



รูปที่ 8-5 อินเทอร์เฟซของไคลเอนต์ในการที่ผู้ใช้เข้าสู่ระบบ



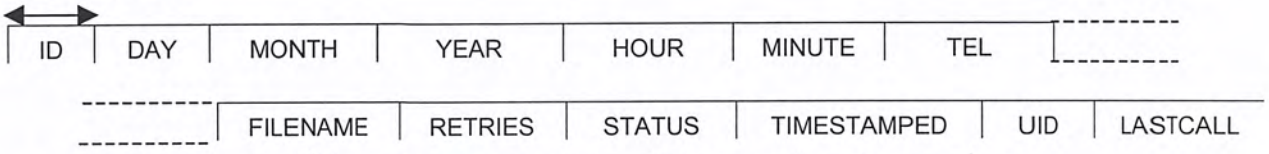
รูปที่ 8-6 อินเทอร์เฟซของไคลเอนต์ในการเพิ่มรายการที่มีข้อมูลต่างๆ ในการโทรออก



รูปที่ 8-7 อินเทอร์เฟซของไคลเอนต์ในการลงทะเบียนเป็นผู้ใช้คนใหม่

8.2.4 ตารางฐานข้อมูลของระบบ

ตาราง RecordT



ตาราง RecordT เป็นตารางที่เก็บบันทึกรายการโทรศัพท์ของผู้ใช้ มีรายละเอียดดังในตารางที่ 8-2

ชื่อแอตทริบิวต์	รายละเอียด	ชนิด
ID	รหัสประจำรายการโทรออกในแต่ละครั้ง	Long Integer
DAY	วันที่ทำการโทรออก	Integer
MONTH	เดือนที่ทำการโทรออก	Integer
YEAR	ปีที่ทำการโทรออก	Integer
HOUR	ชั่วโมงที่ทำการโทรออก	Integer
MINUTE	นาทีที่ทำการโทรออก	Integer
TELNO	หมายเลขโทรศัพท์ปลายทางที่ทำการโทรออก	Text
FILENAME	ชื่อไฟล์เสียงที่เล่นเมื่อทำการโทรออก	Integer
RETRIES	จำนวนการโทรซ้ำเมื่อไม่มีผู้รับสายหรือสายไม่ว่าง	Integer
STATUS	สถานะของรายการโทร	Text
TIMESTAMPED	ลำดับความสำคัญในการเลือกรายการโทรออก	Integer
UID	รหัสผู้ใช้ที่เป็นเจ้าของรายการโทรออก	Long Integer
LASTCALL	วันและเวลาครั้งสุดท้ายที่รายการนี้ถูกโทรออก	Text

ตารางที่ 8-2 รายละเอียดของการเก็บข้อมูลในตาราง RecordT

ตาราง CardT

ตาราง CardT เป็นตารางที่เก็บบันทึกบัตรโทรศัพท์ มีรายละเอียดดังในตารางที่ 8-3

ชื่อแอตทริบิวต์	รายละเอียด	ชนิด
CID	รหัสหมายเลขของบัตรโทรศัพท์	Long Integer
CARD_NO	รหัสที่ใช้ในการลงทะเบียนและ ในการเพิ่มเครดิตให้ผู้ใช้	Text
UID	รหัสผู้ใช้ที่ใช้บัตรนี้	Long Integer
ACTIVATE	วันและเวลาที่บัตรนี้ถูกใช้	Text
C_TYPE	ชนิดของบัตรโทรศัพท์	Text

ตารางที่ 8-3 รายละเอียดของการเก็บข้อมูลในตาราง CardT

ตาราง CardTypeT

ตาราง CardTypeT เป็นตารางที่เก็บบันทึกชนิดของบัตรโทรศัพท์ มีรายละเอียดดังในตารางที่ 8-4

ชื่อแอตทริบิวต์	รายละเอียด	ชนิดและขนาด
C_TYPE	ชนิดของบัตรโทรศัพท์	Text
VALUE	มูลค่าของบัตรที่ใช้เพิ่มเครดิตให้ผู้ใช้	Integer
PRICE	ราคาของบัตร	Integer

ตารางที่ 8-4 รายละเอียดของการเก็บข้อมูลในตาราง CardTypeT

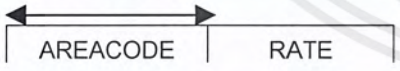
ตาราง UserT

ตาราง UserT เป็นตารางที่เก็บบันทึกรายชื่อของผู้ใช้บริการในระบบ มีรายละเอียดดังในตารางที่ 8-5

ชื่อแอตทริบิวต์	รายละเอียด	ชนิดและขนาด
UID	รหัสของผู้ใช้บริการ	Long Integer
USER	ชื่อผู้ใช้บริการ	Text
PASSWORD	รหัสส่วนตัวของผู้ใช้บริการ	Text
CREDIT	จำนวนเงินที่ผู้ใช้สามารถใช้ในการโทรออกได้	Long Integer
EMAIL	อีเมล์ของผู้ใช้บริการ	Text

ตารางที่ 8-5 รายละเอียดของการเก็บข้อมูลในตาราง UserT

ตาราง CallRateT



ตาราง CallRateT เป็นตารางที่เก็บบันทึกอัตราค่าบริการในการโทรออก มีรายละเอียดดังในตารางที่ 8-6

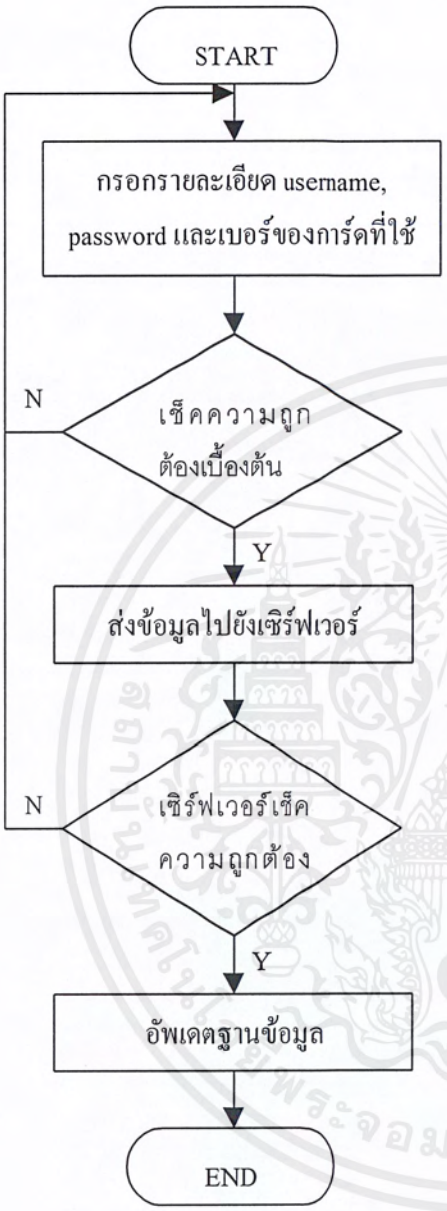
ชื่อแอตทริบิวต์	รายละเอียด	ชนิดและขนาด
AREACODE	รหัสของพื้นที่ที่ให้บริการ	Text
RATE	อัตราค่าบริการในการโทรออก	Double

ตารางที่ 8-6 รายละเอียดของการเก็บข้อมูลในตาราง CallRateT

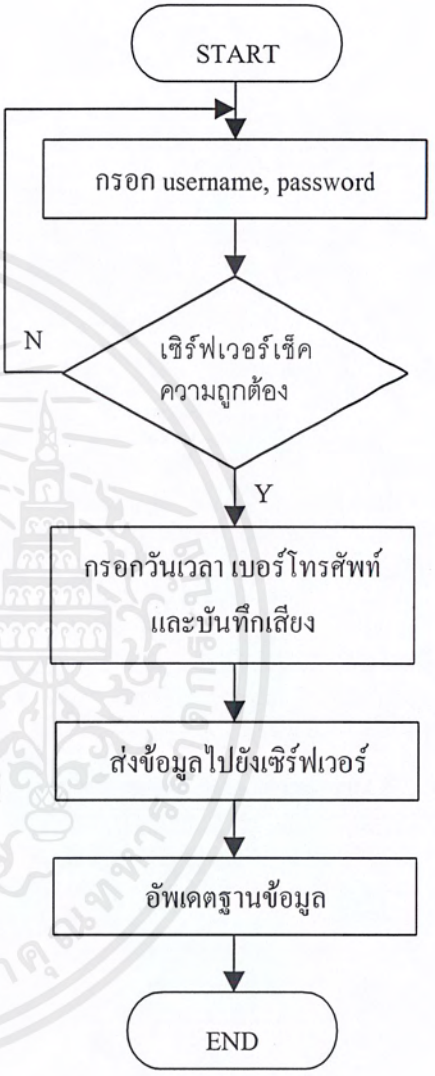
8.3 วิธีการทำงานของระบบโดยรวม

1. ผู้ใช้ทำการดาว์นโหลดโปรแกรมที่เวบ http://161.246.5.219
2. ในการต้องการใช้งาน จะต้องสั่งซื้อการ์ดเพื่อได้รับเครดิตในการใช้งาน ทางฝั่งผู้ให้บริการจะส่งการ์ดให้ทางไปรษณีย์
3. เมื่อผู้ใช้ได้รับการ์ดแล้ว ก็สามารถเข้าใช้บริการโดยกรอกชื่อผู้ใช้ รหัสส่วนตัวและรหัสของการ์ดลงไป (ในกรณีผู้ที่มีเครดิตอยู่แล้วสามารถเข้าใช้บริการได้โดยกรอกเพียงชื่อผู้ใช้และรหัสส่วนตัว)
4. จากนั้นฝั่งเซิร์ฟเวอร์จะทำการบันทึกว่าการ์ดนั้นได้ใช้งานไปแล้วหรือยัง หากยังก็จะทำการเพิ่มเครดิตตามประเภทของการ์ดและการ์ดนั้นก็ไม่สามารถจะใช้งานได้อีกต่อไป แต่ถ้าหากใช้งานไปแล้วก็ไม่สามารถเพิ่มเครดิตได้
5. เมื่อผู้ใช้ต้องการส่งข้อความเสียงไปยังปลายทางก็สามารถเลือกวันเวลาที่จะต้องการส่งข้อความเสียง หลังจากนั้นจึงเลือกรูปแบบของข้อความเสียง จะเป็น Text-to-Speech หรือบันทึกเสียงของผู้ใช้เองก็ได้ หากผู้ใช้ไม่มีเครดิตเพียงพอก็จะไม่สามารถเพิ่มรายการที่โทรออกไปให้ฝั่งเซิร์ฟเวอร์ได้
6. ทางฝั่งเซิร์ฟเวอร์เมื่อได้รับข้อความเสียงและเวลาที่จะโทรออกแล้ว ก็จะเช็คทุกๆ วินาทีว่าถึงเวลาโทรออกแล้วหรือยัง ถ้าถึงแล้ว ก็จะเช็คว่ามีเครดิตเพียงพอที่จะโทรหรือไม่ หากมีเพียงพอก็จะทำการโทรออกไปยังเลขหมายโทรศัพท์ปลายทาง แล้วก็จะเล่นไฟล์เสียงกล่าวนำออกไปก่อน
7. ในกรณีที่ผู้ใช้โทรศัพท์ปลายทางไม่รับสาย พอถึงเวลาที่กำหนดไว้ (ตั้งไว้ที่ 25 วินาที) ก็จะทำการวางหูและเลือกรายการโทรออกครั้งต่อไป ซึ่งเป็นเช่นเดียวกันกับในกรณีที่สายไม่ว่างเพียงแต่จะทำการวางหูภายใน 2-3 วินาทีเท่านั้น
8. เมื่อผู้ใช้รับโทรศัพท์และทำการกดปุ่มโทรศัพท์ปุ่มใดๆ ทางฝั่งเซิร์ฟเวอร์ก็จะเล่นไฟล์เสียงของผู้ใช้ต้นทางออกไป (หากทำการรับโทรศัพท์แล้วแต่ไม่ได้กดปุ่มก็จะเหมือนกับในกรณีที่ผู้ใช้ปลายทางไม่รับสาย)
9. ขณะที่เล่นไฟล์เสียงที่บันทึกไว้ออกนั้น ก็จะทำการจับเวลาเพื่อคิดค่าบริการตามอัตราค่าบริการโทรศัพท์ที่ได้ตั้งเอาไว้ ถ้าหากเครดิตของผู้ใช้ต้นทางไม่เพียงพอในขณะที่เล่นไฟล์อยู่นั้น ก็จะทำการตัดการติดต่อนั้นทันที
10. เมื่อผู้ใช้ปลายทางวางสายลง ทางฝั่งเซิร์ฟเวอร์ก็จะหักค่าบริการจากเครดิตที่มีอยู่

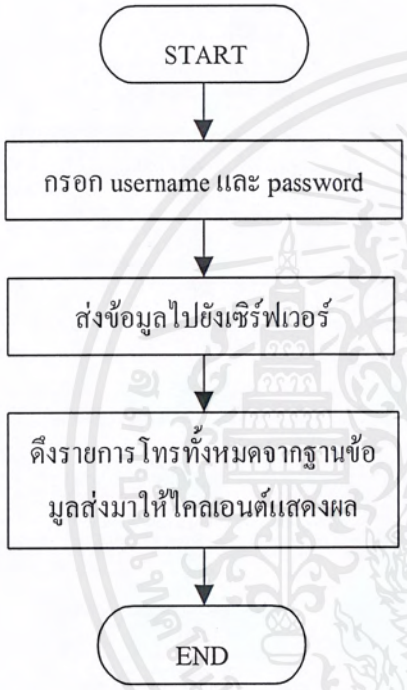
ซึ่งโดยรวมของระบบจะสามารถแสดงเป็น Flowchart ของไคลเอนต์และเซิร์ฟเวอร์ได้ โดยตามรูปที่ 8-8 เป็นการแสดง Flowchart ของไคลเอนต์ในการลงทะเบียนของผู้ใช้ใหม่และเพิ่มเครดิต ส่วนในรูปที่ 8-9 เป็นการแสดง Flowchart ของไคลเอนต์ในการเพิ่มรายการในการโทรออก รูปที่ 8-10 เป็นการแสดง Flowchart ของไคลเอนต์ที่ทำการรับรายการโทรทั้งหมดของผู้ใช้คนนั้นจากเซิร์ฟเวอร์ และจากรูปที่ 8-11 เป็นการแสดง Flowchart ของเซิร์ฟเวอร์ที่จะทำการโทรศัพท์ออกไปยังเลขหมายปลายทาง



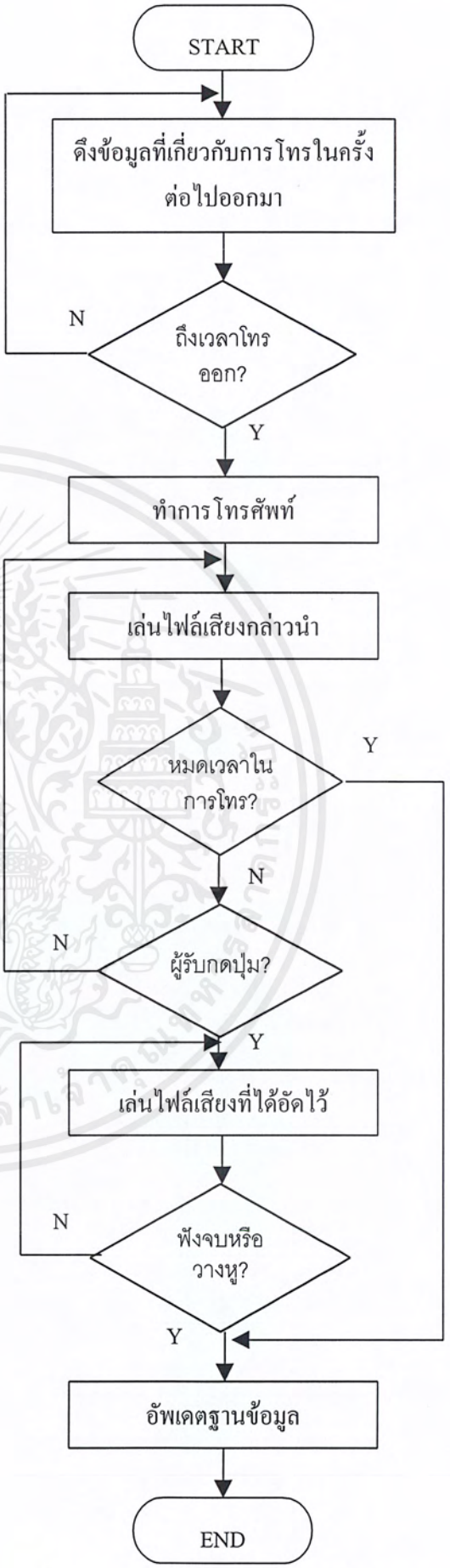
รูปที่ 8-8 Flowchart ของไคลเอนต์ในการเข้าสู่ระบบ



รูปที่ 8-9 Flowchart ของไคลเอนต์ในการเพิ่มรายการโทรออก



รูปที่ 8-10 Flowchart ของไคลเอนต์ในการดูรายการโทรทั้งหมดของผู้ใช้



รูปที่ 8-11 Flowchart ของเซิร์ฟเวอร์ที่จะทำการโทรศัพท์ออก

บทที่ 9
ผลการทดลอง บทวิจารณ์และสรุป

9.1 ผลการทดลองใช้งาน

เมื่อคอมไพล์ไฟล์ของไคลเอนต์ทั้งหมดซึ่งมีโค้ดทั้งหมดประมาณ 3,928 บรรทัดจะได้ไฟล์ .EXE ที่มีขนาด 909,312 ไบต์ ส่วนเซิร์ฟเวอร์เมื่อคอมไพล์ไฟล์ทั้งหมดซึ่งมีโค้ดประมาณ 5,416 บรรทัดจะมีขนาดไฟล์ 194,606 ไบต์ จะเห็นได้ว่าทางฝั่งไคลเอนต์ที่ใช้ Visual Basic 6.0 จะมีขนาดของไฟล์ที่ใหญ่กว่าทางเซิร์ฟเวอร์ที่ใช้ Visual C++ 6.0 ค่อนข้างมาก

ในการรันแอพพลิเคชันไคลเอนต์จะใช้หน่วยความจำหลักบนระบบปฏิบัติการ Microsoft Windows 2000 Advanced Server เท่ากับ 3,220 KB และแอพพลิเคชันเซิร์ฟเวอร์จะใช้หน่วยความจำหลักเท่ากับ 7,784 KB

จากการทดลองใช้งานจริงในระบบเครือข่ายแลนและเครือข่ายโทรศัพท์ภายในคณะวิศวกรรมศาสตร์ภาควิชาวิศวกรรมคอมพิวเตอร์ของสถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบังสรุปได้ว่าคุณภาพเสียงโดยมีอัตราแซมปลิ้งที่ 8 KHz เรโซลูชันที่ 16 บิต จำนวน 1 ช่องสัญญาณ (mono) จะเหมาะสมในการใช้งานเนื่องจากไฟล์มีขนาดเล็กและคุณภาพเสียงเป็นที่ยอมรับได้ค่อนข้างดี

9.2. สรุปและวิจารณ์

จากโครงงานนี้ ทำให้ทราบถึงการอิมพลีเมนต์เทคโนโลยีต่างๆ เพื่อนำมาใช้ร่วมกันได้อย่างมีประโยชน์มากขึ้น เทคโนโลยีเหล่านั้นได้แก่ TAPI 3.0, DirectShow, SAPI 5.0, Winsock 2.0 และ FTP แต่การนำมาผสมผสานการใช้ในเครือข่ายไอพีและเครือข่ายโทรศัพท์ยังมีอุปสรรคซึ่งเทคโนโลยีนั้นไม่สนับสนุนในเครื่องนั้นๆ อย่างเช่น ใน TAPI 3.0 ของไมโครซอฟท์ยังไม่สามารถที่จะตรวจดักสถานะของโทรศัพท์ได้ ซึ่งได้แก่ สัญญาณของสายไม่ว่างและสัญญาณสายเมื่อมีคนรับสายแล้ว เป็นต้น และในการส่งข้อมูลเสียงไปยังโมเด็มยังเป็นเพียง half-duplex เท่านั้น ทำให้การสื่อสารเป็นไปได้เพียงทางเดียวและผู้ใช้ปลายทางก็ไม่สามารถตอบสนองกลับไปยังเซิร์ฟเวอร์ได้อย่างเต็มที่ รวมทั้งขอบเขตของการใช้งานอยู่เพียงในวงแคบๆ เท่านั้น ทำให้การอิมพลีเมนต์ระบบส่งข้อความเสียงอัตโนมัติต้องมีอุปสรรค เกิดความล่าช้าและเกิดข้อจำกัดขึ้นด้วยเช่นกัน จึงต้องหาแนวทางหรือวิธีการอื่นมาใช้แทน ดังนั้นก่อนที่จะเริ่มทำโครงงานใดๆ ก็ควรที่จะต้องศึกษาความเป็นไปได้ (feasibility) และข้อจำกัดของเทคโนโลยีที่นำมาใช้นั้นเสียก่อน แต่เนื่องด้วยใน TAPI 3.1 จะมีอินเทอร์เฟซที่ใช้ในการตรวจดักสถานะของโทรศัพท์ได้ จึงคาดว่าสามารถทำให้ระบบที่จะพัฒนาขึ้นมาในอนาคตมีความสามารถที่ดีขึ้น

9.3 ข้อเสนอแนะ

1. ทำการเพิ่มฟังก์ชันในการส่งรูปภาพแทนการส่งเสียงไปที่เซิร์ฟเวอร์ และทำการส่งแฟกซ์ไปยังผู้ใช้ปลายทาง เพื่อลดค่าใช้จ่ายลง
2. ทำการพัฒนาเอนจิ้นภาษาไทยของ Text-to-Speech เพื่อสามารถทำการสร้างเสียงเป็นภาษาไทยจากข้อความที่เป็นเท็กซ์ธรรมดา และเพิ่มความสามารถของ Text-to-Speech ให้พูดไฟล์เสียงเป็นน้ำเสียงอื่นๆ ได้
3. เพิ่มไฟล์เสียงส่งท้ายหลังจากที่เล่นไฟล์เสียงที่บันทึกไว้เสร็จเรียบร้อยแล้ว

บรรณานุกรม

[1] Microsoft Corporation (August 2000) : "*MSDN Library: Platform SDK*" http://msdn.microsoft.com/library/default.asp, August 2000.

[2] Microsoft Corporation (September 2000) : " *Microsoft Windows 2000 Server*" http://www.microsoft.com/WINDOWS2000/library/howitworks/communications/telephony/iptelephony.asp, September 2000.

[3] Maher H., Oliver V., Jean-Pierre H. and others (May 1999) : "*Voice Service Interworking for PSTN and IP Networks*" IEEE Communication Magazine, May 1999, pp. 104-125.

[4] Vinodkrishnan K. (November 1999) : "*Voice over IP: Products, Services and Issues*", http://www.cis.ohio-state.edu/~jain/cis788-99/voip_products/index.html, November 1999.

[5] Lijun F. (1997) : "*Internet Telephony*", http://www.cis.ohio-state.edu/~jain/cis788-97/internet_telephony/index.htm, 1997.

[6] Walter J.G. and Matthew C.K. (1999) : "*IP Telephony*", McGraw-Hill, New York, 1999.

[7] Viktor T. (1996) : "*Visual C++ 4 UNLEASHED*", Sams Publishing, Indiana, 1996.

[8] นิรุธ อำนวยศิลป์ : "*คู่มือการเขียนโปรแกรม Microsoft Visual C++ Version 6.0 ฉบับเพื่อการใช้งานจริง*", ซัคเซส มีเดีย, กรุงเทพฯ.

[9] Michael D. (May 1999) : "*Michael Dunn's TAPI Site*", http://members.home.net/tapifaq/tapi.htm, May 1999.

[10] Google (2001) : "*Google Groups : microsoft.public.platformsdk.telephony.tapi_3*", http://groups.google.com, 2001.

[11] Ippolito I. (1997) : "*Planet Source Code*", http://www.planet-source-code.com, Exhedra Solutions Inc.