การออกแบบและพัฒนาระบบแจ้งการเปลี่ยนแปลงของบริการในระบบกระจาย

นาย ภาสิน สุริเยนทรากร

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2544
ISBN 974-03-0431-1

DESIGN AND DEVELOPMENT OF A CHANGE NOTIFICATION SYSTEM FOR DISTRIBUTED SERVICES

Mr. Pasin Suriyentrakorn

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Engineering in Computer Engineering

Department of Computer Engineering
Faculty of Engineering
Chulalongkorn University

Academic Year 2001

ISBN 974-03-0431-1

หัวข้อวิทยานิพนธ์	การออกแบบและพัฒนาระบบแจ้งการเปลี่ยนแปลงของบริการในระบบกระจาย
โดย	นาย ภาสิน สุริเยนทรากร
สาขาวิชา	วิศวกรรมคอมพิวเตอร์
อาจารย์ที่ปรึกษา	อาจารย์ ดร. ทวิตีย์ เสนีวงศ์ ณ อยุธยา

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

..... คณบดีคณะวิศวกรรมศาสตร์
(ศาสตราจารย์ ดร.สมศักดิ์ ปัญญาแก้ว)

คณะกรรมการสอบวิทยานิพนธ์

..... ประธานกรรมการ
(อาจารย์ ดร.ยรรยง เต็งอำนวย)

..... อาจารย์ที่ปรึกษา
(อาจารย์ ดร.ทวิตีย์ เสนีวงศ์ ณ อยุธยา)

.....กรรมการ
(อาจารย์ ดร.ณัฐวุฒิ หนูไพโรจน์)

.....กรรมการ
(อาจารย์ ดร.ธันวดี สุเนตนันท์)

ภาสิน สุริเยนทรากร : การออกแบบและพัฒนาระบบแจ้งการเปลี่ยนแปลงของบริการในระบบกระจาย. (DESIGN AND DEVELOPMENT OF A CHANGE NOTIFICATION SYSTEM FOR DISTRIBUTED SERVICES) อ. ที่ปรึกษา : อ.ดร.ทวิตีย์ เสนีวงศ์ ณ อยุธยา, 139 หน้า, ISBN 974-03-0431-1

งานวิจัยนี้ได้ทำการออบแบบ และพัฒนาบริการแจ้งการเปลี่ยนแปลงบริการ หรือ เอสซีเอ็น เพื่อแจ้งการเปลี่ยนแปลงประเภทต่างๆ ที่เกิดขึ้นกับบริการ อันได้แก่ การเปลี่ยนแปลงชนิดของบริการ การเปลี่ยนแปลงค่าคุณสมบัติของบริการ การยกเลิกบริการ และการเพิ่มบริการ ให้กับผู้ใช้งานบริการในระบบกระจายคอร์บา บริการเอสซีเอ็นสามารถรับข้อมูลการเปลี่ยนแปลงบริการจากทั้งผู้เปลี่ยนแปลงบริการ โดยมีจุดประสงค์เพื่อเป็นการแจ้งการเปลี่ยนแปลงบริการล่วงหน้าก่อนที่จะมีการเปลี่ยนแปลงจริง และจากตัวตรวจบริการเทรดเดอร์ ซึ่งทำหน้าที่ตรวจสอบการเปลี่ยนแปลงบริการโดยอัตโนมัติในขณะที่ข้อเสนอของบริการที่อยู่ในบริการเทรดเดอร์ได้รับการเปลี่ยนแปลงเพื่อเป็นการแจ้งการเปลี่ยนแปลงบริการในขณะที่ได้รับการเปลี่ยนแปลง นอกจากนี้บริการเอสซีเอ็นยังสนับสนุนต่อการแจ้งการเปลี่ยนแปลงในรูปแบบต่างๆ โดยกำหนดให้ผู้ต้องการรับทราบการเปลี่ยนแปลงบริการสามารถกำหนดรูปแบบการจัดส่งรวมทั้งรูปแบบของข้อมูลที่ใช้ในการจัดส่งได้

ภาควิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อนิสิต .....

สาขาวิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่ออาจารย์ที่ปรึกษา .....

ปีการศึกษา 2544 ลายมือชื่ออาจารย์ที่ปรึกษาร่วม .....

#4270483221: MAJOR COMPUTER ENGINEERING

KEY WORD : CHANGE MANAGEMENT / NOTIFICATION SERVICE / CORBA

PASIN SURIYENTRAKORN: DESIGN AND DEVELOPMENT OF A CHANGE NOTIFICATION SYSTEM FOR DISTRIBUTED SERVICES. THESIS ADVISOR: DR. TWITTIE SENIVONGSE, 139 pp. ISBN 974-03-0431-1

This research proposes a design and development of a Service Change Notification Service (SCN) that will enable clients within a CORBA distributed environment to be informed of change of several kinds that may happen to service objects they use. Such changes consist of change of service type of service, change of property values of service, removal of service, and instantiation of service. SCN can receive change information from service evolvers for advance change notification before actual change process and trader monitor which is responsible for automatically detect changes of service offers within trader service for immediate change notification. Furthermore, SCN provides various kinds of notifications that allow users to subscribe their event delivery styles and notification formats.

Department Computer Engineering Student's signature .....

Field of study Computer Engineering Advisor's signature .....

Academic year 2001 Co-advisor's signature .....

กิตติกรรมประกาศ

ขอขอบคุณ อาจารย์ ดร.ทวิตีย์ เสนีวงศ์ ณ อยุธยา ที่ให้ความช่วยเหลือเป็นอย่างมากให้การทำงานวิจัยนี้ และสละเวลาให้คำปรึกษาและคำแนะนำที่ดีเสมอมา

ขอขอบคุณ อาจารย์ ดร.ยรรยง เต็งอำนวย อาจารย์ ดร. ณัฐวุฒิ หนุไพโรจน์ และอาจารย์ ดร. ธันวดี สุเนตนันท์ กรรมการวิทยานิพนธ์ ที่กรุณาให้คำแนะนำและตรวจสอบแก้ไขต้นฉบับวิทยานิพนธ์นี้

ขอขอบคุณ เพื่อนๆ พี่ๆ และน้องๆ ทั้งหลาย ที่ให้คำปรึกษา รวมทั้งความสนุกสนาน และความบันเทิงมาโดยตลอด

สุดท้ายนี้ ขอขอบพระคุณ บิดา มารดา และครอบครัว ที่ให้การสนับสนุนในด้านต่างๆ และให้กำลังใจที่ดีเสมอมา

ภาสิน สุริเยนทรากร

สารบัญ

สารบัญ (ต่อ)

สารบัญ (ต่อ)

สารบัญตาราง

สารบัญรูปภาพ

สารบัญรูปภาพ (ต่อ)

สารบัญรูปภาพ (ต่อ)

สารบัญรูปภาพ (ต่อ)

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ในการพัฒนาซอฟต์แวร์หรือแอปพลิเคชันต่างๆนั้นเราไม่สามารถหลีกเลี่ยงการเปลี่ยนแปลงได้ ถึงแม้ว่าซอฟต์แวร์เหล่านั้นจะได้รับการออกแบบหรือพัฒนาขึ้นมาอย่างดีแล้วก็ตาม การเปลี่ยนแปลงต่างๆ ที่เกิดขึ้นมักมีสาเหตุมาจาก

- การแก้ไขข้อผิดพลาด
- การปรับให้เข้ากับยุคสมัยและเทคโนโลยี
- การขยายขีดความสามารถและการเพิ่มประสิทธิภาพ
- ความต้องการที่เปลี่ยนไปของผู้ใช้งาน

ในระบบกระจายก็เช่นเดียวกันย่อมหลีกเลี่ยงการเปลี่ยนแปลงไม่ได้ และด้วยคุณสมบัติของระบบกระจายซึ่งมีบริการ (Service) ต่างๆ อยู่กระจายทั่วไป ทำให้การควบคุมและจัดการกับการเปลี่ยนแปลงมีความลำบากและซับซ้อนมากยิ่งขึ้น การเปลี่ยนแปลงที่เกิดขึ้นกับบริการหนึ่งๆ จะส่งผลต่อเนื่องไปยังแอปพลิเคชันอื่นที่กำลังใช้งานบริการเหล่านั้นอยู่ การเปลี่ยนแปลงที่มักเกิดขึ้นกับบริการสามารถจำแนกออกได้เป็น 3 ประเภทหลักๆ ดังต่อไปนี้

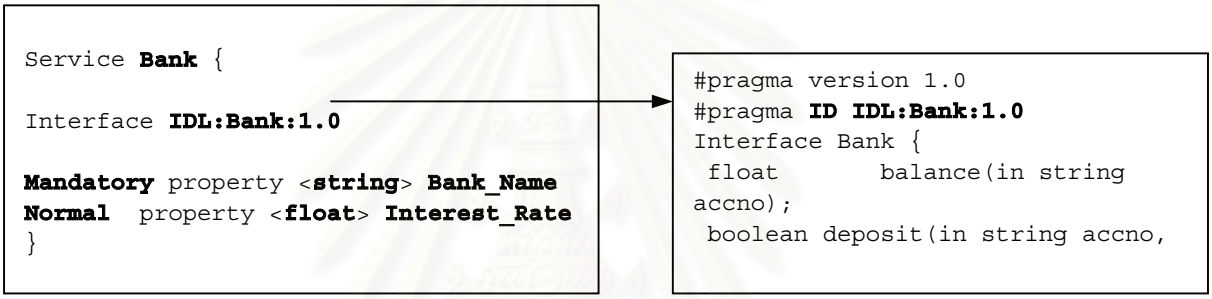
1. การเปลี่ยนแปลงชนิดของบริการ (Service Type)

บริการต่างๆ ในระบบกระจายได้รับการจำแนกออกเป็นกลุ่มต่างๆ ตามชนิดของบริการ ซึ่งมีโครงสร้างดังรูปที่1.1 โดยชนิดของบริการประกอบด้วย ชื่อชนิดส่วนต่อประสานของบริการ (Interface Type Name) รายการคุณสมบัติของบริการ (Service Property Definition) และรายการชื่อชนิดของบริการที่สืบทอด (Based Service Type Name) โดยชื่อชนิดส่วนต่อประสานของบริการจะแสดงถึงส่วนต่อประสาน (Interface Signature) ที่ผู้รับบริการ (Client) ใช้ในการติดต่อสื่อสารกับบริการ ส่วนรายการความหมายคุณสมบัติของบริการจะแสดงถึงคุณสมบัติที่บริการหนึ่งๆ จำเป็นต้องมี

```
Service <ServiceTypeName>[:<BaseServiceTypeName> [,
                        <BaseServiceTypeName>]*] {
        interface <InterfaceTypeName>;
        {[mandatory][read-only][Normal] property <IDL Type> <PropertyName>;]*
};
```

รูปที่ 1.1 BNF แสดงโครงสร้างชนิดของบริการ

ตัวอย่างชนิดของบริการสามารถแสดงได้ดังรูปที่ 1.2 ชนิดของบริการชื่อ Bank ประกอบด้วยชนิดส่วนต่อประสาน คือ IDL:Bank:1.0 ซึ่งอ้างอิงกับส่วนต่อประสาน Bank ซึ่งมีตัวกระทำการ (Operation) 3 ตัวคือ ตัวกระทำการ balance() ตัวกระทำการ deposit () และ ตัวกระทำการ withdraw() และรายการคุณสมบัติที่ประกอบด้วยชื่อธนาคาร (Bank_Name) และอัตราดอกเบี้ย (Interest_Rate) สำหรับบริการที่มีชนิด Bank เหมือนกันก็จะสามารถอธิบายได้ด้วยส่วนต่อประสานและรายการคุณสมบัติเดียวกัน แต่จะแตกต่างกันที่ค่าของคุณสมบัติ เช่นบริการหนึ่งมีค่าคุณสมบัติคือ Bank_Name = “CU-Bank” และ Interest_Rate = 3.0 ส่วนอีกบริการหนึ่งมีค่าคุณสมบัติเป็น Bank_Name = “ISEL-Bank” และ Interest_Rate = 5.0 เป็นต้น



รูปที่ 1.2 ตัวอย่างชนิดของบริการ

ชนิดของบริการจะได้รับการเปลี่ยนแปลงเมื่อมีการเปลี่ยนแปลงของส่วนต่อประสานของบริการ รายการคุณสมบัติของบริการ หรือชนิดของบริการที่สืบทอด โดยผลกระทบที่จะเกิดกับผู้รับบริการนั้นจะมีความแตกต่างกันแล้วแต่กรณี ดังต่อไปนี้

<u>กรณีที่ 1.1 บริการได้รับการเปลี่ยนแปลงส่วนต่อประสาน</u> การเปลี่ยนแปลงส่วนต่อประสานทำให้ผู้รับบริการไม่สามารถติดต่อสื่อสารกับบริการได้ด้วยส่วนต่อประสานแบบเดิม ผู้รับบริการจำเป็นต้องทำการแก้ไขการเรียกใช้งานบริการให้ตรงกับส่วนต่อประสานที่ได้รับการเปลี่ยนแปลงหากมีความประสงค์ที่จะใช้งานบริการนั้นต่อไป การไม่ทราบถึงการเปลี่ยนแปลงจะก่อให้เกิดปัญหากับการเรียกใช้งานบริการในทันที ส่งผลเสียหายต่อระบบงานและก่อให้เกิดความไม่พึงพอใจในการใช้งานบริการ ตัวอย่างการเปลี่ยนแปลงส่วนต่อประสานของบริการสามารถแสดงได้ดังรูปที่ 1.3 ซึ่งพบได้ว่าตัวกระทำการ deposit() และ ตัวกระทำการ withdraw() ไม่สามารถเรียกใช้งานได้ในรูปแบบเดิม โดยหากต้องการเรียกใช้งานจะต้องทำการเพิ่มข้อมูลระบุประเภทของบัญชี (AccountType) เข้าเป็นส่วนหนึ่งของรายการพารามิเตอร์ด้วย ส่วนตัวกระทำการ transfer() เป็นตัวกระทำการที่ถูกเพิ่มเข้ามา

ส่วนต่อประสานรุ่นเก่า

```
#pragma version 1.0
#pragma ID IDL:Bank:1.0
Interface Bank {
  float     balance (in string accno);
  boolean   deposit (in string accno,
                     in float amount);
  boolean  withdraw (in string accno,
                     in float amount);
}
```

ส่วนต่อประสานรุ่นใหม่

```
#pragma version 2.0
#pragma ID IDL:Bank:2.0
Interface Bank {
  float   balance  (in string accno);
  Boolean deposit  (in string accno,
                    in AccountType type,
                    in float amount);
  boolean withdraw (in string accno,
                    in AccountType type,
                    in float amount);
  Boolean transfer (in string accno_src,
                    in string accno_dest,
                    in float amount);
}
```

รูปที่ 1.3 ตัวอย่างการเปลี่ยนแปลงส่วนต่อประสาน

กรณีที่ 1.2 บริการได้รับการเปลี่ยนแปลงรายการคุณสมบัติของบริการ การเปลี่ยนแปลงรายการคุณสมบัติของบริการคือการเปลี่ยนแปลงข้อกำหนดทางด้านคุณลักษณะของบริการที่บริการชนิดนั้นๆ จะต้องสนับสนุน การเปลี่ยนแปลงจะส่งผลกระทบกับผู้รับบริการในด้านของคุณลักษณะที่จะมีเท่านั้น ซึ่งไม่ได้ส่งผลต่อวิธีการเรียกใช้งานบริการแต่อย่างใดแต่จะส่งผลต่อวิธีการเลือก เช่น ถ้ามีการเปลี่ยนชื่อของคุณสมบัติหรือคุณสมบัติเดิมหายไปผู้รับบริการก็จำเป็นต้องมีการเขียนโปรแกรมเพื่อทำการค้นหาและเลือกใช้งานบริการใหม่ การรับทราบถึงข้อมูลการเปลี่ยนแปลงรายการคุณสมบัติของบริการมีความจำเป็นต่อผู้รับบริการทั้งเพื่อความทันสมัยต่อข้อมูลคุณลักษณะของบริการและเพื่อใช้เป็นข้อมูลในการตัดสินใจว่าจะยังคงเลือกใช้งานบริการนั้นๆ ต่อไปหรือไม่อย่างไร ตัวอย่างการเปลี่ยนแปลงรายการคุณสมบัติแสดงดังรูปที่ 1.4 ซึ่งจะพบว่าการเปลี่ยนแปลงรายการคุณสมบัติประกอบด้วยการเปลี่ยนแปลงชื่อคุณสมบัติจาก Interest_Rate ไปเป็น Deposit_Interest_Rate และการเพิ่มรายการคุณสมบัติ คือ ReserveFund เข้าไปใหม่ ซึ่งการเปลี่ยนแปลงนี้ส่งผลทำให้ผู้รับบริการจำเป็นต้องแก้ไขวิธีการค้นหาบริการจากเดิมไป เช่น เพิ่ม ReservedFund เข้าไปในการระบุค่าในการค้นหาบริการ หรือเปลี่ยนการระบุค่าคุณสมบัติ Deposit_Interest_Rate แทน Interest_Rate เป็นต้น

ชนิดของบริการก่อนได้รับการเปลี่ยนแปลงรายการคุณสมบัติ

```
Service Bank {

Interface IDL:Bank:1.0
 Mandatory property <string> Bank_Name
 Normal    property <float>  Interest_Rate

}
```

ชนิดของบริการหลังได้รับการเปลี่ยนแปลงรายการคุณสมบัติ

```
Service Bank_2 {

Interface IDL:Bank:1.0

 Mandatory property <string> Bank_Name
 Normal    property <float>  Deposit_Interest_Rate
 Mandatory property <float>  ReservedFund
}
```

รูปที่ 1.4 ตัวอย่างการเปลี่ยนแปลงรายการคุณสมบัติของบริการ

กรณีที่ 1.3 บริการได้รับการเปลี่ยนแปลงรายการชนิดของบริการที่สืบทอด บริการหนึ่งๆ สามารถสืบทอดส่วนต่อประสานและรายการคุณสมบัติได้ ดังนั้นการเปลี่ยนแปลงรายการชนิดของบริการที่สืบทอดก็จะมีผลเช่นเดียวกับการเปลี่ยนแปลงส่วนต่อประสานและรายการคุณสมบัติที่ทำการสืบทอดมา ซึ่งการเปลี่ยนแปลงลักษณะนี้จะส่งผลต่อผู้รับบริการดังในกรณีที่ 1.1 และ 1.2

2. การเปลี่ยนแปลงค่าคุณสมบัติของบริการ

บริการแต่ละตัวจะได้รับการกำหนดค่าคุณสมบัติเฉพาะซึ่งอาจจะเหมือนหรือแตกต่างกัน ข้อมูลค่าคุณสมบัติของบริการถือเป็นข้อมูลที่สำคัญสำหรับการตัดสินใจเลือกใช้งานบริการหนึ่งๆ ของผู้รับบริการ การเปลี่ยนแปลงค่าคุณสมบัติของบริการโดยที่ผู้รับบริการไม่ทราบถึงรายละเอียดของการเปลี่ยนแปลงอาจส่งผลทำให้ผู้รับบริการได้รับบริการที่ไม่ตรงกับความต้องการที่ตั้งใจไว้ ก่อให้เกิดความไม่พึงพอใจ และความเสียหายต่างๆ ต่อระบบงาน โดยเฉพาะอย่างยิ่งหากค่าคุณสมบัติดังกล่าวเป็นปัจจัยสำคัญในการเลือกใช้งานบริการ ตัวอย่างการเปลี่ยนแปลงค่าคุณสมบัติของบริการสามารถแสดงได้ดังรูปที่ 1.5 ซึ่งพบได้ว่าค่าคุณสมบัติที่ได้รับการเปลี่ยนแปลงคือ Interest_Rate โดยทำการเปลี่ยนแปลงค่าจาก 0.4% ไปเป็น 0.1% ซึ่งการเปลี่ยนแปลงนี้จะส่งผลกระทบต่อผู้รับบริการที่ต้องการค่า Interest_Rate ที่ 0.4% โดยเฉพาะอย่างยิ่งหากผู้รับบริการดังกล่าวไม่ทราบถึงการเปลี่ยนแปลงที่เกิดขึ้นนี้มาก่อน ทำให้ไม่สามารถเตรียมรับสถานการณ์ที่อัตราดอกเบี้ยลดต่ำมากได้อย่างมีประสิทธิภาพ

ค่าคุณสมบัติของบริการ (ก่อนเปลี่ยน)
Bank_Name = "CU-Bank"
Interest_Rate = 0.4

ค่าคุณสมบัติของบริการ (เมื่อได้รับการเปลี่ยนแปลง)
Bank_Name = "CU-Bank"
Interest_Rate = 0.1

รูปที่ 1.5 ตัวอย่างการเปลี่ยนแปลงค่าคุณสมบัติของบริการ

3. **การยกเลิกการใช้งานบริการ**

การยกเลิกการใช้งานหรือหยุดการให้บริการไม่ว่าด้วยเหตุผลใดก็ตาม ย่อมส่งผลกระทบต่อผู้ที่กำลังใช้งานบริการชนิดนั้นอยู่อย่างหลีกเลี่ยงไม่ได้ ซึ่งอาจทำให้แอปพลิเคชันที่เรียกใช้งานบริการนั้นอยู่ไม่สามารถที่จะทำงานต่อไปได้ ส่งผลเสียหายต่อระบบงานต่างๆ

เพื่อบรรเทาปัญหาที่จะเกิดขึ้นกับผู้รับบริการในกรณีที่บริการได้รับการเปลี่ยนแปลงไป ระบบจัดการการเปลี่ยนแปลงบริการ (Service Change Management System) จึงเป็นที่ต้องการและมีความสำคัญเป็นอย่างยิ่ง แต่อย่างไรก็ตามในปัจจุบันงานวิจัยที่เกี่ยวข้องกับการจัดการกับการเปลี่ยนแปลงบริการยังคงมีไม่มากนัก (แสดงรายละเอียดในหัวข้อที่ 2) งานวิจัยนี้จึงมีแนวคิดในการออกแบบและพัฒนาระบบสำหรับแจ้งการเปลี่ยนแปลงของบริการในระบบกระจาย ขึ้นโดยมีวัตถุประสงค์เพื่อให้ผู้รับบริการหนึ่งๆ สามารถรับทราบการเปลี่ยนแปลงต่างๆ ของบริการทั้งที่จะเกิดขึ้นในอนาคตและในขณะที่กำลังเกิดขึ้น และสามารถที่จะนำข้อมูลแจ้งการเปลี่ยนแปลงดังกล่าวไปใช้ในการเตรียมการ รองรับ และแก้ปัญหาที่จะเกิดขึ้นต่อไปได้

ระบบแจ้งการเปลี่ยนแปลงของบริการที่ทำการออกแบบและพัฒนาขึ้นจะทำงานอยู่บนระบบกระจายคอร์บา **ผิดพลาด! ไม่พบแหล่งการอ้างอิง** โดยระบบจะมีการทำงานร่วมกับบริการเทรดเดอร์ **ผิดพลาด! ไม่พบแหล่งการอ้างอิง** ซึ่งเป็นบริการไดเรกทอรีพื้นฐานตัวหนึ่งที่ใช้งานอยู่ภายในระบบกระจายคอร์บา เพื่อตรวจสอบการเปลี่ยนแปลงข้อมูลของบริการต่างๆในขณะที่เกิดการเปลี่ยนแปลงขึ้น นอกจากนี้ระบบยังรองรับการแจ้งการเปลี่ยนแปลงที่จะเกิดขึ้นในอนาคตอีกด้วย การแจ้งการเปลี่ยนแปลงไปยังผู้ต้องการรับทราบการเปลี่ยนแปลงจะยึดรูปแบบตามระบบแจ้งเหตุการณ์ (Event Notification System) **ผิดพลาด! ไม่พบแหล่งการอ้างอิงผิดพลาด! ไม่พบแหล่งการอ้างอิง** และอาศัยการช่วยเหลือจากการทำงานของบริการแจ้งการเปลี่ยนแปลง (Notification Service) **ผิดพลาด! ไม่พบแหล่งการอ้างอิง** ซึ่งเป็นบริการตัวหนึ่งในระบบกระจายคอร์บาเช่นเดียวกัน นอกจากนี้ระบบยังสนับสนุนต่อรูปแบบของข้อมูลแจ้งการเปลี่ยนแปลงหลายรูปแบบเพื่อรองรับต่อความต้องการของผู้รับบริการที่มีความหลากหลาย เช่น การแจ้งด้วยรูปแบบวัตถุเหตุการณ์

(Structured Event Object) สำหรับผู้ใช้งานที่อยู่บนระบบกระจายคอร์บา หรือการแจ้งด้วยรูปแบบของเอกสารเอ็กซ์เอ็มแอล (XML) **ผิดพลาด! ไม่พบแหล่งการอ้างอิง** สำหรับการรองรับผู้ใช้งานบนระบบอินเทอร์เน็ต เป็นต้น

1.2 วัตถุประสงค์ของการวิจัย

เพื่อออกแบบและพัฒนาระบบแจ้งการเปลี่ยนแปลงบริการไปยังผู้ใช้บริการในระบบกระจาย

1.3 ขอบเขตของการวิจัย

1. เหตุการณ์การเปลี่ยนแปลงบริการที่พิจารณามีด้วยกัน 4 เหตุการณ์ คือ
    - เหตุการณ์ที่ 1 บริการได้รับการเปลี่ยนแปลงชนิดของบริการ
    - เหตุการณ์ที่ 2 บริการได้รับการเปลี่ยนแปลงค่าคุณสมบัติของบริการ
    - เหตุการณ์ที่ 3 บริการถูกยกเลิกการใช้งาน
    - เหตุการณ์ที่ 4 เกิดบริการใหม่ขึ้นในระบบ
2. รูปแบบการแจ้งการเปลี่ยนแปลงบริการมีลักษณะเป็นแบบ อะซิงโครนัส ทั้งแบบพุชและพูล
3. รูปแบบของเหตุการณ์แจ้งการเปลี่ยนแปลงบริการที่ผู้รับบริการสามารถระบุได้มีด้วยกัน 3 รูปแบบ คือ
    - วัตถุเหตุการณ์ (Structured Event Object)
    - ชุดของวัตถุเหตุการณ์ (Sequence of Structured Event Objects)
    - เอกสารเอ็กซ์เอ็มแอล (XML Document)
4. การแจ้งการเปลี่ยนแปลงบริการประกอบด้วย การแจ้งการเปลี่ยนแปลงล่วงหน้าโดยกำหนดให้ผู้เปลี่ยนแปลงบริการเป็นผู้ให้ข้อมูล และการแจ้งการเปลี่ยนแปลงบริการในขณะที่เกิดการเปลี่ยนแปลงโดยการตรวจสอบจากข้อมูลการร้องขอใช้งานบริการเทรด เดอร์
5. ระบบแจ้งการเปลี่ยนแปลงบริการ 1 ระบบที่ประกอบด้วยส่วนเพิ่มขยายบริการเทรดเดอร์และตัวจัดการการแจ้งการเปลี่ยนแปลงจะทำงานในระบบกระจายที่มีบริการเทรดเดอร์ 1 ตัวเท่านั้น
6. ต้นแบบของระบบจะพัฒนาขึ้นในรูปแบบของบริการหนึ่งในระบบกระจายและจะทำงานอยู่บนสถาปัตยกรรมระบบกระจายที่ถูกพัฒนาขึ้นตามข้อกำหนดของคอร์บารุ่นปรับปรุงที่ 2.2 บริการเทรดเดอร์ที่จะทำการเพิ่มขยาย อยู่ภายใต้ข้อกำหนดของคอร์บารุ่นปรับปรุงที่ 2.2 และบริการแจ้งเหตุการณ์ที่นำมาใช้ อยู่ภายใต้ข้อกำหนดของคอร์บารุ่นปรับปรุงที่ 2.2

1.4 ขั้นตอนในการดำเนินงาน

1. ศึกษาทฤษฎีและการทำงานของระบบกระจายคอร์บาและลักษณะการเปลี่ยนแปลงต่างๆ ของบริการ
2. ศึกษาทฤษฎีของบริการเทรดเดอร์ ระบบแจ้งเหตุการณ์ บริการแจ้งเหตุการณ์ และทฤษฎีที่เกี่ยวกับเอ็กซ์เอ็มแอล
3. ออกแบบระบบแจ้งการแจ้งการเปลี่ยนแปลงบริการและศึกษาทดลองเครื่องมือต่างๆที่จะนำมาใช้งาน
4. พัฒนาระบบแจ้งการเปลี่ยนแปลงบริการ
5. ทดสอบและปรับปรุงแก้ไข
6. วิเคราะห์และสรุปผล พร้อมข้อเสนอแนะ
7. จัดทำรายงานวิทยานิพนธ์

1.5 ประโยชน์ที่จะได้รับ

ได้ระบบแจ้งการเปลี่ยนแปลงบริการ ซึ่งสามารถบรรเทาปัญหาที่เกิดขึ้นกับผู้ใช้งานบริการในกรณีที่บริการได้รับการเปลี่ยนแปลง

1.6 ผลงานตีพิมพ์

ส่วนหนึ่งของวิทยานิพนธ์นี้ได้ตีพิมพ์ และนำเสนอในงานประชุมวิชาการดังนี้

1. การประชุมทางวิชาการวิทยาการและวิศวกรรมคอมพิวเตอร์แห่งชาติ ครั้งที่ 4 (The 4th National Computer Science and Engineering Conference (NCSEC 2000)) เมื่อวันที่ 16-17 พฤศจิกายน 2543 ในบทความเรื่อง An Approach for Service Change Notification in Distributed System

โดยผู้แต่งคือ Pasin Suriyentrakorn and Twittie Senivongse

2. The 5th International Enterprise Distributed Object Conference (EDOC 2001) September 4-7 2001, Seattle, Washington USA ในบทความเรื่อง A CORBA-Based Architecture for Service Change Notification

โดยผู้แต่งคือ Twittie Senivongse and Pasin Suriyentrakorn

บทที่ 2
งานวิจัยและทฤษฎีที่เกี่ยวข้อง

2.1 งานวิจัยที่เกี่ยวข้อง

2.1.1 ORB Interface Type Version Management Request for Proposals [9]

เป็นบทความประกาศรับข้อเสนอสำหรับการจัดการรุ่นของส่วนต่อประสานในระบบกระจายโดยโอเอ็มจี (OMG : Object Management Group) เพื่อกำหนดมาตรฐานสำหรับบริการจัดการการเปลี่ยนแปลงส่วนต่อประสาน (Interface Change Management Service) ของระบบคอร์บา (CORBA) ภายในบทความได้กล่าวถึงปัญหาที่เกิดขึ้นจากการเปลี่ยนรุ่นของส่วนต่อประสานที่มีผลต่อกลุ่มบุคคลต่างๆ ที่มีความสัมพันธ์กันอยู่ภายในระบบกระจาย คือ ผู้พัฒนาบริการ ผู้พัฒนาแอปพลิเคชันและผู้ใช้งานแอปพลิเคชัน การเปลี่ยนแปลงส่วนต่อประสานของบริการย่อมส่งผลกระทบไม่มากก็น้อยต่อผู้ที่นำบริการหรือส่วนต่อประสานดังกล่าวไปใช้ เช่น การกำหนดส่วนต่อประสานโดยสืบทอดมาจากส่วนต่อประสานที่เปลี่ยนรุ่นไป โดยการเปลี่ยนแปลงอาจไม่ได้เกิดมาจากผู้พัฒนาบริการเองแต่อาจเกิดจากผู้พัฒนาแอปพลิเคชัน หรือผู้ใช้งานแอปพลิเคชันก็เป็นได้ อย่างไรก็ตามผลของการร้องขอนี้พบว่าไม่ได้รับการตอบรับเท่าที่ควร ทั้งนี้อาจเนื่องมาจากความสนใจของกลุ่มนักวิจัยทางด้านระบบกระจายในขณะนั้นมุ่งเน้นไปในด้านอื่น เช่น ความสามารถในการติดต่อสื่อสารกันบนสภาวะแวดล้อมหรือแพลตฟอร์ม (Platform) ของระบบคอร์บาที่พัฒนาโดยผู้พัฒนา (Vendor) คนละราย

2.1.2 Evolution Transparency for Distributed Service Types [10]

งานวิจัยชิ้นนี้เป็นงานวิจัยที่พยายามแก้ปัญหาที่เกิดจากการเปลี่ยนรุ่นของบริการอันเนื่องมาจากการเปลี่ยนแปลงส่วนต่อประสานของบริการ โดยผู้วิจัยได้นำเสนอวิธีการที่ทำให้การเปลี่ยนรุ่นของบริการมีลักษณะโปร่งใส (Transparency) ผู้รับบริการด้วยส่วนต่อประสานแบบเก่าสามารถที่จะใช้งานต่อไปได้โดยไม่จำเป็นต้องทราบถึงการเปลี่ยนแปลง โดยมีการสร้างตัวดำเนินการแปลง (Mapping Operator) ขึ้นมาโดยอัตโนมัติเมื่อมีการเปลี่ยนรุ่นของบริการ ซึ่งข้อมูลในการเปลี่ยนรุ่นจะได้มาจากผู้ที่ทำการเปลี่ยนรุ่น เมื่อมีการขอใช้บริการด้วยส่วนต่อประสานแบบเก่า คำขอใช้บริการนั้นจะถูกแปลงโดยตัวดำเนินการแปลง ซึ่งจะใช้ฟังก์ชันการแปลงในการแปลงคำขอใช้บริการนั้นให้อยู่ในรูปแบบของส่วนต่อประสานตัวใหม่ ทั้งนี้ตัวดำเนินการแปลงจะถูกสร้างให้เสมือนกับเป็นบริการรุ่นเก่า เพื่อที่จะสามารถดัก (Intercept) คำขอใช้บริการรุ่นเก่าได้ แต่อย่างไรก็ตามผู้วิจัยได้ให้ทรรศนะไว้ว่าวิธีการแก้ไขปัญหาในลักษณะเช่นนี้เป็นการแก้ปัญหาเพียงชั่วคราว

เท่านั้น ซึ่งเมื่อถึงเวลาหนึ่งผู้รับบริการก็ควรที่จะรับทราบถึงการเปลี่ยนแปลง และควรที่จะมีการปรับเปลี่ยนการใช้งานให้สอดคล้องกับการเปลี่ยนแปลงที่เกิดขึ้นในที่สุด

2.1.3 A Notification System for Service Providers of Services with Evolved Interfaces in CORBA Distributed System [11]

งานวิจัยนี้เป็นงานวิจัยที่สร้างระบบแจ้งการเปลี่ยนแปลงส่วนต่อประสานของบริการเพื่ออำนวยความสะดวกแก่ผู้พัฒนาบริการรายต่างๆ ที่พัฒนาบริการขึ้นตามส่วนต่อประสาน โดยผู้พัฒนาบริการจะได้รับการแจ้งจากระบบเพื่อให้ทำการพัฒนาบริการขึ้นมาใหม่ หรือปรับเปลี่ยนบริการเดิมให้สอดคล้องกับส่วนต่อประสานที่ได้เปลี่ยนไปแล้ว ทั้งนี้เพื่อให้บริการต่างๆ ที่มีอยู่สามารถปรับตัวให้ทันสมัยและสอดคล้องกับนิยามของบริการซึ่งกำหนดโดยส่วนต่อประสานเสมอ อย่างไรก็ดีระบบแจ้งการเปลี่ยนแปลงนี้จะเป็นประโยชน์กับฝั่งพัฒนาบริการแต่ไม่ได้อำนวยความสะดวกแก่ฝั่งผู้ใช้งานบริการแต่อย่างใด

2.1.4 SMARTS: A Smart CORBA Trader Service [12]

งานวิจัยชิ้นนี้มุ่งพัฒนาบริการเทรดเดอร์ ในหลายลักษณะเช่น การกำหนดให้ผู้ใช้สามารถระบุค่าความสำคัญ (Priority) ของการค้นหาบริการ การกำหนดให้ผู้ใช้สามารถเลือกที่จะให้บริการเทรดเดอร์ทำการพุช (Push) ผลของการค้นหาบริการไปยังแคชที่อยู่ทางฝั่งผู้รับบริการแทนรูปแบบการรอรับผลการค้นหาจากบริการเทรดเดอร์ทุกครั้งตามแบบเดิม นอกจากนี้บริการเทรดเดอร์ยังมีความสามารถในการแจ้งการเกิดบริการใหม่ที่ตรงกับความต้องการของผู้รับบริการที่เคยมาทำการค้นหาบริการก่อนหน้านี้ได้ โดยรูปแบบของการแจ้ง จะอาศัยการส่งข้อมูลแจ้งไปยังตัวแทน (Agent) ที่อยู่ทางฝั่งผู้รับบริการ อย่างไรก็ดี บริการเทรดเดอร์นี้ยังไม่สามารถแจ้งการเกิดการเปลี่ยนแปลงของบริการที่มีอยู่แล้วได้

2.2 ทฤษฎีที่เกี่ยวข้อง

2.2.1 คอร์บา (CORBA) [1]

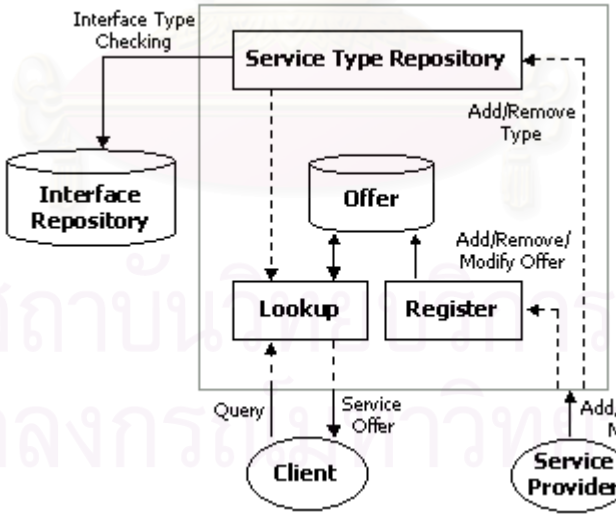
คอร์บาเป็นมาตรฐานสำหรับสถาปัตยกรรมระบบกระจายเชิงวัตถุ (Distributed Object System Architecture) ที่กำหนดขึ้นโดยโอเอ็มจีและมีบทบาทมากในปัจจุบัน โดยมีพื้นฐานมาจากสถาปัตยกรรมการจัดการวัตถุ (OMA: Object Management Architecture) [13] คอร์บาเกิดจากการรวมกันระหว่างการโปรแกรมเชิงวัตถุ (Object-Oriented Programming) และระบบไคลเอนต์/เซิร์ฟเวอร์ เพื่อให้วัตถุในระบบสามารถติดต่อสื่อสารกันได้ โดยวัตถุหนึ่งทำหน้าที่เป็นเซิร์ฟเวอร์

สำหรับบริการ ซึ่งมักเรียกว่า วัตถุบริการ (Object Service) และอีกวัตถุหนึ่งทำหน้าที่เป็นไคลเอนต์สำหรับการขอใช้บริการ

คุณลักษณะที่สำคัญของคอร์บาคือการแยกส่วนการสร้าง (Implementation) ออกจากส่วนข้อกำหนด (Specification) ด้วยการกำหนดให้มีภาษากลางสำหรับการกำหนดรายละเอียดส่วนต่อประสานต่าง ๆ ของวัตถุเซิร์ฟเวอร์ ซึ่งภาษาที่ใช้คือภาษาไอดีแอล (IDL: Interface Definition Language) ส่วนต่อประสานที่กำหนดขึ้นทำหน้าที่เสมือนเป็นโพรโตฑทคอลสำหรับการติดต่อสื่อสารระหว่างวัตถุไคลเอนต์และเซิร์ฟเวอร์ ด้วยวิธีนี้ ไคลเอนต์สามารถติดต่อสื่อสารกับเซิร์ฟเวอร์ได้โดยอาศัยข้อมูลส่วนต่อประสานของเซิร์ฟเวอร์เพียงอย่างเดียวโดยไม่จำเป็นต้องสนใจรายละเอียดของการพัฒนาและภาษาที่ใช้พัฒนา

ส่วนที่สำคัญอีกส่วนหนึ่งซึ่งถือเป็นหัวใจของสถาปัตยกรรมคอร์บาก็คือ ออร์บ (ORB: Object Request Broker) ออร์บทำหน้าที่เป็นตัวกลางในการติดต่อสื่อสารระหว่างวัตถุไคลเอนต์และวัตถุเซิร์ฟเวอร์ ด้วยการใช้งานออร์บนี้ทำให้วัตถุไคลเอนต์สามารถเรียกใช้งานบริการต่างๆ บนเซิร์ฟเวอร์วัตถุ ได้เหมือนๆ กับการเรียกใช้บนตัวไคลเอนต์วัตถุเอง โดยที่ไม่จำเป็นต้องรู้ตำแหน่งที่อยู่ ระบบปฏิบัติการ ลักษณะและภาษาที่ใช้ในการพัฒนาวัตถุเซิร์ฟเวอร์

2.2.2 บริการเทรดเดอร์ (Trader Service) [2]



รูปที่ 2.1 โครงสร้างของบริการเทรดเดอร์

บริการเทรดเดอร์ เป็นบริการพื้นฐานตัวหนึ่งที่กำหนดขึ้นโดยโอเอ็มจี ทำหน้าที่คล้ายกับสมุดหน้าเหลืองสำหรับการค้นหาบริการในระบบกระจายคอร์บา การค้นหาบริการจะอาศัยข้อมูล

ค่าคุณสมบัติต่างๆ ของบริการเป็นคีย์สำคัญในการค้นหาซึ่งต่างจากบริการชื่อ (Naming Service) [14] ที่ใช้ชื่อเป็นคีย์ในการค้นหาบริการ

บริการเทรดเดอร์ประกอบด้วยองค์ประกอบที่สำคัญ 4 ส่วน (รูปที่ 2.1) คือ

- <u>คลังชนิดของบริการ</u> (Service Type Repository) มีลักษณะเป็นฐานข้อมูลสำหรับเก็บข้อมูลชนิดของบริการ (รูปที่ 1.1 ข้างต้น) โดยมีการติดต่อกับคลังส่วนต่อประสาน (Interface Repository) [1] เพื่อตรวจสอบความถูกต้องของชื่อชนิดของส่วนต่อประสาน (Interface Type Name)
- <u>ฐานข้อมูลข้อเสนอบริการ</u> (Offer List) เป็นฐานข้อมูลสำหรับเก็บข้อเสนอของบริการ (Offer) ซึ่งประกอบด้วยข้อมูลดังต่อไปนี้
  1. ข้อมูลอ้างถึงวัตถุ (Object Reference) ที่แสดงตำแหน่งที่อยู่ของตัวบริการ
  2. ชื่อชนิดของบริการ (Service Type Name)
  3. รายการคู่ลำดับของชื่อคุณสมบัติและค่าคุณสมบัติ (Service Properties)
- <u>ส่วนการลงทะเบียน</u> (Register Component) เป็นส่วนที่ผู้พัฒนาบริการใช้ในการติดต่อกับบริการเทรดเดอร์เพื่อขอโฆษณาบริการ โดยปกติแล้วการโฆษณาบริการประกอบด้วยขั้นตอนที่สำคัญ 2 ขั้นตอนด้วยกัน คือ

  <u>ขั้นตอนที่ 1</u> ผู้พัฒนาบริการให้ข้อมูลชนิดของบริการแก่บริการเทรดเดอร์ในกรณีที่ชนิดของบริการนั้นยังไม่มีอยู่ในบริการเทรดเดอร์ซึ่งข้อมูลชนิดของบริการนี้จะได้รับการจัดเก็บยังคลังชนิดของบริการ

  <u>ขั้นตอนที่ 2</u> ผู้พัฒนาบริการให้ข้อมูลข้อเสนอบริการแก่บริการเทรดเดอร์โดยข้อมูลนี้จะได้รับการจัดเก็บยังฐานข้อมูลข้อเสนอบริการ
- <u>ส่วนค้นหาบริการ</u> (Lookup Component) เป็นส่วนที่ผู้รับบริการ ใช้ในการติดต่อเพื่อทำการสืบค้นบริการ ส่วนค้นหาบริการนี้มีการติดต่อกับคลังชนิดของบริการและฐานข้อมูลข้อเสนอบริการสำหรับทำการค้นหาบริการที่ตรงกับความต้องการของผู้ร้องขอมากที่สุด

2.2.3 ระบบแจ้งเหตุการณ์ (Event Notification System) [3] [4]

ระบบแจ้งเหตุการณ์ รูปที่ 2.2 ประกอบด้วยองค์ประกอบที่สำคัญ 3 ส่วนด้วยกัน คือ ผู้แจ้งเหตุการณ์ (Publisher) ผู้ต้องการรับทราบเหตุการณ์ (Subscriber) และ ตัวกลางสำหรับจัดส่งเหตุการณ์ (Event Notification Middleware)

รูปที่ 2.2 โครงสร้างระบบแจ้งเหตุการณ์

ผู้แจ้งเหตุการณ์ทำหน้าที่เป็นแหล่งกำเนิดเหตุการณ์และส่งเหตุการณ์ที่เกิดขึ้นไปยังตัวกลางสำหรับจัดส่งเหตุการณ์ ผู้ที่ต้องการรับทราบเหตุการณ์ต่างๆ จะต้องทำการลงทะเบียนไว้กับตัวกลางสำหรับจัดส่งเหตุการณ์ จากนั้นเมื่อผู้แจ้งเหตุการณ์ทำการส่งเหตุการณ์ผ่านมายังตัวกลาง ตัวกลางก็จะทำการกรองเหตุการณ์ที่ได้รับมาว่าตรงกับความต้องการของผู้รับทราบเหตุการณ์รายใดบ้าง จากนั้นจึงส่งเหตุการณ์ดังกล่าวไปให้แก่วัตถุเรียกกลับ(Callback Object) ของผู้ที่ต้องการรับทราบเหตุการณ์ที่ทำการลงทะเบียนเอาไว้แล้วก่อนหน้านี้

รูปแบบของการจัดส่งเหตุการณ์โดยปกติแล้วสามารถแบ่งออกได้เป็น 2 รูปแบบ คือ

- แบบพุช (Push Model)

  ด้วยรูปแบบการจัดส่งแบบพุช ผู้แจ้งเหตุการณ์จะเป็นผู้ส่งเหตุการณ์ไปยังผู้ที่ต้องการทราบเหตุการณ์ตามทิศทางที่แสดงดังรูปที่ 2.3

รูปที่ 2.3 รูปแบบการจัดส่งเหตุการณ์แบบพุช

- แบบพูล (Pull Model)

  รูปแบบการจัดส่งแบบนี้ (รูปที่ 2.4) มีลักษณะเหมือนการสอบถาม (Polling) โดยที่ผู้ต้องการทราบเหตุการณ์จะเป็นผู้มาร้องขอเหตุการณ์เองจากตัวกลางแจ้งเหตุการณ์ จากนั้นตัวกลางก็จะไปทำการดึงเหตุการณ์จากผู้แจ้งเหตุการณ์มาให้ หากในขณะนั้นมีเหตุการณ์อยู่

รูปที่ 2.4 รูปแบบการจัดส่งเหตุการณ์แบบพูล

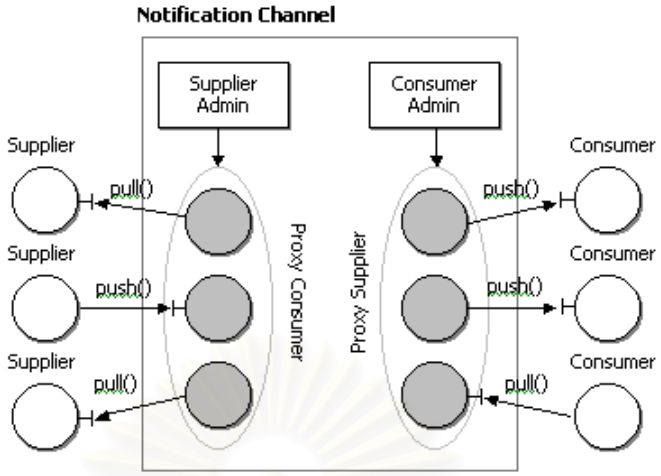
2.2.4 บริการแจ้งเหตุการณ์ (Notification Service) [5]

เป็นบริการพื้นฐานตัวหนึ่งที่กำหนดโดยโอเอ็มจี โดยมีวัตถุประสงค์เพื่อใช้ในการแจ้งเหตุการณ์ต่างๆ ในแบบอะซิงโครนัส (Asynchronous) ในระบบกระจายคอร์บา คุณสมบัติที่สำคัญของบริการแจ้งเตือน คือ

- การแจ้งเหตุการณ์ ที่มีลักษณะข้อมูลทั้งที่เป็นแบบไม่เป็นโครงสร้าง เป็นแบบโครงสร้าง และเป็นแบบชุดของเหตุการณ์แบบโครงสร้าง
- มีความสามารถในการกรองเหตุการณ์ (Event Filtering) โดยการใช้เนื้อหาของเหตุการณ์ (Content) เป็นคีย์ในการกรอง
- รูปแบบการจัดส่งเหตุการณ์เป็นได้ทั้งแบบพุชและพูล
- มีการกำหนดค่าคุณสมบัติหรือค่าคุณภาพของบริการต่างๆ ได้ เช่นการกำหนดรูปแบบการจัดส่งข้อมูล การกำหนดระดับของความน่าเชื่อถือในการจัดส่งข้อมูล เป็นต้น

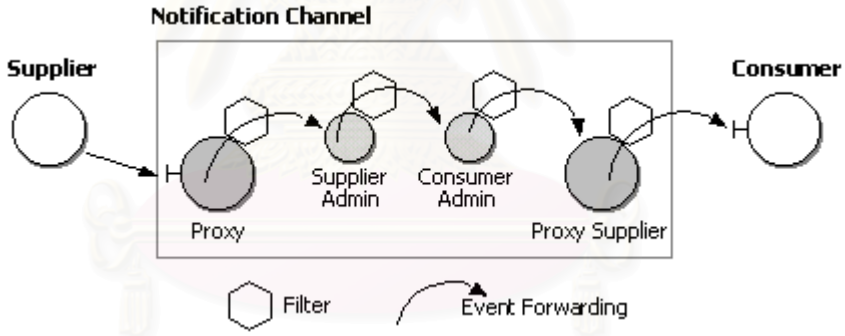
บริการแจ้งเหตุการณ์ประกอบด้วยส่วนที่สำคัญแบ่งออกเป็นฝ่ายผู้แจ้งเหตุการณ์ (Supplier) และฝ่ายผู้ต้องการทราบเหตุการณ์ (Consumer) ดังต่อไปนี้ (รูปที่ 2.5)

1. <u>ฝ่ายผู้แจ้งเหตุการณ์</u> ประกอบด้วย
   1.1 ผู้ดูแลผู้แจ้งเหตุการณ์ (Supplier Admin) ทำหน้าที่สร้าง ลบ และดูแลตัวแทนผู้ต้องการทราบเหตุการณ์ (Proxy Consumer) โดยรับการร้องขอมาจากผู้แจ้งเหตุการณ์
   1.2 ตัวแทนผู้ต้องการทราบเหตุการณ์ (Proxy Consumer) ถูกสร้างขึ้นตามคำสั่งของผู้แจ้งเหตุการณ์สำหรับทำหน้าที่รับเหตุการณ์ต่างๆ แบ่งออกได้เป็น 2 ประเภทตามรูปแบบการจัดส่งเหตุการณ์ คือแบบพุชและพูล
2. <u>ฝ่ายผู้ต้องการทราบเหตุการณ์</u> ประกอบด้วย
   2.1 ผู้ดูแลผู้ต้องการทราบเหตุการณ์ (Consumer Admin) ทำหน้าที่สร้าง ลบ และดูแลตัวแทนผู้แจ้งเหตุการณ์ (Proxy Supplier) โดยรับการร้องขอมาจากผู้ต้องการทราบเหตุการณ์
   2.2 ตัวแทนผู้แจ้งเหตุการณ์ (Proxy Supplier) ถูกสร้างขึ้นตามคำสั่งของผู้ต้องการทราบเหตุการณ์สำหรับทำหน้าที่ในการส่งเหตุการณ์ไปให้แก่ผู้ต้องการทราบเหตุการณ์แบ่งออกได้เป็น 2 ประเภทตามรูปแบบการจัดส่งเหตุการณ์เช่นกันคือ แบบพุชและพูล



รูปที่ 2.5 โครงสร้างของบริการแจ้งเหตุการณ์

ทิศทางของเหตุการณ์ที่ส่งผ่านจากผู้แจ้งเหตุการณ์ผ่านบริการแจ้งเหตุการณ์ไปยังผู้ต้องการทราบเหตุการณ์ในรูปแบบการจัดส่งเหตุการณ์แบบพุชสามารถแสดงได้ดังรูปที่ 2.6 (สำหรับการจัดส่งแบบพูลก็จะมีลักษณะทิศทางที่ตรงกันข้ามกัน)



รูปที่ 2.6 การส่งผ่านเหตุการณ์ภายในบริการแจ้งเหตุการณ์ในรูปแบบการจัดส่งแบบพุช

2.2.5 เอ็กซ์เอ็มแอล (XML) [6]

เอ็กซ์เอ็มแอล (XML: Extensible Markup Language) เป็นภาษาที่ถูกออกแบบเพื่อใช้อธิบายข้อมูลของวัตถุ และพฤติกรรม (Behavior) ของโปรแกรม โดยเป็นภาษาที่มีพื้นฐานจากเอสจีเอ็มแอล (SGML: Standard Generalized Markup Language) แต่ได้ทำการลดทอนความซับซ้อนลง วัตถุประสงค์หลักของเอ็กซ์เอ็มแอล คือ

- เป็นภาษาที่ใช้ง่าย
- สามารถประมวลผลได้ทั้งโดยโปรแกรมประยุกต์และมนุษย์
- สามารถนำไปใช้ในการแลกเปลี่ยนข้อมูลในสภาวะแวดล้อมที่แตกต่างกันได้

เอ็กซ์เอ็มแอลจะประกอบด้วยส่วนประกอบสองส่วนคือ คำอธิบายชนิดของเอกสารที่เรียกว่าดีทีดี (DTD) ที่ทำหน้าที่กำหนดไวยากรณ์ของเอกสาร ส่วนที่สองคือตัวเอกสารเอ็กซ์เอ็มแอลซึ่งเป็นโครงสร้างทางตรรกะ (Logical Structure) โดยอธิบายคุณลักษณะต่างๆ ที่สอดคล้องกับดีทีดีในลักษณะที่เป็นโครงสร้างลำดับชั้น (Hierarchy)

ตัวอย่างเอกสารเอ็กซ์เอ็มแอล แสดงดังรูปที่ 2.7 จากตัวอย่างบรรทัดที่ 1 แสดงถึงการเป็นเอกสารเอ็กซ์เอ็มแอลด้วยรูปแบบ <?xml …. ?> ระบุว่าใช้ตามมาตรฐานรุ่นที่ 1 บรรทัดที่ 2 – 6 เป็นการกำหนดดีทีดีแสดงโครงสร้างของเอกสารเอ็กซ์เอ็มแอล และบรรทัดที่ 7-10 เป็นเนื้อหาของเอกสารเอ็กซ์เอ็มแอล

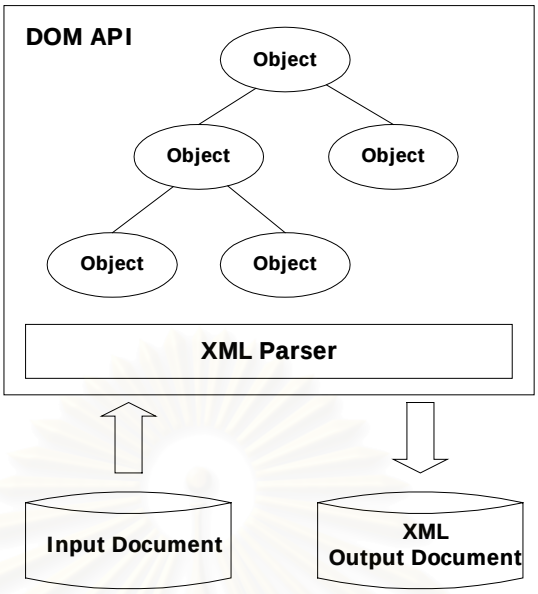
เนื่องจากเอ็กซ์เอ็มแอลเป็นภาษาที่มีคำอธิบายอยู่ในตัวมันเอง มีความยืดหยุ่น ไม่มีโครงสร้างที่ตายตัว การนำเอาเอ็กซ์เอ็มแอลไปใช้งานสำหรับอธิบายข้อมูลต่างๆ จึงมีแนวโน้มที่แพร่หลายมากยิ่งขึ้น โดยเฉพาะอย่างยิ่งการนำไปใช้งานบนเว็บ นอกจากนี้มาตรฐานของเอ็กซ์เอ็มแอลยังถูกกำหนดและพัฒนาจากหลายวงการนอกเหนือจากคอมพิวเตอร์ เช่น คณิตศาสตร์ เคมี และอุตสาหกรรมต่างๆ เป็นต้น

```
1   <?xml version="1.0"?>
2   <!DOCTYPE book [
3   <!ELEMENT book (title, author)>
4   <!ELEMENT title      (#PCDATA)>
5   <!ELEMENT author     (#PCDATA)>
6   ]>
7   <book>
8   <title>Java and XML</title>
9   <author>Brett McLaughlin</author>
10  </book>
```

รูปที่ 2.7 ตัวอย่างเอกสารเอ็กซ์เอ็มแอล

2.2.6 ดีโอเอ็ม (DOM : Document Object Model) [7] [8]

ดีโอเอ็ม คือข้อกำหนดของส่วนต่อประสานในการเขียนโปรแกรมประยุกต์ (Application Programming Interface (API)) เพื่อจัดการข้อมูลเอกสารเอ็กซ์เอ็มแอลให้อยู่ในรูปของวัตถุที่สามารถเขียนชุดคำสั่งเพื่อเข้าถึง และแก้ไขเนื้อความภายในได้ ข้อกำหนดของดีโอเอ็มระดับหนึ่ง (DOM Level 1) ประกอบไปด้วยส่วนต่อประสานสำคัญสองส่วนคือ ส่วนแรกใช้ในการจัดการเอกสารเอ็กซ์เอ็มแอลในรูปของวัตถุ และส่วนที่สองคือส่วนต่อประสานที่ทำงานร่วมกับส่วนแรกเพื่อเพิ่มความสามารถในการจัดการกับแท็ก (Tag) ของเอกสารเอ็กซ์เอ็มแอล ขั้นตอนการสร้างเอกสารเอ็กซ์เอ็มแอลโดยใช้ดีโอเอ็มอาจแสดงได้ดังรูปที่ 2.8



รูปที่ 2.8 การสร้างเอกสารเอ็กซ์เอ็มแอลโดยใช้ดีโอเอ็ม

การจัดการเอกสารโดยใช้ดีโอเอ็มอาจเริ่มจากการอ่านเอกสารเอ็กซ์เอ็มแอลที่มีอยู่แล้วเพื่อนำมาแก้ไข เมื่อมีการอ่านเอกสารเอ็กซ์เอ็มแอลที่มีอยู่แล้วเข้ามา เอกสารนั้นจะผ่านการตรวจสอบไวยากรณ์โดยเอพีไอที่เรียกว่าเอสเอเอ็กซ์ (SAX : Simple API for XML) จากนั้นจึงมีการสร้างวัตถุที่มีเนื้อความภายในเช่นเดียวกับเอกสารขึ้นมา เมื่อผ่านการแก้ไขหรือเพิ่มเติมเนื้อความภายในแล้ว เราสามารถนำเนื้อความภายในวัตถุมาสร้างเอกสารเอ็กซ์เอ็มแอล หรือเราสามารถใช้ดีโอเอ็มนี้ไปใช้ในการสร้างวัตถุเปล่าขึ้นมาเพื่อเตรียมการสร้างเอกสารเอ็กซ์เอ็มแอลได้อีกด้วย การใช้ดีโอเอ็มจะช่วยให้การจัดการ และการเขียนเอกสารเอ็กซ์เอ็มแอลที่อยู่ในรูปวัตถุทำได้ง่ายขึ้น โดยสามารถนำส่วนต่อประสานโปรแกรมประยุกต์เหล่านี้มาพัฒนาสร้างโปรแกรมประยุกต์ให้สามารถนำข้อมูลในรูปของวัตถุมาแสดงผลในรูปของส่วนต่อประสานผู้ใช้แบบกราฟิก (Graphical User Interface (GUI)) หรือนำเสนอผ่านสื่อในลักษณะอื่นได้

ในปัจจุบันได้มีการพัฒนาดีโอเอ็มภายใต้ข้อกำหนดของดับเบิลยูทรีซี (W3C: World Wide Web Consortium) ออกมามากมายเช่น จาวาโปรเจคเอ็กซ์ (Java Project X) จากซันไมโครซิสเต็ม (Sun Microsystems) ออราเคิลเอ็กซ์เอ็มแอลพาร์สเซอร์ (Oracle XML Parser) จาก ออราเคิล (Oracle) และเอ็มเอสไออีไฟว์ดีโอเอ็ม (MSIE5DOM) จากไมโครซอฟท์ (Microsoft) โดยการพัฒนาเหล่านี้ยังได้เพิ่มเติมส่วนจัดการเอกสารที่เป็นประโยชน์ เช่นการสร้างแฟ้มข้อมูลเอกสารเอ็กซ์เอ็มแอล ทำให้สามารถนำการพัฒนาเหล่านี้ไปใช้ในการจัดการเอกสารเอ็กซ์เอ็มแอลได้โดยตรง

บทที่ 3
การออกแบบระบบแจ้งการเปลี่ยนแปลงบริการ

ในบทนี้กล่าวถึงรายละเอียดของการออกแบบระบบแจ้งการเปลี่ยนแปลงบริการ โดยเนื้อหาประกอบด้วยเหตุการณ์การเปลี่ยนแปลงบริการที่พิจารณา โครงสร้างของระบบ การทำงานในส่วนย่อยต่างๆ คุณภาพของบริการ (Quality of Service) ที่ระบบสนับสนุน และโครงสร้างของข้อมูลต่างๆ เช่น ข้อมูลระบุบริการ ข้อมูลการเปลี่ยนแปลงบริการ เป็นต้น

3.1 เหตุการณ์การเปลี่ยนแปลงบริการที่พิจารณา

เหตุการณ์การเปลี่ยนแปลงบริการที่ได้รับการพิจารณาภายในวิทยานิพนธ์นี้มีด้วยกัน 4 ประเภท คือ

เหตุการณ์ที่ 1 บริการได้รับการเปลี่ยนแปลงชนิดของบริการ

จากที่ได้กล่าวมาแล้วข้างต้น บริการจะได้รับการเปลี่ยนแปลงชนิดของบริการก็ต่อเมื่อบริการนั้นได้รับการเปลี่ยนแปลงส่วนต่อประสานของบริการ เปลี่ยนแปลงรายการคุณสมบัติของบริการ และ (หรือ) เปลี่ยนแปลงรายการชนิดของบริการที่สืบทอด

เหตุการณ์ที่ 2 บริการได้รับการเปลี่ยนแปลงค่าคุณสมบัติของบริการ

การเปลี่ยนแปลงค่าคุณสมบัติของบริการอาจไม่ส่งผลกระทบต่อการทำงานของผู้รับบริการหรืออาจทำให้ผู้รับบริการทำงานต่อไม่ได้ก็เป็นได้ ทั้งนี้ขึ้นอยู่กับวิธีการค้นหาและการใช้งานบริการของผู้รับบริการแต่ละราย

เหตุการณ์ที่ 3 บริการถูกยกเลิกการใช้งาน

การยกเลิกการใช้งานบริการส่งผลกระทบโดยตรงต่อรับบริการอย่างหลีกเลี่ยงไม่ได้ การแจ้งเตือนการยกเลิกบริการ โดยเฉพาะการแจ้งเตือนล่วงหน้าจะทำให้ผู้รับบริการสามารถที่จะเตรียมการรองรับได้ทัน

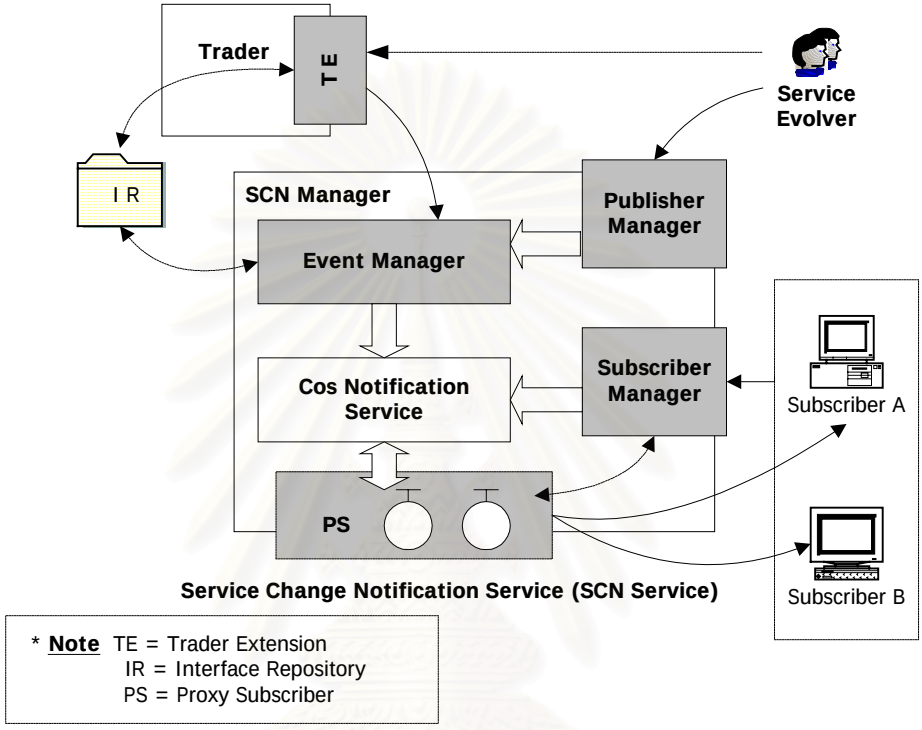
เหตุการณ์ที่ 4 เกิดบริการใหม่ขึ้นในระบบ

การเกิดบริการใหม่ขึ้นในระบบไม่ส่งผลกระทบต่อการใช้งานบริการเดิมแต่อย่างไร แต่เป็นการแจ้งเพื่อให้ผู้รับบริการทราบถึงทางเลือกใหม่ที่เพิ่มขึ้นของการให้บริการ

3.2 โครงสร้างของระบบ

ระบบแจ้งการเปลี่ยนแปลงบริการในวิทยานิพนธ์นี้ได้รับการออกแบบให้เป็นบริการตัวหนึ่งบนระบบกระจายคอร์บา ซึ่งเรียกว่า บริการเอสซีเอ็น (SCN Service: Service Change Notification Service) โดยบริการเอสซีเอ็นนี้จะรองรับการแจ้งการเปลี่ยนแปลงบริการทั้งการแจ้ง

การเปลี่ยนแปลงล่วงหน้า และการแจ้งการเปลี่ยนแปลงในขณะที่บริการได้รับการเปลี่ยนแปลง บริการเอสซีเอ็นสนับสนุนการแจ้งการเปลี่ยนแปลงบริการในรูปแบบที่แตกต่างกันตามความต้องการของผู้ต้องการรับทราบการเปลี่ยนแปลง ทั้งรูปแบบของการจัดส่งและชนิดของข้อมูล โครงสร้างของบริการเอสซีเอ็นแสดงได้ดังรูปที่ 3.1



รูปที่ 3.1 โครงสร้างของบริการแจ้งการเปลี่ยนแปลงบริการ

จากรูปที่ 3.1 บริการเอสซีเอ็นประกอบด้วย 2 ส่วนที่สำคัญคือ ส่วนเพิ่มขยายบริการเทรดเดอร์ (Trader Extension) และตัวจัดการการแจ้งการเปลี่ยนแปลงบริการ (SCN Manager: Service Change Notification Manager) โดยส่วนเพิ่มขยายบริการเทรดเดอร์ทำหน้าที่หลักในการตรวจสอบการเปลี่ยนแปลงข้อมูลข้อเสนอบริการ (Service Offer) ซึ่งกระทำโดยผู้เปลี่ยนแปลงบริการ (Service Evolver) ณ บริการเทรดเดอร์ ตัวจัดการการแจ้งการเปลี่ยนแปลงบริการเป็นส่วนหลักของระบบที่ทำหน้าที่รับข้อมูลแจ้งการเปลี่ยนแปลงบริการจากส่วนเพิ่มขยายบริการเทรดเดอร์ในขณะที่บริการในบริการเทรดเดอร์ได้รับการเปลี่ยนแปลง และรับข้อมูลจากผู้เปลี่ยนแปลงบริการโดยตรงสำหรับข้อมูลแจ้งการเปลี่ยนแปลงบริการล่วงหน้า นอกจากนี้ยังทำหน้าที่รับข้อมูลความต้องการ (Subscription Information) จากผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ (Subscriber) และส่งข้อมูลแจ้งการเปลี่ยนแปลงบริการกลับไปยังผู้ที่ต้องการรับทราบการเปลี่ยน

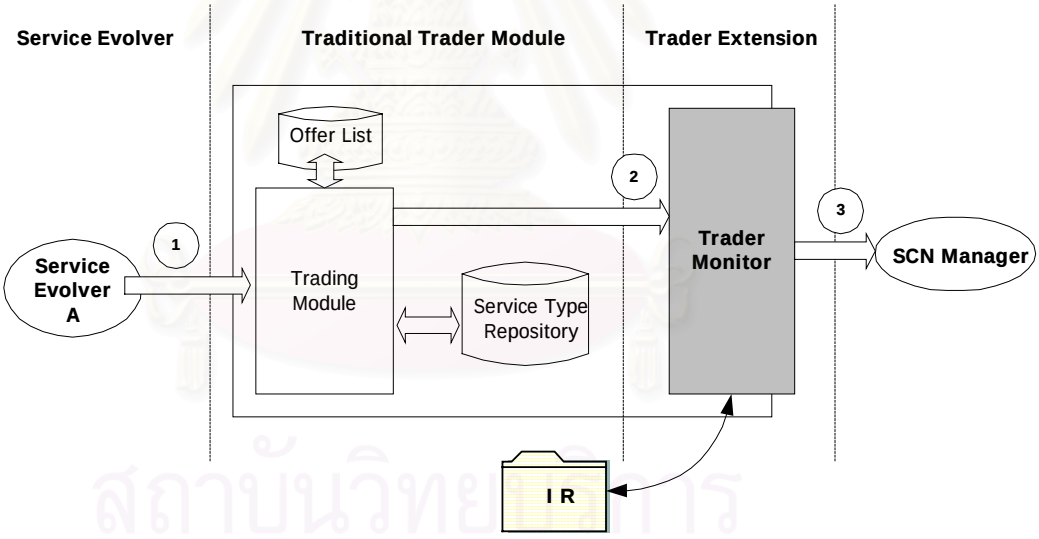
แปลงเมื่อเกิดการเปลี่ยนแปลง ทั้งนี้ข้อมูลการแจ้งจะอยู่ในรูปแบบที่แตกต่างกันได้ขึ้นอยู่กับความต้องการของผู้ที่ต้องการรับทราบการเปลี่ยนแปลงแต่ละราย

สำหรับบริการแจ้งเหตุการณ์ (Notification Service) ซึ่งเป็นบริการพื้นฐานในระบบกระจายคอร์บาได้รับการนำเข้ามาใช้ภายในวิทยานิพนธ์นี้ เพื่อทำหน้านี้สำคัญในการคัดเลือกเหตุการณ์การเปลี่ยนแปลงบริการว่าตรงกับความต้องการของผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการรายใดบ้าง ทำการดูแลในส่วนของการจัดลำดับเหตุการณ์การเปลี่ยนแปลงบริการที่จะทำการจัดส่ง (Order Policy) รวมทั้งการคัดเหตุการณ์การเปลี่ยนแปลงบริการทิ้ง (Discard Policy)ในกรณีที่จำนวนเหตุการณ์มีจำนวนเกินกว่าขนาดของแถวคอยที่ระบุไว้

3.2.1 ส่วนเพิ่มขยายบริการเทรดเดอร์ (Trader Extension)

ส่วนเพิ่มขยายบริการเทรดเดอร์ทำหน้าที่หลักในการตรวจสอบการเปลี่ยนแปลงของบริการโดยอัตโนมัติในขณะที่เกิดการเปลี่ยนแปลงบริการ โดยแบ่งได้เป็น 2 ส่วนด้วยกัน คือ

1. ตัวตรวจบริการเทรดเดอร์ (Trader Monitor)



รูปที่ 3.2 โครงสร้างการทำงานของตัวตรวจบริการเทรดเดอร์

ตัวตรวจบริการเทรดเดอร์ทำงานร่วมกับบริการเทรดเดอร์ในลักษณะของคอร์บาอินเทอร์เซ็พเตอร์เพื่อใช้ในการตรวจสอบการใช้งานตัวกระทำการต่างๆ ที่เกี่ยวข้องกับการเปลี่ยนแปลงข้อมูลของบริการ ผลที่ได้จากการตรวจสอบคือข้อมูลการเปลี่ยนแปลงบริการในขณะที่บริการได้รับการเปลี่ยนแปลง

จากรูปที่ 3.2 การทำงานของตัวตรวจบริการเทรดเดอร์เริ่มต้นจากผู้เปลี่ยนแปลงบริการเรียกใช้งานตัวกระทำการต่างๆ ที่เกี่ยวข้องกับการเปลี่ยนแปลงข้อมูลของบริการบนบริการเทรดเดอร์ ในขั้นตอนที่ 2 ข้อมูลการเรียกใช้งานตัวกระทำการต่างๆ เหล่านั้นจะถูกตรวจจับโดยตัวตรวจบริการเทรดเดอร์ซึ่งคอยเฝ้ามองการเรียกใช้งานตัวกระทำการเหล่านั้นอยู่ ในขั้นตอนที่ 3 ตัวตรวจบริการเทรดเดอร์ทำการส่งข้อมูลการเปลี่ยนแปลงบริการที่ตรวจจับได้ไปยังตัวจัดการการแจ้งการเปลี่ยนแปลงบริการเพื่อทำการแจ้งไปยังผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการต่อไป

ตัวตรวจบริการเทรดเดอร์จะทำการตรวจจับการเรียกใช้งานตัวกระทำการของบริการเทรดเดอร์ (ตามข้อกำหนดของ [2]) เฉพาะที่เกี่ยวข้องกับการเปลี่ยนแปลงข้อมูลของบริการเท่านั้น ตัวกระทำการเหล่านั้นได้แก่

1. <u>ตัวกระทำการเพิ่มบริการ</u> มีส่วนต่อประสานดังนี้

```
OfferId export {
   in Object reference,
   in ServiceTypeName type,
   in PropertySeq properites
};
```

2. <u>ตัวกระทำการยกเลิกบริการ</u> มีส่วนต่อประสานดังนี้

```
void withdraw {
   in OfferId id
};
```

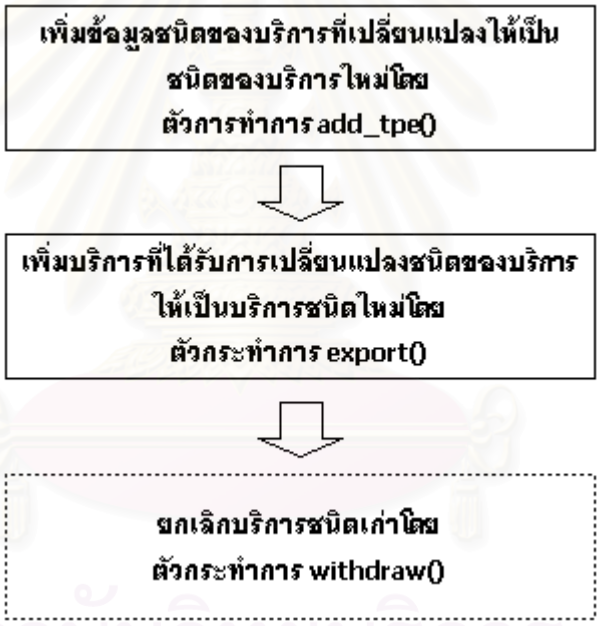
3. <u>ตัวกระทำการเปลี่ยนแปลงค่าคุณสมบัติของบริการ</u> มีส่วนต่อประสานดังนี้

```
void modify {
   in OfferId id,
   in PropertyNameSeq del_list,
   in PropertySeq modify_list
};
```

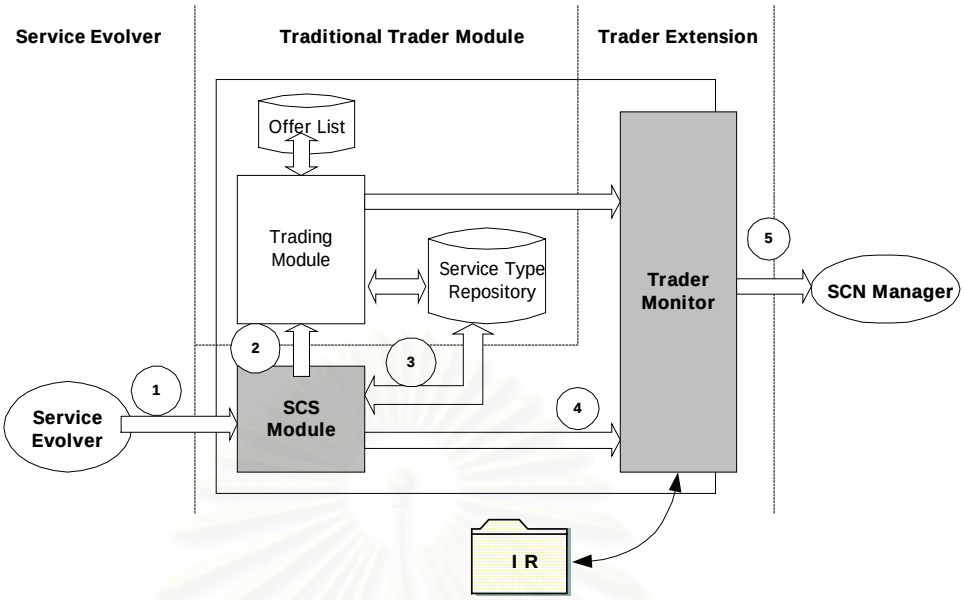
2. <u>มอดูลสนับสนุนการเปลี่ยนแปลงบริการ</u> (Service Change Supporting Module)

ตัวตรวจบริการเทรดเดอร์สามารถตรวจจับเหตุการณ์การเปลี่ยนแปลงบริการในหัวข้อ 3.1 ได้จากการดักการเรียกใช้ตัวกระทำการข้างต้นของบริการเทรดเดอร์ แต่จะไม่สามารถตรวจจับเหตุการณ์การเปลี่ยนแปลงชนิดของบริการได้ ทั้งนี้เนื่องจากการเปลี่ยนแปลงข้อมูลชนิดของบริการให้กับบริการหนึ่งๆ นั้นไม่สามารถกระทำได้ด้วยตัวกระทำการเดียวโดยตรง แต่จะประกอบด้วยชุดของตัวกระทำการของบริการเทรดเดอร์ดังแสดงในรูปที่ 3.3 ดังนั้นตัวตรวจบริการเทรดเดอร์จึงไม่สามารถแยกแยะได้ว่าการกระทำต่างๆ ดังกล่าวเป็นขั้นตอนของการเปลี่ยนชนิดของบริการให้กับบริการหนึ่งๆ หรือเป็นการเพิ่มและยกเลิกบริการที่ไม่เกี่ยวข้องกัน และเพื่อให้ตัวตรวจบริการเทรดเดอร์สามารถแยกแยะได้ ในวิทยานิพนธ์นี้จึงทำการเพิ่มมอดูลสนับสนุนการเปลี่ยนแปลงบริการเข้าไปในบริการเทรดเดอร์

มอดูลสนับสนุนการเปลี่ยนแปลงบริการมีวัตถุประสงค์หลักเพื่อเพิ่มตัวกระทำการสำหรับการเปลี่ยนชนิดของบริการ ซึ่งจะทำให้ตัวตรวจบริการเทรดเดอร์สามารถตรวจจับการเปลี่ยนแปลงชนิดของบริการหนึ่งๆ ได้จากการดักการเรียกใช้ตัวกระทำการที่เพิ่มขึ้นนี้ การทำงานของตัวกระทำการเปลี่ยนชนิดของบริการสามารถแสดงได้ดังรูปที่ 3.4 โดยขั้นที่ 1 ผู้เปลี่ยนแปลงบริการจะทำการเรียกใช้งานตัวกระทำการเพื่อขอเปลี่ยนแปลงชนิดของบริการให้กับบริการหนึ่งๆ ขั้นตอนที่ 2 และ 3 เป็นการทำงานภายในของตัวกระทำการที่มีการติดต่อขอเรียกใช้งานตัวกระทำการในมอดูลมาตรฐานของบริการเทรดเดอร์ ในขั้นตอนที่ 4 ตัวตรวจบริการเทรดเดอร์จะตรวจพบการเรียกใช้งานตัวกระทำการ จากนั้นในขั้นตอนที่ 5 ตัวตรวจบริการเทรดเดอร์จะทำการส่งข้อมูลการเปลี่ยนแปลงบริการที่ตรวจสอบได้ไปยังตัวจัดการการแจ้งการเปลี่ยนแปลงบริการเพื่อทำการแจ้งไปยังผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการต่อไป



รูปที่ 3.3 ขั้นตอนการเปลี่ยนแปลงชนิดของบริการให้กับบริการหนึ่งๆ



รูปที่ 3.4 โครงสร้างการทำงานของส่วนมอดูลสนับสนุนการเปลี่ยนแปลงบริการ

นอกจากตัวกระทำการสำหรับเปลี่ยนแปลงชนิดของบริการแล้ว มอดูลสนับสนุนการเปลี่ยนแปลงบริการยังประกอบด้วยส่วนต่อประสานสำหรับผู้เปลี่ยนแปลงบริการ (ผู้พัฒนาบริการ) และผู้ต้องการรับทราบการเปลี่ยนแปลงบริการในการเริ่มต้นเข้าใช้งานบริการเอสซีเอ็นโดยทั้งนี้มีจุดประสงค์เพียงเพื่อความสะดวกของผู้ใช้งานเท่านั้น

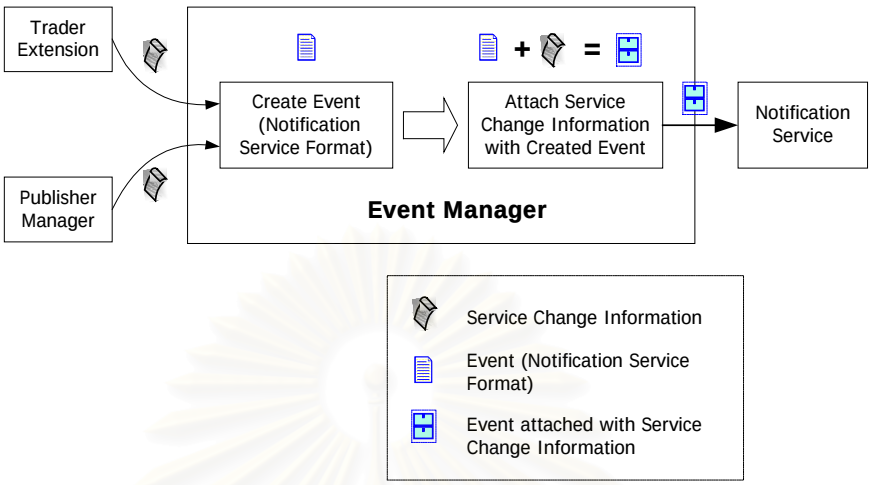
3.2.2 ตัวจัดการการแจ้งการเปลี่ยนแปลงบริการ (SCN Manager: Service Change Notification Manager: SCN Manager)

หน้าที่หลัก คือ รับข้อมูลความต้องการจากผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ (Subscriber) และส่งข้อมูลแจ้งการเปลี่ยนแปลงบริการกลับไปยังผู้ที่ต้องการรับทราบการเปลี่ยนแปลงเมื่อเกิดการเปลี่ยนแปลงบริการในรูปแบบต่างๆ ขึ้นอยู่กับความต้องการของผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการแต่ละราย โครงสร้างภายในของตัวจัดการการแจ้งการเปลี่ยนแปลงประกอบด้วย 5 ส่วนหลักๆ ซึ่งมีหน้าที่ที่แตกต่างกันดังนี้ คือ

1. <u>ตัวจัดการผู้แจ้งการเปลี่ยนแปลง</u> (Publisher Manager)

ทำหน้าที่รับการลงทะเบียนและรับข้อมูลแจ้งการเปลี่ยนแปลงบริการล่วงหน้าจากผู้เปลี่ยนแปลงบริการ ข้อมูลที่ได้รับมาจะถูกส่งต่อไปยังตัวจัดการเหตุการณ์

2. ตัวจัดการเหตุการณ์ (Event Manager)



รูปที่ 3.5 โครงสร้างการทำงานของตัวจัดการเหตุการณ์

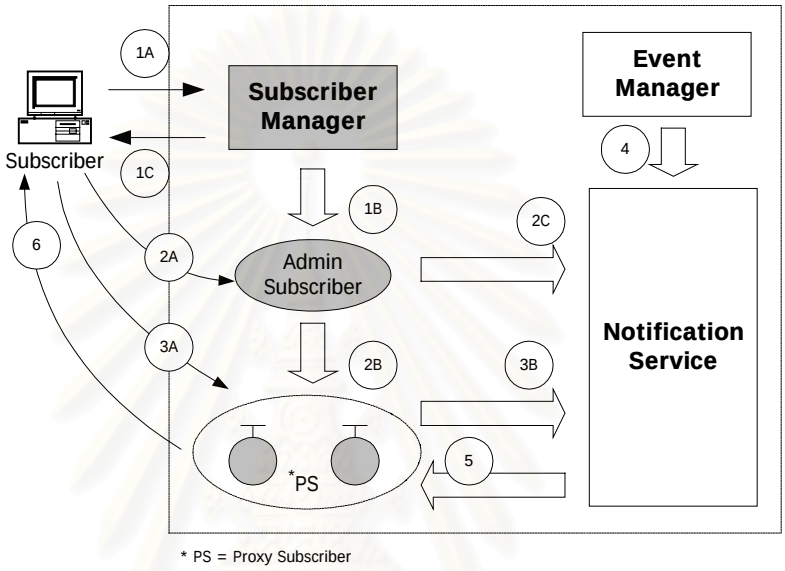
จากรูปที่ 3.5 ตัวจัดการเหตุการณ์จะทำหน้าที่รับข้อมูลการเปลี่ยนแปลงบริการจากทั้งตัวจัดการผู้แจ้งการเปลี่ยนแปลงสำหรับข้อมูลแจ้งการเปลี่ยนแปลงล่วงหน้า และรับข้อมูลจากตัวตรวจบริการเทรดเดอร์สำหรับข้อมูลแจ้งการเปลี่ยนแปลงในขณะที่บริการได้รับการเปลี่ยนแปลง จากนั้นจึงนำข้อมูลที่ได้รับมาสร้างเป็นเหตุการณ์ในรูปแบบที่บริการแจ้งเหตุการณ์กำหนดไว้ พร้อมกับแนบท้ายด้วยข้อมูลการเปลี่ยนแปลงบริการที่ได้รับ จากนั้นจึงนำเหตุการณ์ดังกล่าวส่งต่อไปยังบริการแจ้งเหตุการณ์เพื่อทำการจัดส่งข้อมูลการเปลี่ยนแปลงบริการ ให้ตรงกับความต้องการของผู้ที่ต้องการทราบการเปลี่ยนแปลงบริการต่อไป

3. บริการแจ้งเหตุการณ์ (Notification Service)

บริการแจ้งเหตุการณ์เป็นบริการพื้นฐานในระบบกระจายคอร์บา ซึ่งภายในวิทยานิพนธ์นี้นำเข้ามาใช้งาน จากรูปที่ 3.1 บริการแจ้งเหตุการณ์ได้รับข้อมูลเหตุการณ์การเปลี่ยนแปลงบริการมาจากตัวจัดการเหตุการณ์ บริการแจ้งเหตุการณ์จะทำการตรวจสอบเหตุการณ์ที่ได้รับมาว่าตรงกับความต้องการของผู้ใดบ้าง (Filtering) จากนั้นจึงส่งเหตุการณ์ดังกล่าวไปให้แก่ผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการนั้นๆ โดยผ่านตัวแทนผู้ที่ต้องการทราบการเปลี่ยนแปลงบริการ (Proxy Subscriber) นอกจากนี้บริการแจ้งเหตุการณ์ยังดูแลในส่วนของรูปแบบการจัดส่ง (แบบพุช และ พูล) การจัดลำดับการจัดส่ง (Order Policy) การคัดทิ้ง (Discard Policy) รวมทั้งการจัดเก็บเหตุการณ์การเปลี่ยนแปลงในกรณีที่ผู้ต้องการรับทราบการเปลี่ยนแปลงไม่พร้อมที่จะรับข้อมูลแจ้งการเปลี่ยนแปลงบริการ

4. <u>ตัวจัดการผู้ที่ต้องการทราบการเปลี่ยนแปลงบริการ</u> (Subscriber Manager)

ทำหน้าที่รับการลงทะเบียนและดูแลข้อมูลของผู้ที่ต้องการทราบการเปลี่ยนแปลงบริการ รูปที่ 3.6 แสดงขั้นตอนการเข้าใช้งานตัวจัดการผู้ที่ต้องการทราบการเปลี่ยนแปลงบริการ ซึ่งสามารถสรุปเป็นขั้นตอนได้ดังต่อไปนี้



รูปที่ 3.6 การใช้งานของผู้ที่ต้องการทราบการเปลี่ยนแปลงบริการ

<u>ขั้นตอนที่ 1</u> ผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการทำการลงทะเบียนเพื่อขอเข้าใช้งานระบบ (1A) เมื่อผ่านการตรวจสอบความถูกต้องของข้อมูลเรียบร้อยแล้ว ตัวจัดการผู้ที่ต้องการทราบการเปลี่ยนแปลงบริการจะทำการสร้างวัตถุผู้ดูแล (Admin Subscriber) ขึ้น (1B) ข้อมูลอ้างถึงวัตถุ (Object Reference) ของวัตถุผู้ดูแลจะถูกส่งให้กับผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ (1C)

<u>ขั้นตอนที่ 2</u> ผู้ใช้งานติดต่อกับวัตถุผู้ดูแลเพื่อสร้างตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ (Proxy Subscriber) ขึ้น (2A) ตัวแทนจะถูกสร้างขึ้น (2B) และได้รับการลงทะเบียนไว้กับบริการแจ้งเหตุการณ์ (2C) เพื่อขอรับเหตุการณ์การเปลี่ยนแปลงบริการ (รายละเอียดของตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการจะกล่าวในส่วนถัดไป)

<u>ขั้นตอนที่ 3</u> ผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการทำการกำหนดหรือแก้ไขรายการบริการที่ตนต้องการรับทราบการเปลี่ยนแปลง พร้อมทั้งกำหนดค่าคุณสมบัติต่างๆ

(Subscriber QoS Properties) (3A) ซึ่งรายละเอียดของคุณสมบัติเหล่านี้จะกล่าวในหัวข้อ 3.3เมื่อตัวแทนได้รับข้อมูลก็จะไปทำการบันทึกข้อมูลการกรอง (Filter) และข้อมูลคุณสมบัติต่างๆ ที่บริการแจ้งเหตุการณ์ (3B)

ขั้นตอนที่ 4-6 บริการแจ้งเหตุการณ์ได้รับเหตุการณ์การเปลี่ยนแปลงบริการจากตัวจัดการเหตุการณ์ (4) จากนั้นบริการแจ้งเหตุการณ์ตรวจสอบว่าเหตุการณ์การเปลี่ยนแปลงบริการที่ได้รับมาตรงกับความต้องการของผู้ที่ต้องการรับทราบการเปลี่ยนแปลงรายใดบ้าง การจัดส่งจะกระทำผ่านตัวแทนผู้ที่ต้องการทราบการเปลี่ยนแปลงบริการ (5) โดยรูปแบบข้อมูลที่จัดส่งจะเป็นไปตามชนิดของตัวแทนที่ถูกสร้างขึ้นมา (6)

5. ตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ (PS: Proxy Subscriber)

เป็นส่วนที่สร้างขึ้นโดยคำสั่งของผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ เพื่อทำหน้าที่หลักในการส่งผ่านข้อมูลแจ้งการเปลี่ยนแปลงบริการที่ได้รับมาจากบริการแจ้งเหตุการณ์ไปยังวัตถุเรียกกลับ (Callback Object) ของผู้ที่ต้องการรับทราบการเปลี่ยนแปลง ตามรูปแบบการจัดส่งข้อมูลและรูปแบบของข้อมูลซึ่งขึ้นอยู่กับประเภทของตัวแทนที่ได้รับการสร้างขึ้น

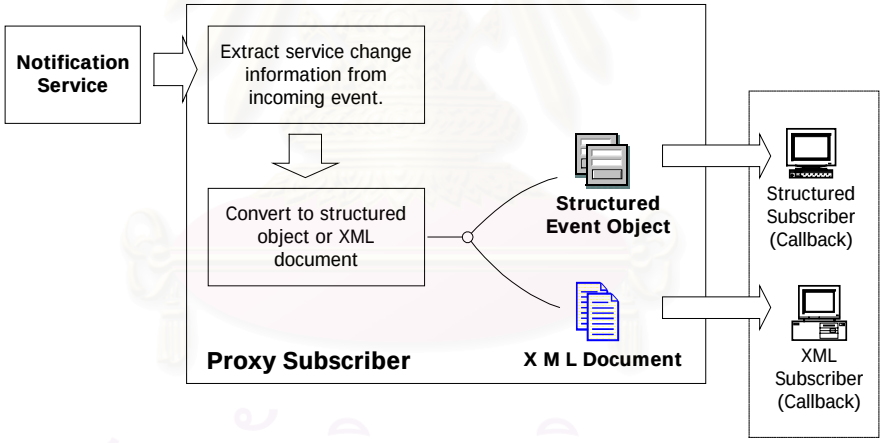
รูปแบบการจัดส่งข้อมูลที่จะสนับสนุน ได้แก่ การจัดส่งแบบพุชและพูล ส่วนรูปแบบของข้อมูลประกอบด้วย วัตถุเหตุการณ์แบบโครงสร้าง ชุดของวัตถุเหตุการณ์แบบโครงสร้าง และเอกสารเอ็กซ์เอ็มแอล ในส่วนของรูปแบบข้อมูลที่เป็นเอกสารเอ็กซ์เอ็มแอลนั้นมีวัตถุประสงค์เพื่อรองรับผู้ใช้งานบริการที่อยู่ในสภาวะแวดล้อมที่ต่างไปจากระบบกระจายคอร์บา รวมทั้งการส่งผ่านข้อมูลการเปลี่ยนแปลงบริการไปให้แก่ระบบอินเทอร์เน็ตด้วย

วิทยานิพนธ์นี้ได้กำหนดให้มีตัวแทนทั้งหมด 6 ประเภท คือ

1. StructuredPushProxy ตัวแทนจัดส่งข้อมูลแบบพุช โดยข้อมูลที่ทำการจัดส่งอยู่ในรูปแบบวัตถุเหตุการณ์แบบโครงสร้าง (Structured Event Object)
2. XMLPushProxy ตัวแทนจัดส่งข้อมูลแบบพุช โดยข้อมูลที่ทำการจัดส่งอยู่ในรูปแบบของเอกสารเอ็กซ์เอ็มแอล
3. SequenceStructuredPushProxy ตัวแทนจัดส่งข้อมูลแบบพุช โดยข้อมูลที่ทำการจัดส่งอยู่ในรูปแบบของชุดของวัตถุเหตุการณ์แบบโครงสร้าง (Sequence of Structured Event Object)
4. StructuredPullProxy ตัวแทนจัดส่งข้อมูลแบบพูล โดยข้อมูลที่ทำการจัดส่งอยู่ในรูปแบบของวัตถุเหตุการณ์แบบโครงสร้าง

5. XMLPullProxy ตัวแทนจัดส่งข้อมูลแบบพูล โดยข้อมูลที่ทำการจัดส่งอยู่ในรูปแบบของเอกสารเอ็กซ์เอ็มแอล
6. SequenceStructuredPullProxy ตัวแทนจัดส่งข้อมูลแบบพูล โดยข้อมูลที่ทำการจัดส่งอยู่ในรูปแบบของชุดของวัตถุเหตุการณ์แบบโครงสร้าง

รูปที่ 3.7 เป็นการทำงานของตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงที่มีการจัดส่งข้อมูลการเปลี่ยนแปลงบริการแบบพุช (StructuredPushProxy, XMLPushProxyและ SequenceStructuredPushProxy) โดยการทำงานเริ่มต้นจากบริการแจ้งเหตุการณ์ทำการส่งเหตุการณ์การเปลี่ยนแปลงบริการให้กับตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ จากนั้นตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงจะทำการแยกเอาแต่ข้อมูลการเปลี่ยนแปลงออกมาจากเหตุการณ์การเปลี่ยนแปลงบริการ ข้อมูลการเปลี่ยนแปลงบริการจะถูกจัดให้อยู่ในรูปแบบของวัตถุเหตุการร์แบบโครงสร้าง หรือ เอกสารเอ็กซ์เอ็มแอล ขึ้นอยู่กับประเภทของตัวแทน จากนั้นทำการส่งให้กับวัตถุเรียกกลับ (Callback) ของผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการต่อไป



รูปที่ 3.7 การทำงานของตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการแบบพุช

สำหรับการทำงานของตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงที่มีการจัดส่งข้อมูลการเปลี่ยนแปลงบริการแบบพูล (StructuredPullProxy, XMLPullProxy และ SequenceStructuredPullProxy) ก็จะมีลักษณะที่คล้ายคลึงกันต่างกันแต่เพียงการทำงานจะเริ่มต้นจากผู้ที่ต้องการรับทราบการเปลี่ยนแปลงต้องมาทำการร้องขอเพื่อรับทราบข้อมูลการเปลี่ยนแปลงบริการจากตัวแทน จากนั้นตัวแทนก็จะไปทำการดึงเหตุการณ์การเปลี่ยนแปลงบริการจากบริการแจ้งเหตุการณ์ โดยเหตุการณ์ที่ดึงมาได้ก็จะได้รับการจัดการในลักษณะเดียวกันกับการทำงานโดยตัวแทนแบบพุช

วิทยานิพนธ์นี้กำหนดให้ผู้ที่ต้องการรับทราบการเปลี่ยนแปลงแต่ละรายสามารถสร้างตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงได้ประเภทละหนึ่งตัวเท่านั้นเพื่อใช้ในจัดรูปแบบข้อมูลการเปลี่ยนแปลงบริการให้ตรงกับความต้องการ

3.3 คุณสมบัติด้านการบริการและคุณภาพของบริการ

คุณสมบัติต่างๆ ที่จะกล่าวถึงต่อไปนี้ เป็นคุณสมบัติที่กำหนดขึ้นตามมาตรฐานของบริการแจ้งการเปลี่ยนแปลง **ผิดพลาด! ไม่พบแหล่งการอ้างอิง** ประกอบด้วย

3.3.1 คุณสมบัติด้านการบริการ (Administrative Properties)

เป็นคุณสมบัติที่กำหนดค่าโดยผู้ดูแลบริการเอสซีเอ็น ได้แก่

- MaxSubscribers คือ จำนวนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการสูงสุดที่สามารถลงทะเบียนขอเข้าใช้งานบริการเอสซีเอ็นได้ ณ เวลาหนึ่ง
- MaxConnectedSubscribers คือ จำนวนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการสูงสุดที่สามารถเชื่อมต่อกับบริการเอสซีเอ็นเพื่อใช้บริการ ณ เวลาหนึ่ง
- MaxEventsPerProxy คือ จำนวนเหตุการณ์รวมสูงสุดที่ตัวแทนของผู้ที่ต้องการทราบการเปลี่ยนแปลงบริการแต่ละรายสามารถดูแลได้ ณ เวลาหนึ่ง

3.3.2 คุณภาพของบริการสำหรับผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ (Subscriber QoS Properties)

เป็นคุณสมบัติที่กำหนดค่าโดยผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ ประกอบด้วยคุณสมบัติต่างๆ ดังต่อไปนี้เป็นอย่างน้อย

- OrderPolicy กำหนดลำดับการจัดส่งเหตุการณ์แจ้งการเปลี่ยนแปลงบริการในกรณีที่มีเหตุการณ์อยู่ภายในแถวคอย (Queue) ของตัวแทนมากกว่า 1 เหตุการณ์ โดยค่าที่สามารถกำหนดได้ประกอบด้วย FIFO, LIFO, Priority และ ExpiryTime
- DiscardPolicy กำหนดลำดับการคัดทิ้งเหตุการณ์แจ้งการเปลี่ยนแปลงบริการหากมีเหตุการณ์เข้ามายังแถวคอยของตัวแทนในขณะที่จำนวนเหตุการณ์ภายในแถวคอยมีค่าเท่ากับค่าของ MaxEventsPerProxy แล้ว ค่าที่สามารถกำหนดได้ประกอบด้วย FIFO, LIFO, Priority และ ExpiryTime
- MaximumBatchSize กำหนดจำนวนเหตุการณ์ที่ต้องการให้เก็บรวบรวมเพื่อจัดส่งในคราวเดียวในกรณีที่เลือกใช้ตัวแทนประเภท SequenceStructuredPushProxy หรือ SequenceStructuredPullProxy

- PacingInterval กำหนดช่วงเวลารอในการเก็บรวบรวมเหตุการณ์เพื่อจัดส่งในคราวเดียว โดยจะมีความสัมพันธ์กับค่า MaximumBatchSize กล่าวคือ หากครบกำหนดเวลารอในการเก็บรวบรวมตามค่าของ PacingInterval แล้ว แต่จำนวนเหตุการณ์ที่รวบรวมได้ยังน้อยกว่าค่า MaximumBatchSize เหตุการณ์ที่เก็บรวบรวมได้เหล่านั้นก็จะได้รับการจัดส่งไปในทันที

3.3.3 คุณภาพของบริการสำหรับเหตุการณ์การเปลี่ยนแปลง (Event QoS Properties)

เป็นคุณสมบัติที่กำหนดค่าโดยผู้แจ้งการเปลี่ยนแปลงบริการ เพื่อระบุลักษณะของข้อมูลแจ้งการเปลี่ยนแปลง โดยลักษณะที่ระบุจะมีผลต่อลำดับการจัดส่งเหตุการณ์และการคัดทิ้งเหตุการณ์ออกจากแถวคอยของตัวแทน ประกอบด้วยคุณสมบัติดังต่อไปนี้

- Priority กำหนดระดับความสำคัญให้กับเหตุการณ์การเปลี่ยนแปลงบริการ
- ExpiryTime กำหนดอายุของเหตุการณ์แจ้งการเปลี่ยนแปลงบริการ

3.4 ข้อมูลระบุความต้องการ ข้อมูลแจ้งการเปลี่ยนแปลงบริการ และข้อมูลการเปลี่ยนแปลงบริการ

ในส่วนนี้กล่าวถึงรายละเอียดของข้อมูลต่างๆ ที่ใช้ในบริการเอสซีเอ็น

3.4.1 ข้อมูลระบุความต้องการ (Subscription Information)

คือข้อมูลที่ผู้ต้องการรับทราบการเปลี่ยนแปลงบริการใช้ในการระบุความต้องการในการรับทราบการเปลี่ยนแปลง สามารถแบ่งออกได้เป็น 2 ส่วน คือ

1. ชนิดของเหตุการณ์การเปลี่ยนแปลงบริการ

ผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการสามารถเลือกที่จะระบุค่าได้ 3 แบบ คือ “เหตุการณ์การเปลี่ยนแปลงบริการ” (เหตุการณ์ที่ 1-3 ในหัวข้อ 3.1) “เหตุการณ์การเกิดบริการใหม่ขึ้นในระบบ” (เหตุการณ์ที่ 4 ในหัวข้อ 3.1) หรือ “เหตุการณ์การเกิดบริการใหม่ขึ้นในระบบและเหตุการณ์การเปลี่ยนแปลงบริการ”

2. บริการ

ผู้ที่ต้องการรับทราบการเปลี่ยนแปลงสามารถเลือกที่จะระบุบริการที่ต้องการให้แจ้งการเปลี่ยนแปลงได้ 2 แบบคือ ทำการระบุด้วยเงื่อนไขค่าคุณสมบัติของบริการซึ่งมีลักษณะเช่นเดียวกับการระบุบริการโดยใช้เงื่อนไขเมื่อทำการสืบค้นบริการจากบริการเทรดเดอร์โดยการใช้ภาษาบังคับ (Constraint Language) โดยภายในงานวิจัยนี้ใช้ภาษาที่ขยายจากที่ใช้ในบริการเทรดเดอร์ (Extended Trader Constraint) [5] ซึ่งถูกใช้เป็นภาษาบังคับสำหรับบริการแจ้งเหตุการณ์ นอกจากนี้การระบุบริการสามารถทำโดยการระบุข้อมูลอ้างถึงวัตถุของบริการที่ต้องการได้

3.4.2 ข้อมูลแจ้งการเปลี่ยนแปลงบริการ (Published Change Information)

คือข้อมูลที่ผู้เปลี่ยนแปลงบริการส่งให้กับบริการเอสซีเอ็นเพื่อใช้ในการแจ้งการเปลี่ยนแปลงบริการล่วงหน้า โครงสร้างของข้อมูลนี้มีลักษณะคล้ายคลึงกับโครงสร้างของข้อมูลการเปลี่ยนแปลงบริการที่จะกล่าวในหัวข้อที่ 3.3.3 โดยมีความแตกต่างกันเพียงส่วนหัวของข้อมูล (Message Header) ซึ่งจะมีเพียงประเภทของข้อมูล (Message Type) ค่าความสำคัญ (Priority) และ วัน-เวลาหมดอายุ (Expiry Time) เท่านั้น (สำหรับรายละเอียดของข้อมูลจะกล่าวในหัวข้อที่ 3.4.3)

3.4.3 ข้อมูลการเปลี่ยนแปลงบริการ (Service Change Information)

คือข้อมูลที่บริการเอสซีเอ็นส่งให้กับผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการเพื่อทำการแจ้งการเปลี่ยนแปลงบริการ รูปที่ 3.8 แสดงโครงสร้างของข้อมูลการเปลี่ยนแปลงบริการ โดยข้อมูลสามารถแบ่งออกได้เป็น 3 ส่วนคือ ส่วนหัวของข้อมูล (Message Header) ส่วนเนื้อความ (Message Body) และส่วนข้อมูลเพิ่มเติม (Additional Data) สำหรับรายละเอียดของแต่ละส่วนสามารถอธิบายได้ดังนี้คือ

1. <u>ส่วนหัวของข้อมูล</u> ประกอบด้วย รหัสข้อมูล (Message ID) ประเภทข้อมูล (Message Type) ข้อมูลบ่งเวลา (Time Indicator) ค่าความสำคัญ (Priority) และวัน-เวลาหมดอายุ (Expiry Time) โดยมีรายละเอียดแสดงดังตารางที่ 3.1

ตารางที่ 3.1 รายละเอียดส่วนหัวของข้อมูลการเปลี่ยนแปลงบริการ

ชื่อข้อมูล	ชนิดของข้อมูล	คำอธิบาย
รหัสข้อมูล (Message ID)	String	รหัสของข้อมูลกำหนดขึ้นโดยระบบ
ประเภทข้อมูล (Message Type)	Enumeration	เป็นข้อมูลที่แสดงถึงประเภทของข้อมูลการเปลี่ยนแปลงบริการ ค่าที่เป็นไปได้ประกอบด้วย 1. *CHANGE_TYPE* สำหรับข้อมูลการเปลี่ยนแปลงชนิดของบริการ 2. *CHANGE_PROPERTY_VAL* สำหรับข้อมูลการเปลี่ยนแปลงค่าคุณสมบัติของบริการ 3. *REMOVAL_SERVICE* สำหรับข้อมูลการยกเลิกบริการ

		4. *INSTANTIATION_SERVICE* สำหรับข้อมูลการเกิดบริการใหม่
ข้อมูลบ่งเวลา (Time Indicator)	Enumeration	เป็นข้อมูลที่ระบุเวลาของการแจ้งการเปลี่ยนแปลง ค่าที่เป็นไปได้ประกอบด้วย 1. *ADVANCE* ระบุถึงการแจ้งการเปลี่ยนแปลงบริการล่วงหน้า 2. *IMMEDIATE* ระบุถึงการแจ้งการเปลี่ยนแปลงในขณะที่เกิดการเปลี่ยนแปลงขึ้น
ค่าความสำคัญ (Priority)	Short	เป็นค่าที่แสดงถึงความสำคัญของข้อมูลการเปลี่ยนแปลงบริการ โดยกำหนดค่าอยู่ในช่วง 1-10 โดย 1 แสดงถึงความสำคัญต่ำสุด และ 10 แสดงถึงความสำคัญสูงสุด
วัน-เวลาหมดอายุ (Expiry Time)	Long	เป็นค่าที่แสดงถึงวันเวลาที่ข้อมูลการเปลี่ยนแปลงบริการยังคงสมเหตุสมผล (Valid) อยู่

2. <u>ส่วนเนื้อความ</u> แบ่งออกได้เป็น 4 ประเภทตามประเภทของข้อมูล ดังนี้คือ

   2.1 เนื้อความข้อมูลการเปลี่ยนแปลงชนิดของบริการของบริการ (Change of Service Type of Service) ประกอบด้วย

   - ข้อมูลคำอธิบายบริการ (Service Description) - ก่อนการเปลี่ยนแปลง
   - ข้อมูลคำอธิบายบริการ - หลังการเปลี่ยนแปลง
   - เหตุผลของการเปลี่ยนแปลง (Reason)
   - วัน-เวลาที่การเปลี่ยนแปลงเริ่มเป็นผล (Effective Time)

   2.2 เนื้อความข้อมูลการเปลี่ยนแปลงค่าคุณสมบัติของบริการ (Change of Property Value of Service) ประกอบด้วย

   - ข้อมูลคำอธิบายบริการ - ก่อนการเปลี่ยนแปลง
   - รายการคุณสมบัติที่ถูกลบออก (List of Deleted Properties)

- รายการคุณสมบัติที่ได้รับการเปลี่ยนแปลงค่า (List of Modified Properties)
- เหตุผลของการเปลี่ยนแปลง
- วัน-เวลาที่การเปลี่ยนแปลงเริ่มเป็นผล

2.3 เนื้อความข้อมูลการยกเลิกบริการ (Removal of Service) ประกอบด้วย

- ข้อมูลคำอธิบายบริการ-ที่ถูกยกเลิก
- เหตุผลของการเปลี่ยนแปลง
- วัน-เวลาทีการเปลี่ยนแปลงเริ่มเป็นผล

2.4 เนื้อความข้อมูลการเกิดบริการใหม่ (Instantiation of Service)

- ข้อมูลคำอธิบายบริการ
- จุดประสงค์ (Objective)
- วัน-เวลาที่การเปลี่ยนแปลงเริ่มเป็นผล

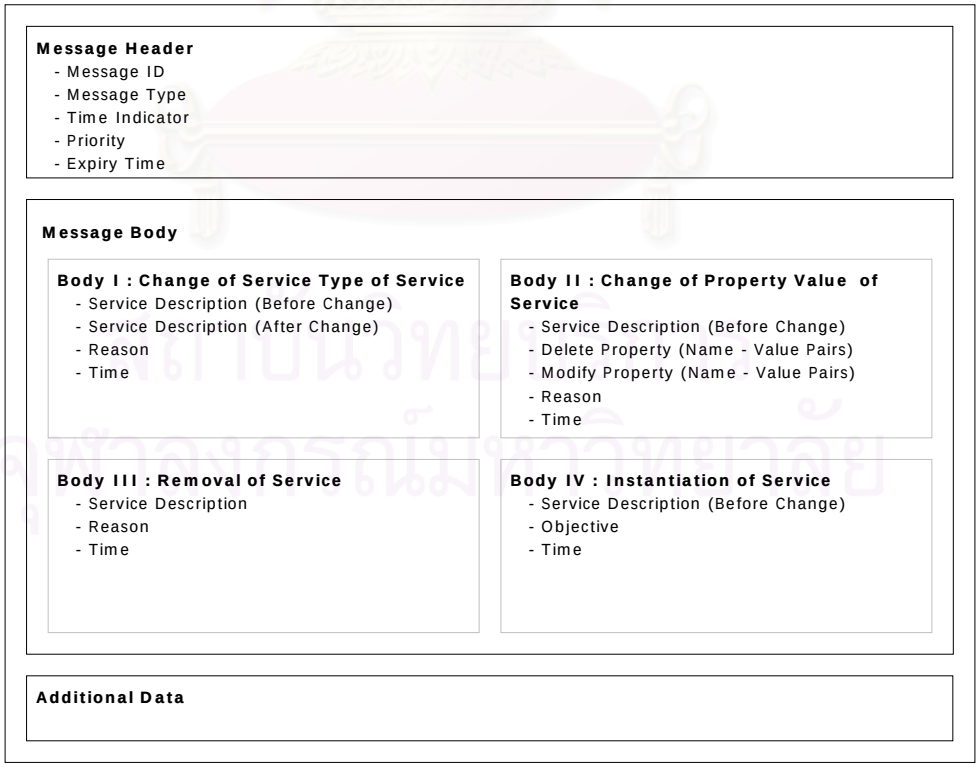
รายละเอียดของข้อมูลของเนื้อความแต่ละประเภทแสดงในตารางที่ 3.2

ตารางที่ 3.2 รายละเอียดข้อมูลเนื้อความ

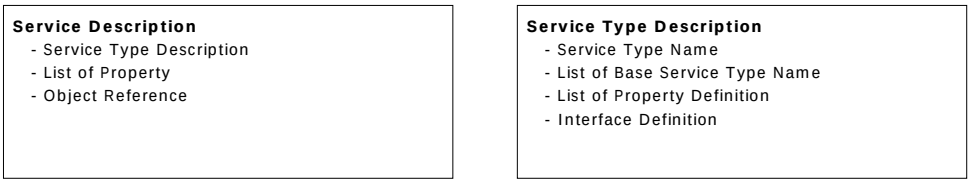
ชื่อข้อมูล	ชนิดของข้อมูล	คำอธิบาย
ข้อมูลคำอธิบายบริการ (Service Description)	Structure	เป็นข้อมูลที่ใช้สำหรับอธิบายบริการทั้งก่อนและหลังจากที่ได้รับการเปลี่ยนแปลง รูปแบบของข้อมูลมีลักษณะเป็นแบบโครงสร้าง (รูปที่ 3.9) ที่ประกอบด้วยข้อมูลคำอธิบายชนิดของบริการ (Service Type Description) รายการนิยามคุณสมบัติ (List of Property Definition) และข้อมูลอ้างถึงวัตถุของบริการ ทั้งนี้คำอธิบายชนิดของบริการเองจะมีรายละเอียดเกี่ยวกับชื่อชนิดของบริการ รายการของชนิดของบริการที่สืบทอดมา รายการนิยามคุณสมบัติและนิยามส่วนต่อประสานของบริการ
รายการคุณสมบัติที่ถูกลบออก (List of Deleted Properties)	Sequence	เป็นรายการคุณสมบัติที่ถูกลบออก มีลักษณะเป็นข้อมูลแบบโครงสร้างที่

		ประกอบด้วยชื่อและค่า (Name-Value Pair)
คุณสมบัติที่ได้รับการแก้ไข (List of Modified Properties)	Sequence	เป็นรายการคุณสมบัติที่ได้รับการเปลี่ยนแปลงค่าแล้ว มีลักษณะเป็นข้อมูลแบบโครงสร้างที่ประกอบด้วยชื่อและค่า
เหตุผล (Reason)	String	เป็นข้อมูลที่ใช้สำหรับระบุเหตุผลการเปลี่ยนแปลงบริการ
จุดประสงค์ (Objective)	String	เป็นข้อมูลที่ใช้สำหรับระบุจุดประสงค์ของบริการ
วัน-เวลาที่ทำการเปลี่ยนแปลง (Effective -Time)	Long	เป็นข้อมูลที่แสดงวัน-เวลาที่ (จะ) ทำการเปลี่ยนแปลงบริการ

3. <u>ส่วนข้อมูลเพิ่มเติม</u> มีชนิดข้อมูลเป็นแบบสายอักขระเป็นส่วนเพิ่มเติมที่สามารถกำหนดข้อมูลใดๆ ได้



รูปที่ 3.8 โครงสร้างข้อมูลการเปลี่ยนแปลงบริการ



รูปที่ 3.9 โครงสร้างข้อมูลคำอธิบายบริการ (Service Description) และ ข้อมูลคำอธิบายชนิดของบริการ (Service Type Description)

สำหรับข้อมูลการเปลี่ยนแปลงบริการที่อยู่ในรูปของเอกสารเอ็กซ์เอ็มแอล จะถูกกำหนดด้วยดีทีดีที่มีเนื้อหาเช่นเดียวกับข้อมูลการเปลี่ยนแปลงในรูปแบบวัตถุเหตุการณ์แบบโครงสร้างดังรูปที่ 3.10

```
1.   <!ELEMENT ServiceChangeNotification (
2.                          NotificationMessageHeader,
3.                          NotificationMessageBody,
4.                          AdditionalData)>
5.   <!--Message Header section -->
6.   <!ELEMENT NotificationMessageHeader (
7.                          MessageID,
8.                          MessageType,
9.                          TimeIndicator,
10.                         Priority,
11.                         ExpiryTime)>
12.  <!ELEMENT MessageID              (#PCDATA)>
13.  <!ELEMENT MessageType            EMPTY>
14.  <!ATTLIST MessageType value (
15.                         Change_Of_Service_Type |
16.                         Change_Of_Property_value |
17.                                     Removal_Of_Service |
18.                         Instantiation_Of_Service)
19.                         " Change_Of_Service_Type">
20.  <!ELEMENT TimeIndicator          EMPTY>
21.  <!ATTLIST TimeIndicator value (
22.                         In_Advance_Notification |
23.                         Immediate_Notification)
24.                         "In_Advance_Notification">
25.  <!ELEMENT Priority               (#PCDATA)>
26.  <!ELEMENT ExpiryTime             (#PCDATA)>
27.  <!--Message Body section -->
28.  <!ELEMENT NotificationMessageBody (
29.                         ChangeOfServiceTypeOfService |
30.                         ChangeOfPropertyValueOfService |
31.                         RemovalOfService |
32.                         InstantiationOfService)>
33.  <!ELEMENT ChangeOfServiceTypeOfService (
34.                         ServiceDescription+,
35.                         Reason,
36.                         Time)>
37.  <!ELEMENT Reason            (#PCDATA)>
38.  <!ELEMENT Time             (#PCDATA)>
39.  <!ELEMENT ChangeOfPropertyValueOfService (
40.                         ServiceDescription,
41.                         DeleteProperties,
42.                         ModifyProperties,
43.                         Reason,
44.                         Time)>
45.  <!ELEMENT DeleteProperties (Property*)>
```

```
46.  <!ELEMENT ModifyProperties (Property*)>
47.  <!ELEMENT RemovalOfService (
48.                       ServiceDescription,
49.                       Reason,
50.                       Time)>
51.  <!ELEMENT InstantiationOfService (
52.                       ServiceDescription,
53.                       Objective,
54.                       Time)>
55.  <!ELEMENT Objective       (#PCDATA)>
56.  <!-- Additional Data section -->
57.  <!ELEMENT AdditionalData  (#PCDATA)>
```

รูปที่ 3.10 ดีทีดีอธิบายโครงสร้างข้อมูลการเปลี่ยนแปลงบริการ

สำหรับดีทีดีที่อธิบายโครงสร้างของส่วนย่อย ServiceDescription (รูปที่ 3.11) และโครงสร้างส่วนย่อย ServiceTypeDescription (รูปที่ 3.12) นั้นถูกดัดแปลงมาจากดีทีดีของคำอธิบายบริการจากงานวิจัย [15]

```
58.  <!ELEMENT ServiceDescription (
59.                        ServiceTypeDescription,
60.                        Property*,
61.                        ObjectReference)>
62.  <!ATTLIST ServiceDescription type (
63.                        ORIGINAL_SERVICE |
64.                        CHANGED_SERVICE)
65.                           "ORIGINAL_SERVICE">
66.  <!ELEMENT Property EMPTY>
67.  <!ATTLIST Property Name CDATA #REQUIRED
68.                      Value CDATA #IMPLIED>
69.  <!ELEMENT  ObjectReference  (#PCDATA)>
```

รูปที่ 3.11 ดีทีดีอธิบายโครงสร้างของส่วนย่อย ServiceDescription

```
70.  <!ELEMENT ServiceTypeDescription (Interface?,
71.  TraderServiceType)>
72.  <!ELEMENT Interface (BaseInterfaces?, Constant*,
73.  Attribute*, Operation*)>
74.  <!ATTLIST Interface Id      CDATA #REQUIRED
75.                     Name    CDATA #REQUIRED
76.                     Version CDATA #REQUIRED>
77.  <!ELEMENT BaseInterfaces (BaseInterface*, Link*)>
78.  <!ELEMENT BaseInterface EMPTY>
79.  <!ATTLIST BaseInterface Id   CDATA #REQUIRED
80.                           Name CDATA #REQUIRED>
81.  <!ELEMENT Link EMPTY>
82.  <!ATTLIST Link Source CDATA #REQUIRED
83.                 Dest   CDATA #REQUIRED>
84.  <!ELEMENT Constant EMPTY>
85.  <!ATTLIST Constant Id       CDATA #REQUIRED
86.                     Name    CDATA #REQUIRED
87.                     Version CDATA #REQUIRED
88.                     Type    CDATA #REQUIRED
89.                     Value   CDATA #REQUIRED
```

```
90.                     Derived (YES | NO) "NO">
91.   <!ELEMENT Attribute EMPTY>
92.   <!ATTLIST Attribute Id      CDATA #REQUIRED
93.                       Name    CDATA #REQUIRED
94.                       Version CDATA #REQUIRED
95.                       Type    CDATA #REQUIRED
96.                       Mode    (NORMAL | READONLY) "NORMAL"
97.                       Derived (YES | NO) "NO">
98.   <!ELEMENT Operation (Parameter*, Exception*, Context*)>
99.   <!ATTLIST Operation Id      CDATA #REQUIRED
100.                      Name    CDATA #REQUIRED
101.                      Version CDATA #REQUIRED
102.                      Type    CDATA #REQUIRED
103.                      Mode    (NORMAL | ONEWAY) "NORMAL"
104.                      Derived (YES | NO) "NO">
105.  <!ELEMENT Parameter EMPTY>
106.  <!ATTLIST Parameter Name    CDATA #REQUIRED
107.                      Type    CDATA #REQUIRED
108.                      Mode    (IN | OUT | INOUT) "IN">
109.  <!ELEMENT Exception (Member)*>
110.  <!ATTLIST Exception Id      CDATA #REQUIRED
111.                      Name    CDATA #REQUIRED
112.                      Version CDATA #REQUIRED
113.                      Derived (YES | NO) "NO">
114.  <!ELEMENT Member EMPTY>
115.  <!ATTLIST Member Name    CDATA #REQUIRED
116.                    Type    CDATA #REQUIRED>
117.  <!ELEMENT Context (#PCDATA)>
118.  <!ELEMENT TraderServiceType (BaseServiceTypes?,
119.                               PropertyDef*)>
120.  <!ATTLIST TraderServiceType Id   CDATA #REQUIRED
121.                              Name CDATA #REQUIRED
122.                              Masked (YES | NO) "NO">
123.  <!ELEMENT BaseServiceTypes (BaseServiceType*, Link*)>
124.  <!ELEMENT BaseServiceType EMPTY>
125.  <!ATTLIST BaseServiceType Name CDATA #REQUIRED>
126.  <!ELEMENT PropertyDef EMPTY>
127.  <!ATTLIST PropertyDef Name CDATA #REQUIRED
128.                                          Type CDATA
129.  #REQUIRED
130.                                          Mode (NORMAL |
131.  READONLY | MANDATORY | MANDATORY_READONLY) "NORMAL"
                            Derived (YES | NO) "NO">
```

รูปที่ 3.12 ดีทีดีอธิบายโครงสร้างของส่วนย่อย ServiceTypeDescription

บทที่ 4
ต้นแบบบริการเอสซีเอ็น

ต้นแบบบริการเอสซีเอ็นภายในวิทยานิพนธ์นี้ได้รับการพัฒนาขึ้นภายใต้ข้อกำหนดของสถาปัตยกรรมคอร์บารุ่นที่ 2.2 โดยใช้ออร์บของวิสิโบรกเกอร์ (Visibroker) สำหรับภาษาจาวา รุ่นที่ 3.4 [16] บริการเทรดเดอร์ของจาคอร์บ (JacORB) รุ่นที่ 1.0 เบต้า 14 [17] บริการแจ้งเหตุการณ์ของดีคอน (dCon) รุ่น 2.2 [18] และใช้ภาษาจาวา (Java) รุ่นที่ 1.2.2 ในการพัฒนา

4.1 ส่วนต่อประสานของบริการเอสซีเอ็น

ในหัวข้อนี้จะกล่าวถึง ส่วนต่อประสานทั้งหมดของบริการเอสซีเอ็น โดยแบ่งออกเป็นส่วนต่างๆ ดังนี้คือ ส่วนต่อประสานของมอดูลสนับสนุนการเปลี่ยนแปลงบริการ ส่วนต่อประสานของตัวจัดการการแจ้งการเปลี่ยนแปลงบริการ และส่วนต่อประสานของตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ

4.1.1 ส่วนต่อประสานของมอดูลสนับสนุนการเปลี่ยนแปลงบริการ

```
module SCSModule {
   interface SCS {

      readonly attribute Object publisherManager_if;
      readonly attribute Object subscriberManager_if;

      CosTrading::OfferId modifyServiceOffer_NewType (
         in CosTrading::OfferId oldOfferId,
         in CosTrading::Register::OfferInfo newOffer,
         in boolean withdrawOldOffer
      )raises (
         CosTrading::Register::InvalidObjectRef,
         CosTrading::Register::InterfaceTypeMismatch,
         CosTrading::IllegalServiceType,
         CosTrading::UnknownServiceType,
         CosTrading::IllegalPropertyName,
         CosTrading::PropertyTypeMismatch,
         CosTrading::MissingMandatoryProperty,
         CosTrading::DuplicatePropertyName,
         CosTrading::IllegalOfferId,
         CosTrading::UnknownOfferId
      );
   };
};
```

ส่วนต่อประสานมอดูลสนับสนุนการเปลี่ยนแปลงประกอบด้วยมอดูลชื่อ SCSModule โดยมีส่วนต่อประสาน SCS อยู่ภายใน รายละเอียดของส่วนต่อประสาน SCS สามารถอธิบายแยกเป็นส่วนๆ ได้ดังต่อไปนี้

ลักษณะประจำของส่วนต่อประสาน

- `readonly attribute Object publisherManager_if;`
  ใช้ระบุค่าข้อมูลอ้างถึงวัตถุของตัวจัดการผู้แจ้งการเปลี่ยนแปลงบริการ
- `readonly attribute Object subscriberManager_if;`
  ใช้ระบุค่าข้อมูลอ้างถึงวัตถุของตัวจัดการผู้ต้องการรับทราบการเปลี่ยนแปลงบริการ

ตัวกระทำการของส่วนต่อประสาน

- modifyServiceOffer_NewType();
  เป็นตัวกระทำการที่ใช้สำหรับเปลี่ยนแปลงชนิดของบริการให้กับบริการหนึ่งๆ
  พารามิเตอร์ของตัวกระทำการ ประกอบด้วย
  - oldOfferId คือ รหัสของข้อเสนอบริการที่จะทำการเปลี่ยนแปลงชนิดของบริการ มีชนิดข้อมูลเป็นแบบ CosTrading::OfferId [2]
  - newOffer คือ ข้อมูลคำอธิบายบริการที่ได้รับการเปลี่ยนแปลงชนิดของบริการเรียบร้อยแล้ว มีรูปแบบข้อมูลเป็นแบบ CosTrading::Register::OfferInfo [2]
  - withdrawOldOffer คือ ข้อมูลที่ระบุว่าต้องการที่จะลบข้อเสนอของบริการตัวเก่าทิ้งด้วยหรือไม่ มีชนิดของมูลเป็นแบบค่าตรรกะ (Boolean) โดยหากมีค่าเท่ากับจริง (True) จะเป็นการระบุว่าต้องการที่จะลบข้อเสนอของบริการตัวเก่า หากมีค่าเท่ากับเท็จ (False) จะเป็นการระบุว่าไม่ต้องการที่จะลบข้อเสนอของบริการตัวเก่า

  เอ็กซ์เซ็พชันของตัวกระทำการ ประกอบด้วย
  - CosTrading::Register::InvalidObjectRef เอ็กซ์เซ็พชันนี้เกิดขึ้นเมื่อข้อมูลอ้างถึงวัตถุของข้อมูลข้อเสนอบริการไม่ถูกต้อง
  - CosTrading::Register::InterfaceTypeMismatch เอ็กซ์เซ็พชันนี้เกิดขึ้นเมื่อข้อมูลชนิดของส่วนต่อประสานของข้อเสนอบริการไม่ถูกต้อง
  - CosTrading::IllegalServiceType เอ็กซ์เซ็พชันนี้เกิดขึ้นเมื่อข้อมูลชื่อชนิดของบริการมีรูปแบบที่ไม่ถูกต้อง
  - CosTrading::UnknownServiceType เอ็กซ์เซ็พชันนี้เกิดขึ้นเมื่อไม่พบข้อมูลชื่อชนิดของบริการในคลังชนิดของบริการ
  - CosTrading::IllegalPropertyName เอ็กซ์เซ็พชันนี้เกิดขึ้นเมื่อชื่อคุณสมบัติของบริการมีรูปแบบที่ไม่ถูกต้อง

- CosTrading::PropertyTypeMismatch เอ็กซ์เซ็พชันนี้เกิดขึ้นเมื่อค่าของคุณสมบัติของบริการมีรูปแบบข้อมูลที่ไม่ตรงกับที่กำหนดไว้
- CosTrading::MissingMandatoryProperty เอ็กซ์เซ็พชันนี้เกิดขึ้นเมื่อคุณสมบัติของบริการ (ที่กำหนดไว้ว่าจำเป็นต้องระบุค่า) ไม่ได้รับการระบุค่าไว้
- CosTrading::DuplicatePropertyName เอ็กซ์เซ็พชันนี้เกิดขึ้นเมื่อเกิดการซ้ำกันของชื่อของคุณสมบัติของบริการ
- CosTrading::IllegalOfferId เอ็กซ์เซ็พชันนี้เกิดขึ้นเมื่อรูปแบบของข้อมูลรหัสของข้อเสนอบริการไม่ถูกต้อง

ค่าคืนกลับของตัวกระทำการ ประกอบด้วย

- OfferId คือ ข้อมูลรหัสของข้อเสนอบริการที่ได้รับการเปลี่ยนแปลงแล้ว โดยมีรูปแบบข้อมูลเป็นแบบ CosTrading::OfferId [2]

4.1.2 ส่วนต่อประสานของตัวจัดการการแจ้งการเปลี่ยนแปลงบริการ

```
module scnmanager {
 interface SCNComponent;
 interface EventManager;
 interface PublisherManager;
 interface SubscriberManager;
```

เป็นการประกาศส่วนต่อประสานย่อยทั้งหมดของตัวจัดการการแจ้งการเปลี่ยนแปลงบริการ โดยมีรายละเอียดของแต่ละส่วนต่อประสานดังต่อไปนี้

1. ส่วนต่อประสานส่วนประกอบของบริการเอสซีเอ็น

```
interface SCNComponent {
 readonly attribute EventManager      eventManager_if;
 readonly attribute PublisherManager  publisherManager_if;
 readonly attribute SubscriberManager subscriberManager_if;
};
```

เป็นส่วนต่อประสานที่ระบุค่าข้อมูลอ้างถึงวัตถุของส่วนประกอบย่อยต่างๆของตัวจัดการการแจ้งการเปลี่ยนแปลง ภายในประกอบด้วยลักษณะประจำแบบอ่านค่าได้อย่างเดียว 3 ตัวคือ

- `readonly attribute EventManager eventManager_if;`
  ใช้ระบุค่าข้อมูลอ้างถึงวัตถุของตัวจัดการเหตุการณ์
- `readonly attribute PublisherManager publisherManager_if;`
  ใช้ระบุค่าข้อมูลอ้างถึงวัตถุของตัวจัดการผู้แจ้งการเปลี่ยนแปลงบริการ
- `readonly attribute SubscriberManager subscriberManager_if;`
  ใช้ระบุค่าข้อมูลอ้างถึงวัตถุของตัวจัดการผู้ต้องการรับทราบการเปลี่ยนแปลงบริการ

2. ส่วนต่อประสานตัวจัดการเหตุการณ์

```
interface EventManager : SCNComponent {
   void processEventMesg(
   in scnmesg::StructuredEventMessage eventMesg)
   raises (scncomm::DummyException);
};
```

เป็นส่วนต่อประสานของตัวจัดการเหตุการณ์ ซึ่งสืบทอดมาจากส่วนต่อประสาน SCNComponent ประกอบด้วยตัวกระทำการเพียงตัวเดียวคือ ตัวกระทำการ processEventMesg() เป็นตัวกระทำการสำหรับให้ตัวตรวจบริการเทรดเดอร์และตัวจัดการผู้แจ้งการเปลี่ยนแปลงบริการ เรียกใช้เพื่อส่งข้อมูลการเปลี่ยนแปลงบริการให้

3. ส่วนต่อประสานตัวจัดการผู้แจ้งการเปลี่ยนแปลงบริการ

```
interface PublisherManager : SCNComponent {
 oneway void notify(
    in scnmesg::PublishedStructuredEventMessage eventMesg);
};
```

เป็นส่วนต่อประสานของตัวจัดการผู้แจ้งการเปลี่ยนแปลง ซึ่งสืบทอดมาจากส่วนต่อประสาน SCNComponent ประกอบด้วยตัวกระทำการเพียงตัวเดียวคือ ตัวกระทำการ notify() ซึ่งเป็นตัวกระทำการสำหรับผู้เปลี่ยนแปลงบริการใช้ในการแจ้งการเปลี่ยนแปลงบริการล่วงหน้า โดยรับพารามิเตอร์ที่มีชนิดข้อมูลเป็นแบบ scnmesg::PublishedStructuredEventMessage

4. ส่วนต่อประสานตัวจัดการผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ

```
interface SubscriberManager :SCNComponent {

   scncomm::AdminSubscriber registerSubscriber (
    in scncomm::SubscriberID subscriberID,
    in scncomm::Password passwd)
      raises (scncomm::ExceedSubscriberLimit,
              scncomm::DuplicatedSubscriberID);
   void unregisterSubscriber (
    in scncomm::SubscriberID subscriberID,
    in scncomm::Password passwd)
      raises (scncomm::NotFoundSubscriberID,
              scncomm::InvalidPassword);
   scncomm::AdminSubscriber getAdminSubscriber (
     in scncomm::SubscriberID subscriberID,
     in scncomm::Password passwd)
       raises (scncomm::NotFoundSubscriberID,
               scncomm::InvalidPassword);
};
```

เป็นส่วนต่อประสานของตัวจัดการผู้ต้องการรับทราบการเปลี่ยนแปลงบริการ ซึ่งสืบทอดมาจากส่วนต่อประสาน SCNComponent

ตัวกระทำการของส่วนต่อประสาน

- ตัวกระทำการ registerSubscriber ()

เป็นตัวกระทำการสำหรับให้ผู้ต้องการรับทราบการเปลี่ยนแปลงบริการเรียกใช้ในการลงทะเบียนเพื่อขอเข้าใช้งานบริการเอสซีเอ็น

พารามิเตอร์ของตัวกระทำการ ประกอบด้วย

- subscriberID คือ ข้อมูลรหัสของผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ
- passwd คือ ข้อมูลรหัสผ่านของผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ

เอ็กซ์เซ็พชันของตัวกระทำการ ประกอบด้วย

- ExceedSubscriberLimit เอ็กซ์เซ็พชันนี้เกิดขึ้นเมื่อจำนวนผู้ที่ลงทะเบียนเพื่อขอรับทราบการเปลี่ยนแปลงครบตามจำนวนสูงสุดที่กำหนดไว้
- DuplicatedSubscriberID เอ็กซ์เซ็พชันนี้เกิดขึ้นเมื่อรหัสของผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการซ้ำกับรหัสของผู้อื่นที่ลงทะเบียนไว้ก่อนหน้านี้

ค่าคืนกลับของตัวกระทำการ ประกอบด้วย

- AdminSubscriber คือ วัตถุผู้ดูแล (Admin Subscriber) ที่ใช้ในการสร้างและดูแลตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงสำหรับรายละเอียดส่วนต่อประสานของ AdminSubscriber ได้อธิบายไว้ในหัวข้อ 4.1.3

- ตัวกระทำการ unregisterSubscriber ()

เป็นตัวกระทำการสำหรับให้ผู้ต้องการรับทราบการเปลี่ยนแปลงบริการเรียกใช้ในการยกเลิกการขอเข้าใช้งานบริการเอสซีเอ็น

พารามิเตอร์ของตัวกระทำการ ประกอบด้วย

- subscriberID คือ ข้อมูลรหัสของผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ
- passwd คือ ข้อมูลรหัสผ่านของผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ

เอ็กซ์เซ็พชันของตัวกระทำการ ประกอบด้วย

- NotFoundSubscriberID เอ็กซ์เซ็พชันนี้เกิดขึ้นเมื่อไม่พบรหัสของผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ
- InvalidPassword เอ็กซ์เซ็พชันนี้เกิดขึ้นเมื่อรหัสผ่านไม่ถูกต้อง

- ตัวกระทำการ getAdminSubscriber ()

เป็นตัวกระทำการสำหรับให้ผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการเรียกใช้เพื่อขอข้อมูลอ้างถึงวัตถุของผู้ดูแล

พารามิเตอร์ของตัวกระทำการ ประกอบด้วย

- subscriberID คือ ข้อมูลรหัสของผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ
- passwd คือ ข้อมูลรหัสผ่านของผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ

เอ็กซ์เซ็พชั่นของตัวกระทำการ ประกอบด้วย

- NotFoundSubscriberID เอ็กซ์เซ็พชันนี้เกิดขึ้นเมื่อไม่พบรหัสของผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ
- InvalidPassword เอ็กซ์เซ็พชันนี้เกิดขึ้นเมื่อรหัสผ่านไม่ถูกต้อง

ค่าคืนกลับของตัวกระทำการ ประกอบด้วย

- AdminSubscriber คือ วัตถุผู้ดูแลที่สร้างขึ้นจากการลงทะเบียนของผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการโดยตัวกระทำการ registerSubscriber()

4.1.3 ส่วนต่อประสานผู้ดูแลและส่วนต่อประสานตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ

ในส่วนนี้จะอธิบายเกี่ยวกับส่วนต่อประสานผู้ดูแล และส่วนต่อประสานตัวแทนทั้ง 6 ประเภท รวมทั้งส่วนต่อประสานอื่นๆ ที่เกี่ยวข้อง ได้แก่ส่วนต่อประสานของส่วนจัดการคุณภาพของบริการและส่วนต่อประสานของส่วนจัดการข้อมูลระบุความต้องการ ส่วนต่อประสานทั้งหมดในส่วนนี้อยู่ภายใต้มอดูลชื่อ scncomm โดยมีการประกาศชนิดของข้อมูลและเอ็กซ์เซ็พชันบางส่วนที่มีการใช้งานร่วมกัน ดังนี้

```
module scncomm {
```

การประกาศชนิดข้อมูล ประกอบด้วย

```
typedef string SubscriberID;
typedef string ProxyID;
typedef string AdminID;
typedef string Password;

enum ProxyType {
    PUSH_STRUCTURED,
    PULL_STRUCTURED,
    PUSH_STRUCTURED_SEQUENCE,
    PULL_STRUCTURED_SEQUENCE,
    PUSH_XML,
    PULL_XML
};
```

```
typedef string XMLDocument;
typedef string PropertyName;
typedef any PropertyValue;

struct Property {
 PropertyName name;
 PropertyValue value;
};
typedef sequence<Property> PropertySeq;
typedef PropertySeq QoSProperties;
```

<u>การประกาศเอ็กซ์เซ็พชัน</u> ประกอบด้วย

```
exception Disconnected{};
exception AlreadyConnected{};
exception AlreadyDisconnected{};
exception TypeError{};
exception ExceedSubscriberLimit {};
exception DupplicatedSubscriberID {};
exception ExceedProxyLimit {};
exception NotFoundSubscriberID {};
exception InvalidPassword {};
exception InvalidProxyType {};
exception NotFoundProxy {};
exception AlreadyHaveTheType {};
exception UnsupportedQoS {};
exception FilterNotFound{};
exception IllegalConstraint{};
```

1. <u>ส่วนต่อประสานผู้ดูแล</u>

```
interface AdminSubscriber : FilterAdmin, QoSAdmin {
 readonly attribute scncomm::AdminID adminID;
 ProxySubscriber createProxySubscriber(
    in scncomm::ProxyType proxyType,
    out scncomm::ProxyID proxyID)
    raises (InvalidProxyType, AlreadyHaveTheType);

 void destroyProxySubscriber(in ProxyID proxyID)
    raises (NotFoundProxy);

 void destroyAllProxySubscriber();

 scncomm::ProxySubscriber getProxySubscriber(in ProxyID proxyID)
     raises (NotFoundProxy);

 ProxySubscriberSeq getAllProxySubscriber();
};
```

เป็นส่วนต่อประสานของผู้ดูแลตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ ซึ่งมีการสืบทอดมาจากส่วนต่อประสาน 2 ตัวคือ ส่วนต่อประสาน FilterAdmin และ ส่วนต่อประสาน QoSAdmin

<u>ลักษณะประจำของส่วนต่อประสาน</u>

- adminID

ใช้ระบุรหัสของผู้ดูแล

ตัวกระทำการของส่วนต่อประสาน

- ตัวกระทำการ createProxySubscriber()

เป็นตัวกระทำการที่ใช้สำหรับสร้างวัตถุตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ

พารามิเตอร์ของตัวกระทำการ ประกอบด้วย

- proxyType เป็นการกำหนดประเภทของวัตถุตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงซึ่งต้องการจะสร้าง
- proxyID เป็นพารามิเตอร์แบบคืนค่ากลับ (out Parameter) โดยค่าที่คืนกลับมาจะเป็นรหัสของตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการที่ได้รับการสร้างขึ้น

เอ็กซ์เซ็พชันของตัวกระทำการ ประกอบด้วย

- InvalidProxyType เอ็กซ์เซ็พชันนี้เกิดขึ้นเมื่อค่าของพารามิเตอร์ proxyType ไม่ถูกต้อง
- AlreadyHaveTheType เอ็กซ์เซ็พชันนี้เกิดขึ้นเมื่อตัวแทนประเภทตามที่ระบุในพารามิเตอร์ proxyType ได้รับการสร้างขึ้นแล้วสำหรับผู้ที่ต้องการรับทราบการเปลี่ยนแปลงรายนี้ (กำหนดให้สามารถมีตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงประเภทหนึ่งๆ ได้เพียงตัวเดียวเท่านั้นสำหรับผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการแต่ละราย)

ค่าคืนกลับของตัวกระทำการ

- ProxySubscriber คือ วัตถุตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงที่ถูกสร้างขึ้น

- ตัวกระทำการ destroyProxySubscriber()

เป็นตัวกระทำการที่ใช้สำหรับลบหรือทำลายตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการที่ถูกสร้างขึ้นก่อนหน้านี้

พารามิเตอร์ของตัวกระทำการ ประกอบด้วย

- proxyID คือ รหัสของตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ

เอ็กซ์เซ็พชันของตัวกระทำการ ประกอบด้วย

- NotFoundProxy เอ็กซ์เซ็พชันนี้เกิดขึ้นเมื่อไม่พบตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงตามที่ระบุไว้ในพารามิเตอร์ proxyID

- ตัวกระทำการ destroyAllProxySubscriber()

เป็นตัวกระทำการที่ใช้สำหรับลบตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการทุกตัวที่ได้สร้างขึ้นไว้

- ตัวกระทำการ getProxySubscriber()

เป็นตัวกระทำการที่ใช้สำหรับร้องขอข้อมูลอ้างถึงวัตถุของตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการที่ได้รับการสร้างขึ้นไว้ก่อนหน้านี้

<u>พารามิเตอร์ของตัวกระทำการ</u> ประกอบด้วย

- proxyID คือ รหัสของตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลง

<u>เอ็กซ์เซ็พชันของตัวกระทำการ</u> ประกอบด้วย

- NotFoundProxy เอ็กซ์เซ็พชันนี้เกิดขึ้นเมื่อไม่พบตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงตามที่ระบุไว้ในพารามิเตอร์ proxyID

<u>ค่าคืนกลับของตัวกระทำการ</u> ประกอบด้วย

- ProxySubscriber คือ วัตถุตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงที่ถูกสร้างขึ้น

- ตัวกระทำการ getAllProxySubscriber()

เป็นตัวกระทำการที่ใช้สำหรับการร้องขอข้อมูลอ้างถึงวัตถุของตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงทุกตัวที่ได้รับการสร้างขึ้นก่อนหน้านี้

<u>ค่าคืนกลับของตัวกระทำการ</u>

- ProxySubscriberSeq คือชุดหรืออาเรย์ของข้อมูลอ้างถึงวัตถุทั้งหมดของตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ

2. <u>ส่วนต่อประสานของตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ</u>

```
interface ProxySubscriber :FilterAdmin, QoSAdmin {
 readonly attribute scncomm::ProxyID proxyID;
 readonly attribute scncomm::ProxyType MyType;

 void disconnect()
    raises(AlreadyDisconnected);
 void destroy();
};
typedef sequence<ProxySubscriber>ProxySubscriberSeq;
```

เป็นส่วนต่อประสานที่ตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการแต่ละประเภทจะต้องสืบทอดส่วนต่อประสานนี้ สืบทอดมาจากส่วนต่อประสาน FilterAdmin และ QoSAdmin และประกอบด้วย

ลักษณะประจำของส่วนต่อประสาน

- `readonly attribute scncomm::ProxyID proxyID;`
  ใช้ระบุค่ารหัสของตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ
- `readonly attribute scncomm::ProxyType MyType;`
  ใช้ระบุประเภทของตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ

ตัวกระทำการของส่วนต่อประสาน

- ตัวกระทำการ disconnect()

เป็นตัวกระทำการที่ผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการใช้สำหรับยกเลิกการติดต่อกับตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ มีเอ็กซ์เซ็พชัน คือ AlreadyDisconnected ซึ่งจะเกิดขึ้นเมื่อมีการเรียกใช้งานตัวกระทำการนี้ในขณะที่ไม่ได้ติดต่ออยู่กับตัวแทนอยู่แล้ว

สำหรับส่วนต่อประสานของตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการแต่ละชนิดนั้น ในที่นี้จะอธิบายเพียง 3 ประเภทเท่านั้น (ส่วนอีก 3 ประเภทที่เหลือก็จะมีความคล้ายคลึงกันต่างกันแต่เพียงพารามิเตอร์ของตัวกระทำการเท่านั้น ดังต่อไปนี้

ส่วนต่อประสานของ StructuredProxyPushSubscriber

```
interface StructuredProxyPushSubscriber : ProxySubscriber {
   void connect (in StructuredPushSubscriber push_subscriber)
      raises(AlreadyConnected,
             TypeError,
             ExceedConnectSubscriberLimit);
};
```

ตัวกระทำการของส่วนต่อประสาน ประกอบด้วย

- ตัวกระทำการ connect()

เป็นตัวกระทำการที่ผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการใช้ในการติดต่อกับตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ เพื่อกำหนดความต้องการในการรับทราบการเปลี่ยนแปลงบริการ หรือเพื่อขอรับข้อมูลการเปลี่ยนแปลงบริการ

พารามิเตอร์ของตัวกระทำการ

- push_subscriber คือวัตถุเรียกกลับ ที่ตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงใช้ในการส่งข้อมูลการเปลี่ยนแปลงบริการไปให้ โดยจะต้องสร้างตามส่วนต่อประสาน StructuredPushSubscriber

เอ็กซ์เซ็พชันของตัวกระทำการ ประกอบด้วย

- AlreadyConnected เอ็กซ์เซ็พชันนี้เกิดขึ้นเมื่อมีการเรียกตัวกระทำการ connect() ในขณะที่ติดต่ออยู่กับตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการอยู่แล้ว
- TypeError เอ็กซ์เซ็พชันนี้เกิดขึ้นเมื่อชนิดของวัตถุเรียกกลับไม่ได้เป็น StrucuturedPushSubscriber
- ExceedConnectSubscriberLimit เอ็กซ์เซ็พชันนี้เกิดขึ้นเมื่อจำนวนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการที่ติดต่ออยู่กับบริการเอสซีเอ็นเกินค่าสูงสุดที่กำหนด

ส่วนต่อประสานของ StructuredProxyPullSubscriber

```
interface StructuredProxyPullSubscriber :ProxySubscriber {
 void connect (in StructuredPullSubscriber pull_subscriber)
    raises(AlreadyConnected,
    TypeError,
    ExceedConnectSubscriberLimit);
 scnmesg::StructuredEventMessage pull_structured_event(
    out boolean has_event)
    raises(Disconnected);
};
```

ตัวกระทำการของส่วนต่อประสาน ประกอบด้วย

- ตัวกระทำการ connect()

เป็นตัวกระทำการที่ใช้ในการติดต่อกับตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ เช่นเดียวกับตัวกระทำการ connect() ของส่วนต่อประสาน StructuredProxyPullSubscriber สิ่งที่แตกต่างกันคือ พารามิเตอร์ของตัวกระทำการนี้จะมีชนิดเป็นแบบ StructuredPullSubscriber

- ตัวกระทำการ pull_structured_event()

เป็นตัวกระทำการที่ผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการใช้ในการดึงข้อมูลการเปลี่ยนแปลงบริการ

พารามิเตอร์ของตัวกระทำการ

- has_event เป็นพารามิเตอร์แบบคืนค่ากลับ ค่าที่คืนกลับมาจะมีชนิดข้อมูลเป็นแบบค่าตรรกะที่แสดงว่าในการดึงข้อมูลการเปลี่ยนแปลงบริการครั้งนั้นๆ มีข้อมูลส่งกลับมาหรือไม่

เอ็กซ์เซ็พชันของตัวกระทำการ

- Disconnected เอ็กซ์เซ็พชันนี้เกิดขึ้นเมื่อมีการเรียกใช้งานตัวกระทำการในขณะที่ไม่ได้ติดต่ออยู่กับตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ

ค่าคืนกลับของตัวกระทำการ

- StructuredEventMessage เป็นข้อมูลการเปลี่ยนแปลงบริการในรูปแบบวัตถุเหตุการณ์แบบโครงสร้าง

ส่วนต่อประสานของ SequenceStructuredProxyPullSubscriber

```
interface SequenceStructuredProxyPullSubscriber : ProxySubscriber{
 void connect (in SequenceStructuredPullSubscriber
    sequence_pull_subscriber)
    raises(AlreadyConnected,
           TypeError,
           ExceedConnectSubscriberLimit);

 scnmesg::StructuredEventBatch pull_sequence_structured_event(
    in long max_number, out boolean has_event)
    raises(Disconnected);

 scnmesg::StructuredEventBatch try_pull_sequence_structured_event(
    in long max_number,
    out boolean has_event)
    raises(Disconnected);
};
```

ตัวกระทำการของส่วนต่อประสาน ประกอบด้วย

- ตัวกระทำการ connect()

เป็นตัวกระทำการที่ผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการใช้ในการติดต่อกับตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการเช่นเดียวกับตัวกระทำการ connect() ของส่วนต่อประสาน StrucuturedProxyPullSubscriber สิ่งที่แตกต่างกันคือ พารามิเตอร์ของตัวกระทำการนี้จะมีชนิดเป็นแบบ SequenceStructuredPullSubscriber

- ตัวกระทำการ pull_sequence_structured_event()

เป็นตัวกระทำการที่ใช้ในการดึงชุดของข้อมูลการเปลี่ยนแปลงบริการ โดยในการดึงข้อมูลจะมีการรอจนกว่าจะได้ข้อมูลครบตามจำนวนที่ระบุไว้ (จำนวนข้อมูลคิดจากการเปรียบเทียบค่าคุณสมบัติ MaximumBatchSize (หัวข้อที่ 3.3.2) กับค่าพารามิเตอร์ max_number โดยค่าที่น้อยกว่าจะเป็นค่าที่ถูกเลือก) หรือครบกำหนดเวลาที่ระบุไว้ในค่าคุณสมบัติ PacingInterval (หัวข้อที่ 3.3.2)

พารามิเตอร์ของตัวกระทำการ ประกอบด้วย

- max_number เป็นการระบุค่าจำนวนข้อมูลการเปลี่ยนแปลงบริการที่ต้องการจะดึงมาในคราวเดียวกัน โดยตัวกระทำการจะนำค่าของ max_number นี้ไปเปรียบเทียบกับค่าของคุณสมบัติ MaximumBatchSize โดยค่าที่น้อยกว่าก็จะเป็นจำนวนของข้อมูลที่ตัวกระทำการจะทำการดึงมา
- has_event เป็นพารามิเตอร์แบบคืนค่ากลับ มีชนิดข้อมูลเป็นแบบค่าตรรกะที่แสดงว่าในการดึงข้อมูลการเปลี่ยนแปลงบริการครั้งนั้นๆ มีข้อมูลส่งกลับมาหรือไม่

เอ็กซ์เซ็พชันของตัวกระทำการ ประกอบด้วย

- Disconnected เอ็กซ์เซ็พชันนี้เกิดขึ้นเมื่อมีการเรียกใช้งานตัวกระทำการในขณะที่ไม่ได้ติดต่ออยู่กับตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ

ค่าคืนกลับของตัวกระทำการ ประกอบด้วย

- StructuredEventBatch เป็นชุดของข้อมูลการเปลี่ยนแปลงบริการในรูปแบบวัตถุเหตุการณ์แบบโครงสร้าง

- ตัวกระทำการ try_pull_sequence_structured_event()

เป็นตัวกระทำการที่ใช้ในการดึงชุดของข้อมูลการเปลี่ยนแปลงบริการเช่นเดียวกับตัวกระทำการ pull_sequence_structured_event() แต่จะต่างกันโดยตัวกระทำการนี้จะไม่มีการรอ นั่นคือตัวกระทำการจะพยายามรวบรวมข้อมูลให้ได้ตามจำนวนที่ระบุ แต่หากในเวลานั้นมีจำนวนข้อมูลไม่ครบ ตัวกระทำการก็จะส่งข้อมูลเท่าที่มีในขณะนั้นมาให้ ซึ่งในลักษณะเช่นนี้ค่าคุณสมบัติ PacingInterval จะไม่ถูกนำมาพิจารณา

พารามิเตอร์ของตัวกระทำการ ประกอบด้วย

- max_number เป็นการระบุค่าจำนวนข้อมูลการเปลี่ยนแปลงบริการที่ต้องการดึงมาในคราวเดียวกัน โดยตัวกระทำการจะนำค่าของ max_number นี้ไปเปรียบเทียบกับค่าของคุณสมบัติ MaximumBatchSize โดยค่าที่น้อยกว่าก็จะเป็นจำนวนของข้อมูลที่ตัวกระทำการจะพยายามดึงมา
- has_event เป็นพารามิเตอร์แบบคืนค่ากลับ มีชนิดข้อมูลเป็นแบบค่าตรรกะที่แสดงว่าในการดึงข้อมูลการเปลี่ยนแปลงบริการครั้งนั้นๆ มีข้อมูลส่งกลับมาหรือไม่

<u>เอ็กซ์เซ็พชันของตัวกระทำการ</u> ประกอบด้วย

- Disconnected เอ็กซ์เซ็พชันนี้เกิดขึ้นเมื่อมีการเรียกใช้งานตัวกระทำการในขณะที่ไม่ได้ติดต่ออยู่กับตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ

<u>ค่าคืนกลับของตัวกระทำการ</u>

- StructredEventBatch เป็นชุดของข้อมูลการเปลี่ยนแปลงบริการในรูปแบบวัตถุเหตุการณ์แบบโครงสร้าง

3. <u>ส่วนต่อประสานของส่วนจัดการคุณภาพบริการ</u>

```
const string OrderPolicy            = "OrderPolicy"
const string DiscardPolicy          =  "DiscardPolicy";
const string MaximumBatchSize       = "MaximumBatchSize";
const string PacingInterval         = "PacingInterval";
const short  FifoOrder              =  1
const short  PriorityOrder          =  2;
const short  DeadlineOrder          =  3;
const short  LifoOrder              =  4;
```

เป็นการประกาศค่าคงที่สำหรับใช้ในการระบุค่าคุณภาพของบริการ

```
interface QoSAdmin {
   void set_qos (in QoSProperties qos)
      raises (UnsupportedQoS);
   QoSProperties get_qos();
};
```

เป็นส่วนต่อประสานของส่วนจัดการคุณภาพของบริการที่เรียกใช้โดยผู้ต้องการรับทราบการเปลี่ยนแปลงบริการ ประกอบด้วยตัวกระทำการ 2 ตัว คือ

- ตัวกระทำการ set_qos()

เป็นตัวกระทำการที่ผู้ต้องการรับทราบการเปลี่ยนแปลงบริการใช้ในการกำหนดค่าคุณสมบัติคุณภาพบริการ

- ตัวกระทำการ get_qos()

เป็นตัวกระทำการที่ผู้ต้องการรับทราบการเปลี่ยนแปลงบริการใช้ในการร้องขอดูค่าของคุณภาพบริการ

4. <u>ส่วนต่อประสานของส่วนจัดการข้อมูลระบุความต้องการ</u>

```
typedef string FilterID;
typedef sequence<FilterID>FilterIDSeq;
typedef string TraderConstraint;
typedef string IORString;
typedef sequence<IORString>IORStringSeq;

struct TraderFilterContent {
```

```
   scnmesg::ServiceTypeName type;
   TraderConstraint constr;
};
typedef sequence<TraderFilterContent>TraderFilterContentSeq;

enum FilterEventType {
   CHANGE_SERVICE_EVENT,
   INSTANTIATION_SERVICE_EVENT,
   CHANGE_AND_INSTATIATION_SERVICE_EVENT
};

struct FilterContent {
   FilterEventType type;
   TraderFilterContentSeq traderConstraints;
   IORStringSeq iors;
};

struct FilterInfo{
   FilterID filterID;
   FilterContent filter;
};
typedef sequence<FilterInfo>FilterInfoSeq;
```

เป็นการประกาศชนิดของข้อมูลที่จำเป็นต่อส่วนต่อประสานของตัวจัดการข้อมูลความต้องการรับทราบการเปลี่ยนแปลงบริการ

```
interface FilterAdmin {

 void addFilter(in FilterContent filter, out FilterID filterID)
   raises(IllegalConstraint);

 void removeFilter(in FilterID filterID)
   raises(FilterNotFound);

 void remove_all_filters();

 FilterIDSeq getAllFilterID();

 FilterInfo getFilter(in FilterID filterID)
   raises(FilterNotFound);

 FilterInfoSeq getAllFilters();

};
```

เป็นส่วนต่อประสานสำหรับตัวจัดการข้อมูลระบุความต้องการ สำหรับให้ผู้ต้องการรับทราบการเปลี่ยนแปลงบริการเรียกใช้ในการระบุถึงบริการที่ต้องการรับแจ้งการเปลี่ยนแปลงต่างๆ ที่เกิดขึ้น ประกอบด้วยตัวกระทำการซึ่งสามารถอธิบายได้ดังนี้

- ตัวกระทำการ addFilter()

  เป็นตัวกระทำการที่ใช้สำหรับการเพิ่มข้อมูลระบุความต้องการ

  <u>พารามิเตอร์ของตัวกระทำการ</u> ประกอบด้วย

  - filter คือข้อมูลระบุความต้องการโดยมีรูปแบบตามโครงสร้าง FilterContent ที่แสดงข้างต้น

- filterID เป็นพารามิเตอร์แบบคืนค่ากลับ โดยจะเป็นค่าของรหัสข้อมูลระบุความต้องการ

<u>เอ็กซ์เซ็พชันของตัวกระทำการ</u> ประกอบด้วย

- IllegalConstraint เอ็กซ์เซ็พชันนี้เกิดขึ้นเมื่อข้อมูลภาษาบังคับที่ใช้ในการระบุความต้องการมีรูปแบบไม่ถูกต้อง

- ตัวกระทำการ removeFilter()

เป็นตัวกระทำการที่ใช้สำหรับลบข้อมูลระบุความต้องการ

<u>พารามิเตอร์ของตัวกระทำการ</u> ประกอบด้วย

- filterID ระบุรหัสของข้อมูลระบุความต้องการที่ต้องการจะลบ

<u>เอ็กซ์เซ็พชันของตัวกระทำการ</u> ประกอบด้วย

- FilterNotFound เอ็กซ์เซ็พชันนี้เกิดขึ้นเมื่อไม่พบรหัสของข้อมูลระบุความต้องการที่ระบุไว้ในพารามิเตอร์ filterID

- ตัวกระทำการ remove_all_filters()

เป็นตัวกระทำการที่ใช้สำหรับลบข้อมูลระบุความต้องการทั้งหมดที่มีอยู่

- ตัวกระทำการ getAllFilterID()

เป็นตัวกระทำการที่ใช้สำหรับขอดูค่ารหัสของข้อมูลระบุความต้องการทั้งหมดที่มีอยู่

- ตัวกระทำการ getFilter()

เป็นตัวกระทำการที่ใช้สำหรับขอดูข้อมูลระบุความต้องการ

<u>พารามิเตอร์ของตัวกระทำการ</u> ประกอบด้วย

- filterID ระบุรหัสของข้อมูลระบุความต้องการที่ต้องการจะดูข้อมูล

<u>เอ็กซ์เซ็พชันของตัวกระทำการ</u> ประกอบด้วย

- FilterNotFound เอ็กซ์เซ็พชันนี้เกิดขึ้นเมื่อไม่พบรหัสของข้อมูลระบุความต้องการที่ระบุไว้ในพารามิเตอร์ filterID

<u>ค่าคืนกลับของตัวกระทำการ</u> ประกอบด้วย

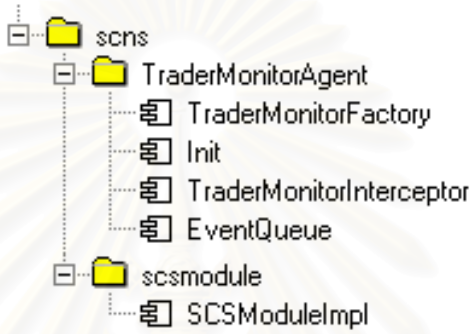
- FilterInfo คือรายละเอียดของข้อมูลระบุความต้องการ ซึ่งมีรูปแบบตามส่วนต่อประสานของ FilterInfo ที่แสดงข้างต้น

- ตัวกระทำการ getAllFilters()

เป็นตัวกระทำการที่ใช้สำหรับการร้องขอดูข้อมูลระบุความต้องการทั้งหมดที่มีอยู่

4.2 ต้นแบบส่วนเพิ่มขยายบริการเทรดเดอร์

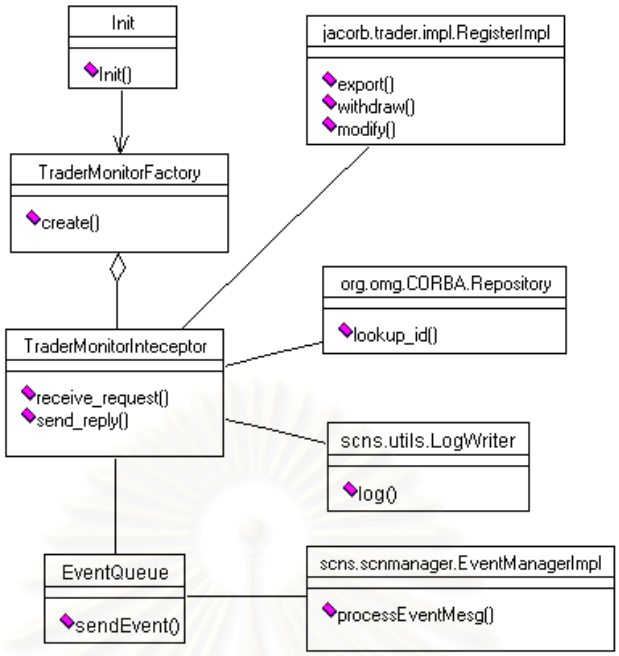
ส่วนเพิ่มขยายบริการเทรดเดอร์ประกอบด้วย 2 ส่วน คือ ตัวตรวจบริการเทรดเดอร์และมอดูลสนับสนุนการเปลี่ยนแปลงบริการ โดยมีแบบจำลองแพคเกจแสดงดังรูปที่ 4.1 ตัวตรวจบริการเทรดเดอร์ได้รับการพัฒนาขึ้นในแพคเกจ scns.TraderMonitorAgent ส่วนมอดูลสนับสนุนการเปลี่ยนแปลงบริการนั้นได้รับการพัฒนาขึ้นในแพคเกจ scns.scsmodule



รูปที่ 4.1 แบบจำลองแพคเกจของส่วนเพิ่มขยายบริการเทรดเดอร์

4.2.1 ต้นแบบตัวตรวจบริการเทรดเดอร์

ตัวตรวจบริการเทรดเดอร์ได้รับการพัฒนาขึ้นในรูปแบบของคอร์บาอินเทอร์เซ็พเตอร์เพื่อดักการเรียกใช้งานตัวกระทำการของบริการเทรดเดอร์ในส่วนที่เกี่ยวข้องกับการเปลี่ยนแปลงบริการ วิทยานิพนธ์นี้เลือกใช้เซิร์ฟเวอร์อินเทอร์เซ็พเตอร์ของวิสิโบรกเกอร์ โดยผูกเข้ากับบริการเทรดเดอร์ของจาคอร์บ จากรูปที่ 4.1 ตัวตรวจบริการเทรดเดอร์ (ภายในแพคเกจ scns.TraderMonitorAgent) ประกอบด้วยคลาส 4 คลาส คือ คลาส TraderMonitorFactory คลาส Init คลาส TraderMonitorInterceptor และคลาส EventQueue โดยมีแบบจำลองคลาสแสดงได้ดังรูปที่ 4.2



รูปที่ 4.2 แบบจำลองคลาสของตัวตรวจบริการเทรดเดอร์

รายละเอียดคลาสของตัวตรวจบริการเทรดเดอร์สามารถอธิบายได้ดังนี้

- คลาส TraderMonitorFactory

เป็นคลาสที่จำเป็นสำหรับวิสิโบรกเกอร์ โดยมีตัวกระทำการที่สำคัญคือ ตัวกระทำการ create() ใช้สำหรับสร้างวัตถุของ TraderMonitorInterceptor ซึ่งทำหน้าที่เป็นตัวตรวจบริการเทรดเดอร์ การสร้างวัตถุของ TraderMonitorInterceptor จะใช้การสร้างในรูปแบบ Per Client

- คลาส Init

เป็นคลาสที่จำเป็นสำหรับวิสิโบรกเกอร์ใช้สำหรับการเริ่มต้นสร้างวัตถุ TraderMonitor Factory และกำหนดชื่ออ้างอิงสำหรับบริการเทรดเดอร์เพื่อให้สามารถค้นหาบริการเทรดเดอร์ได้จากตัวกระทำการ resolve_initial_references() สำหรับรายละเอียดสามารถดูได้ที่ [16]

- คลาส TraderMonitorInterceptor

ทำหน้าที่ดักการเรียกใช้งานตัวกระทำการของบริการเทรดเดอร์ โดยได้รับการพัฒนาขึ้นในรูปแบบของเซิร์ฟเวอร์อินเทอร์เซ็พเตอร์ ที่มีการสืบทอดมาจากคลาส com.visigenic.vbroker.DefaultServerInterceptor ของวิสิโบรกเกอร์ [16] วิทยานิพนธ์นี้ได้ทำการเขียนทับ (Override) ตัวกระทำการที่สำคัญ 2 ตัวคือ receive_request() และ send_reply() ซึ่งมีรายละเอียดดังต่อไปนี้

```
public org.omg.CORBA.portable.InputStream receive_request (
 com.visigenic.vbroker.GIOP.RequestHeader    hdr,
 org.omg.CORBA.ObjectHolder                  object,
 org.omg.CORBA.portable.InputStream          buf,
 com.visigenic.vbroker.interceptor.Closure closure)
```

ตัวกระทำการ receive_request() ทำหน้าที่ดักข้อมูลในขณะที่ตัวกระทำการเทรดเดอร์ถูกเรียกใช้งาน โดยชื่อตัวกระทำการที่ถูกเรียกใช้จะเก็บอยู่ในพารามิเตอร์ชื่อ hdr ในส่วนของ hdr.operation ข้อมูลพารามิเตอร์ต่างๆ จากการเรียกใช้งานตัวกระทำการของบริการเทรดเดอร์เก็บอยู่ในพารามิเตอร์ชื่อ buf ซึ่งมีชนิดข้อมูลเป็นแบบ InputStream ของคอร์บา โดยจะสามารถอ่านข้อมูลพารามิเตอร์แต่ละตัวออกมาได้

```
public org.omg.CORBA.portable.OutputStream send_reply (
    com.visigenic.vbroker.GIOP.RequestHeader     reqHdr,
    com.visigenic.vbroker.GIOP.ReplyHeader     hdr,
    org.omg.CORBA.Object                       target,
    org.omg.CORBA.portable.OutputStream        buf,
    org.omg.CORBA.Environment                  env,
    com.visigenic.vbroker.interceptor.Closure closure)
```

ตัวกระทำการ send_reply() เป็นตัวกระทำการที่ถูกเรียกใช้ก่อนที่ผลการทำงานของตัวกระทำการจะถูกส่งกลับให้แก่ผู้เรียกใช้งานตัวกระทำการ วิทยานิพนธ์นี้ทำการเขียนทับตัวกระทำการ send_reply() โดยมีจุดประสงค์เพื่อตรวจสอบว่าตัวกระทำการที่ถูกเรียกใช้งานสามารถทำงานได้สำเร็จหรือไม่โดยอาศัยการตรวจสอบเอ็กซ์เซ็พชันที่จะส่งกลับให้แก่ผู้เรียกใช้งานตัวกระทำการในกรณีที่มีความผิดพลาดจากการทำงานเกิดขึ้น การตรวจสอบค่าเอ็กซ์เซ็พชันนั้นสามารถตรวจได้จากพารามิเตอร์ชื่อ env

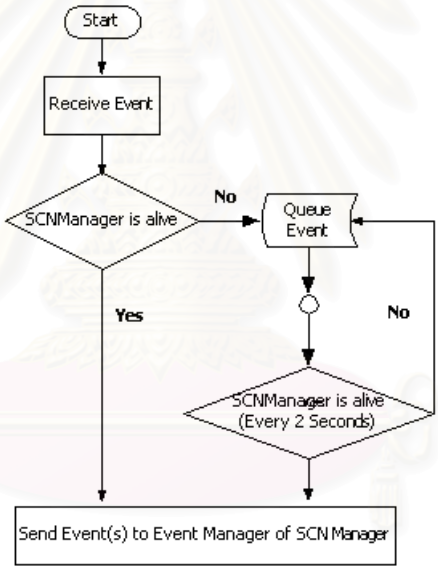
จากรูปที่ 4.2 คลาส TraderMonitorInterceptor จะมีความสัมพันธ์กับคลาสต่างๆ ดังนี้คือ

1. คลาส jacorb.trader.impl.RegisterImpl ซึ่งเป็นคลาสของบริการเทรดเดอร์ โดยมีจุดประสงค์เพื่อสืบค้นข้อมูลของบริการจากบริการเทรดเดอร์ในกรณีที่ข้อมูลเกี่ยวกับบริการที่ได้จากการตรวจสอบตัวกระทำการ receive_request() ไม่เพียงพอ
2. คลาส org.omg.CORBA.Repository ซึ่งเป็นคลาสของคลังส่วนต่อประสาน โดยมีจุดประสงค์เพื่อสืบค้นข้อมูลส่วนต่อประสานของบริการ
3. คลาส scns.utils.LogWriter เพื่อทำการบันทึกการทำงานและข้อผิดพลาดต่างๆ

4. คลาส EventQueue เพื่อส่งผ่านข้อมูลการเปลี่ยนแปลงบริการที่ตรวจสอบได้ไปให้แก่ตัวจัดการการแจ้งการเปลี่ยนแปลงบริการในกรณีที่ไม่พบความผิดพลาดจากการทำงานของตัวกระทำการ

- <u>คลาส EventQueue</u>

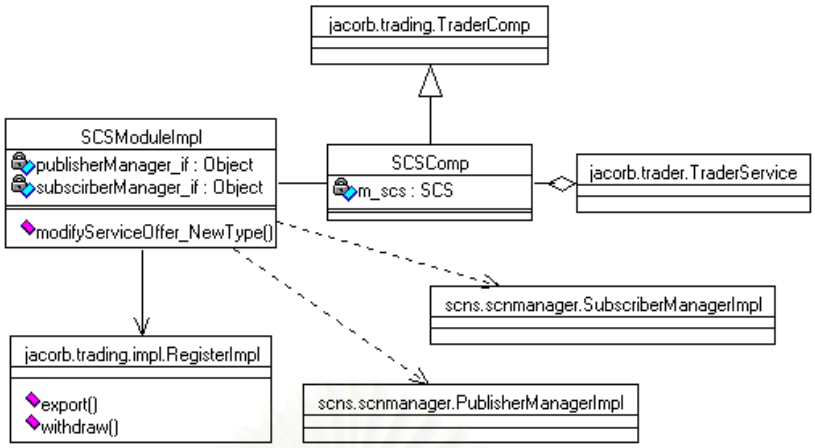
เป็นคลาสที่พัฒนาขึ้นโดยมีจุดประสงค์เพื่อส่งผ่านข้อมูลการเปลี่ยนแปลงบริการที่ตรวจสอบได้โดยตัวตรวจบริการเทรดเดอร์ไปให้แก่ส่วนจัดการเหตุการณ์ (คลาส scns.scnmanager.EventManager) ของตัวจัดการการแจ้งการเปลี่ยนแปลงบริการ นอกจากการส่งผ่านข้อมูลการเปลี่ยนแปลงบริการตามปกติแล้ว คลาส EventQueue ยังทำหน้าที่เก็บรักษาข้อมูลการเปลี่ยนแปลงบริการเอาไว้ในกรณีที่ตรวจพบว่าตัวจัดการการแจ้งการเปลี่ยนแปลงไม่สามารถทำงานได้ ณ เวลาหนึ่งๆ แผนภาพการทำงานเป็นดังรูปที่ 4.3



รูปที่ 4.3 แผนภาพแสดงการทำงานของคลาส EventQueue

4.2.2 ต้นแบบมอดูลสนับสนุนการเปลี่ยนแปลงบริการ

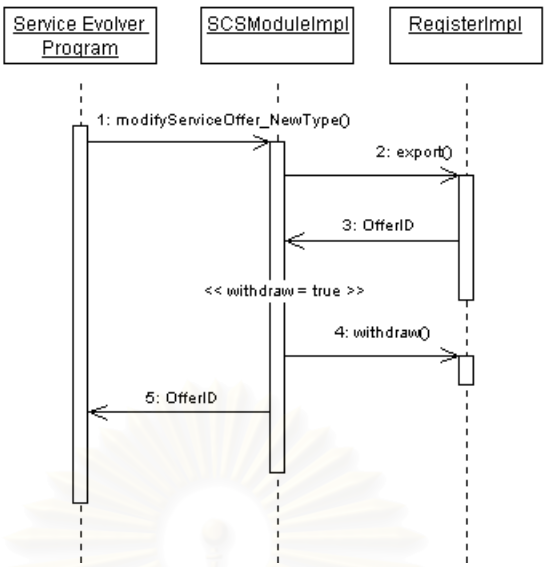
มอดูลสนับสนุนการเปลี่ยนแปลงบริการได้รับการพัฒนาขึ้นเป็นส่วนหนึ่งของบริการเทรดเดอร์โดยมีรายละเอียดของแบบจำลองคลาสดังรูปที่ 4.4



รูปที่ 4.4 แบบจำลองคลาสของมอดูลสนับสนุนการเปลี่ยนแปลงบริการ

จากแบบจำลองคลาสในรูปที่ 4.4 คลาส SCSModuleImpl เป็นคลาสพัฒนาของมอดูลสนับสนุนการเปลี่ยนแปลงบริการ โดยมีคลาส SCSComp ซึ่งสืบทอดมาจากคลาส TraderComp [2] [17] ที่ใช้ในการเก็บข้อมูลอ้างถึงวัตถุของ SCSModuleImpl นอกจากนี้คลาส SCSModuleImpl ยังมีความสัมพันธ์กับคลาส RegisterImpl [2] [17] เพื่อเรียกใช้งานตัวกระทำการ export() สำหรับเพิ่มข้อเสนอบริการ และตัวกระทำการ withdraw()สำหรับยกเลิกข้อเสนอบริการในบริการเทรดเดอร์ และยังมีความสัมพันธ์กับ คลาส PublisherManagerImpl และ SubscriberImpl โดยเก็บข้อมูลอ้างถึงวัตถุของวัตถุทั้ง 2 คลาสเอาไว้สำหรับให้ผู้เปลี่ยนแปลงบริการและผู้ต้องการรับทราบการเปลี่ยนแปลงบริการ สามารถใช้เพื่อติดต่อขอเข้าใช้งานส่วนทั้งสองได้จากบริการเทรดเดอร์

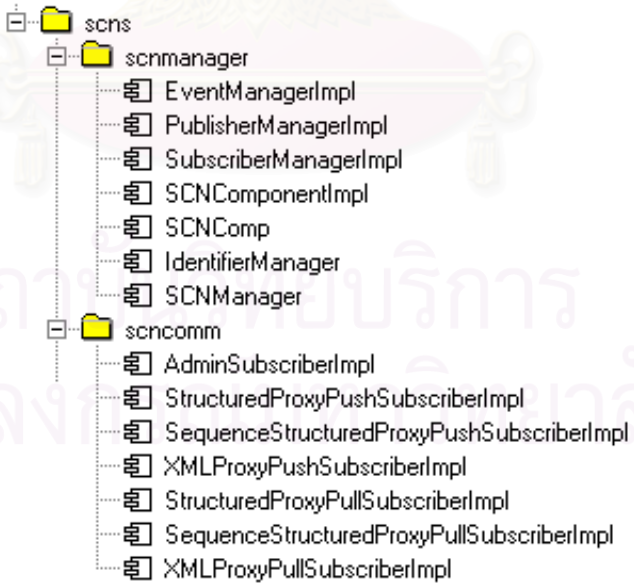
ลำดับขั้นการทำงานเมื่อมีการเรียกใช้งานตัวกระทำการ modifyServiceOfferNewType() ของคลาส SCSModuleImpl (รูปที่ 4.5) เริ่มต้นจากผู้เปลี่ยนแปลงบริการทำการเรียกใช้งานตัวกระทำการ จากนั้นตัวกระทำการนี้จะไปทำการเรียกใช้งานตัวกระทำการ export() ของคลาส RegisterImpl เพื่อทำการเพิ่มข้อเสนอของบริการที่ได้รับการเปลี่ยนแปลงชนิดของบริการเรียบร้อยแล้วเข้าสู่บริการเทรดเดอร์โดยผลที่ได้จากตัวกระทำการ export() คือรหัสข้อเสนอของบริการตัวใหม่ หลังจากนั้นตัวกระทำการจะทำการตรวจสอบค่าพารามิเตอร์ชื่อ withdrawOldOffer โดยหากมีค่าเท่ากับจริง (True) ตัวกระทำการก็จะเรียกตัวกระทำการ withdraw() ของคลาส RegisterImpl เพื่อลบข้อมูลข้อเสนอของบริการตัวเก่าออกจากบริการเทรดเดอร์ อย่างไรก็ตามจากการเรียกใช้งานตัวกระทำการนี้จะส่งผลให้ตัวตรวจบริการเทรดเดอร์สามารถตรวจสอบได้ว่าเกิดการเปลี่ยนแปลงชนิดของบริการแทนที่จะเป็นการเพิ่มบริการใหม่และลบบริการเก่าออกจากระบบจากการเรียกตัวกระทำการ register() และ withdraw() ตามลำดับ



รูปที่ 4.5 แผนภาพลำดับเหตุการณ์เมื่อมีการเรียกใช้งานตัวกระทำการ modifyServiceOffer_NewType()

4.3 ต้นแบบตัวจัดการการแจ้งการเปลี่ยนแปลงบริการ

ในหัวข้อนี้จะอธิบายถึงต้นแบบของตัวจัดการการแจ้งการเปลี่ยนแปลงบริการ แบบจำลองแพคเกจของคลาสต่างๆ ในส่วนนี้สามารถแสดงได้ดังรูปที่ 4.6



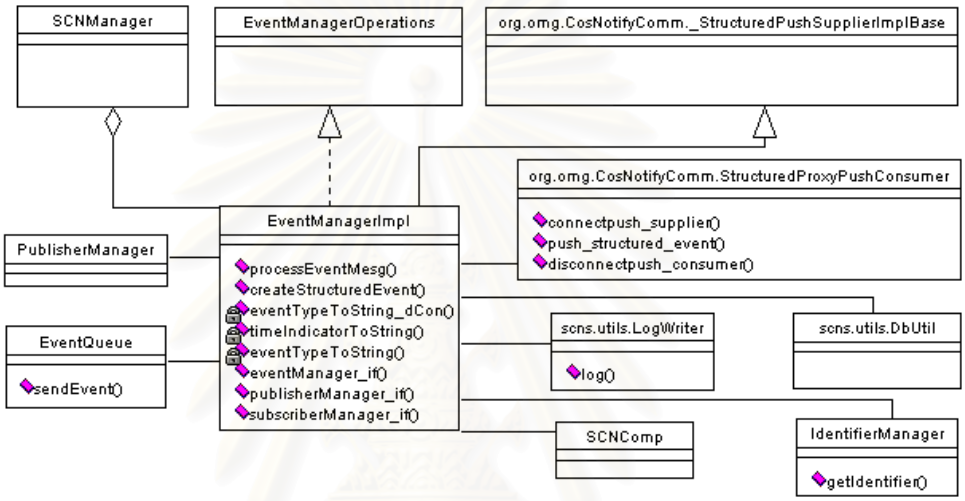
รูปที่ 4.6 แบบจำลองแพคเกจของตัวจัดการการแจ้งการเปลี่ยนแปลง

4.3.1 แพคเกจ scns.scnmanager

คลาสต่างๆ ภายในแพคเกจนี้เป็นคลาสพัฒนาของส่วนย่อยต่างๆ ของตัวจัดการการแจ้งการเปลี่ยนแปลง โดยมีรายละเอียดดังนี้

1. คลาส EventManagerImpl

เป็นคลาสพัฒนาของตัวจัดการเหตุการณ์ มีแบบจำลองคลาสแสดงได้ดังรูปที่ 4.7



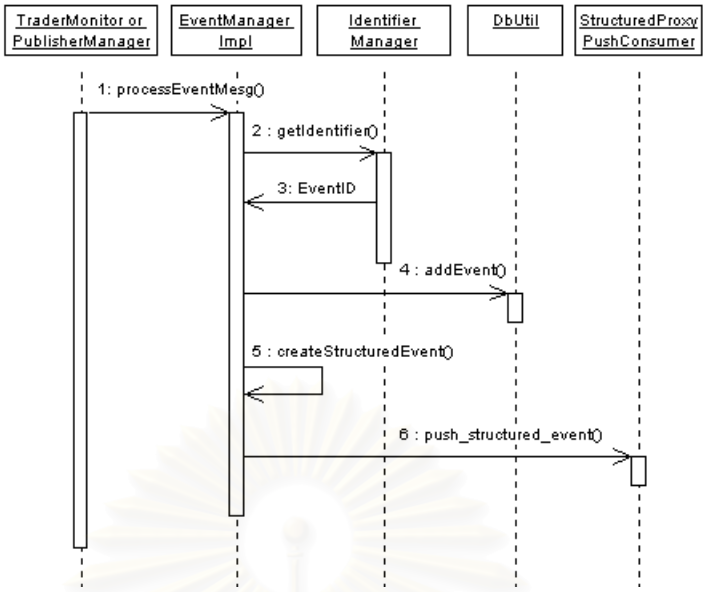
รูปที่ 4.7 แบบจำลองคลาสของตัวจัดการเหตุการณ์

คลาส EventManagerImpl สืบทอดมาจากคลาส _StructuredPushSupplierImpl [5] [18] เพื่อทำหน้าที่เป็นผู้แจ้งเหตุการณ์ (Supplier) สำหรับบริการแจ้งเหตุการณ์ และได้รับการพัฒนาตามส่วนต่อประสานของคลาส EventManagerOperations เพื่อให้บริการตามส่วนต่อประสานของตัวจัดการเหตุการณ์ ตัวกระทำการต่างๆ ของคลาส EventManagerImpl สามารถอธิบายได้ดังนี้

- ตัวกระทำการ processEventMesg() เป็นตัวกระทำการที่พัฒนาขึ้นตามส่วนต่อประสานของตัวจัดการเหตุการณ์ดังที่อธิบายไปแล้วในหัวข้อ 4.1.2
- ตัวกระทำการ createStructuredEvent() เป็นตัวกระทำการส่วนตัว (Private Operation) ที่ใช้สำหรับการแปลงข้อมูลการเปลี่ยนแปลงบริการให้อยู่ในรูปแบบของเหตุการณ์ที่บริการแจ้งเหตุการณ์สามารถเข้าใจได้
- ตัวกระทำการ eventTypeToString() เป็นตัวกระทำการส่วนตัวที่ใช้สำหรับการแปลงชนิดของข้อมูลการเปลี่ยนแปลงบริการให้อยู่ในรูปแบบของสายอักขระ

- ตัวกระทำการ eventTypeToString_dCon() เป็นตัวกระทำการส่วนตัวที่ใช้สำหรับการแปลงชนิดของข้อมูลการเปลี่ยนแปลงบริการให้อยู่ในรูปแบบของสายอักขระ เพื่อใช้งานกับบริการแจ้งเหตุการณ์
- ตัวกระทำการ timeIndicatorToString() เป็นตัวกระทำการส่วนตัวที่ใช้สำหรับการแปลงชนิดข้อมูลบ่งเวลาให้อยู่ในรูปแบบของสายอักขระ
- ตัวกระทำการ eventManager_if() ใช้สำหรับเรียกดูค่าข้อมูลอ้างถึงวัตถุของตัวจัดการเหตุการณ์
- ตัวกระทำการ publisherManager_if() ใช้สำหรับเรียกดูค่าข้อมูลอ้างถึงวัตถุของตัวจัดการผู้แจ้ง
- ตัวกระทำการ subscriberManager_if() ใช้สำหรับเรียกดูค่าข้อมูลอ้างถึงวัตถุของตัวจัดการผู้ต้องการรับทราบการเปลี่ยนแปลงบริการ

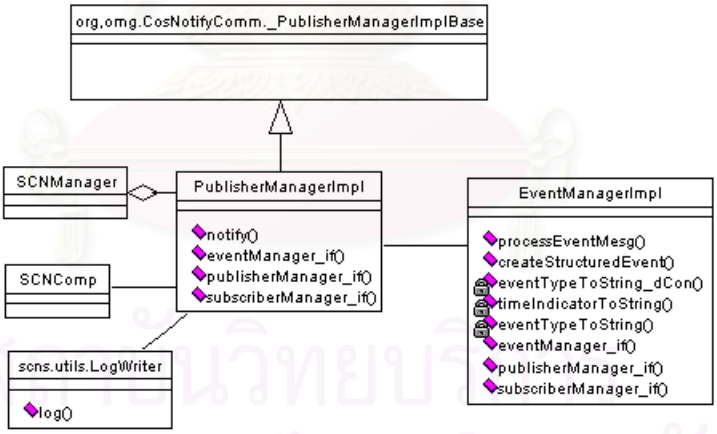
รูปที่ 4.8 แสดงลำดับเหตุการณ์เมื่อตัวตรวจบริการเทรดเดอร์หรือตัวจัดการผู้แจ้งการเปลี่ยนแปลงทำการเรียกใช้ตัวกระทำการ processEventMesg() (1) เพื่อส่งข้อมูลการเปลี่ยนแปลงบริการที่ตรวจสอบได้หรือที่ได้รับมาจากผู้เปลี่ยนแปลงบริการไปให้แก่ตัวจัดการเหตุการณ์ ตัวกระทำการจะร้องขอรหัสของข้อมูลการเปลี่ยนแปลงบริการจากตัวกระทำการ getIdentifier () ของวัตถุ IdentifierManager (2,3) และทำการเรียกตัวกระทำการ addEvent() ของวัตถุ DbUtil เพื่อจัดเก็บข้อมูลการเปลี่ยนแปลงบริการลงในฐานข้อมูล (4) หลังจากนั้นตัวกระทำการนี้จะทำการเรียกใช้งานตัวกระทำการ createStructuredEvent() เพื่อจัดสร้างเหตุการณ์ในรูปแบบที่บริการแจ้งเหตุการณ์สามารถเข้าใจได้ (5) ในขั้นตอนสุดท้าย ตัวกระทำการนี้ทำการจัดส่งเหตุการณ์ดังกล่าวเข้าสู่บริการแจ้งเหตุการณ์ผ่านตัวกระทำการ push_structured_event() ของวัตถุ StructuredProxyPushConsumer (6) เป็นต้น



รูปที่ 4.8 แผนภาพลำดับเหตุการณ์เมื่อมีการเรียกใช้งานตัวกระทำการ processEventMesg()

2. <u>คลาส PublisherManagerImpl</u>

เป็นคลาสพัฒนาของตัวจัดการผู้แจ้งการเปลี่ยนแปลงบริการ โดยมีแบบจำลองคลาสดังรูปที่ 4.9

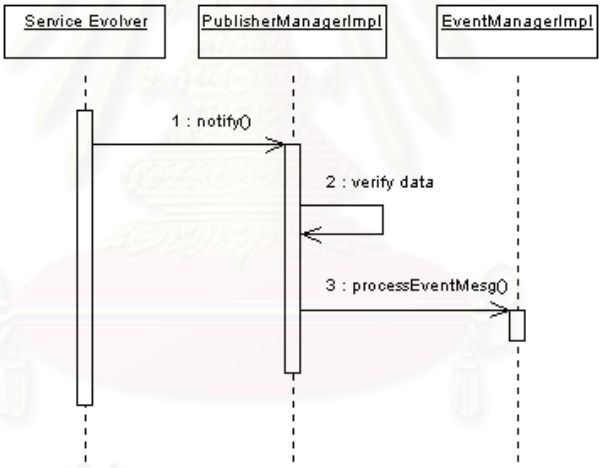


รูปที่ 4.9 แบบจำลองคลาสของตัวจัดการผู้แจ้ง

คลาส PublisherManagerImpl สืบทอดมาจากคลาส _PublisherManagerImplBase ซึ่งเป็นเซิร์ฟเวอร์สเกเลตันของส่วนต่อประสาน PublisherManager (หัวข้อ 4.1.2) โดยมีการติดต่อกับคลาส EventManagerImpl เพื่อส่งข้อมูลการเปลี่ยนแปลงบริการล่วงหน้าที่ได้รับมาจากผู้เปลี่ยนแปลงบริการ โดยประกอบด้วยตัวกระทำการดังต่อไปนี้

- ตัวกระทำการ notify() เป็นตัวกระทำการที่ผู้เปลี่ยนแปลงบริการใช้ในการแจ้งการเปลี่ยนแปลงล่วงหน้า รายละเอียดของพารามิเตอร์ต่างๆ กล่าวไว้ในหัวข้อ 4.1.2
- ตัวกระทำการ eventManager_if() ใช้สำหรับเรียกดูค่าข้อมูลอ้างถึงวัตถุของตัวจัดการเหตุการณ์
- ตัวกระทำการ publisherManager_if() ใช้สำหรับเรียกดูค่าข้อมูลอ้างถึงวัตถุของตัวจัดการผู้แจ้งการเปลี่ยนแปลงบริการ
- ตัวกระทำการ subscriberManager_if() ใช้สำหรับเรียกดูค่าข้อมูลอ้างถึงวัตถุของตัวจัดการผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ

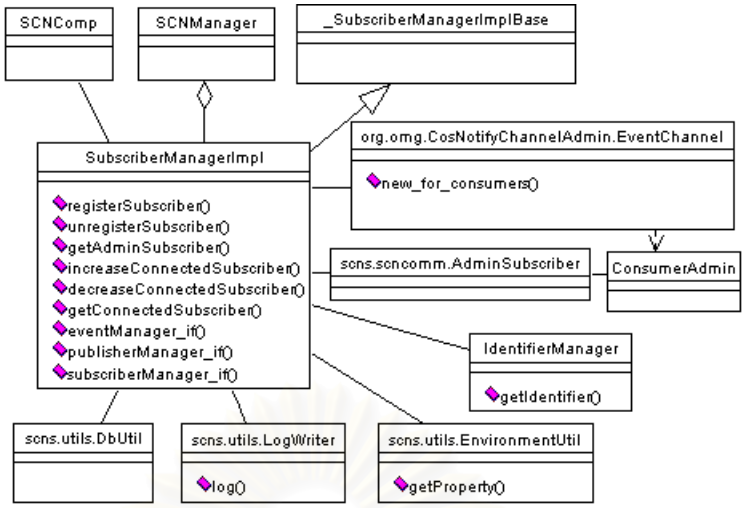
รูปที่ 4.10 แสดงลำดับเหตุการณ์เมื่อผู้เปลี่ยนแปลงบริการทำการเรียกใช้งานตัวกระทำการ notify() เพื่อส่งข้อมูลการเปลี่ยนแปลงบริการล่วงหน้า เมื่อผู้เปลี่ยนแปลงบริการเรียกตัวกระทำการ notify()ตัวกระทำการจะตรวจสอบความถูกต้องของข้อมูล หากไม่พบความผิดพลาดก็จะส่งข้อมูลดังกล่าวไปให้แก่ตัวจัดการเหตุการณ์โดยการเรียกตัวกระทำการ processEventMesg() ของวัตถุ EventManagerImpl



รูปที่ 4.10 แผนภาพลำดับเหตุการณ์เมื่อมีการเรียกใช้งานตัวกระทำการ notify()

3. คลาส SubscriberManagerImpl

เป็นคลาสพัฒนาของตัวจัดการผู้ต้องการรับทราบการเปลี่ยนแปลงบริการ มีแบบจำลองคลาสดังรูปที่ 4.11

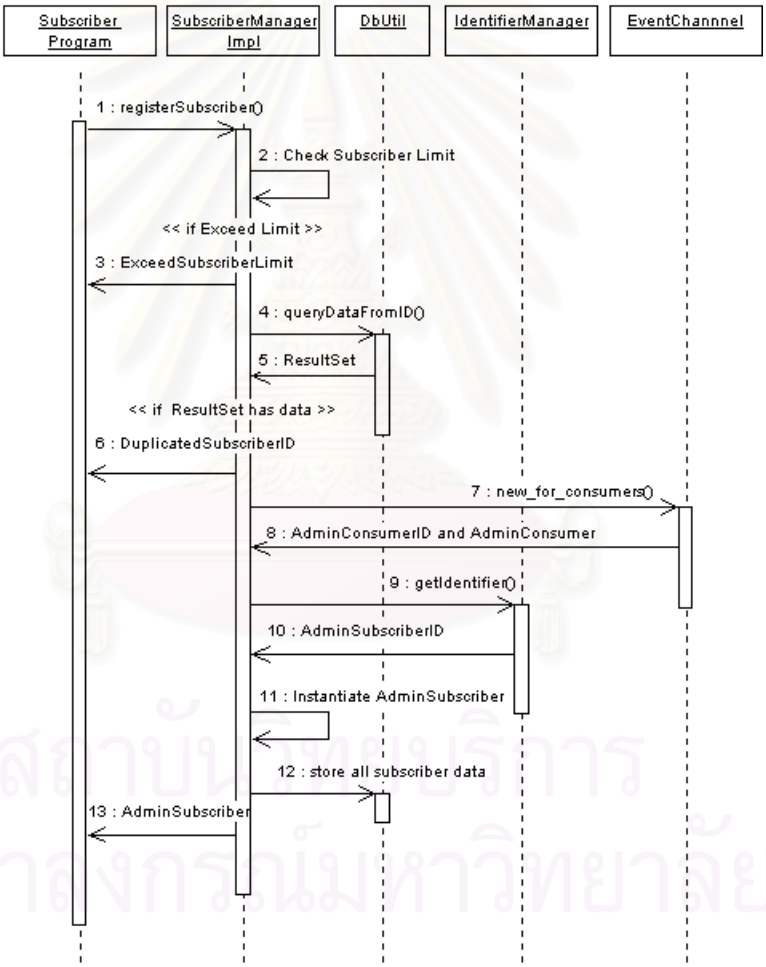


รูปที่ 4.11 แบบจำลองคลาสของตัวจัดการผู้ต้องการรับทราบการเปลี่ยนแปลงบริการ

คลาส SubscriberManagerImpl สืบทอดมาจากคลาส _SubscriberManagerImpl Base ซึ่งเป็นเซิร์ฟเวอร์สเกเลตันของส่วนต่อประสาน SubscriberManager (หัวข้อ 4.1.2) โดยติดต่อกับวัตถุ EventChannel [5] [18] ของบริการแจ้งเหตุการณ์เพื่อสร้างหรือทำลายวัตถุ AdminConsumer [5] [18] สำหรับผู้ต้องการรับทราบการเปลี่ยนแปลงบริการแต่ละราย วัตถุ AdminConsumer นี้ขจะถูกใช้งานโดยวัตถุผู้ดูแล สำหรับการจัดสร้างตัววัตถุ ProxySupplier [5] [18] ซึ่งใช้ในการลงทะเบียนตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ ข้อมูลระบุความต้องการ และค่าคุณสมบัติคุณภาพของบริการ เข้ากับบริการแจ้งเหตุการณ์ ตัวกระทำการของคลาส SubscriberManagerImpl ประกอบด้วย

- ตัวกระทำการ registerSubscriber() เป็นตัวกระทำการที่ผู้ต้องการรับทราบการเปลี่ยนแปลงบริการใช้ในการลงทะเบียนตัวเองเข้ากับบริการเอสซีเอ็น รายละเอียดของพารามิเตอร์ต่างๆ กล่าวไว้แล้วในหัวข้อ 4.1.2
- ตัวกระทำการ unregisterSubscriber() เป็นตัวกระทำการที่ผู้ต้องการรับทราบการเปลี่ยนแปลงบริการใช้ในการยกเลิกการขอเข้าใช้งานบริการเอสซีเอ็น รายละเอียดของพารามิเตอร์ต่างๆ กล่าวไว้แล้วในหัวข้อ 4.1.2
- ตัวกระทำการ getAdminSubscriber() เป็นตัวกระทำการที่ผู้ต้องการรับทราบการเปลี่ยนแปลงบริการใช้ในการขอข้อมูลอ้างถึงวัตถุของวัตถุผู้ดูแลที่ได้มาจากการลงทะเบียน รายละเอียดของพารามิเตอร์ต่างๆ กล่าวไว้แล้วในหัวข้อ 4.1.2
- ตัวกระทำการ increaseConnectedSubscriber() เป็นตัวกระทำการที่ใช้งานโดยตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงสำหรับการเพิ่มจำนวนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการที่ติดต่อกับบริการเอสซีเอ็น ณ เวลาหนึ่งๆ

- ตัวกระทำการ decreaseConnectedSubscriber() เป็นตัวกระทำการที่ใช้งานโดยตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงสำหรับการลดจำนวนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการที่ติดต่อกับบริการเอสซีเอ็น ณ เวลาหนึ่งๆ
- ตัวกระทำการ getConnectedSubscriber() เป็นตัวกระทำการที่ใช้ขอดูจำนวนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการทั้งหมดที่ติดต่ออยู่กับบริการเอสซีเอ็น ณ เวลาหนึ่งๆ
- ตัวกระทำการ eventManager_if() ใช้เรียกดูค่าข้อมูลอ้างถึงวัตถุของตัวจัดการเหตุการณ์
- ตัวกระทำการ publisherManager_if() ใช้เรียกดูค่าข้อมูลอ้างถึงวัตถุของตัวจัดการผู้แจ้ง
- ตัวกระทำการ subscriberManager_if() ใช้เรียกดูค่าข้อมูลอ้างถึงวัตถุของตัวจัดการผู้ต้องการรับทราบการเปลี่ยนแปลงบริการ



รูปที่ 4.12 แผนภาพลำดับเหตุการณ์เมื่อมีการเรียกใช้งานตัวกระทำการ registerSubscriber()

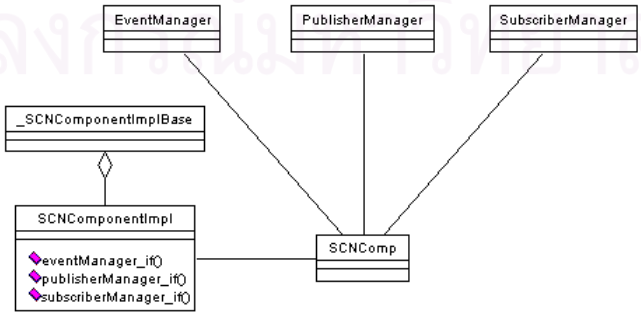
จากรูปที่ 4.12 แสดงลำดับเหตุการณ์เมื่อผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการทำการลงทะเบียนเพื่อขอใช้งานบริการเอสซีเอ็น เริ่มต้นจากผู้ต้องการรับทราบการเปลี่ยนแปลงเรียกใช้งานตัวกระทำการ registerSubscriber() (1) ของวัตถุ SubscriberManagerImpl จากนั้นตัวกระทำการ registerSubscriber() จะทำการตรวจสอบว่าจำนวนผู้ต้องการรับทราบการเปลี่ยนแปลงบริการที่ลงทะเบียนไว้มีค่าเท่ากับค่าของ MaxSubscriber หรือไม่ (2) หากพบว่ามีค่าเท่ากับ MaxSubscriber ตัวกระทำการจะส่งเอ็กซ์เซ็พชัน ExceedSubscriberLimit คืนให้กับผู้ที่ทำการลงทะเบียน (3) และหากมีค่าไม่เท่ากับ MaxSubscriber ตัวกระทำการตรวจสอบค่าของรหัสของผู้ที่ต้องการรับทราบการเปลี่ยนแปลงว่าซ้ำกับของผู้อื่นหรือไม่ (4,5) โดยหากซ้ำ ตัวกระทำการส่งเอ็กซ์เซ็พชัน DuplicatedSubscriberID คืนกลับให้แก่ผู้ลงทะเบียน (6) แต่หากไม่พบการซ้ำกันของรหัสตัวกระทำการจะติดต่อกับบริการแจ้งเหตุการณ์ (EventChannel) เพื่อสร้างวัตถุ AdminConsumer โดยการเรียกตัวกระทำการ new_for_consumers() (7, 8) จากนั้นตัวกระทำการจะทำการร้องขอรหัสของ AdminSubscriber และทำการสร้างวัตถุ AdminSubscriber ขึ้นสำหรับผู้ต้องการรับทราบการเปลี่ยนแปลงที่มาทำการลงทะเบียน (9, 10, 11) ข้อมูลของผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการทั้งหมดจะถูกจัดเก็บในฐานข้อมูล (12) จากนั้นตัวกระทำการจะส่งข้อมูลอ้างถึงวัตถุของ AdminSubscriber คืนให้แก่ผู้ต้องการรับทราบการเปลี่ยนแปลงบริการเพื่อใช้งานต่อไป (13)

4. <u>คลาส IdentifierManager</u>

เป็นคลาสที่ใช้สำหรับกำหนดรหัสให้กับข้อมูลการเปลี่ยนแปลงบริการและรหัสของวัตถุผู้ดูแล ประกอบด้วยตัวกระทำการเพียงตัวเดียวคือ ตัวกระทำการ getIdentifer()

5. <u>คลาส SCNComp และ SCNComponentImpl</u>

รูปที่ 4.13 แสดงแบบจำลองคลาสของคลาส SCNComp และ คลาส SCNComponentImpl



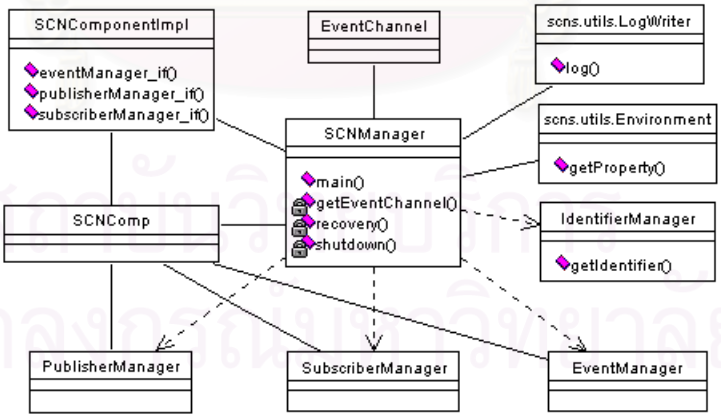
รูปที่ 4.13 แบบจำลองคลาสของ SCNComponentImpl และ SCNComp

คลาส SCNComp เป็นคลาสที่ใช้สำหรับกำหนดและร้องขอข้อมูลอ้างถึงวัตถุของวัตถุ EventManager วัตถุ PublisherManager และวัตถุ SubscriberManager ส่วนคลาส SCNComponentImpl ถูกเรียกใช้โดยผู้ใช้งานบริการเอสซีเอ็นสำหรับการร้องขอข้อมูลอ้างถึงวัตถุของวัตถุทั้ง 3 ตัว

6. <u>คลาส SCNManager</u>

คลาส SCNManager เป็นคลาสที่ใช้สำหรับเริ่มต้นการทำงานของบริการเอสซีเอ็น โดยมีแบบจำลองคลาสแสดงดังรูปที่ 4.14 ประกอบด้วยตัวกระทำการที่สำคัญคือ ตัวกระทำการ main() ซึ่งการทำงานที่สำคัญ แบ่งออกเป็น 3 ส่วนด้วยกัน ขึ้นอยู่กับคำสั่งของผู้สั่งเริ่มต้นบริการ ดังนี้คือ

- คำสั่ง reset เป็นการสั่งเริ่มต้นระบบแบบเริ่มต้นใหม่ทั้งหมด โดยมีการลบข้อมูลในฐานข้อมูลทั้งหมดก่อนที่จะมีการเริ่มต้นระบบใหม่
- คำสั่ง start เป็นการสั่งเริ่มต้นระบบโดยไม่มีการรีเซ็ตระบบใหม่ แต่จะมีการคืนสภาพระบบให้อยู่ในสภาพเดิมก่อนที่บริการจะถูกสั่งให้หยุดการทำงาน ตัวกระทำการที่สำคัญสำหรับการคืนสภาพระบบคือ ตัวกระทำการ recovery()
- คำสั่ง stop เป็นการสั่งยกเลิกการทำงาน เมื่อบริการเอสซีเอ็นถูกสั่งยกเลิกการทำงาน ข้อมูลสถานะทั้งหมดในขณะนั้นจะได้รับการจัดเก็บลงในฐานข้อมูล ตัวกระทำการที่สำคัญคือ ตัวกระทำการ shutdown()



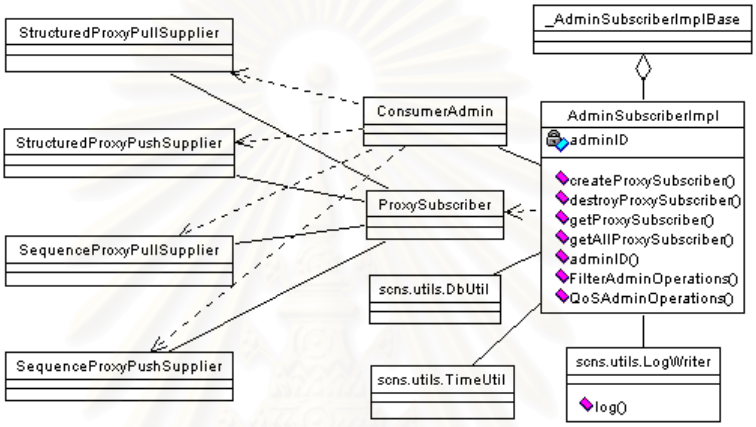
รูป 4.14 แบบจำลองคลาสของ SCNManager

4.3.2 แพคเกจ scns.scncomm

คลาสต่างๆ ภายในแพคเกจนี้เป็นคลาสพัฒนาของผู้ดูแลและตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ

1. คลาส AdminSubscriberImpl

เป็นคลาสพัฒนาของส่วนต่อประสาน AdminSubscriber มีแบบจำลองคลาสดังรูปที่ 4.15



รูปที่ 4.15 แบบจำลองคลาสของ AdminSubscriberImpl

คลาส AdminSubscriberImpl สืบทอดมาจาก คลาส _AdminSubscriberImplBase ซึ่งเป็นเซิร์ฟเวอร์สเกเลตันของส่วนต่อประสาน AdminSubscriber โดยมีการติดต่อกับวัตถุ ConsumerAdmin ของบริการแจ้งเหตุการณ์ เพื่อทำการสร้างวัตถุ ProxySupplier สำหรับรับข้อมูลการเปลี่ยนแปลงบริการ ทั้งนี้ประเภทของ ProxySupplier ที่ทำการสร้างจะขึ้นอยู่กับประเภทของตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ ดังตารางที่ 4.1

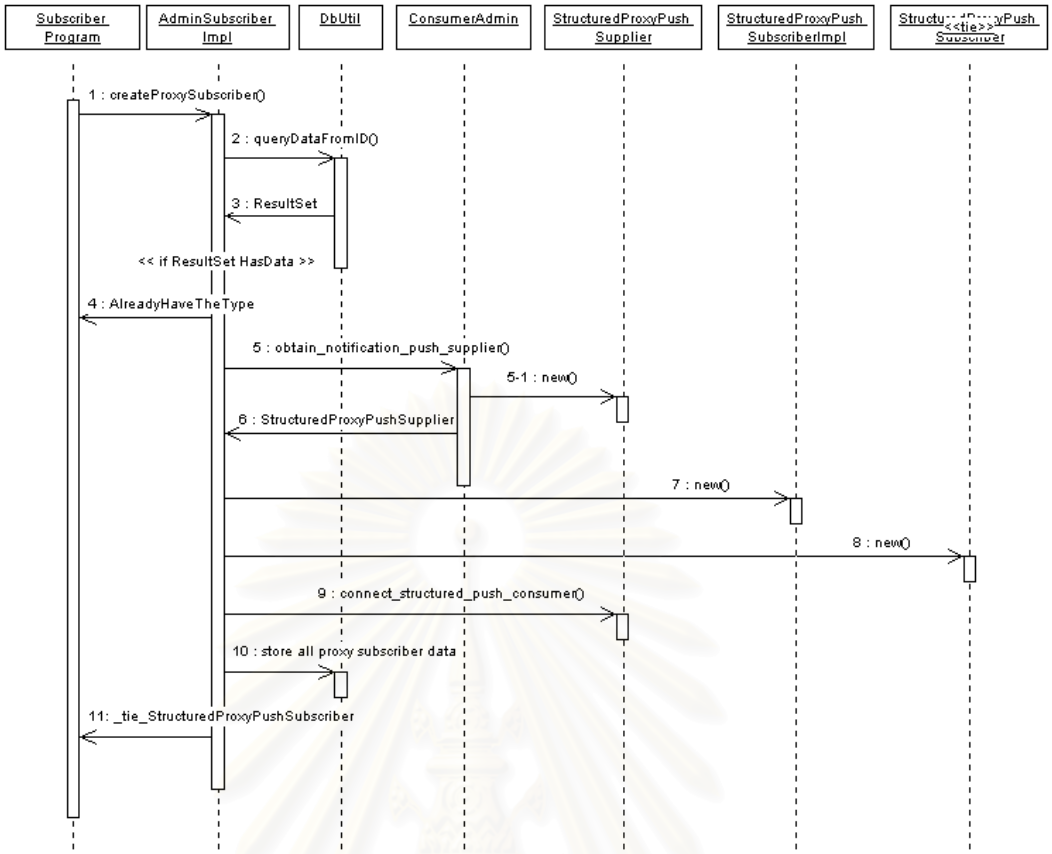
ตารางที่ 4.1 ProxySupplier ที่สร้างขึ้นตามประเภทของ ProxySubscriber

ProxySubscriber	ProxySupplier
StructuredProxyPushSubscriber	StructuredProxyPushSupplier
SequenceStructuredProxyPushSubscriber	SequenceProxyPushSupplier
XMLProxyPushSubscriber	StructuredProxyPushSupplier
StructuredProxyPullSubscriber	StructuredProxyPullSupplier
SequenceStructuredProxyPullSubscriber	SequenceProxyPullSupplier
XMLProxyPullSubscriber	StrucuuredProxyPullSupplier

คลาส AdminSubscriber ประกอบด้วยตัวกระทำการดังต่อไปนี้

- ตัวกระทำการ createProxySubscriber() เป็นตัวกระทำการที่ผู้ต้องการรับทราบการเปลี่ยนแปลงบริการใช้ในการสร้างตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลง รายละเอียดของพารามิเตอร์ต่างๆ กล่าวไว้แล้วในหัวข้อ 4.1.3
- ตัวกระทำการ destroyProxySubscriber() เป็นตัวกระทำการที่ผู้ต้องการรับทราบการเปลี่ยนแปลงบริการใช้ในการทำลายตัวแทนละเอียดของพารามิเตอร์ต่างๆ กล่าวไว้แล้วในหัวข้อ 4.1.3
- ตัวกระทำการ getProxySubscriber() เป็นตัวกระทำการที่ใช้สำหรับร้องขอข้อมูลอ้างถึงวัตถุของตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ โดยระบุรหัสของตัวแทนที่ต้องการ รายละเอียดของพารามิเตอร์ต่างๆ กล่าวไว้แล้วในหัวข้อ 4.1.3
- ตัวกระทำการ getAllProxySubscriber() เป็นตัวกระทำการที่ใช้สำหรับร้องขอข้อมูลอ้างถึงวัตถุของตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการทั้งหมดที่สร้างขึ้น รายละเอียดของพารามิเตอร์ต่างๆ กล่าวในหัวข้อ 4.1.3
- ตัวกระทำการ adminID() เป็นตัวกระทำการที่ใช้สำหรับขอดูข้อมูลรหัสของผู้ดูแลตัวแทน
- ตัวกระทำการ QoSAdminOperations เป็นกลุ่มของตัวกระทำการตามส่วนต่อประสานของส่วนจัดการคุณภาพบริการ สำหรับให้ผู้ที่ต้องการรับทราบการเปลี่ยนแปลงใช้ในการกำหนดคุณภาพของบริการให้แก่วัตถุผู้ดูแลตัวแทน โดยคุณภาพของบริการที่กำหนดให้กับวัตถุผู้ดูแลตัวแทนนี้จะมีผลต่อไปยังวัตถุตัวแทนที่กำลังจะสร้างขึ้น สำหรับรายละเอียดของตัวกระทำการกล่าวในหัวข้อ 4.1.3
- ตัวกระทำการ FilterAdminOperations() เป็นกลุ่มของตัวกระทำการตามส่วนต่อประสานของส่วนจัดการข้อมูลระบุความต้องการ สำหรับให้ผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการใช้ในการกำหนดข้อมูลระบุความต้องการ โดยข้อมูลระบุความต้องการที่กำหนดให้แก่วัตถุผู้ดูแลนี้จะมีความสัมพันธ์กับข้อมูลระบุความต้องการที่กำหนดให้กับวัตถุตัวแทนแบบออร์ (OR) สำหรับรายละเอียดของตัวกระทำการในกลุ่มนี้ได้กล่าวไว้ในหัวข้อ 4.1.3

รูปที่ 4.16 แสดงลำดับเหตุการณ์เมื่อมีการเรียกใช้งานตัวกระทำการ createProxySubscriber() เพื่อทำการสร้างตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการประเภท StructuredProxyPushSubscriber

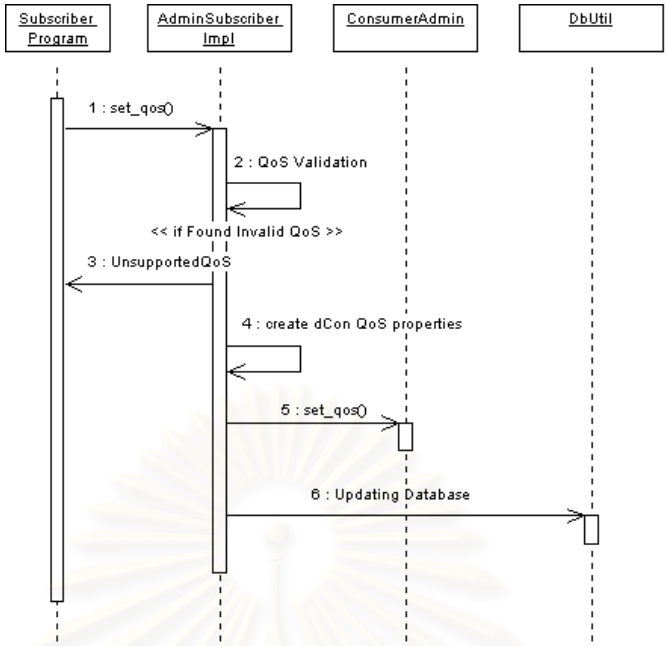


รูปที่ 4.16 แผนภาพลำดับเหตุการณ์ของตัวกระทำการ createProxySubscriber () สำหรับสร้างตัวแทนประเภท StructuredProxyPushSubscriber

ผู้ต้องการรับทราบการเปลี่ยนแปลงบริการจะเรียกตัวกระทำการ createProxy Subscriber() โดยระบุประเภทของตัวแทนเป็นแบบ StructuredProxyPushSubscriber (1) จากนั้นตัวกระทำการเริ่มต้นทำงานโดยตรวจสอบว่าตัวแทนประเภท StructuredProxyPush Subscriber สำหรับผู้ต้องการรับทราบการเปลี่ยนแปลงรายนี้ ได้รับการจัดสร้างแล้วหรือไม่ โดยตรวจสอบจากข้อมูลตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการที่เก็บไว้ในฐานข้อมูล (2, 3) โดยหากพบว่าตัวแทนประเภทนี้ได้รับการจัดสร้างไว้แล้วก็จะส่งเอ็กซ์เซ็พชัน AlreadyHaveThe Type กลับไปให้ (4) แต่หากยังไม่ได้รับการจัดสร้างก็จะการติดต่อกับบริการแจ้งเหตุการณ์ทางวัตถุ ConsumerAdmin เพื่อขอสร้างวัตถุ StructuredProxyPushSupplier โดยการเรียกตัวกระทำการ obtain_notification_push_supplier() (5, 5-1, 6) จากนั้นตัวกระทำการจะสร้างวัตถุ StructuredProxyPushSubscriber ขึ้น โดยทำการสร้างวัตถุของ StructuredProxyPush SubscriberImpl (7) ซึ่งสืบทอดมาจากคลาส org.omg.CosNotifyComm._StructuredPush ConsumerImplBase [5] [18] และพัฒนาตามส่วนต่อประสาน StructuredProxyPush SubscriberOperations และทำการสร้างวัตถุของ _tie_StructuredProxy PushSubscriber (8)

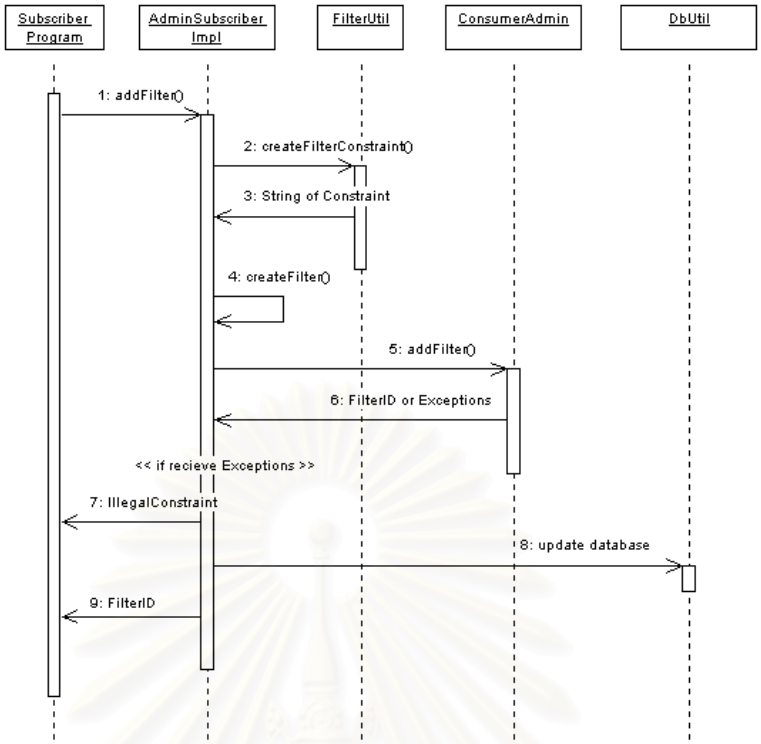
โดยในการสร้างจะต้องให้ข้อมูลอ้างถึงวัตถุของ StructuredProxyPushSubscriberImpl สำหรับคลาส _tie_StructuredProxyPushSubscriber นั้น สืบทอดมาจากคลาสของ StructuredProxy PushSubscriberImplBase ซึ่งเป็นเซิร์ฟเวอร์สเกเลตันของส่วนต่อประสาน StructuredProxy PushSubscriber ในขั้นตอนต่อไปเป็นการลงทะเบียนวัตถุ StructuredProxyPushSubscriber ImplBase เข้ากับวัตถุของ StructuredProxyPushSupplier ของบริการแจ้งเหตุการณ์ (9) เพื่อขอรับข้อมูลการเปลี่ยนแปลงบริการ จากนั้นทำการเก็บข้อมูลทั้งหมดลงฐานข้อมูล (10) และคืนค่าข้อมูลอ้างถึงวัตถุของ _tie_StructuredProxyPushSubscriber และรหัสของตัวแทนที่สร้างขึ้นให้กับผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ (11)

รูปที่ 4.17 แสดงลำดับเหตุการณ์เมื่อผู้ต้องการรับทราบการเปลี่ยนแปลงบริการทำการเรียกใช้งานตัวกระทำการ set_qos() ของวัตถุ AdminSubscriberImpl (1) เพื่อทำการกำหนดคุณสมบัติคุณภาพของบริการ การทำงานของตัวกระทำการเริ่มต้นจากการตรวจสอบคุณสมบัติที่ระบุมาว่ามีความถูกต้องทั้งชื่อของคุณสมบัติและชนิดของค่าข้อมูลที่ระบุให้กับคุณสมบัตินั้นๆ (2) โดยหากพบข้อผิดพลาดตัวกระทำการก็จะส่งเอ็กซ์เซ็พชัน UnsupportedQoS คืนกลับให้แก่ผู้ต้องการรับทราบการเปลี่ยนแปลงบริการ (3) แต่หากไม่พบความผิดพลาดก็จะทำงานต่อไป โดยนำข้อมูลคุณสมบัติคุณภาพของบริการที่ได้รับมาทำการจัดสร้างให้อยู่ในรูปแบบที่สามารถใช้ได้กับบริการแจ้งเหตุการณ์ (4) เช่น หากเป็นคุณภาพของบริการ PacingInterval ก็จะต้องจัดรูปแบบของค่าของข้อมูลให้อยู่ในรูปแบบของวัตถุ TimeT [14] โดยอาศัยการใช้งานตัวกระทำการ scns.utils. TimeUtil.relativeMillisToTimeT() เป็นต้น จากนั้นทำการระบุคุณภาพของบริการต่อบริการแจ้งเหตุการณ์ผ่านวัตถุ ConsumerAdmin (5) และทำการบันทึกค่าส่วนของคุณภาพของบริการในฐานข้อมูลเป็นขั้นตอนสุดท้าย



รูปที่ 4.17 แผนภาพลำดับเหตุการณ์เมื่อมีการเรียกใช้งานตัวกระทำการ set_qos()

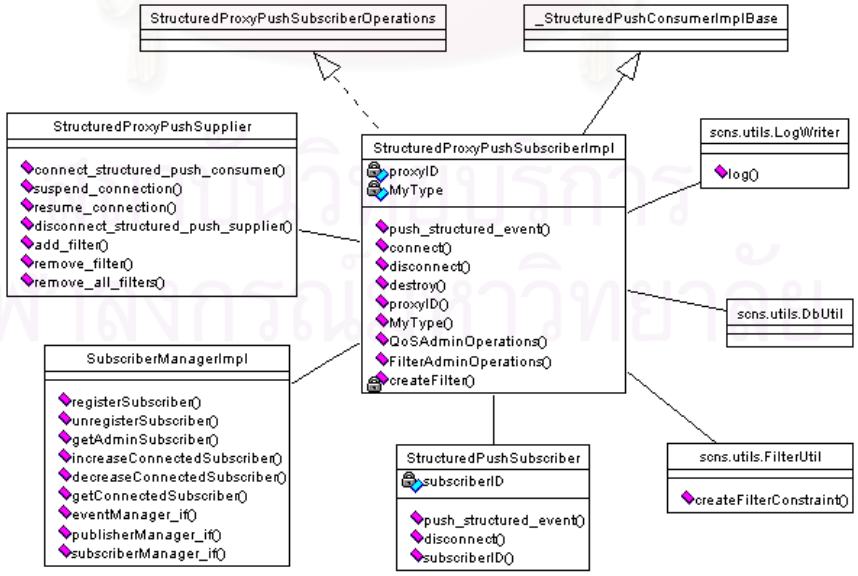
รูปที่ 4.18 แสดงลำดับเหตุการณ์เมื่อผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการทำการเรียกใช้งานตัวกระทำการ addFilter() ของวัตถุ AdminSubscriberImpl เพื่อระบุความต้องการในการรับทราบข้อมูลการเปลี่ยนแปลงบริการ (1) การทำงานของตัวกระทำการเริ่มต้นจากการนำข้อมูลระบุบริการที่ได้มาทำการสร้างเป็นข้อความภาษาบังคับ (Extended Trader Constraint Language) ด้วยการเรียกใช้งานตัวกระทำการ createFilterConstraint() ของวัตถุ FilterUtil (2,3) จากนั้นจึงนำข้อความภาษาบังคับที่ได้มาจัดให้อยู่ในรูปแบบของวัตถุ Filter [5] [18] ที่ใช้งานกับบริการแจ้งเหตุการณ์ (4) ในขั้นตอนต่อไปจะส่งวัตถุ Filter ให้กับบริการแจ้งเหตุการณ์ผ่านวัตถุ ConsumerAdmin โดยเรียกใช้งานตัวกระทำการ addFilter() (5) โดยหากเกิดความผิดพลาดของข้อมูลระบุความต้องการ ตัวกระทำการ addFilter() ก็จะทำการส่งเอ็กซ์เซ็พชันคืนกลับมา (6) และหากได้รับเอ็กซ์เซ็พชันตัวกระทำการก็จะส่งเอ็กซ์เซ็พชัน IllegalConstraint (7) คืนให้กับผู้ต้องการรับทราบการเปลี่ยนแปลง แต่หากไม่พบความผิดพลาดใดๆ ตัวกระทำการ addFilter() ของวัตถุ ConsumerAdmin ก็จะส่งรหัสของวัตถุ Filter คืนกลับมาให้ รหัสที่ได้นี้จะถูกนำไปสร้างเป็นรหัสของข้อมูลระบุความต้องการต่อไป ในขั้นตอนต่อไปจะทำการบันทึกค่าของข้อมูลระบุความต้องการลงยังฐานข้อมูล (8) และส่งค่ารหัสของข้อมูลระบุความต้องการส่งคืนกลับให้แก่ผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ (9)



รูปที่ 4.18 แผนภาพลำดับเหตุการณ์เมื่อมีการเรียกใช้งานตัวกระทำการ addFilter()

2. <u>คลาส StructuredProxyPushSubscriberImpl</u>

เป็นคลาสพัฒนาของส่วนต่อประสาน StructuredProxyPushSubscriber มีแบบจำลองคลาสแสดงดังรูปที่ 4.19



รูปที่ 4.19 แบบจำลองคลาสของ StructuredProxyPushSubscriberImpl

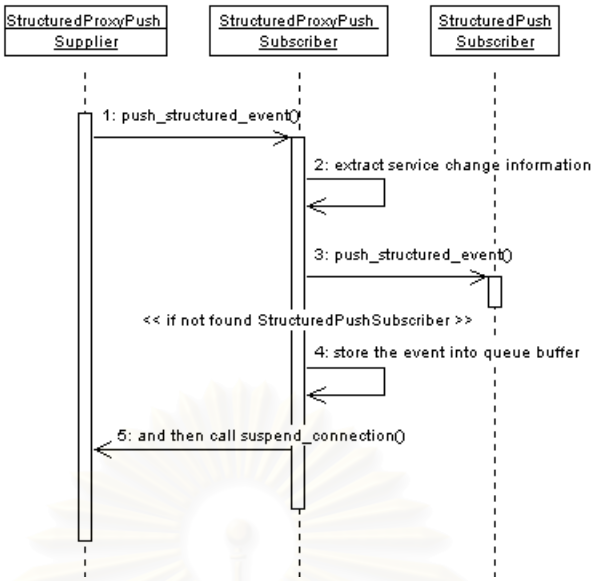
คลาส StructuredProxyPushSubscriberImpl สืบทอดมาจากคลาส _StructuredPushConsumerImplBase ซึ่งเป็นเซิร์ฟเวอร์สเกเลตันของส่วนต่อประสาน StructuredPushConsumer เพื่อทำหน้าที่เป็นผู้รับเหตุการณ์ของบริการแจ้งเหตุการณ์ โดยเหตุการณ์ที่รับจะมีลักษณะเป็นแบบวัตถุเหตุการณ์แบบโครงสร้างที่มีการจัดส่งแบบพุช และพัฒนาตามส่วนต่อประสานของคลาส StructuredProxyPushSubscriberOperations เพื่อทำหน้าที่ให้บริการตามส่วนต่อประสาน StructuredProxyPushSubscriber คลาสนี้ประกอบด้วยตัวกระทำการดังต่อไปนี้

- ตัวกระทำการ push_structured_event() เป็นตัวกระทำการที่บริการแจ้งเหตุการณ์ (StructuredProxyPushSupplier) ใช้ในการส่งเหตุการณ์การเปลี่ยนแปลงบริการมาให้แก่ตัวแทนผู้ต้องการรับทราบการเปลี่ยนแปลงบริการ ข้อมูลการเปลี่ยนแปลงบริการที่อยู่ภายในเหตุการณ์จะถูกส่งต่อไปให้แก่ผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการต่อไป
- ตัวกระทำการ connect() เป็นตัวกระทำการที่ผู้ต้องการรับทราบการเปลี่ยนแปลงใช้ในการติดต่อกับตัวแทน พร้อมทั้งระบุวัตถุเรียกกลับเพื่อให้ตัวแทนใช้ในการส่งข้อมูลการเปลี่ยนแปลงบริการมาให้ รายละเอียดของพารามิเตอร์ต่างๆ กล่าวไว้แล้วในหัวข้อ 4.1.3
- ตัวกระทำการ disconnect() เป็นตัวกระทำการที่ผู้ต้องการรับทราบการเปลี่ยนแปลงใช้ยกเลิกการติดต่อกับตัวแทน รายละเอียดของพารามิเตอร์ต่างๆ กล่าวไว้แล้วในหัวข้อ 4.1.3
- ตัวกระทำการ destroy() เป็นตัวกระทำการที่ผู้ต้องการรับทราบการเปลี่ยนแปลงใช้ในการทำลายตัวแทน รายละเอียดของพารามิเตอร์ต่างๆ กล่าวไว้แล้วในหัวข้อ 4.1.3
- ตัวกระทำการ proxyID() เป็นตัวกระทำการที่ใช้ในการร้องขอรหัสของตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ
- ตัวกระทำการ MyType() เป็นตัวกระทำการที่ใช้ในการร้องขอประเภทของตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ
- ตัวกระทำการ QoSAdminOperations เป็นกลุ่มของตัวกระทำการตามส่วนต่อประสานของส่วนจัดการคุณภาพบริการ สำหรับให้ผู้ที่ต้องการรับทราบการเปลี่ยนแปลงใช้ในการกำหนดคุณภาพของบริการให้แก่ตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ ตัวกระทำการในกลุ่มนี้ประกอบด้วยตัวกระทำการ set_qos () และ get_qos () สำหรับรายละเอียดของตัวกระทำการกล่าวในหัวข้อ 4.1.3
- ตัวกระทำการ FilterAdminOperations() เป็นกลุ่มของตัวกระทำการตามส่วนต่อประสานของส่วนจัดการข้อมูลระบุความต้องการ (FilterAdmin) สำหรับผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการใช้ในการกำหนดข้อมูลระบุความต้องการ โดยข้อมูลระบุความต้องการ

ที่กำหนดนี้จะถูกนำไปออร์ (OR) กับข้อมูลระบุความต้องการที่กำหนดให้แก่วัตถุผู้ดูแลตัวแทน สำหรับรายละเอียดของตัวกระทำการกล่าวในหัวข้อ 4.1.3

- ตัวกระทำการ createFilter() เป็นตัวกระทำการที่ใช้ในการจัดสร้างวัตถุ Filter [5] [18]สำหรับใช้กับบริการแจ้งเหตุการณ์

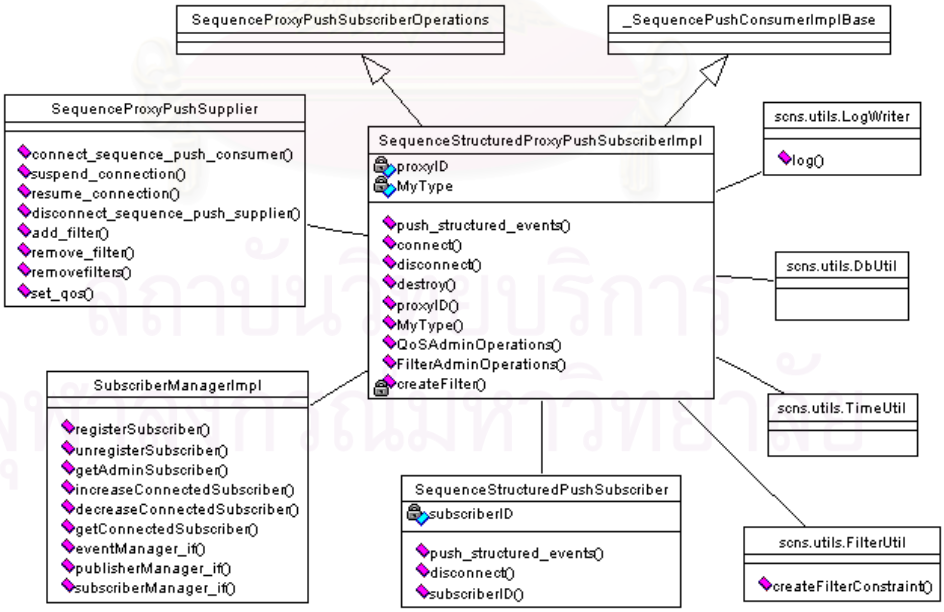
รูปที่ 4.20 แสดงลำดับเหตุการณ์เมื่อวัตถุ StructuredProxyPushSupplier ของบริการแจ้งเหตุการณ์ ทำการส่งเหตุการณ์การเปลี่ยนแปลงที่ตรงกับความต้องการให้กับวัตถุตัวแทน StructuredProxyPushSubscriber โดยการเรียกตัวกระทำการ push_structured_event() (1) จากนั้นการทำงานเริ่มต้นจากวัตถุ StructuredProxyPushSubscriber ทำการแยกเอาข้อมูลการเปลี่ยนแปลงบริการที่แนบมาพร้อมกับเหตุการณ์การเปลี่ยนแปลงบริการ (2) ออกมา โดยข้อมูลที่แยกออกมานี้จะอยู่ในรูปแบบของวัตถุตามส่วนต่อประสาน StructuredEventMessage อยู่แล้ว ข้อมูลการเปลี่ยนแปลงที่ได้มาจะถูกส่งต่อไปให้กับผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการผ่านวัตถุ StructuredPushSubscriber โดยการเรียกตัวกระทำการ push_structured_event() (3) ซึ่งได้รับการกำหนดไว้ตอนนี้ผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการติดต่อกับวัตถุ Structured ProxyPushSubscriber หากไม่พบข้อผิดพลาดใดๆ จากการจัดส่งข้อมูลการเปลี่ยนแปลงบริการก็ถือเป็นการเสร็จสิ้นการทำงาน แต่หากมีข้อผิดพลาดเกิดขึ้นโดยไม่พบวัตถุ StructuredProxyPush Subscriber ที่ต้องการ ก็จะมีการเก็บเหตุการณ์การเปลี่ยนแปลงบริการนั้นเอาไว้เพื่อจัดส่งในคราวหลังเมื่อผู้ที่ต้องการรับทราบการเปลี่ยนแปลงทำการติดต่อเข้ามาใหม่ (4, 5) การเรียกตัวกระทำการ suspend_connection() เป็นการยกเลิกการติดต่อกับวัตถุ StructuredProxyPushSupplier ชั่วคราว โดยหากมีเหตุการณ์การเปลี่ยนแปลงบริการเกิดขึ้นในเวลานี้ วัตถุ StructuredProxyPush Supplier ก็จะทำการเก็บเหตุการณ์นั้นเอาไว้โดยไม่ทำการจัดส่งจนกว่าจะมีการสั่ง resume_ connection() เพื่อติดต่ออีกครั้ง



รูปที่ 4.20 แผนภาพลำดับเหตุการณ์เมื่อบริการแจ้งเหตุการณ์ส่งเหตุการณ์การเปลี่ยนแปลงบริการมาให้แก่วัตถุตัวแทน StructuredProxyPushSubscriber

3. คลาส SequenceStructuredProxyPushSubscriberImpl

เป็นคลาสพัฒนาของส่วนต่อประสาน StructuredProxyPushSubscriber มีแบบจำลองคลาสแสดงดังรูปที่ 4.21



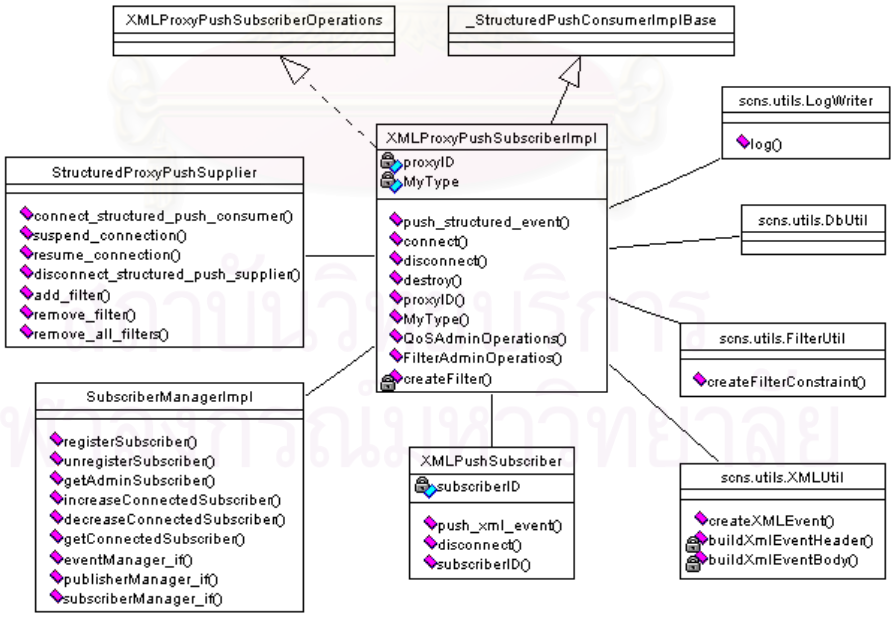
รูปที่ 4.21 แบบจำลองคลาสของ SequenceStructuredProxyPushSubscriberImpl

คลาส SequenceStructuredProxyPushSubscriberImpl สืบทอดมาจากคลาส SequencePushConsumerImplBase ซึ่งเป็นเซิร์ฟเวอร์สเกเลตันของส่วนต่อประสาน SequencePushConsumer เพื่อทำหน้าที่เป็นผู้รับเหตุการณ์ของบริการแจ้งเหตุการณ์ในรูปแบบชุดของวัตถุเหตุการณ์แบบโครงสร้าง ที่มีการจัดส่งแบบพุช และพัฒนาตามส่วนต่อประสานของคลาส SequenceStructuredProxyPushSubscriberOperations เพื่อทำหน้าที่ให้บริการตามส่วนประสาน SequenceStructuredProxyPushSubscriber ตัวกระทำการของคลาสนี้มีความคล้ายคลึงกับตัวกระทำการของคลาส StructuredProxyPushSubscriberImpl แต่มีตัวกระทำการ push_structured_events() ซึ่งจะรับข้อมูลเป็นชุดของเหตุการณ์แทน

สำหรับลำดับเหตุการณ์เมื่อบริการแจ้งเหตุการณ์โดยวัตถุ SequenceProxyPush Supplier ทำการส่งเหตุการณ์ที่เก็บรวบรวมไว้ครบตามจำนวนหรือเวลาที่ระบุไว้มายังวัตถุตัวแทน SequenceStructuredProxyPushSubscriber ก็จะมีลำดับการทำงานที่เหมือนกับของ StructuredProxyPushSubscriber

4. <u>คลาส XMLProxyPushSubscriberImpl</u>

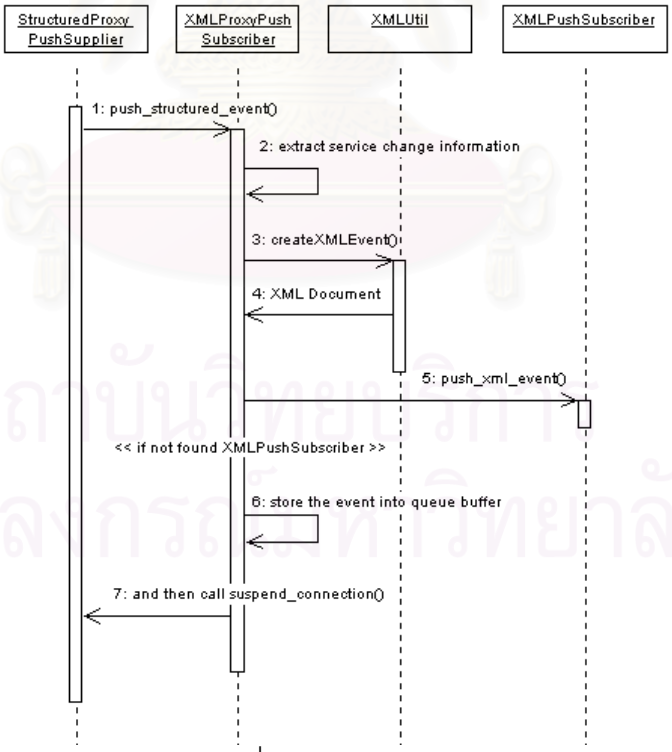
เป็นคลาสพัฒนาของส่วนต่อประสาน XMLProxyPushSubscriber มีแบบจำลองคลาสแสดงดังรูปที่ 4.22



รูปที่ 4.22 แบบจำลองคลาสของ XMLProxyPushSubscriberImpl

คลาส XMLProxyPushSubscriberImpl สืบทอดมาจากคลาส _Structured PushConsumerImplBase ซึ่งเป็นเซิร์ฟเวอร์สเกเลตันของส่วนต่อประสาน StructuredPush Consumer เพื่อทำหน้าที่เป็นผู้รับเหตุการณ์ของบริการแจ้งเหตุการณ์ในแบบวัตถุเหตุการณ์แบบโครงสร้าง ที่มีการจัดส่งแบบพุช และพัฒนาตามส่วนต่อประสานของคลาส StructuredProxy PushSubscriberOperations เพื่อทำหน้าที่ให้บริการตามส่วนประสาน StructuredProxyPush Subscriber ตัวกระทำการของคลาสนี้จะเหมือนกับตัวกระทำการของคลาส StructuredProxy PushSubscriberImpl ดังที่กล่าวแล้วข้างต้น

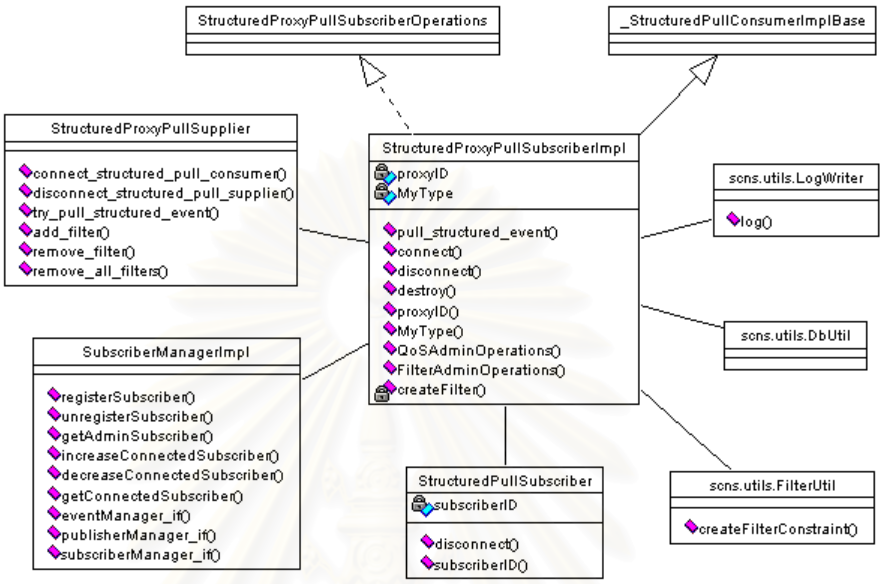
รูปที่ 4.23 แสดงลำดับเหตุการณ์เมื่อวัตถุ StructuredProxyPushSupplier ของบริการแจ้งเหตุการณ์ทำการส่งเหตุการณ์การเปลี่ยนแปลงที่ตรงกับความต้องการให้กับวัตถุตัวแทน XML ProxyPushSubscriber ซึ่งจะเห็นได้ว่ามีความคล้ายคลึงกับกรณีที่วัตถุ StructuredProxyPush Supplier ส่งเหตุการณ์ให้กับ StructuredProxyPushSubscriber (รูปที่ 4.20) แต่จะแตกต่างกันที่ในกรณีนี้ข้อมูลการเปลี่ยนแปลงที่ได้จากขั้นตอนที่ 2 จะถูกนำไปสร้างเป็นเอกสารเอ็กซ์เอ็มแอล โดยการเรียกใช้งานตัวกระทำการ createXMLEvent() ของวัตถุ XMLUtil (3, 4) เอกสารเอ็กซ์เอ็มแอลซึ่งอธิบายข้อมูลการเปลี่ยนแปลงที่ได้ จะถูกจัดส่งไปให้แก่ผู้ที่ต้องการรับทราบการเปลี่ยนแปลง XMLPushSubscriber ต่อไป



รูปที่ 4.23 แผนภาพลำดับเหตุการณ์เมื่อบริการแจ้งเหตุการณ์ส่งเหตุการณ์การเปลี่ยนแปลงบริการมาให้แก่วัตถุตัวแทน XMLProxyPushSubscriber

5. <u>คลาส StructuredProxyPullSubscriberImpl</u>

เป็นคลาสพัฒนาของส่วนต่อประสาน StructuredProxyPullSubscriber มีแบบจำลองคลาสแสดงดังรูปที่ 4.24

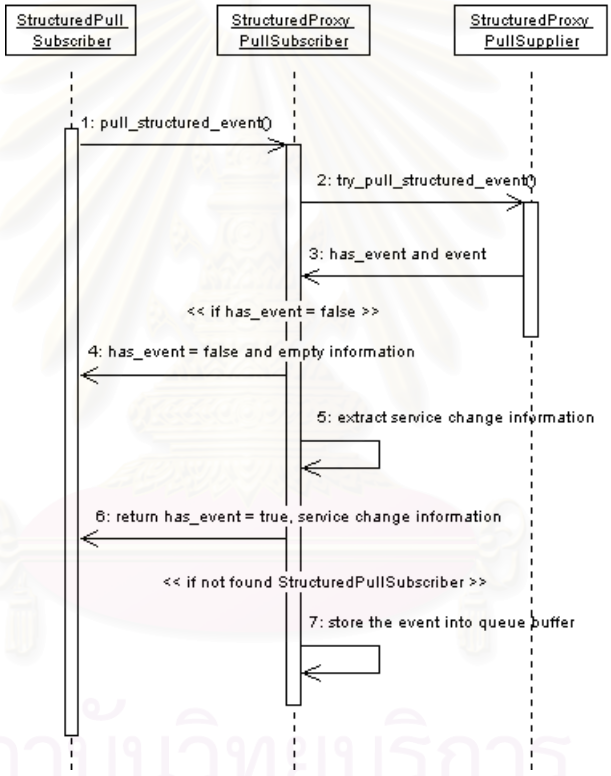


รูปที่ 4.24 แบบจำลองคลาสของ StructuredProxyPullSubscriberImpl

คลาส StructuredProxyPullSubscriberImpl สืบทอดมาจากคลาส _StructuredPullConsumerImplBase ซึ่งเป็นเซิร์ฟเวอร์สเกเลตันของส่วนต่อประสาน StructuredPullConsumer เพื่อทำหน้าที่เป็นผู้รับเหตุการณ์ของบริการแจ้งเหตุการณ์แบบวัตถุเหตุการณ์แบบโครงสร้าง ที่มีการจัดส่งแบบพูล และพัฒนาตามส่วนต่อประสานของคลาส Structured ProxyPullSubscriberOperations เพื่อทำหน้าที่ให้บริการตามส่วนประสาน Structured ProxyPullSubscriber ตัวกระทำการที่สำคัญคือ ตัวกระทำการ pull_structured_event() ซึ่งเป็นตัวกระทำการที่ผู้ต้องการรับทราบการเปลี่ยนแปลงบริการใช้ร้องขอข้อมูลการเปลี่ยนแปลงบริการ (สำหรับรายละเอียดของพารามิเตอร์ต่างๆ สามารถดูได้ที่หัวข้อ 4.1.3)

รูปที่ 4.25 แสดงลำดับเหตุการณ์เมื่อผู้ต้องการรับทราบการเปลี่ยนแปลงบริการเรียกตัวกระทำการ pull_structured_event() เพื่อร้องขอข้อมูลการเปลี่ยนแปลงบริการ (1) โดยการทำงานของตัวกระทำการเริ่มต้นจากการร้องขอเหตุการณ์การเปลี่ยนแปลงบริการไปที่วัตถุ Structured ProxyPullSupplier ของบริการแจ้งเหตุการณ์ โดยเรียกตัวกระทำการ try_pull_structured_event()[5] [18] (2) ซึ่งค่าที่ตัวกระทำการนี้คืนกลับมาประกอบด้วย ค่า has_event ซึ่งมีค่าเป็นแบบตรรกะที่จะบอกว่ามีเหตุการณ์การเปลี่ยนแปลงบริการส่งกลับมาด้วยหรือไม่ พร้อมทั้งเหตุการณ์การเปลี่ยนแปลงบริการ เมื่อได้รับค่าที่คืนกลับมาแล้ว (3) ค่าของ has_event จะได้รับการ

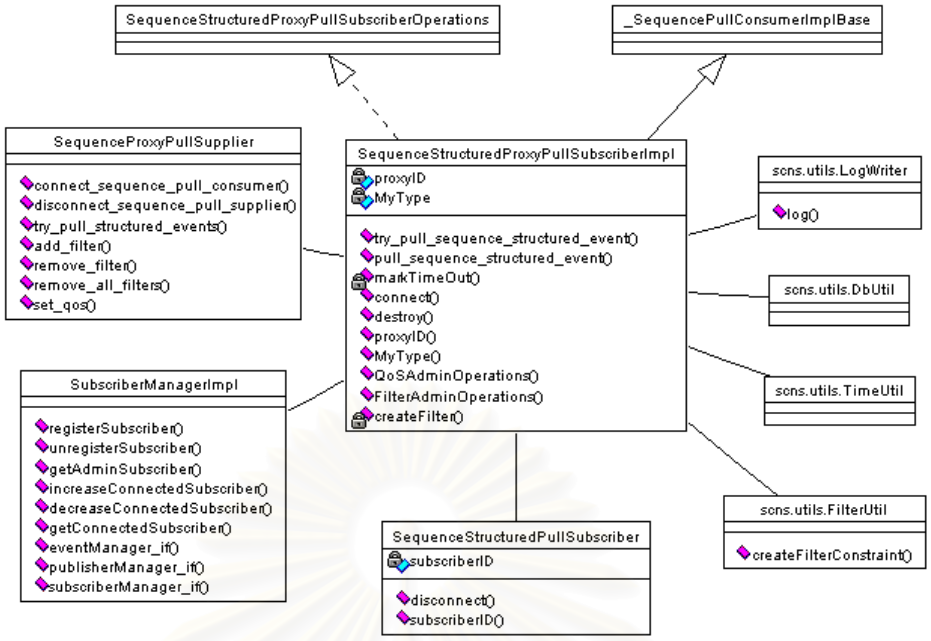
ตรวจสอบโดยหากพบว่ามีค่าเป็นเท็จ นั่นแสดงว่าไม่มีเหตุการณ์การเปลี่ยนแปลงบริการส่งกลับมา ดังนั้นตัวกระทำการจึงคืนค่ากลับให้แก่ผู้ต้องการรับทราบการเปลี่ยนแปลง เป็นค่า has_event ที่มีค่าเท็จ และข้อมูลการเปลี่ยนแปลงบริการที่ว่างเปล่า (4) แต่หากการตรวจสอบพบว่าค่าของ has_event มีค่าเท่ากับจริง ข้อมูลการเปลี่ยนแปลงบริการจะได้รับการแยกออกมาจากเหตุการณ์การเปลี่ยนแปลงบริการ (5) เพื่อใช้ในการจัดส่งให้กับผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการต่อไป (6) และในการจัดส่งหากไม่พบข้อผิดพลาดใดๆ ถือว่าเป็นการเสร็จสิ้นการทำงาน แต่หากในการส่งนั้นมีความผิดพลาดเกิดขึ้นโดยไม่สามารถจัดส่งข้อมูลให้กับผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการได้ เหตุการณ์การเปลี่ยนแปลงบริการนั้นจะได้รับการจัดเก็บไว้สำหรับรอการร้องขอใหม่อีกครั้ง (7)



รูปที่ 4.25 แผนภาพลำดับเหตุการณ์เมื่อผู้ต้องการรับทราบการเปลี่ยนแปลงบริการ StructuredPullSubscriber ร้องขอข้อมูลการเปลี่ยนแปลงบริการ

6. คลาส SequenceStructuredProxyPullSubscriberImpl

เป็นคลาสพัฒนาของส่วนต่อประสาน SequenceStructuredProxyPullSubscriber มีแบบจำลองคลาสแสดงดังรูปที่ 4.26

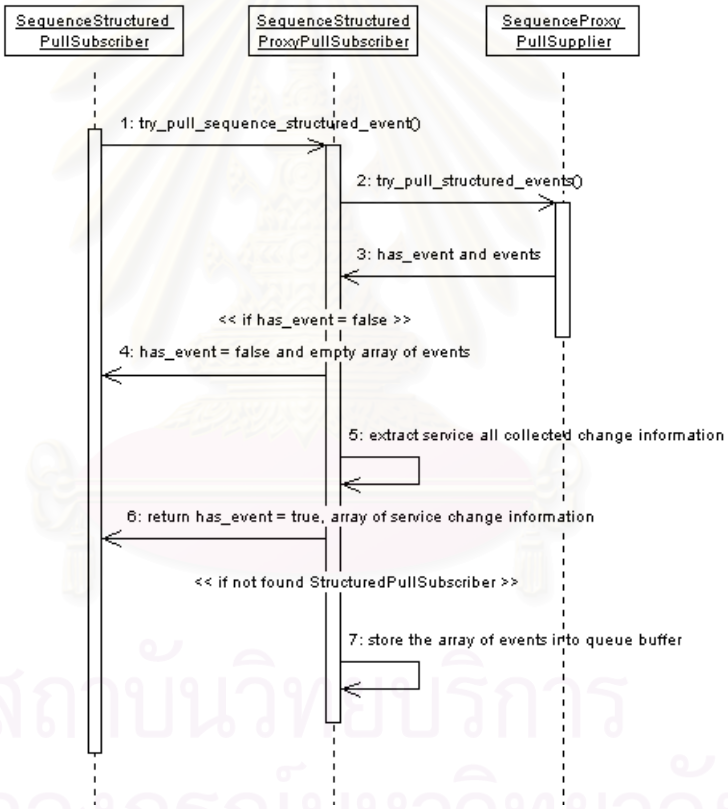


รูปที่ 4.26 แบบจำลองคลาสของ SequenceStructuredProxyPullSubscriberImpl

คลาส SequenceStructuredProxyPullSubscriberImpl สืบทอดมาจากคลาส SequencePullConsumerImplBase ซึ่งเป็นเชิร์ฟเวอร์สเกเลตันของส่วนต่อประสาน Sequence PullConsumer เพื่อทำหน้าที่เป็นผู้รับเหตุการณ์ของบริการแจ้งเหตุการณ์แบบชุดของวัตถุเหตุการร์แบบโครงสร้าง ที่มีการจัดส่งแบบพูล และพัฒนาตามส่วนต่อประสานของคลาส Sequence StructuredProxyPullSubscriberOperations เพื่อทำหน้าที่ให้บริการตามส่วนต่อประสาน SequenceStructuredProxyPullSubscriber ตัวกระทำการที่สำคัญคือ ตัวกระทำการ try_pull_sequence_structured_event() และ pull_sequence_structured_event() โดยทั้ง 2 ตัวกระทำการต่างก็เป็นตัวกระทำการสำหรับให้ผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการใช้ร้องขอข้อมูลการเปลี่ยนแปลงบริการในรูปแบบชุดของเหตุการณ์ แต่แตกต่างกันที่ตัวกระทำการ try_pull_sequence_structured_event() จะไม่มีการรอเพื่อเก็บรวบรวมข้อมูลการเปลี่ยนแปลงให้ครบตามจำนวนที่ต้องการ นั่นคือ ค่าคุณสมบัติ PacingInterval จะไม่มีผลต่อตัวกระทำการนี้ ส่วนตัวกระทำการ pull_sequence_structured_event() จะมีการรอจนกว่าเวลาจะครบตามค่าคุณสมบัติ PacingInterval ในกรณีที่จำนวนข้อมูลการเปลี่ยนแปลงบริการยังไม่ครบตามที่ระบุไว้ (สำหรับรายละเอียดของพารามิเตอร์ของตัวกระทำการสามารถดูได้ที่หัวข้อ 4.1.3)

รูปที่ 4.27 แสดงลำดับเหตุการณ์เมื่อผู้ต้องการรับทราบการเปลี่ยนแปลงบริการเรียกตัวกระทำการ try_pull_sequence_structured_event() เพื่อร้องขอข้อมูลการเปลี่ยนแปลงบริการ (1) โดยการทำงานของตัวกระทำการเริ่มต้นจากการร้องขอเหตุการณ์การเปลี่ยนแปลงบริการไปที่วัตถุ SequenceProxyPullSupplier ของบริการแจ้งเหตุการณ์ โดยเรียกตัวกระทำการ

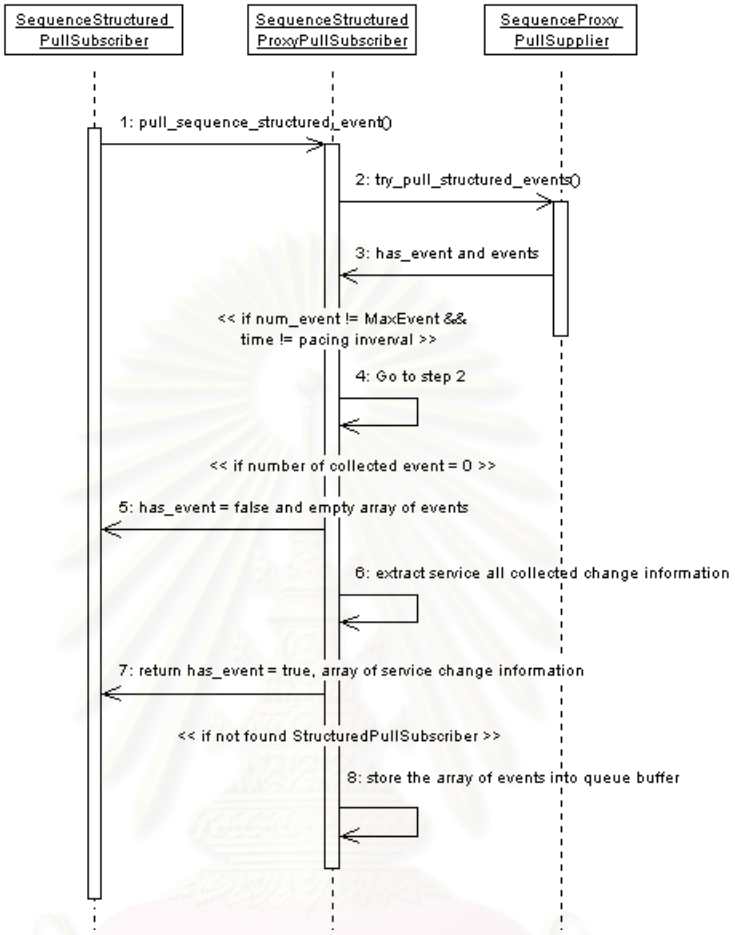
try_pull_structured_events() [5] [18] (2,3) ในขั้นตอนต่อไป ค่าของตัวแปร has_event ที่ได้คืนกลับมาจะได้รับการตรวจสอบโดยหากพบว่ามีค่าเป็นเท็จ นั่นแสดงว่าไม่มีเหตุการณ์การเปลี่ยนแปลงบริการใดเลยในเวลานั้นๆ ตัวกระทำการก็จะคืนค่ากลับให้แก่ผู้ต้องการรับทราบการเปลี่ยนแปลง เป็นค่า has_event ที่มีค่าเท็จ พร้อมกับชุดของข้อมูลการเปลี่ยนแปลงบริการที่ว่างเปล่า (4) แต่หากการตรวจสอบพบว่าค่าของ has_event มีค่าเท่ากับจริง ข้อมูลการเปลี่ยนแปลงบริการทั้งหมดจะได้รับการแยกออกมาจากชุดเหตุการณ์การเปลี่ยนแปลงบริการ (5) เพื่อใช้ในการจัดส่งให้กับผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการต่อไป (6) และในการจัดส่งหากไม่พบข้อผิดพลาดใดๆ ถือว่าเป็นการเสร็จสิ้นการทำงาน แต่หากในการส่งมีความผิดพลาดเกิดขึ้นโดยไม่สามารถส่งข้อมูลให้กับผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการได้ เหตุการณ์การเปลี่ยนแปลงบริการจะได้รับการจัดเก็บไว้เพื่อรอการร้องขอครั้งใหม่ (7)



รูปที่ 4.27 แผนภาพลำดับเหตุการณ์เมื่อผู้ต้องการรับทราบการเปลี่ยนแปลงบริการร้องขอข้อมูลโดยเรียกตัวกระทำการ try_pull_sequence_structured_event()

รูปที่ 4.28 แสดงลำดับเหตุการณ์เมื่อผู้ต้องการรับทราบการเปลี่ยนแปลงบริการเรียกตัวกระทำการ pull_sequence_structured_event() เพื่อร้องขอข้อมูลการเปลี่ยนแปลงบริการซึ่งจะเห็นได้ว่าการทำงานมีความคล้ายคลึงกับเมื่อเรียกตัวกระทำการ try_pull_sequence_structured_event() แต่จะแตกต่างกันตรงที่จะมีการรอจนกว่าเวลาจะครบตามที่กำหนดไว้ในคุณสมบัติ

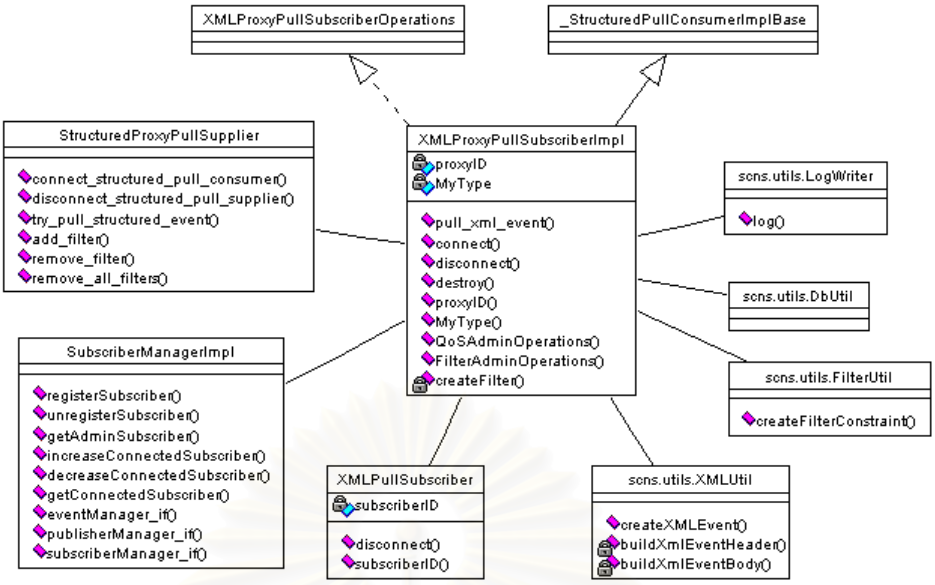
PacingInterval ในกรณีที่จำนวนเหตุการณ์การเปลี่ยนแปลงบริการยังได้รับมาไม่ครบตามจำนวนที่ต้องการ (ขั้นตอนที่ 2–4)



รูปที่ 4.28 แผนภาพลำดับเหตุการณ์เมื่อผู้ต้องการรับทราบการเปลี่ยนแปลงบริการร้องขอข้อมูลโดยเรียกตัวกระทำการ pull_sequence_structured_event()

7. คลาส XMLProxyPullSubscriberImpl

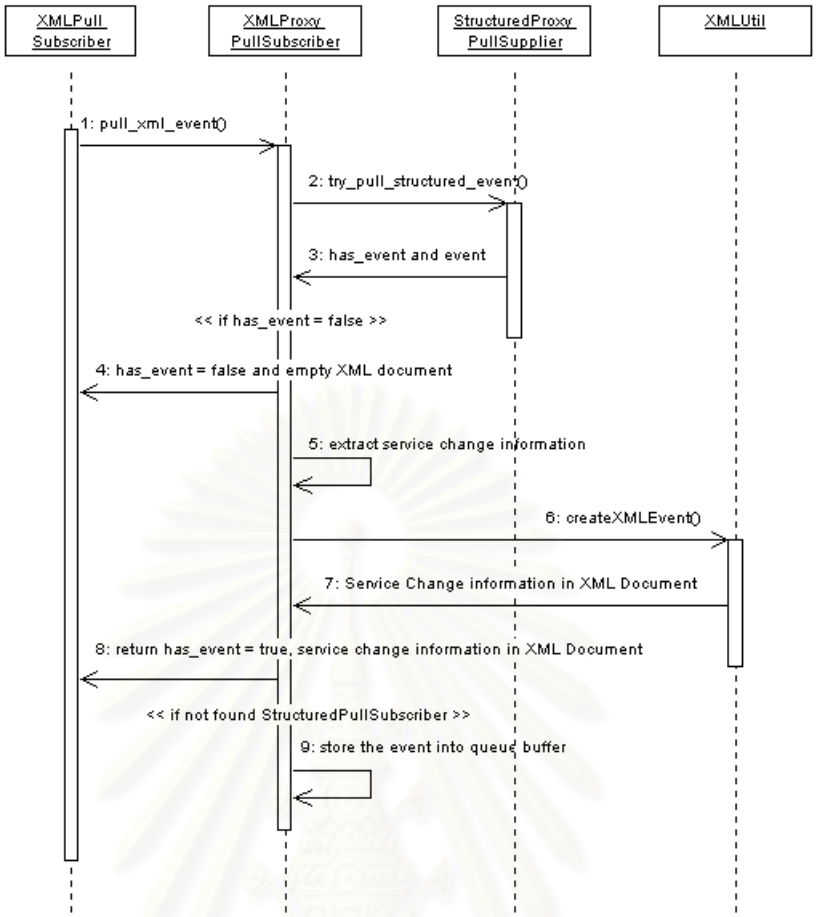
เป็นคลาสพัฒนาของส่วนต่อประสาน XMLProxyPullSubscriber มีแบบจำลองคลาสแสดงดังรูปที่ 4.29



รูปที่ 4.29 แบบจำลองคลาสของ XMLProxyPullSubscriberImpl

คลาส XMLProxyPullSubscriberImpl สืบทอดมาจากคลาส _StructuredPullConsumerImplBase ซึ่งเป็นเซิร์ฟเวอร์สเกเลตันของส่วนต่อประสาน StructuredPullConsumer เพื่อทำหน้าที่เป็นผู้รับเหตุการณ์ของบริการแจ้งเหตุการณ์แบบวัตถุเหตุการณ์แบบโครงสร้าง ที่มีการจัดส่งแบบพูล และพัฒนาตามส่วนต่อประสานของคลาส XMLProxy PullSubscriberOperations เพื่อทำหน้าที่ให้บริการตามส่วนประสาน XMLProxyPullSubscriber ตัวกระทำการที่สำคัญคือ ตัวกระทำการ pull_xml_event() ซึ่งเป็นตัวกระทำการที่ผู้ต้องการรับทราบการเปลี่ยนแปลงบริการใช้ร้องขอข้อมูลการเปลี่ยนแปลงบริการในรูปแบบของเอกสารเอ็กซ์เอ็มแอล (สำหรับรายละเอียดของพารามิเตอร์ต่างๆ สามารถดูได้ที่หัวข้อ 4.1.3)

รูปที่ 4.30 แสดงลำดับเหตุการณ์เมื่อผู้ต้องการรับทราบการเปลี่ยนแปลงบริการเรียกตัวกระทำการ pull_xml_event() เพื่อร้องขอข้อมูลการเปลี่ยนแปลงบริการในรูปแบบเอกสารเอ็กซ์เอ็มแอล ซึ่งจะเห็นได้ว่าการทำงานมีความคล้ายคลึงกันกับตัวกระทำการ pull_structured_event() ของ StructuredProxyPullSubscriber (รูปที่ 4.24) โดยขั้นตอนที่เพิ่มเข้ามาคือข้อมูลการเปลี่ยนแปลงบริการที่แยกออกมาได้จะถูกนำไปสร้างเป็นเอกสารเอ็กซ์เอ็มแอลจากการเรียกใช้งานตัวกระทำการ createXMLEvent() ของวัตถุ XMLUtil (ในขั้นตอนที่ 6, 7) หลังจากนั้นข้อมูลการเปลี่ยนแปลงบริการที่อยู่ในรูปแบบของเอกสารเอ็กซ์เอ็มแอลก็จะถูกจัดส่งให้กับผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการต่อไปด้วยขั้นตอนการทำงานเดียวกันกับของ StructuredProxyPull Subscriber



รูปที่ 4.30 แผนภาพลำดับเหตุการณ์เมื่อผู้ต้องการรับทราบการเปลี่ยนแปลงบริการ XMLPullSubscriber ร้องขอข้อมูลการเปลี่ยนแปลงบริการ

4.4 แฟ้มคุณสมบัติของบริการเอสซีเอ็น

ในหัวข้อนี้จะอธิบายเกี่ยวกับรายละเอียดของแฟ้มคุณสมบัติ (scns.properties) ของบริการเอสซีเอ็น ซึ่งผู้ดูแลบริการเอสซีเอ็นใช้ในการกำหนดค่าคุณสมบัติและสภาวะแวดล้อมเริ่มต้นต่างๆ ให้กับบริการเอสซีเอ็นก่อนที่จะทำการสั่งเริ่มต้นการทำงานให้กับบริการ คุณสมบัติที่กำหนดได้แสดงดังตารางที่ 4.2

ตารางที่ 4.2 คุณสมบัติที่สามารถกำหนดได้ในแฟ้มคุณสมบัติของบริการเอสซีเอ็น

คุณสมบัติ	**คำอธิบาย**
scns_ior_path	ระบุไฟล์ที่ใช้เก็บข้อมูลอ้างถึงวัตถุของบริการเอสซีเอ็น เช่น scns_ior_path= c:\\tmp\\scn.ior

scns_url	ระบุยูอาร์แอลที่สามารถดึงข้อมูลอ้างถึงวัตถุของบริการเอสซีเอ็น เช่น scns_Url=file:///c:/tmp/scn.ior
dcon_url	ระบุยูอาร์แอลที่สามารถดึงข้อมูลอ้างถึงวัตถุของบริการแจ้งเหตุการณ์ เช่น scns_Url= file:///c:/tmp/cosnotif.ior
tm_logfile	ระบุไฟล์ที่ใช้เก็บสถานะการทำงานของตัวตรวจบริการเทรดเดอร์ เช่น tm_logfile= c:\\tmp\\logfiles\\tm.log
scns_logfile	ระบุไฟล์ที่ใช้เก็บสถานะการทำงานของบริการเอสซีเอ็น เช่น scns_logfile= c:\\tmp\\logfiles\\scns.log
max_subscriber	ระบุค่าคุณสมบัติคุณภาพของบริการ MaxSubscribers เช่น max_subscriber = 2000
max_connected_subscriber	ระบุค่าคุณสมบัติคุณภาพของบริการ MaxConnectedSubscribers เช่น max_connected_subscriber = 1000
max_events_per_proxy	ระบุค่าคุณสมบัติคุณภาพของบริการ MaxEventsPerProxy เช่น max_events_per_proxy = 20
orderPolicy_default	ระบุค่าคุณสมบัติคุณภาพของบริการ OrderPolicy เริ่มต้นสำหรับตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ โดยสามารถมีค่าได้ดังนี้ 1 = FIFO order 2 = Priority order 3 = Expiry Time order เช่น orderPolicy_default = 2
discardPolicy_default	ระบุค่าคุณสมบัติคุณภาพของบริการ DiscardPolicy เริ่มต้นสำหรับตัวแทนผู้ที่

	ต้องการรับทราบการเปลี่ยนแปลงบริการ โดยสามารถมีค่าได้ดังนี้ 1 = FIFO order 2 = Priority order 3 = Expiry Time order 4 = LIFO order เช่น discardPolicy_default = 3
maximumBatchSize_default	ระบุค่าคุณสมบัติคุณภาพของบริการ MaximumBatchSize เริ่มต้นสำหรับตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ เช่น maximumBatchSize _default = 5
pacingInterval_default	ระบุค่าคุณสมบัติคุณภาพของบริการ PacingInterval เริ่มต้นสำหรับตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ เช่น pacingInterval _default = 300000 (มีหน่วยเป็นมิลลิวินาที)
instantiation_of_service.priority_default	ระบุค่าความสำคัญ (Priority) เริ่มต้นให้กับข้อมูลการเปลี่ยนแปลงบริการประเภทการเกิดบริการใหม่ขึ้นในระบบ เช่น instantiation_of_service.priority_default = 5
removal_of_service.priority_default	ระบุค่าความสำคัญ (Priority) เริ่มต้นให้กับข้อมูลการเปลี่ยนแปลงบริการประเภทบริการถูกยกเลิกการใช้งาน เช่น removal_of_service.priority_default = 10
change_type_of_service.priority_default	ระบุค่าความสำคัญ (Priority) เริ่มต้นให้กับข้อมูลการเปลี่ยนแปลงบริการประเภทบริการได้รับการเปลี่ยนแปลงชนิดของบริการ เช่น change_type_of_service.priority_default = 8
change_property_of_service.priority_default	ระบุค่าความสำคัญ (Priority) เริ่มต้นให้กับข้อ

	มูลการเปลี่ยนแปลงบริการประเภทบริการได้รับการเปลี่ยนแปลงค่าคุณสมบัติ เช่น change_property_of_service.priority_default = 10
instantiation_of_service.expiry_default	ระบุค่าวัน-เวลาหมดอายุ (Expiry Time) เริ่มต้นให้กับข้อมูลการเปลี่ยนแปลงบริการประเภทการเกิดบริการใหม่ขึ้นในระบบ เช่น instantiation_of_service.expiry_default = 7 (มีหน่วยเป็นวัน)
removal_of_service.expiry_default	ระบุค่าวัน-เวลาหมดอายุ (Expiry Time) เริ่มต้นให้กับข้อมูลการเปลี่ยนแปลงบริการประเภทบริการถูกยกเลิกการใช้งาน เช่น removal_of_service.expiry_default = 7 (มีหน่วยเป็นวัน)
change_type_of_service.expiry_default	ระบุค่าวัน-เวลาหมดอายุ (Expiry Time) เริ่มต้นให้กับข้อมูลการเปลี่ยนแปลงบริการประเภทบริการได้รับการเปลี่ยนแปลงค่าชนิดของบริการ เช่น change_type_of_service.expiry_default = 7 (มีหน่วยเป็นวัน)
change_property_of_service.expiry_default	ระบุค่าวัน-เวลาหมดอายุ (Expiry Time) เริ่มต้นให้กับข้อมูลการเปลี่ยนแปลงบริการประเภทบริการได้รับการเปลี่ยนแปลงค่าคุณสมบัติของบริการ เช่น change_property_of_service.expiry_default = 7 (มีหน่วยเป็นวัน)

บทที่ 5
การทดสอบการใช้งานบริการเอสซีเอ็น

ในบทนี้จะกล่าวถึงรายละเอียดของการทดสอบการทำงานของบริการเอสซีเอ็นประกอบด้วยการทดสอบตัวตรวจบริการเทรดเดอร์ การทดสอบการใช้งานบริการเอสซีเอ็น และตัวอย่างการทดสอบประสิทธิภาพ โดยรายละเอียดการทดสอบมีดังนี้

5.1 สภาวะที่ใช้ในการทดสอบ

1. เครื่อง Intel Celeron 400 หน่วยความจำขนาด 640 เมกะไบต์ ระบบปฏิบัติการลินุกซ์ รุ่น 2.2.14 สำหรับรันบริการเอสซีเอ็น บริการเทรดเดอร์ และบริการแจ้งเหตุการณ์
2. เครื่อง Intel Pentium II 333 หน่วยความจำขนาด 384 เมกะไบต์ ระบบปฏิบัติการวินโดวส์ เอ็มอีสำหรับการจำลองผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ
3. เครื่อง Intel Pentium III 650 หน่วยความจำขนาด 128 เมกะไบต์ ระบบปฏิบัติการวินโดวส์ เอ็มอี สำหรับการจำลองผู้แจ้งการเปลี่ยนแปลงบริการ
4. เครื่องมือพัฒนาภาษาจาวา เจดีเค 1.2.2
5. วิสิโบรกเกอร์รุ่น 3.4 สำหรับภาษาจาวา
6. บริการเทรดเดอร์ของจาคอร์บ จำนวน 1 ตัว
7. บริการแจ้งเหตุการณ์ขอดีคอน รุ่น 2.2 จำนวน 1 ตัว
8. บริการเอสซีเอ็นจำนวน 1 ตัว
9. ระบบฐานข้อมูลไอดีบี รุ่น 3.5

5.2 การทดสอบการทำงานของบริการเอสซีเอ็น

ในส่วนนี้เป็นการทดสอบการทำงานของส่วนต่อประสานต่างๆ ของบริการเอสซีเอ็น โดยการทดสอบจะอาศัยโปรแกรมที่พัฒนาขึ้นสำหรับการทดสอบโดยเฉพาะ โดยภายในโปรแกรมจะมีการติดต่อกับบริการเอสซีเอ็นเพื่อเรียกใช้งานส่วนต่อประสานต่างๆ การทดสอบแบ่งออกเป็นส่วนๆ ดังต่อไปนี้

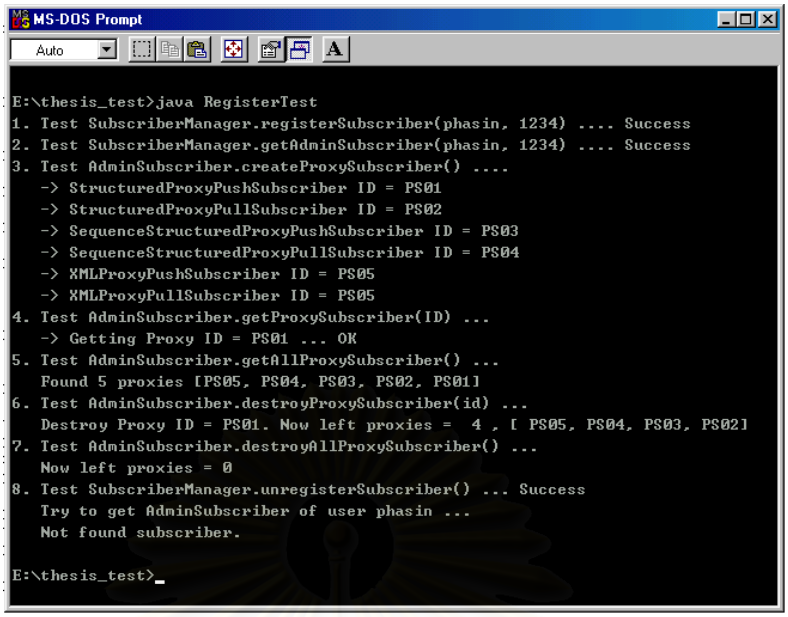
5.2.1 การทดสอบการลงทะเบียนโดยผู้ต้องการรับทราบการเปลี่ยนแปลงบริการและการจัดสร้างวัตถุตัวแทนประเภทต่างๆ

การทดสอบในกลุ่มนี้ประกอบด้วยการทดสอบการใช้งานส่วนต่อประสานตัวจัดการผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ (SubscriberManager Interface) และส่วนต่อประสาน

ผู้ดูแลตัวแทน (AdminSubscriber Interface) โปรแกรมทดสอบประกอบด้วยการเรียกใช้งานตัวกระทำการของส่วนต่อประสานต่างๆ เป็นลำดับขั้นดังต่อไปนี้

1. เรียกใช้งานตัวกระทำการ registerSubscriber() ของ ส่วนต่อประสาน SubscriberManager ในการลงทะเบียนเป็นผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ โดยการระบุชื่อคือ phasin และรหัสผ่านคือ 1234
2. เรียกใช้งานตัวกระทำการ getAdminSubscriber() ของส่วนต่อประสาน SubscriberManager สำหรับการร้องขอวัตถุผู้ดูแลตัวแทน
3. เรียกใช้งานตัวกระทำการ createProxySubscriber() ของส่วนต่อประสาน AdminSubscriber สำหรับการสร้างวัตถุตัวแทน โดยจะทำการสร้างวัตถุตัวแทนทั้ง 6 ประเภทและแสดงรหัสของวัตถุตัวแทนที่สร้างขึ้น
4. เรียกใช้งานตัวกระทำการ getProxySubscriber() ของส่วนต่อประสาน AdminSubscriber สำหรับการร้องขอข้อมูลอ้างถึงวัตถุของวัตถุตัวแทน โดยในที่นี้คือวัตถุตัวแทนประเภท StructuredProxyPushSubscriber ซึ่งมีรหัสคือ PS01
5. เรียกใช้งานตัวกระทำการ getAllProxySubscriber() ของส่วนต่อประสาน AdminSubscriber สำหรับการร้องขอข้อมูลอ้างถึงวัตถุของวัตถุตัวแทนทั้งหมด โดยโปรแกรมจะแสดงจำนวนวัตถุตัวแทนที่ได้รับมาทั้งหมด (6 ตัว) และรหัสของวัตถุตัวแทนแต่ละตัว
6. เรียกใช้งานตัวกระทำการ destroyProxySubscriber() ของส่วนต่อประสาน AdminSubscriber สำหรับการทำลายวัตถุตัวแทน โดยในที่จะทำการทำลายวัตถุตัวแทนที่มีรหัสคือ PS01 จากนั้นจะแสดงจำนวนวัตถุตัวแทนพร้อมกับรหัสของวัตถุตัวแทนแต่ละตัวที่คงเหลืออยู่
7. เรียกใช้งานตัวกระทำการ destroyAllProxySubscriber() ของส่วนต่อประสาน AdminSubscriber สำหรับการทำลายวัตถุตัวแทนทั้งหมด และทำการเรียกตัวกระทำการ getAllProxySubscriber() เพื่อตรวจดูการทำงาน แสดงจำนวนวัตถุตัวแทนที่เหลืออยู่ซึ่งจะต้องมีค่าเท่ากับศูนย์
8. เรียกใช้งานตัวกระทำการ unregisterSubscriber() ของส่วนต่อประสาน SubscriberManager สำหรับการยกเลิกการลงทะเบียน และทำการทดลองร้องขอวัตถุผู้ดูแลตัวแทนหลังจากที่ยกเลิกการลงทะเบียนแล้ว

ผลการทำงานของโปรแกรมสามารถแสดงได้ดังรูปที่ 5.1



รูปที่ 5.1 ผลการทำงานของโปรแกรมทดสอบการลงทะเบียน

จากผลการทำงานในรูปที่ 5.1 สามารถวิเคราะห์ได้ดังต่อไปนี้

1. การทำงานของส่วนต่อประสาน SubscriberManager ที่ประกอบด้วยตัวกระทำการ registerSubscriber() ตัวกระทำการ getAdminSubscriber() และตัวกระทำการ unregisterSubscriber () (จากผลการทำงานในข้อที่ 1, 2 และ 8 ตามลำดับ) สามารถทำงานได้อย่างถูกต้อง
2. การทำงานของส่วนต่อประสาน AdminSubscriber ที่ประกอบด้วยตัวกระทำการ createProxySubscriber() ตัวกระทำการ getProxySubscriber() ตัวกระทำการ getAllProxySubscriber() ตัวกระทำการ destroyProxySubscriber() และตัวกระทำการ destroyAllProxySubscriber() (จากผลการทำงานในข้อที่ 3, 4, 5, 6 และ 7 ตามลำดับ) สามารถทำงานได้อย่างถูกต้อง

5.2.2 การทดสอบการจัดส่งและรับข้อมูลการเปลี่ยนแปลงบริการ

การทดสอบกระทำโดยใช้โปรแกรม ประกอบด้วย

1. โปรแกรมจำลองเป็นผู้แจ้งการเปลี่ยนแปลงบริการล่วงหน้า จำนวน 1 ตัว ทำการส่งข้อมูลการเปลี่ยนแปลงบริการ (ทดสอบการทำงานของส่วนต่อประสานตัวจัดการผู้แจ้งการเปลี่ยนแปลงบริการ (PublisherManager Interface)) โดยโปรแกรมจะทำการสร้างข้อมูลการเปลี่ยนแปลงที่แตกต่างกันจำนวน 5 ข้อมูล โดยจัดส่งตามลำดับจากข้อมูลที่ 1 ถึงข้อมูลที่ 5 โดยข้อมูลแต่ละตัวมีรายละเอียดดังต่อไปนี้

ข้อมูลที่ 1 ข้อมูลการเพิ่มบริการใหม่ มีรายละเอียดดังนี้

ค่าความสำคัญ = 5 ค่าวัน-เวลาหมดอายุ = 3 วัน

ชนิดของบริการ
Service Type Name :Bank
 	
Property Definition :
Name [string]
ReservedFund [long]
Interest_rate [float]
 	
interface Bank {
float balance (in string accno);
boolean deposit (in string accno, in float amount);
boolean withdraw(in string accno, in float amount);
};

รายการคุณสมบัติ Name = CU-Bank ReservedFund = 1000000000 Interest_rate = 5.0

ข้อมูลที่ 2 ข้อมูลการลบบริการ มีรายละเอียดดังนี้

ค่าความสำคัญ = 10 ค่าวัน-เวลาหมดอายุ = 7 วัน

ชนิดของบริการ
Service Type Name :Printer
 	
Property Definition :
Type [string]
Name [string]
Speed [long]
 	
interface PrinterService {
boolean print(in string url);
};

รายการคุณสมบัติ Type =Laser Printer Name =HP LaserJet 1100 Speed =20

ข้อมูลที่ 3 ข้อมูลการเปลี่ยนแปลงค่าคุณสมบัติของบริการ มีรายละเอียดดังนี้

ค่าความสำคัญ = 8 ค่าวัน-เวลาหมดอายุ = 5 วัน
ชนิดของบริการ

```
Service Type Name :Printer

Property Definition :
  Type [string]
  Name [string]
  Speed [long]

interface PrinterService {
  boolean print(in string url);
};
```

รายการคุณสมบัติ (ก่อนการเปลี่ยนแปลง)

```
Type = Laser Printer
Name = HP LaserJet 1100
Speed 20
```

รายการคุณสมบัติ (หลังการเปลี่ยนแปลง)

```
Type = Laser Printer
Name = HP LaserJet Super Turbo
Speed = 30
```

ข้อมูลที่ 4 ข้อมูลการเปลี่ยนแปลงชนิดของบริการ มีรายละเอียดดังนี้

ค่าความสำคัญ = 10

ค่าวัน-เวลาหมดอายุ = 7 วัน

ชนิดของบริการ (ก่อนการเปลี่ยนแปลง)

```
Service Type Name :Bank

Property Definition :
  Name [string]
  ReservedFund [long]
  Interest_rate [float]

 interface Bank {
  float balance (in string accno);
  boolean deposit (in string accno, in float amount);
  boolean withdraw(in string accno, in float amount);
};
```

รายการคุณสมบัติ (ก่อนการเปลี่ยนแปลง)

```
Name = ISEL-Bank
ReservedFund=800000000
Interest_rate=5.0
```

ชนิดของบริการ (หลังการเปลี่ยนแปลง)

```
Service Type Name : Bank_2

Property Definition :
  Name [string]
  ReservedFund [long]
  Interest_rate [float]

 interface Bank_2 {
   float balance   (in string accno);
   boolean deposit (in string accno,
                    in long type,
                    in float amount);
   boolean withdraw(in string accno,
                    in long type
```

```
                    in float amount);
   boolean transfer(in string accno_src,
                    in string accno_dest,
                                in float amount);
};
```

รายการคุณสมบัติ (หลังการเปลี่ยนแปลง)

```
Name = ISEL-Bank
ReservedFund=800000000
Interest_rate=5.0
```

ข้อมูลที่ 5 ข้อมูลการเพิ่มบริการใหม่ มีรายละเอียดดังนี้

ค่าความสำคัญ = 5

ค่าวัน-เวลาหมดอายุ = 3 วัน

ชนิดของบริการ

```
Service Type Name :Bank

Property Definition :
  Name [string]
  ReservedFund [long]
  Interest_rate [float]

 interface Bank {
  float balance (in string accno);
  boolean deposit (in string accno, in float amount);
  boolean withdraw(in string accno, in float amount);
 };
```

รายการคุณสมบัติ

```
Name = CP-Bank
ReservedFund = 15000000
Interest_rate = 5.2
```

2. โปรแกรมจำลองเป็นผู้ต้องการรับทราบการเปลี่ยนแปลง จำนวน 6 ตัว สำหรับการทดสอบส่วนต่อประสานของวัตถุตัวแทน และการทดสอบการรับข้อมูลการเปลี่ยนแปลงบริการ โดยแต่ละตัวมีคุณสมบัติที่แตกต่างกันดังนี้ คือ

ตัวที่ 1 ลงทะเบียนโดยใช้ชื่อ user1 มีรหัสผ่านคือ iamuser1 ทำการสร้างวัตถุตัวแทนประเภท StructuredProxyPushSubscriber

ตัวที่ 2 ลงทะเบียนโดยใช้ชื่อ user2 มีรหัสผ่านคือ iamuser2 ทำการสร้างวัตถุตัวแทนประเภท SequenceStructuredProxyPushSubscriber

ตัวที่ 3 ลงทะเบียนโดยใช้ชื่อ user3 มีรหัสผ่านคือ iamuser3 ทำการสร้างวัตถุตัวแทนประเภท XMLProxyPushSubscriber

ตัวที่ 4 ลงทะเบียนโดยใช้ชื่อ user4 มีรหัสผ่านคือ iamuser4 ทำการสร้างวัตถุตัวแทนประเภท StructuredProxyPullSubscriber

ตัวที่ 5 ลงทะเบียนโดยใช้ชื่อ user5 มีรหัสผ่านคือ iamuser5 ทำการสร้างวัตถุตัวแทนประเภท SequenceStructuredProxyPullSubscriber

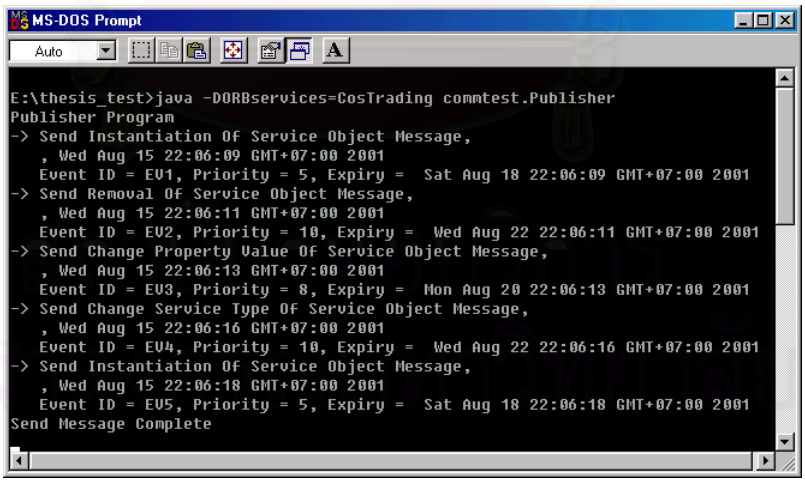
ตัวที่ 6 ลงทะเบียนโดยใช้ชื่อ user6 มีรหัสผ่านคือ iamuser6 ทำการสร้างวัตถุตัวแทนประเภท XMLProxyPullSubscriber

การทำงานของโปรแกรมผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการแต่ละประเภทประกอบด้วย

1. ลงทะเบียนเข้ากับบริการเอสซีเอ็นและสร้างวัตถุตัวแทน
2. เรียกตัวกระทำการ connect () ของส่วนต่อประสานวัตถุตัวแทนแต่ละตัวเพื่อติดต่อเข้ากับวัตถุตัวแทนสำหรับการรอรับข้อมูลการเปลี่ยนแปลงบริการ
3. รอรับข้อมูลการเปลี่ยนแปลงบริการสำหรับผู้ต้องการรับทราบการเปลี่ยนแปลงที่สร้างวัตถุตัวแทนแบบพุช หรือ เรียกตัวกระทำการสำหรับดึง ข้อมูลการเปลี่ยนแปลงบริการสำหรับผู้ต้องการรับทราบการเปลี่ยนแปลงที่สร้างวัตถุตัวแทนแบบดึง

จากนั้นทำการรันโปรแกรมผู้ต้องการรับทราบการเปลี่ยนแปลงบริการทั้ง 6 ตัว แล้วรันโปรแกรมผู้แจ้งการเปลี่ยนแปลง

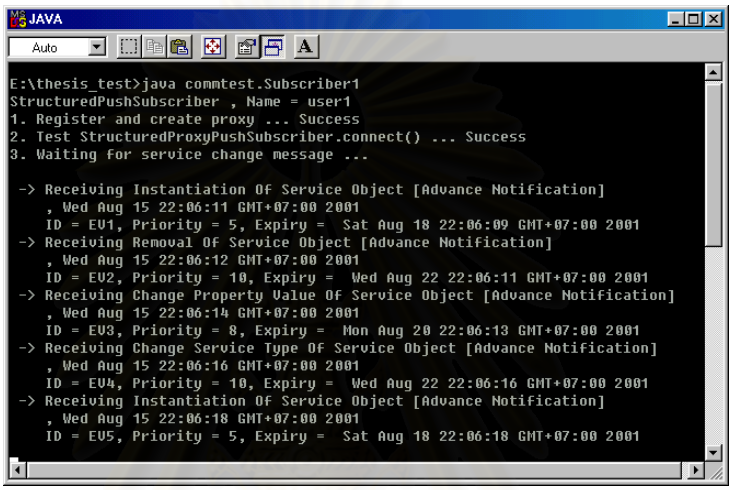
การทำงานของโปรแกรมผู้แจ้งการเปลี่ยนแปลงบริการแสดงได้ดังรูปที่ 5.2



รูปที่ 5.2 การทำงานของโปรแกรมผู้แจ้งการเปลี่ยนแปลงบริการ

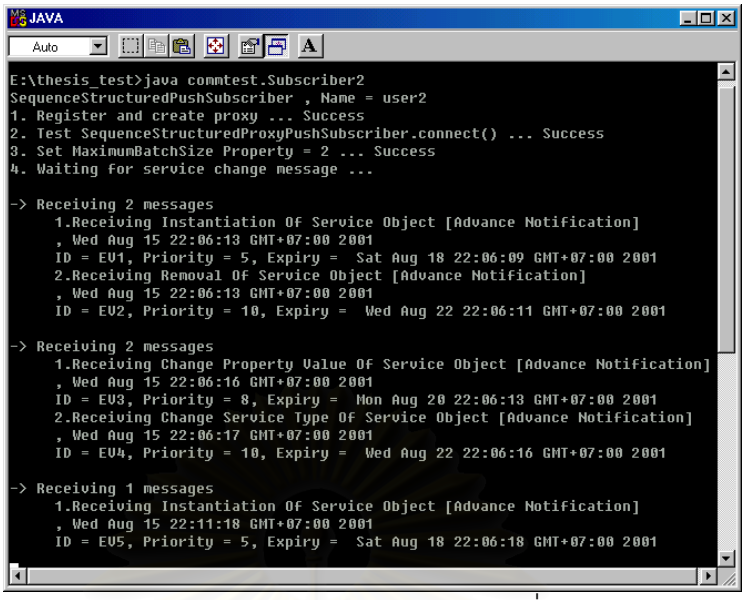
ในวิทยานิพนธ์นี้จะแสดงผลการรับข้อมูลการเปลี่ยนแปลงบริการของโปรแกรมผู้ต้องการรับทราบการเปลี่ยนแปลงตัวที่ 1, 2 และ 6 เท่านั้น ส่วนตัวที่ 3, 4 และ 5 นั้น ได้ผลการทำงานที่ถูกต้องเช่นเดียวกัน

รูปที่ 5.3 แสดงผลการรับข้อมูลการเปลี่ยนแปลงบริการของโปรแกรมผู้ต้องการรับทราบการเปลี่ยนแปลงบริการตัวที่ 1 ประเภท StructuredPushSubscriber ซึ่งพบได้ว่าสามารถรับข้อมูลได้ครบและถูกต้องตามที่โปรแกรมผู้แจ้งการเปลี่ยนแปลงบริการทำการจัดส่ง



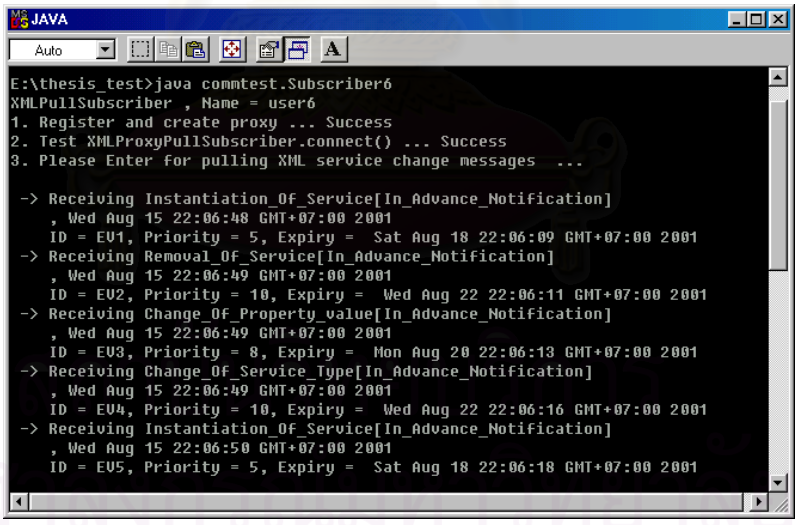
รูปที่ 5.3 การทำงานของโปรแกรมผู้ต้องการรับทราบการเปลี่ยนแปลงบริการตัวที่ 1 ประเภท StructuredPushSubscriber

รูปที่ 5.4 แสดงผลการรับข้อมูลการเปลี่ยนแปลงบริการของโปรแกรมผู้ต้องการรับทราบการเปลี่ยนแปลงบริการตัวที่ 2 ประเภท SequenceStructuredPushSubscriber ซึ่งพบได้ว่าสามารถรับข้อมูลได้ครบและถูกต้องตามที่โปรแกรมผู้แจ้งการเปลี่ยนแปลงบริการทำการจัดส่ง



รูปที่ 5.4 การทำงานของโปรแกรมผู้ต้องการรับทราบการเปลี่ยนแปลงบริการตัวที่ 2 ประเภท SequenceStructuredPushSubscriber

รูปที่ 5.5 แสดงผลการรับข้อมูลการเปลี่ยนแปลงบริการของโปรแกรมผู้ต้องการรับทราบการเปลี่ยนแปลงบริการตัวที่ 6 ประเภท XMLPullSubscriber ซึ่งพบได้ว่าสามารถรับข้อมูลได้ครบถูกต้องตามที่โปรแกรมผู้แจ้งการเปลี่ยนแปลงจัดส่ง



รูปที่ 5.5 การทำงานของโปรแกรมผู้ต้องการรับทราบการเปลี่ยนแปลงบริการตัวที่ 6 ประเภท XMLPullSubscriber

จากผลการทำงานข้างต้นแสดงได้ว่าตัวกระทำการ connect() ของส่วนต่อประสานวัตถุตัวแทนแต่ละประเภทสามารถทำงานได้อย่างถูกต้อง รวมทั้งวัตถุตัวแทนแต่ละตัวสามารถทำการ

จัดส่งข้อมูลไปให้แก่ผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการได้อย่างถูกต้องและครบถ้วน (โดยไม่คำนึงถึงลำดับในการจัดส่ง)

5.2.3 การทดสอบการกำหนดข้อมูลระบุความต้องการ

การทดสอบในส่วนนี้เป็นการทดสอบการใช้งานส่วนต่อประสานตัวจัดการข้อมูลระบุความต้องการ (FilterAdmin Interface) ทำการทดสอบโดยใช้โปรแกรม ซึ่งประกอบด้วย

1. โปรแกรมจำลองเป็นผู้แจ้งการเปลี่ยนแปลงบริการ (ใช้โปรแกรมเดียวกับการทดสอบการจัดส่งและรับข้อมูลการเปลี่ยนแปลงบริการ)

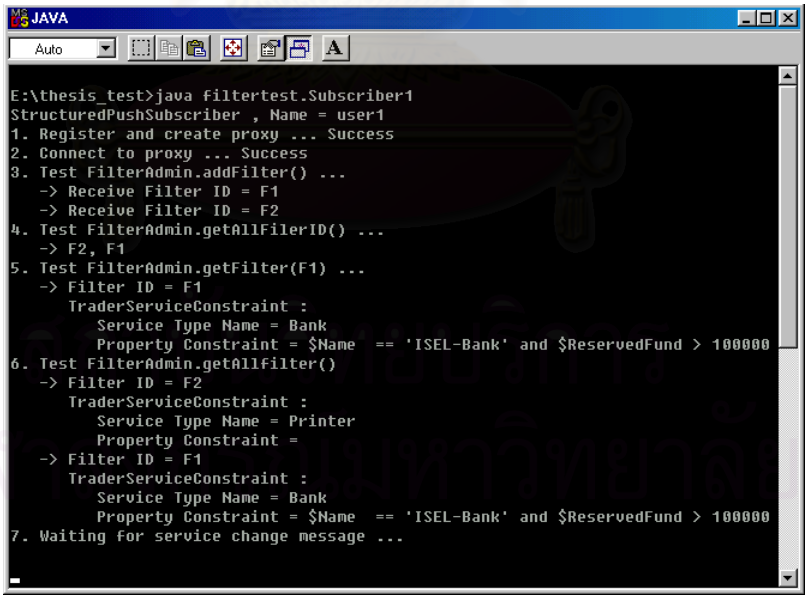
2. โปรแกรมจำลองเป็นผู้ต้องการรับทราบการเปลี่ยนแปลงบริการ ประเภท Structured PushSubscriber จำนวน 1 ตัว เพื่อทำการเรียกใช้งานส่วนต่อประสานตัวจัดการข้อมูลระบุความต้องการ โปรแกรมมีการทำงานหลักๆ ดังต่อไปนี้คือ

1. ทำการลงทะเบียนเข้ากับบริการเอสซีเอ็น และสร้างวัตถุตัวแทนประเภท Structured ProxPushSubscriber
2. ทำการติดต่อเข้ากับวัตถุตัวแทนที่สร้างขึ้น
3. ทำการเรียกใช้งานตัวกระทำการ addFilter() ของส่วนต่อประสาน FilterAdmin เพื่อเพิ่มข้อมูลระบุความต้องการ โดยข้อมูลระบุความต้องการที่กำหนดประกอบด้วย
   - บริการที่มีชนิดของบริการ คือ Bank และมีคุณสมบัติ
     Name = “ISEL-Bank”
     ReservedFund > 100000
   - บริการใดๆ ที่มีชนิดของบริการ คือ Printer

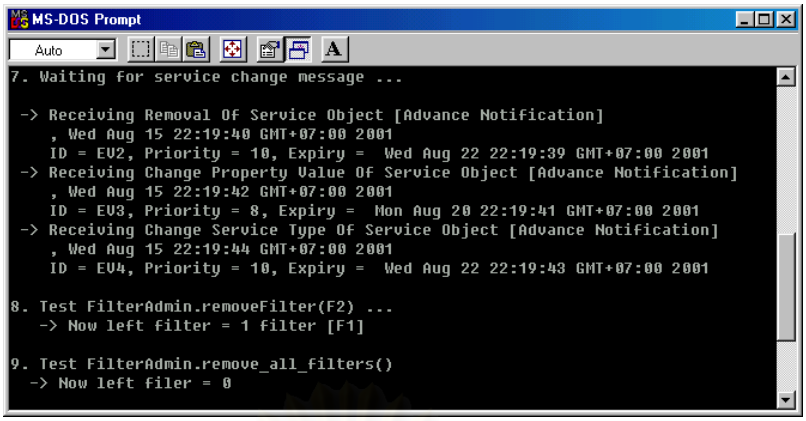
   จากนั้นแสดงรหัสของข้อมูลระบุความต้องการที่ได้รับมาจากการเรียกใช้งานตัวกระทำการ addFilter()
4. ทำการเรียกใช้งานตัวกระทำการ getAllFilterID() ของส่วนต่อประสาน FilterAdmin เพื่อร้องขอรหัสของข้อมูลระบุความต้องการทั้งหมดที่มีอยู่ จากนั้นแสดงรหัสที่ได้รับออกทางจอภาพ
5. ทำการเรียกใช้งานตัวกระทำการ getFilter() ของส่วนต่อประสาน FilterAdmin เพื่อร้องขอรายละเอียดของข้อมูลระบุความต้องการที่ทำการระบุรหัส ซึ่งในที่นี้คือรหัส F1 จากนั้นทำการแสดงรายละเอียดของข้อมูลระบุความต้องการ F1

6. ทำการเรียกใช้งานตัวกระทำการ getAllfilter() ของส่วนต่อประสาน FilterAdmin เพื่อร้องขอรายละเอียดของข้อมูลระบุความต้องการทั้งหมดที่มีอยู่ จากนั้นแสดงผลออกทางจอภาพ
7. รอรับข้อมูลแจ้งการเปลี่ยนแปลงบริการและสังเกตผลการรับข้อมูลแจ้งการเปลี่ยนแปลงบริการ ซึ่งข้อมูลการเปลี่ยนแปลงที่ได้รับจะต้องเป็นไปตามข้อมูลระบุความต้องการที่กำหนดไว้
8. ทำการเรียกใช้งานตัวกระทำการ removeFilter() ของส่วนต่อประสาน FilterAdmin เพื่อลบข้อมูลระบุความต้องการตามรหัสที่กำหนด ซึ่งในที่นี้คือ รหัส F2 จากนั้นเรียกตัวกระทำการ getAllFilterID() เพื่อแสดงถึงข้อมูลระบุความต้องการที่คงเหลืออยู่ ซึ่งจะต้องเหลือข้อมูลระบุความต้องการเพียงตัวเดียวคือข้อมูลระบุความต้องการรหัส F1
9. ทำการเรียกใช้งานตัวกระทำการ remove_all_filters() ของส่วนต่อประสาน FilterAdmin เพื่อลบข้อมูลระบุความต้องการทั้งหมด จากนั้นเรียกตัวกระทำการ getAllFilterID() เพื่อแสดงถึงข้อมูลระบุความต้องการที่คงเหลืออยู่ ซึ่งจะต้องไม่มีข้อมูลระบุความต้องการเหลืออยู่

ผลการทำงานของโปรแกรมแสดงได้ดังรูปที่ 5.6 และ 5.7



รูปที่ 5.6 การทำงานของโปรแกรมผู้ต้องการรับทราบการเปลี่ยนแปลงสำหรับการทดสอบการกำหนดข้อมูลระบุความต้องการก่อนได้รับข้อมูลการเปลี่ยนแปลงบริการ



รูปที่ 5.7 การทำงานของโปรแกรมผู้ต้องการรับทราบการเปลี่ยนแปลงสำหรับการทดสอบการกำหนดข้อมูลระบุความต้องการหลังได้รับข้อมูลการเปลี่ยนแปลงบริการ

จากผลการทำงานข้างต้นสามารถวิเคราะห์ได้ดังต่อไปนี้

1. การทำงานของตัวกระทำการต่างๆ ของส่วนต่อประสานตัวจัดการข้อมูลระบุความต้องการ (FilterAdmin) มีการทำงานถูกต้อง
2. การจัดส่งข้อมูลตามข้อมูลระบุความต้องการมีความถูกต้อง นั่นคือได้รับข้อมูลจำนวน 3 ข้อมูล ประกอบด้วย ข้อมูลที่มีรหัส EV2 ซึ่งเป็นข้อมูลการลบบริการที่มีประเภท Printer (ตามข้อมูลระบุความต้องการรหัส F2) ข้อมูลที่มีรหัส EV3 ซึ่งเป็นข้อมูลการเปลี่ยนแปลงค่าคุณสมบัติของบริการที่มีประเภท Printer (ตามข้อมูลระบุความต้องการรหัส F2) และข้อมูลที่มีรหัส EV4 ซึ่งเป็นข้อมูลเปลี่ยนแปลงชนิดของบริการที่มีประเภท Bank มีค่าคุณสมบัติคือ Name = ISEL-Bank ReservedFund = 800000000 และ Interest_rate = 5.0 (ตามข้อมูลระบุความต้องการรหัส F1)

5.2.4 การทดสอบการกำหนดคุณสมบัติคุณภาพของบริการ

การทดสอบแบ่งออกได้เป็น 2 ส่วน คือ

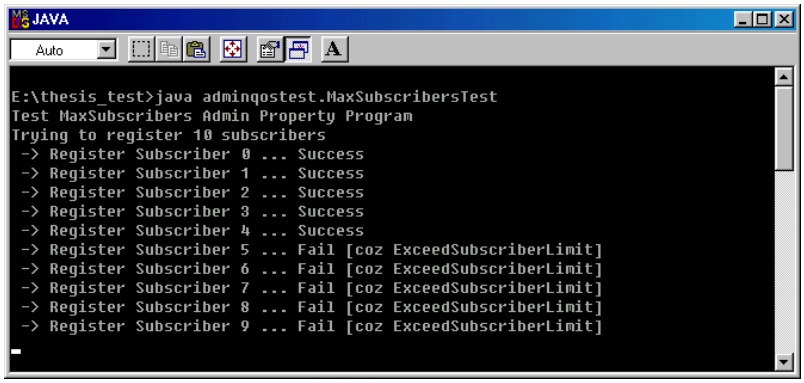
1. <u>การทดสอบคุณสมบัติสำหรับการบริหาร</u>

เป็นการทดสอบการทำงานของบริการเอสซีเอ็นตามคุณสมบัติที่ผู้ดูแลบริการกำหนด ประกอบด้วย

1.1 การทดสอบคุณสมบัติ MaxSubscribers

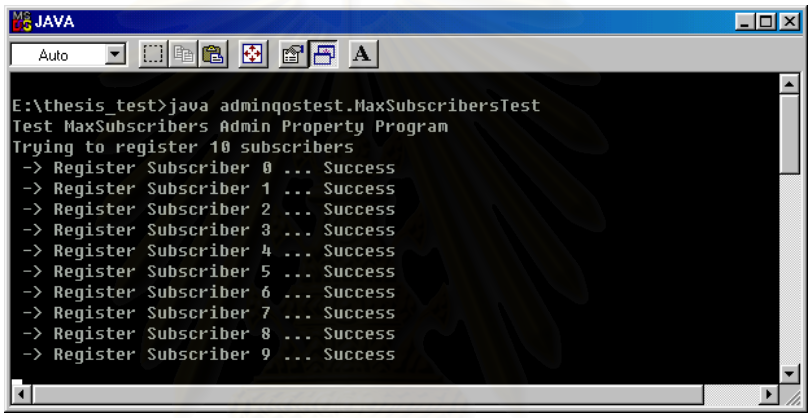
ทำการทดลองโดยกำหนดค่า MaxSubscribers ที่แตกต่างกัน คือ เท่ากับ 5 และ 10 โดยทำการลงทะเบียนผู้ต้องการรับทราบการเปลี่ยนแปลงบริการกับบริการเอสซีเอ็นทั้งหมด 10 คน

- <u>MaxSubscribers เท่ากับ 5</u> (ผลการทำงานแสดงได้ดังรูปที่ 5.8)



รูปที่ 5.8 ผลการทดสอบคุณสมบัติ MaxSubscribers โดยกำหนด MaxSubscribers = 5

- MaxSubscriber เท่ากับ 10 (ผลการทำงานแสดงได้ดังรูปที่ 5.9)



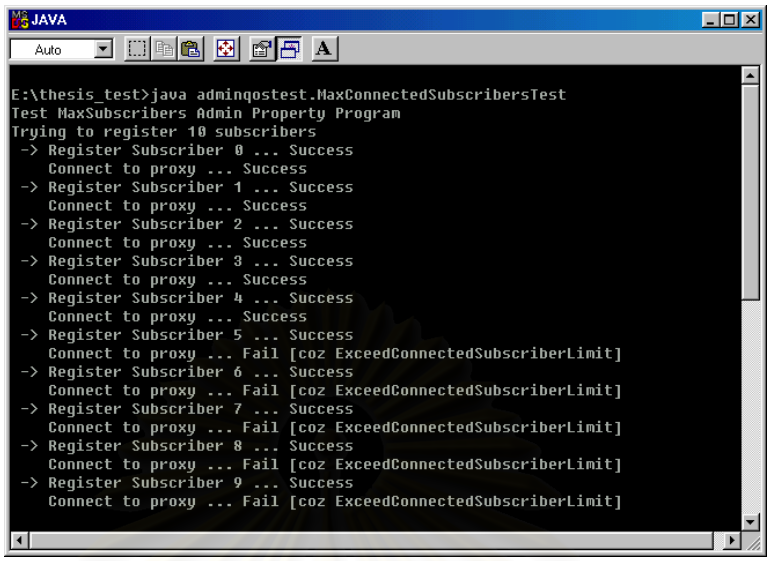
รูปที่ 5.9 ผลการทดสอบคุณสมบัติ MaxSubscribers โดยกำหนด MaxSubscribers = 10

จากผลการทดลองข้างต้นเมื่อทำการกำหนดให้ค่าคุณสมบัติ MaxSubscribers มีค่าเท่ากับ 5 จะไม่สามารถลงทะเบียนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการได้เกิน 5 คน นั่นคือ ผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการคนที่ 6 ถึง 10 จะไม่สามารถลงทะเบียนได้ แต่เมื่อทำการเปลี่ยนแปลงค่าคุณสมบัติ MaxSubscribers ให้มีค่าเท่ากับ 10 จะสามารถลงทะเบียนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการได้ครบทั้ง 10 คน ดังนั้นจึงสรุปได้ว่าการทำงานตามคุณสมบัติ MaxSubscribers มีความถูกต้องตามค่าที่ระบุ

1.2 การทดสอบคุณสมบัติ MaxConnectedSubscribers

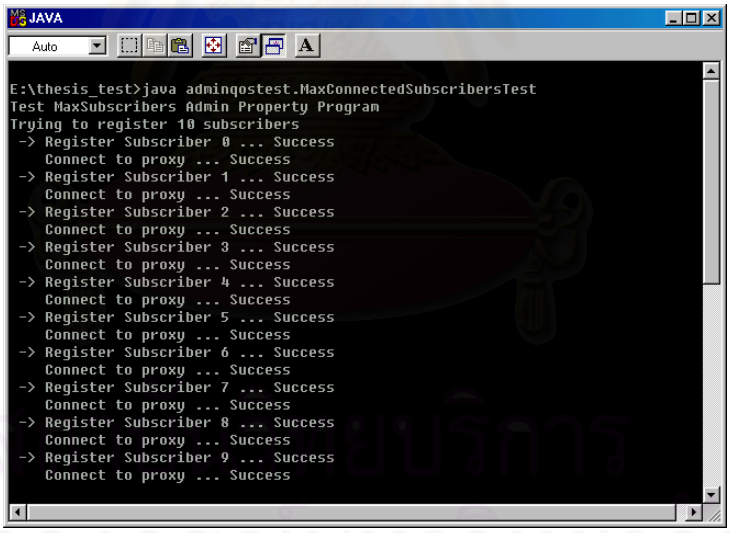
ทำการทดลองโดยกำหนดค่า MaxConnectedSubscribers ที่แตกต่างกัน คือ เท่ากับ 5 และ 10 (สำหรับค่า MaxSubscribers กำหนดไว้ที่ค่ามากกว่า 10) โดยทำการลงทะเบียนผู้ต้องการรับทราบการเปลี่ยนแปลงบริการกับบริการเอสซีเอ็นทั้งหมด 10 คน โดยแต่ละคนทำการสร้างวัตถุตัวแทนประเภท StructuredProxyPushSubscriber และทำการติดต่อเข้ากับวัตถุตัวแทน จากนั้นสังเกตผลการทำงาน

- MaxConnectedSubscribers เท่ากับ 5 (ผลการทำงานแสดงได้ดังรูปที่ 5.10)



รูปที่ 5.10 ผลการทดสอบคุณสมบัติ MaxConnectedSubscribers โดยกำหนด MaxConnectedSubscribers = 5

- กำหนด MaxSubscribers เท่ากับ 10 (ผลการทำงานแสดงได้ดังรูปที่ 5.11)



รูปที่ 5.11 ผลการทดสอบคุณสมบัติ MaxConnectedSubscribers โดยกำหนด MaxConnectedSubscribers = 10

จากผลการทดลองข้างต้นเมื่อทำการกำหนดให้ค่าคุณสมบัติ MaxConnected Subscribers มีค่าเท่ากับ 5 พบว่าผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการคนที่ 6 ถึง 10 จะไม่สามารถติดต่อกับบริการเพื่อขอรับข้อมูลการเปลี่ยนแปลงบริการได้ แต่เมื่อทำการเปลี่ยนแปลงค่าคุณสมบัติ MaxConnetedSubscribers ให้มีค่าเท่ากับ 10 พบได้ว่าผู้ที่ต้องการรับทราบการ

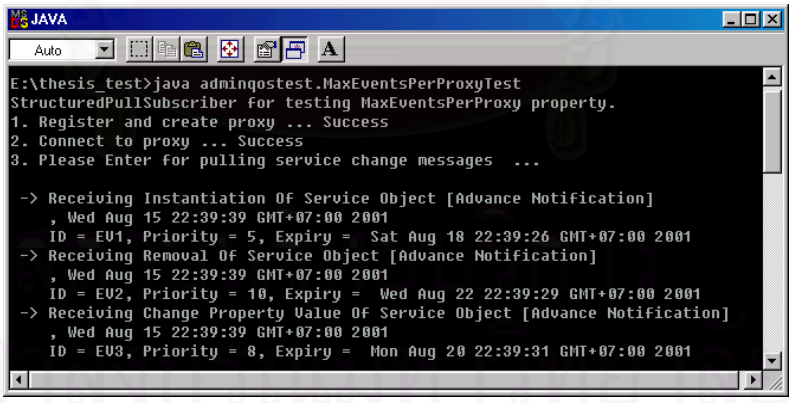
เปลี่ยนแปลงบริการทั้ง 10 คน สามารถติดต่อกับบริการได้ครบทั้งหมด ดังนั้นจึงสรุปได้ว่าการทำงานตามคุณสมบัติ MaxConnectedSubscribers มีความถูกต้องตามค่าที่ระบุ

1.3 การทดสอบคุณสมบัติ MaxEventsPerProxy

ทำการทดลองโดยกำหนดค่า MaxEventsPerProxy ที่แตกต่างกัน คือ เมื่อค่าเท่ากับ 3 และเมื่อค่าเท่ากับ 5 จากนั้นจึงทำการรันโปรแกรมซึ่งจำลองเป็นผู้ต้องการรับทราบการเปลี่ยนแปลงบริการซึ่งมีการทำงานประกอบด้วย

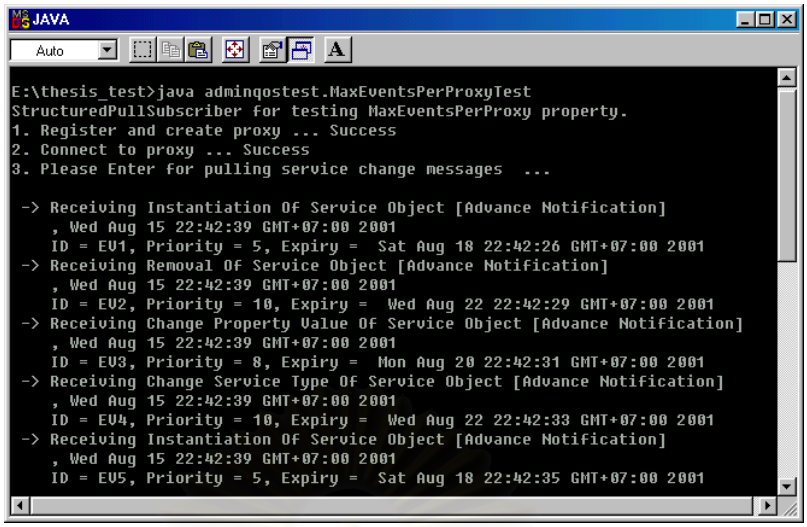
1. ลงทะเบียนและสร้างวัตถุตัวแทนประเภท StructuredProxyPullSubcriber
2. ติดต่อกับวัตถุตัวแทนเพื่อรับข้อมูลการเปลี่ยนแปลงบริการ
3. รอการกดคีย์ Enter เพื่อเรียกตัวกระทำการ pull_structured_event () สำหรับการดึงข้อมูลการเปลี่ยนแปลงบริการ โดยก่อนที่จะทำการกดคีย์ Enter เพื่อให้โปแกรมผู้ต้องการรับทราบการเปลี่ยนแปลงบริการทำการดึงข้อมูล จะทำการรันโปรแกรมผู้แจ้งการเปลี่ยนแปลงบริการ (ใช้โปรแกรมเดียวกับการทดสอบการจัดส่งและรับข้อมูลการเปลี่ยนแปลงบริการ ในหัวข้อที่ 5.2.2) ซึ่งจะทำการส่งข้อมูลการเปลี่ยนแปลงบริการจำนวน 5 ข้อมูล สำหรับผลการทดลองมีรายละเอียดดังต่อไปนี้

- กำหนด MaxEventsPerProxy เท่ากับ 3 (ผลการทำงานแสดงได้ดังรูปที่ 5.12) พบได้ว่าข้อมูลการเปลี่ยนแปลงบริการที่ได้รับมีจำนวน 3 ตัวเท่านั้น ซึ่งสอดคล้องกับขนาดของ MasEventsPerProxy ที่กำหนด



รูปที่ 5.12 ผลการทดสอบคุณสมบัติ MaxEventsPerProxy เมื่อกำหนดค่า MaxEventsPerProxy เท่ากับ 3

- กำหนด MaxEventsPerProxy เท่ากับ 5 (ผลการทำงานแสดงได้ดังรูปที่ 5.13) พบได้ว่าข้อมูลการเปลี่ยนแปลงบริการที่ได้รับมีจำนวน 5 ตัวครบตามที่ส่งทั้งหมด ซึ่งสอดคล้องกับขนาดของ MasEventsPerProxy ที่กำหนด



รูปที่ 5.13 ผลการทดสอบคุณสมบัติ MaxEventsPerProxy
โดยกำหนด MaxEventsPerProxy = 5

จากผลการทดลองข้างต้นสามารถสรุปได้ว่าการทำงานตามค่าคุณสมบัติ MaxEvents PerProxy มีความถูกต้องตามค่าที่กำหนด

2. <u>การทดสอบคุณสมบัติคุณภาพของบริการสำหรับผู้ต้องการรับทราบการเปลี่ยนแปลง</u>

เป็นการทดสอบการใช้งานส่วนต่อประสานตัวจัดการคุณภาพของบริการ (QoSAdmin Interface) และการทำงานตามค่าของคุณสมบัติที่กำหนด โดยการทดสอบจะกระทำโดยการใช้โปรแกรมทดสอบที่ประกอบด้วย

1. โปรแกรมจำลองเป็นผู้แจ้งการเปลี่ยนแปลงบริการ 1 ตัว (ใช้โปรแกรมเดียวกับทดสอบการจัดส่งและรับข้อมูลการเปลี่ยนแปลงบริการ ในหัวข้อที่ 5.2.2) โดยโปรแกรมจะทำการสร้างและส่งข้อมูลการเปลี่ยนแปลงบริการที่มีค่าความสำคัญ และค่าวัน-เวลาหมดอายุ (Expiry Time) ที่แตกต่างกันจำนวน 5 ข้อมูล กล่าวคือ การจัดส่งจะเรียงลำดับจากข้อมูลรหัส EV1, EV2, EV3, EV4, และ EV5 โดยข้อมูลแต่ละตัวจะมีค่าความสำคัญและวัน-เวลาหมดอายุแตกต่างกันดังนี้

ข้อมูลรหัส EV1 มีค่าความสำคัญเท่ากับ 5 และหมดอายุ วันที่ 19 สิงหาคม 2544 เวลา 17:32:00 นาฬิกา

ข้อมูลรหัส EV2 มีค่าความสำคัญเท่ากับ 10 และหมดอายุ วันที่ 23 สิงหาคม 2544 เวลา 17:32:02 นาฬิกา

ข้อมูลรหัส EV3 มีค่าความสำคัญเท่ากับ 8 และหมดอายุ วันที่ 21 สิงหาคม 2544 เวลา 17:32:04 นาฬิกา

ข้อมูลรหัส EV4 มีค่าความสำคัญเท่ากับ 10 และหมดอายุ วันที่ 23 สิงหาคม 2544 เวลา 17:32:06 นาฬิกา

ข้อมูลรหัส EV5 มีค่าความสำคัญเท่ากับ 5 และหมดอายุ วันที่ 19 สิงหาคม 2544 เวลา 17:32:08นาฬิกา

2. โปรแกรมจำลองเป็นผู้ต้องการรับทราบการเปลี่ยนแปลงบริการประเภท Structured PushSubscriber สำหรับทดสอบคุณสมบัติ OrderPolicy และ DiscardPolicy และโปรแกรมจำลองเป็นผู้ต้องการรับทราบการเปลี่ยนแปลงประเภท SequenceStructuredPushSubscriber เพื่อทดสอบคุณสมบัติ MaximumBatchsize และ PacingInterval (สำหรับผู้ต้องการรับทราบการเปลี่ยนแปลงบริการประเภทอื่นๆ ก็ได้ผลการทดสอบที่ถูกต้องเช่นกัน) โปรแกรมจะมีการทำงานหลักๆ ประกอบด้วย

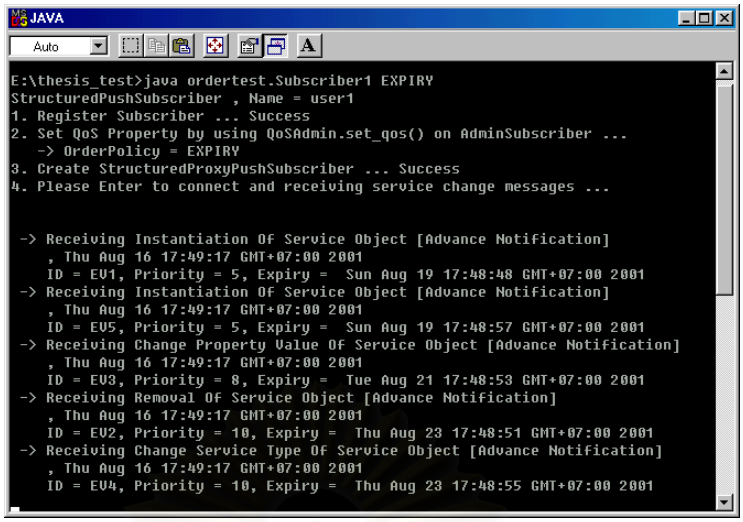
1. การลงทะเบียนเข้ากับบริการเอสซีเอ็น
2. เรียกใช้งานตัวกระทำการ set_qos () ของส่วนต่อประสาน QoSAdmin สำหรับการกำหนดค่าคุณสมบัติต่างๆ ให้กับวัตถุผู้ดูแลตัวแทน
3. สร้างวัตถุตัวแทน
4. รอรับข้อมูลการเปลี่ยนแปลงบริการเพื่อสังเกตผลการทำงาน

2.1 การทดสอบคุณสมบัติ OrderPolicy

ในที่นี้จะแสดงเฉพาะการกำหนดค่า OrderPolicy เป็น Expiry Time ส่วนค่าแบบอื่นก็ได้ผลการทดสอบที่ถูกต้องเช่นกัน

ผลการทดสอบ OrderPolicy = EXPIRY

เมื่อผู้ต้องการรับทราบการเปลี่ยนแปลงบริการเป็นประเภท StructuredPushSubscriber (รูปที่ 5.14)



รูปที่ 5.14 ผลการทำงานของโปรแกรม StructuredPushSubscriber สำหรับการทดสอบค่า OrderPolicy เท่ากับ EXPIRY

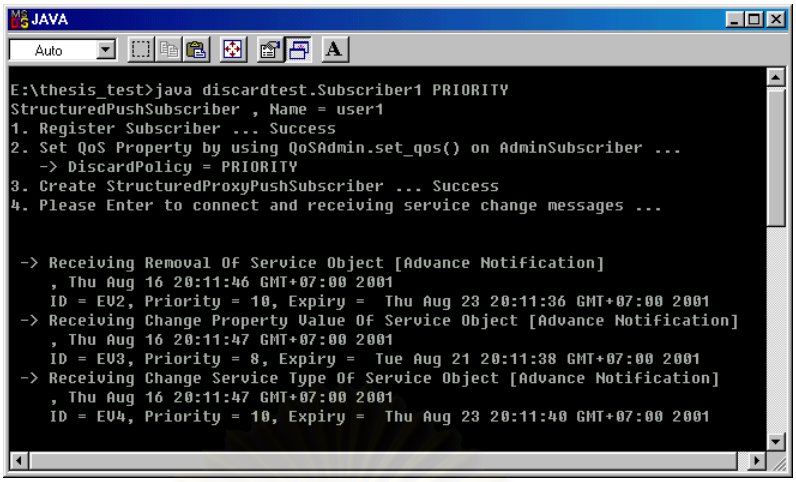
จากผลการทำงานในรูปที่ 5.14 พบได้ว่าโปรแกรมผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการประเภท StructuredPushSubscriber ได้รับข้อมูลการเปลี่ยนแปลงบริการในลำดับที่ถูกต้องตามค่าวัน-เวลาหมดอายุ โดยข้อมูลที่หมดอายุเร็วที่สุดจะได้รับการจัดส่งก่อนนั่นคือ ข้อมูลที่มีรหัส EV1, EV5, EV3, EV2 และ EV4 เรียงตามลำดับ

2.2 การทดสอบคุณสมบัติ DiscardPolicy

ในการทดสอบคุณสมบัติ DiscardPolicy นี้จะทำการกำหนดค่าคุณสมบัติ MaxEventsPerProxy ให้มีค่าเท่ากับ 3 เพื่อจำกัดขนาดของแถวคอยสำหรับเก็บข้อมูลการเปลี่ยนแปลงบริการ ซึ่งจะมีผลทำให้โปรแกรมผู้ต้องการรับทราบการเปลี่ยนแปลงบริการจะได้รับข้อมูลการเปลี่ยนแปลงบริการเพียง 3 ข้อมูลเท่านั้น ส่วนอีก 2 ข้อมูลจะถูกคัดทิ้ง ในที่นี้จะแสดงเฉพาะค่า DiscardPolicy เป็น Priority ส่วนค่าแบบอื่นก็ได้ผลการทดสอบที่ถูกต้องเช่นเดียวกัน

ผลการทดสอบ DiscardPolicy = PRIORITY

เมื่อผู้ต้องการรับทราบการเปลี่ยนแปลงบริการเป็นประเภท StructuredPushSubscriber (รูปที่ 5.15)



รูปที่ 5.15 ผลการทำงานของโปรแกรม StructuredPushSubscriber สำหรับการทดสอบค่า DiscardPolicy เท่ากับ PRIORITY

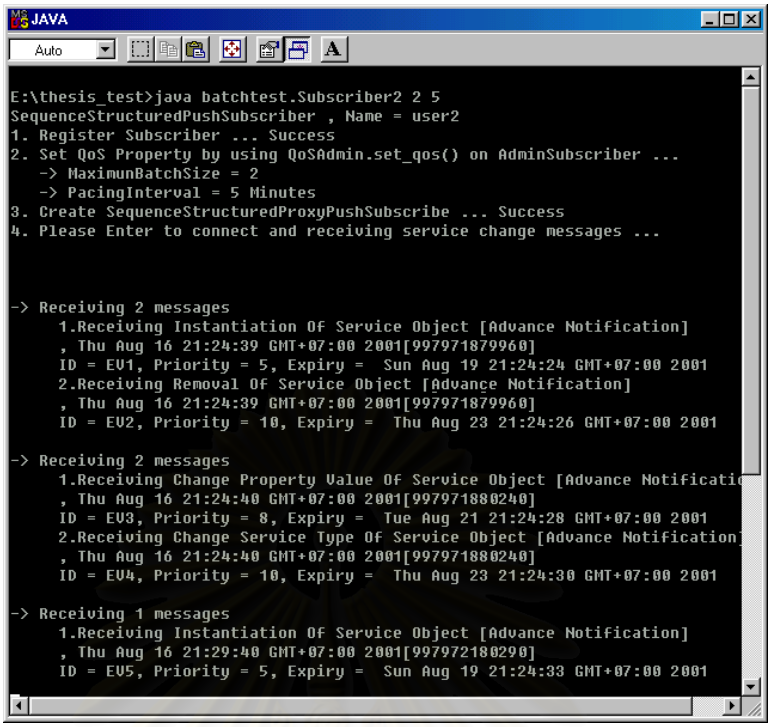
จากผลการทำงานในรูปที่ 5.15 พบได้ว่าโปรแกรมผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ StructuredPushSubscriber ได้รับข้อมูลที่มีรหัสประกอบด้วย EV2 EV3 และ EV4 ส่วนข้อมูลที่ถูกคัดทิ้งได้แก่ข้อมูลรหัส EV1 และ EV5 ซึ่งมีค่าความสำคัญต่ำสุด

2.3 การทดสอบคุณสมบัติ MaximumBatchSize และ PacingInterval

คุณสมบัติ MaximumBatchSize และ PacingInterval จะมีผลต่อการทำงานกับวัตถุตัวแทนประเภทที่จัดส่งข้อมูลการเปลี่ยนแปลงเป็นชุด ในที่นี้จะแสดงผลการทดสอบโดยใช้โปรแกรมจำลองผู้ต้องการรับทราบการเปลี่ยนแปลงประเภท SequenceStructuredPushSubscriber เท่านั้น ส่วนประเภท SequenceStructuredPullSubscriber ก็ได้ผลการทดสอบที่ถูกต้องเช่นกัน ในการทดสอบจะทำการทดลองกำหนดค่า MaximumBatchSize เท่ากับ 2 และค่า PacingInterval เท่ากับ 5 ผลการทดสอบสามารถแสดงได้ดังต่อไปนี้

<u>ผลการทดสอบ MaximumBatchSize = 2 และ PacingInterval = 5 นาที</u>

เมื่อผู้ต้องการรับทราบการเปลี่ยนแปลงบริการเป็นประเภท SequenceStructuredPush Subscriber (รูปที่ 5.16)



รูปที่ 5.16 ผลการทำงานของโปรแกรม SequenceStructuredPushSubscriber สำหรับการทดสอบค่า MaximumBatchSize เท่ากับ 2 และ PacingInterval เท่ากับ 5 นาที

จากผลการทำงานในรูปที่ 5.16 พบว่าข้อมูลที่วัตถุตัวแทนทำการจัดส่งให้กับโปรแกรมผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ SequenceStructuredPushSubscriber มีการรวบรวมส่งทีละ 2 ข้อมูลทันทีตามค่าของคุณสมบัติ MaximumBatchSize แม้ว่าเวลารอในการรวบรวมข้อมูลยังไม่ครบตามค่า PacingInterval ก็ตาม นอกจากนี้จะสังเกตได้ว่าข้อมูลชุดสุดท้ายจะมีเพียง 1 ข้อมูลเท่านั้นเนื่องจากข้อมูลที่ทำการจัดส่งทั้งหมดมีทั้งสิ้นเพียง 5 ข้อมูล สำหรับข้อมูลชุดสุดท้ายพบได้ว่าวัตถุตัวแทนจะทำการรอเพื่อเก็บรวบรวมข้อมูลให้ครบจำนวนเท่ากับ 2 ข้อมูลอยู่เป็นเวลาประมาณ 5 นาที (300000 มิลลิวินาที) ตามค่า PacingInterval แล้วจึงทำการจัดส่ง โดยเวลาที่ทำการรวบรวมสามารถคำนวณได้ดังนี้

เวลาที่ทำการเก็บรวบรวมโดยประมาณ = เวลาที่วัตถุเรียกกลับได้รับข้อมูลชุดสุดท้าย – เวลาที่วัตถุเรียกกลับรับข้อมูลชุดที่ 2

= 997972180290 – 997971880240 มิลลิวินาที

= 300050 มิลลิวินาที

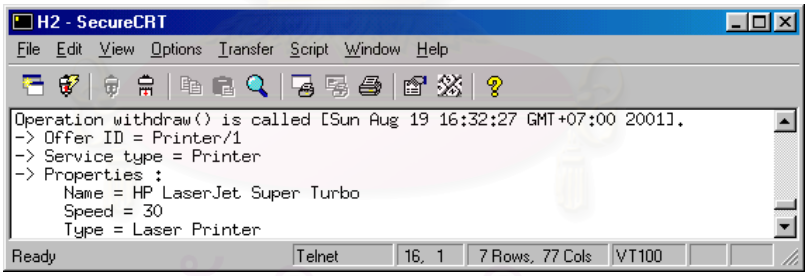
5.3 การทดสอบตัวตรวจบริการเทรดเดอร์

การทดสอบตัวตรวจบริการเทรดเดอร์กระทำโดยจำลองผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการขึ้นมา 1 ตัว สำหรับรอรับข้อมูลการเปลี่ยนแปลงบริการ จากนั้นทำการเรียกใช้งานตัวกระทำการของบริการเทรดเดอร์สำหรับการเปลี่ยนแปลงข้อมูลของบริการ แล้วทำการสังเกตข้อมูลการเปลี่ยนแปลงบริการที่ส่งมาให้กับผู้ที่ต้องการรับทราบการเปลี่ยนแปลง

ในที่นี้จะแสดงเฉพาะการดักการเปลี่ยนแปลงโดยตัวกระทำการ withdraw() ของบริการเทรดเดอร์ เพื่อยกเลิกบริการ และตัวกระทำการ modifyServiceOffer_NewType() ของส่วนเพิ่มขยายของบริการเทรดเดอร์เท่านั้น สำหรับการเปลี่ยนแปลงโดยตัวกระทำการอื่นๆ ก็ได้ผลการทดสอบที่ถูกต้องเช่นกัน

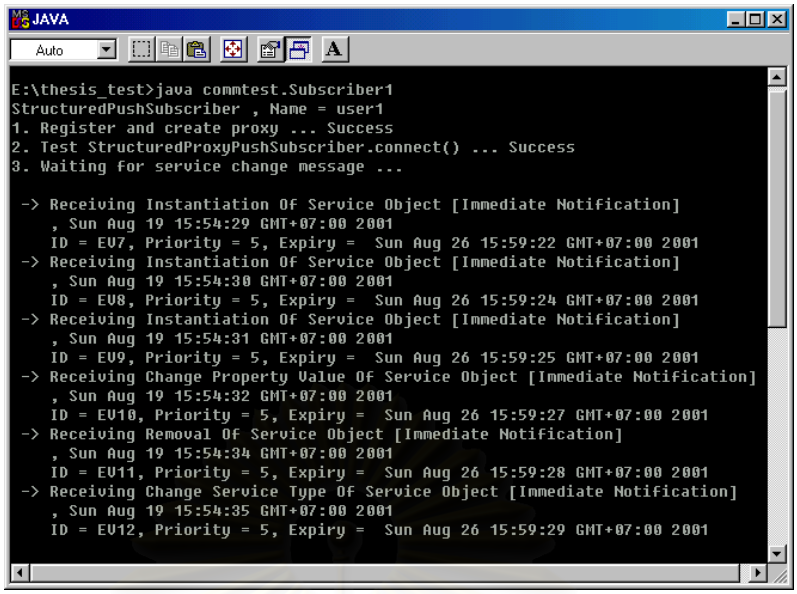
5.3.1 การยกเลิกบริการ (withdraw())

เมื่อผู้เปลี่ยนแปลงบริการทำการยกเลิกบริการ โดยเรียกตัวกระทำการ withdraw() จากบริการเทรดเดอร์ที่มีตัวตรวจบริการเทรดเดอร์ติดตั้งอยู่ ตัวตรวจบริการเทรดเดอร์จะทำงานโดยดักข้อมูลการลบบริการ และแสดงผลออกทางจอภาพ จากนั้นจึงส่งข้อมูลดังกล่าวไปยังตัวจัดการการแจ้งการเปลี่ยนแปลงบริการ การทดสอบเป็นการลบบริการที่มีชนิดของบริการคือ Printer ซึ่งมีค่า Offer Id คือ Printer/1 ผลการทำงานของตัวตรวจบริการเทรดเดอร์แสดงดังรูปที่ 5.17



รูปที่ 5.17 ผลการทำงานของตัวตรวจบริการเทรดเดอร์จากการเรียกตัวกระทำการ withdraw()

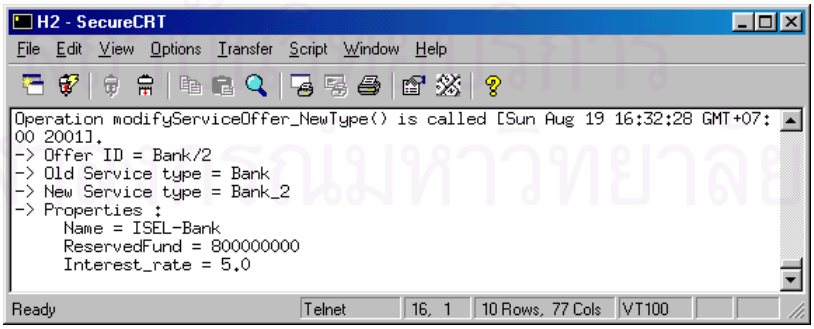
สำหรับตัวจัดการการแจ้งการเปลี่ยนแปลง เมื่อได้รับข้อมูลการยกเลิกจากตัวตรวจบริการเทรดเดอร์ ก็จะทำการจัดส่งข้อมูลดังกล่าวไปให้แก่ตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการเพื่อส่งต่อให้แก่ผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการต่อไป โดยในรูปที่ 5.18 ข้อมูลที่ 5 (รหัส EV11) แสดงการรับข้อมูลการยกเลิกบริการ Printer นี้



รูปที่ 5.18 การรับข้อมูลการเปลี่ยนแปลงบริการที่ตรวจพบโดยตัวตรวจบริการเทรดเดอร์ของโปรแกรมผู้ต้องการรับทราบการเปลี่ยนแปลงบริการ

5.3.2 การเปลี่ยนแปลงชนิดของบริการ (modifyServiceOffer_NewType())

เมื่อผู้เปลี่ยนแปลงบริการทำการเปลี่ยนแปลงชนิดของบริการโดยเรียกตัวกระทำการ modifyServiceOffer_NewType() จากมอดูลสนับสนุนการเปลี่ยนแปลงบริการที่เพิ่มขยายให้กับบริการเทรดเดอร์ที่มีตัวตรวจบริการเทรดเดอร์ติดตั้งอยู่ ตัวตรวจบริการเทรดเดอร์จะทำงานโดยดักข้อมูลการเปลี่ยนแปลงชนิดของบริการ และแสดงผลออกทางจอภาพ จากนั้นจึงส่งข้อมูลดังกล่าวไปยังตัวจัดการการแจ้งการเปลี่ยนแปลงบริการ การทดสอบเป็นการเปลี่ยนแปลงชนิดของบริการที่มีชนิดของบริการคือ Bank มีค่า Offer Id คือ Bank/2 มาเป็นบริการที่มีชนิดของบริการคือ Printer_2 ผลการทำงานของตัวตรวจบริการเทรดเดอร์แสดงดังรูปที่ 5.19



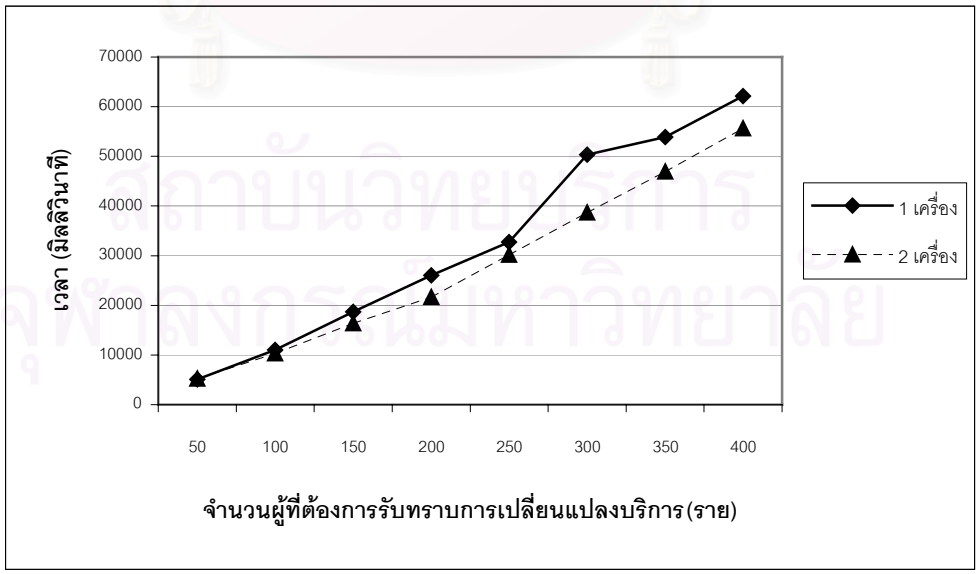
รูปที่ 5.19 ผลการทำงานของตัวตรวจบริการเทรดเดอร์จากการเรียกตัวกระทำการ modifyServiceOffer_NewType()

สำหรับตัวจัดการการแจ้งการเปลี่ยนแปลง เมื่อได้รับข้อมูลการเปลี่ยนแปลงชนิดของบริการจากตัวตรวจบริการเทรดเดอร์ก็จะทำการจัดส่งข้อมูลดังกล่าวไปให้แก่ตัวแทนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการเพื่อส่งต่อให้แก่ผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการต่อไป โดยในรูปที่ 5.18 ข้อมูลที่ 6 (รหัส EV12) แสดงการรับข้อมูลการเปลี่ยนแปลงชนิดของบริการนี้

5.4 ตัวอย่างการทดสอบประสิทธิภาพ

ในส่วนนี้เป็นตัวอย่างการทดสอบประสิทธิภาพของบริการเอสซีเอ็นสำหรับการจัดส่งข้อมูลการเปลี่ยนแปลงบริการไปยังผู้ต้องการรับทราบการเปลี่ยนแปลงบริการ

การทดลองกระทำโดย จำลองผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการประเภท StructuredPushSubscriber ในจำนวนที่แตกต่างกัน คือ จำนวน 50, 100, 150, 200, 250, 300, 350, และ 400 ราย จากนั้นทำการสุ่มสร้างข้อมูลการเปลี่ยนแปลงประเภทต่างๆ และทำการส่งข้อมูลการเปลี่ยนแปลงจำนวน 1 ข้อมูลเข้าสู่บริการเอสซีเอ็น หาความสัมพันธ์ระหว่างเวลาที่ใช้ในการจัดส่งข้อมูลไปยังผู้ที่ต้องการรับทราบการเปลี่ยนแปลงครบทุกรายกับจำนวนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการ การทดลองจัดทำขึ้น 2 ชุด โดยชุดแรกทำการจำลองผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการทั้งหมดอยู่บนเครื่องคอมพิวเตอร์เพียงเครื่องเดียว และชุดที่สองทำการจำลองผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการให้แยกอยู่ในเครื่องคอมพิวเตอร์จำนวน 2 เครื่องในปริมาณที่เท่ากัน ผลการทดลองแสดงได้ดังกราฟในรูปที่ 5.20



รูปที่ 5.20 เวลาที่ใช้ในส่งข้อมูลการเปลี่ยนแปลงบริการจำนวน 1 ข้อมูลไปยังผู้ต้องการรับทราบการเปลี่ยนแปลงบริการ

จากกราฟในรูปที่ 5.20 พบว่า เมื่อจำนวนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการเพิ่มขึ้นเวลาที่ใช้ในการจัดส่งข้อมูลจะมีค่าเพิ่มมากขึ้นด้วย ซึ่งการเพิ่มขึ้นนั้นมีลักษณะเป็นแบบเชิงเส้น สำหรับผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการจำนวน 400 รายนั้น เวลาที่บริการเอสซีเอ็นใช้ในการจัดส่งข้อมูลไปยังผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการครบทั้งหมด ใช้เวลาประมาณ 1 นาที (60000 มิลลิวินาที) ซึ่งให้ผลเป็นที่ยอมรับได้ และจากการทดลองแบ่งการจำลองผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการให้อยู่ในเครื่องคอมพิวเตอร์จำนวน 2 เครื่อง พบได้ว่า เวลาที่บริการเอสซีเอ็นใช้ในการจัดส่งข้อมูลการเปลี่ยนแปลงบริการไปยังผู้ที่ต้องการรับทราบการเปลี่ยนแปลงมีค่าน้อยกว่าเมื่อทำการจำลองให้ผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการทั้งหมดอยู่ภายในเครื่องคอมพิวเตอร์เครื่องเดียวเพียงเล็กน้อยเท่านั้น นอกจากนี้เมื่อจำนวนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการมีจำนวนเพิ่มขึ้น การแบ่งผู้ที่ต้องการรับทราบการเปลี่ยนแปลงไปยังเครื่องคอมพิวเตอร์มากกว่า 1 เครื่อง จะช่วยลดเวลาในการจัดส่งข้อมูลการเปลี่ยนแปลงได้มากขึ้น

บทที่ 6
สรุปผลการวิจัยและข้อเสนอแนะ

ในบทนี้จะกล่าวถึงผลสรุปของงานวิจัย ปัญหาและข้อจำกัดต่างๆ ของระบบ รวมทั้งข้อเสนอแนะที่สามารถนำไปปรับปรุงและพัฒนาได้ต่อไปในอนาคต โดยมีรายละเอียดดังต่อไปนี้

6.1 สรุปผลการวิจัย

งานวิจัยนี้ได้ทำการออกแบบและพัฒนาบริการสำหรับแจ้งการเปลี่ยนแปลงบริการไปยังผู้ใช้บริการในระบบกระจายคอร์บา โดยผลที่ได้รับคือ โครงสร้างและต้นแบบของบริการเอสซีเอ็นที่รองรับต่อการเปลี่ยนแปลงประเภทต่างๆ ของบริการในระบบกระจาย อันได้แก่ การเพิ่มบริการใหม่ การยกเลิกบริการ การเปลี่ยนแปลงค่าคุณสมบัติของบริการ และการเปลี่ยนแปลงชนิดของบริการ บริการเอสซีเอ็นได้รับการออกแบบให้สามารถรับข้อมูลการเปลี่ยนแปลงบริการจากทั้งผู้เปลี่ยนแปลงบริการโดยตรงโดยมีจุดประสงค์เพื่อเป็นการแจ้งการเปลี่ยนแปลงบริการล่วงหน้าก่อนที่จะมีการเปลี่ยนแปลงจริง และจากตัวตรวจบริการเทรดเดอร์ซึ่งเป็นการตรวจสอบและแจ้งการเปลี่ยนแปลงบริการโดยอัตโนมัติในขณะที่ข้อมูลข้อเสนอของบริการที่อยู่ในบริการเทรดเดอร์ได้รับการเปลี่ยนแปลง นอกจากนี้บริการเอสซีเอ็นยังสนับสนุนต่อการแจ้งในรูปแบบต่างๆ โดยกำหนดให้ผู้ต้องการรับทราบการเปลี่ยนแปลงบริการสามารถกำหนดรูปแบบการจัดส่งรวมทั้งรูปแบบของข้อมูลที่ใช้ในการจัดส่งได้

จากการทดสอบการทำงานของบริการเอสซีเอ็นในส่วนต่างๆ ที่ประกอบด้วย การทดสอบการลงทะเบียนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการและการจัดสร้างวัตถุตัวแทนประเภทต่างๆ การทดสอบการจัดส่งและรับข้อมูลการเปลี่ยนแปลงบริการ การทดสอบการกำหนดข้อมูลระบุความต้องการ การทดสอบการกำหนดคุณสมบัติคุณภาพของบริการ และการทดสอบการทำงานของตัวตรวจบริการเทรดเดอร์ พบได้ว่าบริการเอสซีเอ็นสามารถทำงานได้อย่างถูกต้องตามที่กำหนด สำหรับการทดสอบประสิทธิภาพโดยทำการวัดเวลาที่ใช้ในการจัดส่งข้อมูลการเปลี่ยนแปลงบริการไปยังผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการจำนวนต่างๆ นั้น พบได้ว่าที่จำนวนผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการจำนวน 400 คน ใช้เวลาในการจัดส่งประมาณ 1 นาที (60000 มิลลิวินาที) ซึ่งเป็นค่าที่สามารถยอมรับได้ แต่อย่างไรก็ตามประสิทธิภาพในการจัดส่งข้อมูลการเปลี่ยนแปลงบริการยังขึ้นกับปัจจัยอื่นๆ ด้วย เช่น การแบ่งการจำลองผู้ที่ต้องการรับทราบการเปลี่ยนแปลงบริการไปยังเครื่องคอมพิวเตอร์หลายๆ เครื่อง หรือ ขนาดของข้อมูลการเปลี่ยนแปลงบริการ เป็นต้น

6.2 ปัญหาและข้อจำกัดของงานวิจัย

1. บริการเอสซีเอ็นในขณะนี้จำกัดอยู่ที่การทำงานร่วมกับบริการเทรดเดอร์เพียงตัวเดียวโดยที่ไม่สามารถรองรับการทำงานในระบบที่มีบริการเทรดเดอร์ที่เชื่อมต่อกันได้ (Trader Federation)
2. บริการเอสซีเอ็นยังไม่สามารถเชื่อมต่อกันเองเพื่อแบ่งภาระการทำงานกันได้
3. การทำงานของบริการเอสซีเอ็นส่วนใหญ่ขึ้นอยู่กับการทำงานของบริการแจ้งเหตุการณ์ที่นำมาใช้ ในปัจจุบันบริการแจ้งเหตุการณ์ที่ใช้ในระบบกระจายคอร์บาส่วนใหญ่ยังคงอยู่ในระหว่างการพัฒนาและไม่มีตัวเลือกให้ใช้งานได้มากนัก ซึ่งพบได้ว่าบริการแจ้งเหตุการณ์ที่นำมาใช้ยังคงมีการทำงานที่จำกัด เช่น ลำดับการได้รับเหตุการณ์จะมีความผิดพลาดได้ในกรณีที่มีจำนวนผู้ต้องการรับทราบการเปลี่ยนแปลงจำนวนมากและมีอัตราของการเกิดเหตุการณ์การเปลี่ยนแปลงบริการที่สูง เนื่องจากการจัดลำดับเหตุการณ์เป็นการจัดลำดับที่ฝั่งผู้ส่ง
4. บริการแจ้งเหตุการณ์ที่นำมาใช้มีการใช้งานหน่วยความจำค่อนข้างสูง

6.3 ข้อเสนอแนะ

1. ควรปรับปรุงให้บริการเอสซีเอ็นสามารถรองรับต่อการทำงานที่เชื่อมต่อกันได้รวมทั้งการรองรับการทำงานในระบบที่มีบริการเทรดเดอร์เชื่อมต่อกันอยู่ได้

รายการอ้างอิง

[1] Object Management Group. The Common Object Request Broker: Architecture and Specification Revision 2.2.1998.

[2] Object Management Group. Trading Object Service Specification Revised Edition 1997.

[3] Tomono, M. An Event Notification Framework based on Java and CORBA. In Proceedings of the 6th IFIP/IEEE International Symposium on Distributed Management for the Networked Millennium 1999 : 563 –576.

[4] Rifkin, A. and Khar, R. A Survey of Event Systems, Available from : http://www.cs.caltech.edu/~adam/ isen/event-systems.html

[5] Object Management Group. Notification Service Joint Revised Submission. OMG TC Document telecom/98-11-01 1998.

[6] Bray, T., Paoli, J. And McQueen, S. Extensible Markup Language (XML) 1.0 Specification. World Wide Web Consortium Recommendation. 10 February 1998, Available from: http://www.w3.org/TR/REC-xml

[7] Apparao, V., et al. Document Object Model (DOM) Level 1 Specification Version 1.0: World Wide Web Consortium Recommendation. 1998, Available from: http://www.w3.org/TR/REC-DOM-Level-1

[8] Wood, L. Programming the Web: The W3C DOM Specification. IEEE Internet Computing. 31 (January-February 1999): 48-54.

[9] Object Management Group. ORB Interface Type Version Management Request for Proposals, OMG TC Document ORB/96-01-06, 1996.

[10] Senivongse, T. Evolution Transparency for Distributed Service Types. Doctoral Dissertation University of Kent at Canterbury UK 1997.

[11] Thongsisod, K. and Kongkriengkrai, T. A Notification System for Service Providers of Services with Evolved Interfaces in CORBA Distributed System, Senior Project Report, Dept. of Computer Engineering, Chulalongkorn University, Bangkok, Thailand, 1999.

[12] Jae, J.Y and Soo, D.K. SMARTS: A Smart CORBA Trader Service. In Proceeding of 6th Asia Pacific on Software Engineering Conference (APSEC) 1999: 166-173.

[13] Object Management Group. A Discussion of the Object Management Architecture.1997.

[14] Object Management Group. CORBAservices : Common Object Services Specification. 1998.

[15] Senivongse, T. and Nanekrangsan, W. An approach to Standardizing Distributed Service Descriptions Format using XML. In Proceedings of the 3rd IFIP WG 6.1 International Working Conference on Distributed Applications and Interoperable Systems (DAIS'2001), Krakow, Poland, 2001.

[16] Inprise VisiBroker for Java, Available from: http://www.inprise.com

[17] JacORB – a free Java ORB, Available from: http://jacorb.inf.fu-berlin.de

[18] DSTC dCon 2.2, Available from: http://www.dstc.com

[19] InstantDB, Available from: http://www.lutris.com

ภาคผนวก

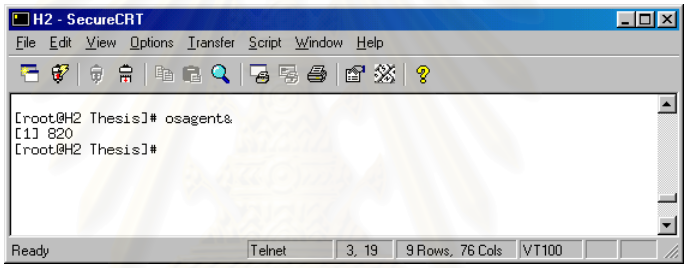
ภาคผนวก ก
ลำดับการทำงานของระบบและการเรียกใช้ส่วนต่าง ๆ ของบริการเอสซีเอ็น

ลำดับการเริ่มต้นบริการเอสซีเอ็นและบริการอื่นๆ ที่เกี่ยวข้อง เป็นดังนี้

1. เริ่มโปรแกรม osagent ของวิสิโบรกเกอร์ ด้วยคำสั่ง

```
osagent [-options]
```

คำสั่ง osagent สามารถระบุพอร์ตด้วยการระบุค่าในตัวแปรสภาวะ (Environment Variable) OSAGENT_PORT หรือทางอาร์กิวเมนต์ของคำสั่ง เช่น osagent –p=14001 ได้ ทั้งนี้เพื่อให้สามารถแยกการทำงานของระบบต่างๆ ออกเป็นโดเมนได้ โดยหากไม่ระบุโปรแกรมจะใช้ค่าโดยปริยาย คือ 14000 (รูปที่ ก1)

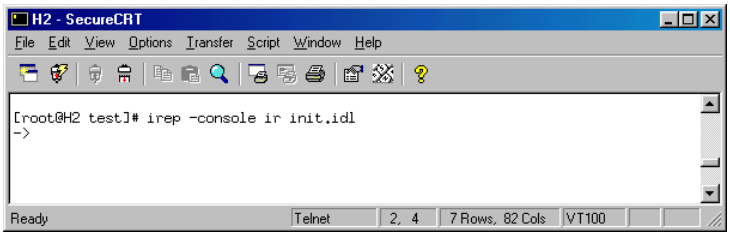


รูปที่ ก1 จอภาพแสดงการเริ่มโปรแกรม osagent

2. เริ่มโปรแกรมคลังส่วนต่อประสาน ด้วยคำสั่ง

```
irep [-options] <irep name> [<idl storage file name>]
```

ผู้ใช้สามารถใช้ทางเลือก –console เพื่อให้โปรแกรมทำงานในโหมดอักษร (Text Mode) ได้ และสามารถระบุชื่อคลังส่วนต่อประสานเพื่อเป็นชื่ออ้างอิงในการเรียกใช้ และสามารถกำหนดชื่อไฟล์ที่เก็บข้อมูลส่วนต่อประสานได้ (รูปที่ ก2)



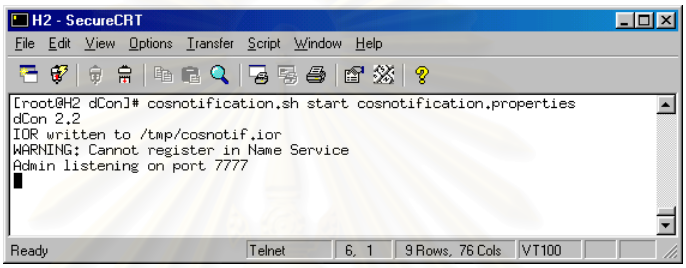
รูปที่ ก2 จอภาพแสดงการเริ่มโปรแกรมคลังส่วนต่อประสาน

3. เริ่มบริการแจ้งเหตุการณ์ของ dCon ด้วยคำสั่ง

cosnotification.sh start [property file] ยูนิกซ์ หรือ

cosnitfication start [property file] วินโดวส์

ผู้ใช้สามารถระบุไฟล์คุณสมบัติเพื่อกำหนดการทำงานเริ่มต้นให้กับบริการแจ้งเหตุการณ์ได้ โดยหากไม่ระบุบริการก็จะทำงานด้วยค่าโดยปริยายที่บริการกำหนด (รูปที่ ก3)

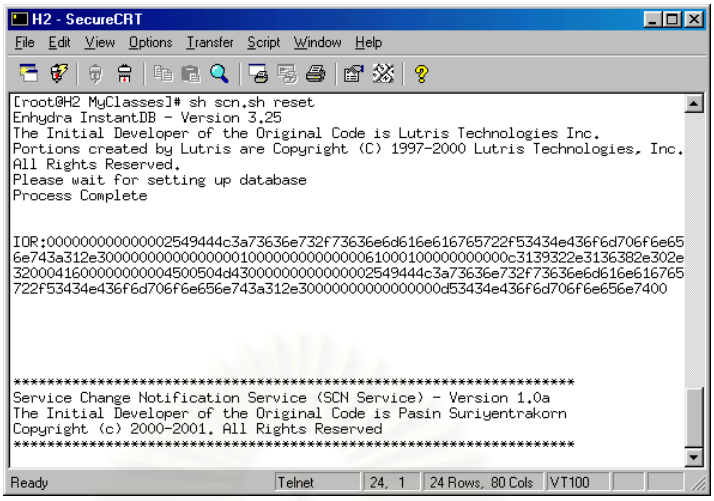


รูปที่ ก3 จอภาพแสดงการเริ่มโปรแกรมบริการแจ้งเหตุการณ์

4. เริ่มบริการเอสซีเอ็น ด้วยคำสั่ง

java -DORBservices=CosTrading scns.scnmanager.SCNManager [reset start stop]

อาร์กิวเมนต์ของการเริ่มบริการเอสซีเอ็นมี ได้แก่ -DORBservices=CosTrading เป็นการระบุว่าต้องการใช้งานบริการเทรดเดอร์ โดยที่scns.scnmanager.SCNManager เป็นไฟล์เริ่มการทำงานของบริการเอสซีเอ็น ส่วน[reset start stop] เป็นทางเลือกให้กับผู้ใช้ โดย ตัวเลือก reset เป็นการสั่งเริ่มต้นบริการโดยมีการเริ่มต้นค่าต่างๆ ใหม่ทั้งหมด ตัวเลือก start เป็นการสั่งเริ่มต้นบริการโดยไม่มีการเริ่มต้นค่าใหม่ และตัวเลือก stop เป็นการสั่งหยุดการทำงานของบริการเอสซีเอ็น (รูปที่ ก4)

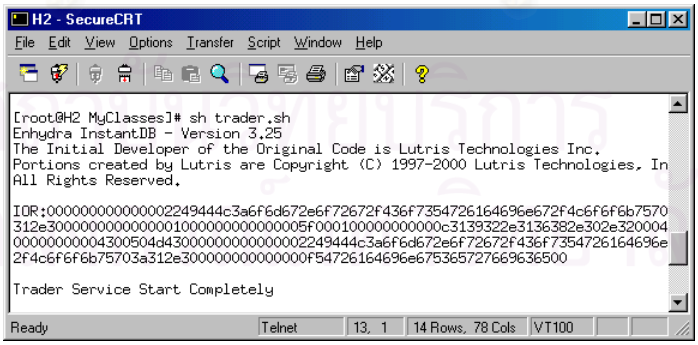


รูปที่ ก4 จอภาพแสดงการเริ่มโปรแกรมบริการเอสซีเอ็น

5. เริ่มบริการเทรดเดอร์ ด้วยคำสั่ง

```
java -DORBservices=
scns.TraderMonitorAgentjacorb.trading.TradingService TS_Ref
```

อาร์กิวเมนต์ของการเริ่มบริการเทรดเดอร์ ได้แก่ -DORBservices=scns.Trader MonitorAgent เป็นการระบุการใช้งานบริการเทรดเดอร์ร่วมกับตัวตรวจบริการเทรดเดอร์ ด้วยการระบุชื่อเพกเกจของตัวตรวจบริการเทรดเดอร์ ส่วน jacorb.trading. TradingService เป็นไฟล์เริ่มการทำงานของบริการเทรดเดอร์ และ TS_Ref เป็นการระบุชื่อไฟล์สำหรับเก็บข้อมูลอ้างถึงวัตถุของบริการเทรดเดอร์ (รูปที่5ก)



รูปที่ 5ก จอภาพแสดงการเริ่มโปรแกรมบริการเทรดเดอร์

ภาคผนวก ข
ผลงานตีพิมพ์

1. การประชุมทางวิชาการวิทยาการและวิศวกรรมคอมพิวเตอร์แห่งชาติ ครั้งที่ 4 (The 4th National Computer Science and Engineering Conference (NCSEC 2000)) เมื่อวันที่ 16-17 พฤศจิกายน 2543 ในบทความเรื่อง An Approach for Service Change Notification in Distributed System โดยผู้แต่งคือ Pasin Suriyentrakorn and Twittie Senivongse

An Approach for Service Change Notification in Distributed Systems

Pasin Suriyentrakorn[1] and Twittie Senivongse[2]
Department of Computer Engineering, Faculty of Engineering, Chulalongkorn University
Pathumwan, Bangkok, Thailand 10330
Phone: (66-2) 2186991, Fax: (66-2) 2186955
E-mail: g42psr@cp.eng.chula.ac.th[1], twittie.s@chula.ac.th[2]

Abstract: In distributed environment, resource services are scattered and their clients can be everywhere. This characteristic makes it difficult to control the impact of service change on the clients and so their current operation can be affected or interrupted. This paper proposes a CORBA-based architecture for a service change notification system that can notify and help users within the system in dealing with various kinds of service changes. The notification system will interface with service providers and a directory service called trader to acquire change information. Users can subscribe for change events of interest specifying various event delivery styles and notification formats, and will be informed of the changes.
Key words: Change management, notification service, CORBA

1. Introduction

In distributed object systems such as RM-ODP [1], DCOM [2], and CORBA[3], a service object is prone to evolution during its lifetime due to change in user requirements, technology change, or unexpected external factors. However, change management for distributed services is more complicated than that for software in non-distributed systems. Changes in distributed services may directly affect their clients. For example, when a service object evolves in such a way that its interface has changed, all of its clients will not be able to use it under the same protocol until they are modified and their systems rebuilt accordingly. This is a very difficult mission to achieve in an open environment.

Change management for distributed systems has not been fully researched or supported despite its importance. The OMG once requested for a proposal for a change management service for CORBA [4] but it was not properly responded because the goal for change management was, at that time, inferior to the goal for cross-vendor-platform interoperability. To date, there are few attempts to provide for change management. The evolution transparency mechanism [5] makes change in service interfaces transparent to clients of those services by providing mediator objects that can convert requests of old-version interfaces to those of new-version ones. This will make existing clients functioned with newer services but the mechanism is ad hoc because eventually those clients will need to evolve themselves as well. The work in [6] supports change management for the service provider side by introducing a notification system that can inform service providers of change in service interfaces so that they can evolve service instances correspondingly. We still lack of a change notification system that also supports the client side.

We propose an approach to notify service changes to users of a CORBA distributed system. A notification message can report on a change event and current status of a service object. Any interested parties (e.g. client providers, service providers, application users) can subscribe for change events of interest and will be notified whenever corresponding changes occur.

This paper reports on our progress to design and develop the change notification system and its remainder is organised as follows. In Section 2, we begin with the classification of notification-triggering events that are supported by our architecture. Then we describe the design of the notification system in Section 3 followed by the detail of subscription data model and notification data model in Section 4. Section 5 concludes the paper with the status of the current prototype implementation and discussions of some future work.

2. Classification of Notification Triggering Events

In general, a service object is an *offer* (i.e. instance) of a particular functionality or *service type*. The service type model of CORBA [7] is described by the following BNF:

```
Service<ServiceTypeName>[:<BaseServiceTypeName>
      [,<BaseServiceTypeName>]*] {
      interface <InterfaceTypeName>;
      [[mandatory][read-only] property <IDL Type>
      <PropertyName>; ]*
};
```

A service type comprises an interface type definition and a service property definition. An interface type definition refers to an interface signature that determines the way objects of this service type can be communicated with, i.e. operational methods and their parameters. A service property definition represents non-computational or behavioural aspects of the service type and is a collection of property names and property types. A service type can inherit interface definitions and property definitions from other service types also. Service objects of the same service type can be contacted via the same set of methods defined in the interface type definition of the service type. They also possess the same set of properties defined in the property definition of the service type but each of them has its own property values.

For example, we may have a service type 'Account' that is described by an interface 'Acc' and properties 'accno' and 'owner, both of type string. The interface 'Acc' supports operations 'withdraw ()' and 'deposit()'. We may have two objects of this service type 'Account'. One has 'accno' = 4324877463 and 'owner' = John Smith, and the other has 'accno' = 8735295637 and 'owner' = Joe Blogg. Both objects respond to messages of 'withdraw()' and 'deposit()' operations.

According to the description above, we classify into four groups the events on service objects that can trigger notification to their users and will cause different reactions from them.

Event 1: Change of service type of object. This comprises changes in various parts of an existing service type description and will require modification in parts of client programs. The change includes

- Change of service type name
- Change of interface signature
- Change of property definition, i.e. property name and property type
- Change of base service type - This will be reflected as changes of interface signature and property definition that are inherited.

Event 2: Change of property value of object. This refers to change of property values of a particular object and has no effect on other objects of the same service type. This change can affect non-computational preference of service usage but may not require change in client programs.

Event 3: Instantiation of object. This refers to the creation of a new object either of an existing service type or a newly introduced service type. The notification is for information purpose.

Event 4: Removal of object. This refers to a permanent removal of an existing object from the system and requires modification of client programs.

3. Architecture of Service Change Notification System

Figure 1 presents the overall architecture of the change notification system.

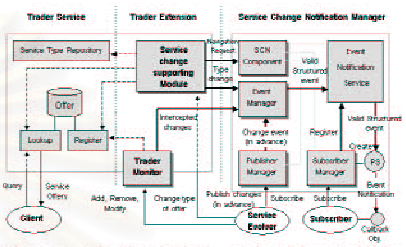


Figure 1. Service Change Notification System Architecture

Change events can be notified either in advance or at the time of their occurrence. For advance notification, service evolvers will first have to subscribe themselves to our system before they can actually publish advance change information. In the case of immediate notification, our Trader Monitor can automatically detect changes in service objects by monitoring a directory service called trader [7], which maintains service descriptions of all service objects and provides operations to manage those descriptions. The Service Change Notification Manager (SCN Manager) is the core of our system. It gathers subscription information from service evolvers who want to publish advance change information and from subscribers who are interested in specific change events. It then later generates appropriate notification messages with the help of a CORBA Event Notification Service whenever it is informed of change events by the Trader Monitor or service evolvers. ProxySubscriber (PS) obtains notification messages and takes care of their presentation and delivery to corresponding subscribers. The architecture is explained in more details below.

3.1 Trader Extension

We associate with the trader a couple of extensions.

- **Trader Monitor.** We can monitor the trader to detect service changes by intercepting request messages sent by service evolvers to the trader to manage service descriptions. Note that a CORBA trader supports operations to change property values of a service object, to add a new service object, and to remove an existing

object (i.e. event 2-4 in Section 2) only. Interception of messages indicates change events that are then forwarded to the SCN Manager (c.f. Event Manager in Section 3.2). We adopt the model a CORBA server interceptor here.

- **Service Change Supporting Module.** This module is added to the trader to facilitate the work of our notification system. As the trader does not provide an operation to change existing service type definitions (event 1 in Section 2), this module provides an interface for such an operation for service evolvers. Invocation of this operation generates a change event to be forwarded to the SCN Manager (c.f. Event Manager in Section 3.2). This added interface also provides operations for service evolvers and subscribers to respectively to look for the object references of the Publisher Manager object and Subscriber Manager object in order to subscribe themselves to the system later (c.f. Publisher Manager and Subscriber Manager in Section 3.2).

3.2 Service Change Notification Manager (SCN Manager)

The SCN Manager is responsible for two main functions, i.e. managing subscriber and service evolver information, and filtering and dispatching service change notification. It consists of five components: SCN Component, Subscriber Manager, Publisher Manager, Event Manager, and Event Notification Service.

- **SCN Component** is a navigator interface that provides contact points (i.e. object references) to the other four components of the SCN Manager.
- **Subscriber Manager** accepts registrations from subscribers and maintains their information, e.g. id, password, etc. In addition, it is a factory object [8] that are responsible for creating and destroying corresponding ProxySubscriber objects for registered subscribers (see Section 3.3.).
- **Publisher Manager** accepts registrations from service evolvers and maintains their information e.g. id, password, address, etc. It also receives advance change information from service evolvers and forwards it to the Event Notification Service through the Event Manager.
- **Event manager** receives all change information from the trader extension and the Publisher Manager and transformed it into valid structured events for the Event Notification Service.
- **Event Notification Service** is an event middleware [9] responsible for filtering and dispatching any arrival events to subscribers who have subscribed their interests in those events. We adopt the model of CORBA CosNotification Service [10] here.

3.3 ProxySubscriber

The ProxySubscriber is responsible for forwarding event messages, received from the Event Notification Service, to registered subscribers in several event delivery styles and formats according to its type. Event delivery styles can be either push or pull [11]. Push style means the ProxySubscriber will notify its subscriber, or in fact the subscriber's callback object (see Section 4.1.3), whenever it receives notification of a particular event. Pull style means a subscriber will query its ProxySubscriber whether a particular change event has occurred. In our system, subscribers can specify, when subscribing for particular events, which formats they desire for notification messages of those events. An event message can be sent as either an individual object of a particular structure, an XML document [12], or a sequence of structured objects or XML documents. The type of a ProxySubscriber is then determined by the delivery style and event data format that it supports, e.g. StructuredProxyPush, XMLProxyPull, SequenceProxyPull, SequenceXMLProxyPush etc. From the example in Section 2, a subscriber may subscribe to be notified when an object Account whose owner is John Smith has an interface change, and may specify that the notification should be sent as an XML document as soon as the change occurs (push model). A ProxySubscriber of type XMLProxyPush will be created for this subscriber.

4. Subscription and Notification Data Models

In this section, we focus on the data model of subscription information from subscribers, and the data model of notification messages from service evolvers to the system and from the system to subscribers.

4.1 Subscription Data Model

A subscriber can register its subscription with the Subscriber Manager specifying when, what, and how it should be notified. Such information consists of the subscription data, QoS properties, callback object reference, notification delivery style, and notification format. The last two are already discussed in Section 3.3.

4.1.1 Subscription Data

A subscription data structure consists of two parts:

1) Kind of event notification. The subscriber can choose to be informed of '*change service*' event or '*new service*' event. The '*change service*' event covers to any change in existing service objects (events 1, 2, 4 in Section 2) whereas

'new service' signifies the creation of new service objects (event 3 in Section 2).

2) Service identification. The subscriber can identify service objects of interest by specifying object reference strings and/or constraints on their properties. The subscriber may identify only one or both kinds of service identification.

Figure 2 shows an example of the subscription data represented in XML.

```
<?xml version = "1.0" ?>
<SubscriptionData>
   <EventType>ChangeEvent</EventType>
    <ServiceDescription>
      <ServiceTypeName>BankService</ServiceTypeName>
       <Property Name="BankName" Value="CUBank" />
        <Property Name="ReservedFund" Value="50000" />
        <ObjectReference>IOR: 001549...
        </ObjectReference>
    </ServiceDescription>
<SubscriptionData>
```

Figure 2. An Example of Subscription Data

4.1.2 Subscriber QoS Properties

We allow the subscriber to specify some QoS properties called subscriber QoS property. The QoS properties and their IDL data types are listed in Table 1.

Name	IDL Type
MaxQueueLength	Long
OrderPolicy	Const short
DiscardPolicy	Const short
MaximumBatchSize	Long
PacingInterval	TimeBase::UtcT

Table 1. Subscriber QoS Properties

The *MaxQueueLength* property determines the maximum number of events that are allowed to queue on the corresponding ProxySubscriber at any time. *OrderPolicy* sets the policy used by the ProxySubscriber to order the events it has buffered for delivery. Possible constant values of *OrderPolicy* are 'FifoOrder', 'PriorityOrder', and 'DeadlineOrder'. *DiscardPolicy* enables the subscriber to specify in what order the ProxySubscriber should begin discarding events in the case of an internal queue overflow according to the value on *MaxQueueLength* property. Possible constant values are 'FifoOrder', 'LifoOrder', 'PriorityOrder', and 'DeadlineOrder'.

The *MaximumBatchSize* and *PacingInterval* properties are applicable when subscribing to a sequence of events. *MaximumBatchSize* indicates the maximum number of events that will be delivered within each sequence. *PacingInterval* defines the maximum period of time for the ProxySubscriber to collect individual events into a sequence before delivery to the subscriber. If the number of events received within a given *PacingInterval* equals or exceeds *MaximumBatchSize*, the subscriber will receive a sequence of events whose length equals *MaximumBatchSize*. Otherwise, the subscriber will receive as many events as arriving at the ProxySubscriber during the *PacingInterval*.

4.1.3 Callback Object Reference

The callback object is an object created by the subscriber to stay active and listen to event notification on behalf of the subscriber. There are several kinds of callback objects depending on the types of the ProxySubscribers. For Example, if the subscriber subscribes to receive event notification in XML format using push delivery style, the callback object of this subscriber must implement the interface *XmlPushSubscriber* which provides an operation *push_xml_event()* to obtain event notification as an XML document. This operation will be invoked by the corresponding ProxySubscriber to inform the callback object of the event. The callback object will then take further action on the event as defined by the subscriber who creates it.

4.2 Notification Data Model

In this section, we explain respectively the structure of notification messages from service evolvers (or publishers) to the system and from the system to subscribers.

4.2.1 Publisher Notification Data Model

A service evolver may want to publish advance notification for future change. The published notification data can be partitioned into event header, event body, and additional data.

Event header consists of two parts, i.e. change type and notification properties. Possible string values of change type are *'Change_Service_Type'*, *'Change_property_val'*, *'Instantiation_Service'*, and *'Removal_Service'* (i.e. events 1-4 in Section 2). Notification properties allow the publisher to specify some attributes of the notification (Table 2). *Priority* is represented as a short value, where 1 is the lowest priority and 5 the highest. *ExpiryTime* allows the subscriber to indicate the time range in which an event is valid. If an event is not delivered within a specified time, e.g. when the callback object is down, it should be discarded. *StartTime* allows the publisher to specify specific time for the event to be delivered.

Name	IDL Type
Priority	Short
ExpiryTime	TimeBase::UtcT
StartTime	TimeBase::UtcT

Table 2. Notification Properties

Event body can be divided into four parts that are service description, change reason, change time, and change description. Service description represents the original description of the service object being changed (i.e. a service offer description in the trader). It consists of the service type name and some service properties. Change reason is the purpose of the change. Change time refers to some specific time in the future that this change will be in effect. Change description describes what changes will occur within the service object. Possible changes are listed in Figure 3.

- Service type change
  - New service type description
  - New service description
- Service property change
  - Delete property list
  - Modify property list
- New service
  - Service type description
- Removed service

Figure 3. Change Description

Additional data is an optional for service evolvers to put any other information such as company name, address, email, website, etc.

4.2.2 Event Notification Data Model

Event notification is the change information that a ProxySubscriber sends to the corresponding subscriber. The data structure of event notification is similar to that of publisher notification except the event header part. In this case, the event header consists of four attributes that are *event id*, *event type*, *event time indicator*, and *event timestamp*. *Event id* is a system-generated id. *Event type* is the same as *change type* specified in the event header of the publisher notification data. *Event time indicator* indicates whether this notification is an advance notification or not. *Event timestamp* displays date and time of arrival of this event to the system.

Figure 4 shows one example of an event notification as an XML document. It represents a notification of the removal of a service. From its content, line 3 to 10 is the header part which states that this is an advance notification. The body part is shown in line 11 to 23 telling the detail of the service instance to be removed and when it will be no longer available. The additional data part is in line 24 to 26.

```
1   <?xml version = "1.0" ?>
2   <ServiceChangeEvent>
3   <Header>
4     <EventID>e1821</EventID>
5     <EventType>Removal_Service</EventType>
6     <EventTimeIndicator>ADVANCE
7     </EventTimeIndicator>
8     <Timestamp>August 18, 2000, 09:00:00 GMT
9     </Timestamp>
10  </Header>
11  <Body type = "Removal_Service" >
12   <RemovalServiceContent>
13    <ChangedServiceDescription>
14      <ServiceTypeName> Bank1.0</ServiceTypeName>
15      <Property Name = "BankName"
16                Value = "CUBank"/>
17                ...
18      <ObjectReference>IOR: 04832...</ObjectReference>
19    </ChangedServiceDescription>
20    <Reason>Economical Crisis</Reason>
21    <CompleteTime>October 18, 2000</CompleteTime>
22   </RemovalSercice Content>
23  </Body>
24  <Additional>
25    Get more info at http://www.cubank.co.th ..
26  <Additional>
27  </ServiceChangeEvent>
```

Figure 4. Example of Event Notification Data

5. Conclusion and Future work

We have presented a design of the service change notification system based on CORBA distributed object environment. The prototype is being implemented on Visibroker 3.4 for Java [13] and we adopt *dcon*2.1 [14], an implementation of CORBA CosNotification service [10], as our event notification service.

Our notification system, with the help of the CORBA trader and event notification service objects, will enable users within a distributed environment to be informed of changes of several kinds that may happen to service objects they use. With such notification, we expect better change management from the client side in such a way that clients can take appropriate actions on service change in an effective way and in a timely manner. The notification in the XML format will also provide useful information for CORBA clients on the Internet. In the future, we may integrate this prototype with the service-provider-side notification system [6] to provide a complete change notification facility for both service providers, who offer service objects, and clients who operate on those objects whenever service changes occur.

6. Acknowledgement

This work is supported by the Development Grants for New Faculty/Researchers of Chulalongkorn University.

7. References

[1] ISO/IEC. ISO/IEC 10746-1 ODP Reference Model Part 1:Overview, 1995.

[2] Sessions, R. COM and DCOM, John Wiley&Sons, 1998.

[3] Object Management Group. The Common Object Request Broker: Architecture and Specification, Revision 2.2, February 1998.
[4] Object Management Group. ORB Interface Type Version Management Request for Proposals, OMG TC Document ORB/96-01-06, 1996.
[5] Senivongse, T. Enabling Flexible Cross-version Interoperability for Distributed Services, Proceedings of the International Symposium on Distributed Objects and Applications, Edinburgh, UK, 201 –210, 1999.
[6] Thongsisod, K. and Kongkriengkrai, T. A Notification System for Service Providers of Services with Evolved Interfaces in CORBA Distributed System, Senior Project Report, Dept. of Computer Engineering, Chulalongkorn University, Bangkok, Thailand, 1999.
[7] Object Management Group. Trading Object Service Specification, Revised Edition, March 1997.
[8] Peter, Y. An Implementation of CORBA's Life Cycle Service, Proceeding of 1st International Conference on Enterprise Distributed Object Computing Workshop, Australia, 111-117, 1997.
[9] Rifkin, A. and Khar, R. A Survey of Event Systems, http://www.cs.caltech.edu/~adam/isen/event-systems.html
[10] Object Management Group. Notification Service Joint Revised Submission, November 1998.
[11] Tomono, M. An Event Notification Framework based on Java and CORBA, Proceedings of the 6th IFIP/IEEE International Symposium on Distributed Management for the Networked Millennium, 563 –576, 1999.
[12] Bray, T., Paoli, J., and McQueen, S. Extensible Markup Language (XML) 1.0 Specification, World Wide Web Consortium Recommendation, http://www.w3.org/TR/REC-xml,10 February 1998.
[13] Inprise Visibroker. http://www.inprise.com
[14] DSTC, dcon 2.1, http://www.dstc.com

2. The 5th International Enterprise Distributed Object Conference (EDOC 2001) September 4-7 2001, Seattle, Washington USA ในบทความเรื่อง A CORBA-Based Architecture for Service Change Notification โดยผู้แต่งคือ Twittie Senivongse and Pasin Suriyentrakorn

A CORBA-Based Architecture for Service Change Notification

Twittie Senivongse[1] and Pasin Suriyentrakorn[2]
Department of Computer Engineering, Faculty of Engineering, Chulalongkorn University
Pathumwan, Bangkok 10330 Thailand
Phone: (662) 218-6996 Fax: (662) 218-6955
E-mail: twittie.s@chula.ac.th[1], g42psr@cp.eng.chula.ac.th[2]

Abstract

The nature of distribution and autonomy in an open environment makes it difficult to maintain a consistent view of service provision among distributed groups of clients. Change in services may have an impact on current operation of clients if they are not properly informed. Better management for change notification is necessary so as for the clients to be able to prepare for the change accordingly. This paper proposes a service change notification system that can manage service change events for subscribing clients. The architecture of this service is CORBA-based as it cooperates with a CORBA Trading Service to gather change events and has an Event Notification Service as the core component for change notification. Service providers can also register with this system to announce change events, and clients can subscribe for notification of change in services of their interest, specifying their preference for delivery style, notification format, and quality of service.

1. Introduction

In a software development process, we cannot avoid changes in software despite careful design and development. Changes may be caused by new user requirements, new technologies, system maintenance, or other external factors. As with other kind of software, service objects in distributed systems such as CORBA [1], RM-ODP [2], or DCOM [3] are also prone to evolution during their lifetime. In an enterprise system where software may be developed, as distributed services, by distributed groups of developers, or may cooperate with other pieces of software to achieve an enterprise function, change in one part of the application may affect the operation of other cooperating parts. When its behaviour no longer matches its clients' expectation, the clients are required to evolve accordingly. Management of change for distributed services can be complicated since there can be several instances of the same service and their clients are scattered throughout the system. Some clients can adjust themselves to change in time while some cannot, as it requires time and proper management to propagate change to them.

According to CORBA Trading Object Service model [4], a service object can be described by its type description that consists of an interface signature and a list of property definitions. Service objects of the same type must have the same type description but with their own property values. Change in the description of a service can interrupt the operation of its clients. For example, the clients may not be able to discover the service object they imported previously because its properties have now changed. Or even they can get to that service object, they may not be able to invoke the service as before because its interface signature has changed and so the clients can no longer communicate with it under the old protocol.

Messaging infrastructures are available for enterprise communication, e.g. JMS [5], MSMQ [6], CORBA Event Notification Service [7]. They allow exchange of messages between producer clients and consumer clients. This paper discusses the problems of service change within a CORBA environment and proposes an architecture for change notification based on CORBA Event Notification Service. This architecture enables various kinds of service change to be reported to clients of the system. It cooperates with the Trading Service or trader so that some changes can be automatically detected and immediately reported. In addition, service providers or evolvers can also publish some change information in advance. With the Event Notification Service as the model for our event notification component, a client can subscribe for notification of particular service change and can specify the preferred delivery style (i.e. push or pull style), delivery policy (e.g. delivery order, delivery interval), and format of notification message (i.e. structured object or XML [8]). Such notification will help clients to prepare or adjust themselves to change and hence can reduce unexpected effects caused by such change.

The rest of this paper is organised as follows. Section 2 discusses related work on supports for service change.

Section 3 explains the service type description model of a trader on which our supported change events are based, and those events are classified in Section 4. The overall architecture of the change notification service is presented in Section 5, while Section 6 details our current implementation. Section 7 concludes the paper and presents some future work.

2. Related work

Despite its importance, support for change in distributed objects has not been extensively researched. The previous request by the OMG for proposals for interface type version management [9] described the need for a management architecture that could support change in interfaces of objects in such a way that it had the least effect on CORBA users. This request was not responded since change management was then a secondary goal for object architectures and also there were issues that required more urgent attention (e.g. cross-vendor-platform interoperability).

Nevertheless, there have been researches in several aspects of change management. The mechanism in [10] allows clients of service objects with changed interfaces to continue their operation by transparently mapping client requests so that they fit new object interfaces, and therefore, interface evolution is made transparent to the clients. However, this mechanism can only prolong the use of old client programs by some period because eventually they will need to change as well. Other attempts [11], [12] provide dynamic development environment for CORBA client programs so that they can operate with dynamically generated stubs and skeletons or with dynamic invocation support, without having to use static stubs and skeletons. Addition of new object interfaces or change of existing interfaces in some compatible way will require no extra effort from the client side, and hence change is transparent. For the server side support, an initial attempt [13] is a notification system for service providers that will help notify responsible parties to properly evolve service objects whose interfaces have changed. This is to maintain consistency between interfaces, which are abstract definitions of services, and actual instances of the services.

Our work here focuses on the support for the client-side of the system with an assumption that some changes cannot be made transparent, and some should not, if they affect clients' expectation. Even though change can be hidden, it may be for some time only and at the end clients will need to change. It is also possible that clients may prefer being informed of change if the change means improvement of services for which they are willing to evolve.

3. Service description model

Clients are serviced through invocation on service objects and change in those service objects will affect clients'operation correspondingly. Since service objects are described by service descriptions, we will first present the model of service descriptions of a CORBA trader [4] on which our change notification service is based.

Each service object is an instance of a service type. A service type description consists of a service type name, an interface type name, definitions of properties, and base service type names (Figure 1). The interface type name refers to the interface signature by which clients of this service object abide in order to communicate with it. Property definitions describe non-computational characteristics of the service type while base service type names refer to other service types whose descriptions this service type inherits.

```
Service<ServiceTypeName>[:<BaseServiceTypeName> [,
                          <BaseServiceTypeName>]*] {
    interface <InterfaceTypeName>;
    {[mandatory][read-only][Normal] property <IDL Type>
    <PropertyName>;}*
};
```

Figure 1. Structure of service type in BNF

An example of a service type description is in Figure 2. A service type Bank consists of an interface type IDL:Bank:1.0 and two property definitions, namely Bank_Name and Interest_Rate. The interface type IDL:Bank:1.0 refers to an interface Bank that has three operations: balance, deposit and withdraw. All service objects that are offers of the type Bank are characterised by the same interface signature and property definitions, but with their own property values, e.g. one service object has Bank_Name = 'CU-Bank' and Interest_Rate = 3.0 whereas another has Bank_Name = 'ISEL-Bank' and Interest_Rate = 2.9.

When trading with a trader, a client can discover a service object by specifying the type name and properties as search criteria, e.g. the client may trade for a service object of type Bank that has the maximum Interest_Rate. After binding to that service object, the client will invoke operations on the object according to its interface signature.

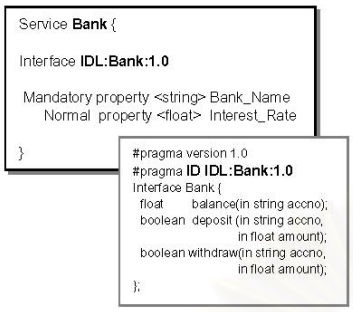


Figure 2. Example of service type description

4. Classification of service change events

In this section, we will describe all kinds of change events that our change notification service will consider based on the model in Section 3. We classify service change events into four categories: change of type of service object, change of property value of service object, removal of service object, and instantiation of service object.

4.1. Change of type of service object

This kind of change is change at the service type level of the service description model. From the context of a service type description, there are four kinds of service type change that can affect clients:

Case 1 Change of interface signature of object: Change of interface signature means the abstraction of service objects has changed and requires those service objects to evolve to conform to the new signature. However, existing clients may or may not affect by this. At present, service substitutability is possible by interface compatibility. This means a service may substitute for another if the former's interface inherits from the latter's and hence is recognised compatible by the underlying type system. By inheritance, the interface change is incremental (i.e. only new operations are added) and the new service with the changed interface will still be operable with existing clients. If change occurs in an incompatible way (e.g. renaming/reordering of operations, addition/removal of operation parameters, change of parameter types), existing clients will be affected and will no longer be able to communicate with those service objects, unless they adjust themselves to the changed interface as well. Figure 3 shows an example of incompatible change. In this case, the clients should be notified since they will need to modify the programs. Even though change is incremental and compatible with the old signature, it is also beneficial if the clients are informed of the added functionality.

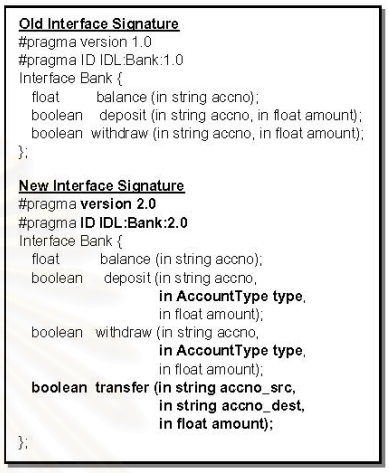


Figure 3. Change of interface signature

Case 2 Change of property definition of object: This case includes modification of property names, types, and modes, and addition and removal of property definitions. This kind of change may refer to change in non-computational behaviour or semantics of a service type which may or may not require actual service objects of the type to evolve. This change will not alter the way clients communicate with service objects, but it will affect the way they discover them, e.g. when trading with a trader. Figure 4 shows an example of this kind of change. If change occurs in a way that is not compatible with the old definition (e.g. change of property name), clients should be informed; otherwise they may not be able to discover the service object they use before if their search criteria still refer to the old property definition. Even though change is incremental (e.g. addition of new property), it is useful that clients are notified, since such change can be thought of as a refinement to the service description that the clients can use to refine their search criteria.

Before Change
Service Bank {
  Interface IDL:Bank:1.0
  Mandatory property <string> Bank_Name
    Normal property <float> Interest_Rate
}

After Change
Service **Bank_2** {
  Interface IDL:Bank:1.0
  Mandatory property <string> Bank_Name
    Normal property <float> **Deposit_Interest_Rate**
  Mandatory property <float> ReservedFund
}

Figure 4. Change of property definition

Case 3 Change of base type of service object: Having base types, a service type inherits interface signatures and property definitions from them. Change of base service types means change of inherited interface signatures and property definitions and this will impact clients in the same way as in Case 1 and 2 respectively.

Case 4 Change of type name of service object: Change of type name affects the way clients discover service objects, e.g. when trading with a trader. Clients may not be able to discover the service objects they use before if their search criteria still refer to the old type name. Change of type name may also reflect change in other parts of the type description. In these cases, clients need to be informed.

4.2. Change of property value of object

This kind of change is change at the service offer level of the service description model. Each service object is an offer of a service type with its own property values that conform to property definitions of its type. These property values indicate personal characteristics of the service object. Depending on how search criteria are defined, change of property values may or may not have an impact on the way clients discover service objects. That is, it is possible that the clients may still be able to get to the objects they use before, but the changed property values may affect their non-computational preference of service usage. Hence, it is safer to report on such change to them. Figure 5 shows an example of change of a property value. If a client used to trade for a Bank object that has Interest_Rate = 0.4, the change in the value of this property will make the client unable to get to that service object again. But if the client trades for a Bank object with the name 'CU-Bank', the client will still be able to get to the previously used service object after change, but its service may produce results different from before because of the new interst rate.

Before Change
  Bank_Name = 'CU-Bank'
  Interest_Rate = 0.4

After Change
  Bank_Name = 'CU-Bank'
  Interest Rate = 0.1

Figure 5. Change of property value

4.3. Removal of object

This kind of change refers to a permanent removal of a service object that stops the clients from using its service. Normally, service substitutability is provided in some manner, e.g. if search criteria are not so specific, clients that trade with a trader can make use of other objects instead if their characteristics also match the requirement. However, if clients specifically require the service object that has been removed, there will be no automatic substitutes and they will need to be informed.

4.4. Instantiation of object

This kind of change refers to instantiation of new service objects of the types that already exist or of the newly introduced types. Notification is useful for information purpose; clients will have more alternatives for the required service or have knowledge of new functionality that becomes available.

The four types of change events described above will be supported by the change notification service.

5. Architecture for Service Change Notification Service

Figure 6 depicts the overall architecture of the Service Change Notification Service (SCNS).

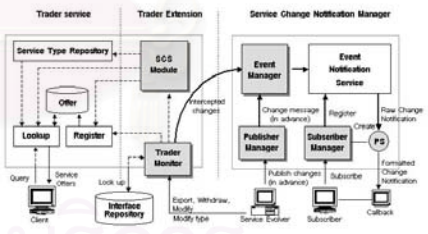


Figure 6. Service Change Notification Service

Change events can be notified either in advance or at the time of their occurrence. For advance notification, service evolvers will have to publish their change information with the SCNS so that the information will be announced further. Change information may also come from the trader that maintains descriptions of all service objects and provides operations for managing these descriptions. The Trader Monitor can detect the requests sent to the trader to alter those descriptions; these requests reflect changes to the advertised service objects. The detected change events will also be reported correspondingly.

The Service Change Notification Manager (SCN Manager) is the core component of the SCNS. It receives subscription information from subscribers who are those users interested in particular change events. It generates appropriate change notifications to the subscribers with the help from a CORBA Event Notification Service whenever it is informed of change events by service evolvers or the Trader Monitor. The PS or Proxy Subscribers, associated with particular subscribers, will obtain change notification messages and take care of their representation formats and delivery styles before delivering them to corresponding subscribers via their predefined callback objects. Each part of the SCNS will be explained in more details below.

5.1. Trader Extension

The Trader Extension interfaces with the trader to get its support for the change notification mechanism. It consists of two components: Trader Monitor and Service Change Supporting Module (SCS Module).

- **Trader Monitor** intercepts requests, invoked by service evolvers on the trader, to detect those that are requests to change service descriptions. The Trader Monitor is also in contact with the Interface Repository (IR) to obtain interface definitions of changed service objects. This information will be forwarded along with the intercepted messages, as change event information, to the SCN Manager for further announcement. From the trader specification, requests sent to modify, withdraw, and export service objects (c.f. change events in Section 4.2-4.4) will be detected by the Trader Monitor. However, the trader provides no direct operation to change the type of a service object (c.f. Section 4.1); the Trader Monitor, for example, cannot distinguish a request to permanently withdraw a service object from a request to withdraw an object before re-exporting it with a new description. To be able to detect this event, the Trader Monitor makes use of the SCS Module below.
- **SCS Module** extends the function of the trader with its IDL shown in Figure 7. The modifyServiceOffer_NewType (line 4-21) provides service evolvers with a direct operation to change the type of a service object. Invocation on this operation will enable the Trader Monitor to detect change of type (c.f. change event in Section 4.1) and this detected event will then be forwarded to the SCN Manager. The attributes publisherManager_if and subscriberManager_if (line 2-3) refer to object references of the Publisher Manager and Subscriber Manager components (see Section 5.2) respectively which will be used by service evolvers and subscribers to get connected to the SCNS.

5.2. SCN Manager

The Service Change Notification Manager or SCN Manager is the core component responsible for managing subscription data and filtering and dispatching service change notifications to subscribers. The SCN Manager is composed of five subcomponents: Publisher Manager, Subscriber Manager, Event Manager, CORBA Event Notification Service, and Proxy Subscriber.

- **Publisher Manager** receives advance change information from service evolvers before sending it to the Event Manager for further notification.
- **Subscriber Manager** receives registration information from subscribers. It is also a factory object that creates and destroys Proxy Subscribers on behalf of registered subscribers.
- **Event Manager** receives all change information from the Trader Monitor and Publisher Manager and transforms it to valid structured event objects before sending them to the CORBA Event Notification Service.
- **CORBA Event Notification Service** or COS Notification is a Common Object Service (COS) standardised by the OMG as an event middleware [7]. We adopt it for filtering, queuing, and dispatching incoming change events to subscribers according to their subscription.
- **Proxy Subscriber** is responsible for notification subscription, notification delivery, and notification formats of a subscriber for which it is created. The subscriber can submit a notification requirement that will be used by the Event Notification Service, specifying the service and changes of interest. The Proxy Subscriber is created by the Subscriber Manager according to the type specified by the

subscriber (Table 1). The subscriber may choose to be notified as soon as change events occur (push style), or notification is to be by demand such that the subscriber itself will request from the Proxy Subscriber (pull style). A notification message may consist of a single change event or a sequence of change events, which may take the form of structured object or XML document. Note that the Subscriber Manager will create at most one Proxy Subscriber of each type per one subscriber. The Proxy Subscriber will deliver notification messages to a callback object, which is the object nominated by the subscriber at subscription time to receive notification.

```
1.  Interface SCS {
2.    readonly attribute Object publisherManager_if;
3.    readonly attribute Object subscriberManager_if;
4.    CosTrading::OfferId modifyServiceOffer_NewType(
5.        in CosTrading::OfferId oldOfferId,
6.        in CosTrading::Register::OfferInfo newOffer,
7.        in boolean withdrawOldOffer)
8.    raises (
9.        CosTrading::Register::InvalidObjectRef,
10.       CosTrading::Register::InterfaceTypeMismatch,
11.       CosTrading::Register::ProxyOfferId,
12.       CosTrading::IllegalServiceType,
13.       CosTrading::UnknownServiceType,
14.       CosTrading::IllegalPropertyName,
15.       CosTrading::PropertyTypeMismatch,
16.       CosTrading::ReadonlyDynamicProperty,
17.       CosTrading::MissingMandatoryProperty,
18.       CosTrading::DuplicatePropertyName,
19.       CosTrading::IllegalOfferId,
20.       CosTrading::UnknownOfferId );
21. };
```

Figure 7. IDL of SCS Module

Table 1. Types of proxy subscribers

Proxy Subscriber	Delivery	Representation
StructuredPushProxy	Push	Single structured event object
XMLPushProxy	Push	XML document
SequenceStructured PushProxy	Push	Sequence of structured event object
StructuredPullProxy	Pull	Single Structured event object
XMLPullProxy	Pull	XML document
SequenceStructured PullProxy	Pull	Sequence of structured event object

5.3. Administrative and QoS properties

In this section, we describe property parameters that can be specified for the required behaviour of the SCNS. Most properties are adopted from the properties specified in the COS Notification. They are:

- **Administrative Properties** are properties that represent the capacity of the SCNS and specified by the administrator of the SCNS.
  - MaxSubscribers is the maximum number of subscribers that can register to the SCNS at any given time.
  - MaxConnectedSubscribers is the maximum number of subscribers that can connect to the SCNS at any given time.
  - MaxEventPerProxy is the maximum number of notification messages that can be queued for each Proxy Subscriber before they are delivered. When the limit is reached, the SCNS will start discarding some messages according to the discard policy below. This property represents a quota of notification messages that the SCNS will maintain for any Proxy Subscriber of a single subscriber.
- **Subscriber QoS Properties** are specified by subscribers to determine delivery policy for notification messages.
  - OrderPolicy specifies the order of delivery when there are more than one message in the queue of a Proxy Subscriber. The order may be 'FifoOrder', 'PriorityOrder', or 'ExpiryTimeOrder'.
  - DiscardPolicy specifies the order by which messages should be discarded when an entry of a notification message to the queue of a Proxy Subscriber reaches the limit specified by MaxEventPerProxy or MaxQueueLength. The discarding order may be 'FifoOrder', 'LifoOrder', 'PriorityOrder', and 'ExpiryTimeOrder'.
  - MaximumBatchSize and PacingInterval apply when notification messages are delivered in a group (c.f. SequenceStructuredPushProxy and SequenceStructuredPullProxy types in Table 1). They determine, respectively, the maximum number of messages that will be delivered in one batch and the frequency of batch delivery. That is, a batch of notification messages will be delivered when either the maximum batch size or the specified time interval is reached.
- **Event QoS Properties** are properties that describe notification messages.
  - Priority is a value describing how important a notification of a change event is. The Trader Monitor will define a default priority value for any change events that it can detect while service evolvers can determine priority for change events they publish as they see

fit. A priority value ranges from 1 (lowest) to 5 (highest). Note that these predefined priority values can be overridden by subscribers.

- ExpiryTime is a TimeBase::UtcT value [14] specifying absolute time that determines the lifetime of a notification message before it can be discarded. The Trader Monitor will define default expiry time for any change events that it can detect while service evolvers can specify expiry time for a particular change event that they publish as they see fit.

5.4. Subscription, published change, and notification information

With the aforementioned characteristics of change events and notification requirements, we will now focus on the detail of subscription information specified by subscribers, published change information submitted by service evolvers, and notification information delivered by the SCNS to subscribers.

- **Subscription information** specifies change events of a particular service object in which a subscriber is interested. The information consists of the kind of service change event and service identification.
  - Kind of service change event can be either 'Change of Service', 'Instantiation of service' or 'Change of Service and Instantiation of service'. The first one covers change events in Section 4.1-4.3 that affect existing service objects. The second one refers to the change event in Section 4.4 that involves new service objects. The last one covers all change events in Section 4.1-4.4.
  - Service identification indicates service object(s) whose changes are of the subscriber's interest. The subscriber can identify service object(s) by using the trader's constraint language [4] in a similar way to importing service object(s), or by using object references to specifically identify service objects.

Figure 8 shows subscription information of a subscriber who requests to be informed of any kinds of change of a service object named 'CU-Bank' with reserved fund valued 100000000.

Kind Of Event = Change of Service
Service Identification type= TraderConstraint
ServiceTypeName =Bank
Property Name="Bank_Name" **Value**="CU-Bank"
Property Name="ReservedFund" **Value**=100000000

Figure 8. Example of subscription information

```
1.  <!ELEMENT ServiceChangeNotification (
2.                  NotificationMessageHeader,
3.                  NotificationMessageBody,
4.                  AdditionalData)>
5.  <!--Message Header section -->
6.  <!ELEMENT NotificationMessageHeader (
7.                  MessageID,
8.                  MessageType,
9.                  TimeIndicator,
10.                 Priority,
11.                 ExpiryTime)>
12. <!ELEMENT MessageID     (#PCDATA)>
13. <!ELEMENT MessageType       EMPTY>
14. <!ATTLIST   MessageType value (
15.                 Change_Of_Service_Type |
16.                 Change_Of_Property_value |
17.                 Removal_Of_Service |
18.                 Instantiation_Of_Service)
19.                 " Change_Of_Service_Type">
20. <!ELEMENT TimeIndicator     EMPTY>
21. <!ATTLIST   TimeIndicator value (
22.                 In_Advance_Notification |
23.                 Immediate_Notification)
24.                 "In_Advance_Notification">
25. <!ELEMENT Priority          (#PCDATA)>
26. <!ELEMENT ExpiryTime    (#PCDATA)>
27. <!--Message Body section -->
28. <!ELEMENT NotificationMessageBody (
29.                 ChangeOfServiceTypeOfService |
30.                 ChangeOfPropertyValueOfService |
31.                 RemovalOfService |
32.                 InstantiationOfService)>
33. <!ELEMENT ChangeOfServiceTypeOfService (
34.                 ServiceDescription+,
35.                 Reason,
36.                 Time)>
37. <!ELEMENT Reason            (#PCDATA)>
38. <!ELEMENT Time              (#PCDATA)>
39. <!ELEMENT ChangeOfPropertyValueOfService (
40.                 ServiceDescription,
41.                 DeleteProperties,
42.                 ModifyProperties,
43.                 Reason,
44.                 Time)>
45. <!ELEMENT DeleteProperties (Property*)>
46. <!ELEMENT ModifyProperties (Property*)>
47. <!ELEMENT RemovalOfService (
48.                 ServiceDescription,
49.                 Reason,
50.                 Time)>
51. <!ELEMENT InstantiationOfService (
52.                 ServiceDescription,
53.                 Objective,
54.                 Time)>
55. <!ELEMENT Objective         (#PCDATA)>
56. <!-- Additional Data section -->
57. <!ELEMENT AdditionalData  (#PCDATA)>
```

Figure 9. DTD of notification information

- **Published change information** is the change events published by a service evolver with the Publisher Manager. Each event consists of priority, expiry time, type of change, service description before and after change, reasons for change, and effective time of change. This information is quite similar to the notification information detailed below.

- **Notification information** is the notification message that a Proxy Subscriber delivers to a callback object of a corresponding subscriber. Figure 9 shows the Document Type Description (DTD) of the notification message if described in XML

```
58  <!ELEMENT  ServiceDescription (
59             ServiceTypeDescription,
60             Property*,
61             ObjectReference)>
62  <!ATTLIST  ServiceDescription type (
63             ORIGINAL_SERVICE |
64             CHANGED_SERVICE)
65             "ORIGINAL_SERVICE">
66  <!ELEMENT  Property EMPTY>
67  <!ATTLIST  Property
68             Name CDATA #REQUIRED
69             Value CDATA #IMPLIED>
70  <!ELEMENT  ObjectReference (#PCDATA)>
```

Figure 10. DTD of ServiceDescription

5.5. Example scenario

To get a better picture, this section describes the workflow within the SCNS (Figure 11).

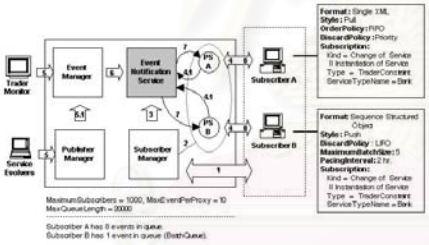


Figure 11. Example scenario

The SCNS starts up with predefined administrative property values, e.g. MaximumSubscriber = 1000, MaxEventPerProxy = 10, and MaxQueueLength = 2000. Two subscribers, A and B, first register themselves with the SCNS via the Subscriber Manager (1) and choose to create a Proxy Subscriber of the required delivery style and message format (2). Subscriber A creates a Proxy Subscriber called 'PS A' of type XMLPullProxy and B instantiates 'PS B' of type SequenceStructuredPushProxy. The Proxy Subscribers instantiated will be automatically registered with the Event Notification Service (3). After that, through 'PS A', subscriber A specifies subscription information, indicating its requirement to receive all change events that involve the service type 'Bank', as well as its QoS properties, i.e. OrderPolicy = 'FIFO' and DiscardPolicy = 'Priority' (4). Subscriber B specifies through 'PS B' its subscription information, similar to that of A, and its QoS properties, i.e. DiscardPolicy = 'LIFO', MaximumBatchSize = 5 events and PacingInterval = 2 hours (4). This input to 'PS A' and 'PS B' will be passed on to use with the Event Notification Service in filtering and notifying events (4.1).

The Trader Monitor intercepts two events of type 'Bank' to remove a service object and to instantiate a new one, and defines priority level and expiry time for these events before sending them to the Event Manager (5). A service evolver registers with the Publisher Manager to publish a future event that involves a change in the service type 'Bank' (5), specify priority level and expiry time for the event, and forward it to the Event Manager (5.1). The Event Manager will reformat all incoming change information to valid structured event objects and send them to the Event Notification Service (6). Filtering the event objects according to subscription information, the Event Notification Service will notice that these three change events are to be sent to both subscribers A and B. It will queue the events for 'PS A' to pull while pushing them to 'PS B' (7). If these Proxy Subscribers are in offline or suspended mode, it will store the event objects until they are reconnected. 'PS A' and 'PS B' will transform the change messages they receive to the format corresponding to their types and send notification messages to callback objects of A and B (8).

Assume that there are eight notification messages in the queue for 'PS A' with the maximum capacity of ten messages (c.f. MaxEventPerProxy) and there is one message stored in the batch queue for 'PS B'. With event objects M1, M2, and M3 whose headers are shown in Figure 12, only two more events can be queued for 'PS A' and thus one with the least priority will be discarded (c.f. DiscardPolicy). So if one of the events in the queue has priority 1, it will be discarded to give enough room for the three incoming events that have more priority. These event messages will wait for 'PS A' to pull from the queue. For subscriber B, the three events will be appended to the queue for 'PS B', awaiting one more event to arrive and fill the queue (c.f. MaximumBatchSize) or until the 2-hour's time is up (c.f. PacingInterval), before the batch is pushed to 'PS B'.

Figure 13 shows a notification message for M2 that 'PS A' creates in XML. The document describes the instantiation of a 'Bank' service object called 'CU-Bank'

detected on Wed 28 Feb 2001, with details of its service type and interface definition. This message is valid until Wed 7 Mar 2001 and can be discarded after that.

Figure 12. Header part of M1, M2, and M3

```
<?xml version="1.0" ?>
<!DOCTYPE ServiceChangeNotification (View Source for full doctype...)>
- <ServiceChangeNotification>
  - <NotificationMessageHeader>
      <MessageID>EV101</MessageID>
      <MessageType value="Instantiation_Of_Service" />
      <TimeIndicator value="Immediate_Notification" />
      <Priority>2</Priority>
      <ExpiryTime>Wed Mar 07 08:00:00 GMT+07:00 2001</ExpiryTime>
    </NotificationMessageHeader>
  - <NotificationMessageBody>
    - <InstantiationOfService>
      - <ServiceDescription type="CHANGED_SERVICE">
        - <ServiceTypeDescription>
          - <TraderServiceType Id="IDL:Bank:1.0" Name="Bank" Masked="NO">
              <Property Name="Bank_Name" Type="string" Mode="NORMAL" />
              <Property Name="Deposit_Interest_Rate" Type="float" Mode="NORMAL" />
              <Property Name="ReservedFund" Type="float" Mode="NORMAL" />
            </TraderServiceType>
          - <Interface Id="IDL:Bank:1.0" Name="Bank" Version="1.0">
              <Operation Id="IDL:Bank/balance:1.0" Name="balance" Version="1.0"
                Type="float" Mode="NORMAL" />
              :
            </Interface>
          </ServiceTypeDescription>
          <Property Name="Bank_Name" Value="CU-Bank" />
          <Property Name="Deposit_Interest_Rate" Value="0.4" />
          <Property Name="ReservedFund" Value="100000000" />
          <ObjectReference>IOR:000000000000001549444c3a62...</ObjectReference>
        </ServiceDescription>
        <Objective>...</Objective>
        <Time>Wed Feb 28 08:00:04 GMT+07:00 2001</Time>
      </InstantiationOfService>
    </NotificationMessageBody>
    <Additional>...</Additional>
  </ServiceChangeNotification>
```

Figure 13. Example of instantiation of service message in XML

6. Prototype implementation

In this section, we describe our implementation of the SCNS followed by a client interface prototype.

6.1. Implementation of the SCNS

We have developed a prototype of the SCNS on VisiBroker 3.4 [16] using Java. We adopt dCon2.2 [17], an implementation of COS Notification, as our Event Notification Service. The trader used in our system is JacORB Trader [18].

- **Trader Monitor** is implemented as a VisiBroker Server Interceptor [16]. Its class diagram is shown in Figure 14. The main class is TraderMonitorInterceptor that implements Visibroker's ServerIntereceptor interface. It is associated with the IR (CORBA:Repository) which supplies interface definitions of service objects, and the EventManager (EventManagerImpl) which is a component of the SCN Manager that processes intercepted change events.
- **SCS Module** is integrated with JacORB trader as shown in Figure 15. The main class is SCSModuleImpl that implements the SCS interface in Figure 7. It is associated with the class RegisterImpl of JacORB trader to make use of the operations for export and withdrawal of a service object in order to implement its own modifyServiceOffer_NewType() operation for change of type of service object. The publisherManager_if and SubscriberManager_if attributes refer to the object references of the instances of PublisherManagerImpl class and SubscriberManagerImpl class respectively.

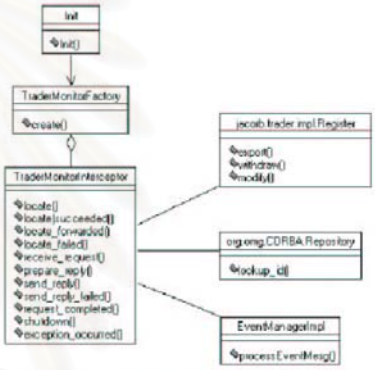


Figure 14. Class diagram of Trader Monitor

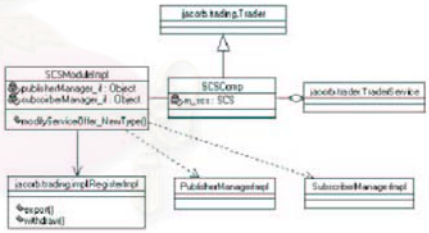


Figure 15. Class diagram of SCS Module

- **SCN Manager** is a composition of several components implemented as separate CORBA

services, i.e. Publisher Manager, Subscriber Manager, Event Manager, Event Notification Service, and Proxy Subscriber. We adopt dCon2.2 whose interface complies with the specification of COS Notification as our Event Notification Service so we will describe only part of the interfaces of other components here.

- **Publisher Manager Interface:**

```
interface PublisherManager : SCNComponent {
  void notify(in PublishedStructuredEventMessage
        eventMesg);
};
```

The operation notify() is invoked by service evolvers to submit published change events (c.f. Section 5.4).

- **Subscriber Manager Interface:**

```
interface SubscriberManager : SCNComponent {
   AdminSubscriber registerSubscriber (
        in string subscriberID,
        in string  passwd)
     raises ( ExceedSubscriberLimit,
              DuplicatedSubscriberID);
   void unregisterSubscriber (
        in string subscriberID,
        in string Password passwd)
     raises (NotFoundSubscriberID, InvalidPassword);
   AdminSubscriber getAdminSubscriber (
        in string subscriberID,
        in string passwd)
     raises (NotFoundSubscriberID, InvalidPassword);
};
interface AdminSubscriber {
   ProxySubscriber createProxySubscriber(
      in ProxyType proxyType,  out ProxyID proxyID)
     raises (InvalidProxyType, ExceedProxyLimit,
             AlreadyHaveTheType);
   void destroyProxySubscriber(in ProxyID proxyID)
     raises (NotFoundProxy);
   scncomm::ProxySubscriber getProxySubscriber(
      in ProxyID proxyID)
     raises (NotFoundProxy);
   ...
};
```

The operation registerSubscriber() is invoked by a subscriber when first registering with the SCNS while unregisterSubscriber() is called to unregister the subscriber. The AdminSubscriber object is a result of registerSubscriber() and getAdminSubscriber() and refers to the administrator object that is responsible for the creation of all Proxy Subscribers for that particular subscriber. The AdminSubscriber interface consists of three main operations: createProxySubscriber() is called by the subscriber to create a Proxy Subscriber object of a particular type, destroyProxySubscriber() is invoked by the subscriber to destroy a proxy object, and getProxySubscriber is used by the subscriber to obtain the reference to a proxy object.

- **Event Manager Interface:**

```
interface EventManager : SCNComponent {
  void processEventMesg (in StructuredEventMessage
        eventMesg);
};
```

The operation processEventMesg() is called by the Trader Monitor and Publisher Manager to pass on change events. These events will be transformed into a structured event format understood by the Event Notification Service.

- **Proxy Subscriber Interface:**

```
interface ProxySubscriber : FilterAdmin, QoSAdmin {
  readonly attribute ProxyID proxyID;
  readonly attribute ProxyType proxyType;
  void disconnect() raises(AlreadyDisconnected);
  void destroy();
};
interface FilterAdmin {
   void addFilter( in FilterContent filter,
                   out FilterID filterID)
     raises(IllegalConstraint);
   void removeFilter(in FilterID filterID)
     raises(FilterNotFound);
   FilterInfo getFilter(in FilterID filterID)
     raises(FilterNotFound);
   ...
};
interface QoSAdmin {
   void set_qos ( in QoSProperties qos)
     raises ( UnsupportedQoS);
   QoSProperties get_qos();
   ...
};
interface XMLProxyPullSubscriber : ProxySubscriber
{
   void connect (in XMLPullSubscriber  subscriber)
     raises(AlreadyConnected, TypeError);
   XMLDocument pull_xml_event(
      out boolean has_event)
     raises(Disconnected);
};
```

The ProxySubscriber interface is the parent interface of all types of Proxy Subscribers in Table 1 above. The disconnect() operation is used by a subscriber to disconnect the Proxy Subscriber temporarily from the SCNS, and no notification will be delivered during the disconnection. (Nevertheless, the Event Notification Service will still maintain notification messages until the connection is re-established.) The destroy()

operation is called when there is a need to destroy the Proxy Subscriber. The ProxySubscriber interface also inherits from FilterAdmin and QoSAdmin interfaces.

The FilterAdmin interface provides the operation addFilter() for the subscriber to specify subscription information (c.f. Section 5.4). The operations removeFilter() and getFilter() are for removing and retrieving subscription information respectively. The QoSAdmin interface allows the subscriber to manage subscriber QoS properties (c.f. Section 5.3). The operation set_qos() is used to set subscriber QoS properties and get_qos() is for obtaining subscriber QoS Property values.

The XMLProxyPullSubscriber interface is an example of a Proxy Subscriber with pull single-event delivery style and XML notification format. Its operation connect() is for a subscriber to connect the Proxy Subscriber to the SCNS and to specify a callback object for this Proxy Subscriber. The operation pull_xml_event() will be called by the subscriber to request for notification messages.

6.2. Client interface prototype

We have developed a GUI based client program called the Subscriber Agent for subscribers to conveniently specify subscription information and receive notification messages. The Subscriber Agent is implemented as a subscriber of a Proxy Subscriber of the type XMLPushProxy. When a notification arrives at the associated Proxy Subscriber, it transforms the XML notification message into HTML by using the API of XSLT [19] and writes the message to a file for later retrieval for display. Sample screens showing notification messages are in Figure 14.

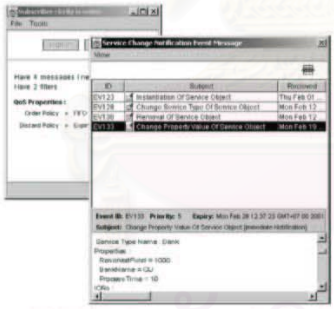


Figure 14. Notification screens of subscriber agent

7. Concluding remarks

We propose a CORBA-based change notification architecture called the SCNS that enables users within a CORBA system to be informed of several kinds of change in services they use. The SCNS allows change to be detected automatically or be published by service evolvers. Clients can specify subscription details for notification of change of their interest. When being informed, clients can take appropriate action and hence unexpected effects that changes may put on the client side operation can be reduced. Since change management is most of the time a secondary goal for application systems, we see that the SCNS is an attempt to stress this issue. It is a useful alternative for an enterprise that would like to alleviate problems caused by service change as it can assist in tracking service evolution in the process of application development and the maintenance of ongoing operations.

As the number of services and the number of subscribers/publishers reflect the workload of the SCNS, we hope that it will be practical to use because, for a large number of services, we expect each service to change only occasionally. At present, we are testing the performance of the SCNS to determine its subscribing capacity and average time to deliver with varying delivery frequency. We are also developing a GUI based client program to provide a more convenient way for service evolvers to work with the SCNS. In the future, we plan to incorporate change notification into trading federation in such a way that change occurring at a trader can be notified at remote clients of the federation. This notification service can also be integrated with the service provider side notification system [13] to provide a more complete change notification facility for both service providers and clients.

The ability of this change notification service to detect and report change is still limited as it requires human evolvers to input change and reasons for change and requires clients to determine by themselves where changes are and whether any response to the changes needs to be made. It is seen that if behavioural comparison of an old service object and its changed counterpart is possible, both syntactic and semantic differences can be identified more automatically. Specifically, some change that is not reflected on service descriptions, e.g. change in implementation behaviour of service objects, can also be detected. With more accurate information, clients will be better helped to decide what action needs to be taken.

Acknowledgement

This work is supported by the Thailand-Japan Technology Transfer Project (TJTTP-OECF) and the Development Grants for New Faculty/Researchers of Chulalongkorn University.

References

[1] Object Management Group, "The Common Object Request Broker: Architecture and Specification, Revision 2.2", February 1998.

[2] ISO/IEC, "ISO/IEC 10746-1 ODP Reference Model Part 1:Overview", 1995.

[3] R. Sessions, *COM and DCOM*, John Wiley&Sons, 1998.

[4] Object Management Group, "Trading Object Service Specification, Revised Edition", March 1997.

[5] Sun Microsystems, "Java Message Service Version 1.0.2", 1999, http://www.javasoft.com/products/jms/docs.html.

[6] Microsoft, "MSMQ 3.0", 2000, http://microsoft.com/msmq/default.htm.

[7] Object Management Group, "Notification Service Joint Revised Submission", November 1998.

[8] T. Bray, J. Paoli, and S. McQueen, "Extensible Markup Language (XML) 1.0 Specification", World Wide Web Consortium Recommendation, 10 Feb. 1998.

[9] Object Management Group, "ORB Interface Type Version Management Request for Proposals", OMG TC Document ORB/96-01-06, 1996.

[10] T. Senivongse, "Enabling Flexible Cross-version Interoperability for Distributed Services", *Proceedings of the International Symposium on Distributed Objects and Applications*, Edinburgh, UK, 1999, pp. 201 –210.

[11] K. Fujieda, T.Watanabe, and K. Ochimizu, "CORBA Application Development Environment Using Reflection", *Proceedings of the International Symposium on Future Software Technology (ISFST'99)*, Nanjing, China, 1999.

[12] R. Ierusalimschy, R. Cerqueira, and N. Rodriguez, "Using Reflexivity to Interface with CORBA", *Proceedings of the International Conference on Computer Languages*, 1998.

[13] K. Thongsisod and T. Kongkriengkrai, "A Notification System for Service Providers of Services with Evolved Interfaces in CORBA Distributed System", Senior Project Report, Dept. of Computer Engineering, Chulalongkorn University, Bangkok, Thailand, 1999.

[14] Object Management Group, "CORBAservices: Common Object Services Specification", 1997.

[15] T. Senivongse and W. Nanekrangsan, "An approach to Standardizing Distributed Service Descriptions Format using XML", *Proceedings of the 3rd IFIP WG 6.1 International Working Conference on Distributed Applications and Interoperable Systems (DAIS'2001)*, Krakow, Poland, 2001.

[16] Inprise, "VisiBroker for Java Programmer's Guide Version 3.3", 1998.

[17] DSTC dCon 2.2, http://www.dstc.com.

[18] JacORB – a free Java ORB, http://jacorb.inf.fu-berlin.de.

[19] C. James, "XSL Transformations (XSLT)", http://www.w3c.org/TR/xslt, 1999.

ประวัติผู้เขียนวิทยานิพนธ์

นายภาสิน สุริเยนทรากร เกิดเมื่อวันที่ 18 สิงหาคม 2521 ที่จังหวัดสงขลา สำเร็จการศึกษาหลักสูตรวิศวกรรมศาสตรบัณฑิต (วศ.บ) สาขาวิศวกรรมคอมพิวเตอร์ จากภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ มหาวิทยาลัยสงขลานครินทร์ เมื่อปีการศึกษา 2542 และเข้าศึกษาต่อหลักสูตรวิศวกรรมศาสตรมหาบัณฑิต (วศ.ม) สาขาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ในปีการศึกษา 2542