สำนักหอสมุดกลาง พระจอมเกล้าลาดกระบัง

การโอนย้ายสถานะโดยใช้คลาสโหลดเดอร์ในระบบรักษาความต่อเนื่องในการทำงานที่มีการเปลี่ยนแปลงออบเจ็กต์ของจาวา

STATE TRANSFER USING MULTIPLE CLASS LOADERS IN CONTINUOUS MIGRATION CONTAINER FOR JAVA OBJECT

T060255

เนตรสุดา อยู่สันเทียะ

NETSUDA YOOSANTHIAH

วพ.
ว785ก
2548

เลขหมู่.....
เลขทะเบียน 60255
วัน,เดือน,ปี 27 มิ.ย. 2549

.b 11589013
.i.....

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต

สาขาวิชาเทคโนโลยีสารสนเทศ

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ. 2548

ISBN 974 – 15 – 1764 - 5

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

STATE TRANSFER USING MULTIPLE CLASS LOADERS IN CONTINUOUS MIGRATION CONTAINER FOR JAVA OBJECT

NETSUDA YOOSANTHIAH

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF

MASTER OF SCIENCE IN INFORMATION TECHNOLOGY

SCHOOL OF GRADUATE STUDIES

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2005

ISBN 974 – 15 – 1764 - 5

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

COPYRIGHT 2005

SCHOOL OF GRADUATE STUDIES

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

หัวข้อวิทยานิพนธ์	การโอนย้ายสถานะโดยใช้คลาสโหลดเดอร์ในระบบรักษาความต่อเนื่องในการทำงานที่มีการเปลี่ยนแปลงออบเจ็กต์ของจาวา
นักศึกษา	นางสาว เนตรสุดา อยู่สันเทียะ
รหัสประจำตัว	43067002
ปริญญา	วิทยาศาสตร์มหาบัณฑิต
สาขาวิชา	เทคโนโลยีสารสนเทศ
พ.ศ.	2548
อาจารย์ผู้ควบคุมวิทยานิพนธ์	ผศ. อัครินทร์ คุณกิตติ

บทคัดย่อ

การปรับปรุงเปลี่ยนแปลงออบเจ็กต์ของแอพพลิเคชั่นในขณะที่กำลังมีการทำงานนั้น หากไม่มีการจัดการและควบคุมการทำงานของออบเจ็กต์ ที่เหมาะสม อาจส่งผลให้การทำงานของแอพพลิเคชั่นขาดความต่อเนื่องได้ ดังนั้น งานวิจัยนี้จึงได้เสนอแนวคิดที่จะแก้ไขปัญหาดังกล่าว โดยการสร้างสภาพแวดล้อมในรูปของระบบคอนเทนเนอร์แบบรักษาความต่อเนื่องให้กับการทำงาน (Continuous Migration Container: CMC) ของแอพพลิเคชั่นที่เขียนด้วยภาษาจาวา โดยใช้กลไกของ Multiple Class Loaders สำหรับโหลดคลาสที่มีชื่อเหมือนกัน เข้าสู่สภาพแวดล้อมเดียวกันได้ในขณะ Runtime แบบไดนามิก ซึ่งมีการตรวจสอบความเข้ากันได้ของรายละเอียดพื้นฐานระหว่างออบเจ็กต์ใหม่และออบเจ็กต์เก่า ก่อนเข้าสู่กระบวนการเปลี่ยนแปลงออบเจ็กต์ที่อาศัยความร่วมมือจากแอพพลิเคชั่นในการอิมพลีเมนต์อินเตอร์เฟส และ Hash Map สำหรับเก็บข้อมูลสถานะของแต่ละออบเจ็กต์ไว้ใช้ในการโอนย้ายสถานะการทำงานต่างๆ โดยเมื่อทำการทดลองตามวิธีการดังกล่าวแล้ว พบว่า ออบเจ็กต์เก่าจะยังคงทำงานในช่วงเวลาเดียวกับที่ออบเจ็กต์ใหม่เริ่มต้นทำงานทันทีเมื่อได้รับสถานะการทำงานของออบเจ็กต์ต่างๆ ในแอพพลิเคชั่นนั้น นั่นคือ ออบเจ็กต์ใหม่และออบเจ็กต์เก่าจะมีการทำงานควบคู่กันไปช่วงระยะเวลาสั้นๆ ช่วงหนึ่งก่อนที่ออบเจ็กต์เก่าจะหยุดการทำงานลง จึงนับว่าในขณะมีการเปลี่ยนแปลงออบเจ็กต์นั้นแอพพลิเคชั่นสามารถทำงานได้อย่างต่อเนื่อง และสามารถทำงานได้อย่างถูกต้องระดับหนึ่งภายหลังการเปลี่ยนแปลง ซึ่งเป็นสิ่งสำคัญในการรักษาเสถียรภาพและเพิ่มประสิทธิภาพในการทำงานมากขึ้น

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Thesis Title	State Transfer using Multiple Class Loaders in Continuous Migration Container for Java Object
Student	Miss Netsuda Yoosanthiah
Student ID.	43067002
Degree	Master of Science
Programme	Information Technology
Year	2005
Thesis Advisor	Asst.Prof. Akharin Khunkitti

ABSTRACT

Java application may discontinuously operate while they are upgrading, if there are not appropriate management and control objects mechanism. This paper proposes a methodology to support continuous system operation and its stability. The adoption is based on dynamic class loading by multiple class loaders mechanism accommodate to construct Continuous Migration Container (CMC). CMC is a runtime environment provides object state transfer through hash map and interfaces implementation with application's cooperation. It also supports object behavior equivalence checking before object migration process. This research can manage and control objects still perform together with newly object and existing object by continuous system operation and right after upgrading object. These operations are crucial for system stability and enhancement efficiency.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

กิตติกรรมประกาศ

วิทยานิพนธ์เล่มนี้สำเร็จได้ด้วยความกรุณาจากอาจารย์ ผศ.อัครินทร์ คุณกิตติ ที่ให้ความช่วยเหลือ ให้คำชี้แนะ ช่วยแก้ปัญหาตลอดจนให้ความรู้และประสบการณ์ที่ดีแก่ข้าพเจ้า

ขอขอบคุณเพื่อน ๆ และพี่ ๆ ในห้องปฏิบัติการทุกคนที่ช่วยให้คำปรึกษา ความช่วยเหลือและให้กำลังใจซึ่งกันและกันตลอดมา

สำหรับคุณงามความดีอันใดที่เกิดจากวิทยานิพนธ์ฉบับนี้ ข้าพเจ้าขอมอบให้กับบิดามารดา ซึ่งเป็นที่รักและเคารพยิ่ง ตลอดจน ครู อาจารย์ที่เคารพทุกท่านที่ได้ประสิทธิ์ประสาทวิชาความรู้และถ่ายทอดประสบการณ์ที่ดีให้แก่ข้าพเจ้า

เนตรสุดา อยู่สันเทียะ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญ

หน้า

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญ (ต่อ)

หน้า

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญ (ต่อ)

หน้า

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญ (ต่อ)

หน้า

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญตาราง

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญรูป

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญรูป (ต่อ)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญรูป (ต่อ)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

เนื่องจากระบบการทำงานที่อยู่บนพื้นฐานของเทคโนโลยีเชิงวัตถุ (Object-Oriented Technology) นั้น กำลังได้รับความนิยมเป็นที่แพร่หลายและมีการนำมาประยุกต์ใช้งานอย่างกว้างขวางในปัจจุบัน ประกอบกับการนำเทคโนโลยีเข้ามาพัฒนาขีดความสามารถของอุปกรณ์ ให้สามารถรองรับงานที่มีความหลากหลายมากขึ้น จึงอาจมีความจำเป็นที่จะต้องเปลี่ยนแปลงหรือปรับปรุงทรัพยากรเดิมที่มีอยู่ให้มีความทันสมัย เพื่อให้สอดคล้องกับการพัฒนาประสิทธิภาพการทำงาน ซึ่งในการเปลี่ยนแปลงโดยทั่วไปนั้น มักจะกระทำเมื่อระบบไม่มีการประมวลผลหรือหยุดพักการทำงานชั่วระยะเวลาหนึ่ง เพื่อให้การเปลี่ยนแปลงที่เกิดขึ้นไม่ส่งผลกระทบต่อระบบการทำงานที่กำลังประมวลผลอยู่ ทำให้ระบบต้องสูญเสียเวลาในการทำงานและการประมวลผลต่างๆ ขาดความต่อเนื่อง ซึ่งนับเป็นสิ่งสำคัญต่อการทำงานโดยรวมของระบบ

การทำงานของโปรแกรมหรือแอพพลิเคชั่นที่เขียนขึ้นด้วยภาษาเชิงวัตถุ (Object-Oriented Programming) เช่น ภาษาจาวา ซึ่งเป็นภาษาหนึ่งที่ใช้ในการเขียนโปรแกรมสำหรับทำงานต่าง ๆ ด้วยข้อดีที่โปรแกรมหรือแอพพลิเคชั่นที่ถูกสร้างขึ้นด้วยภาษานี้ สามารถทำงานได้อย่างอิสระไม่ยึดติดกับแพลตฟอร์มใดๆ จึงทำให้มีการนำภาษานี้มาใช้ในการเขียนโปรแกรมหรือแอพพลิเคชั่นสำหรับใช้งานแพร่หลายมากขึ้น โดยรวมถึงการนำมาใช้ในทำงานบางประเภทที่มีความจำเป็นต้องมีการประมวลผลอยู่ตลอดเวลา เช่น ระบบการประมวลผลงานเกี่ยวกับการเงิน การธนาคาร โทรคมนาคม หรือสายการบิน ซึ่งอาจมีการใช้งานโปรแกรมหรือแอพพลิเคชั่นในงานประเภทดังกล่าวอยู่ตลอดเวลา แต่ในบางครั้งอาจมีความต้องการเปลี่ยนแปลงแก้ไขโปรแกรมหรือฟังก์ชั่นการทำงานบางอย่าง ซึ่งการเปลี่ยนแปลงนี้อาจก่อให้เกิดความเสียหายหรือมีผลกระทบต่องานต่างๆ ที่ประมวลผลอยู่ในขณะนั้นได้ ดังนั้นระบบที่ดีจึงควรสามารถจัดการและควบคุมการเปลี่ยนแปลงที่เกิดขึ้นภายในโปรแกรมหรือแอพพลิเคชั่นในลักษณะของการเปลี่ยนแปลงคลาสหรืออออบเจ็กต์ภายในโปรแกรมที่กำลังทำงานอยู่ โดยที่การทำงานยังคงดำเนินไปอย่างต่อเนื่องและมีความปลอดภัยในระหว่างที่มีการเปลี่ยนแปลงเกิดขึ้นภายในระบบ ซึ่งจะเป็นการรักษาเสถียรภาพและเพิ่มประสิทธิภาพการทำงานมากขึ้น

จากที่กล่าวมาข้างต้น งานวิจัยนี้จึงนำเสนอวิธีการจัดการและควบคุมการโอนย้ายสถานะของออบเจ็กต์ต่างๆ ภายใต้สภาพแวดล้อมสำหรับการทำงานของแอพพลิเคชั่นหรือโปรแกรมที่เขียนด้วยภาษาจาวา โดยสนับสนุนการเปลี่ยนแปลงคลาสที่ใช้ในการทำงานของแอพพลิเคชั่นที่ทำงานภายใต้สภาพแวดล้อมของระบบคอนเทนเนอร์แบบรักษาความต่อเนื่อง

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

(Continuous Migration Container: CMC) ที่มีการใช้กลไลของ Multiple Class Loaders ในการควบคุมและจัดการนำออบเจ็กต์ใหม่เข้ามาทำงานแทนออบเจ็กต์ใดๆ ที่ยังคงมีการทำงานร่วมกับออบเจ็กต์อื่นๆ ในลักษณะการเปลี่ยนแปลงแบบอัพเกรดออบเจ็กต์ (Upgrade Object) โดยระบบ CMC นี้จะสนับสนุนการตรวจสอบความเข้ากันได้ระหว่างคลาสของออบเจ็กต์ใหม่และคลาสของออบเจ็กต์เก่าก่อนจะเข้าสู่กระบวนการโอนย้ายสถานะการทำงานผ่านอินเตอร์เฟส โดยอาศัยความร่วมมือจากแอพพลิเคชั่นที่จะมีการเปลี่ยนแปลงคลาสหรือออบเจ็กต์ ซึ่งจะทำให้มั่นใจได้ว่าระบบยังคงทำงานได้อย่างถูกต้องและต่อเนื่องภายหลังการเปลี่ยนแปลงที่เกิดขึ้น

1.2 ความมุ่งหมายและวัตถุประสงค์ของการศึกษา

จากปัญหาที่กล่าวมาข้างต้น ความมุ่งหมายและวัตถุประสงค์ของการศึกษา คือ

1. เพื่อศึกษาการทำงานของระบบที่อยู่บนพื้นฐานของเทคโนโลยีเชิงวัตถุ (Object-Oriented Technology)
2. เพื่อปรับปรุงประสิทธิภาพการทำงานโดยรวมให้แก่แอพพลิเคชั่นที่ต้องการพัฒนาออบเจ็กต์ให้มีความทันสมัยมากขึ้น
3. เพื่อให้แอพพลิเคชั่นสามารถทำงานได้อย่างต่อเนื่องในขณะมีการเปลี่ยนแปลงออบเจ็กต์
4. เพื่อนำวิธีการรักษาความต่อเนื่องในการทำงาน และระบบคอนเทนเนอร์มาประยุกต์ใช้ในการควบคุมการโอนย้ายสถานะการทำงานของออบเจ็กต์ต่างๆ ผ่านอินเตอร์เฟสเพื่อรองรับการเปลี่ยนแปลงที่จะเกิดขึ้น

1.3 สมมุติฐานของการศึกษา

สมมุติฐานของการศึกษาเพื่อให้บรรลุถึงความมุ่งหมายและวัตถุประสงค์ที่กล่าวมาข้างต้น ประกอบด้วย

1. ระบบสามารถทำงานได้อย่างต่อเนื่องในระหว่างการเปลี่ยนแปลง

   เนื่องจากการทำงานของแอพพลิเคชั่นหรือโปรแกรมภายใต้สภาพแวดล้อมของระบบคอนเทนเนอร์นั้น ระบบ CMC ได้มีการสนับสนุนให้สามารถเปลี่ยนแปลงออบเจ็กต์ภายในแอพพลิเคชั่นที่กำลังทำงานภายใต้ระบบ CMC ได้ โดยคลาสของออบเจ็กต์ใหม่ที่ถูกนำเข้ามาทำงานแทนที่คลาสของออบเจ็กต์เก่าที่กำลังทำงานอยู่ภายในแอพพลิเคชั่นจะสามารถทำงานควบคู่ไปด้วยกันได้ โดยไม่มีการหยุดพักการทำงานของออบเจ็กต์ใดๆ ในแอพพลิเคชั่นที่กำลังทำงานภายใต้ระบบ CMC จึงจัดได้ว่าในระหว่างการเปลี่ยนแปลงที่เกิดขึ้นนั้นระบบการทำงานสามารถทำงานได้อย่างต่อเนื่อง

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2. มีการกระจายการทำงานภายในระบบ CMC

เนื่องจากการทำงานของระบบ CMC มีการทำงานร่วมกันระหว่างคอมโพเนนต์ต่างๆ และในการเปลี่ยนแปลงออบเจ็กต์นั้น ได้มีการใช้กลไกลของ Multiple Class Loaders สำหรับโหลดคลาสของออบเจ็กต์ใหม่ด้วยคลาสโหลดเดอร์ที่แตกต่างกัน เพื่อให้สามารถโหลดคลาสที่มีชื่อเดียวกันภายใต้สภาพแวดล้อมได้ในขณะ Runtime ซึ่งจะทำให้มีการกระจายการทำงานของแต่ละคลาสโหลดเดอร์ได้ รวมทั้งมีการกระจายการทำงานต่างๆ ไปยังแต่ละคอมโพเนนต์ตามหน้าที่ของคอมโพเนนต์นั้นๆ เพื่อให้สามารถจัดการและควบคุมการทำงานต่างๆ ได้ร่วมกัน

1.4 ขอบเขตของการศึกษา

จากสมมุติฐานของการศึกษาข้างต้น สามารถกำหนดขอบเขตของการศึกษาได้ดังนี้

1. มีระบบคอนเทนเนอร์เป็นสภาพแวดล้อมให้กับการทำงานที่มีการนำออบเจ็กต์ใหม่เข้ามาทำงานแทนที่ออบเจ็กต์ที่กำลังทำงานอยู่ในระบบ
2. การตรวจสอบความเข้ากันได้ จะตรวจสอบเฉพาะรายละเอียดพื้นฐาน (Verify Object Behavior) เท่านั้น ไม่มีการตรวจสอบผลการทำงานระหว่างออบเจ็กต์ที่เปลี่ยนแปลง
3. ใช้กลไกของ Multiple Class Loaders ในการจัดการและควบคุมการโอนย้ายสถานะการทำงานของออบเจ็กต์ผ่านอินเตอร์เฟสในขณะอัพเกรดออบเจ็กต์
4. อาศัยความร่วมมือจากแอพพลิเคชั่นในการอิมพลีเมนต์อินเตอร์เฟส เพื่อให้ระบบคอนเทนเนอร์สามารถติดต่อกับแอพพลิเคชั่นเพื่อโอนย้ายสถานะการทำงานของออบเจ็กต์ขณะทำการเปลี่ยนแปลงออบเจ็กต์ได้

1.5 ขั้นตอนของการศึกษา

ขั้นตอนการศึกษาประกอบด้วยขั้นตอนหลัก ๆ ดังนี้

1. ศึกษาทฤษฎีและหลักการทำงานเกี่ยวกับการจัดการการเปลี่ยนแปลงภายในระบบ (Chang Management) การอพยพหรือโอนย้ายสถานะการทำงานของ ออบเจ็กต์ (Object Migration) ผ่านอินเตอร์เฟส
2. ศึกษาทฤษฎีและหลักการของระบบคอนเทนเนอร์เพื่อนำมาประยุกต์ใช้ในการควบคุมการทำงานของคลาสต่างๆ
3. ศึกษาทฤษฎีและหลักการทำงานของคลาสโหลดเดอร์ในจาวา
4. ศึกษาวิธีการที่เหมาะสมต่างๆ เพื่อนำมาใช้ในการวิจัย

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

5. ออกแบบและจัดทำระบบคอนเทนเนอร์แบบรักษาความต่อเนื่อง (Continuous Migration Container: CMC)
6. ทำการทดลองนำแอพพลิเคชั่นเข้ามาทำงานพร้อมทั้งปรับปรุงแก้ไขและวัดผลการทดลอง
7. นำผลการทดลองมาพิสูจน์สมมุติฐานของการศึกษาที่ตั้งไว้
8. วิเคราะห์ วิจารณ์ และสรุปผลที่ได้จากการทดลอง

1.6 ข้อตกลงเบื้องต้น

งานวิจัยนี้มีการจัดทำสภาพแวดล้อมในลักษณะของระบบคอนเทนเนอร์แบบรักษาความต่อเนื่องให้กับการทำงาน (Continuous Migration Container: CMC) เพื่อสนับสนุนการเปลี่ยนแปลงคลาสที่ใช้ในการทำงานของแอพพลิเคชั่นหรือโปรแกรมที่เขียนด้วยภาษาจาวาเท่านั้น

1.7 คำจำกัดความที่ใช้ในการศึกษา

คำว่า "ระบบ CMC" ในวิทยานิพนธ์นี้ให้มีความหมายแทนระบบคอนเทนเนอร์แบบรักษาความต่อเนื่องให้กับการทำงานของแอพพลิเคชั่นหรือโปรแกรมที่เขียนด้วยภาษาจาวา

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 2
การเปลี่ยนแปลงออบเจ็กต์เพื่อให้เกิดความต่อเนื่องในการทำงานและงานวิจัยที่เกี่ยวข้อง

เนื่องจากงานวิจัยนี้เป็นการนำเสนอวิธีการจัดการและควบคุมการเปลี่ยนแปลงออบเจ็กต์เพื่อให้เกิดความต่อเนื่องในการทำงาน ซึ่งจำเป็นต้องใช้หลักการต่างๆ เกี่ยวกับการเปลี่ยนแปลงในระบบเชิงวัตถุและอินเตอร์เฟสของภาษาจาวา มาใช้เป็นแนวทางในการเตรียมดำเนินการตามแนวคิดสำหรับการโอนย้ายสถานะการทำงานของออบเจ็กต์ที่วางไว้ โดยได้มีการนำโครงร่างการทำงานของระบบคอนเทนเนอร์บนพื้นฐานของจาวา (Java Container Framework) มาประยุกต์ใช้ในการสร้างสภาพแวดล้อมให้กับการทำงานของแอพพลิเคชั่น เพื่อให้สามารถจัดการและควบคุมให้แอพพลิเคชั่นนั้นยังคงทำงานได้อย่างต่อเนื่องในขณะมีการเปลี่ยนแปลงออบเจ็กต์ ซึ่งในงานวิจัยต่างๆ ที่ผ่านมานั้นได้มีการนำเสนอวิธีการจัดการและควบคุมการโอนย้ายสถานะการทำงานของทรัพยากรต่างๆ ภายในระบบด้วยวิธีที่แตกต่างกัน จึงได้นำแนวความคิดและวิธีการของงานวิจัยที่เกี่ยวข้องเหล่านี้มาพิจารณาประกอบการจัดทำงานวิจัยนี้ เพื่อศึกษาหาวิธีการที่เหมาะสมและบรรลุวัตถุประสงค์ที่วางไว้ต่อไป

2.1 การเปลี่ยนแปลงออบเจ็กต์เพื่อให้เกิดความต่อเนื่องในการทำงาน

วิทยานิพนธ์นี้ได้มีการนำทฤษฎีและหลักการต่าง ๆ มาใช้ในการออกแบบและจัดทำโครงสร้างของระบบ CMC และส่วนที่ใช้จัดการการเปลี่ยนแปลงที่เกิดขึ้น เพื่อสนับสนุนการรักษาเสถียรภาพให้กับระบบการทำงาน ในขณะที่มีการเปลี่ยนแปลงคลาสที่เป็นส่วนประกอบภายในโปรแกรมหรือแอพพลิเคชั่นที่เขียนด้วยภาษาจาวา ดังนี้

2.1.1 การเปลี่ยนแปลงในระบบเชิงวัตถุ (Change Identification in Object-Oriented Software)

สำหรับระบบที่อยู่บนพื้นฐานของเทคโนโลยีเชิงวัตถุนั้น การจัดการการเปลี่ยนแปลงเป็นการสร้างความมั่นใจในการเปลี่ยนแปลงรายละเอียดที่เกิดขึ้นกับออบเจ็กต์ (Configuration Object) [4] ซึ่งการกำหนดการเปลี่ยนแปลงที่เกิดขึ้นในระบบเชิงวัตถุแบ่งออกเป็น 3 ประเภท [5] คือ

1. Data Change เป็นการเปลี่ยนแปลงตัวแปรประเภท Global และ Local และสมาชิกของคลาสโดยการเปลี่ยนแปลงชนิดการเข้าถึง ขอบเขตการเข้าถึงตัวแปร และการ Initial รวมทั้งการเพิ่ม หรือลบข้อมูลที่มีอยู่

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2. Method Change เป็นการเปลี่ยนแปลงฟังก์ชั่นการทำงานของโปรแกรม แบ่งออกเป็น 3 ชนิด คือ
   2.1 Component Change เป็นการเพิ่ม ลบ หรือเปลี่ยนแปลงลำดับการทำงานของคอมโพเนนต์
   2.2 Structure Change เป็นการเพิ่ม ลบ หรือเปลี่ยนแปลงโครงสร้างการทำงาน เช่น การวนลูปของโปรแกรม
   2.3 Interface Change เป็นการเพิ่ม ลบ หรือเปลี่ยนแปลงรูปแบบ การโต้ตอบกับฟังก์ชั่นอื่นๆ ผ่านอินเตอร์เฟส
3. Class Change เป็นการเปลี่ยนแปลงคลาสโดยตรง แบ่งเป็น 3 ชนิด
   3.1 Component Change เป็นการเปลี่ยนแปลงฟังก์ชั่นที่เป็นสมาชิกของคลาส หรือแอตทริบิวต์
   3.2 Interface Change เป็นการเพิ่มหรือลบแอตทริบิวต์หรือขอบเขตการเข้าถึงอินเตอร์เฟสที่กำหนดไว้
   3.3 Relation Change เป็นการเพิ่มหรือลบ Inheritance หรือ Aggregation หรือความสัมพันธ์ระหว่างคลาสอื่นๆ

โดยวิทยานิพนธ์นี้จะพิจารณาการเปลี่ยนแปลงในระดับของคลาส (Class Change) โดยคลาสของออบเจ็กต์ใหม่จะถูกนำเข้ามาทำงานแทนที่คลาสของออบเจ็กต์เดิมที่ทำงานอยู่ในระบบ

2.1.2 การอินเตอร์เฟสออบเจ็กต์ในภาษาจาวา (Object Interface in Java)

การอินเตอร์เฟสในภาษาจาวาเป็นการรวบรวมชื่อของ Method ในคลาสที่ต้องการมาไว้ในที่แห่งหนึ่งที่สามารถอ้างถึงและเรียกใช้ได้ ในกรณีที่ต้องการใช้คุณสมบัติที่มีอยู่ในคลาสที่ต่างกันโดยไม่ได้เป็นการสร้างคลาสขึ้นมาใหม่แต่เป็นการรวบรวมชื่อของ Method ที่ต้องการเท่านั้น ซึ่งการอินเตอร์เฟสในภาษาจาวาจะใช้ในการรวบรวมค่าคงที่ (Constant) และ Abstract method (เป็น Method ที่ไม่มีการระบุรายละเอียดของการทำงานไว้ภายใน Method) การทำงานจะคล้ายกับคลาสแต่ไม่ใช่คลาสที่เรียกใช้งานได้จริง ๆ

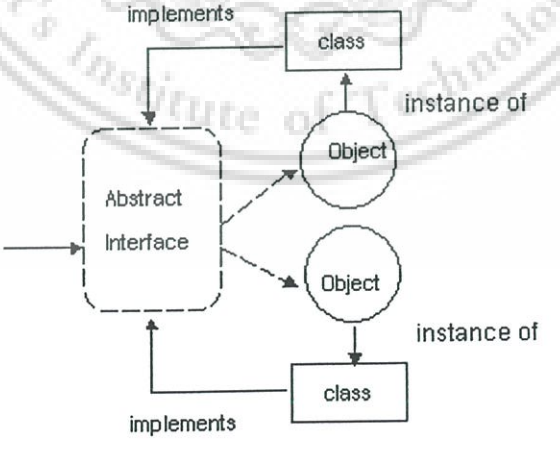
ในการกำหนดรายละเอียดของอินเตอร์เฟส (Object Interface Specifiation) นั้นเป็นการกำหนดรายละเอียดของการอินเตอร์เฟสไปยังออบเจ็กต์หรือกลุ่มของออบเจ็กต์ ที่สามารถกำหนดได้โดยการใช้ภาษาที่ใช้เขียนโปรแกรม ซึ่งวิธีนี้จะมีประสิทธิภาพมากขึ้นหากอินเตอร์เฟสมีความซับซ้อน เนื่องจากจะมีการตรวจสอบไวยากรณ์ การคอมไพล์ทำได้ง่าย และสามารถตรวจสอบความถูกต้องในรายละเอียดของการอินเตอร์เฟสได้ ดังรูปที่ 2.1 เป็นการแสดงการใช้ภาษาจาวาในการกำหนดการอินเตอร์เฟส ที่แสดง method บางตัวที่สามารถใช้กับจำนวนพารามิเตอร์ที่ต่างกันได้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
interface WeatherStation {
        public void WeatherStation () ;
        public void startup () ;
        public void startup (Instrument) ;
        public void shutdown () ;
        public void shutdown (Instrument) ;
        public void reportWeather () ;
        public void test () ;
        public void test (Instrument) ;
        public void calibrate (Instrument i) ;
        public int getID () ;
} // WeatherStation
```

รูปที่ 2.1 Java description of weather station interface

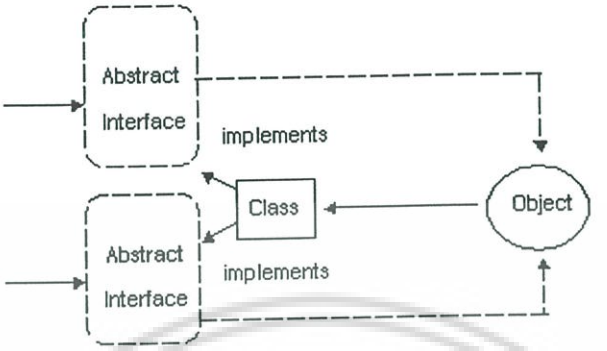
ในภาษาจาวานั้นจะมีการกำหนดอินเตอร์เฟสแยกออกจากออบเจ็กต์ และออบเจ็กต์จะมีการอิมพลีเมนต์เป็นอินเตอร์เฟส นั่นคือ กลุ่มของออบเจ็กต์ทั้งหมดอาจจะถูกเข้าถึงได้โดยผ่านทางอินเตอร์เฟสเพียงอินเตอร์เฟสเดียวเท่านั้น ซึ่งในการกำหนดอินเตอร์เฟสจะเหมือนกับการกำหนดคลาสเพียงแต่ภายในอินเตอร์เฟสจะมีเฉพาะรายชื่อ Method ไม่มีเนื้อหา (Body) ของ Method เมื่อมีการนิยาม "อินเตอร์เฟส" จะหมายถึงการกำหนดให้ทุกคลาสในโปรแกรมนั้นจะต้องสร้าง Method ที่มีชื่อตามที่ระบุไว้ในอินเตอร์เฟสด้วย โดยเนื้อหาของ Method อาจแตกต่างกันไปในแต่ละคลาสขึ้นอยู่กับความต้องการในการใช้งาน แต่ชื่อ Method ต้องเป็นชื่อเดียวกัน



รูปที่ 2.2 Interchangeability of Objects Using an Interface

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สำหรับข้อดีในการอินเตอร์เฟส ประการแรกแสดงในรูปที่ 2.2 [5] เป็นความสามารถที่คลาสต่าง ๆ สามารถอิมพลีเมนต์ Abstract Interface เดียวกันได้ ทำให้สามารถจัดการออบเจ็กต์ของคลาสเหล่านี้ผ่านทางอินเตอร์เฟสเพียงแค่อินเตอร์เฟส เดียวเท่านั้น



รูปที่ 2.3 Manipulating an Object via Different Interfaces

ข้อดีประการที่สองดังรูปที่ 2.3 [5] คือ คลาสหนึ่งคลาสสามารถอิมพลีเมนต์ Abstract Interface ได้หลายอินเตอร์เฟส (Multiple Abstract Interfaces) ซึ่งทำให้สามารถเข้าถึงออบเจ็กต์ของคลาสผ่านทางอินเตอร์เฟสที่แตกต่างกันได้

2.1.3 การอพยพสถานะการทำงานของออบเจ็กต์ (Object Migration)

เป็นการเคลื่อนย้ายสถานะการทำงานหรือเอนทิตี้ที่ทำงานอยู่ที่หนึ่งไปยังอีกที่หนึ่งในระหว่างการทำงาน (Execute) โดยฟังก์ชั่นการทำงานไม่สูญหายไป [5] หรือหมายถึงการทำให้เอนทิตี้หนึ่งถูกย้ายจากที่หนึ่งไปยังที่อื่นที่มีการเปลี่ยนแปลงในสภาพแวดล้อมการทำงานเดียวกัน (Execution Environment) ซึ่งเอนทิตี้ คือ ออบเจ็กต์ หรือกลุ่มของออบเจ็กต์ หรือคอมโพเนนต์ หรือกลุ่มของคอมโพเนนต์ [6]

โดยวิทยานิพนธ์นี้ได้ใช้ลักษณะการอพยพสถานะการทำงานของออบเจ็กต์แบบการโอนย้ายสถานะ (State Transfer) ซึ่งหลักการโอนย้ายสถานะนั้นสามารถทำได้ 2 วิธี [6] คือ

1. Take Snapshot เป็นการนำสถานะของออบเจ็กต์ที่กำลังทำงานขณะนั้นโอนย้ายไปยัง ออบเจ็กต์เป้าหมาย
2. Chang Reference เป็นการเปลี่ยนแปลงการอ้างถึงออบเจ็กต์ที่กำลังทำงานภายในระบบ เพื่อให้ออบเจ็กต์ที่เกี่ยวข้องสามารถเรียกใช้ออบเจ็กต์ที่มีการเปลี่ยนแปลงได้โดยอัตโนมัติ

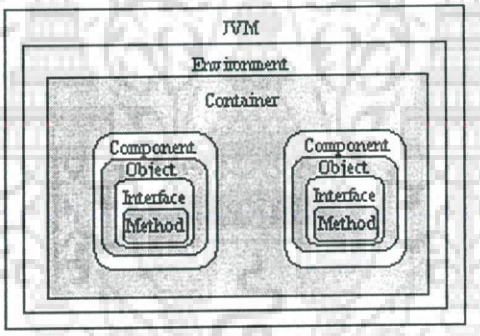
วิทยานิพนธ์นี้ได้ใช้วิธีการ Take Snapshot เนื่องจากเป็นวิธีที่สามารถนำมาอิมพลีเมนต์ได้สะดวกและมีความเหมาะสมกับการทำงานของแอพพลิเคชั่นที่ใช้ในการทดลอง

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2.1.4 ระบบคอนเทนเนอร์บนพื้นฐานของจาวา

เนื่องจากแอพพลิเคชั่นหรือโปรแกรมเชิงวัตถุที่สร้างขึ้นด้วยภาษาจาวานั้น ส่วนใหญ่จะประกอบด้วยคลาสต่างๆ ที่มีการทำงานร่วมกัน ดังนั้น วิทยานิพนธ์นี้จึงได้นำหลักการของระบบคอนเทนเนอร์บนพื้นฐานของจาวามาใช้ในการออกแบบและสร้างสภาพแวดล้อมให้กับการทำงานของแอพพลิเคชั่น เพื่อให้สามารถจัดการและควบคุมการเปลี่ยนแปลงคลาสของออบเจ็กต์ภายในแอพพลิเคชั่นได้อย่างมีประสิทธิภาพ

ระบบคอนเทนเนอร์ คือ ระบบที่เป็นสภาพแวดล้อมสำหรับการทำงานของโปรแกรมหรือแอพพลิเคชั่น (Runtime environment) ซึ่งอยู่บนพื้นฐานของคอมโพเนนต์ (Component-Base Software Engineering[7]) โดยในเทคโนโลยีของจาวานั้น ระบบคอนเทนเนอร์จะเป็นระบบที่มีการทำงานภายใต้สภาพแวดล้อมของ JVM (Java Virtual Machine) ที่มีการทำงานร่วมกันระหว่างคอมโพเนนต์ต่าง ๆ ซึ่งภายในแต่ละคอมโพเนนต์จะประกอบด้วย กลุ่มของอินเตอร์เฟส และการอิมพลีเมนต์เตชั่นของอินเตอร์เฟส หรือคลาสต่างๆ ที่ทำงานร่วมกันตามหน้าที่ของคอมโพเนนต์นั้นๆ ดังรูปที่ 2.4



รูปที่ 2.4 สภาพแวดล้อมของระบบคอนเทนเนอร์

โดยทั่วไปแล้วสามารถจัดประเภทของคอนแทนเนอร์ได้ตามประเภทของแอพพลิเคชั่นแต่ละชนิด ดังนี้

1. Application Client Container ใช้เป็นสภาพแวดล้อมสำหรับการทำงานของแอพพลิเคชั่นชนิด Application Client โดยทั่วไปจะเป็นโปรแกรมที่มีรูปแบบเป็น GUI (Graphic User Interface) ที่มีการทำงานบนคอมพิวเตอร์ประเภท Desktop Computer โดยแอพพลิเคชั่นนี้จะถูกเรียกใช้ที่ Method main ของโปรแกรมและทำงานไปจนกว่า JVM จะถูกจบการทำงานหรือโปรแกรมสิ้นสุดการทำงาน

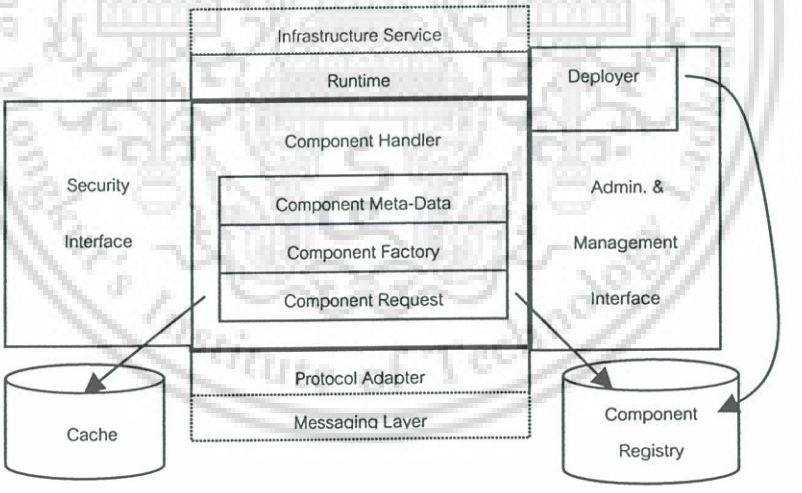
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2. Web Component Container ใช้เป็นสภาพแวดล้อมสำหรับการทำงานของแอพพลิเคชั่นชนิด Web Component คือ Servlet และ JSP (Java Server Page) ที่มีการทำงานใน Web Server และตอบสนองการร้องขอทาง HTTP จาก Web Client ตัวอย่างคอนเทนเนอร์ประเภทนี้ ได้แก่ Servlet Container และ JSP Container

3. Applet Container ใช้เป็นสภาพแวดล้อมสำหรับการทำงานของแอพพลิเคชั่นที่ประกอบด้วยคอมโพเนนต์ชนิด GUI (Graphic User Interface) ที่มีการทำงานใน Web browser แต่สามารถทำงานในแอพพลิเคชั่นอื่นๆ ที่มีความหลากหลายหรืออุปกรณ์ที่สนับสนุนโมเดลการเขียนโปรแกรมแบบ Applet (Applet Programming Model)

4. Enterprise Bean Container ใช้เป็นสภาพแวดล้อมสำหรับการทำงานของคอมโพเนนต์ประเภท EJB (Enterprise Java Bean Component) ที่เป็นแอพพลิเคชั่นแบบธุรกิจเชิงลอจิก (Business Logic Application)

สำหรับระบบคอนเทนเนอร์ที่จัดทำขึ้นในวิทยานิพนธ์นี้ จัดอยู่ในระบบคอนเทนเนอร์ประเภท Application Client Container ซึ่งระบบคอนเทนเนอร์ที่จัดทำขึ้นนี้จะอาศัยพื้นฐานจากโครงสร้างภายในระบบคอนเทนเนอร์บนพื้นฐานของจาวา [7] (Java Container Framework) ที่ประกอบด้วยส่วนต่าง ๆ ดังรูปที่ 2.5



รูปที่ 2.5 โครงสร้างภายในของระบบคอนเทนเนอร์บนพื้นฐานจาวา

1. Protocol Adapter เป็นส่วนที่ใช้รับการร้องขอ (request) แล้วส่งไปยัง Component Request เพื่อส่งไปยัง Component Handler

2. Component Activation Framework เป็นส่วนที่จัดการวงจรชีวิตและการทำงานของคอมโพเนนต์ต่างๆ ประกอบด้วย

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2.1 Component Meta-Data เป็นส่วนที่มีการเก็บข้อมูลเกี่ยวกับการอินเตอร์เฟสและการอิมพลีเมนต์คลาสต่างๆ ของคอมโพเนนต์ประกอบด้วย ชื่อคอมโพเนนต์ที่ใช้ติดต่อกับภายนอก และข้อมูลการทำงาน ได้แก่ ข้อมูลการประมวลผล , สถานะ, กระบวนการทำงาน และนโยบายรักษาความปลอดภัยต่างๆ

2.2 Component Registry เป็นพื้นที่สำหรับบันทึกการมีอยู่ของคอมโพเนนต์ทั้งหมดที่ถูกติดตั้งในคอนเทนเนอร์ เมื่อมีการร้องขอ เข้ามาในคอนเทนเนอร์ ส่วนนี้จะทำหน้าที่เข้าไปค้นหาข้อมูลของคอมโพเนนต์ที่ต้องการ

2.3 Component Request ประกอบด้วยข้อมูลของ method ที่ถูกเรียกใช้ ซึ่งข้อมูลของการเรียกใช้จะประกอบด้วยรหัสของคอมโพเนนต์เป้าหมายและข้อมูลสำหรับการรักษาความปลอดภัยและกระบวนการทำงานต่างๆ

2.4 Component Factory ใช้สร้างและค้นหาคอมโพเนนต์ที่อาจจะถูกอิมพลีเมนต์โดยคอนเทนเนอร์หรือเครื่องมือสำหรับการติดตั้ง (deployment tool) ซึ่งคอนเทนเนอร์จะใช้ส่วนนี้ในการจัดการวงจรชีวิตของคอมโพเนนต์

2.5 Component Handler เป็นตัวรับการร้องขอ และเรียกใช้บนคอมโพเนนต์เป้าหมาย โดยจะควบคุมการกระทำงานและถูกกระทำงาน (activation/passivation) รวมทั้งจัดการสถานะ กระบวนการทำงานและการตรวจสอบสิทธิต่างๆ

2.6 Interceptor ใช้จัดการการทำงานและการตรวจสอบสิทธิต่างๆ ที่กำหนดไว้บนคอมโพเนนต์เพื่อแบ่งเขตในการตรวจสอบการทำงานและการรักษาความปลอดภัย

3. Security Interface เป็นอินเตอร์เฟสที่คอนเทนเนอร์ใช้ในการเข้าถึงสภาพแวดล้อมที่มีการรักษาความปลอดภัยต่าง ๆ

4. Cache เป็นพื้นที่เก็บคอมโพเนนต์ที่กำลังทำงานเอาไว้เพื่อเป็นการปรับปรุงเวลาที่ใช้ในการตอบสนอง (response time) และช่วยให้การใช้งานทรัพยากรต่างๆ เกิดประสิทธิภาพสูงสุด (resource utilization) โดยจะเริ่มเก็บเมื่อกระบวนการทำงานหรือการประมวลผล ได้เสร็จสิ้นลง

5. Runtime เป็นอินเตอร์เฟสให้กับคอนเทนเนอร์และโครงสร้างการให้บริการพื้นฐานภายใน (Infrastructure Service) ช่วยให้สามารถเข้าถึงคอนเทนเนอร์ได้

6. Deployer เป็นอินเตอร์เฟสระหว่าง Runtime และ Administration and Management Interface ทำหน้าที่ติดตั้งคอมโพเนนต์ลงในคอนเทนเนอร์

7. Administration and Management Interface เป็นอินเตอร์เฟสในการติดตั้งและทำลายคอมโพนเนนต์ต่างๆ รวมทั้งทำการรวบรวมข้อมูลทางสถิติของ Runtime

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2.1.5 กลไกของคลาสโหลดเดอร์ (Class Loader Mechanism)

เนื่องจากระบบคอนเทนเนอร์มีการทำงานภายใต้ JVM ซึ่งหน้าที่หลักของ JVM คือ โหลดคลาส (Load class file) และ Execute bytecode ของคลาสต่างๆ โดยภายใน JVM จะมี Class loader ที่มีความยืดหยุ่น เพื่อให้แอพพลิเคชั่นที่เขียนด้วยภาษาจาวาสามารถทำการโหลดคลาสได้ตามความต้องการ [9] แอพพลิเคชั่นที่เขียนด้วยภาษาจาวาสามารถใช้คลาสโหลดเดอร์ได้ 2 ประเภท คือ

1. Primordial Class Loader คลาสโหลดเดอร์ชนิดนี้จัดเป็นส่วนหนึ่งของ JVM ซึ่งแต่ละ JVM จะมีคลาสโหลดเดอร์ชนิดนี้เพียงคลาสโหลดเดอร์เดียวเท่านั้น โดยคลาสโหลดเดอร์ชนิดนี้จะทำหน้าที่โหลดคลาสที่ผ่านการตรวจสอบความถูกต้องแล้ว (Trusted class) ประกอบด้วยคลาสต่างๆ ของ JAVA API

2. Custom Class Loader (Class Loader Object) เป็นคลาสโหลดเดอร์ที่ถูกเขียนหรือกำหนดโดยความต้องการของผู้พัฒนาแอพพลิเคชั่นเพื่อให้สามารถโหลดคลาสต่าง ๆ ได้ตามความต้องการ โดยในเวลา Runtime นั้นแอพพลิเคชั่นที่เขียนด้วยภาษาจาวาจะสามารถติดตั้งคลาสโหลดเดอร์ชนิดนี้สำหรับโหลดคลาสต่างๆ ตามความต้องการได้ ซึ่งคลาสต่าง ๆ ที่จะถูกโหลดผ่านคลาสโหลดเดอร์ชนิดนี้จะจัดเป็นคลาสที่ยังไม่ถูกตรวจสอบความถูกต้อง (Untrusted class) โดยคลาสโหลดเดอร์ประเภทนี้จะมีความสามารถในการโหลดคลาสแบบไดนามิกได้ [10]

วิทยานิพนธ์นี้ได้มีการนำหลักการของคลาสโหลดเดอร์ชนิด Custom Class Loader มาใช้เป็นส่วนหนึ่งของระบบ CMC ซึ่งคลาสโหลดเดอร์ชนิดนี้จะถูกเขียนขึ้นด้วยภาษาจาวา และถูกคอมไพล์เป็นคลาสไฟล์ถูกโหลดเข้าสู่ VM และถูก instantiate เช่นเดียวกับออบเจ็กต์อื่นๆ ซึ่งจัดเป็นส่วนหนึ่งของคำสั่งที่สามารถถูกทำงานภายใต้แอพพลิเคชั่นที่เขียนด้วยภาษาจาวา โดยคลาสโหลดเดอร์มีขั้นตอนการโหลดคลาส [11] ดังนี้

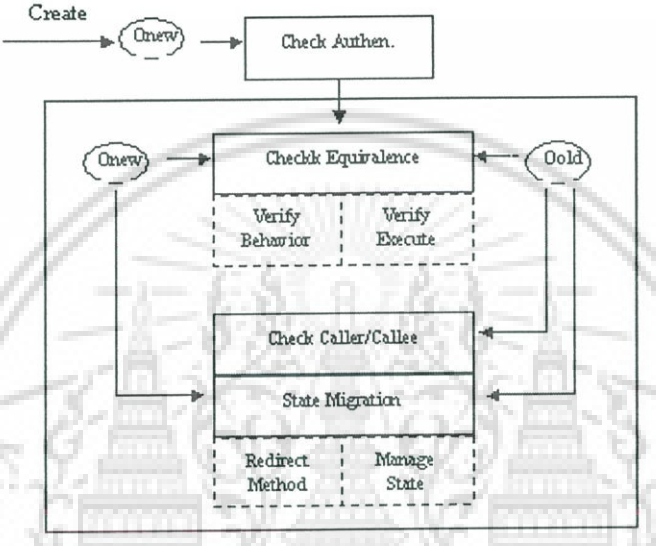
1. กำหนดคลาสที่ต้องการโหลด โดยจะตรวจสอบคลาสนั้นว่าเคยถูกโหลดมาแล้วหรือไม่ ถ้าคลาสนั้นเคยถูกโหลดให้ส่งคลาสที่เคยถูกโหลดนั้นออกไป
2. ถ้าคลาสนั้นยังไม่เคยถูกโหลด จะตรวจสอบว่าเป็นคลาสของระบบ (System Class) หรือเป็นคลาสที่มาจาก JAVA API หรือไม่
3. ถ้าเป็น System Class จะติดต่อ Primordial Class Loader เพื่อทำการโหลดคลาสนั้น
4. ถ้าไม่เป็น System Class จะอ่าน Class file เข้าสู่ array ของไบต์ ซึ่งในขั้นตอนนี้จะเกิดข้อแตกต่างตามชนิดของคลาสโหลดเดอร์ โดยบางคลาสโหลดเดอร์จะโหลดคลาสจากฐานข้อมูลหรือแหล่งที่อยู่ของคลาส ส่วนคลาสโหลดเดอร์ชนิดอื่นอาจจะโหลดคลาสผ่านเครือข่าย
5. ทำการสร้างออบเจ็กต์และ Method ของคลาสนั้นจาก Class file ที่อ่านได้จากข้อ 4
6. Resolve Class ต่างๆ ที่ถูกอ้างถึง
7. ตรวจสอบ Class file ด้วย Verifier

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2.1.6 ขั้นตอนการเปลี่ยนแปลงออบเจ็กต์เพื่อให้เกิดความต่อเนื่องในการทำงาน

การรักษาความต่อเนื่องให้กับการทำงานเมื่อมีการนำออบเจ็กต์ที่สร้างขึ้นใหม่เข้ามาทำงานแทนที่ออบเจ็กต์ที่ยังคงทำงานอยู่ภายในระบบ ประกอบด้วยขั้นตอนดังรูปที่ 2.6

1. การสร้างออบเจ็กต์ใหม่ (Object Creation) เป็นการสร้างออบเจ็กต์ขึ้นมาใหม่ (Onew) เพื่อเข้ามาทำงานแทนที่ ออบเจ็กต์ที่มีอยู่ในระบบ (Oold) โดย Onew จะมีการพัฒนามาจาก Oold นั่นคือ Onew เป็นเวอร์ชั่นใหม่ของ Oold



รูปที่ 2.6 ขั้นตอนการเปลี่ยนแปลง ออบเจ็กต์

2. การตรวจสอบความถูกต้อง (Object Validation) เป็นการตรวจสอบรายละเอียดเพื่อเตรียมการเปลี่ยนแปลงในลักษณะของการอัพเกรดออบเจ็กต์ที่มีอยู่ในระบบ โดยจะตรวจสอบดังนี้

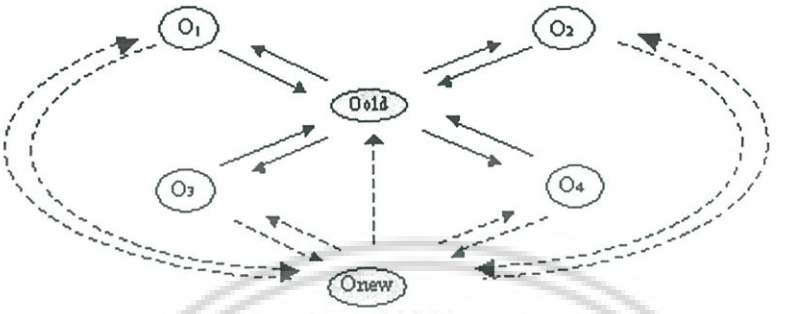
2.1 การตรวจสอบการได้รับสิทธิ (Check Authentication) เป็นการตรวจสอบ Username และ Password ของผู้ที่จะทำการเปลี่ยนแปลงออบเจ็กต์ภายในคอนเทนเนอร์ว่าเป็นผู้ได้รับสิทธิในระบบหรือไม่

2.2 การตรวจสอบความเข้ากันได้ (Check Equivalence) เป็นการระบุออบเจ็กต์ที่จะทำการตรวจสอบความเข้ากันได้ คือ Oold และ Onew ซึ่งจะแบ่งการตรวจสอบเป็น 2 ส่วน คือ

2.2.1 การตรวจสอบรายละเอียด (Verify Behavior) เป็นการตรวจสอบรายละเอียดพื้นฐานของแต่ละออบเจ็กต์ซึ่งประกอบด้วย ข้อมูลเกี่ยวกับฟิลด์ Method และ Constructor

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2.2.2 การตรวจสอบผลการทำงาน (Check Execute) เป็นการตรวจสอบผลการทำงานของแต่ละออบเจ็กต์เพื่อนำมาเปรียบเทียบผลที่ได้จากการทำงานของออบเจ็กต์ทั้งสองว่าให้ผลการทำงานตรงกันหรือไม่ ก่อนจะนำออบเจ็กต์ใหม่เข้ามาทำงานแทนที่ออบเจ็กต์ที่มีอยู่



รูปที่ 2.7 การเปลี่ยนแปลงทิศทางการเรียกใช้ออบเจ็กต์ต่างๆ

3. การอพยพออบเจ็กต์ (Object Migration) เป็นการตรวจสอบการเรียกใช้ Method ของ Oold เพื่อเตรียมความพร้อมในการนำ Onew เข้ามาทำงานแทนที่ Oold และการจัดการการทำงานของ Oold และ Onew

3.1 การตรวจสอบการเรียกใช้ (Check Call Method) เป็นการตรวจสอบว่า Oold มีการเรียกใช้ Method ของออบเจ็กต์ใดในระบบ และมีออบเจ็กต์ใดในระบบที่มีการเรียกใช้ Oold ซึ่งจากการตรวจสอบนี้จะทำให้ทราบว่า Oold มีการเรียกใช้และถูกเรียกใช้โดยออบเจ็กต์ใดบ้าง

3.2 การโอนย้ายสถานะ (State Transfer) เป็นการเปลี่ยนแปลงทิศทางการเรียกใช้ Method ของ Oold ให้ไปเรียกใช้ Onew และจัดการการเรียกใช้ Method ใน Oold เพื่ออพยพสถานะการทำงานที่ค้างอยู่ใน Oold ไปทำงานต่อใน Onew

4. การลบหรือทำลายออบเจ็กต์เก่า (Release Old Object) เป็นการลบหรือทำลาย Oold ออกจากระบบ โดยตรวจสอบจาก Garbage Collection ที่เก็บออบเจ็กต์ที่ไม่มีการเรียกใช้หรือไม่มีการอ้างถึงเอาไว้เพื่อเตรียมทำลาย โดยการใช้ Method finalize เพื่อปลดปล่อยทรัพยากรที่ Oold ยึดครองอยู่ จากนั้น Garbage Collection จะทำลาย Oold โดยอัตโนมัติภายหลังการเรียกใช้ Method นี้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2.2 งานวิจัยที่เกี่ยวข้อง

งานวิจัยที่เกี่ยวข้องกับวิทยานิพนธ์นี้เป็นงานวิจัยเกี่ยวกับการรักษาเสถียรภาพให้กับระบบการทำงานในขณะที่มีการเปลี่ยนแปลงส่วนประกอบภายในระบบต่าง ๆ ซึ่งในที่นี้จะขอยกตัวอย่างงานวิจัยที่เกี่ยวข้อง ดังนี้คือ การเปลี่ยนแปลงทรัพยากรแบบไดนามิก การจัดการและควบคุมวงจรชีวิตของออบเจ็กต์ที่มีการทำงานแบบกระจาย และเทคนิคการปรับปรุงซอฟแวร์ของจาวาแบบไดนามิก โดยงานวิจัยที่เกี่ยวข้องเหล่านี้ได้มีการนำเสนอวิธีการในการจัดการการเปลี่ยนแปลงออบเจ็กต์ที่มีการทำงานภายในระบบต่างๆ ในรูปแบบที่แตกต่างกัน ดังจะกล่าวในรายละเอียดต่อไป

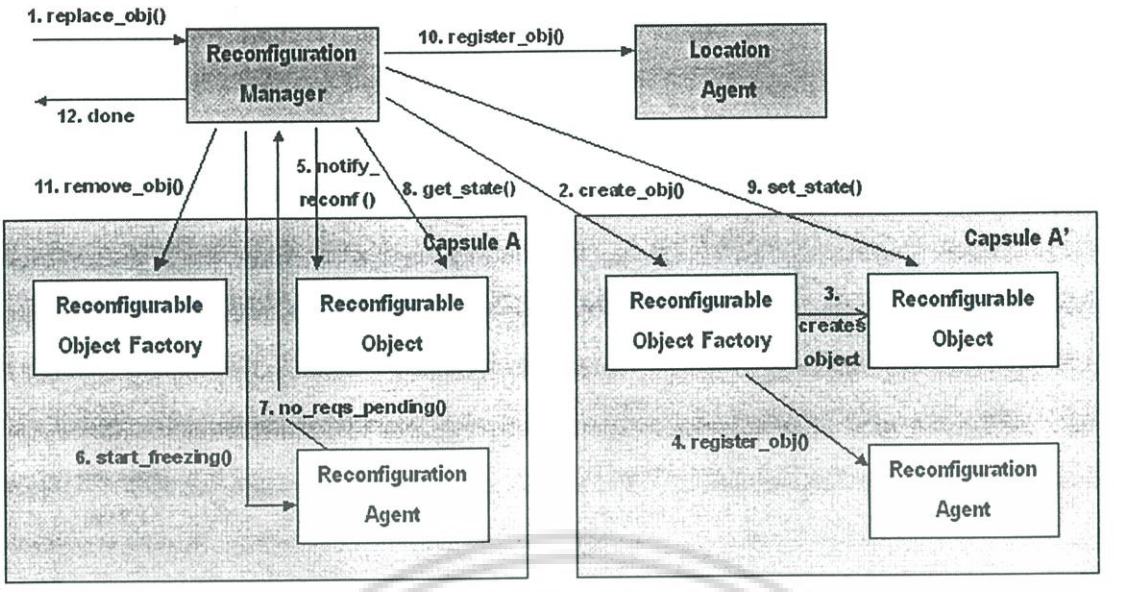
2.2.1 การเปลี่ยนแปลงทรัพยากรแบบไดนามิก (Dynamic Reconfiguration)

การเปลี่ยนแปลงทรัพยากรแบบไดนามิกในสถาปัตยกรรม CORBA [1] ได้มีการนำเสนอวิธีการเปลี่ยนแปลงออบเจ็กต์เพื่อให้การอัพเกรดทรัพยากรในระบบการทำงานแบบกระจายสามารถทำงานได้อย่างต่อเนื่องในขณะที่มีการเปลี่ยนแปลง โดยมีการจัดให้ออบเจ็กต์ต่างๆ อยู่ในแคปซูลที่มีส่วนประกอบสำหรับสร้างออบเจ็กต์ที่จะใช้ในการทำงานขึ้นมาภายในระบบ เมื่อมีความต้องการเปลี่ยนแปลงออบเจ็กต์ในลักษณะการนำออบเจ็กต์ใหม่เข้ามาทำงานแทนที่ออบเจ็กต์ที่มีอยู่ภายในระบบนั้น จะมีวิธีการจัดการดังรูปที่ 2.8

จากรูปการเปลี่ยนแปลงออบเจ็กต์ในงานวิจัยนี้มีส่วนที่ใช้จัดการและควบคุมการเปลี่ยนแปลงที่เกิดขึ้นทั้งหมด (Reconfiguration Manager) โดยส่วนนี้จะติดต่อไปยังส่วนที่ทำการสร้างออบเจ็กต์ (Reconfigurable Object Factory) ทำการสร้างออบเจ็กต์ (Reconfigurable Object) ขึ้นภายในแคปซูล และจะติดต่อไปยังส่วนที่ทำหน้าที่จำกัดพฤติกรรม (Behavior) ของออบเจ็กต์ที่จะได้รับผลกระทบในระหว่างการเปลี่ยนแปลง (Reconfiguration Agent)

จากนั้น Reconfiguration Manager จะแจ้งการเปลี่ยนแปลงไปยังแคปซูลที่จะมีการเปลี่ยนแปลงออบเจ็กต์ให้หยุดพักการทำงานของออบเจ็กต์ที่จะถูกเปลี่ยนแปลงชั่วคราว เพื่อนำสถานะของออบเจ็กต์ในขณะนั้นไปใช้ในการตั้งค่าสถานะของออบเจ็กต์ใหม่ที่ถูกสร้างขึ้น เมื่อออบเจ็กต์ใหม่ถูกตั้งค่าเรียบร้อยแล้ว Reconfiguration Manager จะติดต่อไปยัง Location Agent เพื่อบันทึกตำแหน่งที่อยู่ (Location) ของออบเจ็กต์ในแคปซูลนั้น และติดต่อไปยัง Reconfiguration Object Factory เพื่อลบหรือทำลายออบเจ็กต์เก่าออกไปจากแคปซูล

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 2.8 การโอนย้ายสถานะออบเจ็กต์ของงานวิจัย Transparent Dynamic Reconfiguration for CORBA

<u>วิธีการโอนย้ายสถานะ</u> ใช้วิธีหยุดพักการทำงานของออบเจ็กต์ที่ต้องการเปลี่ยนแปลงชั่วระยะเวลาหนึ่ง เพื่อโอนย้ายสถานะนั้นไปยังออบเจ็กต์ใหม่ที่จะนำเข้ามาทำงานแทนที่ เมื่อศึกษาวิธีการของงานวิจัยนี้พบว่า

1. จำเป็นต้องหยุดพักการทำงานของออบเจ็กต์ที่ต้องการเปลี่ยนแปลงชั่วคราว
2. ระบบการทำงานอาจเกิดปัญหาคอขวด เนื่องจากมีส่วนที่ใช้จัดการและควบคุมการเปลี่ยนแปลงที่เกิดขึ้นทั้งหมดเพียงส่วนเดียว
3. เป็นวิธีการที่ใช้ได้เฉพาะออบเจ็กต์ที่เป็น Non-redundant object เท่านั้น (Non-redundant object คือ ออบเจ็กต์ที่มีการทำงานไม่เป็น Multi-thread)
4. ยังไม่มีการตรวจสอบการทำงานของระบบหลังจากการเปลี่ยนแปลงว่าออบเจ็กต์ใหม่จะทำหน้าที่แทนออบเจ็กต์เก่าได้อย่างถูกต้องหรือไม่

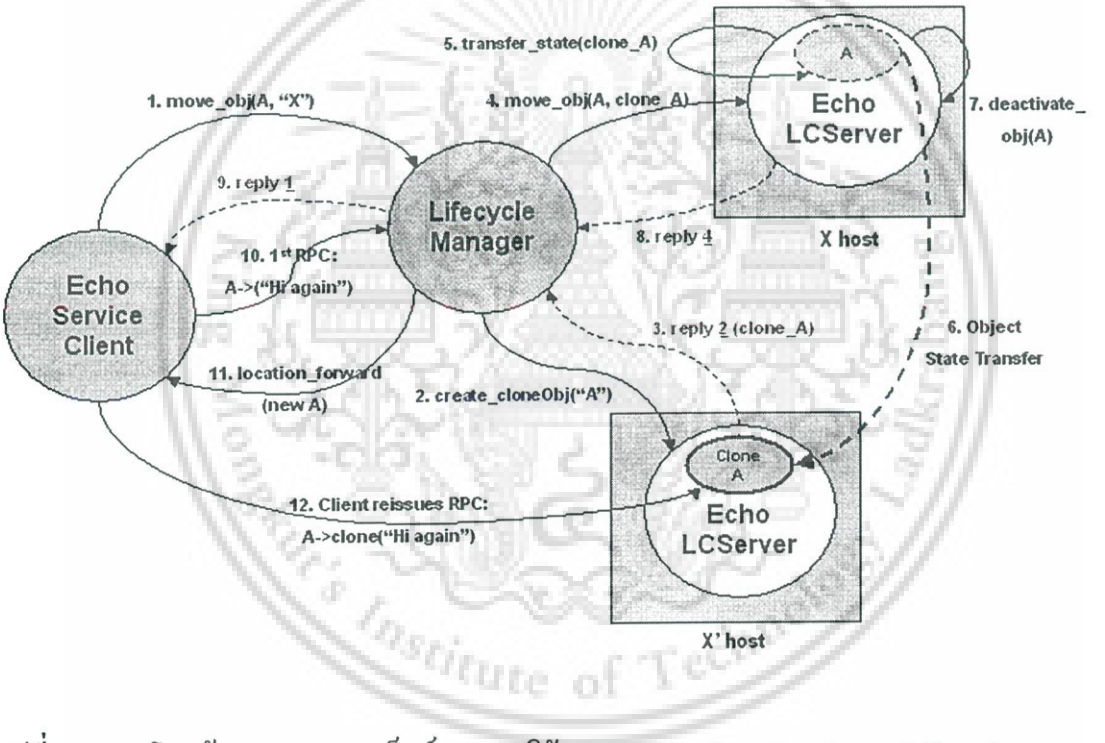
2.2.2 การจัดการและควบคุมวงจรชีวิตของออบเจ็กต์ที่มีการทำงานแบบกระจาย

สำหรับโครงร่างการทำงาน (Framework) ที่มีการใช้อินเตอร์เฟสสำหรับจัดการและควบคุมวงจรชีวิตของออบเจ็กต์ที่มีการทำงานแบบกระจายในสถาปัตยกรรม CORBA [2] นั้น เป็นการนำเสนอวิธีการเปลี่ยนแปลงออบเจ็กต์สำหรับการทำงานแบบ Client/Server ที่มีการใช้วิธีการโอนย้ายสถานะการทำงานของออบเจ็กต์เก่าเมื่อได้รับคำสั่งให้ทำการอพยพสถานะ (Migrate Object) ดังรูปที่ 2.9

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สำนักหอสมุดกลาง พระจอมเกล้าลาดกระบัง

การเปลี่ยนแปลงออบเจ็กต์ของงานวิจัยนี้มีส่วนที่ใช้จัดการและควบคุมการเปลี่ยนแปลงที่เกิดขึ้นซึ่งคล้ายกับงานวิจัยที่กล่าวไว้ในข้อ 2.1.1 คือ Lifecycle Manager ทำหน้าที่ควบคุมวงจรชีวิตและภาระการทำงานของออบเจ็กต์ที่อยู่ในระบบ แต่มีวิธีการโอนย้ายสถานะที่แตกต่างกัน โดยเครื่องปลายทาง (Echo LCServer หรือ X' host) จะทำหน้าที่สร้างออบเจ็กต์ใหม่ขึ้นมา (Clone Object) เมื่อ LC Manager ได้รับคำสั่งจาก Echo Service Client ให้ทำการอพยพหรือเปลี่ยนแปลงออบเจ็กต์ที่มีอยู่เดิมไปยังที่ใหม่ จากนั้น LCManager จะติดต่อไปยังเครื่องต้นทาง (X host) เพื่อหยุดการทำงานของออบเจ็กต์ที่จะถูกเปลี่ยนแปลงแล้วนำสถานะของออบเจ็กต์นั้นไปใช้ตั้งค่าสถานะของออบเจ็กต์บนเครื่องปลายทาง จากนั้นจะติดต่อไปยัง Echo Service Client เพื่อตอบรับการเปลี่ยนแปลงที่เกิดขึ้นเพื่อให้ Client ส่งสัญญาณการติดต่อเพื่อขอรับตำแหน่งที่อยู่ใหม่ของออบเจ็กต์ที่มีการเปลี่ยนแปลง



รูปที่ 2.9 การโอนย้ายสถานะออบเจ็กต์ของงานวิจัย LocALE: a Location-Aware Lifecycle Environment for Ubiquitous Computing

<u>วิธีการโอนย้ายสถานะ</u> เริ่มเมื่อได้รับคำสั่งให้ทำการอพยพสถานะ โดยใช้วิธีการสร้างออบเจ็กต์ใหม่ขึ้นมา (Clone Object) เพื่อใช้ตั้งค่าสถานะของมัน (Set state) ให้เป็นไปตามสถานะของออบเจ็กต์เก่า ซึ่งจะทำให้สามารถอพยพสถานะของออบเจ็กต์ต่างๆ ในระบบ Client/Server ได้ เมื่อศึกษาวิธีการของงานวิจัยนี้พบว่า

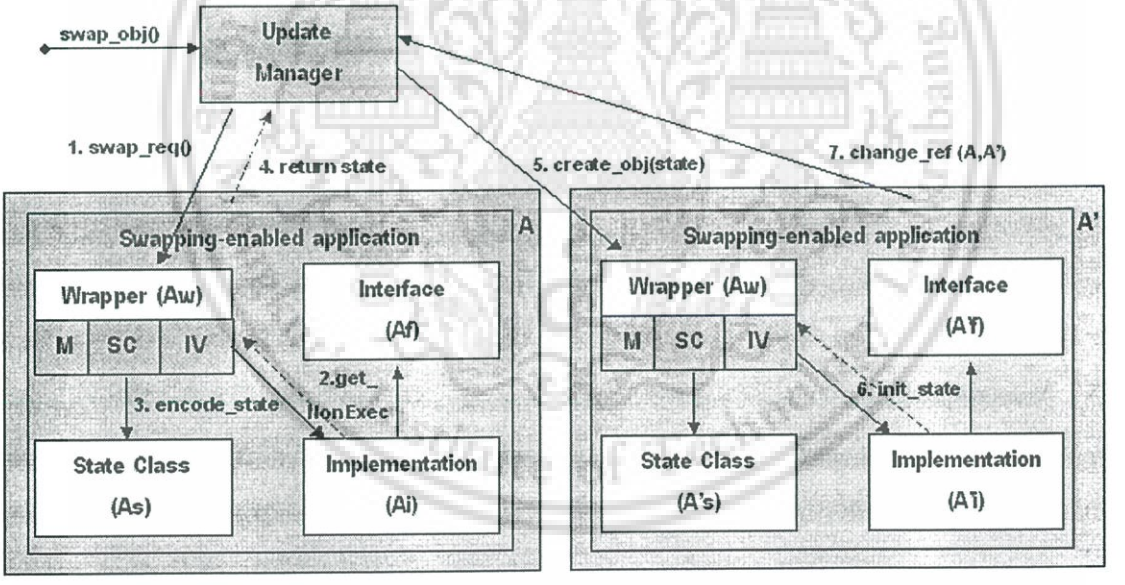
1. จำเป็นต้องหยุดพักการทำงานของออบเจ็กต์ที่ต้องการเปลี่ยนแปลงชั่วคราว

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้
60255

2. ในการร้องขอระหว่าง Client และ Server สำหรับการเปลี่ยนแปลงออบเจ็กต์จะผ่านอินเตอร์เฟสเพียงจุดเดียว จึงอาจเกิดการเสียหายหรือมีปัญหาคอขวดซึ่งจะส่งผลให้การทำงานล่าช้าหรือหยุดชะงักได้
3. ภายหลังการเปลี่ยนแปลงนั้น Client ต้องสูญเสียเวลาในการติดต่อไปยังอินเตอร์เฟสเพื่อให้อินเตอร์เฟสตอบกลับตำแหน่งที่อยู่ของออบเจ็กต์ใหม่ (location forward) เพื่อให้ Client สามารถติดต่อกับออบเจ็กต์ใหม่ครั้งต่อไปได้
4. ยังไม่มีการตรวจสอบการทำงานของระบบหลังจากการเปลี่ยนแปลงว่าออบเจ็กต์ใหม่จะทำหน้าที่แทนออบเจ็กต์เก่าได้อย่างถูกต้องหรือไม่

2.2.3 เทคนิคการปรับปรุงซอฟแวร์ของจาวาแบบไดนามิก

งานวิจัยนี้เป็นการนำเสนอเทคนิคสำหรับการปรับปรุงซอฟต์แวร์ (Update Software) ของจาวาแบบไดนามิก [3] โดยเป็นเทคนิคที่ใช้ในการเปลี่ยนแปลงแอพพลิเคชั่นในลักษณะคงที่ (Static Modify) เพื่อให้สามารถนำมาปรับปรุงได้แบบไดนามิก (Dynamic Update) โดยมีขั้นตอนการเปลี่ยนแปลงออบเจ็กต์ของแอพพลิเคชั่นดังรูปที่ 2.10



รูปที่ 2.10 การโอนย้ายสถานะออบเจ็กต์ของงานวิจัย A Technique for Dynamic Updating of Java Software

การเปลี่ยนแปลงออบเจ็กต์ของแอพพลิเคชั่นที่งานวิจัยนี้นำเสนอนั้นจะกระทำภายใต้การสร้างแอพพลิเคชั่นที่สามารถเปลี่ยนแปลงออบเจ็กต์ได้ (Swapping-enabled application) โดยมีส่วนที่ใช้จัดการและควบคุมการเปลี่ยนแปลงที่เกิดขึ้นเช่นเดียวกับงานวิจัยที่กล่าวมาข้างต้น คือ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Update Manager โดยใช้หลักการของ Wrapper Class ในการสร้างคลาสที่มีชื่อเหมือนกันและมีอินเตอร์เฟสเดียวกันกับคลาสต้นแบบ และทำหน้าที่เป็น Proxy สำหรับคลาสนั้น ดังนั้นทุกครั้งที่ Method ถูกเรียกใช้แล้ว Wrapper Class จะ Forward การเรียกใช้ Method ที่สอดคล้องไปยังออบเจกต์ที่เหมาะสม แต่เนื่องจากคุณสมบัติของ Wrapper Class จะไม่สามารถรองรับการทำ Reflection ได้ จึงทำให้งานวิจัยนี้ยังมีข้อจำกัดบางประการ (การทำ Reflection คือ การยอมให้โปรแกรมสามารถเข้าถึงข้อมูลเกี่ยวกับ Field, Method และ Constructor ของคลาสที่ถูกโหลด และใช้ข้อมูลเหล่านี้มาทำงานตามที่ต้องการบนออบเจ็กต์ภายในระบบ ซึ่งนับว่ามีประโยชน์มากในการทำงานที่ต้องการข้อมูลพื้นฐานของคลาสต่างๆ ในแอพพลิเคชั่น)

วิธีการโอนย้ายสถานะ มีการใช้ Wrapper Class ทำหน้าที่เป็น Proxy Class เพื่อทำการเปลี่ยนแปลงโอนย้ายคลาสต่างๆ (Hot Swap) ในเวลา Run time เมื่อศึกษาวิธีการของงานวิจัยนี้พบว่า

1. ภายในแอพพลิเคชั่นที่จะนำมา Update นั้นจะต้องไม่มีคลาสใดที่เข้าถึง public หรือ protect field ของคลาสที่ต้องการ Update (Target class) โดยตรง การเข้าถึงฟิลด์ดังกล่าวจะต้องทำผ่านตัวเข้าถึง (accessor) ที่เหมาะสม เช่น get หรือ set method ข้อจำกัดนี้จึงทำให้ไม่สามารถนำมาประยุกต์ใช้กับฟิลด์ที่เป็น constant และไม่สามารถเปลี่ยนแปลงค่าของฟิลด์จากเวอร์ชั่นเก่าไปยังเวอร์ชั่นใหม่ได้
2. เนื่องจากงานวิจัยนี้ได้ตั้งสมมติฐานไว้ว่า จะไม่มีการนำคุณสมบัติของ Reflection ไปประยุกต์ใช้ใน Target Class หรือส่วนประกอบของ Target Class ส่งผลให้ถ้ามีคำสั่งในแอพพลิเคชั่นที่มีการใช้ข้อมูลจากการทำ Reflection เกี่ยวกับ Target class หรือสมาชิกของมันแล้ว การทำหน้าที่แทนของ Target class ด้วย Wrapper class จะส่งผลกระทบต่อพฤติกรรม (Behavior) ของแอพพลิเคชั่นได้
3. เวอร์ชั่นใหม่ของ Target Class ต้องมีอินเตอร์เฟสเดียวกับเวอร์ชั่นเก่าของคลาสนั้น นั่นคือ แต่ละเวอร์ชั่นของ Target Class ต้องมีกลุ่มของ Public Method เหมือนกัน ซึ่งจะส่งผลให้ไม่สามารถประยุกต์ใช้กับ Private Method และ Instance Variable ที่สามารถเพิ่มเติมหรือลบออกจากเวอร์ชั่นใหม่ของคลาสได้อย่างอิสระ

2.2.4 สรุปปัญหาและข้อจำกัดของงานวิจัยที่เกี่ยวข้อง

จากการศึกษางานวิจัยที่เกี่ยวข้องดังกล่าว สามารถสรุปปัญหาและข้อจำกัดได้ดังนี้

1. จำเป็นต้องหยุดพักการทำงานของออบเจ็กต์ที่ต้องการเปลี่ยนแปลง ซึ่งอาจส่งผลให้การทำงานที่มีการใช้งานออบเจ็กต์ในขณะนั้นต้องหยุดชะงักตามไปด้วย นั่นหมายถึงขาดความต่อเนื่องในการทำงานขณะทำการเปลี่ยนแปลงออบเจ็กต์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2. ระบบการทำงานอาจเกิดปัญหาคอขวด เนื่องจากมีส่วนที่ใช้จัดการและควบคุมการเปลี่ยนแปลงที่เกิดขึ้นทั้งหมดเพียงส่วนเดียว
3. ยังไม่มีการตรวจสอบการทำงานของระบบหลังจากการเปลี่ยนแปลงว่าออบเจ็กต์ใหม่จะทำหน้าที่แทนออบเจ็กต์เก่าได้อย่างถูกต้องหรือไม่
4. ไม่สามารถเปลี่ยนแปลงออบเจ็กต์ในแอพพลิเคชั่นที่มีการใช้คุณสมบัติของ Reflection ได้ เนื่องจากข้อจำกัดของ Wrapper Class

จากปัญหาและข้อจำกัดของงานวิจัยที่เกี่ยวข้องต่างๆ วิทยานิพนธ์นี้จึงมีแนวคิดและวิธีการสำหรับจัดการและควบคุมการเปลี่ยนแปลงคลาสที่ใช้ในการทำงานของแอพพลิเคชั่นที่ทำงานภายใต้สภาพแวดล้อมของระบบคอนเทนเนอร์แบบรักษาความต่อเนื่อง (Continuous Migration Container: CMC) โดยระบบ CMC นี้จะสนับสนุนการตรวจสอบความเข้ากันได้ระหว่างคลาสของออบเจ็กต์ใหม่และคลาสของออบเจ็กต์เก่าก่อนจะเข้าสู่กระบวนการเปลี่ยนแปลง ซึ่งจะทำให้มั่นใจได้ว่าระบบยังคงทำงานได้อย่างถูกต้องและต่อเนื่องภายหลังการเปลี่ยนแปลงที่เกิดขึ้น

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 3
การโอนย้ายสถานะของออบเจ็กต์ภายใน ระบบคอนเทนเนอร์แบบรักษาความต่อเนื่องในการทำงาน

ในการทำงานของแอพพลิเคชั่นหรือโปรแกรมที่เขียนด้วยภาษาจาวาบางประเภทที่มีลักษณะการทำงานหรือมีการประมวลผลอยู่ตลอดเวลานั้น หากมีความต้องการเปลี่ยนแปลงส่วนประกอบหรือออบเจ็กต์ใดๆ ของแอพพลิเคชั่นแล้ว ส่วนใหญ่มักจะทำการเปลี่ยนแปลงเมื่อแอพพลิเคชั่นหยุดทำงานเพื่อไม่ให้ส่งผลกระทบต่อการทำงานส่วนอื่นๆ ซึ่งเมื่อแอพพลิเคชั่นหยุดการทำงาน สิ่งที่ตามมาคือ ความสูญเสียเวลาในการทำงานซึ่งเป็นสิ่งสำคัญต่อประสิทธิภาพโดยรวมของระบบ และในกรณีที่มีการเปลี่ยนแปลงออบเจ็กต์ใดๆ ในระหว่างที่แอพพลิเคชั่นกำลังทำงานนั้น หากไม่มีการจัดการและควบคุมการเปลี่ยนแปลงที่เหมาะสม การเปลี่ยนแปลงที่เกิดขึ้นอาจส่งผลกระทบต่อออบเจ็กต์อื่นๆ ที่เกี่ยวข้องภายในแอพพลิเคชั่นนั้นได้

ดังนั้นงานวิจัยนี้จึงได้มีแนวความคิดสำหรับจัดการและควบคุมการเปลี่ยนแปลงออบเจ็กต์ภายในแอพพลิเคชั่นหรือโปรแกรมที่เขียนด้วยภาษาจาวา โดยแอพพลิเคชั่นสามารถทำงานได้อย่างต่อเนื่องในขณะที่มีการเปลี่ยนแปลงออบเจ็กต์ ภายใต้สภาพแวดล้อมในรูปของระบบคอนเทนเนอร์แบบรักษาความต่อเนื่อง (Continuous Migration Container : CMC) ที่มีการใช้กลไกของ Multiple Class Loaders ในการนำคลาสใหม่ที่มีชื่อเดียวกับคลาสที่กำลังทำงานเข้าสู่สภาพแวดล้อมเดียวกันในขณะ Runtime ได้แบบไดนามิก และมีการตรวจสอบความเข้ากันได้ระหว่างคลาสที่จะมีการเปลี่ยนแปลงก่อนทำการโอนย้ายสถานะการทำงานของออบเจ็กต์ต่างๆ ไปยังออบเจ็กต์ใหม่ผ่านอินเตอร์เฟส โดยในการโอนย้ายสถานะการทำงานของออบเจ็ตน์นั้นจะอาศัยความร่วมมือจากแอพพลิเคชั่นในการอิมพลีเมนต์อินเตอร์เฟสสำหรับใช้ในการติดต่อระหว่างแอพพลิเคชั่นและระบบ CMC เพื่อช่วยให้การทำงานของระบบยังคงต่อเนื่องและมีความปลอดภัยในระหว่างการเปลี่ยนแปลงออบเจ็กต์ของแอพพลิเคชั่นที่มีความต้องการทำงานตลอดเวลา

3.1 ระบบคอนเทนแนอร์แบบรักษาความต่อเนื่องในการทำงาน (CMC)

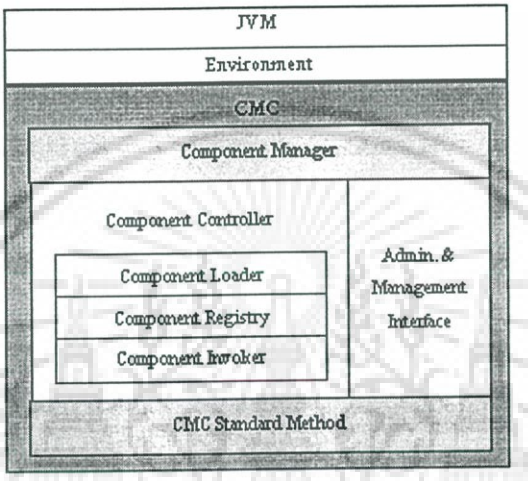
เมื่อพิจารณาสภาพการทำงานร่วมกันระหว่างออบเจ็กต์ ตามกระบวนการรักษาความต่อเนื่องให้กับการทำงานแล้ว พบว่าควรจัดให้ออบเจ็กต์ต่างๆ อยู่ในสภาพแวดล้อมเดียวกัน เพื่อให้สามารถจัดการและควบคุมการทำงานได้อย่างมีประสิทธิภาพ วิทยานิพนธ์นี้จึงได้นำโครงสร้างของระบบคอนเทนเนอร์ของจาวา (Java Container Framework) [7] ที่มีการนำมาใช้ควบคุมและจัดการการทำงานของคอมโพเนนต์ต่างๆ ที่อยู่ภายใน และนำหลักการของ Collections Framework [8]

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

มาใช้สำหรับออกแบบและจัดทำระบบคอนเทนเนอร์แบบรักษาความต่อเนื่อง (Continuous Migration Container : CMC) ซึ่งมีรายละเอียดดังนี้

3.1.1 โครงสร้างของ CMC

จากการนำระบบคอนเทนเนอร์ของจาวามาใช้ ทำให้มีการปรับปรุงโครงสร้างของระบบ CMC ให้มีการรองรับการเปลี่ยนแปลงออบเจ็กต์ใดๆ โดยภายในระบบ CMC จะประกอบด้วยส่วนต่างๆ ดังรูปที่ 3.1



รูปที่ 3.1 โครงสร้างของระบบ CMC

จากภาพระบบ CMC จะมีการทำงานบนสภาพแวดล้อมภายใต้การทำงานของ JVM (Java Virtual Machine) โดย CMC จัดเป็นระบบคอนเทนเนอร์สำหรับการทำงานของแอพพลิเคชั่น (Application Container) ซึ่งเป็นสภาพแวดล้อมประเภท Runtime Environment ชนิดหนึ่งเพื่อให้ออบเจ็กต์ต่าง ๆ สามารถทำงานภายใต้สภาพแวดล้อมเดียวกัน ภายในโครงสร้างของ CMC จะมีการทำงานร่วมกันระหว่างคอมโพเนนต์ที่เป็นกลุ่มของคลาส อินเตอร์เฟส และการอิมพลีเมนต์ของอินเตอร์เฟสต่างๆ ดังนี้

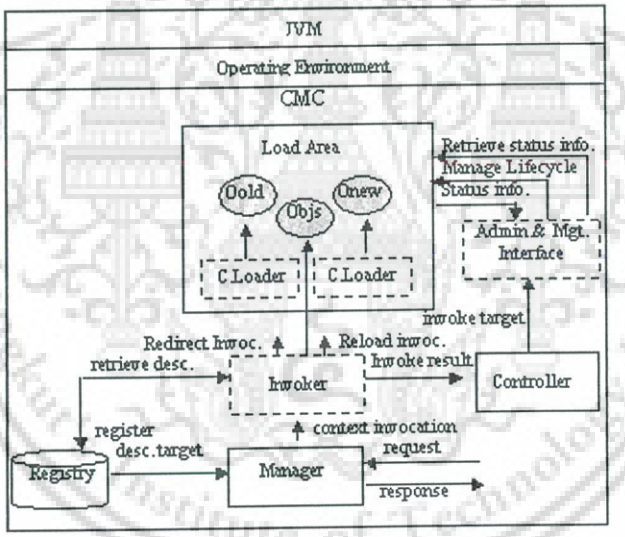
1. Component Manager : เป็น User Interface ของ CMC ทำหน้าที่ติดต่อกับส่วนอื่นๆ ภายใน CMC เพื่อให้ผู้ใช้สามารถเข้าถึงการทำงานของ CMCได้
2. Component Controller: เป็นส่วนสนับสนุนการตรวจจับเหตุการณ์และสถานะการทำงานและสถานะการเรียกใช้ของ ออบเจ็กต์ ต่างๆ ภายใน CMC
3. Component Loader: ทำหน้าที่โหลดคลาสต่างๆ เมื่อทำการนำคลาสเหล่านั้นเข้าสู่การทำงานของ CMC

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

4. Component Registry: ทำหน้าที่ตรวจสอบและจัดเก็บข้อมูลพื้นฐาน (Behavior) ของคลาสหรือออบเจ็กต์ต่างๆ ที่มีการทำงานภายใน CMC
5. Component Invoker: ทำหน้าที่ควบคุมการทำงานของออบเจ็กต์เป้าหมายตามที่ได้รับคำสั่ง
6. Administration and Management Interface: เป็นอินเตอร์เฟสสำหรับให้ ออบเจ็กต์ต่างๆ ภายใน CMC สามารถเรียกใช้กรณีที่มีการเปลี่ยนแปลงออบเจ็กต์
7. CMC Standard Method: เป็นส่วนที่รวบรวม method ของ CMC เพื่อใช้ในการทำงานที่เกี่ยวข้องกับ CMC โดยตรงทั้งหมด

3.1.2 การทำงานร่วมกันระหว่างส่วนประกอบต่างๆ ของ CMC

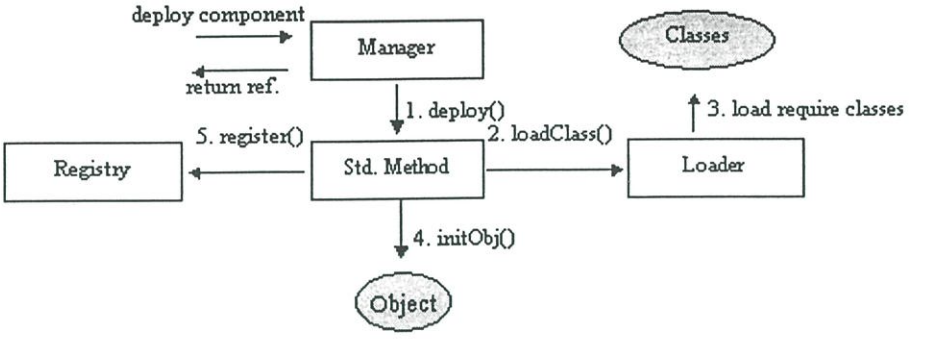
CMC เป็นสภาพแวดล้อมสำหรับจัดการการทำงานของออบเจ็กต์ และจัดการการทำงานของแอพพลิเคชั่นให้มีความต่อเนื่องในระหว่างการอัพเกรดออบเจ็กต์ โดยภายในจะประกอบด้วยคอมโพเนนต์ต่างๆ ที่มีการทำงานร่วมกันดังภาพ



รูปที่ 3.2 การทำงานร่วมกันระหว่างคอมโพเนนต์ต่างๆ ภายใน CMC

การนำคลาสหรือคอมโพเนนต์เข้าสู่ CMC นั้น Component Manager จะติดต่อไปยัง Standard Method เพื่อทำการโหลดคลาสต่างๆ โดย Component Loader และ Component Registry จะทำการตรวจสอบและจัดเก็บข้อมูลรายละเอียดของคลาสหรือคอมโพเนนต์ที่ถูกนำเข้าสู่ CMC ซึ่งกระบวนการติดตั้งคลาสหรือคอมโพเนนต์เข้าสู่ CMC (Deployment Process) ประกอบด้วยขั้นตอนดังภาพ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 3.3 การนำคลาสหรือคอมโพเนนต์เข้าสู่ CMC

สำหรับการกระทำการใดๆ กับออบเจ็กต์ที่อยู่ภายใน CMC นั้น เมื่อ Component Manager รับการร้องขอจากภายนอก Component Invoker จะทำการควบคุมการทำงานที่เกิดขึ้นบนออบเจ็กต์ รวมทั้งจัดการสถานะและการทำงานของออบเจ็กต์ ต่างๆ ภายใน CMC

3.2 วิธีการจัดการและควบคุมการโอนย้ายสถานะของออบเจ็กต์ภายในระบบ CMC

เมื่อนำคลาสของออบเจ็กต์ใหม่ (Cnew) เข้ามาทำงานแทนที่คลาสของออบเจ็กต์ที่กำลังทำงาน (Cold) ภายใน CMC กระบวนการอัพเกรดออบเจ็กต์ จะดำเนินการตามอัลกอริทึมดังรูปที่ 3.4

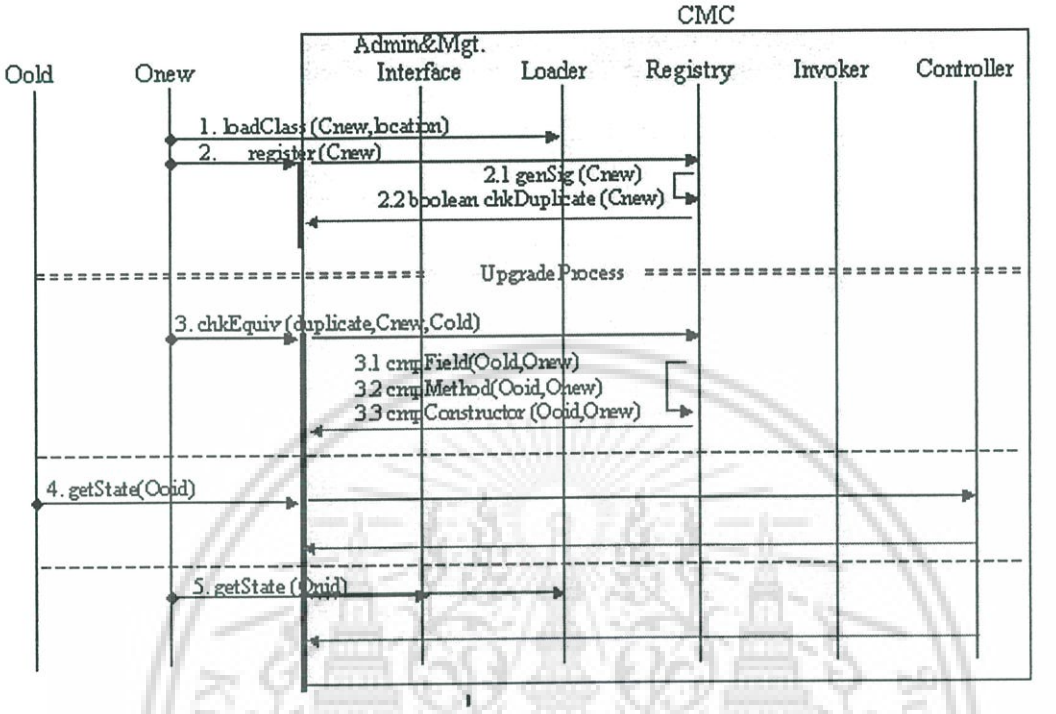
```
UpgradeObj(Cold,Cnew,LocCnew)
        LoadClass(Cnew,LocCew)                          // Step1
        Boolean chkEquiv(Cnew,Cold)                     // Step2
        If(chkEquiv = true) then
                StateCold getState(Cold)                // Step 3
                setState(Cnew,StateCold)
                destroy(Cold)
                Else
                        Unload(Cnew)
        End if
End
```

รูปที่ 3.4 อัลกอริทึมการอัพเกรดออบเจ็กต์ของ CMC

จากอัลกอริทึมสามารถแบ่งขั้นตอนการอัพเกรดออบเจ็กต์ ออกเป็น 3 ขั้นตอน คือ การโหลดคลาสใหม่เข้าสู่ระบบ CMC การตรวจสอบความเข้ากันได้ของคุณลักษณะพื้นฐานระหว่าง

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

คลาสเก่าและคลาสใหม่ การตรวจสอบและโอนย้ายสถานะของออบเจ็กต์เก่าไปยังออบเจ็กต์ใหม่ ดังรูปที่ 3.5



รูปที่ 3.5 กระบวนการอัพเกรดออบเจ็กต์

3.2.1 การโหลดคลาสใหม่เข้าสู่ระบบ CMC (Dynamic Class Loading)

เนื่องจาก Dynamic Class Loading เป็นคุณลักษณะที่สำคัญของ JVM ที่ช่วยให้แพลตฟอร์มต่างๆ ของจาวาสามารถติดตั้งคอมโพเนนต์ ได้อย่างทันทีทันใดในขณะที่อยู่ในสภาวะ runtime [10] ระบบ CMC จึงได้นำคุณสมบัตินี้มาใช้ในการทำงานเมื่อมีการอัพเกรดออบเจ็กต์ เพื่อทำหน้าที่กำหนดว่าคลาสต่าง ๆ จะถูกเพิ่มเข้าไปในสภาพแวดล้อมของจาวาได้เมื่อไร และอย่างไร โดยใช้ method ทำการค้นหาไบต์โค้ดหรือไบนารีไฟล์ (byte code) เพื่อส่งไปยัง JVM ทำการโหลดคลาสไฟล์และสร้างออบเจ็กต์ของคลาส

สำหรับคลาสโหลดเดอร์ที่นำมาใช้ใน CMC นั้นจะถูกโหลดโดย JVM ซึ่งสามารถโหลดคลาสแบบไดนามิก ได้ โดยมีขั้นตอนการโหลดคลาสดังอัลกอริทึม

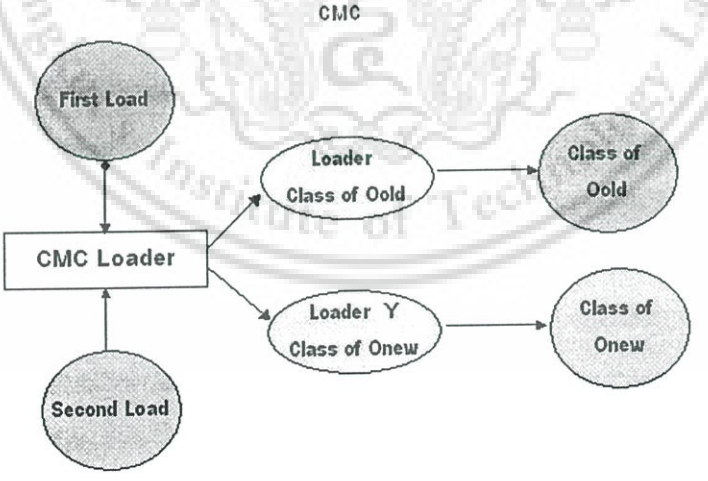
จากอัลกอริทึมในรูปที่ 3.6 ได้มีการระบุสถานที่เก็บคลาสไว้เป็นพารามิเตอร์เนื่องจากออบเจ็กต์ทั้งสองมีชื่อคลาสเหมือนกัน จึงต้องกำหนดสถานที่เก็บคลาสของออบเจ็กต์ใหม่ไว้แยกออกจากคลาสของออบเจ็กต์เก่า เพื่อไม่ให้คลาสใหม่ทับคลาสเก่า และเพื่อให้คลาสโหลดเดอร์สามารถโหลดคลาสชื่อเดียวกันได้ โดยคลาสใหม่ที่ถูกโหลดมาจะไม่ทับคลาสเดิมที่เคยโหลด

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
public CMCLoader(String dir) {..}
public Class loadClass(String nameClass)
    if (nameClass = loaded class)              //ตรวจสอบว่าเคยถูก load หรือไม่
            retum previous loaded class        // return class ที่เคยถูก load ก่อนหน้า
        else if (nameClass = system class)     //ตรวจสอบว่าเป็น system class หรือไม่
            load class from class path         //ติดต่อ Primordial Class Loader
        else
            read class file into array of byte          //เก็บข้อมูล class file
            construct class object and its method       // สร้าง object และ method
            specify class loader of nameClass           //ระบุ class loader ของ class นั้น
            load class from class loader                // load class จาก class loader ที่ระบุ
         end if
    end if
end
```

รูปที่ 3.6 อัลกอริทึมการ Load Class

โดยในการนำออบเจ็กต์ใหม่เข้ามาทำงานแทนที่ออบเจ็กต์เก่าที่กำลังทำงานอยู่ใน CMC นั้น จะต้องอาศัยกลไกของ Multiple Class Loaders ในการโหลดคลาสของออบเจ็กต์ใหม่และออบเจ็กต์เก่าด้วยคลาสโหลดเดอร์ที่แตกต่างกัน เพื่อให้คลาสของออบเจ็กต์ใหม่สามารถถูกโหลดเข้าสู่สภาพแวดล้อมเดียวกันกับคลาสของออบเจ็กต์เก่าได้แบบไดนามิก ดังรูปที่ 3.7

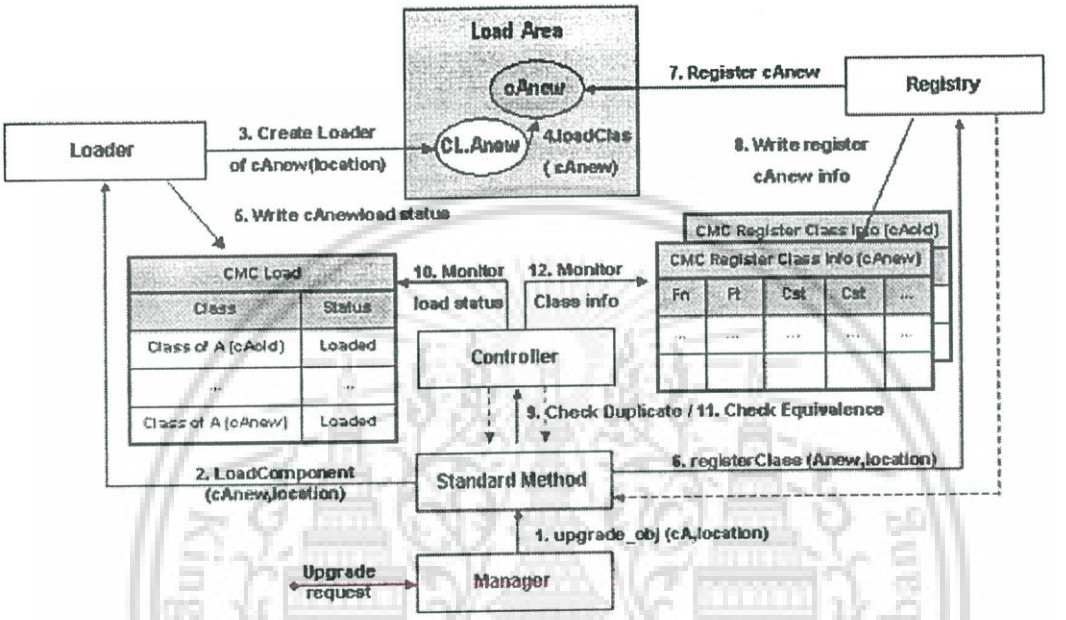


รูปที่ 3.7 การใช้ Multiple Class Loaders สำหรับโหลดคลาสของออบเจ็กต์เก่าและใหม่

สำหรับการโหลดคลาสของออบเจ็กต์ใหม่แบบไดนามิกนั้น เป็นการโหลดคลาสของออบเจ็กต์ใหม่เข้าสู่ JVM เพื่อนำเข้าสู่ CMC เป็นครั้งแรก โดยหลังจาก Component Loader (CMC

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Loader) ทำการโหลดคลาสแล้วจะบันทึกข้อมูลสถานะการโหลดคลาสไว้ และ Component Registry จะบันทึกข้อมูลรายละเอียดพื้นฐานของคลาสที่ถูกโหลดเก็บไว้เพื่อใช้ตรวจสอบความเข้ากันได้ เมื่อพบว่าชื่อของคลาสที่ถูกโหลดเข้ามาใหม่ซ้ำกับชื่อของคลาสที่เคยถูกโหลด นั่นหมายถึงมีการเตรียมการเปลี่ยนแปลงออบเจ็กต์ที่มีอยู่ ซึ่งจะเริ่มเข้าสู่กระบวนการตรวจสอบความเข้ากันได้ระหว่างคลาสทั้งสอง ดังรูปที่ 3.8



รูปที่ 3.8 การโหลดคลาสของออบเจ็กต์ ใหม่แบบไดนามิกเพื่อตรวจสอบความเข้ากันได้

3.2.2 การตรวจสอบความเข้ากันได้ (Check Equivalence)

เป็นการตรวจสอบคุณลักษณะพื้นฐาน (Object Behavior) ของ Oold (ออบเจ็กต์เก่า) และ Onew (ออบเจ็กต์ใหม่) ว่ามีความสอดคล้องกันหรือไม่ ก่อนจะนำ Onew เข้ามาทำงานแทนที่ Oold โดยจะตรวจสอบจากข้อมูลเกี่ยวกับ Field, Method และ Constructor ของคลาสเพื่อนำข้อมูลที่ได้จากการตรวจสอบจาก Component Registry มาทำการเปรียบเทียบว่า Oold และ Onew มีรายละเอียดพื้นฐานตรงกันหรือไม่ โดยจะตรวจสอบจาก

Field :
- ชื่อของฟิลด์ (Field name)
- ชนิดของฟิลด์ (Field type)

Method :
- ชื่อของ method (Method name)
- ชนิดของ method (Method modifier)
- ชนิดของค่าที่จะถูกส่งกลับโดย method (Return type)
- จำนวนและชนิดของพารามิเตอร์ (Parameter listing)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Constructor : - ชื่อของ Constructor (Constructor name)

- จำนวนและชนิดของพารามิเตอร์ (Parameter listing)

การตรวจสอบดังกล่าวจะดำเนินการตามอัลกอริทึมดังนี้

```
Boolean chkEquiv(Cnew,Cold)
        // เปรียบเทียบชื่อ และชนิดของการส่งค่ากลับของ field ระหว่าง Cnew และ Cold
        Boolean cmpField(fCnew[],fCold[])
        // เปรียบเทียบชื่อ ชนิด และชนิดพารามิเตอร์ของ method ระหว่าง Cnew และ Cold
        Boolean cmpMethod(mCnew[],mCold[])
        // เปรียบเทียบชื่อ และชนิดพารามิเตอร์ของ constructor ระหว่าง Cnew และ Cold
        Boolean cmpConstructor(cCnew[],cCold[])
end
```

รูปที่ 3.9 อัลกอริทึมการตรวจสอบรายละเอียดพื้นฐาน (Check Equivalence of Object Behavior)

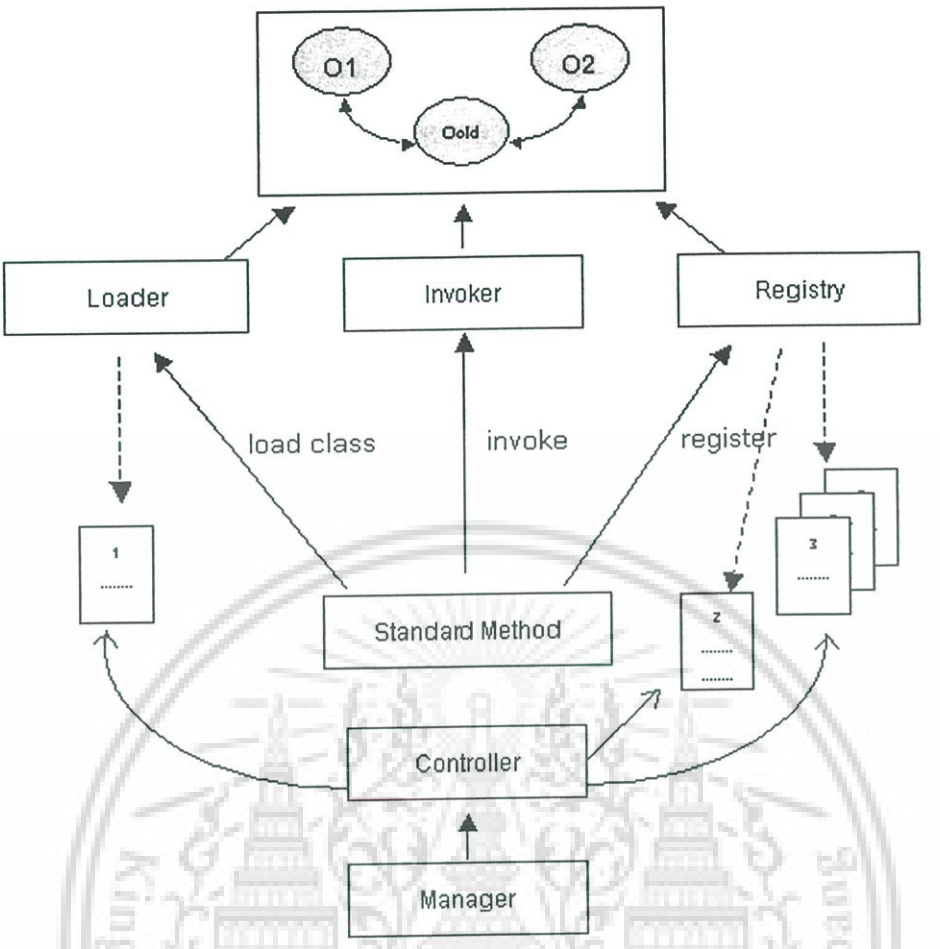
สำหรับการกำหนดความเข้ากันได้ระหว่างออบเจ็กต์ทั้งสองนั้นจะตั้งอยู่บนสมมติฐาน คือ

"ออบเจ็กต์ทั้งสองจะมีความเข้ากันได้ ก็ต่อเมื่อข้อมูลพื้นฐานของคลาสของออบเจ็กต์เก่าเป็นเซตย่อย (Sub set) ของข้อมูลพื้นฐานของคลาสของออบเจ็กต์ใหม่"

เมื่อพบว่าผลการตรวจสอบรายละเอียดของคลาสทั้งสองแล้วมีข้อมูลพื้นฐานตรงกัน แสดงว่าออบเจ็กต์ใหม่สามารถทำงานแทนที่ออบเจ็กต์ได้ โดยการทำงานจะยังคงถูกต้องภายหลังการเปลี่ยนแปลง

จากรูปที่ 3.10 แสดงการตรวจสอบข้อมูลต่างๆ ที่บันทึกลงในไฟล์ ซึ่งเกิดจากทำงานร่วมกันระหว่างคอมโพเนนต์ต่างๆ เมื่อมีการโหลดคลาสของออบเจ็กต์ใหม่เข้าสู่ระบบ CMC โดย CMC Standard Method จะติดต่อกับ Component Loader เพื่อทำการ Load class โดยบันทึกผลการ Load class ลงในไฟล์ (1) ส่วน Component Registry จะบันทึกผลการ Register class ลงในไฟล์ (2) และผลการตรวจสอบ Behavior ของแต่ละ ออบเจ็กต์ ลงในไฟล์ (3) ซึ่ง Component Controller จะสามารถ Monitor การทำงานของ CMC ได้จากข้อมูลภายในแต่ละไฟล์ ที่จะทำให้สามารถทราบได้ว่าสถานะของแต่ละออบเจ็กต์ในขณะนั้นเป็นอย่างไร สำหรับไฟล์ที่เกิดขึ้นจากการทำงานของ CMC มีดังนี้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 3.10 การตรวจสอบข้อมูลเมื่อมีการโหลดคลาสของออบเจ็กต์ใหม่เข้าสู่ระบบ CMC

1. cmc_statusLoad : เก็บข้อมูลสถานะการโหลดคลาสของแต่ละออบเจ็กต์ ใน cmc ถูกบันทึกโดย Component Loader

ตารางที่ 3.1 การเก็บข้อมูลสถานะการโหลดคลาสของแต่ละออบเจ็กต์ใน CMC

Object/Class Name	Status
t3	Loaded

2. cmc_statusRegister : เก็บข้อมูลสถานะการมีอยู่ของแต่ละออบเจ็กต์ใน cmc ถูกบันทึกโดย Component Registry โดยผลการเก็บข้อมูลจะคล้ายกับข้อมูลสถานะการโหลด

ตารางที่ 3.2 การเก็บข้อมูลสถานะการมีอยู่ของแต่ละออบเจ็กต์ใน CMC.

Object/Class Name	Status
t3	Registered

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3. cmc_classInfo : เก็บข้อมูลพื้นฐานของแต่ละคลาส ถูกบันทึกโดย Component Registry โดยข้อมูลนี้ประกอบด้วย ชื่อและชนิดของฟิลด์ (field name and type) ชื่อและชนิดการส่งค่ากลับของ Method (Method name and return type) ชนิดพารามิเตอร์ของ Method (Method parameter listing) ชื่อและชนิดพารามิเตอร์ของ Constructor (Constructor name and parameter listing)

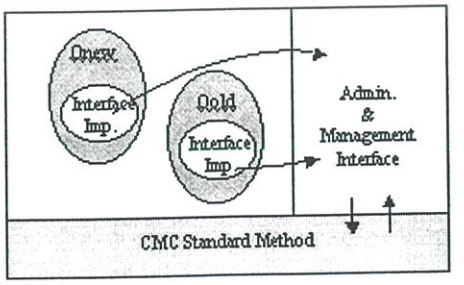
ตารางที่ 3.3 การเก็บข้อมูลพื้นฐานของแต่ละคลาส

field name	field type	method name	Method return type	method parameter	constructor name	constructor parameter
A	Int	main	Void	string	t3	Null

Component Controller จะทำหน้าที่เข้าไปอ่านข้อมูลจากไฟล์เหล่านี้เพื่อส่งข้อมูลที่ได้จากการอ่านกลับไปยัง Standard Method ของ CMC เพื่อเตรียมการโอนย้ายสถานะต่อไป

3.2.3 การตรวจสอบและโอนย้ายสถานะของออบเจ็กต์เก่าไปยังออบเจ็กต์ใหม่ (State Transfer)

เป็นการตรวจสอบสถานะการเรียกใช้และการทำงานของออบเจ็กต์เก่า ในลักษณะของการมอนิเตอร์ (Monitor) โดย CMC Standard Method จะติดต่อกับ Component Invoker เพื่อตรวจสอบสถานะของออบเจ็กต์ต่างๆ ภายใน CMC ผ่านอินเตอร์เฟส (Admin. & Management Interface) ดังรูปที่ 3.11 จะเห็นว่าระบบ CMC จำเป็นต้องอาศัยความร่วมมือจากแอพพลิเคชั่นที่จะนำมาทำงานภายใต้ระบบ CMC คือ คลาสหลักของแอพพลิเคชั่นจะต้องมีการอิมพลีเมนต์อินเตอร์เฟสนี้ เพื่อให้ CMC สามารถตรวจสอบสถานะของออบเจ็กต์ต่างๆ ในขณะที่ทำงานใน CMC ได้ รวมทั้งคลาสใหม่ที่จะนำมาทำงานแทนที่ก็จะต้องมีการอิมพลีเมนต์อินเตอร์เฟสนี้เช่นกัน เพื่อให้สามารถนำสถานะของออบเจ็กต์เก่าที่ได้จากการตรวจสอบมาใช้ในการปรับค่าสถานะของออบเจ็กต์ใหม่ก่อนที่ออบเจ็กต์ใหม่จะเริ่มต้นทำงานแทนที่ออบเจ็กต์เก่า

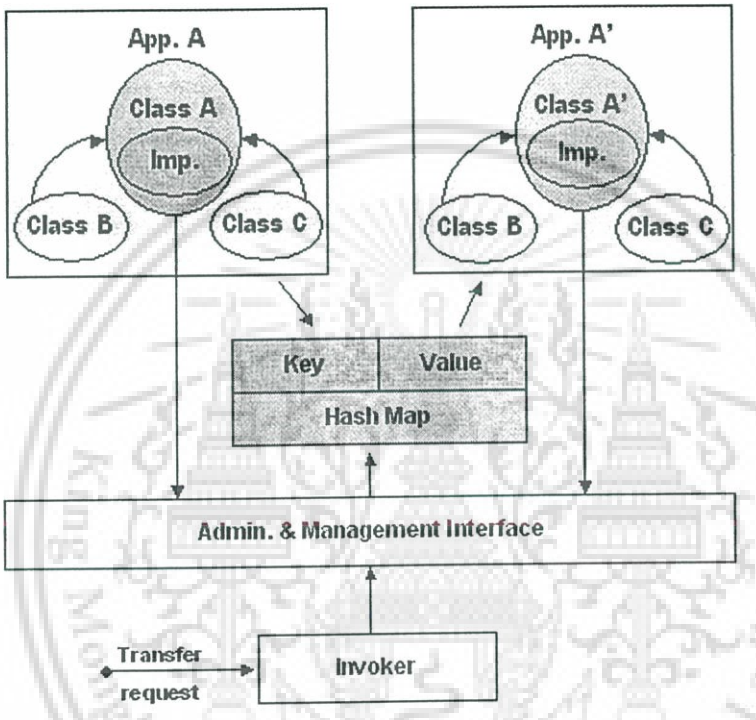


รูปที่ 3.11 การใช้อินเตอร์เฟสสำหรับติดต่อกับออบเจ็กต์ต่างๆ ภายในระบบ CMC

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3.2.3.1 การโอนย้ายสถานะของออบเจ็กต์ผ่านอินเตอร์เฟส

ในการตรวจสอบสถานะการทำงานของออบเจ็กต์ที่ทำงานในระบบ CMC นั้น จะมีการสร้างฟิลด์ชนิด Hash Map สำหรับเก็บข้อมูลสถานะการทำงานของแต่ละออบเจ็กต์ และมีการเพิ่มคำสั่ง (Statement) สำหรับบันทึกสถานะการทำงานแต่ละออบเจ็กต์ไว้ในคลาสหลักของแอพพลิเคชั่น ซึ่งจะทำให้ระบบ CMC สามารถทราบได้ว่า แต่ละออบเจ็กต์ในขณะนั้นมีสถานะการทำงานเป็นอย่างไร



รูปที่ 3.12 การโอนย้ายสถานะของออบเจ็กต์ต่างๆ ภายในระบบ CMC ผ่านอินเตอร์เฟส

ขั้นตอนในการโอนย้ายสถานะของออบเจ็กต์นั้น จะแสดงในรูปที่ 3.12 เมื่อผ่านการตรวจสอบความเข้ากันได้แล้ว Standard Method ของ CMC จะติดต่อไปยัง Component Invoker เพื่อทำหน้าที่ติดต่อไปยังแอพพลิเคชั่นที่กำลังทำงานในขณะนั้นผ่าน Admin.&Mgt. Interface โดย Component Invoker จะนำสถานะที่จัดเก็บไว้ใน Hash Map ซึ่งมีการแบ่งชนิดการเก็บข้อมูลออกเป็นชื่อของออบเจ็กต์ (Key) และสถานะของออบเจ็กต์ (Value) ออกมาและส่งสถานะนั้นไปยังออบเจ็กต์ของคลาสใหม่ที่จะนำมาทำงานแทนที่ โดยสถานะที่ถูกนำออกมานั้น จะถูกนำไปใช้เป็นค่าเริ่มต้นของสถานะของออบเจ็กต์ใหม่และเริ่มการทำงานของออบเจ็กต์ใหม่ด้วยสถานะนั้นทันที

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
public interface AdminMgtInterface{
        void printState(Map objMap);
        Map getState();
        void setState(Map recvObjMap);
        void stop();
}
```

รูปที่ 3.13 อินเตอร์เฟสที่ใช้ในการตรวจสอบและโอนย้ายสถานะการทำงานระหว่างออบเจ็กต์

สำหรับภายในอินเตอร์เฟส Admin. & Management Interface ดังรูปที่ 3.13 จะประกอบด้วย Method ที่เกี่ยวข้องกับการทำงานดังนี้

1. void printState(Map objMap) ใช้แสดงสถานะของแต่ละออบเจ็กต์ที่มีการโอนย้ายสถานะการทำงาน โดยการนำสถานะออกมาจากฟิลด์ที่เก็บข้อมูล
2. Map getState() ใช้นำสถานะการทำงานของแต่ละออบเจ็กต์ออกมาใช้งาน
3. void setState(Map recvObjMap) ใช้ตั้งค่าสถานะการทำงานของออบเจ็กต์ตามค่าสถานะการทำงานที่ได้รับมาจากฟิลด์ (recvObjMap)
4. void stop() ใช้หยุดการทำงานของออบเจ็กต์

3.2.3.2 การใช้ความร่วมมือจากแอพพลิเคชั่น

เนื่องจากอินเตอร์เฟสนี้จัดเป็นส่วนหนึ่งของระบบ CMC ที่ช่วยให้สามารถเข้าถึงการทำงานของแอพพลิเคชั่นที่ทำงานภายใต้ CMC ได้ โดยในการจัดการและควบคุมการโอนย้ายสถานะการทำงานของออบเจ็กต์ต่างๆ ของแอพพลิเคชั่นนั้น คลาสหลักของแอพพลิเคชั่นจำเป็นต้องให้ความร่วมมือในการเปลี่ยนแปลงที่จะเกิดขึ้น โดยสรุปดังนี้

1. เพิ่มฟิลด์สำหรับเก็บข้อมูลสถานะการทำงานของแต่ละออบเจ็กต์ โดยฟิลด์นี้มีชนิดเป็น Hash Map ที่จะเก็บชื่อ (Key) และสถานะของออบเจ็กต์ (Value) ที่จำเป็นต้องมีการโอนย้ายสถานะไปยังออบเจ็กต์ต่างๆ ของคลาสใหม่ที่นำมาทำงานแทนที่คลาสเก่า โดยจะประกาศฟิลด์นี้ไว้ที่ส่วนเริ่มต้นของคลาส

```
Private static Map objMap = new HashMap();
```

รูปที่ 3.14 การใช้ฟิลด์ประเภท Hash Map สำหรับเก็บข้อมูลสถานะการทำงานของออบเจ็กต์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2. เพิ่มคำสั่ง (Statement) สำหรับเก็บข้อมูลสานะการทำงานของแต่ละออบเจ็กต์ไว้ในฟิลด์ที่กำหนดขึ้น โดยจะบันทึกตามจำนวนออบเจ็กต์หรือฟิลด์ที่จะมีการโอนย้ายสถานะการทำงานของแต่ละออบเจ็กต์

```
public static void main(String[] args) {
        ...
        while(stopped = false){
                ...
                ObjMap.put (key1,value1);
                ObjMap.put (key2,value2);
                ObjMap.put (key3,value3);
        }
}
```

รูปที่ 3.15 การเพิ่มคำสั่งสำหรับเก็บข้อมูลสถานะการทำงานของออบเจ็กต์ไว้ในฟิลด์

3. อิมพลีเมนต์อินเตอร์เฟส Admin.&Mgt. Interface ของแอพพลิเคชั่นที่ใช้ในการติดต่อกับ Component Invoker ประกอบด้วย Method ดังนี้

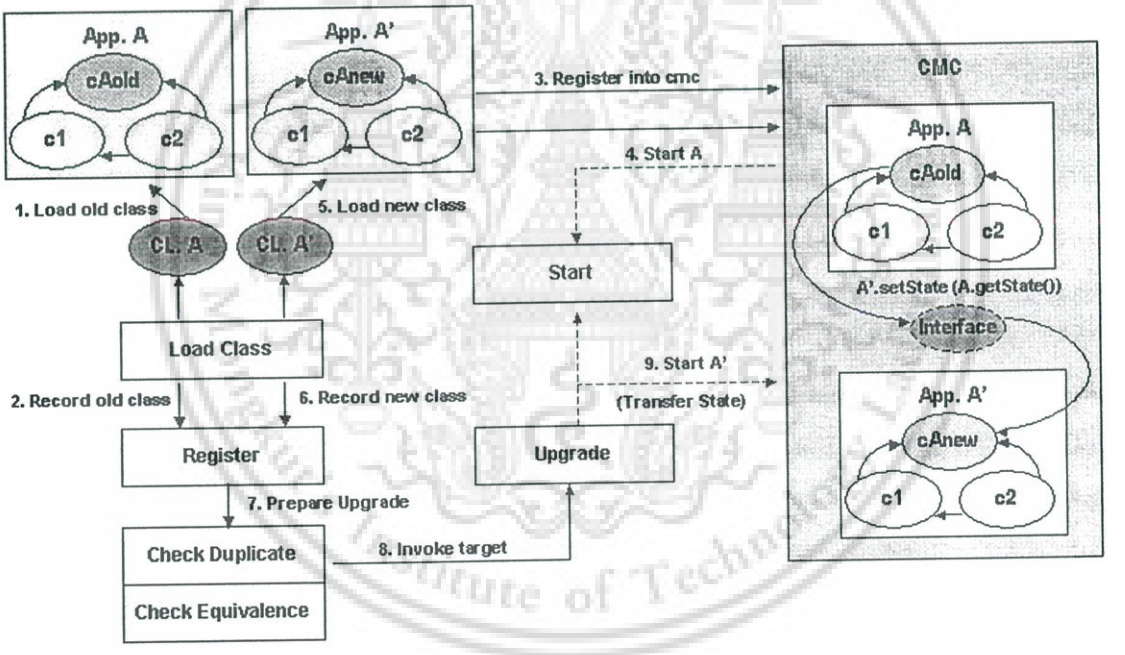
```
void printState(Map objMap) {
        objMap.get(key);
        print state;
}
Map getState() {
        return objMap;
}
void setState(Map recvObjMap){
        ObjMap.get (key1,value1);
        ObjMap.get (key2,value2);
}
void stop();
        stopped = true;
}
```

รูปที่ 3.16 การอิมพลีเมนต์อินเตอร์เฟสของแอพพลิเคชั่น

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3.3 สรุปภาพรวมของการจัดการและควบคุมการโอนย้ายสถานะของออบเจ็กต์ภายในระบบ CMC

จากการนำเสนอแนวความคิดสำหรับการจัดการและควบคุมการเปลี่ยนแปลงออบเจ็กต์ของแอพพลิเคชั่นในขณะที่แอพพลิเคชั่นกำลังทำงาน เพื่อให้แอพพลิเคชั่นสามารถทำงานได้อย่างต่อเนื่องในระหว่างที่มีการเปลี่ยนแปลงออบเจ็กต์ ภายใต้สภาพแวดล้อมสำหรับการทำงานของแอพพลิเคชั่น (Runtime environment) ที่เขียนด้วยภาษาจาวาในรูปของระบบ CMC ที่มีการนำกลไกของ Multiple Class Loaders มาใช้ในการโหลดคลาสที่มีชื่อเดียวกันเข้าสู่สภาพแวดล้อมเดียวกันได้แบบไดนามิกในขณะ runtime โดยสามารถโอนย้ายสถานะการทำงานของออบเจ็กต์ผ่านอินเตอร์เฟสที่ได้รับความร่วมมือจากแอพพลิเคชั่น ดังที่ได้กล่าวรายละเอียดของวิธีการดังกล่าว ในข้อ 3.1 และ 3.2 นั้น สามารถสรุปภาพรวมของวิธีการจัดการและควบคุมการโอนย้ายสถานะของออบเจ็กต์ภายในระบบ CMC ได้ดังรูปที่ 3.17



รูปที่ 3.17 ภาพรวมของการจัดการและควบคุมการโอนย้ายสถานะของออบเจ็กต์ภายใน CMC

จากรูปที่ 3.17 การทำงานทั้งหมดจะเริ่มต้นจาก Component Loader ทำการโหลดคลาส (Load Class) ทั้งหมดของแอพพลิเคชั่นเข้าสู่ JVM จากนั้นระบบ CMC จะติดต่อไปยัง Component Registry เพื่อบันทึกสถานะการโหลดและข้อมูลรายละเอียดพื้นฐานของแต่ละคลาส และบันทึกคลาสต่างๆ ไว้ในระบบ CMC (Register) ซึ่งระบบ CMC จะติดต่อไปยัง Component Invoker เพื่อติดต่อให้แอพพลิเคชั่นเริ่มต้นทำงานทันที เมื่อแอพพลิเคชั่นกำลังทำงาน Component Manager จะ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ติดต่อให้แอพพลิเคชั่นเริ่มต้นทำงานทันที เมื่อแอพพลิเคชั่นกำลังทำงาน Component Manager จะรับข้อมูลตำแหน่งที่อยู่และคลาสของออบเจ็กต์ใหม่เพื่อติดต่อไปยัง Component Loader ทำการโหลดคลาสใหม่เข้าสู่ JVM และบันทึกข้อมูลสถานะการโหลดและข้อมูลรายละเอียดพื้นฐานของคลาสใหม่นั้น เมื่อตรวจสอบสถานะการโหลดของคลาสใหม่ (Check Duplicate) พบว่ามีชื่อคลาสซ้ำกับคลาสที่เคยถูกโหลด จะเข้าสู่กระบวนการตรวจสอบความเข้ากันได้ของรายละเอียดพื้นฐานระหว่างคลาสเก่าและคลาสใหม่ (Check Equivalence) เมื่อพบว่ามีความเข้ากันได้จะติดต่อไปยัง Component Invoker เพื่อเข้าสู่กระบวนการเปลี่ยนแปลงออบเจ็กต์ (Upgrade) โดยจะติดต่อไปยังแอพพลิเคชั่นที่จะมีการเปลี่ยนแปลงผ่านอินเตอร์เฟส เพื่อนำสถานะของออบเจ็กต์เก่ามาใช้ตั้งค่าสถานะเริ่มต้นทำงานของออบเจ็กต์ใหม่ ซึ่งออบเจ็กต์ใหม่จะเริ่มทำงานด้วยสถานะที่ได้รับมาทันทีก่อนที่ออบเจ็กต์เก่าจะหยุดการทำงานหลังจากออบเจ็กต์ใหม่ได้ทำงานไปแล้วชั่วระยะเวลาหนึ่ง จึงเป็นการส่งผลให้แอพพลิเคชั่นสามารถทำงานได้อย่างต่อเนื่อง เนื่องจากออบเจ็กต์เก่าไม่ได้หยุดชะงักการทำงานใดๆ ก่อนที่ออบเจ็กต์ใหม่จะเริ่มทำงาน

จะเห็นได้ว่าแนวความคิดและรายละเอียดของวิธีการดังที่ได้กล่าวมาแล้วนั้น เป็นการจัดการและควบคุมการเปลี่ยนแปลงออบเจ็กต์ในอีกรูปแบบหนึ่ง ที่สามารถช่วยให้การทำงานของแอพพลิเคชั่นยังคงดำเนินไปได้อย่างต่อเนื่อง และเนื่องจากมีการตรวจสอบความเข้ากันได้ของข้อมูลรายละเอียดพื้นฐานระหว่างออบเจ็กต์เก่าและออบเจ็กต์ใหม่ ก่อนที่จะเข้าสู่กระบวนการเปลี่ยนแปลง จึงทำให้มั่นใจได้ในระดับหนึ่งว่าภายหลังการเปลี่ยนแปลงที่เกิดขึ้น ระบบการทำงานจะสามารถทำงานได้อย่างถูกต้องต่อไป

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 4
ผลการทดลองโอนย้ายสถานะของออบเจ็กต์ ภายในระบบรักษาความต่อเนื่องในการทำงาน

ในการจัดทำระบบ CMC ขึ้นเพื่อใช้เป็นสภาพแวดล้อมที่สนับสนุนการเปลี่ยนแปลงแอพพลิเคชั่นแบบอัพเกรดออบเจ็กต์นั้น วิทยานิพนธ์นี้ได้มีการจัดทำระบบ CMC และแอพพลิเคชั่นที่ใช้ในการทดลองด้วย Java 2 SDK 1.5.1_01 ภายใต้สภาพแวดล้อม JRE 2 SDK 1.5.1_01 โดยในการทดลองเปลี่ยนแปลงแอพพลิเคชั่นภายใต้ระบบ CMC นั้น จะทดลองกับเครื่องที่มีการทำงานแบบ Standalone บนระบบปฏิบัติการ Windows XP และ Windows 2000 Professional ซึ่งได้จัดให้คลาสต่างๆ ของแอพพลิเคชั่นที่จะนำเข้าสู่การทำงานของระบบ CMC (คลาสเก่า) และคลาสที่จะนำมาอัพเกรดหรือนำเข้ามาทำงานแทนที่ (คลาสใหม่) อยู่ในตำแหน่งโฟล์เดอร์หรือไดเรกทอรีต่างกัน เนื่องจากการจัดตำแหน่งที่อยู่ให้คลาสเก่าและคลาสใหม่ แยกออกจากกัน จะทำให้คลาสโหลดเดอร์สามารถโหลดคลาสที่มีชื่อเดียวกันเข้ามาทำงานในระบบ CMC ได้โดยไม่เกิดปัญหาคลาสทับซ้อนกันได้ ทั้งนี้สามารถดูรายละเอียดของแอพพลิเคชั่นที่ใช้ในการทดลองได้ในภาคผนวก ก

4.1 ภาพรวมของการทดลองโอนย้ายสถานะของออบเจ็กต์

เป็นการทดลองนำแอพพลิเคชั่นหรือโปรแกรม “AppDemo2.java” ที่ประกอบด้วยคลาสต่างๆ ที่มีการทำงานร่วมกันจำนวน 5 คลาสเข้าสู่ระบบ CMC โดยจะทดลองเปลี่ยนแปลงคลาสหลัก หรือคลาสที่มี Method main ของแอพพลิเคชั่น ซึ่งแอพพลิเคชั่นที่จะนำมาใช้ในการทดลองนี้จะมีการโอนย้ายสถานะของออบเจ็กต์ทั้งหมด 3 ออบเจ็กต์ สำหรับคลาสต่างๆ ของแอพพลิเคชั่นที่เป็นคลาสเก่านั้นจะถูกจัดเก็บไว้ในไดเรกทอรี “app2” ส่วนคลาสใหม่ที่จะนำมาทำงานแทนที่คลาสเก่านั้นจะจัดเก็บไว้ในไดเรกทอรี “app2-1” โดยภาพรวมของการทดลองจะประกอบด้วยขั้นตอนดังนี้

1. การโหลดแอพพลิเคชั่นของคลาสเก่า เข้าสู่ JVM โดยระบบ CMC จะตรวจสอบและบันทึกผลการโหลดคลาส และข้อมูลรายละเอียดพื้นฐานของคลาสต่างๆ เอาไว้เพื่อใช้ในการตรวจสอบความเข้ากันได้ระหว่างคลาสเก่าและคลาสใหม่
2. แอพพลิเคชั่นของคลาสเก่าที่ถูกโหลดเข้าสู่ JVM จะเริ่มต้นทำงานภายใต้การควบคุมของระบบ CMC

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3. ในขณะที่แอพพลิเคชั่นกำลังทำงานอยู่นั้น จะทดลองโหลดคลาสใหม่ที่ต้องการนำมาทำงานแทนที่เข้าสู่ JVM ซึ่งจะมีการตรวจสอบและบันทึกผลการโหลดคลาสและข้อมูลรายละเอียดพื้นฐานเช่นเดียวกัน

4. เมื่อระบบ CMC จะตรวจสอบพบว่าคลาสใหม่ที่ถูกโหลดมีชื่อซ้ำกับคลาสเก่าที่เคยถูกโหลดแล้วจะเข้าสู่กระบวนการตรวจสอบความเข้ากันได้ระหว่างคลาสเก่าและคลาสใหม่

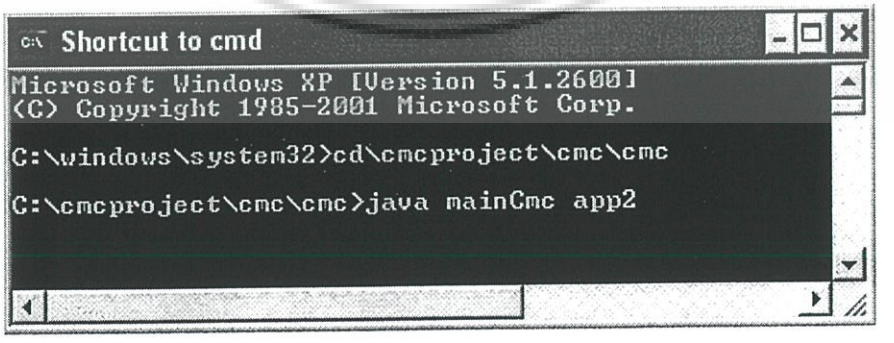
5. เมื่อตรวจสอบพบว่าคลาสใหม่สามารถทำงานแทนที่คลาสเก่าได้ จะเริ่มตรวจสอบสถานะปัจจุบันของคลาสเก่าและโอนย้ายสถานะนั้นไปยังคลาสใหม่ เพื่อให้คลาสใหม่สามารถเริ่มทำงานต่อจากคลาสเก่าได้อย่างต่อเนื่อง

4.2 การตรวจสอบผลการโหลดคลาสเข้าสู่ JVM

ในการทดลองนี้ระบบ CMC จะใช้ Component Loader ทำการโหลดคลาสต่างๆ ของแอพพลิเคชั่นเข้าสู่ JVM โดยใช้หลักการของ Multiple Class Loaders เพื่อให้สามารถโหลดคลาสชื่อเดียวกันภายใต้สภาพแวดล้อมเดียวกันในขณะ Runtime ได้ ซึ่งในการทดลองนี้จะมีการโหลดคลาส 2 ครั้ง คือ ครั้งแรกจะเป็นการโหลดแอพพลิเคชั่น (คลาสเก่า) สำหรับทำงานภายใต้การควบคุมของระบบ CMC เมื่อแอพพลิเคชั่นเริ่มทำงาน Component Loader จะทำการโหลดคลาสครั้งที่สองซึ่งเป็นคลาสใหม่ที่ต้องการนำมาทำงานแทนที่ โดยการโหลดคลาสแต่ละครั้งจะใช้คลาสโหลดเดอร์ที่แตกต่างกัน

4.2.1 การโหลดแอพพลิเคชั่นเข้าสู่ JVM

เมื่อนำแอพพลิเคชั่นเข้าสู่ระบบ CMC ขั้นตอนแรกเป็นการโหลดคลาสต่างๆ ของแอพพลิเคชั่นนั้นเข้าสู่ JVM โดยในการโหลดคลาสนั้น จำเป็นต้องระบุสถานที่เก็บหรือตำแหน่งที่อยู่ของแอพพลิเคชั่นเพื่อให้ Component Loader ของ CMC ทำการค้นหาและโหลดคลาสตามที่ระบุ ดังรูปที่ 4.1



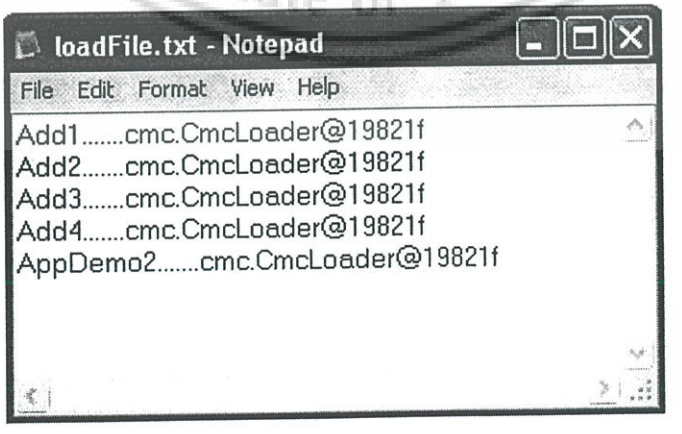
รูปที่ 4.1 การระบุตำแหน่งที่อยู่ของแอพพลิเคชั่นที่ต้องการนำเข้าสู่การทำงานของระบบ CMC

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

จากรูปที่ 4.1 เป็นการป้อนชื่อตำแหน่งที่อยู่ของแอพพลิเคชั่นผ่าน Command line โดยจะระบุว่าแอพพลิเคชั่นที่ต้องการนำเข้าสู่ระบบ CMC อยู่ในโฟลเดอร์ "app2" ซึ่งระบบ CMC จะติดต่อไปยัง Component Loader เพื่อค้นหาคลาสต่างๆ ที่อยู่ในโฟลเดอร์นี้ เพื่อทำการโหลดคลาสทั้งหมดเข้าสู่ JVM แล้วแสดงผลการโหลดคลาสและคลาสโหลดเดอร์ที่ทำหน้าที่โหลดแต่ละคลาสเหล่านั้น ดังรูปที่ 4.2

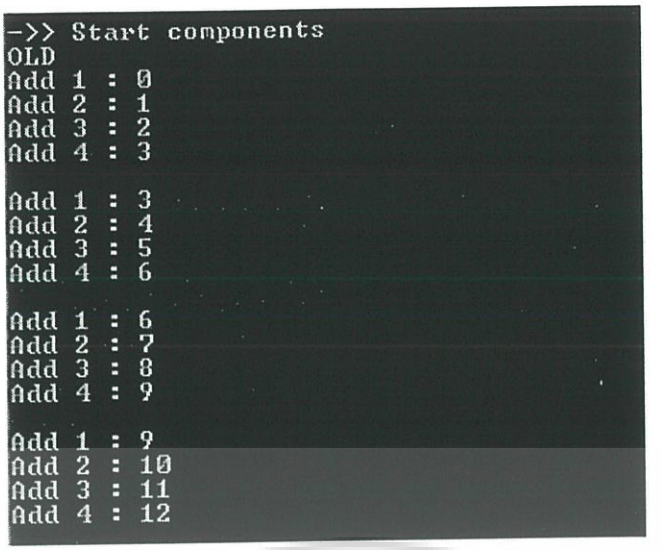
รูปที่ 4.2 ผลการโหลดคลาสต่างๆ ของแอพพลิเคชั่นเข้าสู่ JVM

จากรูปที่ 4.2 จะเห็นว่าคลาสต่างๆ จะถูกโหลดโดยคลาสโหลดเดอร์เดียวกันทั้งหมด เนื่องจากเป็นการโหลดคลาสเหล่านี้พร้อมกันในครั้งแรก เมื่อคลาสต่างๆ ถูกโหลดเข้าสู่ JVM แล้ว ระบบ CMC จะบันทึกข้อมูลรายชื่อคลาสที่ถูกโหลดไว้ในไฟล์ข้อความ (Text file) ชื่อ "loadFile.txt" ดังรูปที่ 4.3 เพื่อนำไปใช้ตรวจสอบเมื่อมีการนำคลาสใหม่เข้าสู่ระบบว่ามีคลาสใดที่ถูกโหลดเข้ามาซ้ำกับคลาสที่เคยถูกโหลดไปแล้ว



รูปที่ 4.3 ผลการบันทึกข้อมูลรายชื่อคลาสต่างๆ ของแอพพลิเคชั่นที่ถูกโหลดเข้าสู่ JVM

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



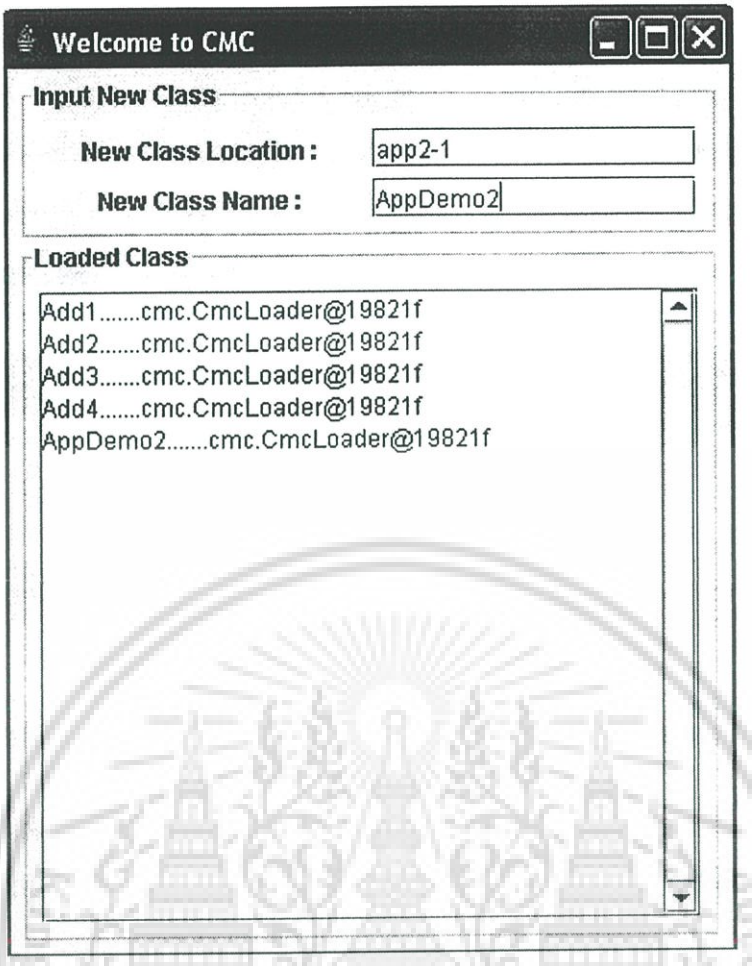
รูปที่ 4.4 ผลการทำงานของแอพพลิเคชั่นหลังจากถูกโหลดเข้าสู่ JVM

เมื่อแอพพลิเคชั่นถูกโหลดเข้าสู่ JVM แล้วแอพพลิเคชั่นนั้นจะเริ่มต้นทำงานภายใต้การจัดการและควบคุมการทำงานของคอมโพเนนต์ต่างๆ ภายในระบบ CMC ให้แอพพลิเคชั่นสามารถทำงานได้อย่างถูกต้องดังรูปที่ 4.4 แสดงการทำงานของแอพพลิเคชั่น “AppDemo2” ภายใต้ระบบ CMC โดยแอพพลิเคชั่นที่นำมาใช้ในการทดลองนี้เป็นแอพพลิเคชั่นที่ใช้ในการคำนวณตัวเลขที่ใช้เวลาในการทำงานค่อนข้างยาวนาน เพื่อให้มีสภาพการทำงานใกล้เคียงกับแอพพลิเคชั่นที่ต้องมีการทำงานอยู่ตลอดเวลามากที่สุด

4.2.2 การโหลดคลาสใหม่ที่จะนำมาทำงานแทนที่เข้าสู่ JVM

เมื่อแอพพลิเคชั่นเริ่มทำงานภายใต้ระบบ CMC แล้วจะติดต่อไปยัง Component Manager เพื่อทำหน้าที่ติดต่อกับผู้ใช้ในการนำคลาสที่เป็นเวอร์ชั่นใหม่หรือคลาสใหม่โหลดเข้าสู่ JVM ซึ่งจำเป็นต้องระบุตำแหน่งที่อยู่ของคลาสใหม่ก่อน คือ “app2-1” จากนั้นจะเป็นการระบุชื่อคลาสใหม่ที่ต้องการนำมาทำงานแทนที่คลาสเก่า ในการทดลองนี้จะเป็นการอัพเกรดคลาสหลักของแอพพลิเคชั่น คือ คลาส “AppDemo2” ดังรูปที่ 4.5 เป็นการระบุตำแหน่งที่อยู่และชื่อของคลาสใหม่ที่ต้องการนำเข้ามาทำงานแทนที่คลาสเดิมที่มีอยู่ โดยจะมีส่วนที่แสดงคลาสต่างๆ ที่เคยถูกโหลดเข้าสู่ระบบแล้ว

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 4.5 การนำคลาสใหม่เข้าสู่ JVM

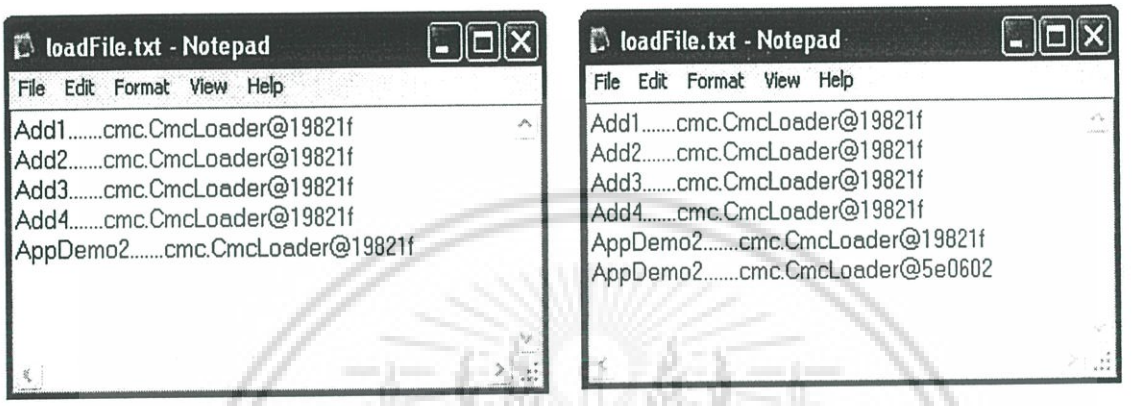
เมื่อระบุตำแหน่งที่อยู่และชื่อของคลาสใหม่แล้ว Component Manager จะติดต่อไปยัง Component Loader เพื่อทำการโหลดคลาสใหม่เข้าสู่ JVM ดังรูปที่ 4.6 แสดงผลการโหลดคลาสใหม่เข้าสู่ JVM และบันทึกข้อมูลการโหลดคลาสใหม่ลงใน loadFile.txt อีกครั้ง จากนั้น Component Registry จะบันทึกข้อมูลรายละเอียดพื้นฐานของคลาสใหม่เพื่อใช้ในการตรวจสอบความเข้ากันได้ต่อไป

```
==> STEP 1 : Load New Class .......
class AppDemo2...Loaded
loaded by : cmc.CmcLoader@5e0602
```

รูปที่ 4.6 ผลการโหลดคลาสใหม่เข้าสู่ JVM

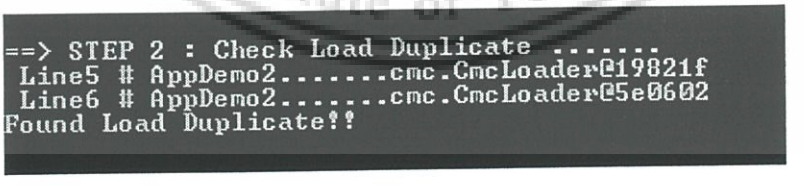
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

เมื่อบันทึกข้อมูลรายละเอียดพื้นฐานของคลาสใหม่แล้วระบบ CMC จะติดต่อไปยัง Component Controller เพื่อทำการตรวจสอบข้อมูลการโหลดคลาสใน loadFile.txt ดังรูป 4.7 ซึ่งจะมีการบันทึกชื่อคลาสทุกครั้งที่ถูกโหลดเข้าสู่ JVM หากพบว่าคลาสที่ถูกโหลดเข้าไปใหม่นี้มีชื่อซ้ำกับคลาสที่เคยถูกโหลด แสดงว่าคลาสใหม่ที่ถูกโหลดเข้ามานี้จะถูกตั้งสมมติฐานว่าเป็นคลาสที่จะนำเข้ามาทำงานแทนที่คลาสที่มีอยู่ ซึ่งจะแสดงผลการตรวจสอบดังรูปที่ 4.8



รูปที่ 4.7 ข้อมูลผลการโหลดคลาสต่างๆ ของแอพพลิเคชั่น (ซ้าย) เปรียบเทียบกับข้อมูลผลการโหลดคลาสใหม่เข้าสู่ JVM (ขวา)

จากการตรวจสอบผลการโหลดคลาสใหม่เข้าสู่ JVM พบว่าคลาสใหม่ที่ถูกโหลดเข้ามาคือ AppDemo2 มีชื่อคลาสซ้ำกับคลาสที่เคยถูกโหลดก่อนหน้านี้ ซึ่งหมายถึงการเข้าสู่กระบวนการเตรียมการเปลี่ยนแปลงออบเจ็กต์ของคลาส AppDemo2 ภายในแอพพลิเคชั่นที่กำลังทำงาน คือ การตรวจสอบความเข้ากันได้ระหว่างคลาสใหม่ที่ถูกโหลดเข้ามากับคลาสเก่าที่กำลังทำงานอยู่ในแอพพลิเคชั่นนั้น



รูปที่ 4.8 ผลการตรวจสอบการโหลดคลาสใหม่เข้าสู่ JVM

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

4.3 การตรวจสอบความเข้ากันได้ระหว่างคลาสที่มีการเปลี่ยนแปลง

เมื่อคลาสใหม่ถูกโหลดเข้าสู่ JVM แล้ว ระบบ CMC ตรวจสอบพบว่าคลาสใหม่ที่ถูกโหลดมีชื่อซ้ำกับคลาสที่เคยถูกโหลดก่อนหน้านี้ จะเข้าสู่ขั้นตอนการตรวจสอบและเปรียบเทียบข้อมูลรายละเอียดพื้นฐานของคลาสใหม่และคลาสเก่าว่ามีข้อมูลตรงกันหรือต่างกันหรือไม่

4.3.1 ข้อมูลรายละเอียดพื้นฐานของคลาสสำหรับทำการตรวจสอบ

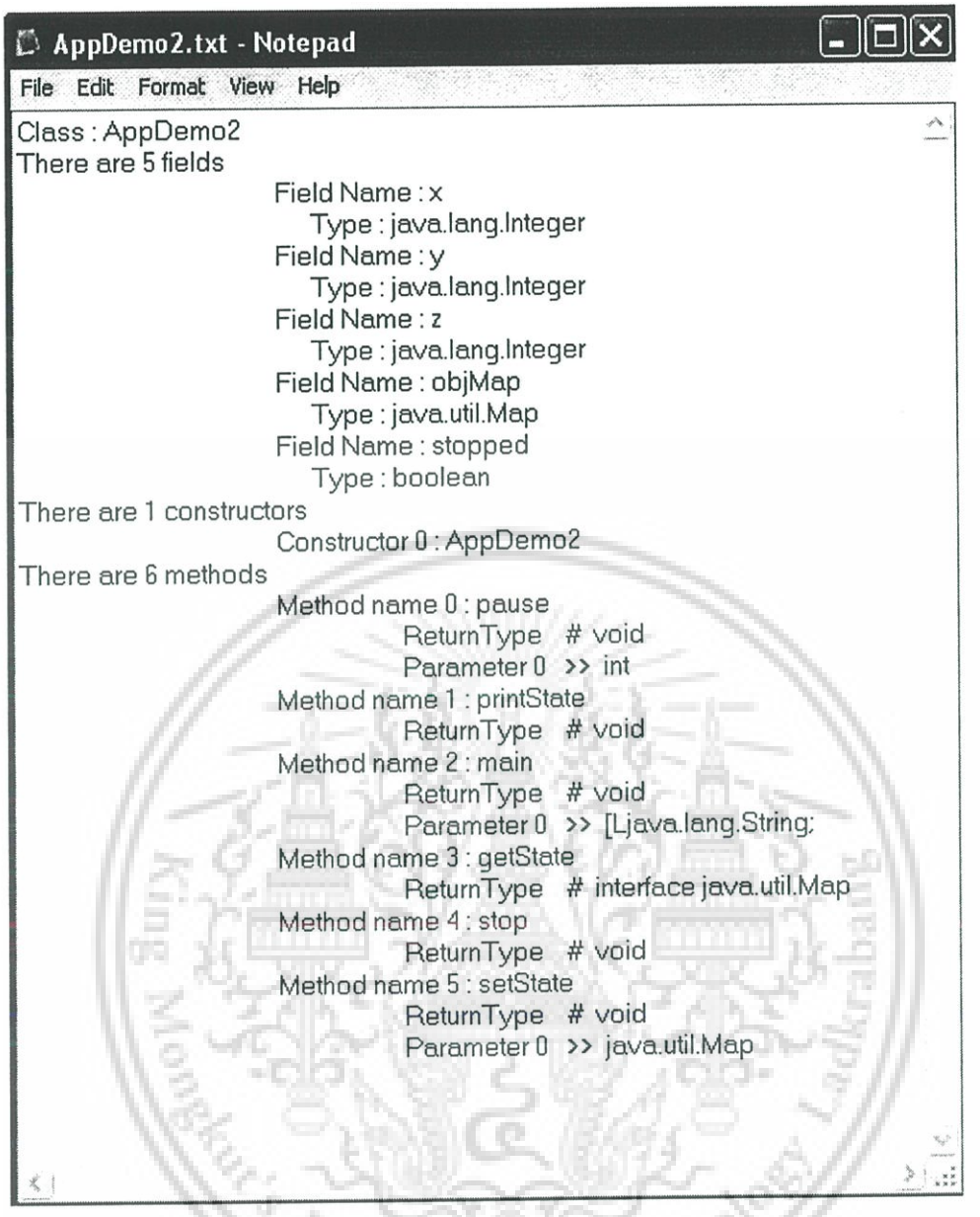
เป็นการตรวจสอบคุณลักษณะพื้นฐาน (Object Behavior) ของคลาสเก่าและคลาสใหม่ว่ามีความสอดคล้องกันหรือไม่ ก่อนจะนำคลาสใหม่เข้ามาทำงานแทนที่ โดยจะตรวจสอบจากข้อมูลเกี่ยวกับ Field, Method และ Constructor ของคลาสที่ได้บันทึกไว้หลังจากโหลดคลาส ซึ่ง Component Registry จะตรวจสอบและบันทึกข้อมูลเหล่านี้ไว้ในไฟล์ข้อความ โดยชื่อของไฟล์จะถูกตั้งตามชื่อของคลาสนั้นๆ แยกออกจากกัน โดยจะตรวจสอบจาก

Field :
- ชื่อของฟิลด์ (Field name)
- ชนิดของฟิลด์ (Field type)

Method :
- ชื่อของ method (Method name)
- ชนิดของ method (Method modifier)
- ชนิดของค่าที่จะถูกส่งกลับโดย method (Return type)
- จำนวนและชนิดของพารามิเตอร์ (Parameter listing)

Constructor :
- ชื่อของ Constructor (Constructor name)
- จำนวนและชนิดของพารามิเตอร์ (Parameter listing)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 4.9 ข้อมูลรายละเอียดพื้นฐานของคลาสเก่าที่จะถูกนำมาใช้ตรวจสอบ

จากรูปที่ 4.9 เป็นการแสดงข้อมูลรายละเอียดพื้นฐานของคลาส “AppDemo2” ซึ่งเป็นคลาสเก่าที่จะถูกนำมาใช้ในการตรวจสอบ ซึ่งข้อมูลนี้จะถูกบันทึกไว้ในไฟล์ข้อความที่มีชื่อเดียวกับชื่อของคลาสคือ “AppDemo2.txt”

สำหรับคลาสที่จะนำมาทำงานแทนที่ซึ่งมีชื่อเดียวกับคลาสเก่านั้น Component Registry จะบันทึกข้อมูลรายละเอียดพื้นฐานไว้ในไฟล์ข้อความที่มีชื่อเดียวกับชื่อคลาสเช่นกัน แต่จะเพิ่มชื่อไฟล์ต่อท้ายจากเดิมด้วยคำว่า “Upgrade” เพื่อไม่ให้ไฟล์ข้อความซ้อนทับกัน ข้อมูลรายละเอียดพื้นฐานของคลาสใหม่จึงถูกบันทึกไว้ในไฟล์ข้อความที่มีชื่อว่า “AppDemo2Upgrade.txt” ซึ่งจะทำให้สามารถตรวจสอบข้อมูลได้จากไฟล์ทั้งสองได้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

4.3.2 การเปรียบเทียบข้อมูลรายละเอียดพื้นฐานเพื่อตรวจสอบความเข้ากันได้ระหว่างคลาสที่มีการเปลี่ยนแปลง

ในการตรวจสอบความเข้ากันได้ระหว่างคลาสใหม่ที่นำเข้าสู่ระบบ CMC และคลาสที่กำลังทำงานนั้น (คลาสเก่า) จะตรวจสอบจากข้อมูลรายละเอียดพื้นฐานของแต่ละคลาสที่ได้จัดเก็บเป็นไฟล์ไว้ ดังรูปที่ 4.9 ซึ่งการตรวจสอบนั้นจะอยู่บนสมมุติฐานที่ว่า

"คลาสใหม่จะมีความเข้ากันได้กับคลาสเก่า ก็ต่อเมื่อคลาสใหม่มีรายละเอียดพื้นฐานที่เป็นเซตย่อย (Sub set) ของคลาสเก่า"

จากสมมุติฐานดังกล่าว จึงสามารถแบ่งผลการตรวจสอบออกเป็น 2 กรณี คือ

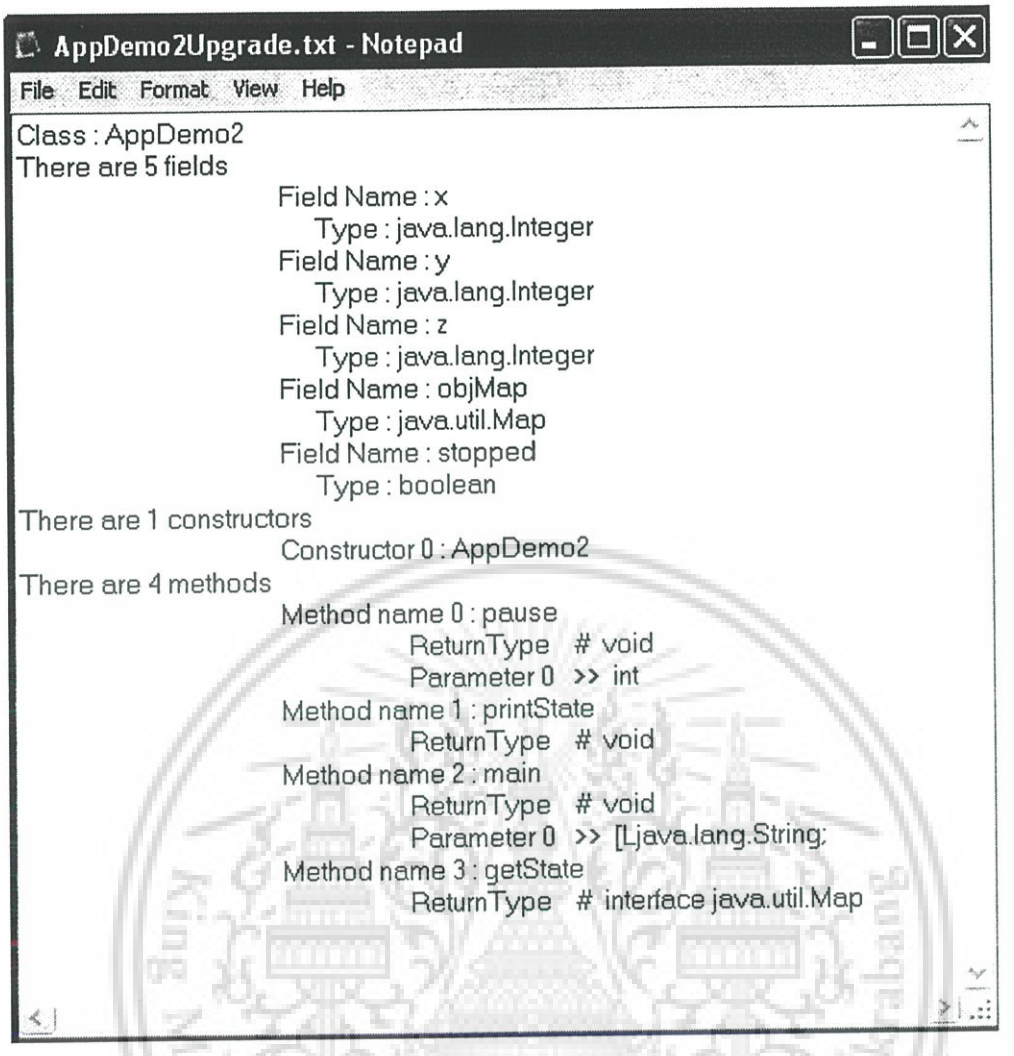
1. กรณีที่คลาสใหม่ไม่สามารถทำงานแทนที่คลาสเก่าได้

เป็นกรณีที่ตรวจพบว่าข้อมูลฟิลด์ หรือ Method หรือ Constructor ของคลาสเก่าไม่มีอยู่ในคลาสใหม่ หรือคลาสใหม่ไม่มีฟิลด์ หรือ Method หรือ Constructor ใดๆ ของคลาสเก่า นั่นหมายถึงคลาสใหม่มีรายละเอียดพื้นฐานไม่สมบูรณ์เพียงพอที่จะทำงานแทนคลาสเก่าได้ ระบบ CMC จะไม่ยินยอมให้คลาสใหม่เข้ามาทำงานแทนที่คลาสเก่า และคลาสใหม่นี้จะถูกลบหรือทำลายออกไปจากระบบทันที

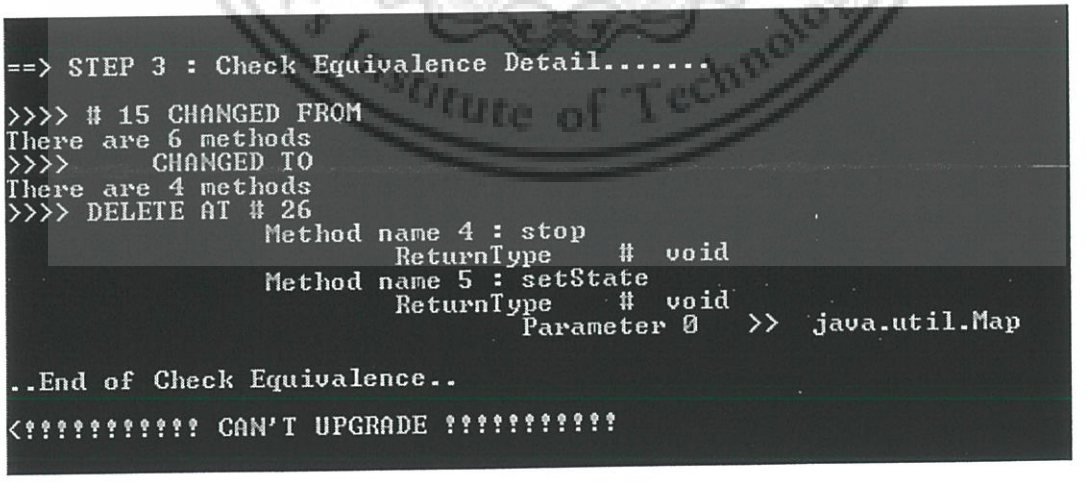
จากรูปที่ 4.10 เป็นการทดลองนำคลาสใหม่ที่ถูกลด Method ลงจำนวน 2 Method เพื่อให้มีความแตกต่างจากคลาสเก่าในรูปที่ 4.9 โดยสิ่งที่ส่งผลให้คลาสใหม่ไม่สามารถทำงานแทนที่คลาสเก่าได้ คือ Method บางตัวของคลาสเก่าในรูปที่ 4.9 ไม่มีในคลาสใหม่ ซึ่งหากนำคลาสใหม่นี้เข้ามาทำงานแทนที่อาจจะส่งผลให้การทำงานมีความเสียหายได้ เนื่องจากคลาสใหม่มีรายละเอียดพื้นฐานไม่สมบูรณ์เพียงพอ

ส่วนในรูปที่ 4.11 เมื่อเปรียบเทียบข้อมูลรายละเอียดพื้นฐานระหว่างคลาสใหม่และคลาสเก่าแล้วจะพบว่าคลาสใหม่มี Method ที่หายไป คือ ไม่มี Method "stop" และ "setState" ซึ่งทั้งสอง Method นี้มีอยู่ในคลาสเก่า จึงทำให้คลาสใหม่มีคุณสมบัติไม่เพียงพอที่จะทำงานแทนที่คลาสเก่าได้ ระบบ CMC จะแสดงผลการตรวจสอบว่าคลาสใหม่ที่ถูกโหลดเข้ามานั้นไม่สามารถทำงานแทนที่คลาสเก่าได้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 4.10 ข้อมูลรายละเอียดพื้นฐานของคลาสใหม่กรณีข้อมูลไม่ตรงกันกับคลาสเก่า



รูปที่ 4.11 ผลการตรวจสอบกรณีคลาสทั้งสองไม่สามารถนำมาทำงานแทนกันได้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2. กรณีที่คลาสใหม่สามารถทำงานแทนที่คลาสเก่าได้

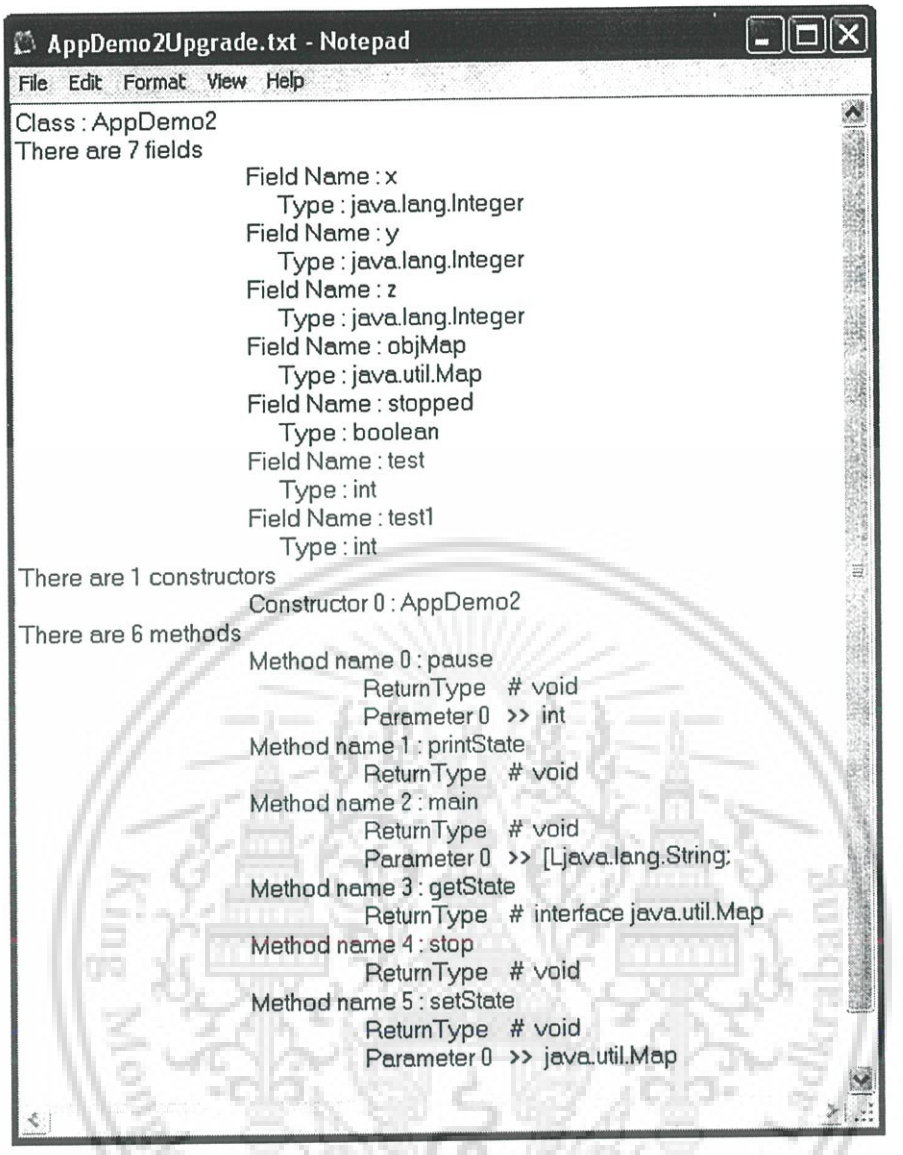
กรณีที่คลาสใหม่มีรายละเอียดของฟิลด์ Method และ Constructor ตรงกับคลาสเก่าหรืออาจมีรายละเอียดเพิ่มจากเดิมโดยไม่มีข้อมูลใดๆ ของคลาสเก่าสูญหายไป ดังรูปที่ 4.12 จะถือว่าคลาสใหม่มีความเข้ากันได้กับคลาสเก่า ระบบ CMC จะยินยอมให้สามารถทำการอัพเกรดหรือเปลี่ยนแปลงออบเจ็กต์ได้ ดังรูปที่ 4.13 ซึ่งจะเข้าสู่กระบวนการตรวจสอบและโอนย้ายสถานะการทำงานต่อไป

จากรูปที่ 4.12 เป็นการทดลองนำคลาสใหม่ที่ถูกเพิ่มฟิลด์จำนวน 2 ฟิลด์ และเพิ่ม Constructor จำนวน 1 Constructor เข้าไป เมื่อเปรียบเทียบรายละเอียดพื้นฐานของคลาสเก่าในรูปที่ 4.9 แล้วจะเห็นว่าคลาสใหม่มีข้อมูลตรงกันกับคลาสเก่า ไม่มีข้อมูลใดๆ ของคลาสเก่าที่สูญหายไปจากสมมุติฐานที่ตั้งไว้ จึงพบว่าคลาสใหม่นี้สามารถนำมาทำงานแทนที่คลาสเก่าได้

ในรูปที่ 4.12 คลาสใหม่นี้ต่างจากคลาสเก่าในรูปที่ 4.9 ตรงที่คลาสใหม่ได้เพิ่มฟิลด์ “Test1” และ “Test2” และเพิ่ม Constructor “ProgressBarDemo” ที่มีพารามิเตอร์ ในส่วนของ Method นั้นมีความตรงกันทั้งหมด

ส่วนในรูปที่ 4.13 แสดงผลการตรวจสอบรายละเอียดพื้นฐานระหว่างคลาสที่จะมีการเปลี่ยนแปลง ซึ่งจะเห็นได้ว่าคลาสใหม่สามารถทำงานแทนที่คลาสเก่าได้ เนื่องจากรายละเอียดพื้นฐานของคลาสเก่ายังคงมีอยู่ในคลาสใหม่ครบถ้วนสมบูรณ์ ซึ่งเมื่อการตรวจสอบพบว่ารายละเอียดพื้นฐานมีความตรงกันหรือเข้ากันได้แล้วจะเข้าสู่ขั้นตอนต่อไป คือ การตรวจสอบและโอนย้ายสถานะการทำงานไปยังออบเจ็กต์ใหม่เพื่อให้ออบเจ็กต์ใหม่สามารถทำงานแทนที่ออบเจ็กต์เก่าได้อย่างต่อเนื่อง

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 4.12 ข้อมูลรายละเอียดพื้นฐานของคลาสใหม่กรณีข้อมูลตรงกันกับคลาสเก่า

```
==> STEP 3 : Check Equivalence Detail.......

>>>> # 2 CHANGED FROM
There are 5 fields
>>>>      CHANGED TO
There are 7 fields
>>>> INSERT BEFORE # 13
                Field Name : test
                     Type : int
                Field Name : test1
                     Type : int

..End of Check Equivalence..

<============== CAN UPGRADE ==============>
```

รูปที่ 4.13 ผลการตรวจสอบกรณีคลาสทั้งสองสามารถนำมาทำงานแทนกันได้

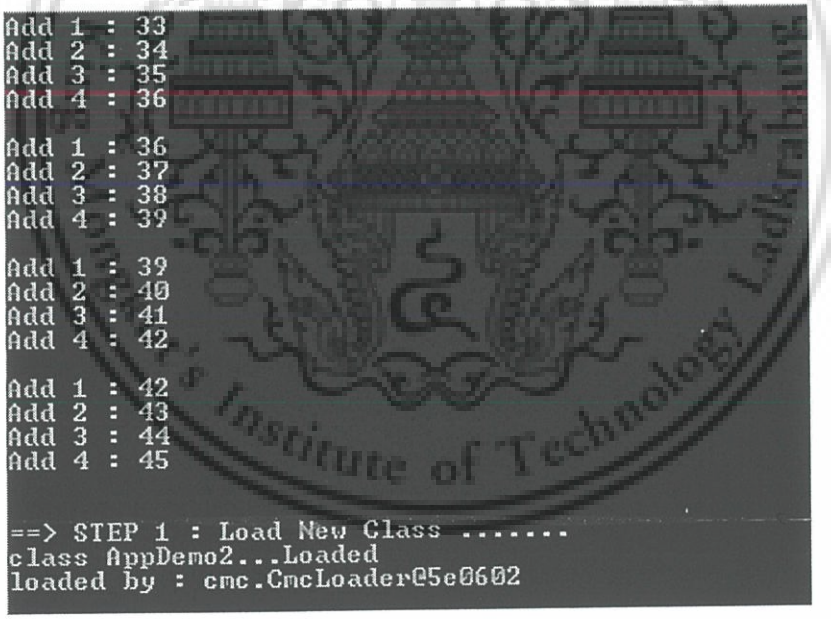
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

4.4 การตรวจสอบและโอนย้ายสถานะการทำงานไปยังออบเจ็กต์ใหม่

เมื่อผ่านขั้นตอนการตรวจสอบรายละเอียดพื้นฐานของคลาสเก่าและคลาสใหม่แล้วพบว่ามีความเข้ากันได้ นั่นคือ ระบบ CMC ได้ตรวจสอบพบว่าคลาสใหม่สามารถทำงานแทนที่คลาสเก่าได้อย่างถูกต้องภายหลังการเปลี่ยนแปลงแล้ว จะติดต่อไปยัง Component Invoker เพื่อจัดการและควบคุมสถานะการทำงานของแอพพลิเคชั่นที่กำลังทำงานอยู่นั้น โดยจะติดต่อกับแอพพลิเคชั่นผ่าน Admin.&Mgt. Interface เพื่อตรวจสอบและโอนย้ายสถานะการทำงานของแต่ละออบเจ็กต์ภายในแอพพลิเคชั่นนั้นไปยังคลาสใหม่ที่จะเข้ามาทำงานห้สามารถทำงานได้อย่างต่อเนื่องต่อไป

4.4.1 การตรวจสอบสถานะการทำงานของออบเจ็กต์เก่า

เมื่อแอพพลิเคชั่นเริ่มทำงานไปได้ระยะหนึ่งแล้ว มีการโหลดคลาสใหม่เข้ามาและผ่านการตรวจสอบความเข้ากันได้เรียบร้อยแล้ว ณ เวลาที่พบว่าคลาสใหม่สามารถทำงานแทนที่คลาสเก่าได้จะนับเป็นจุดเริ่มต้นที่ Component Invoker จะทำการติดต่อไปยังแอพพลิเคชั่นเพื่อตรวจสอบสถานะการทำงานของออบเจ็กต์เก่าในขณะนั้นผ่านอินเตอร์เฟสทันที ดังรูปที่ 4.14 จะเห็นการทำงานของแอพพลิเคชั่นก่อนที่จะนำคลาสใหม่เข้าสู่ระบบ



รูปที่ 4.14 ผลการทำงานของแอพพลิเคชั่นก่อนการอัพเกรดออบเจ็กต์

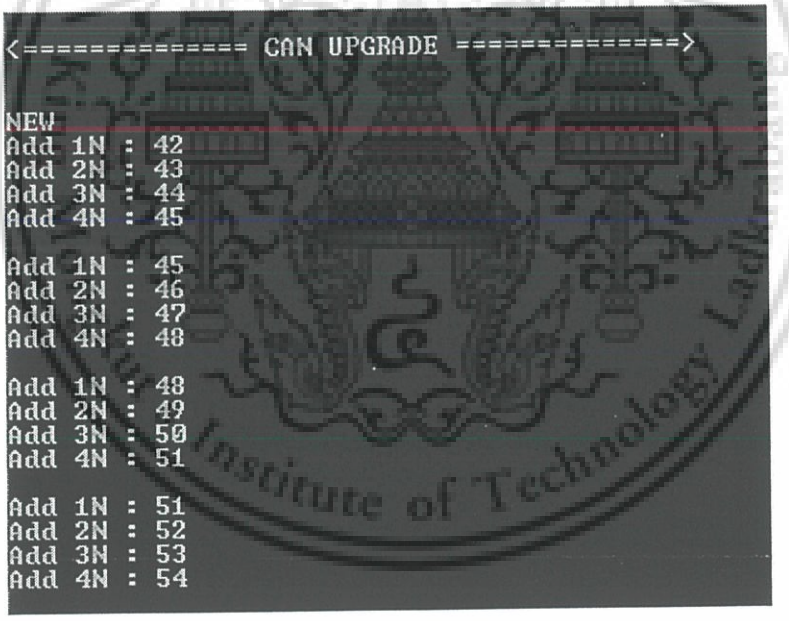
จากรูปเป็นการแสดงให้เห็นการทำงานของแอพพลิเคชั่นก่อนที่จะนำคลาสใหม่เข้าสู่ระบบ ซึ่งในช่วงระยะเวลาที่ระบบ CMC เริ่มต้นโหลดคลาสใหม่เข้าสู่ JVM ไปจนถึงการตรวจสอบความเข้ากันได้นั้น แอพพลิเคชั่นจะยังคงทำงานอยู่อย่างต่อเนื่องโดยไม่มีการหยุดชะงักภายใต้การตรวจสอบต่างๆ ของระบบ CMC

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

4.4.2 การโอนย้ายสถานะการทำงานไปยังออบเจ็กต์ใหม่

เมื่อคลาสใหม่ที่ถูกโหลดเข้ามาได้ผ่านกระบวนการตรวจสอบรายละเอียดพื้นฐานแล้ว ระบบ CMC พบว่าคลาสใหม่สามารถทำงานแทนที่คลาสเก่าได้แล้ว ในเวลานี้ Component Invoker จะเริ่มต้นสร้างออบเจ็กต์ของคลาสใหม่ขึ้นมา พร้อมทั้งตรวจสอบสถานะของออบเจ็กต์ของคลาสเก่าผ่านอินเตอร์เฟส เพื่อนำสถานะที่ได้จากการตรวจสอบส่งไปเป็นค่าเริ่มต้นให้กับออบเจ็กต์ของคลาสใหม่ที่สร้างขึ้น เพื่อให้ออบเจ็กต์ของคลาสใหม่สามารถเริ่มต้นทำงานตามสถานะที่ได้รับมานั้น ซึ่งจะส่งผลให้แอพพลิเคชั่นยังคงทำงานได้อย่างต่อเนื่องภายหลังการนำออบเจ็กต์ของคลาสใหม่เข้ามาทำงาน ดังแสดงผลการทำงานของออบเจ็กต์ใหม่ในรูปที่ 4.15

จากรูปที่ 4.15 จะเห็นว่าแอพพลิเคชั่นยังคงสามารถทำงานได้อย่างต่อเนื่อง โดยในคลาสใหม่นี้ได้มีการเปลี่ยนแปลงการแสดงผลการทำงานให้ต่างจากเดิม เพื่อให้สามารถสังเกตการทำงานได้ง่ายขึ้น โดยเมื่อเปรียบเทียบกับรูปที่ 4.14 แล้วจะพบว่าออบเจ็กต์ของคลาสใหม่ในรูปที่ 4.15 สามารถเริ่มต้นทำงานต่อเนื่องจากการทำงานเดิมของแอพพลิเคชั่นก่อนที่จะนำคลาสใหม่เข้าสู่ระบบในรูปที่ 4.14 ได้อย่างชัดเจน



รูปที่ 4.15 ผลการทำงานของแอพพลิเคชั่นหลังการอัพเกรดออบเจ็กต์

ผลการทำงานของแอพพลิเคชั่นภายหลังการอัพเกรดออบเจ็กต์นั้นจะมีข้อที่น่าสังเกตประการหนึ่ง คือ ออบเจ็กต์ของคลาสใหม่จะเริ่มต้นทำงานด้วยสถานะสุดท้ายของออบเจ็กต์เก่า ซึ่งจะคล้ายกับการทำงานยังไม่มีความต่อเนื่องได้อย่างแท้จริงเนื่องจากมีการทำงานย้อนหลังไปเล็กน้อยนั้น เพราะต้องการสร้างความมั่นใจให้กับการทำงานได้ว่าหากแอพพลิเคชั่นที่กำลังทำงานนั้นมี

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สถานะของออบเจ็กต์ใดๆ ที่ทำงานค้างอยู่ในขณะนั้นแล้ว การทำงานของออบเจ็กต์เหล่านั้นจะไม่มีการสูญหายไปในระหว่างการนำออบเจ็กต์ของคลาสใหม่เข้ามาทำงาน เพราะออบเจ็กต์ของคลาสใหม่จะเริ่มทำงานที่ค้างอยู่ให้เสร็จสิ้นอย่างแท้จริงได้อย่างสมบูรณ์

เมื่อออบเจ็กต์ของคลาสใหม่ทำงานแล้ว Component Invoker จะติดต่อไปยังออบเจ็กต์ของคลาสเก่าให้หยุดการทำงาน จากนั้นระบบ CMC จะติดต่อไปยัง Garbage Collector เพื่อตรวจสอบการอ้างถึงออบเจ็กต์ของคลาสเก่า เมื่อตรวจพบว่าไม่มีออบเจ็กต์ใดในแอพพลิเคชั่นที่มีการอ้างถึงออบเจ็กต์ของคลาสเก่านี้แล้ว ระบบ CMC จะทำลายออบเจ็กต์นี้ออกไปจากระบบซึ่งเป็นขั้นตอนสุดท้ายในการอัพเกรดออบเจ็กต์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 5
การวิเคราะห์ผลการทดลองและทดสอบสมมุติฐาน

วิทยานิพนธ์นี้ได้ทำการทดลองเปลี่ยนแปลงออบเจ็กต์ในลักษณะของการอัพเกรดออบเจ็กต์ คือ การนำออบเจ็กต์ของคลาสใหม่ที่เป็นคลาสหลักของแอพพลิเคชั่น เข้ามาทำงานแทนที่ออบเจ็กต์ของคลาสเก่าซึ่งเป็นคลาสหลักของแอพพลิเคชั่นที่กำลังประมวลผล หรือมีการทำงานภายใต้การจัดการและควบคุมโดยระบบ CMC ซึ่งการเปลี่ยนแปลงออบเจ็กต์นี้นับเป็นการเปลี่ยนแปลงออบเจ็กต์แบบไดนามิก คือ ออบเจ็กต์ของคลาสใหม่สามารถเข้ามาทำงานได้โดยไม่ต้องหยุดการทำงานของแอพพลิเคชั่น และในส่วนของการทำงานทั้งหมดของระบบ CMC นั้นจะอยู่ภายใต้ JVM ซึ่งเป็นสภาพแวดล้อมหลักของการทำงานทั้งหมดในงานวิจัยนี้

ในการทดลองอัพเกรดออบเจ็กต์นั้นจำเป็นอย่างยิ่งที่จะต้องมีการวิเคราะห์ผลการทดลองเพื่อนำไปใช้ในการพิจารณาว่าเป็นไปตามสมมุติฐานที่งานวิจัยนี้ได้ตั้งไว้หรือไม่ โดยวิทยานิพนธ์นี้ได้ตั้งสมมุติฐานของการศึกษาไว้ 2 ประการ คือ ระบบสามารถทำงานได้อย่างต่อเนื่องในระหว่างการเปลี่ยนแปลง และมีการกระจายการทำงานภายในระบบ CMCในขณะเปลี่ยนแปลงออบเจ็กต์

สำหรับสภาพแวดล้อมในการทดลองของงานวิจัยนี้ประกอบด้วย เครื่องคอมพิวเตอร์ที่มีการทำงานแบบ Standalone ซึ่งมีรายละเอียดของอุปกรณ์ต่างๆ ดังนี้

ใช้ระบบปฏิบัติการ Microsoft Windows XP Professional V.2002 Service Pack 1

มีหน่วยประมวลผล (CPU) ชนิด Intel Pentium(R) ความเร็ว 2.80 GHz

มีหน่วยความจำชนิด RAM ขนาด 2.80 GB

ขนาดความจุของฮาร์ดดิสก์ทั้งหมด 75 GB

และใช้เครื่องมือ (Tool) ในการวัดค่าหรือสิ่งที่ต้องการเพื่อนำมาใช้วิเคราะห์ผลการทดลอง คือ Jprofiler เวอร์ชั่น 3.31 ซึ่งเป็นเครื่องมือในการวัดผลและรวบรวมข้อมูลต่างๆ ที่เกี่ยวข้องกับการทำงานของแอพพลิเคชั่นที่เขียนด้วยภาษาจาวา

เมื่อได้ทดลองอัพเกรดออบเจ็กต์ตามสภาพแวดล้อมดังกล่าวแล้ว ได้ทำการวิเคราะห์ผลการทดลองต่างๆ ดังรายละเอียดต่อไปนี้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

5.1 ผลการทดสอบความต่อเนื่องของการทำงานในระหว่างการเปลี่ยนแปลง

จากการทดลองนำออบเจ็กต์ของคลาสใหม่เข้ามาทำงานแทนที่ออบเจ็กต์ของคลาสเก่าที่มีการทำงานอยู่นั้น จัดได้ว่าเป็นการเปลี่ยนแปลงการทำงานของแอพพลิเคชั่น ซึ่งงานวิจัยนี้ได้ทดลองเปลี่ยนแปลงออบเจ็กต์กับการทำงานของแอพพลิเคชั่น ดังนี้

5.1.1 ทดลองวัด Upgrade Delay

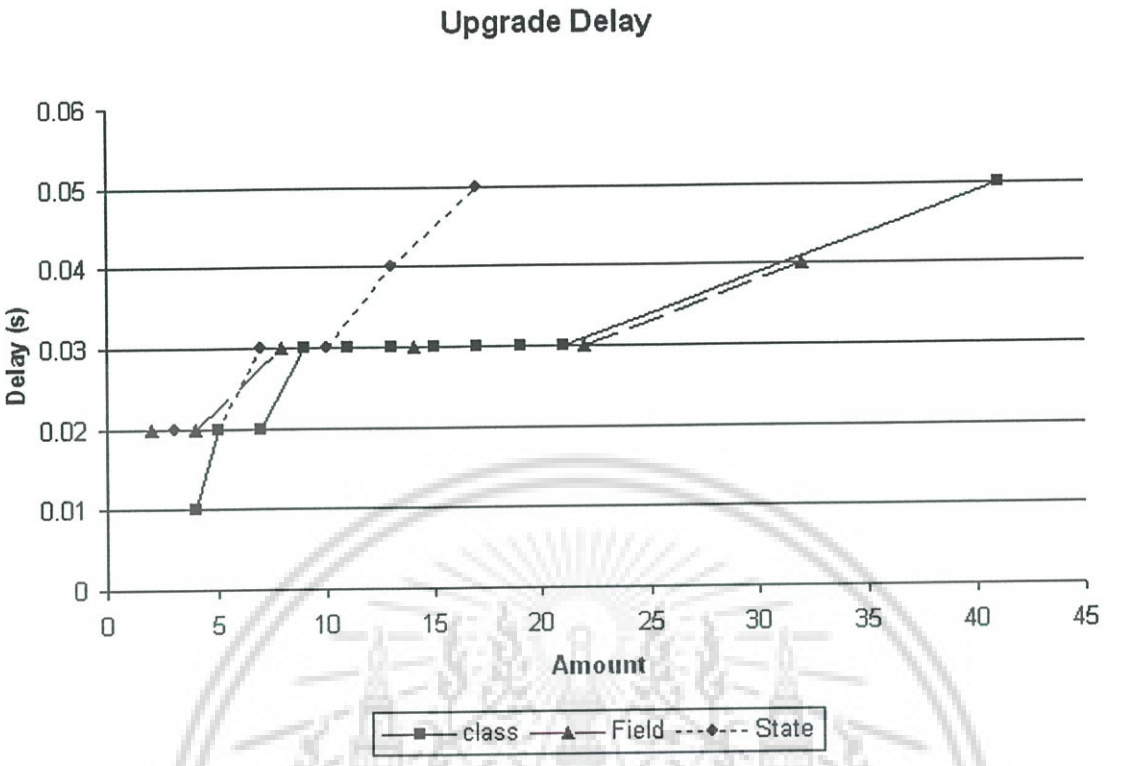
เป็นการทดลองวัดระยะเวลาการทำงานปกติของแอพพลิเคชั่น และระยะเวลาการทำงานเมื่อมีการอัพเกรดออบเจ็กต์ เพื่อหาระยะเวลาที่ใช้ในการอัพเกรดออบเจ็กต์ (Upgrade Delay)

ตารางที่ 5.1 ผลการวัด Upgrade Delay เมื่อเพิ่มจำนวนคลาส ฟิลด์ และสถานะของออบเจ็กต์ของคลาสที่จะนำมาอัพเกรด

จำนวนคลาส/แอพพลิเคชั่น		จำนวนฟิลด์/คลาส		จำนวนสถานะ/คลาส	
จำนวนคลาสทั้งหมดของแอพพลิเคชั่น	Upgrade Delay (วินาที)	จำนวนฟิลด์ที่เพิ่มขึ้น	Upgrade Delay (วินาที)	จำนวนสถานะที่เพิ่มขึ้น	Upgrade Delay (วินาที)
4	0.01	2	0.02	3	0.02
5	0.02	4	0.02	5	0.02
7	0.03	8	0.03	7	0.03
9	0.03	14	0.03	10	0.03
11	0.03	22	0.03	13	0.04
13	0.03	32	0.04	17	0.05
15	0.03	-	-	-	-
17	0.03	-	-	-	-
19	0.03	-	-	-	-
21	0.03	-	-	-	-
41	0.05	-	-	-	-

จากผลการทดลองในตารางที่ 5.1 สามารถวิเคราะห์ผลการทดลองได้ว่า การอัพเกรดออบเจ็กต์ของแอพพลิเคชั่นที่มีลักษณะการทำงานคล้ายกัน จะให้ผลการทดลองที่ค่อนข้างใกล้เคียงกัน คือ ใช้เวลาในการอัพเกรดออบเจ็กต์ไม่ต่างกันมากนัก ทั้งนี้ขึ้นอยู่กับจำนวนคลาสที่มีการทำงานร่วมกันและจำนวนของออบเจ็กต์ที่มีการโอนย้ายสถานะไปยังออบเจ็กต์ของคลาสใหม่ ดังรูปที่ 5.1

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



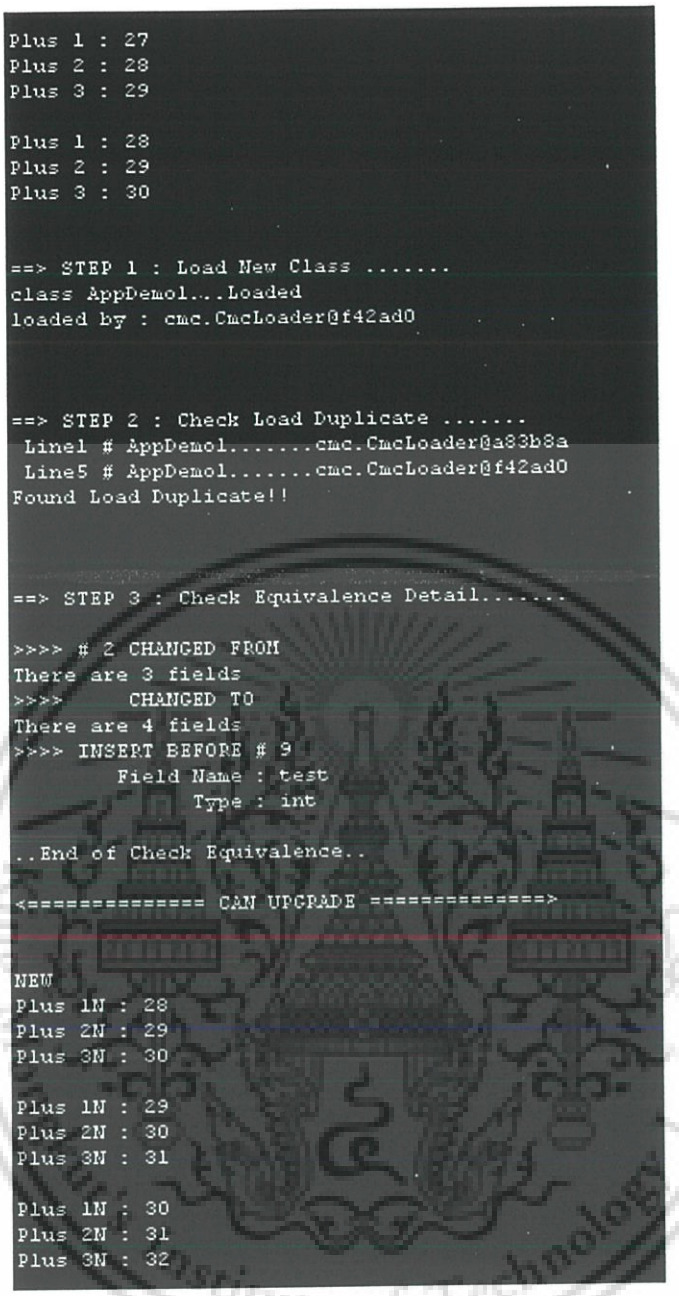
รูปที่ 5.1 ผลการวัด Upgrade Delay ของการอัพเกรดออบเจ็กต์

จากรูปที่ 5.1 จะเห็นว่า Upgrade Delay ของแต่ละกรณีมีความแตกต่างกัน คือ ในกรณีที่เพิ่มจำนวนคลาสของแอพพลิเคชั่นนั้น Delay จะคงที่อยู่ช่วงระยะเวลาหนึ่งซึ่งมีลักษณะเช่นเดียวกับกรณีที่เพิ่มจำนวนฟิลด์ ในขณะที่กรณีเพิ่มจำนวนสถานะของออบเจ็กต์ที่มีการโอนย้ายสถานะนั้นจะมี Delay ที่เพิ่มสูงขึ้นอย่างชัดเจนเมื่อเทียบกับสองกรณีแรก

จึงสามารถวิเคราะห์ได้ว่า การเพิ่มจำนวนออบเจ็กต์ที่มีการโอนย้ายสถานะการทำงานเป็นปัจจัยที่สำคัญมากที่สุดต่อ Upgrade Delay ที่เกิดขึ้น เมื่อเทียบกับการเพิ่มจำนวนคลาสและจำนวนฟิลด์

รูปที่ 5.2 เป็นการแสดงผลการทำงานของแอพพลิเคชั่นก่อนเข้าสู่กระบวนการเตรียมการอัพเกรด คือ ตรวจสอบการโหลดคลาสใหม่ และการตรวจสอบความเข้ากันได้ของรายละเอียดพื้นฐานระหว่างคลาสเก่าและคลาสใหม่ จนกระทั่งการนำคลาสใหม่เข้ามาทำงานแทนที่คลาสเก่า จะเห็นได้ว่าคลาสใหม่สามารถทำงานได้อย่างต่อเนื่อง โดยที่คลาสเก่าไม่หยุดชะงักการทำงานใดๆ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



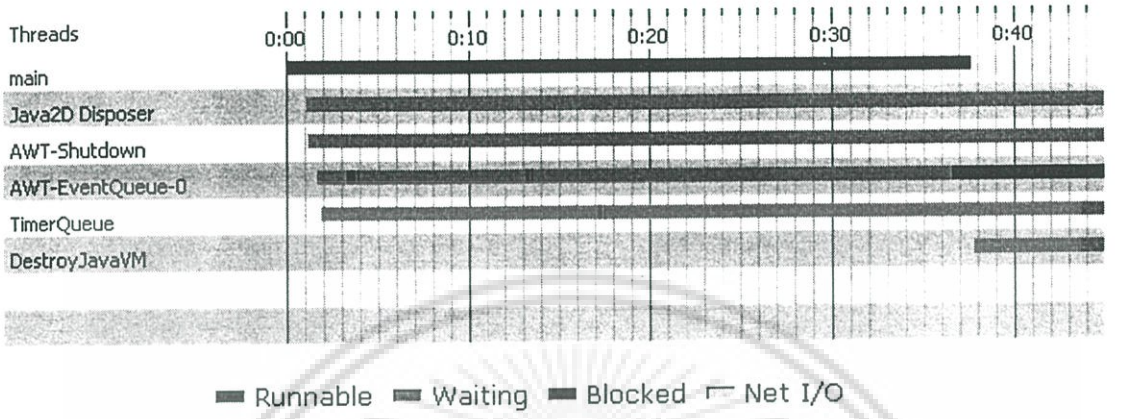
รูปที่ 5.2 ผลการทดลองอัพเกรดออบเจ็กต์ของแอพพลิเคชั่น AppDemo2

5.1.2 การตรวจสอบสถานะการทำงานของเทรด (Thread History)

เมื่อใช้โปรแกรม JProfiler ซึ่งเป็นเครื่องมือช่วยในการตรวจสอบและวัดผลการทำงานของแอพพลิเคชั่นแล้วจะเห็นได้อย่างชัดเจนว่า เมื่อมีการเปลี่ยนแปลงเกิดขึ้นการทำงานของแอพพลิเคชั่นยังคงดำเนินไปอย่างต่อเนื่อง โดยในที่นี่จะเป็นการทดลองอัพเกรดออบเจ็กต์ของแอพพลิเคชั่น AppDemo2

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

โดยผลที่ได้จากการตรวจสอบในรูปที่ 5.3 จะแสดงเทรดทั้งหมดที่เกิดขึ้นจากการทำงานของแอพพลิเคชั่น ซึ่งจะทำให้สามารถทราบได้ว่าแต่ละเทรดมีสถานะการทำงานเป็นอย่างไรในเวลาต่างๆ



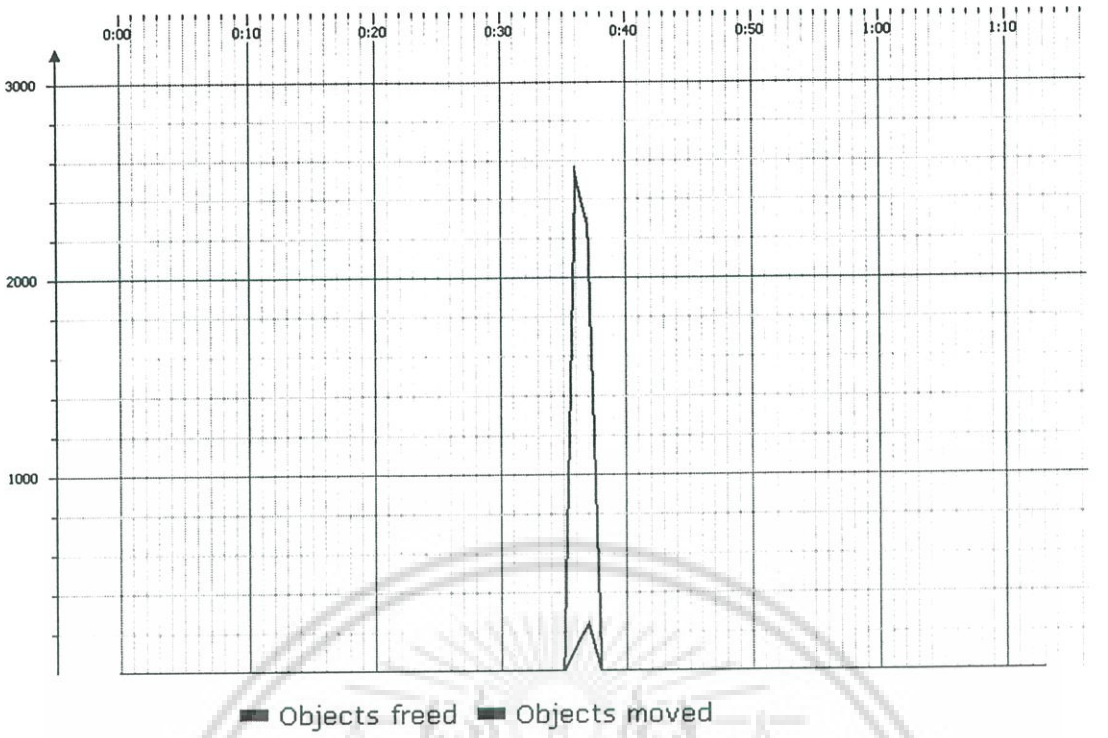
รูปที่ 5.3 สถานะของเทรดต่างๆ ที่เกิดขึ้นภายในการทำงานของแอพพลิเคชั่น

จากรูปที่ 5.3 จะเห็นว่า เทรดหลัก (Main thread) ของแอพพลิเคชั่นซึ่งเป็นเทรดของคลาสที่จะมีการเปลี่ยนแปลงเกิดขึ้นนั้น จะทำงานไปจนกระทั่งมีการสร้างเทรดของการนำคลาสใหม่เข้าสู่ระบบ (AWT-EventQueue-0) ขึ้นมา จะสังเกตเห็นว่าทั้งสองเทรดมีการทำงานควบคู่กันไประยะหนึ่ง ก่อนที่เทรดหลักจะหยุดการทำงานในขณะที่เทรดของคลาสใหม่ยังคงทำงานต่อไปแทนที่เทรดหลัก จึงสามารถสรุปได้ว่าคลาสใหม่สามารถทำงานได้อย่างต่อเนื่องโดยการทำงานของคลาสเก่าไม่หยุดชะงัก

5.1.3 การตรวจสอบ Garbage Collector

เมื่อตรวจสอบการทำงานของออบเจ็กต์ภายใน Garbage Collector แล้วจะพบว่าออบเจ็กต์ต่างๆ มีความเคลื่อนไหวอย่างไรในแอพพลิเคชั่นที่กำลังทำงานภายใต้ระบบ CMC ดังรูปที่ 5.4 จะเห็นได้อย่างชัดเจนว่าออบเจ็กต์เก่ายังคงทำงานในขณะที่ออบเจ็กต์ใหม่เข้ามาทำงานแทนที่ ซึ่งจะส่งผลให้การทำงานของแอพพลิเคชั่นไม่หยุดชะงัก จึงนับว่าการทดลองนี้ให้ผลลัพธ์ที่สอดคล้องกับสมมุติฐานที่ตั้งไว้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 5.4 การเปลี่ยนแปลงออบเจ็กต์ภายใน Garbage Collector

5.2 ผลการตรวจสอบการกระจายการทำงานภายในระบบ CMC

เนื่องจากการทำงานของระบบ CMC เกิดจากการทำงานร่วมกันของคอมโพเนนต์ต่างๆ และในการเปลี่ยนแปลงออบเจ็กต์นั้น ได้มีการใช้กลไกลของ Multiple Class Loaders ในการโหลดคลาสของออบเจ็กต์ใหม่ด้วยคลาสโหลดเดอร์ที่แตกต่างกัน เพื่อให้สามารถโหลดคลาสที่มีชื่อเดียวกันเข้าสู่สภาพแวดล้อมเดียวกันได้ในขณะ Runtime และในขณะที่มีการเปลี่ยนแปลงออบเจ็กต์นั้น ระบบ CMC จะมีการติดต่อไปยังคอมโพเนนต์ต่างๆ ตามหน้าที่การทำงานของแต่ละคอมโพเนนต์ เพื่อให้สามารถจัดการและควบคุมการทำงานของแอพพลิเคชั่นที่กำลังทำงานในขณะนั้นร่วมกัน จึงจัดได้ว่าในขณะเปลี่ยนแปลงออบเจ็กต์นั้น ระบบมีการกระจายการทำงานไปยังส่วนต่างๆ ภายในระบบ CMC ซึ่งในการตรวจสอบว่าการทำงานของระบบ CMC ในขณะเปลี่ยนแปลงออบเจ็กต์นั้นมีการกระจายการทำงานไปยังส่วนต่างๆ ของระบบ

สำหรับการวิเคราะห์การกระจายภาระการทำงานในส่วนนี้ จะเป็นการวิเคราะห์ผลที่ได้จากการตรวจสอบและบันทึกข้อมูลผลการใช้ทรัพยากรต่างๆ โดยใช้เครื่องมือ JProfiler ในขณะนำคลาสใหม่เข้าสู่ระบบ CMC เพื่อนำมาทำงานแทนที่คลาสของแอพพลิเคชั่นที่กำลังทำงาน โดยจะวัดค่าการใช้ทรัพยากร 3 ประเภท คือ ปริมาณการใช้พื้นที่หน่วยความจำ (Memory requirement) ปริมาณการใช้พื้นที่ของ Heap ในการทำงานของแอพพลิเคชั่น และเวลาที่ใช้ภายในหน่วยประมวลผลกลาง (CPU Time) ซึ่งสามารถดูรายละเอียดของผลการวัดต่างๆ ได้ดังนี้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

5.2.1 ปริมาณการใช้พื้นที่หน่วยความจำ (Memory requirement)

ในขณะที่นำออบเจ็กต์ของคลาสใหม่เข้ามาทำงานแทนที่นั้น แอพพลิเคชั่นจะมีการใช้พื้นที่หน่วยความจำ ดังรายละเอียดต่อไปนี้

1. ผลการตรวจสอบในขณะอัพเกรดออบเจ็กต์

ในขณะที่นำออบเจ็กต์ของคลาสใหม่เข้ามาทำงานแทนที่นั้น คอมโพเนนต์ต่างๆ ของระบบ CMC จะมีการใช้พื้นที่หน่วยความจำดังตารางที่ 5.2

ตารางที่ 5.2 ปริมาณการใช้พื้นที่หน่วยความจำของแต่ละคอมโพเนนต์ภายในระบบ CMC ขณะอัพเกรดเทียบกับหลังการอัพเกรดออบเจ็กต์

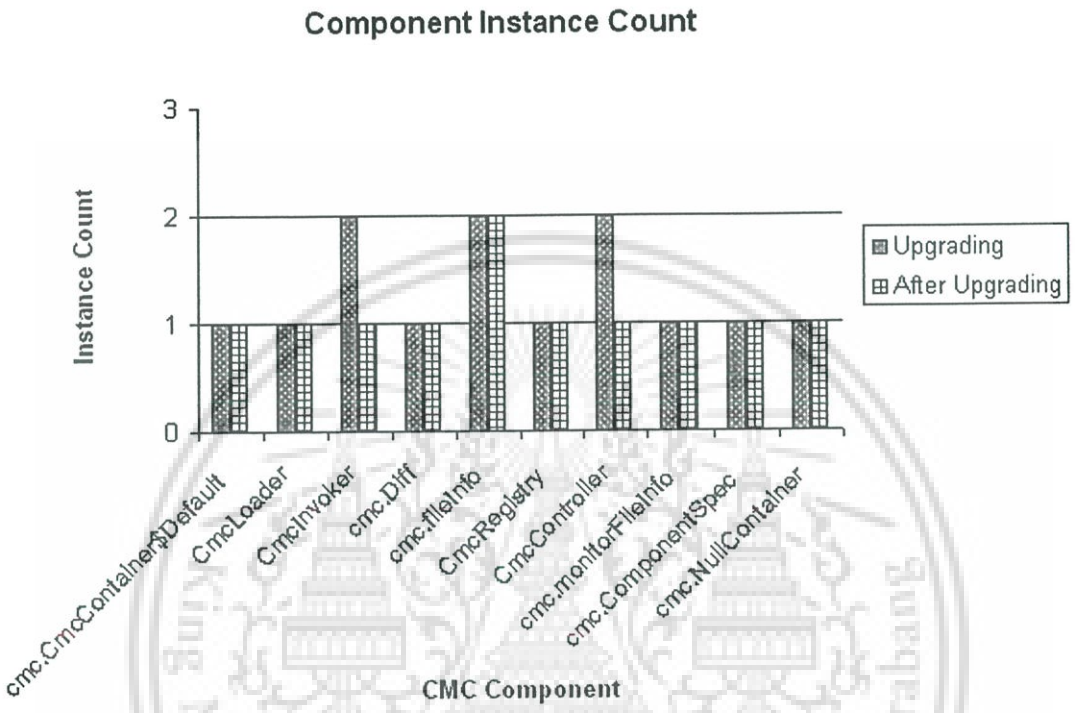
Name	Upgrading		After Upgrading	
	Instance count	Size	Instance count	Size
cmc.CmcContainer$Default	1	72	1	72
CmcLoader	1	64	1	64
CmcInvoker	2	48	1	24
cmc.Diff	1	40	1	40
cmc.fileInfo	2	48	2	48
CmcRegistry	1	16	1	16
CmcController	2	16	1	8
cmc.monitorFileInfo	1	16	1	16
cmc.ComponentSpec	1	16	1	16
cmc.NullContainer	1	8	1	8

จากตารางที่ 5.2 ในส่วนของขณะทำการอัพเกรดออบเจ็กต์ (Upgrading) พบว่าแต่ละคอมโพเนนต์ที่มีการทำงานร่วมกันภายในระบบ CMC มีจำนวนออบเจ็กต์ที่ (Instance Count) ใกล้เคียงกัน ซึ่งหมายถึงมีการกระจายภาระงานให้กับแต่ละคอมโพเนนต์ แต่ในส่วนของขนาดของออบเจ็กต์ที่แต่ละคอมโพเนนต์มีการใช้งานนั้นจะเห็นว่า cmc.CmcContainer$Default ซึ่งเป็น Standard Method ของระบบ CMC มีขนาดออบเจ็กต์สูงที่สุด เนื่องจากเป็นส่วนที่ทำหน้าที่ควบคุมการทำงานโดยรวมของการอัพเกรดออบเจ็กต์ ในขณะที่ CmcLoader มีขนาดออบเจ็กต์รองลงมาเพราะมีหน้าที่หลักในการโหลดคลาสต่างๆ เข้าสู่ JVM

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2. <u>ผลการตรวจสอบเมื่อผ่านการอัพเกรดออบเจ็กต์</u>

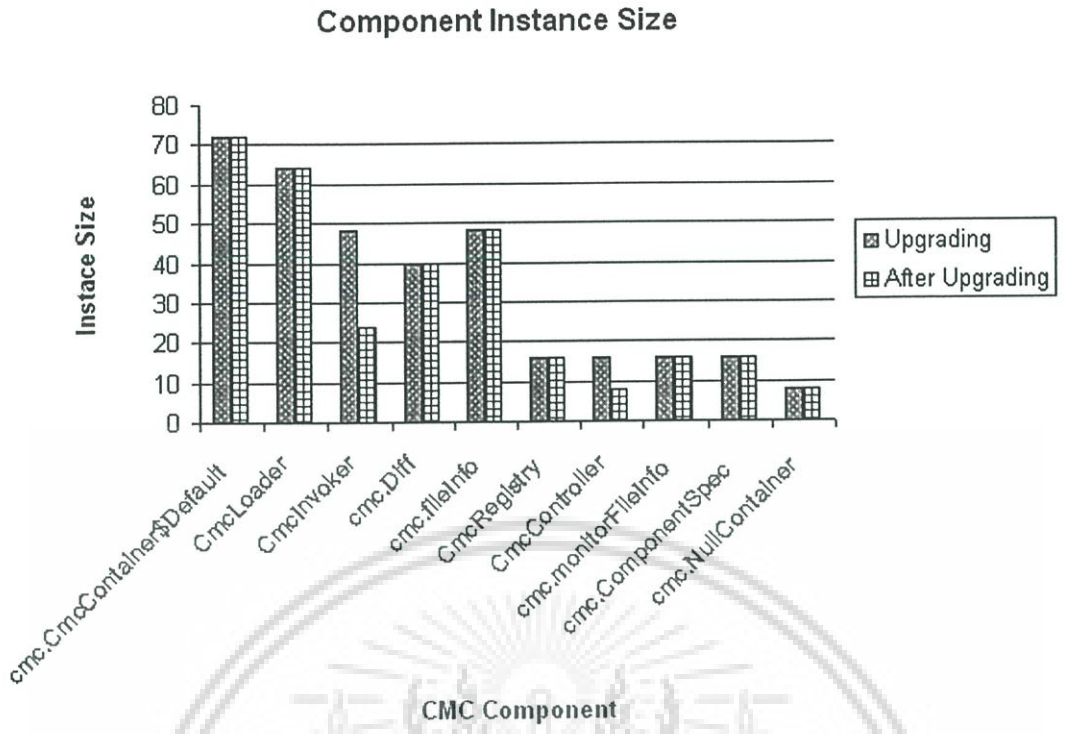
เมื่อแอพพลิเคชั่นเริ่มทำงานแล้วได้เปลี่ยนแปลงการออบเจ็กต์โดยการโอนย้ายสถานะการทำงานของออบเจ็กต์เก่าไปยังออบเจ็กต์ใหม่แล้ว คอมโพเนนต์ต่างๆ ของระบบ CMC จะมีการใช้พื้นที่หน่วยความจำดังตารางที่ 5.2 ในส่วนของ After Upgrading



รูปที่ 5.5 จำนวนของออบเจ็กต์ที่เกิดขึ้นของแต่ละคอมโพเนนต์ขณะอัพเกรดเทียบกับหลังการอัพเกรดออบเจ็กต์

จากตารางที่ 5.2 ในส่วนของหลังการอัพเกรดออบเจ็กต์ (After Upgrading) นั้นเมื่อเปรียบเทียบกับข้อมูลปริมาณการใช้พื้นที่หน่วยความจำในขณะอัพเกรดออบเจ็กต์ (Upgrading) แล้ว จะเห็นว่าส่วนใหญ่มีข้อมูลการใช้พื้นที่เท่ากันทั้งหมด ยกเว้นข้อมูลของ CmcController และ CmcInvoker ที่มีการใช้พื้นที่หน่วยความจำลดลง เนื่องจากเมื่อผ่านการอัพเกรดออบเจ็กต์แล้ว คอมโพเนนต์นี้จะลดบทบาทหน้าที่ในการจัดการและควบคุมการทำงานของแอพพลิเคชั่นลง ทำให้มีการใช้พื้นที่หน่วยความจำลดลงตามภาระการทำงาน จากข้อมูลในตารางที่ 5.2 นั้นสามารถสร้างกราฟได้ดังรูปที่ 5.5 และ 5.6

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 5.6 ขนาดของออบเจ็กต์ที่เกิดขึ้นของแต่ละคอมโพเนนต์ ขณะอัพเกรดเทียบกับหลังการอัพเกรดออบเจ็กต์

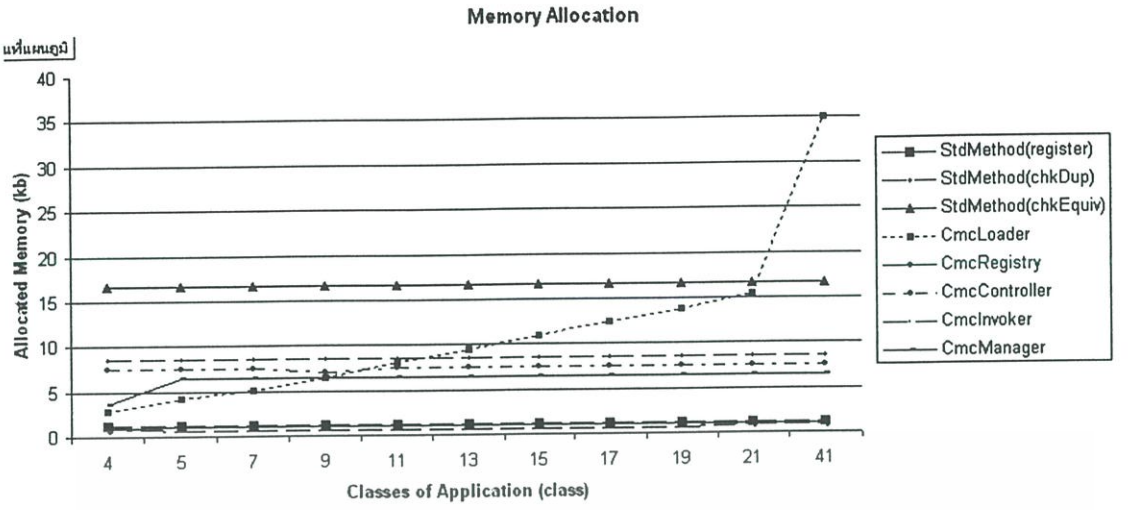
3. <u>ผลการตรวจสอบการใช้พื้นที่หน่วยความจำเมื่อเพิ่มจำนวนคลาสของแอพพลิเคชั่น</u>

เมื่อทดลองเปลี่ยนแปลงออบเจ็กต์ของแอพพลิเคชั่นที่มีจำนวนคลาสเพิ่มขึ้นแล้ว คอมโพเนนต์ต่างๆ ของระบบ CMC จะมีการใช้พื้นที่หน่วยความจำดังตารางที่ 5.3

ตารางที่ 5.3 ปริมาณการใช้พื้นที่หน่วยความจำของแต่ละคอมโพเนนต์ภายในระบบ CMC เมื่อเพิ่มจำนวนคลาสของแอพพลิเคชั่น

Class	StdMethod (register)	StdMethod (chkDup)	StdMethod (chkEquiv)	Cmc Loader	Cmc Registry	Cmc Controller	Cmc Invoker	Cmc Manager
4	1.24	8.52	16.59	2.91	0.91	7.45	0.97	3.54
5	1.24	8.52	16.61	4.18	1	7.45	0.66	6.37
7	1.24	8.52	16.61	5.12	1	7.45	0.66	
9	1.17	8.52	16.61	6.5	1	7.45	0.66	6.37
11	1.24	8.52	16.61	8.06	1	7.45	0.66	6.37
13	1.17	8.52	16.61	9.45	1	7.45	0.66	6.37
15	1.24	8.52	16.61	10.94	1	7.45	0.66	6.37
17	1.24	8.52	16.61	12.38	1	7.45	0.66	6.37
19	1.17	8.52	16.61	13.77	1	7.45	0.66	6.37
21	1.17	8.52	16.61	15.4	1	7.45	0.66	6.37
41	1.17	8.52	16.61	35	1	7.45	0.66	6.37

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 5.7 การใช้พื้นที่หน่วยความจำของแต่ละคอมโพเนนต์ เมื่อเพิ่มจำนวนคลาสของแอพพลิเคชั่น

เมื่อทดลองอัพเกรดออบเจ็กต์ของแอพพลิเคชั่นที่มีจำนวนคลาสเพิ่มขึ้น จะพบว่าแต่ละคอมโพเนนต์มีการใช้พื้นที่หน่วยความจำใกล้เคียงกัน มีเพียง CmcLoader ที่มีการใช้พื้นที่ค่อนข้างมาก เนื่องจากมีหน้าที่โหลดคลาสต่างๆ เข้าสู่ JVM และมีการติดต่อกับ Class Loader ของระบบ (System Class Loader) จึงจำเป็นต้องใช้พื้นที่ในการทำงานนี้เพิ่มขึ้นตามจำนวนคลาสของแอพพลิเคชั่น ดังรูปที่ 5.7

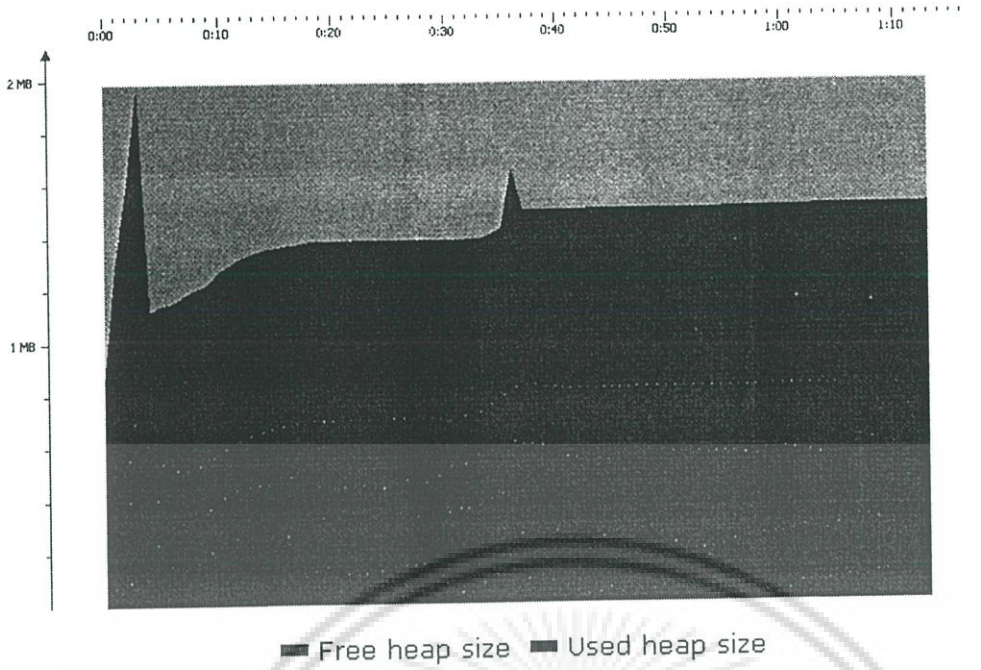
จากการใช้กลไกของ Multiple Class Loaders สำหรับโหลดคลาสชื่อเดียวกันเข้าสู่สภาพแวดล้อมเดียวกันได้ในขณะ runtime แบบไดนามิก นับว่ามีประโยชน์อย่างมากในกรณีที่มีการนำแอพพลิเคชั่นต่างๆ เข้ามาทำงานภายใต้ระบบ CMC พร้อมๆ กัน เนื่องจากการนำกลไกนี้มาใช้จะช่วยให้ระบบมีการกระจายการโหลดคลาสไปยังคลาสโหลดเดอร์ที่แตกต่างกันได้ ซึ่งจะส่งผลให้เกิดการกระจายปริมาณการใช้พื้นที่หน่วยความจำไปยังคลาสโหลดเดอร์ต่างๆ ได้

5.2.2 ปริมาณพื้นที่ของ Heap ที่ใช้ในการทำงานของแอพพลิเคชั่น

ในขณะที่นำออบเจ็กต์ของคลาสใหม่เข้ามาทำงานแทนที่นั้น แอพพลิเคชั่นจะมีการใช้พื้นที่ Heap ในการทำงาน ดังรูปที่ 5.9

จากรูปที่ 5.9 จะเห็นว่าเมื่อแอพพลิเคชั่นเริ่มทำงานออบเจ็กต์ต่างๆ ของแอพพลิเคชั่นจะมีการใช้พื้นที่ Heap สูงมากในช่วงแรกและลดลงตามลำดับ จากนั้นเมื่อนำออบเจ็กต์ใหม่เข้ามาทำงานแทนที่จะพบว่าออบเจ็กต์ต่างๆ มีการใช้พื้นที่ Heap สูงขึ้นเพียงเล็กน้อยในระยะเวลาที่ออบเจ็กต์ใหม่เริ่มต้นทำงานในระยะเวลาสั้นๆ จึงสามารถสรุปได้ว่าในการนำออบเจ็กต์ใหม่เข้ามาทำงานนั้นมีผลกระทบต่อการใช้พื้นที่ Heap ค่อนข้างน้อยมาก

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 5.8 ปริมาณการใช้พื้นที่ Heap ในขณะอัพเกรดออบเจ็กต์

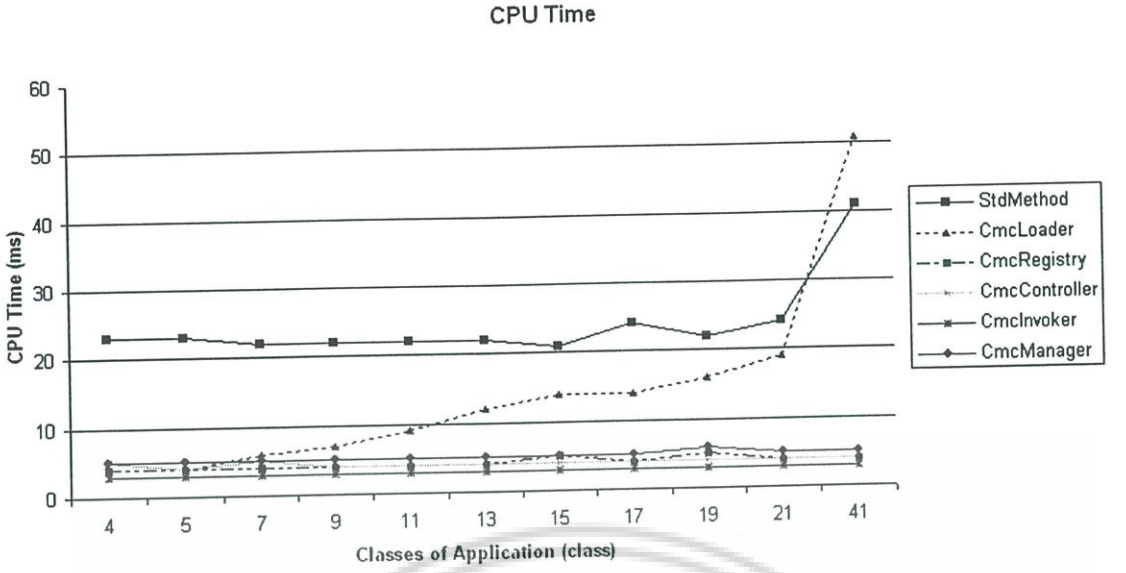
5.2.3 เวลาที่ใช้ภายในหน่วยประมวลผลกลาง (CPU Time)

ในขณะที่นำออบเจ็กต์ของคลาสใหม่เข้ามาทำงานแทนที่นั้น แต่ละคอมโพเนนต์ภายในระบบ CMC มีการใช้เวลาภายในหน่วยประมวลผลกลาง ดังรายละเอียดต่อไปนี้

ตารางที่ 5.6 ปริมาณการใช้ CPU Time ของแต่ละคอมโพเนนต์ภายในระบบ CMC เมื่อเพิ่มจำนวนคลาสของแอพพลิเคชั่น

Class	StdMethod	CmcLoader	CmcRegistry	CmcController	CmcInvoker	CmcManager
4	23	4	4	5	3	5
5	23	4	4	4	3	5
7	22	6	4	5	3	5
9	22	7	4	4	3	5
11	22	9	4	4	3	5
13	22	12	4	4	3	5
15	21	14	5	4	3	5
17	24	14	4	4	3	5
19	22	16	5	4	3	6
21	24	19	4	4	3	5
41	41	51	4	4	3	5

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 5.9 การใช้เวลาภายในหน่วยประมวลผลกลางของแต่ละคอมโพเนนต์เมื่อเพิ่มจำนวนคลาสของแอพพลิเคชั่น

เมื่อทดลองอัพเกรดออบเจ็กต์ของแอพพลิเคชั่นที่มีจำนวนคลาสเพิ่มขึ้น ในรูปที่ 5.8 จะพบว่าแต่ละคอมโพเนนต์มีการใช้ CPU Time ใกล้เคียงกัน โดย StdMethod (CMC Sandard Method) มีการใช้ CPU Time สูงกว่าคอมโพเนนต์อื่น เนื่องจากเป็นส่วนที่มีหน้าที่ติดต่อกับคอมโพเนนต์ต่างๆ ในขณะที่มีการอัพเกรดออบเจ็กต์ และเป็นส่วนที่รวบรวมการทำงานหลักๆ ของระบบ CMC จึงทำให้ปริมาณการใช้ CPU Time สูงที่สุด โดยจากรูปจะเห็นว่าเมื่อทดลองเพิ่มจำนวนคลาสของแอพพลิเคชั่นมากขึ้น StdMethod จะยังคงใช้ CPU Time ในระดับที่ใกล้เคียงกัน จนกระทั่งทดลองกับแอพพลิเคชั่นที่มีจำนวน 41 คลาสจึงจะมีปริมาณ CPU Time สูงขึ้นอย่างชัดเจน ซึ่งในความเป็นจริงนั้นปริมาณการใช้ CPU Time ของ StdMethod ในรูปนั้น ไม่ได้เป็นของ StdMethod เพียงส่วนเดียว แต่เกิดจากการรวมค่า CPU Time ของส่วนประกอบอื่นๆ ที่มีการทำงานเกี่ยวข้องกับ StdMethod ด้วย เช่น cmc.NullContainer cmc.ComponentSpec เป็นต้น

สำหรับคอมโพเนนต์ CmcLoader (Component Loader) นั้นจะพบว่ามีการใช้ CPU Time เพิ่มขึ้นตามจำนวนคลาสของแอพพลิเคชั่น เนื่องจากเป็นส่วนที่ทำหน้าที่โหลดคลาสทั้งหมดของแอพพลิเคชั่นเข้าสู่ JVM ซึ่งนับเป็นการทำงานโดยปกติของระบบที่ทำงานภายใต้สภาพแวดล้อมของ JVM คือ Class Loader ที่กำหนดขึ้นมาใช้งานเอง (Custom Class Loader) ซึ่งในที่นี้ คือ Component Loader จะมีการใช้ CPU Time ร่วมกับ Class Loader ของระบบ (System Class Loader) จึงมีผลทำให้คอมโพเนนต์นี้ของระบบ CMC ใช้ปริมาณ CPU Time ค่อนข้างสูงเมื่อเทียบกับคอมโพเนนต์อื่นๆ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

5.3 ผลกระทบจากการเปลี่ยนแปลงออบเจ็กต์เพื่อให้แอพพลิเคชั่นทำงานได้อย่างต่อเนื่อง

จากแนวความคิดและวิธีการเปลี่ยนแปลงออบเจ็กต์โดยการนำออบเจ็กต์ใหม่เข้ามาทำงานแทนที่ออบเจ็กต์ของแอพพลิเคชั่นโดยไม่มีการหยุดชะงักการทำงานนั้น เมื่อทำการสร้างสภาพแวดล้อมสำหรับจัดการและควบคุมการทำงานของแอพพลิเคชั่นในขณะเปลี่ยนแปลงออบเจ็กต์ในรูปของระบบ CMC แล้ว พบว่ายังต้องอาศัยความร่วมมือจากแอพพลิเคชั่นในการอิมพลีเมนต์อินเตอร์เฟส สำหรับใช้ในการติดต่อระหว่างระบบ CMC และออบเจ็กต์ต่างๆ ของแอพพลิเคชั่น

เมื่อทำการทดลองเปลี่ยนแปลงออบเจ็กต์ตามวิธีการดังกล่าว พบว่าแอพพลิเคชั่นสามารถทำงานได้อย่างต่อเนื่องในขณะมีการเปลี่ยนแปลงออบเจ็กต์จริง โดยออบเจ็กต์เก่ายังคงทำงานในช่วงเวลาเดียวกับที่ออบเจ็กต์ใหม่เริ่มต้นทำงานทันทีเมื่อได้รับสถานะการทำงานของออบเจ็กต์ต่างๆ ในแอพพลิเคชั่นนั้น นั่นคือ ออบเจ็กต์ใหม่และออบเจ็กต์เก่าจะมีการทำงานควบคู่กันไปช่วงระยะเวลาสั้นๆ ช่วงหนึ่ง ก่อนที่ออบเจ็กต์เก่าจะหยุดการทำงานลง ซึ่งจากการที่ออบเจ็กต์ทั้งสองมีการทำงานควบคู่กันไปนั้น อาจส่งผลให้การทำงานของแอพพลิเคชั่นในเวลานั้นมีความผิดพลาดได้ เนื่องจากในกระบวนการเปลี่ยนแปลงออบเจ็กต์ตามวิธีการนี้ ยังไม่มีการตรวจสอบผลการทำงานของออบเจ็กต์ใหม่ที่จะนำมาทำงานแทนที่ ว่าสามารถให้ผลการทำงานตรงกันกับออบเจ็กต์เก่าหรือไม่ ดังนั้น หากออบเจ็กต์ใหม่มีการเปลี่ยนแปลงฟังก์ชั่นการทำงานหรือมีรูปแบบการประมวลผลที่แตกต่างจากเดิมแล้ว การนำออบเจ็กต์ใหม่นี้มาทำงานแทนที่ออบเจ็กต์เก่า จึงอาจส่งผลให้การทำงานของแอพพลิเคชั่นนั้นมีผลการทำงานที่ขัดแย้งกับการทำงานเดิมได้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 6
สรุปผลการวิจัยและข้อเสนอแนะ

จากแนวความคิดและวิธีการจัดการและควบคุมการเปลี่ยนแปลงออบเจ็กต์โดยใช้วิธีการโอนย้ายสถานะการทำงานของออบเจ็กต์ของแอพพลิคัชั่นในขณะทำงาน ภายใต้สภาพแวดล้อมของระบบ CMC นั้น เมื่อได้ดำเนินการอิมพลีเมนต์ระบบ CMC ตามแนวความคิดที่วางไว้ และทดลองนำแอพพลิเคชั่นมาทำการเปลี่ยนแปลงออบเจ็กต์ในขณะทำงานแล้ว สามารถสรุปผล และทำการวิเคราะห์เกี่ยวกับข้อจำกัดของงานวิจัย รวมทั้งนำเสนอข้อเสนอแนะเพิ่มเติม เพื่อใช้เป็นแนวทางในการพัฒนาประสิทธิภาพการทำงานของระบบได้ ดังรายละเอียดต่อไปนี้

6.1 สรุปผลการวิจัย

งานวิจัยนี้เป็นการนำเสนอสภาพแวดล้อมชนิด Runtime environment ในรูปของระบบคอนเทนเนอร์แบบรักษาความต่อเนื่อง (Continuous Migration Container หรือ CMC) ให้กับการทำงานของแอพลิเคชั่นหรือโปรแกรมที่เขียนด้วยภาษาจาวา ที่สนับสนุนการเปลี่ยนแปลงคลาสที่ใช้ในการทำงานของแอพลิเคชั่นในขณะ Runtime ได้แบบไดนามิก โดยสามารถสรุปเป็นประเด็นต่างๆ ได้ดังนี้

1. เป็นการสร้างสภาพแวดล้อมสำหรับการทำงานของแอพลิเคชั่น (Runtime environment) ในรูปแบบของระบบคอนเทนเนอร์แบบรักษาความต่อเนื่อง ให้กับการทำงานของแอพลิเคชั่นหรือโปรแกรมที่เขียนด้วยภาษาจาวา
2. มีการใช้กลไก Multiple class loaders ในการโหลดคลาสของออบเจ็กต์ใหม่เข้าสู่ระบบ CMC ในขณะ Runtime เพื่อให้สามารถใช้งานคลาสของออบเจ็กต์เก่าและใหม่ที่มีชื่อคลาสเดียวกันได้ในสภาพแวดล้อมเดียวกัน
3. มีการตรวจสอบรายละเอียดพื้นฐานของคลาสที่จะมีการเปลี่ยนแปลงก่อนทำการโอนย้ายสถานะการทำงาน เพื่อให้เกิดความมั่นใจได้ในระดับหนึ่งว่า คลาสใหม่จะสามารถทำงานแทนที่คลาสเก่าได้อย่างถูกต้องภายหลังการเปลี่ยนแปลงที่เกิดขึ้น
4. มีการใช้อินเตอร์เฟสสำหรับติดต่อระหว่างระบบ CMC และแอพพลิเคชั่นหรือโปรแกรมที่กำลังทำงาน เพื่อจัดการและควบคุมการโอนย้ายสถานะการทำงานของออบเจ็กต์ ซึ่งจำเป็นอาศัยความร่วมมือจากแอพพลิเคชั่น คือ คลาสที่จะสามารถเปลี่ยนแปลงได้ภายใต้ระบบ CMC จะต้องมีการอิมพลีเมนต์อินเตอร์เฟสนี้
5. ออบเจ็กต์ของคลาสใหม่จะเริ่มต้นทำงานตามสถานะที่ได้โอนย้ายมาจากการตรวจสอบสถานะของออบเจ็กต์เก่า ซึ่งการนำสถานะของออบเจ็กต์เก่านั้นจะกระทำหลังจากผ่าน

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

กระบวนการตรวจสอบรายละเอียดพื้นฐานแล้ว พบว่าออบเจ็กต์ของคลาสใหม่สามารถทำงานเทนที่ออบเจ็กต์ของคลาสเก่าได้

6. ออบเจ็กต์ของคลาสเก่าที่กำลังทำงานอยู่นั้นไม่จำเป็นต้องหยุดการทำงานเพื่อให้ออบเจ็กต์ใหม่เริ่มต้นทำงานแทนที่ เนื่องจากระบบ CMC สามารถสนับสนุนให้ออบเจ็กต์ของคลาสใหม่เข้ามาทำงานแทนที่ออบเจ็กต์ของคลาสเก่าได้ในขณะที่ออบเจ็กต์ของคลาสเก่ายังคงมีการทำงานค้างอยู่ และออบเจ็กต์ของคลาสเก่าจะหยุดการทำงานเมื่อการทำงานที่ค้างอยู่นั้นได้สิ้นสุดลงพร้อมๆ กับที่ออบเจ็กต์ของคลาสใหม่ทำงานต่อจากสถานะเดิมที่ได้รับมาทันที

จากสรุปผลการวิจัยดังกล่าว สภาพแวดล้อมในรูปของระบบคอนเทนเนอร์ที่จัดทำขึ้นนั้นสามารถรองรับการเปลี่ยนแปลง โดยการนำออบเจ็กต์ใหม่เข้ามาทำงานแทนที่ออบเจ็กต์เดิมที่มีอยู่ในระบบ โดยที่ระบบหรือแอพพลิเคชั่นยังคงทำงานได้อย่างต่อเนื่อง รวมทั้งสามารถเชื่อมั่นได้ในระดับหนึ่งว่าภายหลังการเปลี่ยนแปลงที่เกิดขึ้นนั้น แอพพลิเคชั่นสามารถทำงานได้อย่างถูกต้อง เนื่องจากมีการตรวจสอบความเข้ากันได้ของรายละเอียดพื้นฐาน ระหว่างออบเจ็กต์ใหม่และออบเจ็กต์ที่มีอยู่ก่อนที่จะโอนย้ายสถานะการทำงานของออบเจ็กต์

6.2 ข้อจำกัดของงานวิจัย

เนื่องจากงานวิจัยนี้ได้มีการทดลองนำแอพพลิเคชั่นที่มีการประมวลผลหรือมีการทำงานไม่ซับซ้อนมากนักมาใช้ในการทดลอง และมีการตรวจสอบความเข้ากันได้เฉพาะรายละเอียดพื้นฐานของคลาสที่มีการเปลี่ยนแปลงเท่านั้น ยังไม่มีการตรวจสอบผลการทำงานจึงอาจมั่นใจได้เพียงระดับหนึ่งว่าผลการทำงานภายหลังการเปลี่ยนแปลงที่เกิดขึ้นจะถูกต้อง รวมทั้งวิธีการโอนย้ายสถานะของออบเจ็กต์นั้น ยังต้องอาศัยความร่วมมือจากแอพพลิเคชั่นที่นำมาทำงานภายใต้สภาพแวดล้อมของระบบ CMC จึงทำให้งานวิจัยนี้มีข้อจำกัดต่างๆ ดังรายละเอียดต่อไปนี้

1. เนื่องจากการโอนย้ายสถานะการทำงานของออบเจ็กต์ภายใต้ระบบ CMC มีลักษณะการทำงานของออบเจ็กต์ที่มีการเปลี่ยนแปลงคาบเกี่ยวกัน คือ ออบเจ็กต์ใหม่และออบเจ็กต์เก่าจะมีการทำงานควบคู่กันไปช่วงระยะเวลาสั้นๆ ช่วงหนึ่ง ก่อนที่ออบเจ็กต์เก่าจะหยุดการทำงานลง ซึ่งจากการที่ออบเจ็กต์ทั้งสองมีการทำงานควบคู่กันไปนั้น อาจส่งผลให้การทำงานของแอพพลิเคชั่นในเวลานั้นมีความผิดพลาดได้ เนื่องจากในกระบวนการเปลี่ยนแปลงออบเจ็กต์ตามวิธีการนี้ มีการตรวจสอบความเข้ากันได้เฉพาะรายละเอียดพื้นฐานของคลาสที่มีการเปลี่ยนแปลงเท่านั้น ยังไม่มีการตรวจสอบผลการทำงานของออบเจ็กต์ใหม่ที่จะนำมาทำงานแทนที่ ว่าสามารถให้ผลการทำงานตรงกันกับออบเจ็กต์เก่าหรือไม่ ดังนั้น หากออบเจ็กต์ใหม่มีการเปลี่ยนแปลงฟังก์ชั่นการทำงาน หรือมีรูปแบบการประมวลผลที่แตกต่างจากเดิมแล้ว การนำออบเจ็กต์ใหม่นี้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

มาทำงานแทนที่ออบเจ็กต์เก่า จึงอาจส่งผลให้การทำงานของแอพพลิเคชั่นนั้นมีผลการทำงานที่ขัดแย้งกับการทำงานเดิมได้

2. จากความจำเป็นที่แอพพลิเคชั่นต้องให้ความร่วมมือในการอิมพลีเมนต์อินเตอร์เฟสสำหรับใช้ในการติดต่อกับระบบ CMC ทำให้แอพพลิเคชั่นที่สามารถนำมาใช้กับงานวิจัยนี้ได้นั้นมีความยืดหยุ่นค่อนข้างน้อย คือ ต้องเป็นแอพพลิเคชั่นที่มีลักษณะการทำงานดังนี้

2.1 ต้องเป็นแอพพลิเคชั่นที่มีการทำงานที่ไม่เป็น Multi-threads เนื่องจากระบบ CMC ยังไม่สามารถรองรับการทำงานของแอพพลิเคชั่นในลักษณะนี้ได้

2.2 ต้องเป็นแอพพลิเคชั่นที่ไม่มีการใช้ไลบรารี่ที่เป็น Swing Component หรือ GUI (Graphic User Interface) เนื่องจากแอพพลิเคชั่นในลักษณะนี้จะมีการเรียกใช้ Component ที่มีลักษณะของ Multi-threads ซึ่งระบบ CMC ยังไม่สามารถควบคุมการทำงานและโอนย้ายสถานะของออบเจ็กต์ที่เป็น Multi-threads ได้

2.3 ต้องเป็นแอพพลิเคชั่นที่มีการทำงานหรือการประมวลผล (Execute) บนคอมพิวเตอร์ประเภทตั้งโต๊ะ (Desktop Computer) โดยแอพพลิเคชั่นนี้จะเป็นโปรแกรมที่มีการทำงานภายใน JVM ซึ่งจะถูกเรียกใช้ที่ Method main ของโปรแกรมและทำงานไปจนกว่า JVM จะถูกจบการทำงาน หรือโปรแกรมสิ้นสุดการทำงาน นั่นหมายถึง ระบบ CMC ยังไม่สามารถรองรับแอพพลิเคชั่นที่มีชนิดเป็น Web Component (แอพพลิเคชั่นประเภท Servlet และ JSP ที่มีการทำงานใน Web Server) และ Applet Component (แอพพลิเคชั่นที่มีการทำงานใน Web browser) ได้

3. คลาสของออบเจ็กต์ใหม่ที่ต้องการนำมาทำงานแทนที่ออบเจ็กต์เก่าควรมีลักษณะการทำงานหรือมีฟังก์ชั่นการประมวลผลที่ให้ผลตรงกันกับคลาสของออบเจ็กต์เก่า เพื่อให้แอพพลิเคชั่นสามารถทำงานได้อย่างถูกต้องภายหลังการเปลี่ยนแปลงที่เกิดขึ้น

6.3 ข้อเสนอแนะเพิ่มเติม

จากข้อจำกัดที่กล่าวมาข้างต้น นับเป็นปัจจัยสำคัญต่อการพัฒนาประสิทธิภาพให้กับการโอนย้ายสถานะการทำงานของออบเจ็กต์ภายในระบบ CMC ซึ่งสามารถนำมาพิจารณาเป็นข้อเสนอแนะเพิ่มเติมในการปรับปรุงและพัฒนาให้ระบบ CMC สามารถสนับสนุนการเปลี่ยนแปลงออบเจ็กต์ได้หลากหลายมากขึ้นต่อไปในอนาคต ดังนี้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

1. มีการทดลองนำแอพพลิเคชั่นที่มีความหลากหลายในการประมวลผล หรือมีการทำงานในรูปแบบต่างๆ เช่น แอพพลิเคชั่นประเภทกราฟฟิก (Graphic User Interface) หรือ Applet มาใช้ เพื่อเป็นแนวทางในการปรับปรุงให้สามารถโอนย้ายสถานะของออบเจ็กต์ในแอพพลิเคชั่นที่มีความซับซ้อนมากขึ้น

2. ควรมีการพัฒนาในส่วนของการตรวจสอบความเข้ากันได้ระหว่างคลาสเก่าและคลาสใหม่ที่จะนำมาทำงานแทนที่ เนื่องจากงานวิจัยนี้ได้ทำการตรวจสอบเฉพาะรายละเอียดพื้นฐานของคลาสเท่านั้น หากมีการตรวจสอบผลการทำงานเพิ่มขึ้น จะช่วยให้มั่นใจได้ว่าคลาสใหม่สามารถทำงานแทนที่คลาสเก่าได้อย่างถูกต้องและสมบูรณ์มากขึ้น

3. มีการปรับปรุงให้ระบบ CMC สามารถรองรับการทำงานของแอพพลิเคชั่นที่มีการทำงานแบบ Multi-threads ได้ โดยการดัดแปลง Hash Map ที่ใช้เก็บข้อมูลสถานะการทำงานของออบเจ็กต์ ให้สามารถเก็บข้อมูลสถานะการทำงานของออบเจ็กต์ที่มีความซับซ้อนของการประมวลผลในแต่ละ Method ได้ เพื่อให้สามารถโอนย้ายสถานะการทำงานของออบเจ็กต์แบบ Multi-threads ได้

4. มีการพัฒนาให้สามารถโอนย้ายสถานะการทำงานของออบเจ็กต์ในแอพพลิเคชั่นที่มีการทำงานในสภาพแวดล้อมของการประมวลผลลักษณะอื่นๆ เช่น Client/Server หรือมีการทำงานในลักษณะ Real time ที่มีการเชื่อมโยงติดต่อกับเครื่องอื่นๆ เพื่อเป็นการขยายขีดความสามารถของระบบ CMC ให้สามารถทำงานบน Application Server ได้อย่างแท้จริง

ข้อเสนอแนะเพิ่มเติมเหล่านี้นับเป็นสิ่งสำคัญที่ควรนำไปพิจารณาเพื่อเป็นแนวทางในการพัฒนาและปรับปรุงให้งานวิจัยนี้มีความสมบูรณ์มากยิ่งขึ้นต่อไปในอนาคต

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บรรณานุกรม

[1] J.P.A. Almeida, M. Wegdam, M. van Sinderen, and L. Nieuwenhuis., "Transparent Dynamic Reconfiguration for CORBA", In **Proceedings of the 3rd International Symposium on Distributed Objects and Applications**, IEEE Computer Society, September 2001, pp. 197--207.

[2] Lpez de Ipia D. and Lo S. "LocALE: a Location-Aware Lifecycle Environment for Ubiquitous Computing", In **Proceedings of the 15th International Conference on Information Networking (ICOIN-15)**, February 2001.

[3] A. Orso, A. Rao, and M. J. Harrold. "A Technique for Dynamic Updating of Java Software". In **Proceedings of the IEEE International Conference on Software Maintenance (ICSM 2002)**.

[4] Jeff Kramer and Jeff Magee, "The Evolving Philosopher Problem: Dynamic Change Management", IEEE TSE 16, 11, November 1990.

[5] Ian Sommerville, **Software Engineering**, Third Edition, Addison-Wesley, 1998, pp. 379-398.

[6] Lakshmikanth s. Jonnalagadda, "The Role of NetworkTraffic Statistics in Devising Object Migration Policies", Master's Thesis, New Brunswick, May. 1997.

[7] Sanjay Dalal, "A Java Container Framework for Server Component Models", eCommerce Server Division, BEA Systems, Inc., pp. 10-16.

[8] http://www.java.sun.com/java_tutorial_3nd/collections

[9] Sheng Liang and Gilad Bracha., "Dynamic Class Loading in the Java™ Virtual Machine", **Annual ACM SIGPLAN Conference (OOPSLA'98)**, Canada, October 1998.

[10] Li Gong, "Secure Java Class Loading", 1069- 780/98 IEEE, November-December 1998, pp. 58-61.

[11] Zhenyu Qian, Allen Goldberg, and Alessandro Coglio, "A Formal Specification of JavaTM Class Loading", Kestrel Institute July 21, 2000.

[12] http://java.sun.com/product/jdk/1.1/docs/guide/reflection

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ภาคผนวก ก.

Source Code ของระบบ CMC

CmcContainer

```
package cmc;
import java.io.*;
import java.util.*;
import java.lang.reflect.*;

public class CmcContainer{

    private final NullContainer parentContainer;

    // component preparation in cmc
    private List registeredComponents = new ArrayList();
    private Map componentTypeToInstanceMap = new HashMap();
    private List orderedComponents = new ArrayList();
    private Map parametersForComponent = new HashMap
    private Class refClass,loadedClass;
    private CmcController monitorLoad = new CmcController();
    private Runtime rtime;

    //test
    private List registeredComponents1 = new ArrayList();
    private List orderedComponents1 = new ArrayList();


    // component operation in cmc
    private boolean initialized;
    private boolean started;
    private boolean destroyed;

    private boolean loadDup,chkEquiv;
    private boolean upgrade;
    private Method mainMethod;
    private FileOutputStream fout;


    public CmcContainer(NullContainer parentContainer) {
        this.parentContainer = parentContainer;
    }

    public static class Default extends CmcContainer {
        public Default() {
            super(new NullContainer());
        }
    }

    public CmcContainer(){this.parentContainer = null;}


    // load class
    public void loadComponent(String appDir){

        String[] allClass = findLoadClass(appDir);
        CmcLoader CL = new CmcLoader(appDir);
        for(int c=0; c<allClass.length; c++){
            try{
                loadedClass = CL.loadClass(allClass[c]);
                System.out.println(loadedClass+ "...Loaded");
                ClassLoader CLApp = loadedClass.getClassLoader();
                System.out.println("loaded by : "+CLApp+"\n");
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
                CL.writeLoadStatus(loadedClass);
                registerComponent(loadedClass);

            }catch(Exception e){
                        e.printStackTrace();
            }
        }
    }

    public void loadUpgradeComponent(String appDir, String upgDir,
            String upgName){
        CmcLoader CL = new CmcLoader(upgDir);
        try{
            loadedClass = CL.loadClass(upgName);
            System.out.println(loadedClass+ "...Loaded");
            ClassLoader CLApp = loadedClass.getClassLoader();
            System.out.println("loaded by : "+CLApp+"\n\n");

            CL.writeLoadStatus(loadedClass);
            registerComponent(appDir,upgDir,loadedClass);
        }catch(Exception e){
            System.out.println(e);
        }
    }

    public String[] findLoadClass(String appDir){
        String[] classDir = findClass(appDir);

        for(int i=0;i<classDir.length;i++){
            if(classDir[i].endsWith(".class")){
                int a = classDir[i].lastIndexOf(".class");
                classDir[i]=classDir[i].substring(0,a);
            }
        }
        return classDir;
    }

    public String[] findClass(String directoryPath) {
        File directory = new File(directoryPath);
        if (directory.exists() && !directory.isDirectory()) {
            throw new IllegalArgumentException(directoryPath +
                " is not a directory");
        }
        String[] entries = directory.list();
        return entries;
    }


    //  register app. class
    public void registerComponent(Class registAppClass) {
        registeredComponents.add(
                new ComponentSpec(registAppClass,registAppClass));
        CmcRegistry registry = new CmcRegistry(registAppClass);
    }


    // register upgrade class
    public void registerComponent(String appDir, String upgradeDir,
            Class registUpgradeClass)throws IOException {
        CmcRegistryUpgrade registry = new
                CmcRegistryUpgrade(upgradeDir,registUpgradeClass);
        System.out.print("\n==> STEP 2 : Check Load Duplicate .......\n");

        if(chkLoadDup(registUpgradeClass)==true){
            System.out.println("Found Load Duplicate!!\n\n");
            String oldRegistInfo =
                registUpgradeClass.toString().substring(6).
                concat(".txt");
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
            String newRegistInfo =
                registUpgradeClass.toString().substring(6).
                concat("Upgrade.txt");

            System.out.print("\n==> STEP 3 : Check Equivalence
                Detail.......\n");
            if(chkEquiv(oldRegistInfo,newRegistInfo)==true){
                System.out.println(
                    "<!!!!!!!!!!!! CAN'T UPGRADE !!!!!!!!!!! ");
                upgrade = false;
            }
            else{
                System.out.println("<============== CAN UPGRADE
                    ==============>\n\n");
                Class appClass = registUpgradeClass;
                registeredComponents1.add(new
                    ComponentSpec(registUpgradeClass,registUpgradeClass));

            }
        }
    }

    public boolean chkLoadDup(Class upgradeClass){
        try{
            if(monitorLoad.chkLoadDuplicate(upgradeClass)==true)
                loadDup = true;
            else
                loadDup = false;
        }catch(Exception e){System.out.println(e);}
        return loadDup;
    }

    public boolean chkEquiv(String oldfile, String newfile){
        Diff diff = new Diff();
        if(diff.doDiff(oldfile,newfile)==true)
            chkEquiv = true;
        else
            chkEquiv = false;
        return chkEquiv;
    }

    // start cmc
    public void start() {
        try{
            if (initialized == false) {
                System.out.println("\n->> Initial components");
                initializeComponents();
                initialized = true;
            }

            if (started == false) {
                System.out.println("\n->> Start components");
                startComponents();
                started = true;
            } else {
                System.out.println("Container Started Already");
            }
        }catch(Exception e){
            System.out.println(e);
        }
    }


    // init component
    private void initializeComponents() {
        boolean progress = true;
        while (progress == true) {
            progress = false;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
            // loop for registered component
            for (Iterator iterator = registeredComponents.iterator();
                iterator.hasNext();) {
                // specify initialized component
                ComponentSpec componentSpec = (ComponentSpec)
                    iterator.next();
                Class componentImplementation =
                    componentSpec.getComponentImpl();
                Class componentType = componentSpec.getComponentType();
                if (componentTypeToInstanceMap.get(componentType) == null)
                    try{
                        progress = joinComponents(componentImplementation,
                            componentType, progress);
                    }catch(Throwable e){
                        System.out.println(e);
                    }
                }
            }
        }

        // component ordering for operating preparation
        protected boolean joinComponents(Class componentImplementation,
            Class componentType, boolean progress) {
        try {
            // get parameter of constructor to be parameter of component
            Constructor[] constructors =
                componentImplementation.getConstructors();
            Constructor constructor = constructors[0];
            Class[] parameters = constructor.getParameterTypes();

            // get param of component in List for check -> if null param,
            // create new empty list
            List paramSpecs = (List)
                parametersForComponent.get(componentImplementation);

            paramSpecs = paramSpecs == null ? Collections.EMPTY_LIST :
                new LinkedList(paramSpecs);

            // create object follow param number of instantiated component
            Object[] args = new Object[parameters.length];
                        // loop for param
            for (int i = 0; i < parameters.length; i++) {
                Class param = parameters[i];
                args[i] = getParamComponent(param);
            }

            // no null arg
            if(hasAnyNullArguments(args) == false){
                Object componentInstance = null;
                // create instance of instantiated component and ordering
                // object for preparation
                componentInstance = constructor.newInstance(args);
                componentTypeToInstanceMap.put(componentType,
                    componentInstance);
                orderedComponents.add(componentInstance);
                progress = true;
            }
        }catch(Throwable e){
            System.out.println(e);
        }
        return progress;
    }

    // create instance of component in cmc & get param of component
    private Object getParamComponent(Class parameter) {
        Object result = null;
        // get param of component in cmc
        if (parentContainer.hasComponent(parameter)) {
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
            return parentContainer.getComponent(parameter);
        }
        // keep track of param from related component for use case not clear
        List candidateClasses = new ArrayList();
        for (Iterator iterator =
                componentTypeToInstanceMap.entrySet().iterator();
                iterator.hasNext();) {
            Map.Entry entry = (Map.Entry) iterator.next();
            Class clazz = (Class) entry.getKey();
            if (parameter.isAssignableFrom(clazz)) {
                candidateClasses.add(clazz);
                result = entry.getValue();
            }
        }
        return result;
    }


    // check object has null arg ?
    private boolean hasAnyNullArguments(Object[] args) {
        for (int i = 0; i < args.length; i++) {
            Object arg = args[i];
            if (arg == null) {
                return true;
            }
        }
        return false;
    }


    // start component in cmc
    protected void startComponents() {
        try{
            for (int i = 0; i < orderedComponents.size(); i++) {
                Object component = orderedComponents.get(i);
                Class clazz = component.getClass();
                hasMainMethod(clazz);
            }
        }catch(Exception e){
            System.out.println(e);
        }
    }

    protected void hasMainMethod(Class clazz){
        CmcInvoker task = new CmcInvoker(clazz);
        Thread thread = new Thread(task);
        thread.setName(clazz.getName());
    }

    // get specify component
    public Object getComponent(Class componentType) {
        Object result = componentTypeToInstanceMap.get(componentType);
        if (result == null) {
            result = parentContainer.getComponent(componentType);
        }
        return result;
    }


//--------------------------------------------------------------------------
    protected void restartComponents(){
        try{
            for (Iterator iterator = registeredComponents1.iterator();
                    iterator.hasNext();) {
                ComponentSpec componentSpec = (ComponentSpec)
                    iterator.next();
                Class componentImplementation =
                    componentSpec.getComponentImpl();
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
                Class componentType = componentSpec.getComponentType();

                if (componentTypeToInstanceMap.get(componentType) == null) {
                    Constructor[] constructors =
                        componentImplementation.getConstructors();
                    Constructor constructor = constructors[0];
                    Class[] parameters = constructor.getParameterTypes();

                    List paramSpecs = (List)
                        parametersForComponent.get(componentImplementation);

                    paramSpecs = paramSpecs == null ? Collections.EMPTY_LIST
                        : new LinkedList(paramSpecs);

                    Object[] args = new Object[parameters.length];

                    for (int i = 0; i < parameters.length; i++) {
                        Class param = parameters[i];
                        args[i] = getParamComponent(param);
                    }

                    if(hasAnyNullArguments(args) == false){
                        Object componentInstance = null;

                        componentInstance = constructor.newInstance(args);
                        componentTypeToInstanceMap.put(componentType,
                            componentInstance);
                        orderedComponents1.add(componentInstance);
                    }
                }
            }

            for (int i = 0; i < orderedComponents1.size(); i++) {
                Object component = orderedComponents1.get(i);
                Class clazz = component.getClass();

                hasMainMethod(clazz);

            }
        }catch(Exception e){
            System.out.println(e);
        }
    }

//---------------------------------------------------------------------------

    public void getOrderedComponents(){
        //for (Iterator iterator = orderedComponents.iterator();
        //iterator.hasNext();) {
        for (Iterator iterator = registeredComponents.iterator();
                iterator.hasNext();) {
            ComponentSpec componentSpec = (ComponentSpec) iterator.next();
            Class componentImplementation =
                componentSpec.getComponentImpl
            System.out.println(componentImplementation.getName());
        }
    }
}
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

CmcController

```
package cmc;
import java.io.*;
import java.util.regex.*;

class monitorFileInfo{

    DataInputStream dFile;
    public int maxLine;

    monitorFileInfo(String filename) {
        try {
            dFile = new DataInputStream(new FileInputStream(filename));
            maxLine = countLines(filename);
        } catch (IOException e) {
            System.err.println("Can't read " +filename+ "\n Error :"+e);

            System.exit(1);
        }
    }

    int countLines(String fileName) throws IOException {
        File file = new File(fileName);
        Reader reader = new InputStreamReader(new FileInputStream(file));

        int lineCount = 0;
        char[] buffer = new char[4096];
        for (int charsRead = reader.read(buffer); charsRead >= 0; charsRead
                = reader.read(buffer)) {
            for (int charIndex = 0; charIndex < charsRead ; charIndex++) {
                if (buffer[charIndex] == '\n')
                    lineCount++;
            }
        }
        reader.close();
        return lineCount;
    }

};


public class CmcController{
    static LineNumberReader loadLine;
    static monitorFileInfo loadInfo;
    static boolean chk=false;

    public CmcController(){}

    public CmcController(String loadfile){
        try{
            loadLine = new LineNumberReader(new FileReader(loadfile));

            loadInfo = new monitorFileInfo(loadfile);
        }catch(Exception e){
            System.out.println(e);
        }

    }

    public static boolean chkLoadDuplicate(Class loadedClass){
            CmcController x = new CmcController("loadFile.txt");
        try {
            String line = null;
            for(int i=0; i<= loadInfo.maxLine; i++){
                while ((line = loadLine.readLine()) != null){
                    Pattern regexp =
                        Pattern.compile(loadedClass.toString().
                        substring(6));
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
                    Matcher matcher = regexp.matcher("");

                    matcher.reset( line ); //reset the input

                    if (matcher.find() ) {
                        System.out.println(" Line" +
                            loadLine.getLineNumber()+ " # " +line);

                        chk = true;
                    }
                    else
                        chk = false;
                }
            }
        } catch(Exception e){
            System.out.println(e);
        }
        return chk;
    }
}
```

CmcInvoker

```
package cmc;
import java.lang.reflect.*;
import java.util.*;

public class CmcInvoker implements Runnable{
    private boolean stopped = true;
    private int count = 0;


    Method foundMethod, targetMethod;


    static List keepObjList = new ArrayList
    public CmcInvoker(Class targetClass){
        try{
            AdminInterface objClass =
                (AdminInterface)targetClass.newInstance();
            keepObjList.add(objClass);
            if(keepObjList.size() >1){  // may increase more for future ***1
                AdminInterface o_objClass =
                    (AdminInterface)keepObjList.get(0);//0
                objClass.setState(o_objClass.getState());
                o_objClass.stop();

            }

            objClass.printState();
            targetMethod = targetClass.getMethod("main", new Class[] {
                String[].class });
            targetMethod.invoke(null,new Object[] {new String[0]});



        }catch(Exception e){}
    }
    public void run(){}
}
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

CmcLoader

```
package cmc;

import java.util.List;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.StringTokenizer;
import java.util.Hashtable;
import java.io.*;

public class CmcLoader extends ClassLoader {
    private Hashtable classes = new Hashtable();
    private List classRepository;
    FileOutputStream fout;

    public CmcLoader(ClassLoader parent,String searchPath) {
        super(parent);
        initLoader(searchPath);
    }

    public CmcLoader(String searchPath) {
        super(CmcLoader.class.getClassLoader());
        initLoader(searchPath);
    }

    protected Class findClass(String className) throws
            ClassNotFoundException {
        byte[] classBytes = loadFromCustomRepository(className);
        if(classBytes != null) {
            return defineClass(className,classBytes,0,classBytes.length);
        }
        //else
        throw new ClassNotFoundException(className);
    }


    private byte[] loadFromCustomRepository(String classFileName) throws
            ClassNotFoundException {
        Iterator dirs = classRepository.iterator();
        byte[] classBytes = null;
        while (dirs.hasNext()) {
            String dir = (String) dirs.next();
            classFileName.replace('.',File.separatorChar);
            classFileName += ".class";
            try {
                File file = new File(dir,classFileName);
                if(file.exists () ) {
                    //read file
                    InputStream is = new FileInputStream(file);
                    classBytes = new byte[is.available()];
                    is.read(classBytes);
                    break;
                }
            } catch(IOException ex) {
                System.out.println("IOException raised while reading class
                    file data");
                ex.printStackTrace();
                            return null;
            }
        }
        return classBytes;
    }


    private void initLoader(String searchPath) {
        classRepository = new ArrayList();
        if( (searchPath != null) && !(searchPath.equals("")) ) {
            StringTokenizer tokenizer = new
```

เอกสารนี้เป็นเอกสาร[illegible]เท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
                StringTokenizer(searchPath,File.pathSeparator);
            while(tokenizer.hasMoreTokens()) {
                classRepository.add(tokenizer.nextToken());
            }
        }
    }

    public void writeLoadStatus(Class classLoaded){
        try{
            fout = new FileOutputStream ("loadFile.txt",true);
            new PrintStream(fout).println
                (classLoaded.toString().substring(6)+ "......." +
                classLoaded.getClassLoader());
            fout.close();
        }catch(Exception e){
            System.out.println(e);
        }
    }
}
```

CmcManager

```
package cmc;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.text.*;
import java.io.*;

public class CmcManager extends JPanel implements ActionListener{

    JTextField dirField = new JTextField();
    JTextField classField = new JTextField();
    String dirName = new String();
    String className = new String();
    //JTextPane pane;

    public CmcManager(){

        // Input new class frame
        JPanel panelInput = new JPanel(new GridLayout(2,2,5,5));

        panelInput.add(new JLabel("New Class Location : ", JLabel.CENTER));
        panelInput.add(dirField);
        panelInput.add(new JLabel("New Class Name : ", JLabel.CENTER));
        panelInput.add(classField);

        dirField.addActionListener(this);
        classField.addActionListener(this);


        panelInput.setBorder( BorderFactory.createCompoundBorder(
            BorderFactory.createTitledBorder("Input New Class"),
            BorderFactory.createEmptyBorder(5,5,5,5)));

        // show loaded class frame
        JTextArea textArea = new JTextArea();
        try{
            Reader reader = new FileReader("loadFile.txt");
            textArea.read(reader, null);
        }catch(Exception e){}
        JScrollPane areaScrollPane = new JScrollPane(textArea);
        areaScrollPane.setVerticalScrollBarPolicy(
            JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
        areaScrollPane.setPreferredSize(new Dimension(350, 350));
        areaScrollPane.setBorder(BorderFactory.createCompoundBorder
            (BorderFactory.createCompoundBorder
```

เอกสารนี้เป็นเอกสาร... ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
            (BorderFactory.createTitledBorder("Loaded Class"),
            BorderFactory.createEmptyBorder(5,5,5,5)),
            areaScrollPane.getBorder()));
        JPanel leftPane = new JPanel(new BorderLayout());
        leftPane.add(panelInput, BorderLayout.PAGE_START);
        leftPane.add(areaScrollPane, BorderLayout.CENTER);
        add(leftPane, BorderLayout.LINE_START);
    }

    public static void display(){
        JFrame frame = new JFrame("Welcome to CMC");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JComponent newContentPane = new CmcManager();
        newContentPane.setOpaque(true);
        //content panes must be opaque
        frame.setContentPane(newContentPane);
        //Display the window.
        frame.pack();
        frame.setVisible(true);
    }


    public void actionPerformed(ActionEvent e) {
        dirName = dirField.getText();
        className = classField.getText();
        try{
            CmcContainer cmc = new CmcContainer.Default();

            System.out.println("\n==> STEP 1 : Load New Class .......");
            cmc.loadUpgradeComponent("appDir",dirName,className);

            cmc.restartComponents();

        }catch(Exception x){}

    }

    public static void main(String[] args){
        display();
    }
}
```

CmcRegistry

```
package cmc;
import java.io.*;
import java.lang.reflect.*;

public class CmcRegistry {

    FileOutputStream fout;
    String nameFile ="";

    public CmcRegistry(Class clazz){
        nameFile = clazz.toString().substring(6).concat(".txt");

        try{
            fout = new FileOutputStream (nameFile,true);
            new PrintStream(fout).println ("Class : "+clazz.getName());
            showField(clazz);
            showConstructor(clazz);
            showMethod(clazz);
            new PrintStream(fout).println ("-----------------------------");
            fout.close();
        }catch(IOException e){
            System.err.println("Unable to write to file");
            System.exit(-1);
        }
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
    }

    public void showField(Class className) {
        try{
            Field fields[] = className.getDeclaredFields();
            fout = new FileOutputStream (nameFile,true);
            new PrintStream(fout).println("There are " + fields.length + "
                fields ");
            for(int i=0;i<fields.length;i++) {
                new PrintStream(fout).println("              Field Name : " +
                    fields[i].getName());
                new PrintStream(fout).println("                    Type : " +
                    fields[i].getType().getName());
            }
        }catch(Exception e){
            System.out.println(e);
        }
    }

    public void showConstructor(Class className) {
        try{
            Constructor constructors[] =
                className.getDeclaredConstructors();
            fout = new FileOutputStream (nameFile,true);
            new PrintStream(fout).println("There are " + constructors.length
                + "          constructors ");
            for(int i=0;i<constructors.length;i++) {
                new PrintStream(fout).println("              Constructor " +
                    i + " :                " + constructors[i].getName()) ;
                showParameter(constructors[i].getParameterTypes());
            }
        }catch(Exception e){
            System.out.println(e);
        }
    }

    public void showMethod(Class className) {
        try{
            Method methods[] = className.getDeclaredMethods();
            fout = new FileOutputStream (nameFile,true);
            new PrintStream(fout).println("There are " + methods.length +
                "          methods          ");

            for(int i=0;i<methods.length;i++) {
                new PrintStream(fout).println("              Method name " +
                    i + " : " + methods[i].getName());
                new PrintStream(fout).println("      ReturnType    #    "
                    + methods[i].getReturnType());
                showParameter(methods[i].getParameterTypes());
            }
        }catch(Exception e){
            System.out.println(e);
        }
    }

    public void showParameter(Class[] params) {
        try{
            fout = new FileOutputStream (nameFile,true);
            for(int i=0;i<params.length;i++) {
                new PrintStream(fout).println("              Parameter "
                    + i + "    >>   " + params[i].getName());
            }
        }catch(Exception e){
            System.out.println(e);
        }
    }
}
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

CmcRegistryUpgrade

```
package cmc;

import java.io.*;
import java.lang.reflect.*;

public class CmcRegistryUpgrade {

    FileOutputStream fout;
    String nameFile = "";

    public CmcRegistryUpgrade(String tempDir, Class newClazz){
        nameFile = newClazz.toString().substring(6).concat("Upgrade.txt");

        try{
            fout = new FileOutputStream (nameFile,true);
            new PrintStream(fout).println ("Class : "+newClazz.getName());
            showField(newClazz);
            showConstructor(newClazz);
            showMethod(newClazz);
            new PrintStream(fout).println ("----------------------------");
            fout.close();
        }catch(IOException e){
            System.err.println("Unable to write to file");
            System.exit(-1);
        }
    }

    public void showField(Class className) {
        try{
            Field fields[] = className.getDeclaredFields();
            //fout = new FileOutputStream ("registFileUpgrade.txt",true);
            fout = new FileOutputStream (nameFile,true);
            new PrintStream(fout).println("There are " + fields.length +
                " fields ");
            for(int i=0;i<fields.length;i++) {
                new PrintStream(fout).println("                Field Name : " +
                    fields[i].getName());
                new PrintStream(fout).println("                Type : " +
                    fields[i].getType().getName());
            }
        }catch(Exception e){
            System.out.println(e);
        }
    }


    public void showConstructor(Class className) {
        try{
            Constructor constructors[] =
                className.getDeclaredConstructors();
            //fout = new FileOutputStream("registFileUpgrade.txt",true);
            fout = new FileOutputStream (nameFile,true);
            new PrintStream(fout).println("There are " + constructors.length
                + " constructors ");
            for(int i=0;i<constructors.length;i++) {
                new PrintStream(fout).println("            Constructor " +
                    i + " : " + constructors[i].getName()) ;
                showParameter(constructors[i].getParameterTypes());
            }
        }catch(Exception e){
            System.out.println(e);
        }
    }
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
    public void showMethod(Class className) {
        try{
            Method methods[] = className.getDeclaredMethods();
            //fout = new FileOutputStream("registFileUpgrade.txt",true);
            fout = new FileOutputStream (nameFile,true);
            new PrintStream(fout).println("There are " + methods.length +
                " methods ");
            for(int i=0;i<methods.length;i++) {
                new PrintStream(fout).println("              Method name " +
                    i + " : " + methods[i].getName());
                new PrintStream(fout).println("      ReturnType     #   " +
                    methods[i].getReturnType());
                showParameter(methods[i].getParameterTypes());
            }
        }catch(Exception e){
            System.out.println(e);
        }
    }

    public void showParameter(Class[] params) {
        try{
            //fout = new FileOutputStream("registFileUpgrade.txt",true);
            fout = new FileOutputStream (nameFile,true);
            for(int i=0;i<params.length;i++) {
                new PrintStream(fout).println("              Parameter " + I
                    + "    >>   " + params[i].getName());
            }
        }catch(Exception e){
            System.out.println(e);
        }
    }
}
```

ComponentSpec

```
package cmc;

public class ComponentSpec {

    private final Class compType;
    private final Class comp;

    public ComponentSpec(final Class compType, final Class comp) {
        this.compType = compType;
        this.comp = comp;
    }

    public Class getComponentType() {
        return compType;
    }

    public Class getComponentImpl() {
        return comp;
    }
}
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

NullContainer

```
package cmc;

//public class NullContainer implements CmcContainerInt {
public class NullContainer {

	public boolean hasComponent(Class componentType) { return false;   }

	public Object getComponent(Class componentType) { return null;     }

	public Object[] getComponents() {return new Object[0];      }

	public Class[] getComponentTypes() {return new Class[0];    }

	public void registerComponent(String componentImplementation) {    }

	public void registerComponent(Class componentImplementation) {     }

	public void registerComponent(Class componentType,
		Class componentImplementation) { }

	public void start() {}
}
```

AdminInterface

```
package cmc;

import java.util.*;

public interface AdminInterface {
	public int state = 0;

	public void printState();
	public Map getState();
	public void setState(Map recieveObjMap);
	public void stop();
}
```

MainCmc

```
package cmc;

import java.lang.reflect.*;
import java.io.*;


public class mainCmc{
	public static CmcManager cmcMgr;

	public static void main(String [] args) {
		CmcContainer cmc = new CmcContainer.Default();
		try {
			System.out.println("\n<-- Load Application For Run-->\n");
			cmc.loadComponent(args[0]);
			System.out.println("\n-------- Start CMC ---------\n");
			cmcMgr.display();
			cmc.start();
		  } catch(Exception e) {
			System.out.println(e);
		  }
	}
}
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ภาคผนวก ข.
ตัวอย่างแอพพลิเคชั่นที่ใช้ในการทดลอง

1. แอพพลิเคชั่น AppDemo2 (คลาสเก่า)

```
//OLD
import java.util.*;
public class AppDemo2 implements cmc.AdminInterface{
     private static Integer x = 0;
     private static Integer y = 0;
     private static Integer z = 0;
     private static Map objMap = new HashMap();
     private static boolean stopped = true;

     public static void main(String[] args){
            stopped = false;
            while(stopped == false){
                  int add1 = Add1.cal1 (x,y,z);
                  int add2 = Add2.cal2 (x,y,z);
                  int add3 = Add3.cal3 (x,y,z);
                  int add4 = Add4.cal4 (x,y,z);
                  System.out.println("Add 1 : "+add1);
                  System.out.println("Add 2 : "+add2);
                  System.out.println("Add 3 : "+add3);
                  System.out.println("Add 4 : "+add4+"\n");
                  pause(2000);
                  x++;
                  y++;
                  z++;
                  objMap.put("x",x);
                  objMap.put("y",y);
                  objMap.put("z",z);
            }
     }

     public static void pause(int millis) {
            int start = (int) System.currentTimeMillis();
            while(((int)System.currentTimeMillis()) < start + millis) {
                  // empty loop body!
            }
     }

     public void printState(){
            System.out.println("OLD");
     }

     public Map getState(){
            return objMap;
     }

     public void setState(Map recieveObjMap){
            x = (Integer)recieveObjMap.get("x");
            y = (Integer)recieveObjMap.get("y");
            z = (Integer)recieveObjMap.get("z");
     }
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
    public void stop(){
        stopped = true;
    }

}
```

```
class Add1{
    public static int cal1(int a, int b, int c){
        int sum = a+b+c;
        return sum;
    }
}

class Add2{
    public static int cal2(int a, int b, int c){
        int sum = a+b+c+1;
        return sum;
    }
}

class Add3{
    public static int cal3(int a, int b, int c){
        int sum = a+b+c+2;
        return sum;
    }
}


class Add4{
    public static int cal4(int a, int b, int c){
        int sum = a+b+c+3;
        return sum;
    }
}
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2. แอพพลิเคชั่น AppDemo2 (คลาสใหม่)

```
//NEW
import java.util.*;
public class AppDemo2 implements cmc.AdminInterface{
	private static Integer x = 0;
	private static Integer y = 0;
	private static Integer z = 0;
	private static Map objMap = new HashMap();
	private static boolean stopped = true;
	//add test fields
	private static int test = 0;
	private static int test1 = 0;

	public static void main(String[] args){
		stopped = false;
		while(stopped == false){
			int add1 = Add1.cal1 (x,y,z);
			int add2 = Add2.cal2 (x,y,z);
			int add3 = Add3.cal3 (x,y,z);
			int add4 = Add4.cal4 (x,y,z);
			System.out.println("Add 1N : "+add1);
			System.out.println("Add 2N : "+add2);
			System.out.println("Add 3N : "+add3);
			System.out.println("Add 4N : "+add4+"\n");
			pause(2000);
			x++;
			y++;
			z++;
			objMap.put("x",x);
			objMap.put("y",y);
			objMap.put("z",z);
		}
	}

	public static void pause(int millis) {
		int start = (int) System.currentTimeMillis();
		while(((int)System.currentTimeMillis()) < start + millis) {
			// empty loop body!
		}
	}

	public void printState(){
		System.out.println("NEW");
	}

	public Map getState(){
		return objMap;
	}

	public void setState(Map recieveObjMap){
		x = (Integer)recieveObjMap.get("x");
		y = (Integer)recieveObjMap.get("y");
		z = (Integer)recieveObjMap.get("z");
	}

	public void stop(){
		stopped = true;
	}
}
```

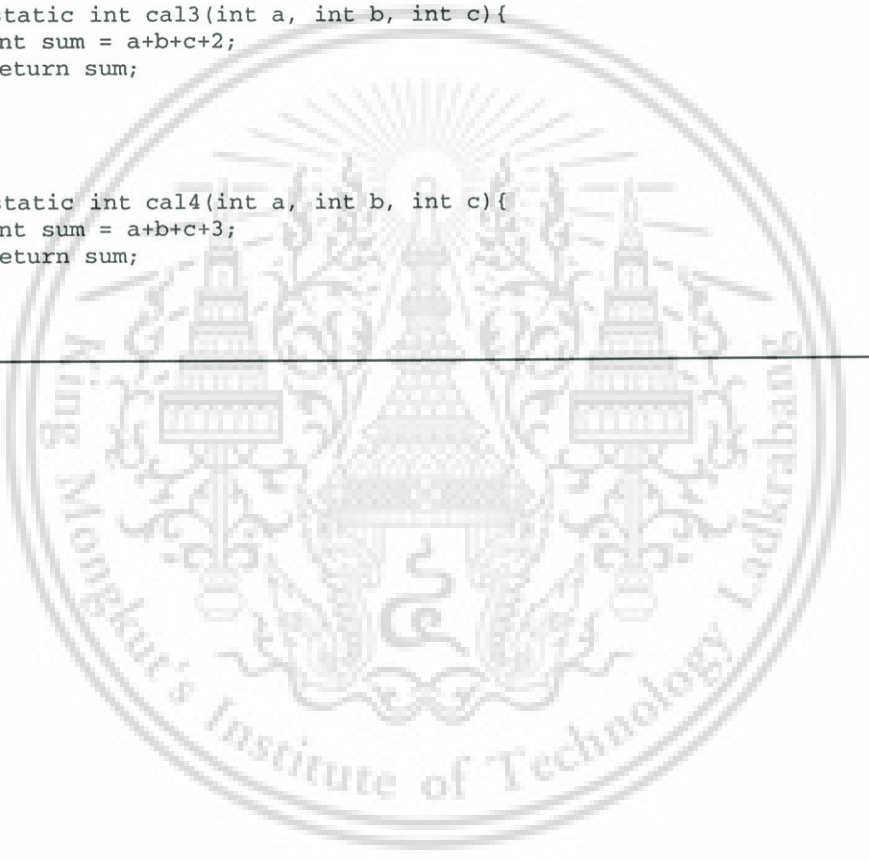
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
class Add1{
     public static int cal1(int a, int b, int c){
            int sum = a+b+c;
            return sum;
     }
}

class Add2{
     public static int cal2(int a, int b, int c){
            int sum = a+b+c+1;
            return sum;
     }
}

class Add3{
     public static int cal3(int a, int b, int c){
            int sum = a+b+c+2;
            return sum;
     }
}

class Add4{
     public static int cal4(int a, int b, int c){
            int sum = a+b+c+3;
            return sum;
     }
}
```



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ภาคผนวก ค.
บทความทางวิชาการที่ได้รับการตีพิมพ์

1. N. Yoosanthiah and A. Khunkitti. "Continuous Migration Container System for Upgrading Object" International Conference on Control, Automation and System (ICCAS2004). August 25-27, 2004 The Shangri-La Hotel, Bangkok, Thailand

2. เนตรสุดา อยู่สันเทียะ และ อัครินทร์ คุณกิตติ. "ระบบคอนเทนเนอร์รักษาความต่อเนื่องในการ Upgrade Object (Continuous Migration Container System for Upgrading Object)." การประชุมวิชาการทางวิศวกรรมไฟฟ้า ครั้งที่ 27 (EECON-27) 11-12 พฤศจิกายน 2547 มหาวิทยาลัยขอนแก่น

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

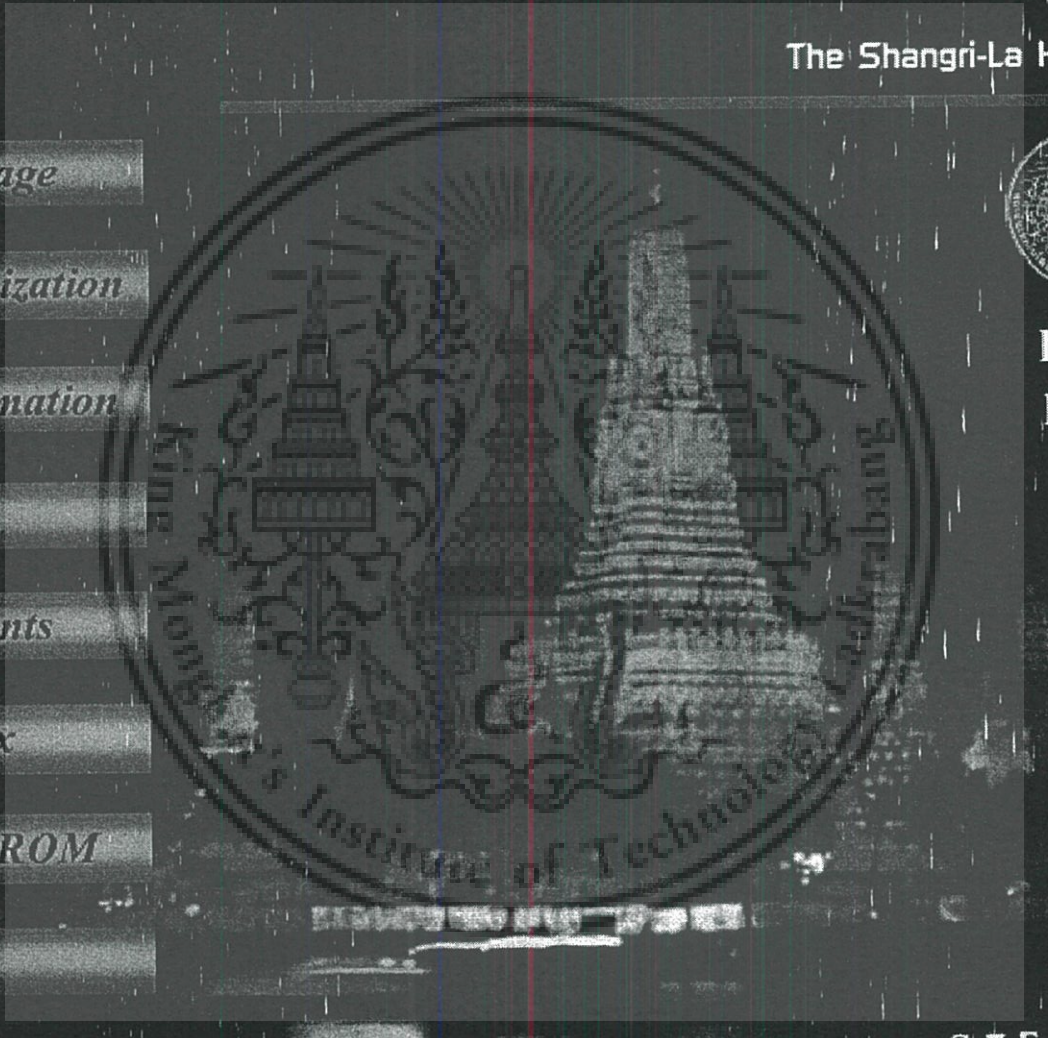
ICCAS '04

ICCAS 2004

2004 International Conference on Control, Automation and Systems

August 25-27, 2004

The Shangri-La Hotel, Bangkok, Thailand



ICASE

http://www.kmitl.ac.th

http://www.iccas.org

- Welcome Message
- Conference Organization
- Conference Information
- Sponsors
- Table of Contents
- Author Index
- Search This CD-ROM
- Exit

SICE CAA AAI ECTI

Continuous Migration Container System for Upgrading Object

N. Yoosanthiah and A. Khunkitti

Faculty of Information Technology
King Mongkut's Institute of Technology Ladkrabang
Ladkrabang , Bangkok 10520 , Thailland
Email: netsuda@yahoo.com and akharin@it.kmitl.ac.th

Abstract: During system resource improvement process that based on Object-Oriented technology could be affect to the continuous system performance if lack appropriate management and control objects mechanism. This paper proposes a methodology to support continuous system performance and its stability. The adoption is based on Java Container Framework and Collections Framework for object collection. Also includes Software Engineering, Object Migration and Multiple Class Loaders mechanism accommodate to construct Continuous Migration Container (CMC). CMC is a runtime environment provides interfaces for management and control to support upgrading object process. Upgrade object methodology of CMC can be divided into two phase are object equivalence checking and object migration process. Object equivalence checking include object behavior verification and functional conformance verification before object migration process. In addition, CMC use Multiple Class Loaders mechanism to support reload effected classes instead of state transfer in migration process while upgrading object. These operations are crucial for system stability and enhancement efficiency.

Keywords: Java Container Framework, Multiple Class Loaders, Continuous Migration Container and Object Equivalence Checking

1. INTRODUCTION

Nowadays the system applicability based on Object-Oriented has been used widespread. In composite with new technology to improve the capability of device or environment to support variety works developed. There are essential needs to enable or enhance the existing resources represent objects to updating system performance. Hence systems must serve management and control the transactions in continuous pattern while the changing occur.

Several researches system stability maintenance in upgrading transition in the following approaches:

Dynamic Load Distribution [1] approach supporting Parallel Objects (PO) programming environment has provide a transparent and dynamic distribution of system load via remote creation and migration of objects. At least one execution thread is associated with each object. This approach have perspective that allocation policies cannot be completely automated, but new created object allocation need also to be directed by user.

The principle of Dynamic Class Loading in JVM [2] provides mechanism for dynamically loading software components on the Java platform. It is often desirable upgrade software component in a long-running application. By organizing software components in separate class loaders and avoid dealing with schema evolution with new classes that loaded by a separate loader. This approach depend on loaded class caches because JVM cannot trust any user-defined load class method to maintain them for mapping class names and initiating loaders to class type. These can affect downtimes that occur from the system that loaded class cache.

Transparent Dynamic Reconfiguration for CORBA [3] approach support distributed system provides the ability to maintain or upgrade without being taken off-line. The mechanisms drive the system under reconfiguration to a safe state, which avoids abortion interactions. This approach focuses on reconfiguration of non-redundant objects and has impact on execution during reconfiguration. There are effects of reconfiguration on the performance of the new object right after this process.

Location-Aware Lifecycle Environment framework (Loc-ALE) [4] provides a simple management interface that control the lifecycle (LCManager) of CORBA distributed objects. Every object creation, movements or deletions requests are routed through LCManager. The advantages are that the LCManager presents object fault-tolerance and mobility support facilities without breaking client held references. But LCManager as a single point of the system, so there are problem of bottleneck within a location domain.

From above, this paper proposes a mechanism of *Continuous Migration Container (CMC)* for maintain stability while upgrading object by use Java Container Framework [6] which provide objects environment based on Collections Framework [7]. Container infrastructure composes management and control interfaces that can be applied to object migration and multiple class loaders that support new object creation and existing object equivalence checking before initiate upgrading process.

This paper is further structured as follows. Section 2 represents continuous upgrading object approach. That is a support guide for management and control objects under *Continuous Migration Container (CMC)* environment implementation, Section 3 describe about infrastructure and operation approach. Finally, the conclusion and future work in Section 4.

2. CONTINUOUS UPGRADING OBJECT APPROACH

This section is a object migration concept which process based on software engineering [5] that provide upgrading object operation within the system compose as Figure 1.

1. Object Creation: newly object (Onew) being created to the existing object (Oold) within the system.

2. Object Validation: functional and operation compatibility validation check point between Onew and Oold that can divided into 2 phase.

2.1 Check Authentication: verify agent username and password permitted to operate.

2.2 Check Equivalence: verify functional equivalence between Onew and Oold consist of:

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Behavior verification is descriptive check both objects regarding the field, parameter, method, modifier and return type to compare similarity.

Execution verification is executed objects operations check within their method and compare results are conformable execution.

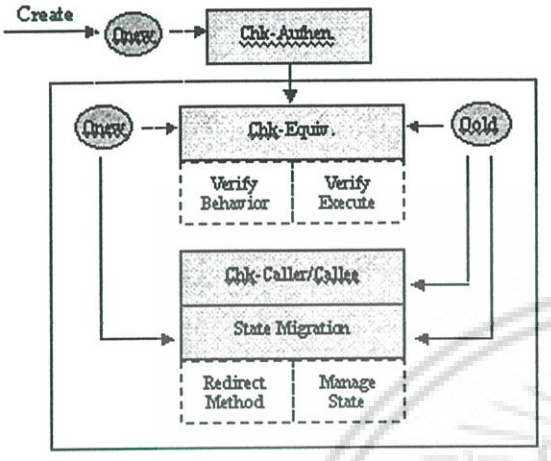


Fig. 1 Object Migration Approach

3. Object Transfer: check effected objects for support state transfer process.

3.1 Check Call Method: check effected objects, which are called or call to Oold's method. Then the system will obtain information about caller and callee objects and their status for support state migration process.

3.2 State Migration: redirect messages and manage call status between effected objects to migrate.

Case of Oold's call methods are waiting for execution, terminated and recall to Onew for proceeds. In the case of call methods are executing, will be checked:

If these executing have long-time operation, pack and migrate method's state for continue executing to Onew.

If their have short-time operation, freeze method execution and return value to related object and sends these status for along continue into Onew.

4. Destroy Old Object: termination of Oold when there are non remaining transactions or reference to Oold.

3. CONTINUOUS MIGRATION CONTAINER (CMC) IMPLEMENTATION

From section 2, the environment to manage and control objects should be provided. This paper proposes *Continuous Migration Container (CMC)* is a runtime environment for manage component execution and maintain continuous performance while upgrading object.

3.1 CMC System

System of CMC is shown in Figure 2 consists of the following elements:

- *Component Manager:* is the central component of CMC that interacts with other component as a Runtime [6] provide interface to the container with infrastructure service like transaction and security. This component is user interface that accessible everywhere in the container and usually works as a single point of access to obtain infrastructure services and other elements of CMC.
- *Component Handler:* usually take a component request as input and invokes it on the target component instance. In addition, provide hook at various well-defined points (e.g. before and after method invocation) and adopted from Reactor pattern [9] for support handler dispatching object's state. This component can be divided into *Reactor* provides interface to call methods and *Event Handler* provides a standard interface for dispatching event object's state, will describe in 3.2.2.
- *Component Meta-Data*: contains information about necessary interface and implementation classes of the components within container. This information include object's signature which useful for object behavior verification in equivalence checking process.
- *Component Registry*: is a place where all the components are registered in the container. Responsible maps a unique key with the component to provide a registry for the location of object.
- *Component Factory*: is an interface for the components. These factories are used to create and find object on behalf of Component Manager. CMC uses factory to manage lifecycle of components.

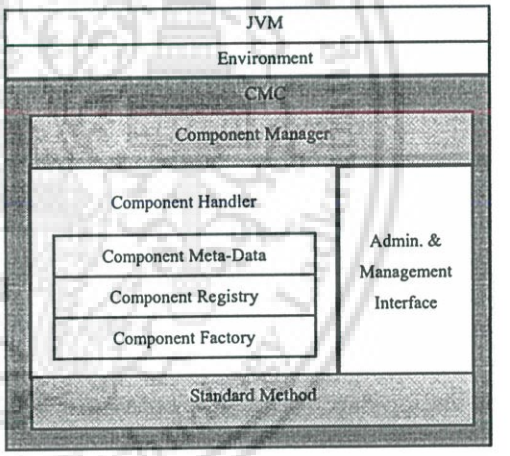


Fig. 2 CMC System

- *Administration & management Interface:* include method to provide equivalence checking to support object migration process.
- *CMC Standard Method:* provide method implementation for support CMC operation together with all of above component.

3.2 Upgrading Object Methodology of CMC

Since newly object creation is completely deployed into CMC and duplicate object discovery checking, the migrating operation for upgrading object perform include the following steps as show in Figure 3.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

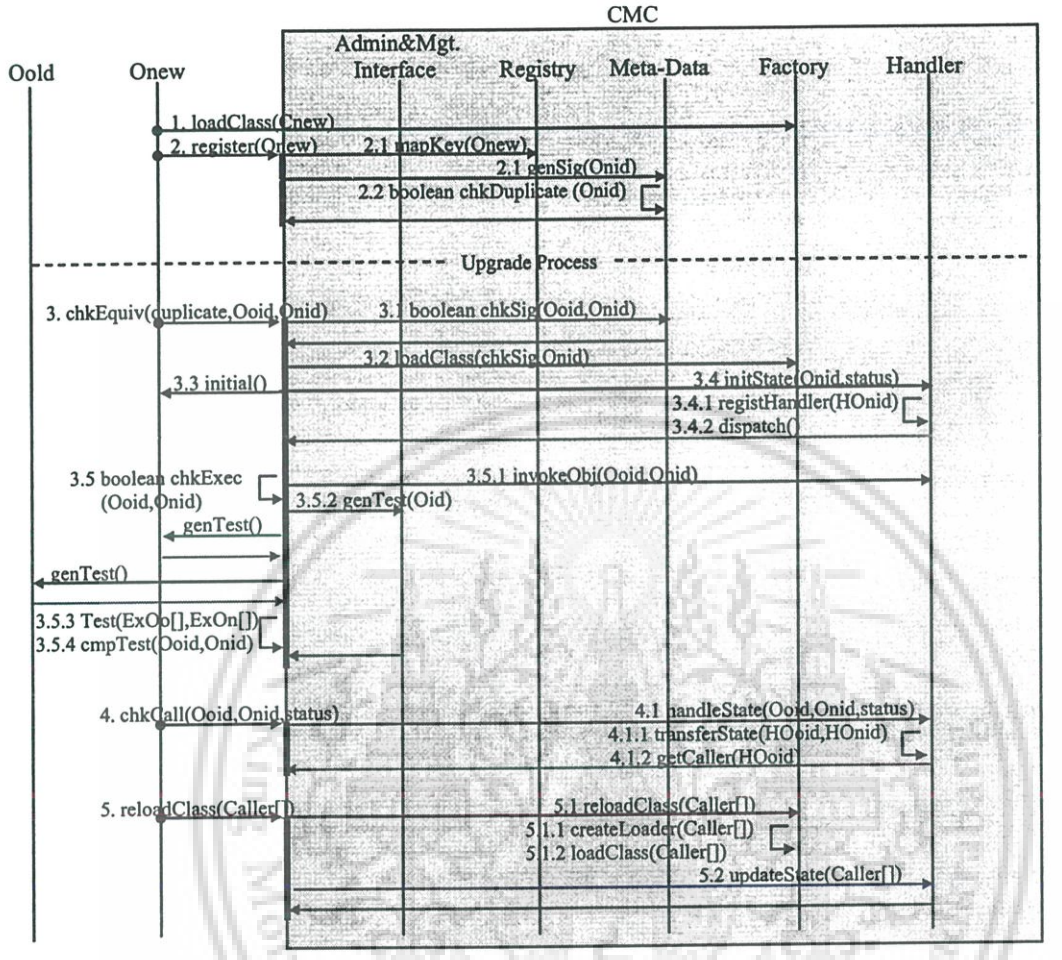


Fig 3. Object Upgrading Process

3.2.1 Check Equivalence

This process provides objects equivalence checking for verify compatibility between Onew and Oold within CMC operation before migration process. In addition, these operations are implemented within objects as interfaces are associated with *Admin. & Management Interface* as Figure 4.

This process can be divided into two partition-checks:

- *Object Behavior Verification*

The first partition-check to determine the compatibility between Onew and Oold directly to get the information about object signature and descriptor e.g. field name, field type, method's name and return type, method's parameter type, constructor's name and parameter type from *Component Meta-Data* to compare. Usually Onew may have any additional signature which may effect to the system but not.

- *Object Functional Conformance Verification*

Since the object behavior process is completed, the next step is to verify functional conformance, which controlled *by CMC Standard Method.* By comparing methods if they give the same direction of result or not. The operation will conduct the test of all methods generation of Oold and Onew through *Administration & Management Interface*. Thus all of objects within CMC, interface for functional conformance checking must be implemented.

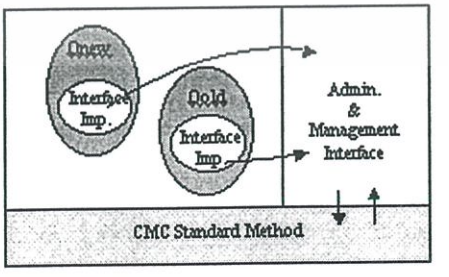


Fig. 4 Interface implementation

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

After test case generation of Oold and Onew complete, CMC will compare result of testing all methods by test generation for check functional conformance.

When the equivalence checking process has been done, it will initiate *Object Migration Process*. This phase perform check effected objects call or called to Oold's method for support state transfer process. Object migration phase compose two steps: check call method and state transfer or reload effected classes.

3.2.2 Check Call Method

After the equivalence checking process has been done, *CMC Standard Method* invoke chkCall() on *Handler* to handle event by using Object Behavioral Pattern [8] and capture state of Oold by Object Serialization.

Since *Component Manager* receive chkCall(), delegate freeze() for enforce to stop execution of CMC and handleState() is invoked for handle state of Oold to check call method as Figure 5.

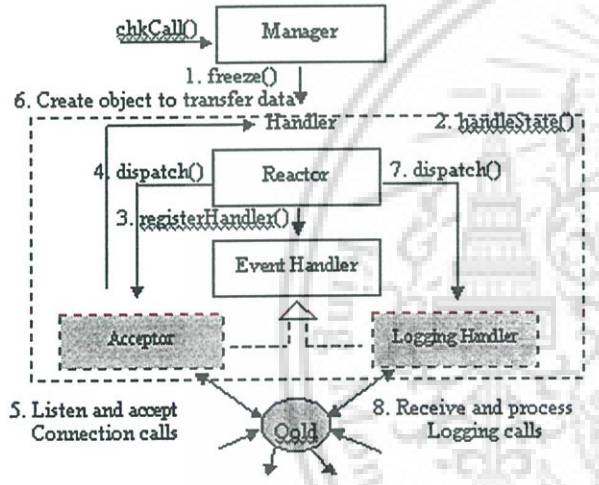


Fig. 5 State Handler for Check Call Method of Oold

Component Handler provide handle Oold's states and event as following:

- *Reactor:* Define an interface for registering, removing and dispatching Event Handler object. An implementation of this interface provides mechanisms perform event demultiplexing and dispatching of Oold's event handlers in response to events.
- *Event Handler:* Define base class specifies an interface use by Reactor to dispatch call methods of Oold defined by objects that are pre-registered to handle event. There are two subclasses: Acceptor and Logging Handler are implements the callback method that process events in object.
- *Acceptor:* provide a factory for creating Logging Handler object to establish connection to Oold for listen and accept connection request of Oold.
- *Logging Handler:* responsible for receiving and processing logging records sent from Oold.

When a logging record arrives, the *Reactor* automatically dispatches the handle event method of the associated Logging Handler object. This object then receives and processes the logging record.

3.2.3 Reload Effected Class

Because of *State Transfer* operation similar redirect messages from effected objects that coming to Oold outgoing to Onew, which can use class loader mechanism. JVM uses class loaders to load class files and create class objects that is a powerful mechanism for dynamically loading software components on the Java platform [9] and responsible to determine when and how classes can be added into a running Java environment [10]. *CMC Standard Method* has method for load class as following algorithm:

```
MyClassloader(String nameClass)
        // check nameClass has been loaded
        if (nameClass = loaded class)
         return previous loaded class
        // check nameClass is system class
       else if (nameClass = system class)
        // primordial class loader perform load class
            Load class from class path
            else
                read class file into array of byte
                construct class object and its method
                specify class loader of nameClass
                load class from class loader
            end if
      end if
end
```

Furthermore CMC's operation use separate class loaders called *Multiple class loaders* for support the key technique is to load the caller class of Oold, class of Oold and class of Onew into separate class loaders. This mechanism provides reload effected classes instead of state transfer in migration process. Thus, this step is called alternative name as *Reload Effected Classes*. CMC perform reload classes operation as Figure 6.

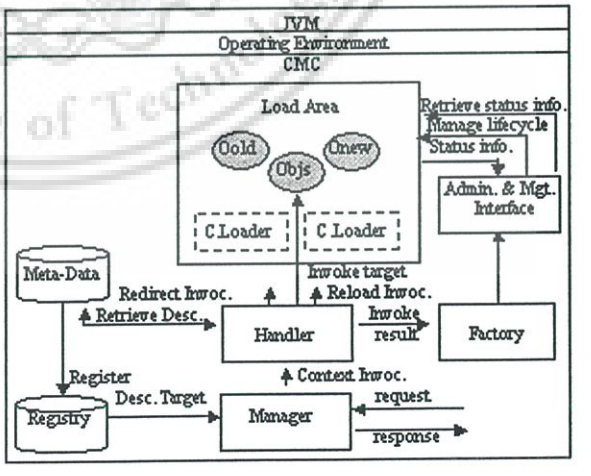


Fig. 6 Reload effected classes of CMC

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Referring to the reload method, the processRequest() redirects all incoming requests to a service object stored in a private field to invoke the "run" method on the service object. In addition, the updateService() allows a new version of Oold (Onew) to dynamically loaded, replacing the existing service object. updateService() callers could supply the location of new class files. All of calls will be redirected to the new object referenced to by services. These methods are the following algorithm:

```
reloadClass()
        private Object service;
        updateService(String location)
        processRequest()
end
updateService(String location)
        create MyClassLoader (location)
        // load class which want to redirect
        Class c = loadClass(nameClass)
        service = create new instance of c
end
processRequest()
        Class c = get class of service
        Method m = get method (run) of c
        invoke m
end
```

After call updateService(), Onew will process all of calls in the future. However the Oold may not have finished processing some of the earlier calls. Thus two classes may coexist for a short period of time. Until all uses of the Oold are completed, all references to Oold are dropped and unloaded by *Admin. & Management Interface* through Garbage Collection.

4. CONCLUSION AND FUTURE WORK

The modification of the existing resource to update and performance enhancement is essential for the systems that based on Object-Oriented technology. Typically, the process may have the impact on continuous of system operation.

This paper proposes mechanisms that provide system stability and continuous performance by CMC construction. This is a runtime environment that equipped interface for continuous upgrading object and support object equivalence checking by objects behavior and functional conformance verification before migration process. In addition, object behavioral pattern and object serialization mechanisms are effectively handle event for capture and restore object's state. Finally, CMC use Multiple Class Loaders mechanism to support reload effected classes instead of state transfer in migration process. These operations can manage and control objects still perform together with newly object and existing object by continuous system performance.

Currently, we work according to interface implementation proposed that can check equivalence object for upgrade process preparation. Simultaneously get up object serialization and multiple class loaders in depth to improve algorithm for captures object states and reload effected classes in accurately the future.

REFERENCES

[1] Antonio Corradi, Letizia Leonardi and Franco Zambonelli, "Dynamic Load Distribution in Massively Parallel Architectures: the Parallel Objects Example", IEEE MPCS'94, May 1994.

[2] Sheng Liang and Gilad Bracha., "Dynamic Class Loading in the Java™ Virtual Machine", Annual ACM SIGPLAN Conference (OOPSLA'98), Canada, October 1998.

[3] J.P.A. Almeida, M. Wegdam, M. van Sinderen, and L. Nieuwenhuis. Transparent Dynamic Reconfiguration for CORBA. In Proceedings of the 3rd International Symposium on Distributed Objects and Applications, IEEE Computer Society, September 2001, pp. 197--207.

[4] Lpez de Ipia D. and Lo S. "LocALE: a Location-Aware Lifecycle Environment for Ubiquitous Computing", Proceedings of the 15th International Conference on Information Networking (ICOIN-15), February 2001.

[5] Ian Sommerville, Software Engineering, Third Edition, Addison-Wesley, 1998, pp. 379-398.

[6] Sanjay Dalal, "A Java Container Framework for Server Component Models", eCommerce Server Division, BEA Systems, Inc., pp. 10-16.

[7] http://www.java.sun.com/java_tutorial_3nd/collections

[8] D. C. Schmidt, "Reactor: An Object Behavioral Pattern for Concurrent Event Demultiplexing and Event Handler Dispatching", in Pattern Languages of Program Design, AddisonWesley, 1995, pp. 529-545.

[9] Li Gong, "Secure Java Class Loading", 1069- 780/98 IEEE, November-December 1998, pp. 58-61.

[10] Zhenyu Qian, Allen Goldberg, and Alessandro Coglio, "A Formal Specification of JavaTM Class Loading", Kestrel Institute July 21, 2000.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

การประชุมวิชาการทางวิศวกรรมไฟฟ้า ครั้งที่ 27

27th Electrical Engineering Conference (EECON 27)

บทความทางวิศวกรทางวิศวกรรมไฟฟ้า

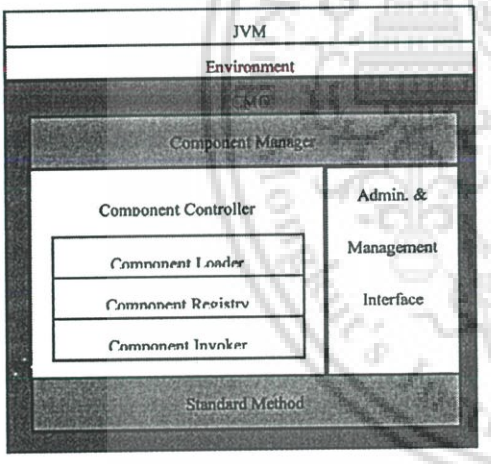
- ไฟฟ้ากำลัง (PW)
- อิเล็กทรอนิกส์กำลัง (PE)
- ไฟฟ้าสื่อสาร (CM)
- อิเล็กทรอนิกส์ (EL)
- การประมวลผลสัญญาณดิจิตอล (DS)
- ระบบควบคุมและการวัดคุม (CT)
- คอมพิวเตอร์และเทคโนโลยีสารสนเทศ (CP)
- งานวิจัยที่เกี่ยวข้องกับวิศวกรรมไฟฟ้า (GN)

ภาควิชาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์ มหาวิทยาลัยขอนแก่น

จากปัญหาดังกล่าว บทความนี้จึงนำเสนอการจัดการและควบคุม Object ต่างๆ ภายใต้สภาพแวดล้อมของระบบคอนเทนเนอร์แบบรักษาความต่อเนื่อง (Continuous Migration Container: CMC) ที่มีการนำเสนอโครงสร้างและการทำงานภายในของระบบขณะมีการนำ Object ใหม่เข้ามาทำงานแทน Object ใดๆ ที่ยังคงมีการทำงานร่วมกับ Object อื่นๆ โดยระบบคอนเทนเนอร์นี้จะสามารถรักษาเสถียรภาพและความปลอดภัยในการเปลี่ยนแปลง Object ที่เกิดขึ้นได้

2. ระบบ CMC (Continuous Migration Container)

เมื่อพิจารณาสภาพการทำงานร่วมกันระหว่าง Object ตามกระบวนการรักษาความต่อเนื่องให้กับการทำงานแล้วพบว่าควรจัดให้ Object ต่างๆ อยู่ในสภาพแวดล้อมเดียวกัน เพื่อให้สามารถจัดการและควบคุมการทำงานได้อย่างมีประสิทธิภาพ บทความนี้จึงได้นำโครงสร้างของระบบคอนเทนเนอร์ของจาวา (Java Container Framework)[5] ที่มีการนำมาใช้ควบคุมและจัดการการทำงานของ Component ที่อยู่ภายในตามหลักการของ Collections Framework [6] มาใช้สำหรับออกแบบระบบคอนเทนเนอร์แบบรักษาความต่อเนื่อง (Continuous Migration Container : CMC) ซึ่งมีโครงสร้างภายในระบบดังรูปที่ 1



รูปที่ 1 โครงสร้างของ CMC

1. Component Manager : เป็น User Interface ทำหน้าที่ติดต่อกับส่วนอื่นๆ ภายใน CMC เพื่อให้ผู้ใช้สามารถเข้าถึงการทำงานของคอนเทนเนอร์ได้

2. Component Controller: เป็นส่วนสนับสนุนการตรวจจับเหตุการณ์และสถานะการทำงานและสถานะการเรียกใช้ของ Object ต่างๆ ภายใน CMC

3. Component Loader: ทำหน้าที่โหลด Object ต่างๆ เมื่อทำการนำ Object เหล่านั้นเข้าสู่การทำงานของ CMC

4. Component Registry: ทำหน้าที่จัดเก็บข้อมูลเกี่ยวกับการ implement คลาสต่างๆ ของ component และรายละเอียด (Signature) ของ Object ต่างๆ ที่มีการทำงานภายใน CMC

5. Component Invoker: ทำหน้าที่ควบคุมการทำงานของ Object เป้าหมายตามที่ได้รับคำสั่ง

6. Administration and Management Interface: เป็นอินเตอร์เฟสสำหรับให้ Object ต่าง ๆ ภายใน CMC สามารถเรียกใช้กรณีที่มีการเปลี่ยนแปลง Object

7. Standard Method: เป็นส่วนที่รวบรวม method ของ CMC เพื่อใช้สำหรับการทำงานที่เกี่ยวข้องกับ CMC โดยตรงทั้งหมด

3. การทำงานของ CMC เมื่อมีการ Upgrade Object

การนำ Object ใหม่ (Onew) เข้ามาทำงานแทนที่ Object ที่ยังคงทำงานอยู่ภายในระบบ (Oold) นั้น เมื่อนำ Onew เข้าสู่ CMC จะทำการเก็บข้อมูลรายละเอียดพื้นฐานของ Object นั้นหากตรวจสอบพบว่า Onew มาจากกำเนิดเดียวกับ Oold จะเข้าสู่กระบวนการ Upgrade Object ที่มีการทำงานดังรูปที่ 2 ซึ่งแบ่งเป็นขั้นตอนต่าง ๆ ดังนี้

3.2.1 การตรวจสอบความเข้ากันได้ระหว่าง Object (Object Equivalence Checking)

เป็นการตรวจสอบคุณลักษณะและผลการทำงานระหว่าง Oold และ Onew ว่ามีความสอดคล้องกันหรือไม่ก่อนที่ Onew จะทำงานแทนที่ Oold โดย Standard Method จะทำการควบคุมกระบวนการตรวจสอบความเข้ากันได้ระหว่าง Oold และ Onew ผ่าน Admin. & Management Interface ซึ่งภายในแต่ละ Object จะมี method สำหรับตรวจสอบภายในตัวเอง (Interface Implementation) การตรวจสอบในขั้นตอนนี้แบ่งออกเป็น 2 ส่วน คือ

1. การตรวจสอบรายละเอียดพื้นฐาน (Verify Object Behavior) เป็นการตรวจสอบ Signature ซึ่งประกอบด้วยชื่อและชนิดของ Method ชนิดของค่าที่จะถูกส่งกลับโดย method จำนวนและชนิดของ parameter ของ Method ชื่อของ Constructor จำนวนและชนิดของ parameter ของ Constructor รวมทั้ง ชื่อและชนิดของ Field เพื่อนำข้อมูลที่ได้มาทำการเปรียบเทียบว่า มีรายละเอียดพื้นฐานตรงกันหรือไม่

2. การตรวจสอบผลการทำงาน (Verify Functional Conformance) เป็นการตรวจสอบผลการทำงานซึ่งจะถูกควบคุมโดย CMC Standard Method ที่จะทำการทดสอบการทำงานของ method ทั้งหมดของ Oold และ Onew ดังนั้น Object ทั้งหมดที่มีการทำงานภายใน CMC จึงจำเป็นต้องมีการ implement interface สำหรับสร้าง Test Case ของตัวมันเองพร้อมทั้งตรวจสอบผลการทำงานจาก Test Case ที่ Object นั้นๆ สร้างขึ้นซึ่งถูกควบคุมโดย CMC Standard Method ที่จะทำการติดต่อกับ Object ทั้งสองผ่าน Admin. & Management Interface

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
reloadClass()
        private Object service;
        updateService(String location)
            create MyClassLoader (location)
            // load class which want to redirect
            Class c = loadClass(nameClass)
            service = create new instance of c
        processRequest()
            Class c = get class of service
            Method m = get method (run) of c
            invoke m
end
```

processRequest() จะ Redirect request ทั้งหมดที่กำลังจะเข้ามาไปยัง service object ซึ่ง Onew จะถูก load แบบ dynamic แทนที่ Oold หลังจากเรียกใช้ updateService() และ request ใหม่จะถูกประมวลผลโดย Onew โดยอาจจะมีบาง request ที่ยังประมวลผลใน Oold ในส่วนนี้ระบบ CMC จะสนับสนุนให้ Onew และ Oold ยังคงทำงานร่วมกันไปจนกว่าการใช้ Oold จะเสร็จสมบูรณ์และทำการ drop Oold โดย Garbage Collection เมื่อไม่มีการทำงานใดที่อ้างอิงไปยัง Oold ซึ่งนับเป็นขั้นตอนสุดท้ายของกระบวนการ Upgrade Object

4. สรุป

ระบบการทำงานที่มีความต้องการเปลี่ยนแปลงทรัพยากรเดิมที่มีอยู่ให้มีความทันสมัยเพื่อรองรับเทคโนโลยีและเพิ่มประสิทธิภาพให้มากขึ้นนั้น โดยทั่วไปอาจจำเป็นต้องหยุดพักการทำงานเพื่อทำการเปลี่ยนแปลง ซึ่งมักส่งผลให้ไม่สามารถดำเนินงานได้อย่างต่อเนื่อง

บทความนี้เป็นการนำเสนอวิธีการรักษาความต่อเนื่องให้กับการทำงาน โดยการจัดสภาพแวดล้อมของ Object ให้มีการทำงานภายใต้ CMC ซึ่งเป็นระบบคอนเทนเนอร์ที่สามารถรักษาความต่อเนื่องให้กับการทำงานในระหว่างการ Upgrade Object โดยมีการใช้อินเตอร์เฟสสำหรับจัดการและควบคุมการตรวจสอบความเข้ากันได้ระหว่าง Object ทั้งสองเพื่อให้สามารถทำงานได้สอดคล้องกันและใช้กลไกของ Multiple Class Loaders ในการโหลด Object ซึ่งจะทำให้ Object ใหม่สามารถทำงานพร้อมกับ Object เดิมได้อย่างทันที ซึ่งนับเป็นวิธีการหนึ่งที่สามารถนำมาปรับใช้กับระบบการทำงานประเภทอื่น ๆ ได้เป็นอย่างดี

สำหรับงานในขณะนี้อยู่ในระหว่างการอิมพลีเมนต์ในส่วนของการตรวจสอบการเรียกใช้ Object ภายใน CMC เพื่อเตรียมการ Reload Class ตามกลไกของ Multiple Class Loaders

เอกสารอ้างอิง

[1] Antonio Corradi, Letizia Leonardi and Franco Zambonelli, "Dynamic Load Distribution in Massively Parallel Architectures: the Parallel Objects Example", IEEE MPCS'94, May 1994.

[2] Sheng Liang and Gilad Bracha., "Dynamic Class Loading in the Java™ Virtual Machine", Annual ACM SIGPLAN Conference (OOPSLA'98), Canada, October 1998.

[3] J.P.A. Almeida, M. Wegdam, M. van Sinderen, and L. Nieuwenhuis. Transparent Dynamic Reconfiguration for CORBA. In Proceedings of the 3rd International Symposium on Distributed Objects and Applications, IEEE Computer Society, September 2001, pp. 197--207.

[4] Ian Sommerville, Software Engineering, Third Edition, Addison-Wesley, 1998, pp. 379-398.

[5] Sanjay Dalal, "A Java Container Framework for Server Component Models", eCommerce Server Division, BEA Systems, Inc., pp. 10-16.

[6] http://www.java.sun.com/java_tutorial_3nd/collections

[7] D. C. Schmidt, "Reactor: An Object Behavioral Pattern for Concurrent Event Demultiplexing and Event Handler Dispatching", in Pattern Languages of Program Design, AddisonWesley, 1995. pp. 529-545.

[8] Li Gong, "Secure Java Class Loading", 1069- 780/98 IEEE, November-December 1998, pp. 58-61.

[9] Zhenyu Qian, Allen Goldberg, and Alessandro Coglio, "A Formal Specification of JavaTM Class Loading", Kestrel Institute July 21, 2000.

อัครินทร์ คุณกิตติ : อาจารย์ประจำคณะเทคโนโลยีสารสนเทศ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

งานวิจัย : - Computer Systems
- Distributed Systems
- Data Communication and Network

เนตรสุดา อยู่สันเทียะ : จบปริญญาตรีสาขาเทคโนโลยีสารสนเทศเพื่ออุตสาหกรรม สถาบันเทคโนโลยีพระจอมเกล้าพระนครเหนือ

ปัจจุบัน นักศึกษาระดับปริญญาโท คณะเทคโนโลยีสารสนเทศ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

งานวิจัย : Continuous Migration Container System for Upgrading Object

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ประวัติผู้เขียน

ชื่อ – นามสกุล	นางสาวเนตรสุดา อยู่สันเทียะ
ที่อยู่	3077 ถ.สืบศิริ ต.ในเมือง อ.เมือง จ.นครราชสีมา 30000
ประวัติการศึกษา	ระดับปริญญาตรีอุตสาหกรรมศาสตรบัณฑิต (เกียรตินิยมอันดับ 2)
	สาขาเทคโนโลยีสารสนเทศเพื่ออุตสาหกรรม
	คณะเทคโนโลยีและการจัดการอุตสาหกรรม
	สถาบันเทคโนโลยีพระจอมเกล้าพระนครเหนือ ปีการศึกษา 2541

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้