A^2V: AN ALTERNATIVE ACTIVE VIEW CONSTRUCTION BASED ON DECENTRALIZED CONTROL

CHAINARONG KONGCHOKE

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR
THE DEGREE OF MASTER OF SCIENCE
(COMPUTER SCIENCE)
FACULTY OF GRADUATE STUDIES
MAHIDOL UNIVERSITY
2006

ISBN 974-04-7409-8
COPYRIGHT OF MAHIDOL UNIVERSITY

Copyright by Mahidol University

Thesis
entitled

A^2V: AN ALTERNATIVE ACTIVE VIEW CONSTRUCTION BASED ON DECENTRALIZED CONTROL

was submitted to the Faculty of Graduate Studies, Mahidol University
for the degree of Master of Science (Computer Science)

on

May 16, 2006

Chainarong Kongchoke
.....
Mr. Chainarong Kongchoke
Candidate

Thanwadee Sunetnanta
.....
Asst.Prof. Thanwadee Sunetnanta, Ph.D.
Chair

Jarernsri L. Mitrpanont
.....
Assoc.Prof. Jarernsri L. Mitrpanont, Ph.D.
Member

.....
Asst.Prof. Charnyote Pluempitiwiriyawej, Ph.D.
Member

.....
Asst.Prof. Yongyuth Permpoontanalarp, Ph.D.
Member

Salee Kiewkarnka
.....
Asst. Prof. Salee Kiewkarnka, Ph.D
Acting Dean
Faculty of Graduate Studies
Mahidol University

.....
Prof. Amaret Bhumiratana, Ph.D.
Dean
Faculty of Science
Mahidol University

Copyright by Mahidol University

Thesis
entitled

A^2V: AN ALTERNATIVE ACTIVE VIEW CONSTRUCTION BASED ON DECENTRALIZED CONTROL

Chainarong Kongchoke
.....
Mr. Chainarong Kongchoke
Candidate

Thanwadee Sunetnanta
.....
Asst.Prof. Thanwadee Sunetnanta, Ph.D.
Major-Advisor

Jarernsri L. Mitrpanont
.....
Assoc.Prof. Jarernsri L. Mitrpanont, Ph.D.
Co-Advisor

Salee Kiewkarnka
.....
Asst. Prof. Salee Kiewkarnka, Ph.D
Acting Dean
Faculty of Graduate Studies

Supachai Tangwongsarn
.....
Assoc.Prof. Supachai Tangwongsarn, Ph.D.
Chair
Master of Science Programme
in Computer Science
Faculty of Science

Copyright by Mahidol University

ACKNOWLEDGEMENT

I am most grateful to my respectful advisor Asst. Prof. Thanwadee Sunetnanta for her motivation, worthful guidance, deep encouragement, very strong patience, and all memorable advice, given to me through such very long time of this thesis. My thanks I would like to give her are actually beyond any thankful words. I would not be able to succeed in this thesis without her numberless help.

Also, I am deeply thankful to Assoc. Prof. Jarernsri L. Mitrpanont, Asst. Prof. Charnyote Pluempitiwiriyawej, and Asst. Prof. Yongyuth Permpoontanalarp for their comments and valuable suggestions on this thesis.

I would like to show my deep appreciation to MUCC officers, Miss Thunyathorn Sutthijaroen and Miss Jarussri Pakkadthong, for such great contact points, and dealing with all paper work for me especially during the extended period. Thanks Department of Computer Science, Mahidol University for useful materials.

My special thanks are extended to the following colleagues, Mr. Aswit Peungsema and Mrs. Lalitha Sivakumar for helping me through the phase of thesis implementation, and Miss Pitchaya Suraprapaphit and Miss Busaba Sangbuathongvilai for helping prepare resources for conducting the experiments. Million thanks also go to Mr. Olarn Wattanasawang for his understanding and implicitly allowing me to complete the thesis during the business hours.

I also wish to convey my big thanks to Miss Supotchanee Nakwichian, Miss Tussawan Kulsiriwiwat, Miss Chantima Wungdeelert, and Mr. Sathitpong Krajangrat for so many nice, cheerful, meaningful words when I was weary and got nervous breakdown.

Finally and foremost, I am forever grateful to my parents, the 2nd lieutenant Vichian and Mrs. Montha Kongchoke, my grandmother, and my mother's sister, who are the wind beneath my wings, for their everlasting love, understanding, warmth, inspiration, and constant encouragement. I was blessed because I was loved by them. And also, thanks the hero lying in me and making it happen.

Chainarong Kongchoke

Copyright by Mahidol University

A^2V: AN ALTERNATIVE ACTIVE VIEW CONSTRUCTION BASED ON DECENTRALIZED CONTROL.
CHAINARONG KONGCHOKE 4337419 SCCS/M
M.Sc. (COMPUTER SCIENCE)
THESIS ADVISORS: THANWADEE SUNETNANTA, Ph.D., JARERNSRI L. MITRPANONT, Ph.D.

ABSTRACT

Active Views offers an efficient mechanism to provide Web Access Control. Each individual Active View represents a view of a web page, with certain restrictions on the display of transactional data and business activities according to the user's access rights. The contents of an Active View are dynamically adjusted in the event of either change of data or other particular action performed by a user.

The construction of Active Views is controlled by the application of Active Rules. However, the typical execution of Active Rules is generally in a manner of centralized structural control. In other words, *all relevant components are deployed as a centered presence at a server side*. This apparently causes a low performance of Active View construction process since the process at a server side has to sort the entire workflow out alone. Apart from that, there can be a high bottleneck when the number of user sessions arises, thereby reducing the overall performance since network bandwidth and server workload are shared among all end-users.

This thesis proposes an alternative Active View construction approach, namely the A^2V framework, *adapting the principle of Active XML to reform the construction process of Active Views by departing some portions to be executed in a decentralized style*. To do that, we developed an XML-based language, *named AVML (short for Active Views Markup Language)*, which is designed to separately describe data and a set of Active Rules affecting the business logic of an individual Active View. An AVML-formatted document is also termed an *Active Web Page* since part of such a web page can be actively executed to construct Active Views.

AVML works with two AVML processing units, the *AV Client* and the *AVML Processor*, which handle part of the Active View construction process at a client side and a server side respectively. The AV Client and the AVML Processor collaboratively work to construct Active Views.

As a proof-of-concept, both critical analysis and some hands-on experiments on the A^2V framework were conducted. The critical analysis compared the formulated response times of the A^2V framework against those of the existing Active View construction approaches. The hands-on experiments tested the response times of the A^2V framework in comparison to traditional web applications development approaches by using a demonstrated case study. The results of both the critical analysis and the hands-on experiments revealed that an improvement of a response time used for constructing an Active View will be achieved if some communication processes between clients and servers are eliminated. In other words, the more we can distribute part of Active Views construction process to be executed at a client side, the better the performance of the construction process is.

KEY WORDS: E-SERVICES / ACCESS CONTROL / DECENTRALIZED VIEW CONSTRUCTION / ACTIVE VIEWS MARKUP LANGUAGE
323 P. ISBN 974-04-7409-8

Copyright by Mahidol University

เอทูวี: ทางเลือกในการสร้างแอคทีฟวิวโดยวิธีควบคุมแบบกระจาย (A^2V: AN ALTERNATIVE ACTIVE VIEW CONSTRUCTION BASED ON DECENTRALIZED CONTROL).

ชัยณรงค์ กองโชค 4337419 SCCS/M
วท.ม. (วิทยาการคอมพิวเตอร์)
คณะกรรมการควบคุมวิทยานิพนธ์ : ธันวดี สุเนตนันท์ Ph.D., เจริญศรี มิตรภานนท์ Ph.D.

บทคัดย่อ

แอคทีฟวิว (Active View) เป็นเทคนิคการควบคุมการแสดงผลของข้อมูลธุรกรรมและกิจกรรมต่างๆ ในเว็บเบราเซอร์บนพื้นฐานสิทธิการใช้งานของผู้ใช้แต่ละคน เนื้อหาของแอคทีฟวิวสามารถปรับเปลี่ยนไปตามการเปลี่ยนแปลงของข้อมูลหรือการกระทำของผู้ใช้ที่ใช้งานผ่านเว็บเบราเซอร์ กระบวนการสร้างแอคทีฟวิวเกิดขึ้นโดยการประมวลผลของ Active Rules ซึ่งโดยทั่วไปถูกควบคุมให้เป็นไปในลักษณะรวมศูนย์ (Centralized control) กล่าวคือ Component ทั้งหมดที่เกี่ยวข้องในการประมวลผลเพื่อสร้างแอคทีฟวิวจะถูกวางให้ทำงานที่ Server side เพียงแห่งเดียว จึงก่อให้เกิดปัญหาความล่าช้าในการประมวลผลเป็นอย่างมาก นอกจากนั้นยังก่อให้เกิดปัญหาคอขวดในการประมวลผลเมื่อมีจำนวนผู้ใช้งานเพิ่มสูงขึ้น เพื่อแก้ปัญหาดังกล่าว งานวิจัยนี้จึงได้นำเสนอทางเลือกใหม่ที่เรียกว่าเอทูวี (A^2V) เพื่อปรับปรุงประสิทธิภาพของกระบวนการสร้างแอคทีฟวิวแก่ผู้ใช้ให้ดีขึ้น โดยงานวิจัยนี้ได้ประยุกต์หลักการ Active XML โดยปรับเปลี่ยนรูปแบบการควบคุมการประมวลผลให้เป็นแบบกระจาย (Decentralized control) โดยย้ายงานส่วนการสร้างแอคทีฟวิวไปประมวลผลที่ Client side และเพื่อให้สามารถประมวลผลแบบกระจายได้ งานวิจัยนี้จึงเสนอภาษาใหม่ซึ่งมีพื้นฐานมาจากภาษา XML เรียกว่า AVML (Active Views Markup Language) ในการกำหนดข้อมูลธุรกรรมและชุดข้อมูล Active Rules ที่สอดคล้องกับตรรกะทางธุรกิจของแต่ละแอคทีฟวิว นอกจากนั้นงานวิจัยยังนำเสนอ Active Web Page ซึ่งเป็นเอกสารที่ลักษณะสอดคล้องตามข้อกำหนดของภาษา AVML โดยส่วนของ Web page ที่อยู่ใน Active Web Page สามารถนำไปประมวลผลเพื่อสร้างแอคทีฟวิวได้

หน่วยประมวลผลสำคัญ 2 หน่วยที่ทำหน้าที่ประมวลผลเอกสารที่สอดคล้องกับข้อกำหนดของ AVML ได้แก่ AV Client ซึ่งอยู่ที่ Client side และ AVML Processor ซึ่งอยู่ที่ Server side ทั้ง AV Client และ AVML Processor จะทำงานร่วมกันในการประมวลผลเพื่อสร้างแอคทีฟวิว

ในการพิสูจน์หลักการของเอทูวี งานวิจัยนี้ได้เสนอการวิเคราะห์อย่างวิกฤตโดยคำนวณค่า Response time ในทางทฤษฎีของเอทูวีเทียบกับค่าที่ได้จากงานวิจัยเดิมซึ่งมีการควบคุมการสร้างแอคทีฟวิวแบบ Centralized control นอกจากนั้นยังได้เสนอการพิสูจน์โดยทำการทดสอบภายใต้กรณีศึกษาเดียวกันเพื่อวัดค่า Response time ที่ได้จากเอทูวีเทียบกับแนวทางการพัฒนาเว็บแอพพลิเคชั่นที่มีอยู่เดิม และผลลัพธ์ที่ได้จากการพิสูจน์ทั้ง 2 แบบทำให้ทราบว่าหากทำให้ Communication process ระหว่าง Client และ Server ลดลง พบว่าทำให้ Response time ในการสร้างแอคทีฟวิวดีขึ้น และหากสามารถสร้างแอคทีฟวิวที่ Client side ได้จะทำให้ประสิทธิภาพโดยรวมของการประมวลผลเพื่อสร้างแอคทีฟวิวดีขึ้น

323 หน้า. ISBN 974-04-7409-8

Copyright by Mahidol University

CONTENTS

		Page
ACKNOWLEDGEMENT		iii
ABSTRACT		iv
LIST OF TABLES		xi
LIST OF FIGURES		xiii
CHAPTER		
I	**INTRODUCTION**	1
	1.1 Introduction to Web Access Control	1
	1.2 Motivation of Using Active Views	2
	1.3 Problem Statement	3
	1.4 Our Approach	3
	1.5 Thesis Objectives	4
	1.6 Scope of Work	5
	1.7 Thesis Outline	5
II	**LITERATURES REVIEW**	7
	2.1 The Definition of Active Rules	7
	2.2 A Review of Related Works in Existing Applications of Active Rules to Implement Active Views for E-Services	8
	2.2.1 What Are Active Views?	8
	2.2.2 Active Views for Electronic Commerce by Serge et al.	8
	2.2.3 Active View Agents System for Electronic Commerce (AVAS) by Kesorn	11

Copyright by Mahidol University

CONTENTS (CONT.)

		Page
	2.2.4 ActiveWeb: XML-Based Active Rules for Web View Derivation and Access Control by Hidenari et al.	14
	2.2.5 Active Rules for XML: A New Paradigm for e-Services by Angela et al.	19
	2.2.6 Pushing Reactive Services to XML Repositories Using Active Rules by Angela et al.	22
	2.2.7 Comparison of Related Works	24
	2.3 Related Technologies	28
	2.3.1 Active XML	28
	2.3.3.1 An Introduction of Active XML	28
	2.3.3.2 Active XML Peer-to-Peer Architecture	30
	2.3.3.3 Active XML Specifications	32
	2.3.2 XML Schema	35
	2.3.3 XSL	35
	2.3.4 XPath	36
	2.3.5 XLink	36
III	**A^2V FRAMEWORK**	38
	3.1 An Overview of A^2V: Distributed Control versus Centralized Control	38
	3.2 Applying Active XML to A^2V Framework	43
	3.3 Active Web Page Structure	45
	3.4 Active Views Markup Language (AVML)	51
	3.5 A^2V Architecture	63
	3.5.1 AV Client Architecture	64
	3.5.2 AVML Processor Architecture	67
	3.6 A^2V Data Model	69
	3.7 A^2V Functionalities	71

Copyright by Mahidol University

CONTENTS (CONT.)

		Page
	3.7.1 AV Client Functions	72
	3.7.2 AVML Processor Functions	75
	3.7.3 Communication between Client Side and Server Side	77
	3.7.4 Materialization Process	77
	3.7.5 The Execution Scheme of a Generic Web Page and Active Rules	78
	3.8 A^2V Usage Guideline	81
	3.8.1 For Web Developers	81
	3.8.2 For Users	84
IV	**IMPLEMENTATION OF A^2V**	85
	4.1 A^2V Data Design	85
	4.1.1 Implementation of Local Active Rule	85
	4.1.2 Implementation of Remote Active Rule	86
	4.1.3 User Activity Document	87
	4.1.4 Merged Items Document	88
	4.1.5 Service Call Parameters Document	89
	4.2 A^2V Functional Design	90
	4.2.1 AV Client	90
	4.2.1.1 Proxy Adaptor	90
	4.2.1.2 User Activity Evaluator	99
	4.2.1.3 Lite Rule Processor	120
	4.2.2 AVML Processor	148
	4.2.2.1 AVML Controller	148
	4.2.2.2 Rule Processor	152
	4.3 Communication Interface Design	169
	4.3.1 SOAP Protocol	170
	4.3.2 HTTP Protocol	170

Copyright by Mahidol University

CONTENTS (CONT.)

		Page
	4.4 Output Design	171
	4.4.1 Explicit Output	171
	4.4.2 Application Log	171
	4.5 Implementation Environment	171
	4.5.1 AV Client Specification	172
	4.5.2 AVML Processor Specification	172
	4.5.3 Network Requirement	173
V	**DEMONSTRATED CASE STUDY**	174
	5.1 Process Description of Ordering Process	174
	5.2 Web Interface of Ordering Process	177
	5.3 Represent Ordering Process in the A^2V Framework	181
VI	**EXPERIMENTS**	189
	6.1 Experimental Objectives	189
	6.2 Experimental Methods	189
	6.3 Critical Analysis of A^2V Performance against Existing Active View Approaches	190
	6.4 Performance Measurement of A^2V	197
	6.4.1 Test Environment	198
	6.4.2 Log Capture Implementation	200
	6.4.3 Experimental Results	201
	6.4.4 Critical Analysis	214
	6.5 Experiment Summary	216
VII	**DISCUSSION AND CONCLUSION**	218
	7.1 Discussion and Conclusion	218
	7.2 Limitations	220
	7.3 Further Research	221
REFERENCES		223

Copyright by Mahidol University

CONTENTS (CONT.)

	Page
APPENDIX	226
BIOGRAPHY	323

Copyright by Mahidol University

LIST OF TABLES

Table		Page
2-1	Comparison of Related Works	25
2-2	Attributes of Web Service Information	32
2-3	Attributes of Service Call Information	32
2-4	Attributes of Service Call Result Handling	34
3-1	Components Mapping of Active XML Peer Components and AVML Processing Units' Components	45
3-2	Comparison of an Active XML Document and an Active Web Page Document	47
3-3	Selected Attributes of Active XML Service Call Element Using in the A^2V Framework	55
3-4	The Body of a Local Active Rule	82
3-5	The Body of a Remote Active Rule	82
3-6	The Body of an Implicit AVML Document Satisfying a Condition	83
3-7	The Body of an Implicit AVML Document Not Satisfying a Condition	83
4-1	List of Relational Operators [34]	157
5-1	Data Member of Cart Viewing Web Page	182
5-2	Data Member of Shipping Address Entering Web Page	184
5-3	Data Members of Payment Method Choosing Web Page	184
5-4	Data Member of Billing Address Choosing Web Page	185
5-5	Local Active Rules Representing the Ordering Process	186
5-6	Remote Active Rules Representing the Ordering Process	187

Copyright by Mahidol University

LIST OF TABLES (CONT.)

Table		Page
5-7	Implicit AVML Documents Representing the Ordering Process	188
6-1	Comparison of Cumulative Response Time on Single Machine between A^2V and Traditional Web Applications Development Approaches	202
6-2	Comparison of Cumulative Response Time on Duo Machines between A^2V and Traditional Web Applications Development Approaches	205
6-3	Comparison of Cumulative Response Time on Multiple Machines (15 machines) between A^2V and Traditional Web Applications Development Approaches	208
6-4	Average Accumulated Response Times and Average Memory Consumed by a Single AV Client in Processing Multiple User Requests	211
B-1	XML Schema Representation for an Implicit AVML Document	233
B-2	XML Schema Representation for a Local Active Rule	234
B-3	XML Schema Representation for a Remote Active Rule	237
B-4	XML Schema Representation for a User Activity Document	241
B-5	XML Schema Representation for a Merged Items Document	243
B-6	Schema Representation for a Service Call Parameters Document	244

Copyright by Mahidol University

LIST OF FIGURES

Figure		Page
2-1	The Syntax of Active Rule	7
2-2	An Example of an Active Rule in [2]	9
2-3	The Architectural Framework of [2]	10
2-4	The Architectural Framework of AVAS [3]	12
2-5	Sub-modules of Active View Agent [3]	13
2-6	The Syntax of the Dynamic Reconfiguration Rule [4]	15
2-7	Examples of Dynamic Reconfiguration Rule [4]	15
2-8	An Example of XML-Based Active Rule [4]	17
2-9	The Architectural Structure of [4]	18
2-10	An Example of Active Rule Encoded in XSLT [10]	20
2-11	The Architectural Framework of the Work of Angela et al.	21
2-12	An Example of Active Rule Represented by XML-Based Format [11]	23
2-13	Application Scenario of [11]	24
2-14	AXML Document Before the Materialization Process of Calling forcast@weather.com [6]	29
2-15	AXML Document After the Materialization Process of Calling forecast@weather.com [6]	30
2-16	AXML Peer-to-Peer Architecture [5, 6, 7, 8, 9]	31
2-17	An Explicit Declaration of Service Call Parameter [14]	33
2-18	An Implicit Declaration of Service Call Parameter [14]	34
2-19	An Example of Web Service Call in AXML Document	35
3-1	Active View Construction in a Centralized Control	39

Copyright by Mahidol University

LIST OF FIGURES (CONT.)

Figure		Page
	Framework	
3-2	Conceptual View of Distributed Active View Construction Manner by the A^2V Framework	41
3-3	The Structure of an Active XML Document	44
3-4	The Structure of an Active Web Page Document	46
3-5	The High Level Structure of an AVML Document	52
3-6	The Graphical Representation of a Web Contents Section	52
3-7	The Graphical Representation of a Local Active Rule	57
3-8	The Graphical Representation of a Remote Active Rule	61
3-9	Sub-modules of the AV Client	65
3-10	Sub-modules of the AVML Processor	68
3-11	Conceptual Data Model of A^2V	70
3-12	USE CASE of the AV Client	72
3-13	USE CASE of the AVML Processor	75
3-14	Conceptual View of Materialization Process	78
3-15	Sequence Diagram of Generic Web Page/Graphical File and Active Rules Execution Procedure	79
4-1	Entity-Relationship Diagram of Remote Active Rule	86
4-2	The Graphical Representation of a User Activity Document	88
4-3	The Graphical Representation of a Merged Items Document	89
4-4	The Graphical Representation of a Service Call Parameters Document	89
4-5	Class Diagram of Proxy Adaptor Module	91
4-6	Activity Diagram of ProxyConnection Class	95
4-7	Sequence Diagram of Proxy Adaptor Module	97

Copyright by Mahidol University

LIST OF FIGURES (CONT.)

Figure		Page
4-8	Class Diagram of User Activity Evaluator Module	101
4-9	Activity Diagram of User Activity Evaluator Module Regarding an Upcoming Active Web Page	107
4-10	Sequence Diagram of User Activity Evaluator Module Regarding an Upcoming Active Web Page	108
4-11	Sequence Diagram of User Activity Evaluator Module Regarding an Upcoming Active Web Page (cont.)	111
4-12	Activity Diagram of User Activity Evaluator Module Regarding an Upcoming User Request	115
4-13	Sequence Diagram of User Activity Evaluator Module Regarding an Upcoming User Request	116
4-14	Class Diagram of Rule Loader Sub-module	122
4-15	Activity Diagram of Rule Loader Sub-module Upon a Coming Active Web Page Received	123
4-16	Sequence Diagram of Rule Loader Sub-module Upon a Coming Active Web Page Received	124
4-17	Activity Diagram of Rule Loader Sub-module Upon a Coming User Request Detected	125
4-18	Sequence Diagram of Rule Loader Sub-module Upon a Coming User Request Detected	126
4-19	Class Diagram of Parameter Binder Sub-module	129
4-20	Activity Diagram of Parameter Binder Sub-module as to a Local Active Rule	130
4-21	Sequence Diagram of Parameter Binder Sub-module in Binding Parameters of a Local Active Rule	133

Copyright by Mahidol University

LIST OF FIGURES (CONT.)

Figure		Page
4-22	Activity Diagram of Parameter Binder Sub-module as to a Remote Active Rule	135
4-23	Sequence Diagram of Parameter Binder Sub-module in Binding Parameters of a Remote Active Rule	136
4-24	Class Diagram of Active View Generator Sub-module	138
4-25	Sequence Diagram of LocalRuleController Class in Constructing an Active View	146
4-26	Sequence Diagram of RemoteRuleController Class in Constructing an Active View	147
4-27	Class Diagram of AVML Controller Module	149
4-28	Sequence Diagram of AVML Controller Module	150
4-29	Class Diagram of Active Rule Loader Sub-module	153
4-30	Sequence Diagram of Active Rule Loader Sub-module	153
4-31	Class Diagram of Parameter Binder and Condition Evaluator Sub-modules	155
4-32	Activity Diagram of Condition Evaluator Sub-module	158
4-33	Sequence Diagram of Condition Evaluation	159
4-34	Sequence Diagram of Condition Evaluation (cont.)	161
4-35	Class Diagram of Active Web Page Generator Sub-module	163
4-36	Sequence Diagram of Active Web Page Generator Sub-module	165
4-37	Sequence Diagram of Active Web Page Generator Sub-module (cont.)	167
5-1	Activity Diagram of Ordering Process	176
5-2	First screen (dummy screen)	177

Copyright by Mahidol University

LIST OF FIGURES (CONT.)

Figure		Page
5-3	Sign-in Screen	177
5-4	Cart Viewing Screen	178
5-5	Choose a Shipping Address Screen	178
5-6	Choose a Shipping Option Screen	179
5-7	Select a Payment Method Screen	179
5-8	Choose a Billing Address Screen	180
5-9	Review Order Screen	180
5-10	Review Order Screen (cont.)	181
6-1	Conceptual Structure of Existing Active View Approaches	192
6-2	Conceptual Structure of the A^2V Framework	192
6-3	Conceptual Structure of Traditional Web Applications Development Approaches	201
6-4	Comparison Chart of Average Response Time on a Single Machine between A^2V and Traditional Web Applications Development Approaches	204
6-5	Comparison Chart of Average Response Time on Duo Machines between A^2V and Traditional Web Applications Development Approaches	207
6-6	Comparison Chart of Average Response Time on Multiple Machines between A^2V and Traditional Web Applications Development Approaches	210
6-7	Cumulative Distribution of Average Response Times by a Single AV Client in Processing Multiple User Requests	212
6-8	Cumulative Distribution of Average Memory Consumed of a Single AV Client in Processing Multiple User Requests	213

Copyright by Mahidol University

CHAPTER I
INTRODUCTION

1.1 Introduction to Web Access Control

Presently, the rapid growth of the Internet and the World Wide Web technologies enables a seamless and convenient channel for those who desire to communicate to other people worldwide. The Internet is applied to make our living more comfortable, convenient, and easier in many aspects. It brings the unprecedented opportunity to create various kinds of *e-service*, which is a general term for commercial services and non-commercial services conducted via the Internet [1].

Generally, an e-service is implemented in the form of web applications that let users interact thru a front-end web-based interface. Many e-services commonly present lots of data and activities to users. However, this inadvertently creates a new burdensome task for the users since they might have to spend lots of time searching for some desired items. Additionally, most users mind doing a business activity or disclosing their personal information due to sensitiveness of the information and the fact that it is risky that the information may be exploded by an unauthorized person. In this regard, many e-service providers employ an access control strategy to keep users' confidentiality and gain the visiting number of users.

One possible Web access control technique is to ascertain that a user could get access to his/her *right data and available activities* based on his/her authorized privilege at the *right time*. This is an alternative security measurement that can be applied to e-services so as to control access of users to e-service data and activities, customizing the content of web view toward each individual user's access right.

To provide such web access control, technically, some web developers implement the business rules that designate who is able to see what by using server-side scripting programs. Such implementation is usually hard-coded inside a web page. However, the business requirements are by nature dynamically changed upon

Copyright by Mahidol University

marketing strategies, so the embedded codes for web access control may have to be frequently updated to reflect new requirements. This not only creates a routine of code maintenance for web developers, but may also introduce changes on existing modules of a web application.

1.2 Motivation of Using Active Views

To resolve some problems of hard-coded access control module as discussed earlier, Active Views were proposed as a dynamic mechanism to accomplish web access control with less effort on code maintenance. An *Active View* [2] is derived from the coordination of a view in a database theory, a web technology, and an active rule. It is physically seen as a view of a web page, also known as a *web view*, in a web browser. An Active View controls the display of data values and also activities that a user can perform according to that user's access right. In other words, an Active View is *an adaptability of content and look and feel* shown in a web browser according to the events of either changes of data or other particular actions performed by users.

The construction of Active Views is featured by the application of Active Rules [2, 3, 4]. In practice, a web developer gathers business requirements that specify what are controlled under which circumstances and implements those requirements in terms of *Active Rules*. To adjust Active Views according to requirement changes in application logic, web developers can simply just update the corresponding Active Rule to reflect the changes. This is easier than changing programming codes at server-side because an Active Rule is represented in a declarative manner. So changing Active Rules does not require much programming knowledge. It is also more intuitive to amend such a descriptive Active Rule than hard coding the rule by using server-side scripting.

Copyright by Mahidol University

1.3 Problem Statement

Although Active Views construction based on Active Rules offers an efficient and dynamic mechanism to provide Web access control, the typical execution of Active Rules to evolve the Active Views is generally in a manner of centralized structural control. In other words, *System components related to Active View construction, such as an Active Rule engine and web servers, were deployed as a centered presence at the server side.* This apparently causes a low performance of Active View construction process. The more complicated view content is, the longer a user may be held for Active View construction since the construction process at server side has to sort the entire workflow out alone. Apart from that, there can be a high bottleneck when the number of user sessions arises, thereby reducing the overall performance of Active View construction process since network bandwidth and server workload for the construction are shared among all end-users.

1.4 Our Approach

To resolve such drawbacks of centralized Active View construction, this thesis proposes a decentralized control model of Active View construction, namely ***A²V***, shortened from an **A**lternative **A**ctive **V**iew construction model. Our A^2V adapts the principle of Active XML [5, 6, 7, 8, 9] to create a distributed framework to enhance the performance of Active View construction for end-users. Along with that adaptation, we develop an *AVML language* (short for Active Views Markup Language) and its processors to reform the process of Active View construction to be executed in a distributed style. AVML is used to specify which portions of Active View construction process are distributively handled by a client side component, namely *AV Client*, (one of two AVML processing units), and which portions are still left to a server side component, namely *AVML Processor*, (another AVML processing unit).

To distribute Active View construction process, the AVML Processor at a server side supplies transactional data, business logic (or application logic), and

Copyright by Mahidol University

presentational style together in the form of XML-based document, namely *Active Web Page*, which is going to be used by the AV Client at a client side to produce an Active View. In this presented way, the client side shares the workload, which has been fully occupied at a server side as seen in the conventional work [2, 3, 4, 10, 11]. This reasonably helps improve the overall performance since the amount of work is proportioned among the client side and the server side and also the number of communication between both sides is decreased.

In our design, an Active Web Page contains a set of Active Rules, which is a primary element of AVML structure. We divide these Active Rules in our A^2V framework into two types which are *Remote Active Rule* and *Local Active Rule* that require different execution schemes to complete. When triggered by a user request, one of these Active Rules is executed to construct an Active View. If the Remote Active Rule is triggered, the AV Client will execute it with the AVML Processor cooperatively, whereas, if the Local Active Rule is triggered, the AV Client will execute it alone, *with no communication between both sides*. As a result, this partly eliminates some communication.

In addition, we provide a generic procedure for constructing Active Web Pages of an e-service application which deploys our A^2V. Details of the procedure are given in subsequent chapters of this thesis.

1.5 Thesis Objectives

- To study how the Active View-related technology fulfils a goal of web access control.
- To study how to enhance the performance of Active View construction by using distributed system architecture.
- To design a framework to help in constructing an Active View efficiently and to decrease workload on a server side by removing some communication processes.

Copyright by Mahidol University

1.6 Scope of Work

- Design an architectural framework which reduces the latency in the process of Active View construction.
- Develop a prototype of our framework by applying Active XML and Active Rules.
- Conduct an experiment hypothesizing if an Active View is locally constructed with no communication to a server, time consumed for Active View construction process should be reduced.
- Evaluate the use of our framework against the traditional web applications development approaches.

1.7 Thesis Outline

The rest of the thesis is organized as follows:

Chapter II Literatures Review: gives a survey of Active View technology-related researches, the related works that applied Active Rule theory to use in e-services development, and related technologies used to implement the A^2V framework.

Chapter III A^2V Framework: introduces our decentralized control of Active View construction approach, our proposed AVML language, including main components and their functionalities. Also, this chapter describes the specification of Active Rules used in our framework and the generic procedure of using our framework for web application developers and end-users.

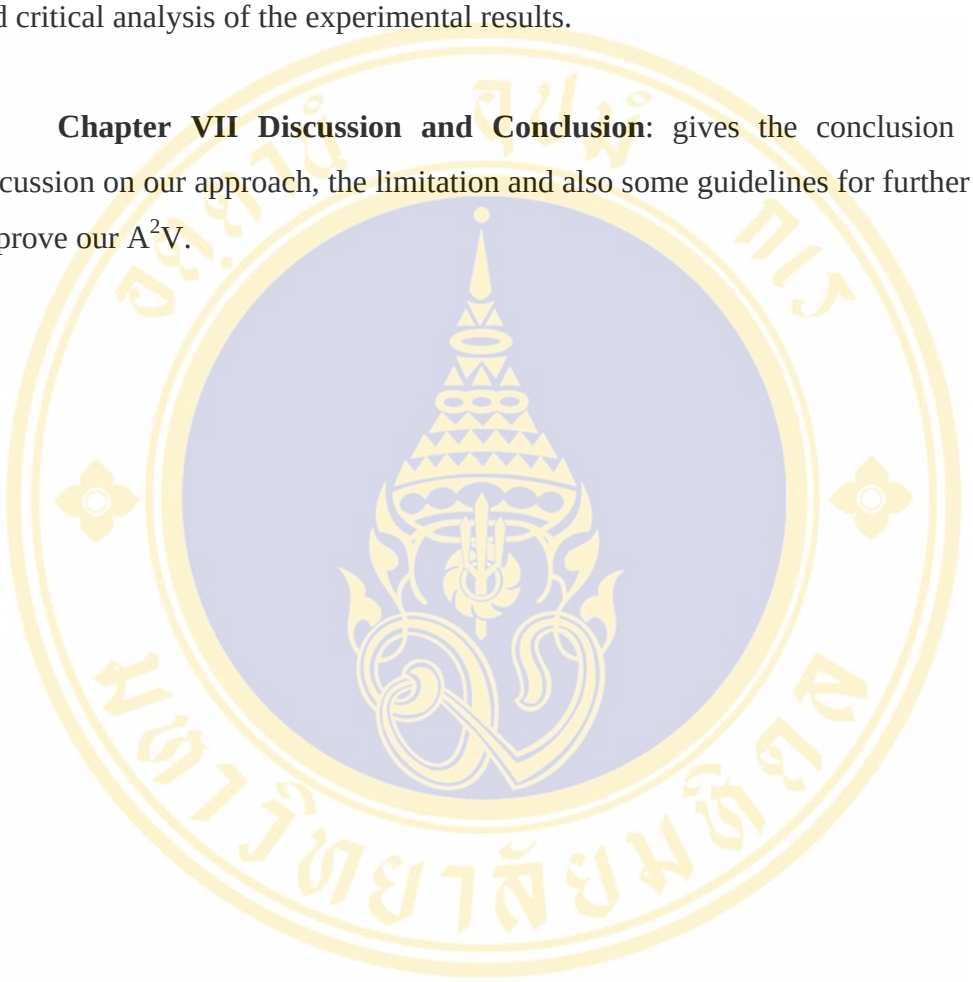
Chapter IV Implementation of A^2V: describes the implementation details of this A^2V prototype, including components both on a client side and a server side.

Copyright by Mahidol University

Chapter V Demonstrated Case Study: illustrates how to implement our customized web application of ordering process by using our A^2V.

Chapter VI Experiments: provides experimental details of our A^2V prototype and critical analysis of the experimental results.

Chapter VII Discussion and Conclusion: gives the conclusion and the discussion on our approach, the limitation and also some guidelines for further work to improve our A^2V.



Copyright by Mahidol University

CHAPTER II
LITERATURES REVIEW

This chapter gives a survey of related works that introduced the Active View technology, and the use of Active Views for e-services. We also give a comparison of the reviewed related works towards our A^2V framework. Finally, we summarize the technologies that are used in our work.

2.1 The Definition of Active Rules

Active Rules [12, 3] are emerged from the basis of Active Database System with added features of Active Rule management. An Active Rule makes a reactive action which is activated by a specified event. To create such a reactive service or action, a conditional clause is inserted into the Active Rule body to ensure if the condition is satisfied prior to issuing a reactive action.

Basically, an Active Rule, also known as an ECA rule, consists of three parts, an event (E), a condition (C), and an action (A). Figure 2-1 illustrates the syntax of an Active Rule. In practice, when a particular event activates an Active Rule and the rule's condition is not omitted, the Active Rule processor will evaluate the condition first. If the condition is satisfied, the rule's action will then be executed.

```
when event
if condition
then action
```

Figure 2-1 The Syntax of Active Rule

Copyright by Mahidol University

2.2 A Review of Related Works in Existing Applications of Active Rules to Implement Active Views for E-Services

This section provides a review of related work that applied Active Rules to implement Active Views for both generating new paradigms for the World Wide Web application development and coping with some security issues for the World Wide Web applications – especially for access control.

2.2.1 What Are Active Views?

"An Active View is generally related to an actual Web window opened by a user of the system. It reacts to user commands and may be refreshed according to notifications sent by the server or the view manager. The methods available on a view depend on the users access rights and may allow him/her to read, load, write, lock, etc. part of the whole the data it sees." (defined in [2]) "An Active View is generally related to an actual Web window opened based on user access rights" (defined in [3]) "ActiveWeb, the view of a Web page including its hyperlinks is changed according to each user's browsing situation including access history or the aggregated information about all users' access histories." (defined in [4]) From these definitions stated in the related work, we conclude that an Active View is a view of a web page (or web view) which is displayed on a web browser and controls the display of data and activities that a user can perform according to a user's access right. Apart from that, it is adaptable upon the events of either changes of data or other particular actions performed by users.

2.2.2 Active Views for Electronic Commerce by Serge et al. [2]

This research worked by Serge et al. is one of the pioneer works that applied the Active View theory and pointed out its pros as an efficient access control mechanism. The work proposed a newly-invented view definition language for

Copyright by Mahidol University

specifying view characteristics for several roles of user (e.g. a visitor and a vendor) and for defining view behavior that were adapted upon user actions. The work coined the new term “Active View”, which was resulted from executing of the view definition language. An Active View delimited what an actor could do with which data. To dynamically generate Active Views upon user actions, this work applied Active Rules as defined in Section 2.1. The implementation details of Active Rules in this work were as follows:

- Events: were actions generated by remote method calls or read/write/append/remove operations on session variables residing in an Active View.
- Conditions: were implemented as XML queries made to an XML repository.
- Actions: could be invocation of remote method calls, operations on actor session variables, or creation of notification messages or trace logs.

```
on goto (owner, activity)
if activity = Customer::search and “good-customer” in owner_ group
do owner_ missive(“Welcome back. We appreciate your business.”)
```

Figure 2-2 An Example of an Active Rule in [2]

Figure 2-2 exemplified the specification of Active Rule in the work of [2]. When an actor caused the action “goto” on the actor’s Active View, an Active Rule would be activated. An Active Rule processor then checked whether the bound argument “activity” was “search” and also the bound argument “owner” fell in the group of “good-customer”. Unless the condition was proved false, there was a message “Welcome back. We appreciate your business” appeared on an Active View.

We describe main components in this framework in Figure 2-3 below:

- OO XML Repository was a XML database server that provided the features of data persistency and a concurrency control.
- Active View Object was an object of user views seen as a middle tier interface between an end user and the XML repository. It maintained a current state of user sessions that affected to what should be displayed on a web browser.

Copyright by Mahidol University

- Active View Application Manager was a single central controller of this framework. It coordinated a client tier to a data tier (XML repository). This component consisted of the sub-components which were an authentication module, a tracing manager, and Active Rule managers.

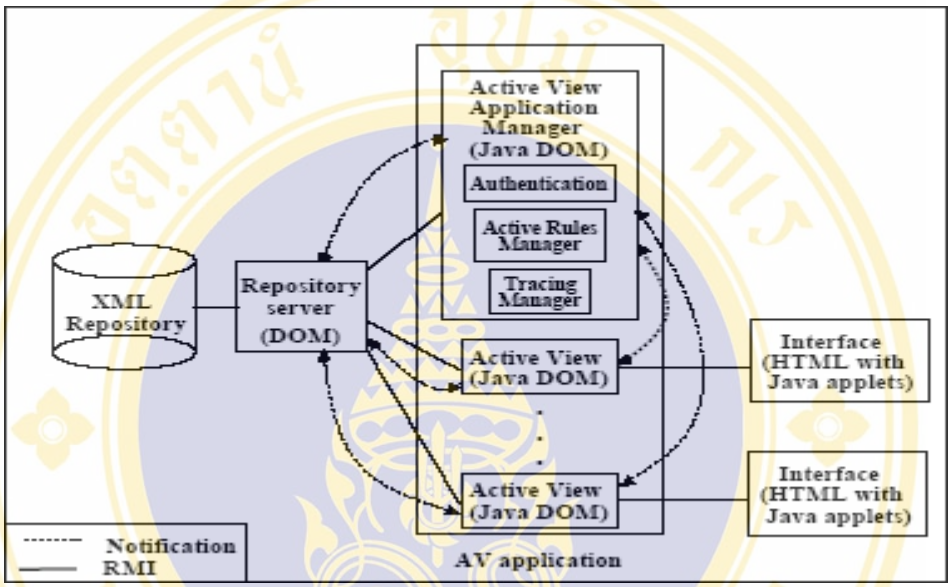


Figure 2-3 The Architectural Framework of [2]

Considering the execution of Active View construction process, this research had the architectural structure in a centralized control manner because, in the center part, the single Active View application manager had to conduct all concerned components in the system and the Active View objects of all users. When the number of Active View objects increased, it evidently caused low performance of the system. Also, the Active Rule manager (the sub-module of the Active View application manager) that was in charge of this centrally settled, so the response time in refreshing a content of Active View was pretty slow.

Copyright by Mahidol University

2.2.3 Active View Agents System for Electronic Commerce (AVAS) by Kesorn [3]

This work could be seen as the succeeding of the previous research. Kesorn et al. intended to overcome the limitation found in the original work.

Their work proposed such an Active View generator based on the approach of multi-agent based system. Its interesting ability was not only to integrate multiple e-commerce application sites as one, but also to administer a diversity of individual e-commerce application sites in many aspects, for instance, an application domain, a database system, an architectural system, and hardware and software specification whereas the previous research could not fill this. The output of this work was an Active View that controlled data in a database system, a web content data (e.g. static data, dynamic data, and multimedia data), and also the user activities with respect to a user access right. An active property enriching a capability of Active View made it dynamically updated. An Active Rule theory enabled Active Views to be active.

In the work of Kesorn et al., they applied the concepts of multi-agent to succeed over the heterogeneous existing e-commerce application sites. They designed the co-working of four agents as explained in details here:

- Administrative Agent (AMA) was for managing AVAS configuration settings.
- User Interface Agent (UIA) was for handling a user request and an output display.
- Active View Agent (AVA) was for handling any requests for data to each individual e-commerce application sites and for an Active View construction process.
- Rule Managing Agent (RMA) was for defining an Active Rule upon business logic of each application site.

Figure 2-4 illustrated the architectural framework of AVAS that consisted of three main components as given in brief here:

Copyright by Mahidol University

- Multiple e-commerce application sites (acted as a data tier) were the existing e-commerce web applications that had different application domains and data sources.
- AVAS server site (acted as a business tier) was a combination of the afore-mentioned agents, the AVAS knowledge, and agent working storage.
- Client site (acted as a presentation tier) represented as users in the system which could be classified into two types: an administrator who took into account all relevant configuration settings on the AVAS server site via the custom administrative program. Also, users who saw the final outputs ceased by the system - Active Views - on users' web browsers.

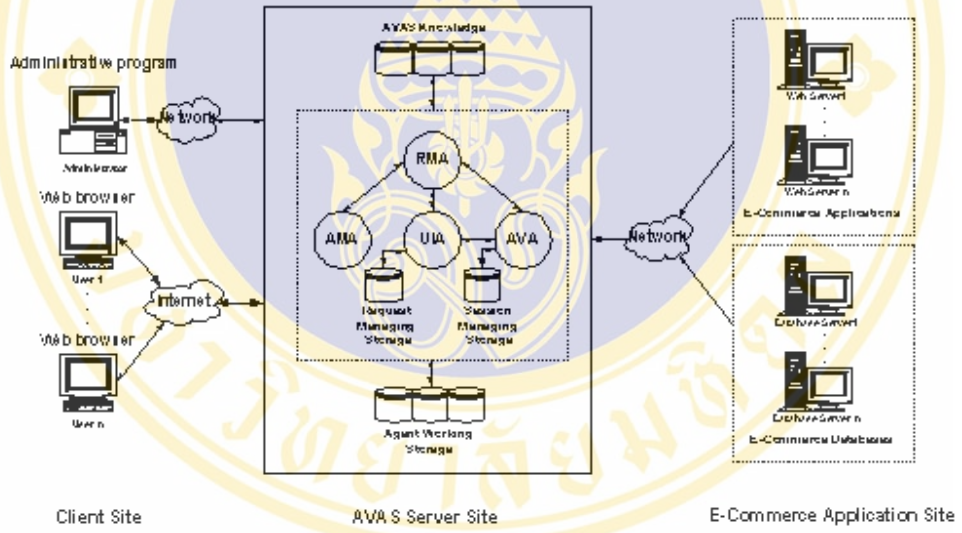


Figure 2-4 The Architectural Framework of AVAS [3]

One above-mentioned item - AVAS knowledge - was significant to AVAS as it assembled essential data which was the profiles of all e-commerce application sites controlled by AVAS, the specification of Active View for different roles of user, and Active Rule Specification. In this work, XML was an interchangeable message format amidst the four agents and the collections of e-commerce application sites.

Copyright by Mahidol University

In addition, an Active Rule, one of main ingredients, was predefined and kept away from a web application page. The Active Rules of this research were categorized into three types as noted here:

- Priority Rule was for classifying the users in the system based on user roles and individual profiles.
- Access Rule was for controlling available contents on an Active View having a data extracted from database servers, web elements, and activities as well.
- Trigger Rule was for actively monitoring on data and activities of user and automatically updating users' Active Views.

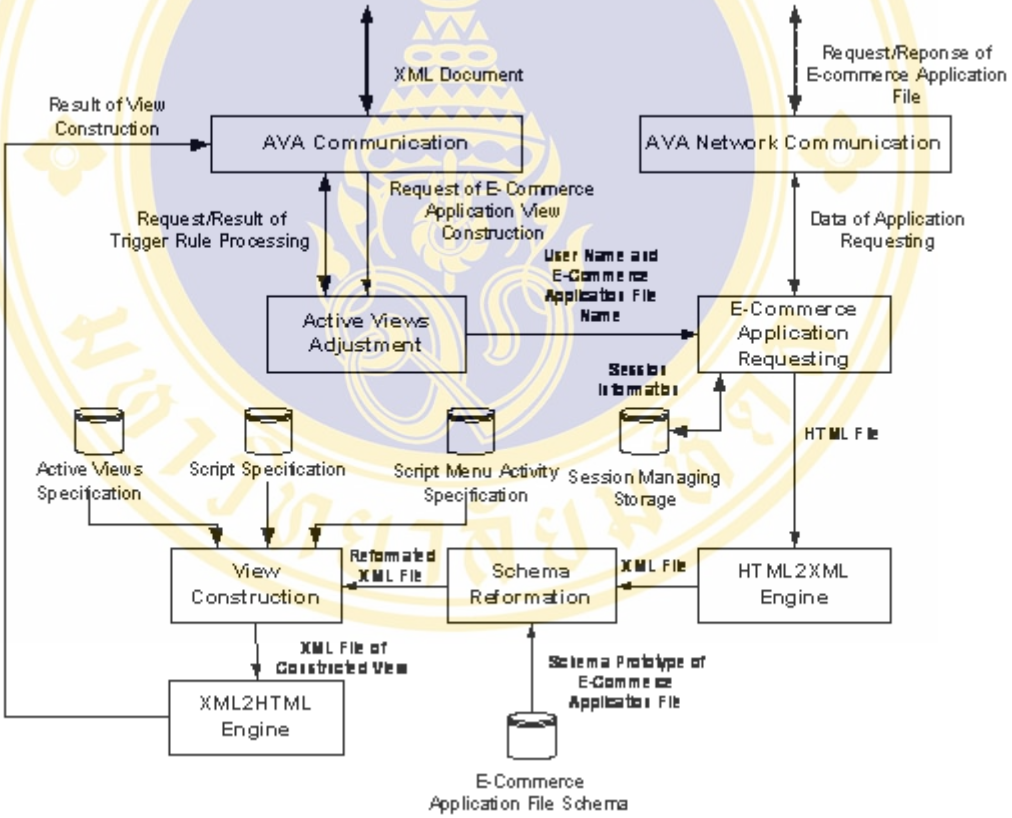


Figure 2-5 Sub-modules of Active View Agent [3]

Figure 2-5 depicted sub-modules of Active View Agent which played a very important role in constructing and adjusting users' Active Views. Firstly, there was a request for updating a view of web page to the Active View adjustment sub-module. Secondly, it passed the requested web page filename on to the e-commerce requesting

Copyright by Mahidol University

sub-module in order to retrieve this web page in HTML and forwarded to the HTML2XML sub-module that converted the HTML web page to XML-formatted one. Thirdly, the HTML2XML sub-module gave the schema reformation sub-module to reformat the XML-formatted web page to an appropriate format for the view construction sub-module. This sub-module aggregated the Active View specification, the script specification, and the script menu activity specification to create a new XML content (either data or activities) that was made upon a user access right. Lastly, the XML content was input to the XML2HTML sub-module so that it would be reformed to HTML-formatted and output to a user's web browser through the User Interface Agent.

In summary, the control of this framework was formed in a centralized control fashion since an Active View construction scheme was managed by the four agents situated on the sole centered server that had to combine data from all concerned e-commerce application sites and process to an end-user.

2.2.4 ActiveWeb: XML-Based Active Rules for Web View Derivation and Access Control by Hidenari et al. [4]

This work by Kiyomitsu et al. presented the idea of using an XML-based Active Rule in deriving a web view and in defining an access control upon access behavior of a user. The outcome of this work was ActiveWeb, which was a view of web page containing web components, for example, page content, URL links, styles. A user access right could dynamically be updated dependent on an aggregation of user access history.

The dynamic reconfiguration rule (for us, it is commonly known as an Active Rule) was proposed to acquire a view of web page upon access behavior of a user. It was a declarative specification that was needed not know of a web programming knowledge. It was predefined by a web page author. A web author designed what would be shown in a view of web page with an analysis of user access histories. Eventually, the dynamic reconfiguration rule was converted to an XML-based Active Rule to be included in other web contents harmoniously.

Copyright by Mahidol University

The dynamic reconfiguration rule was declared in the form of user behavior, historical condition, and server action. This was equivalent to an event- condition-action or ECA rule as seen its syntax in Figure 2-6.

```
User's behavior : Historical condition → Server action
```

Figure 2-6 The Syntax of the Dynamic Reconfiguration Rule [4]

- User Behavior (Event) was counted from a requested web page, a physical/logical location that a user approached, a requesting date, and a referrer web address.
- Historical Condition (Condition) addressed in user access histories (time, the latest accessing period, the referrer pages, and the referrer URL links) or the aggregation of access history.
- Server Action (Action) was what would be performed.

```
default.asp,(referrer, advl.asp): (),()\
➔ add(anchor, "./special1.asp",\
"SpecialOffers");

default.asp, (referrer, adv2.asp): (),()\
➔ add(anchor, "./special2.asp",\
"Special0ffer");
```

Figure 2-7 Examples of Dynamic Reconfiguration Rule [4]

Figure 2-7 illustrated examples of the dynamic reconfiguration rule specification. In the first example, if a user requested the web page "default.asp" and previously browsed the web page "adv1.asp", the system would insert the link-to-the-web-page-"special1.asp" at the label "SpecialOffers". Similarly, in the below example, if a user requested the web page "default.asp" and previously browsed the web page "adv2.asp", the system would insert the link-to-the-web-page-"special2.asp" at the label "SpecialOffer".

Copyright by Mahidol University

```
<rules>
   <rule id=“rule1”>
   <event>
      <read>
         <url>default.asp</url>
         <referrer>adv1.asp</referrer>
      </read>
   </event>
   <condition>
      <user>
         <times operator=mt>10</times>
      </user>
   </condition>
   <action>
      <add>
         <link>
            <url>special1.asp</url>
            <label>SpecialOffer</label>
         </link>
      </add>
   </action>
   </rule>
<rules>
   <rule id=“rule1”>
   <event>
      <read>
         <url>default.asp</url>
         <referrer>adv1.asp</referrer>
      </read>
   </event>
   <condition>
      <user>
         <times operator=mt>10</times>
      </user>
   </condition>
   <action>
      <add>
         <link>
            <url>special1.asp</url>
            <label>SpecialOffer</label>
         </link>
      </add>
   </action>
   </rule>
<rule id=“rule2”>
   <event>
      <read>
```

Copyright by Mahidol University

```
         <url>default.asp</url>
         <referrer>adv2.asp</referrer>
      </read>
   </event>

   <action>
      <add>
         <link>
            <url>special2.asp</url>
            <label>SpecialOffer</label>
         </link>
      </add>
   </action>
   </rule>

</rules>
<rule id="rule2">
   <event>
      <read>
         <url>default.asp</url>
         <referrer>adv2.asp</referrer>
      </read>
   </event>

   <action>
      <add>
         <link>
            <url>special2.asp</url>
            <label>SpecialOffer</label>
         </link>
      </add>
   </action>
   </rule>

</rules>
```

Figure 2-8 An Example of XML-Based Active Rule [4]

As mentioned earlier, the dynamic reconfiguration rule was ceased to an XML-based Active Rule, as depicted in Figure 2-8, and then needed to be combined with a content of web page.

Copyright by Mahidol University

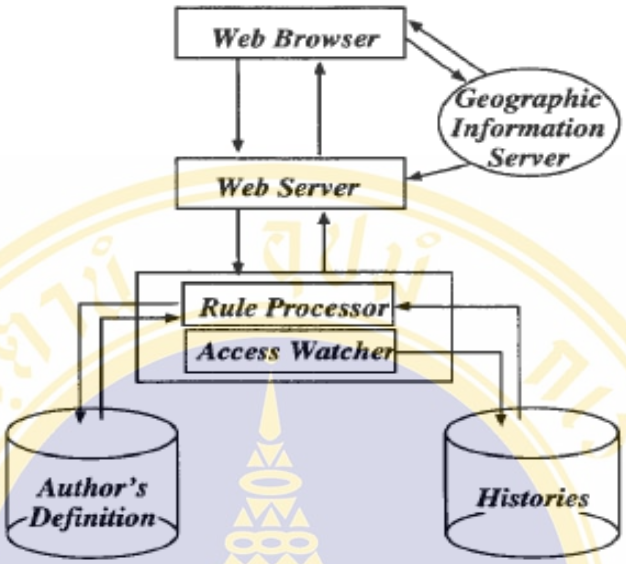


Figure 2-9 The Architectural Structure of [4]

Figure 2-9 illustrated the model of ActiveWeb, which comprised three main components as follows:

- Web Browser was an application tool that a user would use for having access to the system.
- Geographic Information Server archived locations from where a user accessed.
- Web Server served the requested web page in co-working with the rule processor sub-module and the access watcher sub-module so as to extract the access histories data for the execution of the dynamic reconfiguration rule to output an ActiveWeb.

In conclusion, similarly, the architectural structure of ActiveWeb is typed in a centralized control manner because an ActiveWeb is centrally generated by a collaboration of a web server, an access watcher, and a rule processor.

Copyright by Mahidol University

2.2.5 Active Rules for XML: A New Paradigm for e-Services by Angela et al. [10]

This research by Angela et al. presented the evolving of active XML rule that specially handled XML documents in an XML repository when some changes were made. Such an active XML rule would help develop of e-Services rapidly due to the following reasons.

- When directed by an active XML rule, e-Services were automatically invoked once there were changes made to XML documents in an XML repository.
- It was quick to design and develop e-Services based on an active XML rule. Also, an administration of active XML rule was flexibly maintainable.
- The action part of active XML rule could invoke a remote procedure by passing XML-based format parameters, being easy to integrate with other e-Services built on a basis of active XML rule.

Moreover, this work proposed an expression of Active Rule by the two familiar XML query languages which were XSLT (short for eXtensible Stylesheet Language Transformations), a pattern-based language for transforming XML data and Lorel, another query language. An Active Rule was still formed of an event-condition-action (ECA) paradigm as clarified here.

- Event was an update on an XML document caused by a data manipulation in an XML repository and high level operations.
- Condition was a query made over an XML repository expressed by XML query languages.
- Action caused a creation or a modification of an XML document.

Briefly, the representation of Active XML Rule by XSLT consisted of a group of template rule (counted as a function call) which had a pattern matching for a condition checking and a result formulation. XPath was brought to refer to any elements in XML document used in an XSLT processing.

Copyright by Mahidol University

```
CREATE RULE CheckQuotation
PRIORITY: 0
EVENT: <xsl:variable name="Q" select="Quotation">
           <rule:update select="$Q/CurrentValue">
COND_ACT:
   <xsl:variable name="U" select="User">
      <xsl:template match="$U">
         <xsl:if    test="QuotationMonitored=$Q/Name    and    number($Q/CurrentValue)-
number($Q/Opening) > (number($U/RaiseFactor) * number($Q/Opening)]"/>
            <xsl:message>
               <xsl:call-template name="Alerter"/>
            </xsl:message>
         </xsl:if>
      </xsl:template>
      <xsl:template name="Alerter" match="Alerting-Service">
        <sendto>
           <xsl:value-of select="$U/MobilePhone"/>
              <subject>Alert from the Stock Market:
                <xsl:value-of select="$U/QuotationMonitored"/>
                has gone up <xsl:value-of select="$U/RaiseFactor"/>!
              </subject>
        </sendto>
      </xsl:template>
```

Figure 2-10 An Example of Active Rule Encoded in XSLT [10]

Figure 2-10 exemplified the example of Active XML Rule that was expressed by XSLT. It was triggered when there was an update of quotation value. An Active Rule processor had to verify a quotation name and its proper value as specified by the element <xsl:if />. If this condition was satisfied, an Active Rule processor would provoke the template “Alerter” that sent a user’s cell phone a warning message.

The Active Rule execution process was a major element in this framework for executing an Active XML Rule. We try sketching their potential framework based on our understanding as shown in Figure 2-11.

Copyright by Mahidol University

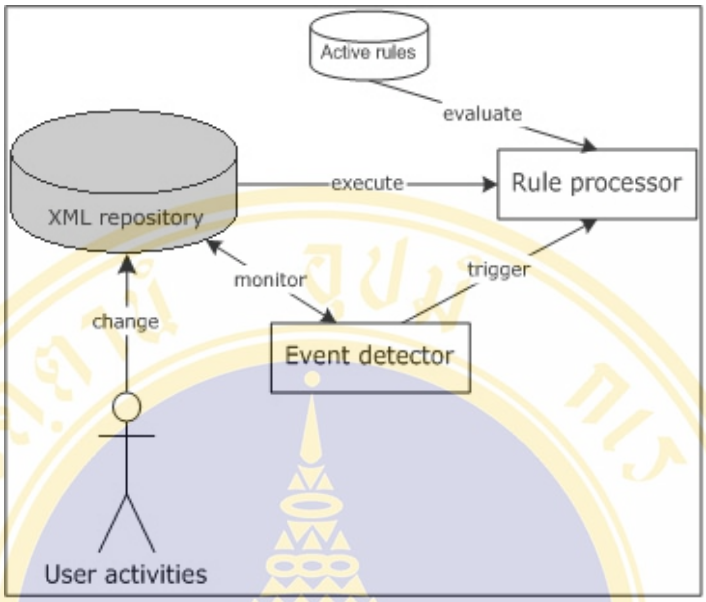


Figure 2-11 The Architectural Framework of the Work of Angela et al.

As shown in Figure 2-11, it comprised the following components:

- XML repository provided a direct XML data source and an open layer to integrate with a legacy data source like file system.
- Event Detector monitored the specified event caused by user activities on an XML document.
- Rule Processor validated the condition part of the Active Rule fired and then, if satisfied, executed its action, causing an update to an XML document.
- Active XML Rule represented the business logic and stored apart from an XML data.

Like the previous work, the architectural structure of this work is typed in a centralized control manner since there is a single Active Rule processor on the center server that serves all changes made to an XML repository.

Copyright by Mahidol University

2.2.6 Pushing Reactive Services to XML Repositories Using Active Rules by Angela et al. [11]

This work was presented by the same working group as in the earlier research, but they hereby tried to expand an effectiveness of an active XML rule to be applied in the idea of pushing technology. More clearly, an active XML rule was brought to use with an XML repository so as to provide a reactive service for a remote subscriber when there was a change of XML document in an XML repository (termed Pushing Technology). For example, when a new product that was interested by any customers arrived (this was practically considered as the coming event for one XML document), the details of this product should be alerted to customers as soon as possible. To accomplish this, an Active Rule involving this released item was created and submitted via SOAP protocol to run nearby a remote XML repository that its detail was already stored.

In the work of Angela et al., they did modify an active XML rule to more loosely express in an XML-based format so that it could fit in the Internet environment and easily integrate with any provided components.

```
<event>insert(//cd)</event>

<condition>
   FOR $a IN //cd
   WHERE $a=$new AND
         $a/price < 20 AND
   contains($a/author,"Milli Vanilli")
</condition>

<action>
   <soap-header>
     <uri>/notification</uri>
     <host>131.175.16.105</host>
     <soap-action>notify</soap-action>
   </soap-header>
   <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/
soap/envelope" SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding">
     <SOAP-ENV:Body>
        <m:Notify xmlns:m="http://131.175.16.105/methods">
           <cdfound>
              $a//*
```

Copyright by Mahidol University

```
            </cdfound>
         </m:Notify>
      </SOAP-ENV:Body>
   </SOAP-ENV:Envelope>
</action>
```

Figure 2-12 An Example of Active Rule Represented by XML-Based Format [11]

When there was an insertion of new CD, this caused the above-mentioned Active Rule, shown in Figure 2-12, triggered. If a CD was newly inserted, its price was lower than 20 units, and also its author is Milli Vanilli, there would send the destined host 131.175.16.105 a SOAP message detailing of this new arrival.

Identically, an Active Rule in this work was still in the form of event-condition-action (ECA), but unlike the previous work, a bit different semantic of ECA rule was noted here.

- Event was triggered when there were changes on XML content in an XML repository, for instance, insert, delete, and update.
- Condition was expressed in an XML query style inquiring data to an XML repository.
- Action was a SOAP method that was invoked once the condition was satisfied.

As shown in Figure 2-13, this paper gave an application scenario relying on the reactive e-service approach that was the coordination of the following components:

- Service Suppliers delivered a service described in XML content in an XML repository that supplied an Active Rule processor. It monitored a new coming service and notified the service reseller.
- Service Reseller received a notification message informing the new products and services.
- Rule Broker, sub-module of service broker, dealt with service suppliers and service reseller in accepting a product searching inquiry via the Internet. It created an Active Rule corresponding to the searched product item and sent to run on an XML repository of the remote targeted service supplier.

Copyright by Mahidol University

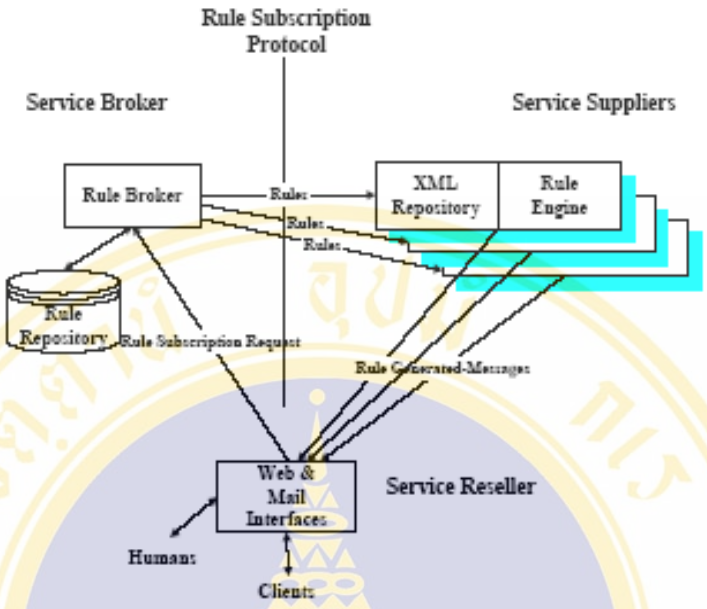


Figure 2-13 Application Scenario of [11]

By overall, we consider an architectural structure of this work as such the centralized control manner. Due to the fact that a single one rule broker accepts all requests sent by subscribed users and also submits a newly suitable created Active Rule to install on the remote repositories of service suppliers, it will cause low performance of a rule broker if the number of subscribers arises. Nonetheless, this work seemingly attempts to reduce workloads of rule broker in decentralizing to complete on the service suppliers. This is for putting Active Rules closer to an intended XML data server. The individual service suppliers potentially execute an Active Rule themselves since having their own rule executor engines. Also, they can alert the resulting messages to the subscribers directly.

2.2.7 Comparison of Related Works

According to our surveys of related works adapting the theory of Active Rule in their works, we compare their critical characteristics as in Table 2-1. Also, we pay a strong attention to some crucial aspects amongst related works against our framework.

Copyright by Mahidol University

Table 2-1 Comparison of Related Works

	Active Views for E-Commerce [2]	**Active View Agents System for E-Commerce (AVAS) [3]**	**ActiveWeb: XML-based Active Rules for Web View Derivations and Access Control [4]**	**Active Rules for XML: A New Paradigm for e-Services [10]**	**Pushing Reactive Services to XML Repositories Using Active Rules [11]**
1. Type of control architecture	Centralized (at server side)	Centralized (at server side)	Centralized (at server side)	Centralized	Centralized (at a server side)
2. Who executes an Active Rule?	Rule Manager at server side	AVA at a server side	Rule Processor at a server side	Active Rule engine at a server side	Rule Engine at Service suppliers' side
3. What does an Active Rule do in framework?	Business Flow Control and Access Control for data	Access Control for data and activities	Access Control for data	Business Flow Control	Business Flow Control
4. What stage of application is suitable to follow the approach?	E-Commerce application built from scratch	The existing E-Commerce application / built from scratch	E-Commerce application built from scratch	E-Service application built from scratch	E-Commerce application built from scratch

Copyright by Mahidol University

Critical Analysis of Related-Works Comparison

This section discusses some crucial points toward the related works and briefs our A^2V approach regarding these points as well.

- **Type of Control Architecture**: [2, 3, 4, 10] were categorized as a centralized control since an Active Rule engine, the main component, centrally located on the centered server. Apparently, this became a bottleneck problem in all processing. The architecture in [11] kept an Active Rule and data repository together on every supplier's site. Certainly, an Active Rule engine was installed where Active Rules were stored. Interestingly, they strived to be ahead to a decentralized control by having more Active Rule engines placed in distributed locations upon the number of suppliers. These engines took care of Active Rules within their sites. Even so, let us consider an Active Rule engine in just one site, there is only one engine serving all the incoming events from customers of this site. This easily causes many queues waiting in line if, in particular, there are a large number of events coming. Besides, in [2, 4, 10, 11], their Active Rule engines were positioned in a business tier in the three-tier architectural model. In [3], their engine was arranged as an additional tier – the fourth tier, this affects another communication cost.

- **Active Rule Execution Processor**: Continuously discussing from the previous point, this concerns who is able to execute an Active Rule. [2, 3, 4, 10, 11] had their own Active Rule engines complete Active Rules as to their custom rule specifications. Their Active Rule engines fixedly located on a certain place – for example, the Active Rule engines of [2, 3, 4, 10] were at a server side while in [11], they were distributively on suppliers' sites. More especially, in [3], an Active Rule engine was based on an agent-based system and was one of the four cooperative agents. It could be said that their Active Rule engines had to complete Active Rules processing task absolutely alone, becoming a bottleneck in the event that a usage increases.

Copyright by Mahidol University

- **Aim of Active Rule**: we classified the aims of using Active Rules into two types which are *a business flow control* and *an access control*. An execution of Active Rules in [2, 10, 11] controlled a business flow whereas, in [2, 3, 4], it controlled only an access control of data and also activities, particularly in [2, 3].

- **Appropriate Stage to Implement the Approach**: [2, 3, 4, 10, 11] were applicable to e-commerce applications built from scratch since they have defined their own protocols. In other words, for any applications which wanted to follow their approaches, they had to configure the concerned components to conform to their specifications. Advantageously, [3] also worked for the existing e-commerce applications because there was the specific agent-based component which was able to handle the heterogeneity of e-commerce applications - this issue is what [3] has aimed to do.

Our approach of A^2V will *first* adjust to a decentralized control structure by attempting to move some processes of Active View construction to be completed by an Active Rule engine at a client side instead of doing at a server side as a whole. An Active Rule engine at a client side could self-carries out an execution task in some cases. This aims at reducing a communication process between both sides. *Second*, we propose the two accommodative Active Rule engines cooperatively working, but they run in different locations and different aims. One is for producing an Active View for a user at a client side and the other one at a server side is for supplying business data and a presentational style for the Active Rule engine at a client side. They all share total workloads. Furthermore, an Active Rule engine at a client side is designed to totally proceed some task with no communication to the other Active Rule engine at a server side, reasonably reducing traffic congestion heading to a server side that has been in [2, 3, 4, 10, 11]. *Third*, we focus on both a business flow control and an access control. We design the two types of Active Rule that sort out different goals: to maintain an access control and a business flow control. *Finally*, our framework is suitable for e-services built from scratch due to the same reason as other works. Nevertheless, it possibly adjusts the existing e-service applications to be deployed in our framework – this requires a migration process to convert from the existing e-

Copyright by Mahidol University

services to ours. It is not just an arbitrary interface bridging between the existing one side and our approach, but we provide an instruction for implementing our framework for the existing e-services applications or the ones built from scratch.

2.3 Related Technologies

In this section, we describe the main ingredients adopted in our framework. Such are Active XML and XML components.

2.3.1 Active XML [5, 6, 7, 8, 9]

Active XML (AXML) was a declarative framework which made use of the approaches of web service and data integration put to work in the peer-to-peer environment. It focused on a distributed data management efficiently because it intended to overcome the diversity of data source and web scalability.

2.3.3.1 An Introduction of Active XML

AXML document, a primary data that flowed through the AXML framework, was an XML document with embedded web service calls. The web service call could somewhere embed in an AXML document. To activate a web service call, it was through the SOAP protocol [13] and the result, returned from a call activation, was updated to the original AXML document. This was termed the *materialization process*.

Therefore, an AXML document comprised two types of data, which were an explicit data (or a solid XML data) and an implicit data (or a data describing how to acquire the exact data). An AXML document was such a valid AXML document that was able to be parsed and use in any web services-aware systems. Thus, an AXML document supported the concept of interoperability.

Copyright by Mahidol University

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<newspaper xmlns="http://lemonde.fr"
           xmlns:rss="http://purl.org/rss"
           xmlns:axml="http://purl.org/net/axml">
 <title>Le Monde</title>
 <date>2-Apr-2003</date>
 <weather>
  <axml:call service="forecast@weather.com">        % SERVICE CALL
   <city>Paris</city>
   <unit>Celsius</unit>
  </axml:call>
 </weather>
 <exhibits>
  <axml:call service="getEvents@TimeOut.com">       % SERVICE CALL
   exhibits
  </axml:call>
 </exhibits>

 <stories>
  <rss:item id="cx_ah_0_218">
   <rss:title>Google goes Blog-Crazy</title>
   <rss:pubDate>Feb 18, 2003 10:36:03 GMT</pubDate>
   <rss:description>Google just acquired Pyra labs, the company
that makes Blogger.
   </rss:description>
  </rss:item>
  ...
 </stories>
</newspaper>
```

Figure 2-14 AXML Document Before the Materialization Process of Calling forcast@weather.com [6]

Figure 2-14 exemplified the AXML document that contained an extensional data, which was, for instance, specified by the elements title, date, and stories, and an intensional data, which was specified by the element “axml:call”. For each service call, the destination web service call was specified by the element by the value of the attribute “service” like “getEvents@timeout.com”. In the case that a targeted service call required some parameters, it needed to put them as child elements of the element “axml:call”, and name an element a parameter name and also set the value of parameter to a content of element. Figure 2-15 showed the AXML document after an activation of service call (as seen in boldface).

Copyright by Mahidol University

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<newspaper xmlns="http://lemonde.fr"
           xmlns:rss="http://purl.org/rss"
           xmlns:axml="http://purl.org/net/axml">
 <title>Le Monde</title>
 <date>2-Apr-2003</date>
 <weather>
  <temp>16</temp>
 </weather>
 <exhibits>
  <axml:call service="getEvents@TimeOut.com"> % SERVICE CALL
   exhibits
  </axml:call>
 </exhibits>
 <stories>
  <rss:item id="cx_ah_0_218">
   <rss:title>Google goes Blog-Crazy</title>
   <rss:pubDate>Feb 18, 2003 10:36:03 GMT</pubDate>
   <rss:description>Google just acquired Pyra labs, the company
that makes Blogger.
   </rss:description>
  </rss:item>
  ...
 </stories>
</newspaper>
```

Figure 2-15 AXML Document After the Materialization Process of Calling forecast@weather.com [6]

2.3.3.2 Active XML Peer-to-Peer Architecture

Theoretically, participants in the Peer-to-Peer framework [16, 17, 18, 19] could possibly be either clients or servers. As a result, in the context of AXML peer, the participant, called AXML peer, held this fact automatically. An interaction between AXML peers was mainly made via an invocation of web service call. For each AXML peer, in Figure 2-16, there were three common components as described here:

- Repository (AXML document store) could be seen as a database server storing AXML documents.

Copyright by Mahidol University

- Client was an AXML peer who invoked a web service call to another AXML peer.
- Server was an AXML peer who provided other AXML peers for the web services. When receiving a request from an AXML peer as a client, an AXML peer as a server issued a query to its own AXML repository and gave the AXML peer as a client back the resulting AXML document.

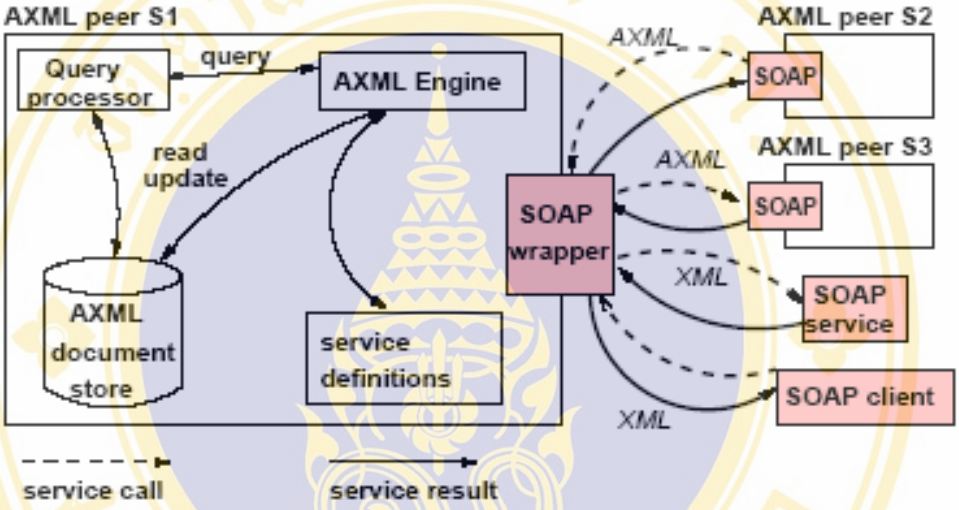


Figure 2-16 AXML Peer-to-Peer Architecture [5, 6, 7, 8, 9]

Figure 2-16 gave the clearer picture of AXML framework. The left side concentrated on a single AXML peer.

The main idea for *AXML peer as a client* was how to manage AXML documents stored in an AXML repository and also a result returned from an invocation of web service call. There was a need to evaluate if it was time to activate a web service call by addressing in four points, which were (1) when a call was activated, (2) where to find an argument as defined in a call signature, (3) how to handle the returned result after a call, and (4) how long the result was still valid. An *AXML peer as a server* took into account how to define or make known of its existing web services.

Copyright by Mahidol University

2.3.3.3 Active XML Specifications

Again, an AXML document was an XML document in which dynamic service calls were embedded somewhere. An AXML document has declared the element “sc” (short for a service call) [14, 15] and the namespace prefix “xmlns:axml=http://www-rocq.inria.fr/verso/AXML” as an area of service call. The ingredients of the “sc” element were made from four parts as described here:

- *Web Service Information* gave details of callable web service specified by the attributes in Table 2-2:

Table 2-2 Attributes of Web Service Information

Attribute	Type	Default value	Status
serviceURL	anyURI		required
serviceNameSpace	anyURI		required
methodName	NCName		required
signature	anyURI		optional
useWSDLDefinition	Boolean	false	optional

- *Service Call Information* specified when and how embedded service calls should be activated. Table 2-3 showed the attributes required:

Table 2-3 Attributes of Service Call Information

Attribute	Type	Default value	Status
id	ID		generated
name	String		optional
frequency	string		optional
lastCalled	unsignedLong		generated
callable	boolean	true	optional

Copyright by Mahidol University

- Service Call Parameter was a non-trivial parameter of a web service call that was put as a child node of the “sc” element in two patterns which were an explicit declaration as shown in Figure 2-17 and an implicit one as shown in Figure 2-18.

```
<axml:params>
  <axml:param name="from">
    <axml:value>
      <currency symbol="EUR">1</currency>
    </axml:value>
  <axml:param>
  <axml:param name="to">
    <axml:value>
      <currency symbol="USD">1</currency>
    </axml:value>
  <axml:param>
</axml:params>
```

Figure 2-17 An Explicit Declaration of Service Call Parameter [14]

Figure 2-17 exemplified a declaration of service call parameter as an explicit type – see the elements <currency> symbol=”EUR”>1<currency> and <currency> symbol=”USD”>1<currency>

```
<?xml version="1.0" encoding="UTF-8"?>
  <currencies xmlns:axml="http://www-rocq.inria.fr/verso/AXML">
  <currency symbol="EUR">1</currency>
  <currency symbol="GBP">1</currency>
  <currency symbol="CHF">1</currency>
  <currency symbol="USD">1</currency>
  <currency symbol="CAD">1</currency>
  <currency symbol="JPY">1</currency>
  <axml:sc serviceURL="http://finance.yahoo.com/services"
serviceNameSpace="urn:YahooFinanceCurrencyConverter"
methodName="convert" frequency="every 1800000">
    <axml:params>
      <axml:param name="from">
        <axml:xpath>//currency</axml:xpath>
      </axml:param>
      <axml:param name="to">
        <axml:xpath>//currency</axml:xpath>
      </axml:param>
    </axml:params>
  </axml:sc>
```

Copyright by Mahidol University

```
</currencies>
```

Figure 2-18 An Implicit Declaration of Service Call Parameter [14]

Figure 2-18 illustrated a declaration of service call parameter as an implicit type – see the elements <axml:xpath>//currency</axml:xpath> and <axml:xpath>//currency</axml:xpath>. In this way, the system would search for all occurrences of the element name starting with "currency" at a runtime phase, but this was quite flexible if the source of AXML document has been changed.

- Service Call Result Handling explained how the returned result from a web service call was handled. The attributes in Table 2-4 helped succeed the stated aim.

Table 2-4 Attributes of Service Call Result Handling

Attribute	Type	Default value	Status
mode	string	merge	optional
doNesting	boolean	false	optional

Upon the syntactical structure of a web service call above, Jerome [14] gave an example of a web service call definition of an AXML document as shown in Figure 2-19. This service call activated a SOAP call to the Google for using a searching service.

Copyright by Mahidol University

```
<axml:sc xmlns:axml="http://www-rocq.inria.fr/verso/AXML"
serviceURL="http://api.google.com/search/beta2"
serviceNameSpace="urn:GoogleSearch"
methodName="doGoogleSearch"
signature="http://api.google.com/GoogleSearch.wsdl"
useWSDLDefinition="true"
id="038BD50A-C353-D490-0082-D603FA7A81AC"
name="GoogleSearch"
frequency="lazy"
lastCalled="1082395879693"
followedBy="//axml:sc[@serviceNameSpace='Crawler']"
...>
[skipped children elements]
</axml:sc>
```

Figure 2-19 An Example of Web Service Call in AXML Document

In our framework, XML [20] is used to structure data sending from a server side to a client side and vice versa. Besides, there are other XML components more employed in our framework as described here:

2.3.2 XML Schema [20, 21, 22]

XML Schema was a meta data that structures a content of XML document. It was able to define a sequence, an occurrence, and data type of XML element through XML Schema. It syntactically ensured validity of data content according to our requirements. In our framework, we use XML Schema specification [23] to define the specifications of our newly-proposed Active Rules and data content flowed throughout our framework.

2.3.3 XSL [21, 24]

XSL, shortened from eXtensible Stylesheet Language, provided an XML document for a presentation style. XSL was such an XML-based language, but was syntactically forced by XSL specification. We had to design the XSL stylesheet in

Copyright by Mahidol University

considering how to address any parts of source XML document and how to define a layout of this data. There are two steps for processing the XSL instruction:

1) *Data Transformation*: to prepare an input data for the second step, done by the transformation language called XSLT [21, 25, 26]

2) *Data Presentation*: to format the transformed XML document for displaying to an end-user, done by the formatting language called XSL Formatting Objects or XSL-FO.

Since XSL is capable of working out a condition processing, in our framework, XSL is adapted to be a condition-action part of an Active Rule. Additionally, XSL plays a crucial part for constructing an Active View displayed on a web browser at a client side.

2.3.4 XPath [21, 24, 25, 27]

XPath provided a powerful searching mechanism to address a portion of XML document under the given expression. The expression gave a location path to the intended part of an XML document. In our framework, XPath flexibly turns out the method to find an XML fragment required to execute Active Rules at a client side and a server side.

2.3.5 XLink [24, 25, 28]

XLink, shortened from an eXtensible Linking Language, extended the linking capabilities from HTML and mainly contributed ability to link two resources. XLink had two types which were a simple link and extend link. In our framework, a simple type of XLink is slightly applied to incorporate a source XML document into an execution of Active Rule at a client side and a server side. To make it standard, we just follow the XLink specification in defining where a source XML document is located on a client side.

Copyright by Mahidol University

In the next chapters, we will discuss about our A^2V framework and an implementation of A^2V prototype in details.

Copyright by Mahidol University

CHAPTER III
A^{2}V FRAMEWORK

In this chapter, we introduce our A^{2}V framework for constructing Active Views in a decentralized control in details. First, we present an overview of the A^{2}V framework in comparison with the former Active View approaches. Next we describe the primary elements of the A^{2}V framework – Active Views Markup Language – and present the architecture of the A^{2}V processing units along with their components, characteristics, and functionalities.

3.1 An Overview of A^{2}V: Distributed Control versus Centralized Control

As reviewed in Chapter II, all conventional models for constructing an Active View have centralized control which leads to many pros and cons. Figure 3-1 depicts a summary picture of those conventional centralized models for constructing an Active View.

As seen in Figure 3-1, all processes for Active View construction in conventional models are executed by an Active Rule engine at a server side. When a user requests a desired web page by submitting a URL thru a web browser or clicking at any objects or hyperlinks in a web page, the Active Rule engine accepts the request and forwards it to a web server in which the destined web page is located. The web server then returns the requested web page as a HTML output to the Active Rule engine. Once receiving the requested web page, the Active Rule engine checks for the Active Rules of that particular web page from an Active Rules repository. Then it uses the rules to transform the page into an Active View. Finally, it returns the Active View as a response back to the user via HTTP. The Active Rules that the Active Rule engine used in the transformation can be varied according to user groups and their activities.

Copyright by Mahidol University

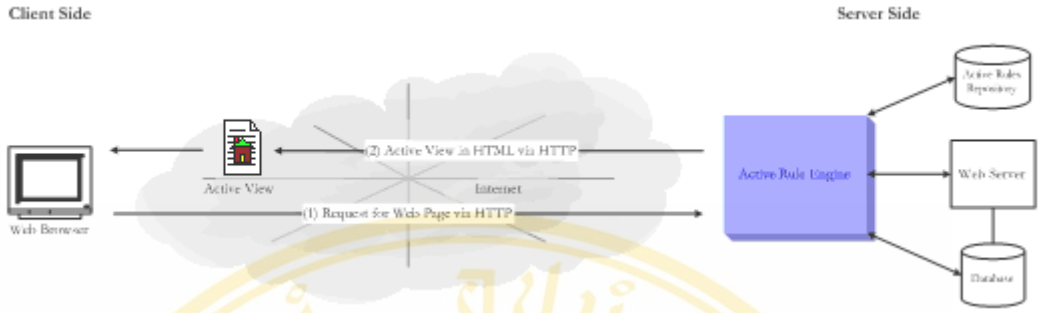


Figure 3-1 Active View Construction in a Centralized Control Framework

From Figure 3-1, once a new event caused by a user action fires an Active Rule in the Active Rules repository, there are the following communication processes occurred:

- *Inbound communication from a web browser to an Active Rule engine*: this is to give the Active Rule engine a request for a web page.
- *Outbound communication from an Active Rule engine to a web browser:* this is to give back the outcome, a new refreshed Active View, to the web browser of the user.

Accordingly, a pair of inbound and outbound communications is created *every time* a user performs an action. This obviously causes low performance issues for Active View construction, especially when a user session grows rapidly because there might be a long waiting queue of Active View construction in line for being executed by a *single* Active Rule engine.

To resolve such cons of centralized control of Active View construction, our A^2V framework adopts the theoretical idea of distributed data processing framework of Active XML in decentralizing Active View construction process to be executed at client sides, so the workload at a server side is partly brought down. In this regard, we create the following components in the A^2V framework.

- *AVML* is our newly proposed XML-based language which describes a domain of web pages, including their data items, under the same decision point

Copyright by Mahidol University

according to the derived activity diagram or the page transition diagram. Its specification details will be further discussed in Section 3.4.

- *AVML document* is a document instance which complies with the specification of AVML.
- *Implicit AVML document* is an AVML-based document whose contents have not been realized in the materialization process. Its details will be further discussed in Section 3.4.
- *Explicit AVML document* (or *Active Web Page*) is an AVML-based document whose contents have been already enriched by the materialization process. Its details will be further discussed in Section 3.3 and Section 3.4.
- *AVML processing units* are two processors, AV Client and AVML Processor, which work together to process all AVML-based documents. The details of the processors will be given in Section 3.5, Section 3.7.1, and Section 3.7.2.
- Materialization process, done by the AVML Processor, is a process which transforms implicit AVML documents to explicit AVML documents by binding references of data items and Active Rules, defined in implicit AVML documents, with their actual contents, mapped from various data sources and the Active Rules repository. Its details will be further discussed in Section 3.7.4.
- Local Active Rule, one of our newly proposed Active Rules, is remotely and completely executed by the AV Client at a client side to construct an Active View. Its definition and structure in details will be further discussed in Section 3.3 and Section 3.4.
- Remote Active Rule, another newly proposed Active Rule, is completely and cooperatively executed by both the AV Client at a client side and the AVML Processor at a server side to generate an Active Web Page for the AV Client. Its definition and structure in details will be further discussed in Section 3.3 and Section 3.4.

Figure 3-2 depicts our modified framework of constructing an Active View which is based on a distributed control. We move an Active View construction process to be remotely carried out by our newly proposed component – named the *AV Client* – which acts as an Active Rule engine at a client side. Upon a request for a web page,

Copyright by Mahidol University

the AV Client is able to succeed in executing an Active Rule to construct an Active View completely. As a result, this helps save time consumed in a communication process to a server side and shortens a queue of Active View construction at a server side.

To enable an execution of Active View construction in a distributed manner, we proposed *AVML* (*short for Active Views Markup Language*) which is designed to separately describe data and a set of Active Rules affecting business logic of an e-service application. AVML is used to describe Active Web Pages, the newly-invented components of our A^2V framework. An *Active Web Page* is an XML-formatted document which contains a set of Active Rules and data items required in constructing Active Views. The details of AVML and Active Web Pages will be further described in the next section.

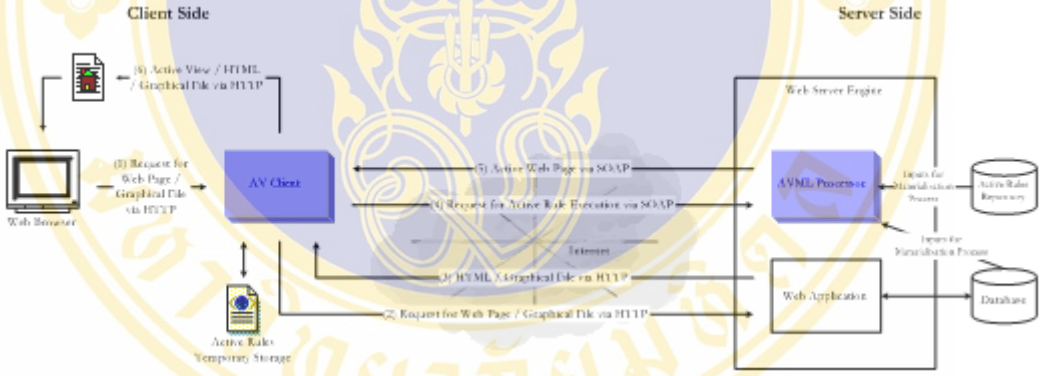


Figure 3-2 Conceptual View of Distributed Active View Construction Manner by the A^2V Framework

As our decentralized control framework depicted in Figure 3-2, there are two main parts in the A^2V Framework as follows:

1) **Client Side**: is where an end-user gets access to e-service applications. This side consists of two components, which are a web browser and the AV Client.

2) **Server Side**: is where a web application system in the A^2V Framework operates. The system consists of a web server, a database server, the *AVML Processor*,

Copyright by Mahidol University

and the *Active Rules Repository* containing business rules of e-service applications. Technically, there is one AVML Processor per e-service applications site. That is, a single AVML Processor will manage all web applications within a single e-service site.

From Figure 3-2, an interaction between the two AVML processing units – the AV Client and the AVML Processor, in constructing Active Views can be further described as follows:

1) Flow labeled (1) in Figure 3-2: when a user requests a desired web page by submitting a URL thru a web browser or clicking at any objects or hyperlinks in a web page, the request is transparently detected by the AV Client installed at the user's machine. By resolving the content of the request's header, the AV Client will then determine whether the request is a typical HTTP request for a typical web page or a graphical file or it is a special kind of requests for Active View transformation. The details of how the request's header are specified and the resolution process of the request's header by the AV Client will be further described in Section 4.2.1.1 of Chapter IV.

2) Flows labeled (2), (3) in Figure 3-2: if the request is for a generic web page or a graphical file, the AV Client will forward the request to a web server via HTTP protocol. The web server then returns the requested web page as HTML output to the AV Client. Once receiving the requested web page, the AV Client will forward the requested web page to the web browser to display the web page for the user. This process is the same as a communication between a client side and a server side in a centralized control framework.

3) Flows labeled (4), (5) in Figure 3-2: alternatively, if the request is for Active View transformation, the AV Client will first determine whether the request triggers any Active Rules that are stored in the Active Rules temporary storage in the user's machine. If the triggered Active Rule is a *Local Active Rule* that can be completely executed by the AV Client, the AV Client will construct an Active View from the triggered Active Rules execution and return a new Active View to the web browser with no connection to a server side. If the triggered Active Rule is a *Remote Active Rule* that the AV Client cannot complete the Active View execution solely, it invokes

Copyright by Mahidol University

a service call, together with passing necessary parameters, to the AVML Processor at a server side for executing the Active Rule via SOAP protocol. The AVML Processor then executes the call to return the resulting Active Web Page consisting of the new data items and a new set of Active Rules to the AV Client via SOAP protocol. Upon the receiving of the new Active Web Page, the AV Client will then execute the Active Web Page to generate a new Active View to return to the web browser for displaying to the user. The details of a Local Active Rule and a Remote Active Rule will be further discussed in the next section.

3.2 Applying Active XML to A^{2}V Framework

We adapted the arguments of Active XML approach to be used in our A^{2}V framework as follows:

1) **Distributed Control of Process Execution**: The Active XML framework is put to work in a distributed control manner, based on the Peer-to-Peer network architecture. One Active XML peer interacts with another peer by a service call invocation via SOAP protocol. In the A^{2}V framework, we adapt such interaction to distribute some processes to be executed by the newly-invented component at a client side, namely the AV Client, based on the Client-Server network architecture. The component at a server side is termed the AVML Processor. Technically, the AV Client is kind of a proxy-like application whereas the AVML Processor is kind of a Web Service-based application. Similar to the Active XML framework, the AV Client interacts with the AVML Processor by invoking a service call to the AVML Processor via SOAP protocol.

2) **Data Definition & Structure**: In the Active XML framework, the primary data sent and received between two Active XML peers is called an Active XML document which is an XML-formatted data which its detail is provided in Section 2.3.1 of Chapter II. Figure 3-3 shows the structure of an Active XML document which is composed of XML data and embedded Web Service calls in a mixing manner.

Copyright by Mahidol University

Likewise, in our A2V framework, the major data sent and received between the AVML Processor and the AV Client is an Active Web Page which is an XML-formatted document instance of AVML specification. In the next section, we illustrate the structure of an Active Web Page document that is adjusted from that of an Active XML document, including the characteristics comparison between these two documents. In addition, we adapt some attributes of Web Service call element in an Active XML document to constitute the specification of Active Rules embedded in an Active Web Page, presented in details in Section 3.4.

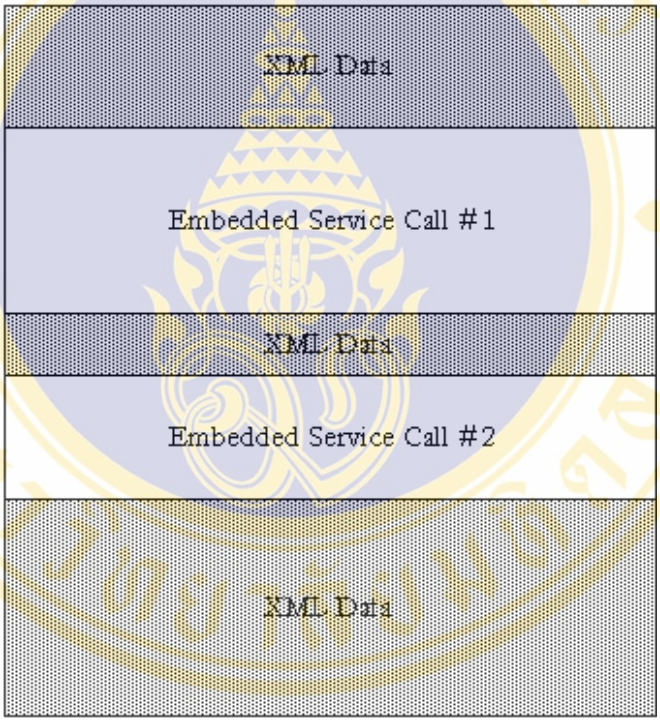


Figure 3-3 The Structure of an Active XML Document

3) **Components Design**: We apply the design and organization of components of an Active XML peer in the Active XML framework to form up the components of the AV Client and the AVML Processor whose details will be further discussed in Section 3.5. Table 3-1 shows an effort to map out the Active XML Peer components and the AVML processing units' components.

Copyright by Mahidol University

Table 3-1 Components Mapping of Active XML Peer Components and AVML Processing Units' Components

Active XML Peer Components	**A^2V Framework**	
	AV Client Components	**AVML Processor Components**
SOAP Wrapper	Proxy Adaptor	AVML Controller
Active XML Engine & Query Processor	User Activity Evaluator & Lite Rule Processor	Rule Processor
Active XML Document Store	Active Rules Temporary Storage	Active Rules Repository
Service Definitions	Active Rule Definitions	Active Rule Definitions

4) **Idea of Materialization Process**: In Active XML framework, the process which a web service call is invoked to keep contents of an Active XML document updated is termed the materialization. We adopt this idea and its name to term the process which an implicit AVML document is transformed to an explicit AVML document in our A^2V framework. The details of materialization process will be provided in Section 3.7.4.

3.3 Active Web Page Structure

As we have mentioned earlier, the results of Active Rules execution in the A^2V framework are called Active Web Pages. An Active Web Page is an XML-formatted document which conforms to the specification of AVML (c.f. Section 3.4). In other words, **an Active Web Page is *an explicit AVML document* resulted from the materialization process of our A^2V**. The materialization process will be further discussed in Section 3.7.4.

A single Active Web Page enwraps a domain of web pages under the same business flow condition. An Active Web Page describes a set of Active Rules and data items required to execute these Active Rules at a client side. Conceptually, we keep an

Copyright by Mahidol University

Active Rule with its data items together in order to enable an Active Rule to be distributively executed on different client sides. This thought has been supported by the idea of data exchange among Active XML peers. To keep it simple, we envision an Active Web Page as *a data and Active Rules carrier* sent to the AV Client at a client side. A web developer has to determine which Active Rules and their needed data items are required in term of *what* items are needed and *where* items are obtained.

Figure 3-4 illustrates the structure of an Active Web Page document which has the Web Contents section at the top area and the embedded Active Rules section at the lower area (the filled one) separately.

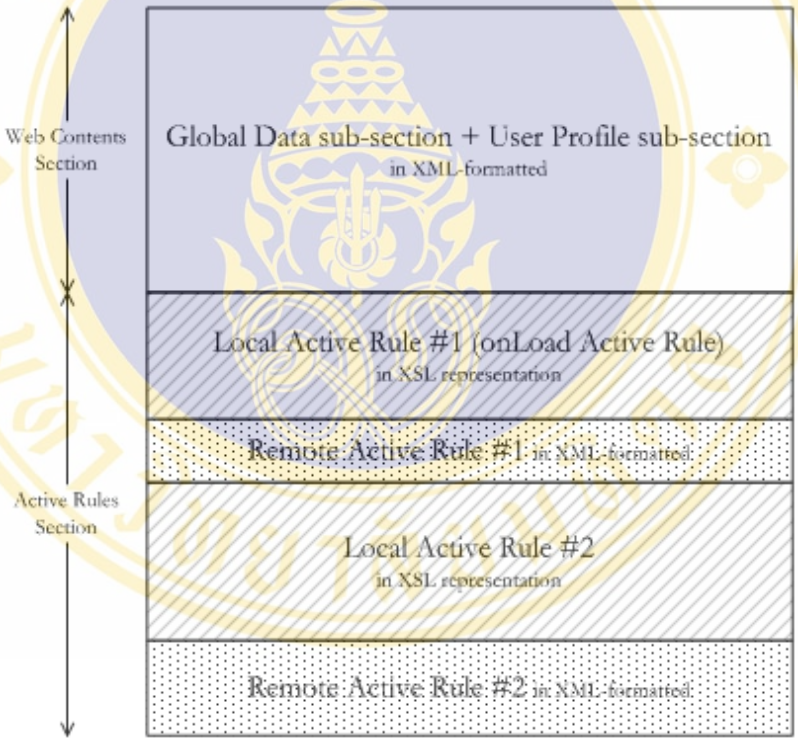


Figure 3-4 The Structure of an Active Web Page Document

Table 3-2 presents the characteristics comparison of an Active XML document and an Active Web Page document.

Copyright by Mahidol University

Table 3-2 Comparison of an Active XML Document and an Active Web Page Document

	Active XML Document	**Active Web Page Document**
1. Type of data	XML-formatted document with embedded Web Service calls	XML-formatted Document with embedded Active Rules
2. What to do with the embedded data?	Used for an invocation of web service call to keep Active XML document updated	Used for an execution of Local Active Rule and Remote Active Rule
3. Outcome after processing the document	Revised Active XML document	Active Views
4. Who processes the document	Active XML Peer in which the Active XML document is being stored cooperates with the destination Active XML Peer	Depends on types of Active Rule contained in the document: - If it is a *Local Active Rule*, the AV Client will process the document alone; - If it is a *Remote Active Rule*, the AV Client will cooperate with the AVML Processor to process the document.
5. When to call	Depends on the value of the attribute “frequency”	Every time a user performs an activity
6. Call parameters	As required by the target service call	As specified in the data binding part of each Active Rule.

Copyright by Mahidol University

The structure of an Active Web Page is composed of two portions as follows:

- *Web Content*s contains a collection of data items used by an Active Rule execution process. These data items are classified into two types of data which are *Global Data* and *User Profile*. A Global Data is business data item commonly and globally used for executing all Active Rules embedded in an Active Web Page – For example, a list of product items, and a payment order. A User Profile holds the preference details of user given at the first time registration of e-service applications.

- *Active Rule*s section contains a collection of Active Rules that represents web pages with respect to a certain business flow. The Active Rules in the Active Web Page are executed to turn out the Active Views which belong to the specified flow *or* to introduce another flow of business logic, which is represented by another Active Web Page. The Active Rule in our framework is divided into two types which are a Local Active Rule and a Remote Active Rule as discussed below.

1) Local Active Rules

This is a type of Active Rules which is locally and completely executed by the AV Client at a client side. The execution of a Local Active Rule causes no communication between a client side and a server side.

Primarily, a Local Active Rule is executed to construct an Active View for a user. In practice, one Active View is represented by one Local Active Rule. It aims at fulfilling the goal of *presentational control* which controls *what and how* data items and activities are displayed on a user's web browser. The presentational control can be considered in the following aspects:

- *Access Control* ("what"): this is to guarantee that the contents and the activities, appearing on the derived Active View, are controlled toward an access right of a user.
- *Personalized Control* ("how"): this is to concern about a look and feel of a resulting Active View personalized as to a user preference.

Copyright by Mahidol University

There is always one Local Active Rule needed in a single Active Web Page. It is a kind of Active Rule that is by default *first* executed when an Active Web Page is delivered to a client side. Such an Active Rule is named onLoad Active Rule.

Section 3.4 will show how a Local Active Rule is constructed in the A^2V framework.

2) Remote Active Rules

This is a type of Active Rules which **cannot** be locally and completely executed by the AV Client at a client side solely, but needs to be cooperatively executed by the AVML Processor at a server side. An execution of a Remote Active Rule certainly causes a communication between a client side (the AV Client) and a server side (the AVML Processor).

In general, there is a decision point in a flow of web pages to check which web page should be browsed. For example, in our demonstrated case study, the system needs to verify if the entered sign-in name and password are correct. If this is the case, the system will show the cart viewing page, otherwise, the system redirects to the sign-in page to accept the new sign-in name and password again. Such these flows of business logic were defined by a web developer. The result, yielded by the evaluation process of this condition point, comes to two possible ways which are *satisfied* or *unsatisfied*. To evaluate the decision point, different user data produces different evaluation results. That is, if the condition is satisfied, it will induce one flow of web pages; otherwise, the condition is not satisfied, it will induce another flow of web pages.

In our work, a Remote Active Rule represents a decision point which introduces different flows of web pages, represented by different Active Web Pages. Primarily, a Remote Active Rule is executed to produce an appropriate Active Web Page which is dependent on the results of conditional evaluation. As explained earlier, a Remote Active Rule aims at fulfilling the goal of *controlling when to start a new flow of web pages and when a client side has to make connection to a server side*.

Technically, there is possibly no Remote Active Rule in an Active Web Page. The more this type of Active Rule is embedded in an Active Web Page, the more a

Copyright by Mahidol University

communication process is made to a server side. To deliver the better performance, the number of Remote Active Rules should be kept least so that it does not waste of communication time. *The number of communication processes linking to a server side when we process user requests to construct Active Views is equal to the number of all Remote Active Rules that lie in that Active Web Page.*

A Remote Active Rule comprises the following two parts:

- *Data Binding Part* holds binding parameters that are going to be passed along with a service call to the AVML Processor. *This part is encoded in an Active Web Page and is interpreted by the AV Client at a client side.*

- *Rule Body Part* holds the condition part and the action part of a Remote Active Rule. A *condition part*, reflecting business decision point, might need some parameters, which are accompanied by a service call from the AV Client, to be inputted into a condition evaluation process. An *action part* holds the possibly resulting implicit AVML documents upon an outcome resulted from an evaluation process. *This part is stored in the Active Rules repository at a server side and is interpreted by the AVML Processor at a server side*.

When we process a Remote Active Rule, the Data Binding part will be processed first so as to acquire all needed parameters. The Rule Body part is processed in subsequent order.

Logically comparing with the three-tier architecture, the Web Contents section can be seen as a data tier while the Active Rules section is seen as either a business tier or a presentational tier. As a consequence, an Active Web Page is an integration of all the three tiers, well making an Active View construction at a client side complete as if it is done by an Active Rule engine and a web server at a server side as in a centralized control manner.

Copyright by Mahidol University

3.4 Active Views Markup Language (AVML)

To derive an Active Web Page according to its characteristics stated in the earlier section, we present the specification of AVML here.

AVML, short for an Active Views Markup Language, is an XML-based language which is a meta-data for creating an AVML document. Note that there are two states of AVML document. **First**, an *implicit AVML document* is an AVML document before being materialized by the materialization process. In other words, it is an AVML document that describes what, where, and how to obtain its actual contents. **Second**, an *explicit AVML document or an Active Web Page* is an AVML document whose contents have been enriched by the materialization process. We will give the details of the materialization process in Section 3.7.4. *An implicit AVML document is manually created by a web developer, whilst an explicit AVML document is automatically output by the materialization process, which is the last step of a Remote Active Rule execution.* As from now, we will discuss the specification of implicit AVML document.

1) Structure of AVML Document

Figure 3-5 and Figure 3-6 show the graphical representation of an AVML document layout and its Web Contents section. An AVML document is wrapped into an “ActiveWebPage” element, which contains one “WebContents” element and one “ActiveRules” element. The full XML Schema representation of an implicit AVML document is illustrated in Section B.1 of Appendix B.

Copyright by Mahidol University

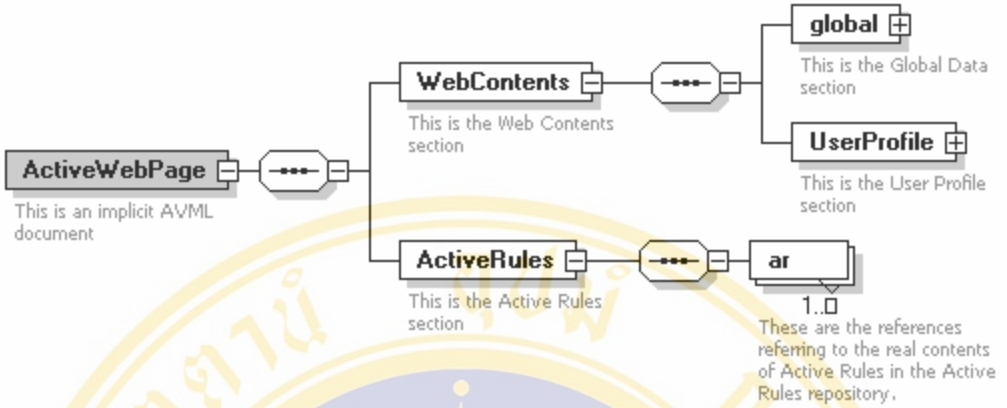


Figure 3-5 The High Level Structure of an AVML Document

Figure 3-6 depicts sub-elements of the element “WebContents”. In a “WebContents” element, all Global Data references are specified by the element “global” while all User Profile references are specified by the element “UserProfile”. Both Global Data references and User Profile references are elements specified in term of a simple type of XLink specification [20, 25] referring to the actual Global Data items and the actual User Profile items respectively. A set of Active Rules is grouped by the element “ActiveRules”, which contains the number of “ar” elements representing each individual Active Rule reference.

Figure 3-6 The Graphical Representation of a Web Contents Section

Copyright by Mahidol University

Example 3-1 A Simple Example of an Implicit AVML Document

```
<?xml version="1.0" encoding="UTF-8"?>
<ActiveWebPage xmlns:xlink="http://www.w3.org/1999/xlink">
  <WebContents>
    <global>
     <product   moreparams="//username"   xlink:type="simple"   xlink:href="DB#select
cast(c.id as char) "id", c.title, author, category, cast(price as char) "price",
cast(discount as char) as "discount", cast(quantity as char) as "quantity" from cart a,
order_line b, product c, user_profile d where b.order_id = a.order_id and b.id = c.id
and a.userid = d.userid and d.username = ?" xlink:actuate="onRequest"/>
    </global>
    <UserProfile>
     <fullname   moreparams="//username"   xlink:type="simple"   xlink:href="DB#select
fullname from user_profile where username = ?" xlink:actuate="onRequest"/>
    </UserProfile>
  </WebContents>
  <ActiveRules>
    <ar id="1" type="local" frequency="onLoad"/>
    <ar id="2" type="remote" frequency="lazy"/>
  </ActiveRules>
</ActiveWebPage>
```

Example 3-1 illustrates a simplified example of an implicit AVML document. In its Global Data sub-section, there is the element “product” expressed in term of a simple XLink. The keyword “DB” means the real content of the element “product” can be obtained from a database server by executing the statement “select” next to the mark “#”. Likewise, in its User Profile sub-section, the real content of the “fullname” element can be obtained from a database server. The domain of sources where Global Data items and User Profile items can be obtained will be discussed in Section 4.2.2.2.3 of Chapter IV. In its Active Rules section, it needs to obtain the real content of Local Active Rule identified by the id. “1” and the type “local” and also the real content of Remote Active Rule identified by the “2” id. and the type “remote” from the Active Rules repository.

Copyright by Mahidol University

Example 3-2 A Simple Example of an Explicit AVML Document or an Active Web Page

```
<?xml version="1.0" encoding="UTF-8"?>
<ActiveWebPage xmlns:xlink="http://www.w3.org/1999/xlink">
  <WebContents>
    <global>
     <product id="cd1" title="The Emancipation of Mimi (Limited Edition)"
author="Mariah Carey" category="Audio CD" price="19.99" discount="0.25" quantity="1"/>
    </global>
    <UserProfile>
     <fullname>Riam Samutrnava</fullname>
    </UserProfile>
  </WebContents>
  <ActiveRules>
    <ar type="local" frequency="onLoad">
      <DataBinding>
        <global/>
        <UserProfile/>
      </DataBinding>
      <CondAction>
       …
      </CondAction>
    </ar>
    <ar type="remote" frequency="lazy">
      <DataBinding>
        <params/>
      </DataBinding>
    </ar>
  </ActiveRules>
</ActiveWebPage>
```

Example 3-2 illustrates a simplified example of an explicit AVML document or an Active Web Page (after materializing the implicit one shown in Example 3-1). There is the only one item in either the Global Data sub-section or the User Profile sub-section. It also shows the simple layouts of a Local Active Rule and a Remote Active Rule in the Active Rules section.

We customized the existing service call components of an Active XML document to define the specification of our Active Rules. We adapted the part of Service Call Information to define the specifications of a Local Active Rule and a Remote Active Rule as well as the part of Service Call Parameter to declare parameter items necessary for an invocation of service call to the AVML Processor. Table 3-3

Copyright by Mahidol University

shows some attributes of Active XML service call element chosen to use in our A^2V framework.

Table 3-3 Selected Attributes of Active XML Service Call Element Using in the A^2V Framework

Active XML Service Call Specification		**Usage in the A^2V Framework**
Active XML Service Call Sub-components	**Attribute**	
Service Call Information	id	Specify an Active Rule identifier (Rule ID.) for a Local Active Rule and a Remote Active Rule.
	name	Specify the readable name of Local Active Rule and Remote Active Rule.
	frequency	Specify when an Active Rule is triggered. There are two domain values of this attribute: - **onLoad** means that an Active Rule is by default fired when an Active Web Page is brought to a client side - **lazy** means that it is caused by the activity event that a user performed.
Service Call Parameter	Value parameter	Write the parameters passed for an execution of Remote Active Rule by the AVML Processor.

Note that the structures of a Local Active Rule and a Remote Active Rule, which will be presented next, fully appear in complete structures. Also, their examples remaining in this section illustrate an actual content. However, all the Active Rules in the Active Rules section of an implicit AVML document are always put in the form of their *references*.

Copyright by Mahidol University

2) Structure of Local Active Rule

Typically, the structure of a Local Active Rule is composed of an event part, a data binding part, and a condition-action part.

In our approach, an *event part*, triggering an Active Rule in the Active Rules temporary storage at a client side, is not explicitly presented in the body of Active Rule, but instead encoded in an XSL code since an event part ties with the event handling mechanism of HTML web components and a client side scripting. The event part is caused by an activity of user performed on a current Active View in a web browser – for example, a hyperlink activation, a button clicking, and HTML form submission, and so forth.

The line #2 in Example 3-3 illustrates an event part of a Local Active Rule. It is the event "onClick" which occurs when a user clicks over an image. The line #3 – #5 are three major hidden fields that identify which Active Rule will be triggered. The example shows that the Local Active Rule, which is identified by the id. "6" and which belongs to the e-service name "www.booklover.com", will be triggered.

Example 3-3 An Example of an Event Part of a Local Active Rule

```
1 <form name="billaddr" method="post" action="viewcart.xml">
2  <input    type="image"    src="../images/address.gif"    height="23"    width="125"
  name="submit" onClick="firerule( billaddr, this )"/>
3  <input type="hidden" name="ruleid" value="6"/>
4  <input type="hidden" name="ruletype" value="local"/>
5  <input type="hidden" name="ecAppName" value="www.booklover.com"/>
6  <input type="hidden" name="activity" value=""/>
7 </form>
```

We present the other two parts of Local Active Rule structure in a graphical representation in Figure 3-7. The full XML Schema representation of a Local Active Rule is illustrated in Section B.2 of Appendix B. As shown in Figure 3-7, a Local Active Rule consists of the following parts:

Copyright by Mahidol University

- *Data Binding part* declares references to *which* data items and *where* at a client side to obtain the actual values of the items given in this part. They can possibly be sourced by three places that are the Global Data, the User Profile, and additional parameters passed along with a *User Activity document*. A reference is expressed in a simple XLink specification specified by the element “DataBinding”. In the process, the AV Client recognizes this part first to bind all variables with their actual values prior to stepping further.

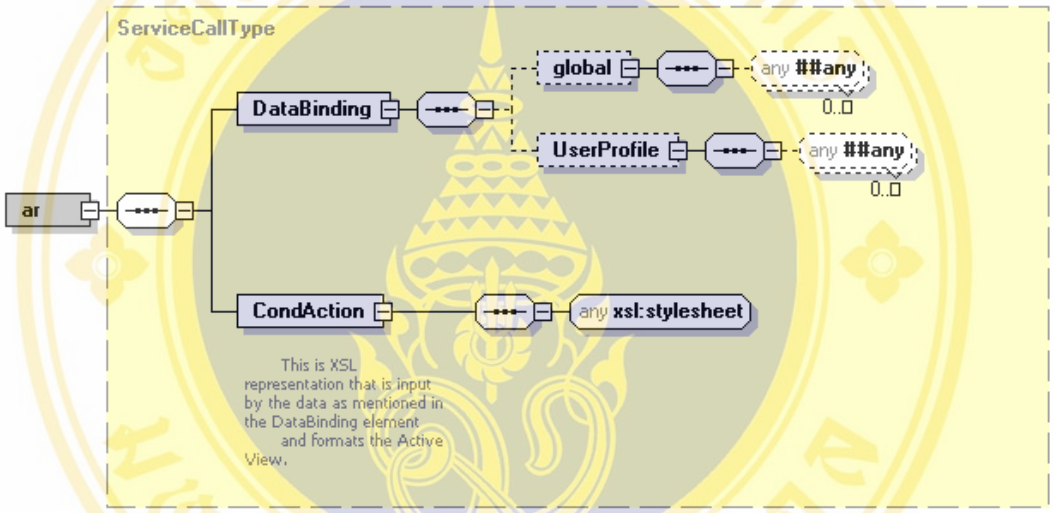


Figure 3-7 The Graphical Representation of a Local Active Rule

Example 3-4 illustrates how a Data Binding part of a Local Active Rule looks like. In the example, the Global Data sub-section contains no item while the User Profile sub-section contains the one item seen at the line #4. To obtain its actual value, the AV Client has to find the file “userprof.xml” and address the path “//billingaddress”.

Copyright by Mahidol University

Example 3-4 An Example of a Data Binding Part of a Local Active Rule

```
1  <DataBinding xmlns:xlink="http://www.w3.org/1999/xlink">
2   <global/>
3   <UserProfile>
4    <billingaddress   xlink:type="simple"   xlink:href="userprof.xml#//billingaddress"
   xlink:actuate="onRequest"/>
5   </UserProfile>
6  </DataBinding>
```

- *Condition-Action Part,* which belongs together in our approach, is specified by the element "CondAction". This part enforces an access control (what data and activities are shown) and a personalization (how data and associated activities are going to be presented to a user). It also represents business constraints of an e-service application. An execution of condition-action part gives a new Active View. In the design phase, we express this part in term of the XSL specification [21, 26]. An output of XSL transformation here is HTML-formatted. Practically, an XSL instruction expressed in the condition and action part is mixed of the following two parts:

  - *Application logic* controls the transition of web pages when an activity was performed. It is represented by a client-side script which is inserted into the XSL body. This cooperates with a HTML event handling. This is a vital part to arrange for parameters to be passed along with a request for a web page to be in XML-formatted. All the parameters are contained in the so-called "User Activity document" in which the specification will be further discussed the next chapter. Some Active Views are possibly designed to have various HTML form components such as a text field, a select field, or a button field. In this case, if a web developer designs to have any activities, performed on these HTML form components, trigger any Active Rules in the Active Rules temporary storage, in accordance with the POST method, he/she will need to put the mandatory parameters, which are Rule ID., Rule Type, and e-service name owning the triggered Active Rule, as hidden fields under a HTML form element (<form>…</form>). Alternatively, in accordance with the GET method, he/she needs to append these three parameters to a URL submitted to. Yet, it depends

Copyright by Mahidol University

on the design view of web developers. As in Example 3-2, the line #3 – #5 show three parameters that indicate which an Active Rule will be triggered when a user clicks an image. Once it happens, this causes an “onClick” event which needs to execute instructions of the function “firerule” shown in Example 3-5. The function “firerule”, which was implemented by a JavaScript language, collects necessary parameters and formats them compliance with the specification of a User Activity document (the line #7). After that, this function submits the form with the parameters, which are now formatted in the form of a User Activity document, in order to call the page “viewcart.xml”.

Example 3-5 An Example of a Condition-Action Part of a Local Active Rule Which Illustrates an Application Logic

```
1  …
2  function firerule( form, image )
3  {
4   if ( image.name == 'submit' )
5    {
6     var atts = { id: ""+form.id.value, selected: "true" };
7     form.activity.value = element( 'activity', element( 'update', element(
   'UserProfile', element( 'billingaddress', '', atts ) ) ) );
8     form.submit();
9    }
10 }
11 …
```

o *View Display* is XSL code which generates HTML code as well as controls the look of an Active View. The line #14 – #23 in Example 3-6 implies the view display control which generates the object “table” in an Active View.

Example 3-6 An Example of a Local Active Rule Which Illustrates a View Display Control

```
1  <ar id="3" name="billing-address" type="local" frequency="lazy">
2   <DataBinding xmlns:xlink="http://www.w3.org/1999/xlink">
3    …
4   </DataBinding>
5   <CondAction>
```

Copyright by Mahidol University

```
6    <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
7     <xsl:output method="html" media-type="text/html" indent="yes"/>
8     <xsl:template match="/">
9      <html>
10      <head>
11       <title>Checkout: Payment</title>
12      </head>
13      <body   bgcolor="#FFFFFF"   text="#000000"   link="#003399"   vlink="#003399"
    alink="#FF9933">
14       <table    border="0"    cellspacing="0"    cellpadding="4"    width="100%"
    align="center">
15        <tr>
16         <td class="headerCell" colspan="2">
17          <b>Address Book</b>
18         </td>
19        </tr>
20        <tr>
21         <xsl:apply-templates select="//billingaddress"/>
22        </tr>
23       </table>
24      </body>
25     </html>
26    </xsl:template>
27   </xsl:stylesheet>
28  </CondAction>
29 </ar>
```

3) Structure of Remote Active Rule

Similar to a Local Active Rule's structure, the structure of a Remote Active Rule is typically composed of an event part, a data binding part, a condition part, and an action part. Likewise, an event part still adapts an event handling mechanism of HTML form components. It is not explicitly defined in the body of Remote Active Rule.

The full XML Schema representation of a Remote Active Rule is illustrated in Section B.3 of Appendix B. As depicted in Figure 3-8, a Remote Active Rule is formed by the following parts:

- *Data Binding part* declares references to *which* data items and *where* at a client side to obtain the actual values of the items given in this part. It applies the same definition as the data binding part of a Local Active Rule. But, unlike a Local Active Rule, as stated earlier, this is the only one part which is placed in the body of Active

Copyright by Mahidol University

Web Page. The AV Client recognizes this part to bind all items with their actual values and packs them in a User Activity document which is passed along with a service call invocation to the AVML Processor.

- *Condition Part*, included in a Rule Body Part specified by the element “if”, represents a decision point in the flows of web pages. Unlike a Local Active Rule, a condition part of a Remote Active Rule is apparently separated from an action part and also contains a collection of conditional clauses which is specified by the element “predicate”. Each of two conditional clauses is operated by the operators “and’ and “or”, which are specified by the element “<and/>” and the element “<or/>” respectively, according to a decision point of business constraint. If a condition part is specified, at least a single conditional clause (the only element “predicate”) will be required.

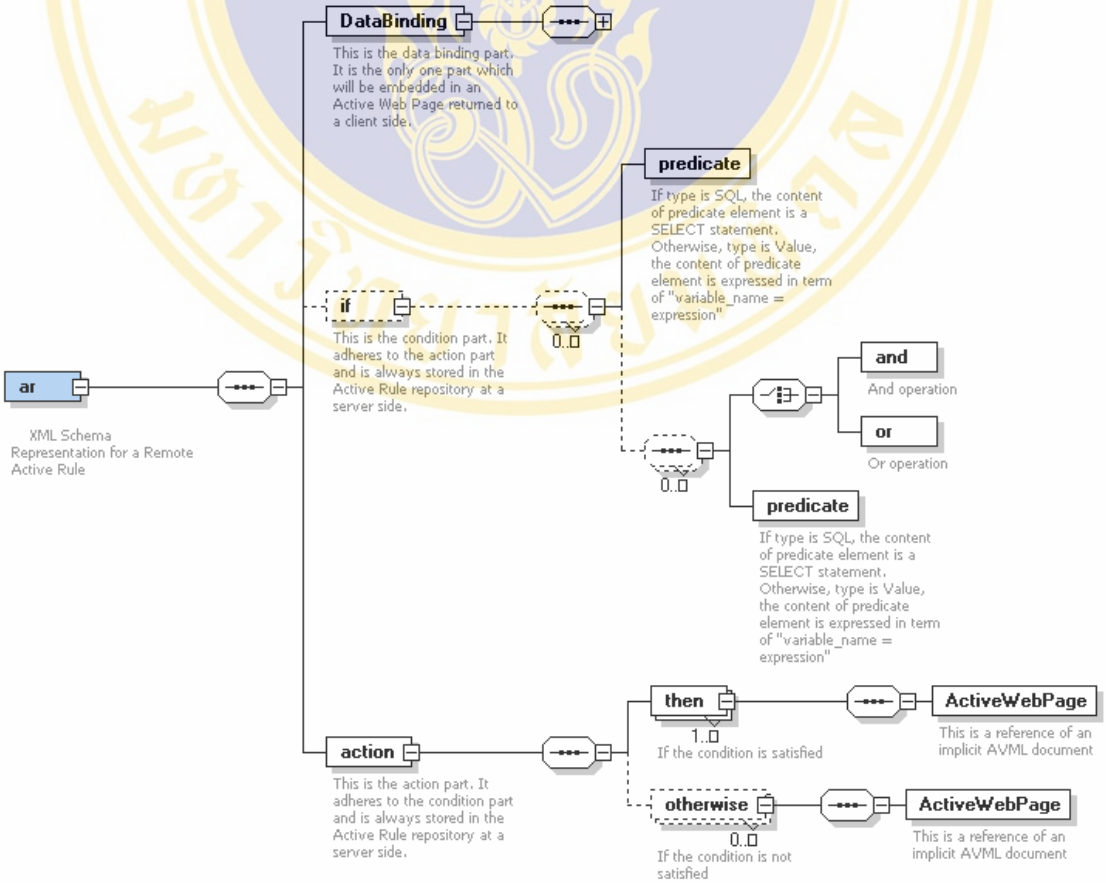


Figure 3-8 The Graphical Representation of a Remote Active Rule

Copyright by Mahidol University

Example 3-7 illustrates a condition part of a Remote Active Rule which consists of a single conditional clause. It is type of “SQL” predicate which the AVML Processor needs to connect a database server and execute the statement “select” so as to get this clause evaluated. This SQL statement requires two arguments corresponding to the occurrences of the number of question marks. The two arguments are obtained from those which are in a User Activity document, received from the AV Client. After running the SQL statement, if there is at least a row returned, the conditional clause will be satisfied; otherwise, no row returned, the conditional clause will be not satisfied. We will further discuss how to specify a conditional clause in Section 4.2.2.2.2 of Chapter IV.

Example 3-7 An Example of the Condition Part of a Remote Active Rule

```
<if>
 <predicate type="SQL">
  select 1 from users where username = ? and password = ?
 </predicate>
</if>
```

- *Action Part* is included in the Rule Body Part, specified by the element “action”. It states which an implicit AVML document should be returned toward the result of a condition evaluation. The different implicit AVML documents will be returned if the condition is satisfying or unsatisfying. If satisfied, the desired implicit AVML document will be specified by the element “then”. It not satisfied, the desired implicit AVML document will be specified by the “otherwise” element. An implicit AVML document that would be given to the materialization process must have already been pre-defined by a web developer. The element “ActiveWebPage” references the desired implicit AVML document.

Copyright by Mahidol University

Example 3-8 An Example of the Action Part of a Remote Active Rule

```
<action>
 <then>
  <ActiveWebPage name="A" ecAppName="www.booklover.com" referrerURI="…"
lastUpdated="…"/>
 </then>
 <otherwise>
  <ActiveWebPage name="B" ecAppName="www.booklover.com" referrerURI="…"
lastUpdated="…"/>
 </otherwise>
</action>
```

As a simple example of an action part of a Remote Active Rule in Example 3-8, if a condition is satisfied, the implicit AVML document "A" will be materialized; otherwise, the implicit AVML document "B" will be materialized.

Since AVML is intentionally designed to be effective in a distributed control structure and to be a primary element in the A^2V framework, we need the collaboration of AVML processing units to proceed an AVML document. They are composed of two components – one is the *AV Client* located on a client side and the other one is the *AVML Processor* located on a server side. We provide the architecture and functionality of AVML processing units in Section 3.5 and Section 3.6 respectively.

3.5 A^2V Architecture

Even though we have tried to displace the task of Active View construction to be completed by the AV Client at a client side, there are some constraints that cause the AV Client not being able to accomplish the whole task in place of the AVML Processor as follows:

- *The diversity of a capacity of client processor and resource*: Certainly, such low-end client processors are not capable of handling the tasks efficiently.

Copyright by Mahidol University

- *The inability to reach some data items*: An execution of an Active Rule deriving an Active View might refer to some data items that are not possible to resolve its value by the AV Client or might be too risk to compromise security policies of the organizations.

As a consequence, the AV Client has to cooperatively work with the AVML Processor in achieving an Active View construction.

This section discusses the components architecture and the responsibilities of AVML processing units involved in processing an AVML document in our framework: the AV Client and the AVML Processor.

3.5.1 AV Client Architecture

The AV Client, a light Active Rule engine component located on a client side, is mainly responsible for handling a request for a web page from a user and then constructing an Active View. We illustrate the architecture of the AV Client in Figure 3-9. In Figure 3-9, we organize all working modules to meet the functionalities of the AV Client stated earlier as follows:

- **Proxy Adaptor** is the first interface that keeps monitoring all the incoming connections from a user's web browser at a client side as well as from a web server at a server side.

  - *For any incoming connections from a web browser*, the Proxy Adaptor obtains a request's header, and then performs a preliminary test into which the following conditions the use request falls.

    - If it is a request for a web page, the Proxy Adaptor will pass it to a web server for taking into account;
    - If it is a request for a graphical file found in a mini-cache on a user's machine, the Proxy Adaptor will read from the cache and return to a user's web browser immediately;

Copyright by Mahidol University

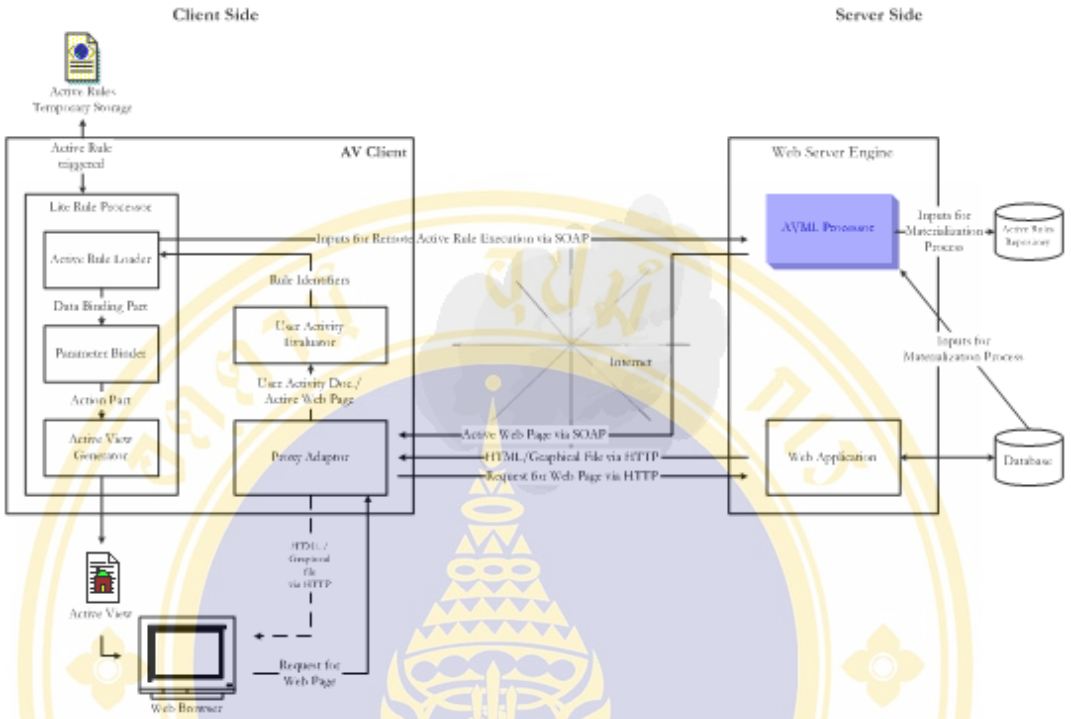


Figure 3-9 Sub-modules of the AV Client

- If the requested graphical file is not found in a cache, the Proxy Adaptor will forward it to a web server for taking into account;

- Otherwise, the Proxy Adaptor will forward the user request, which the *three composite key parameters* which are used to identify which the Active Rule in the Active Rules temporary storage at a client side will be triggered – the Rule ID., the Rule Type and the e-service application name – as well as a *User Activity document*, being included in the user request, to the User Activity Evaluator.

o *For any incoming connections from a web server*, the Proxy Adaptor obtains a response's header and performs a preliminary test if it is a response HTML or a response graphical file. If this is the case, the Proxy Adaptor will post them to a web browser for displaying to a user. *Otherwise, it is a particular Active Web Page (or a dummy Active Web Page) received from a web server for the purpose of experiments conduction especially* – note that this is not a usual way to accept an Active Web Page from a server side, it will forward to the User Activity Evaluator.

Copyright by Mahidol University

- **User Activity Evaluator** is the second module which does the following tasks toward the coming inputs:

- If the coming input is *a request's header containing the three key parameters and a User Activity document* known by scanning through a content of header, the User Activity Evaluator will first extract a User Activity document from the request's header. It finds whether any data items which are related to a User Profile and/or a Global Data were submitted by a user to add, update, or delete their existing ones in the sharing workspace on a user's machine. Second, it extracts the three key parameters from the request's header to search the Active Rules temporary storage for the destined Active Rule, identified by these parameters. We term this "An Active Rule Triggering". Third, it will notify the Lite Rule Processor to retrieve the destined Active Rule from the Active Rules temporary storage to be executed.

- If the coming input is *an Active Web Page*, the User Activity Evaluator will first take a Web Contents sub-section and an Active Rules sub-section off an Active Web Page. The removal Web Contents sub-section is stored in the sharing workspace on a user's machine whereas the detached Active Rules sub-section is stored in the Active Rules temporary storage, grouped by the index of Active Web Page. Second, the User Activity Evaluator needs to prompt the Lite Rule Processor to hold an onLoad Active Rule in memory so that this Active Rule is ready to be executed to construct an Active View for a user.

- **Lite Rule Processor** is the third module which accepts the fired Active Rule from the User Activity Evaluator and then proceeds it properly according to its type – Local Active Rule or Remote Active Rule. The Lite Rule Processor is composed of three sub-modules as follows:

  - **Active Rule Loader** is responsible for reading the fired Active Rule from the Active Rules temporary storage into memory in order to identify its type. Later, it starts to check out what arguments need to be bound and then gives the Parameter Binder the list of arguments to carry out the binding task. Afterward, if the

Copyright by Mahidol University

identified type is *a Local Active Rule*, the already-bound parameters will be sent to the Active View Generator. Otherwise, the identified type is *a Remote Active Rule*, the Active Rule Loader will invoke a service call to the AVML Processor along with the already-bound parameters. Once the AVML Processor returns an output, a resulting Active Web Page, the Active Rule Loader is prompted to execute an onLoad Active Rule of this resulting Active Web Page to construct an Active View.

- **Parameter Binder** is responsible for obtaining the actual value of items listed on the data binding part of the triggered Active Rule, which are sourced from the three possible data sources – a User Profile, a Global Data stored in the sharing workspace on a user's machine, and parameters in a User Activity document.

- **Active View Generator** is responsible for executing an XSL instruction in the *action part* of the triggered Active Rule by inputting the bound parameters at an executing time and displaying a resulting Active View on a user's web browser.

- **Active Rules Temporary Storage** is a particular storage on a user's machine which aims for storing sets of Active Rules that have been detached from their original Active Web Pages. The Active Rules temporary storage can be seen like a business tier at a client side. In comparing with the centralized control model, the storage of Active Rules will be located on a server side only.

3.5.2 AVML Processor Architecture

The AVML Processor, another Active Rule engine component based on a Web Service-oriented architecture, located on a server side, is responsible for serving a proper Active Web Page upon a service call request for executing the triggered Remote Active Rule, received from the AV Client.

Copyright by Mahidol University

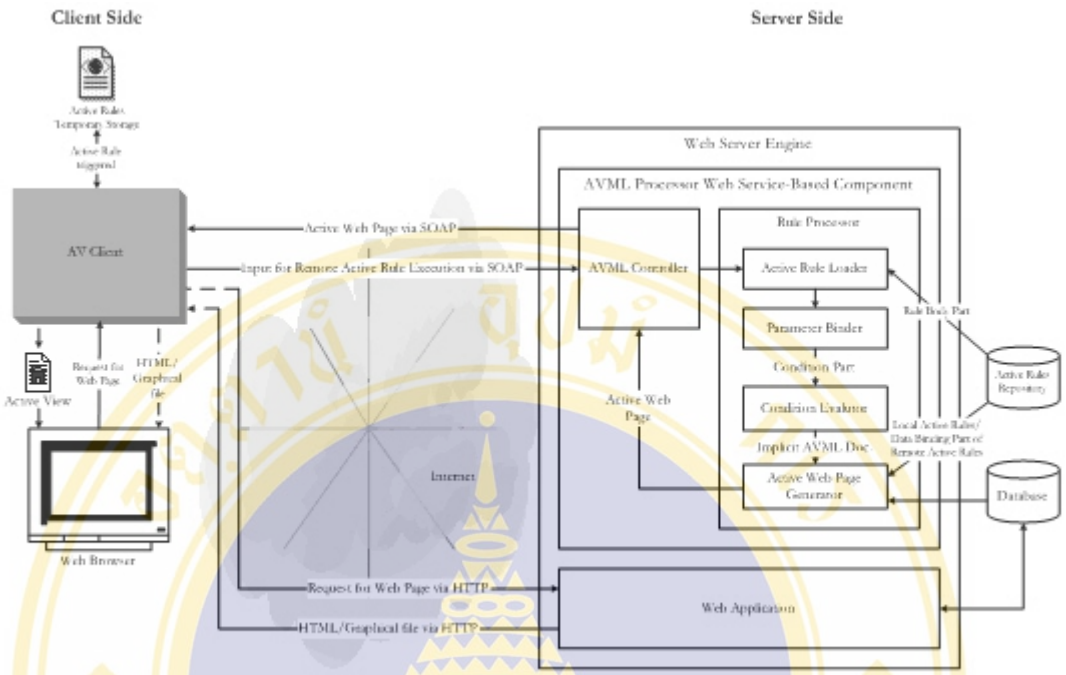


Figure 3-10 Sub-modules of the AVML Processor

Figure 3-10 devises all working modules according to the afore-mentioned functionalities of the AVML Processor as follows:

- **AVML Controller** is a front module that listens to a service call request invoked by the AV Client. Fundamentally, two primary items which need to be submitted along when invoking a service call to the AVML Processor are the Rule ID. of the ongoing Remote Active Rule and the name of referrer web page. Besides of them, there is a set of parameters that is very concerned in the Rule Processor. The AVML Processor is going to pass all three parameters to the Rule Processor. Later, when the Rule Processor has the work done, it returns an output, a resulting new Active Web Page, to the AVML Controller to manage the response back to the AV Client.

- **Rule Processor** takes up a set of data items sent from the prior sub-module, the AVML Controller, into an Active Rule execution procedure. The Rule Processor is a join of the four sub-modules as follows:

Copyright by Mahidol University

- **Active Rule Loader** uses the Rule ID. of the triggered Remote Active Rule and the name of referrer web page to look up the rule body part of the destined Remote Active Rule in the Active Rules repository and puts into the state of ready-to-run.

- **Parameter Binder** is responsible for binding the items, defined in a condition part of the in-memory Remote Active Rule, with their actual values from another receiving parameter left.

- **Condition Evaluator** starts assessing whether a condition part of the triggered Remote Active Rule is satisfied. The result of assessment comes to two possible ways: true or false that brings about different implicit AVML documents returned.

- **Active Web Page Generator** takes an action part of the triggered Remote Active Rule upon the result of condition evaluation to perform the materialization process so as to yield an Active Web Page. The AVML Processor Controller will be prompted to accept the resulting Active Web Page.

- **Active Rules Repository** is a logical storage at a server side in where all Active Rules used in an e-service application are reposited and organized.

3.6 A^2V Data Model

As we introduced the architectural structure of the A^2V in the last section, we present the conceptual class diagram of the A^2V as depicted in Figure 3-11.

Copyright by Mahidol University

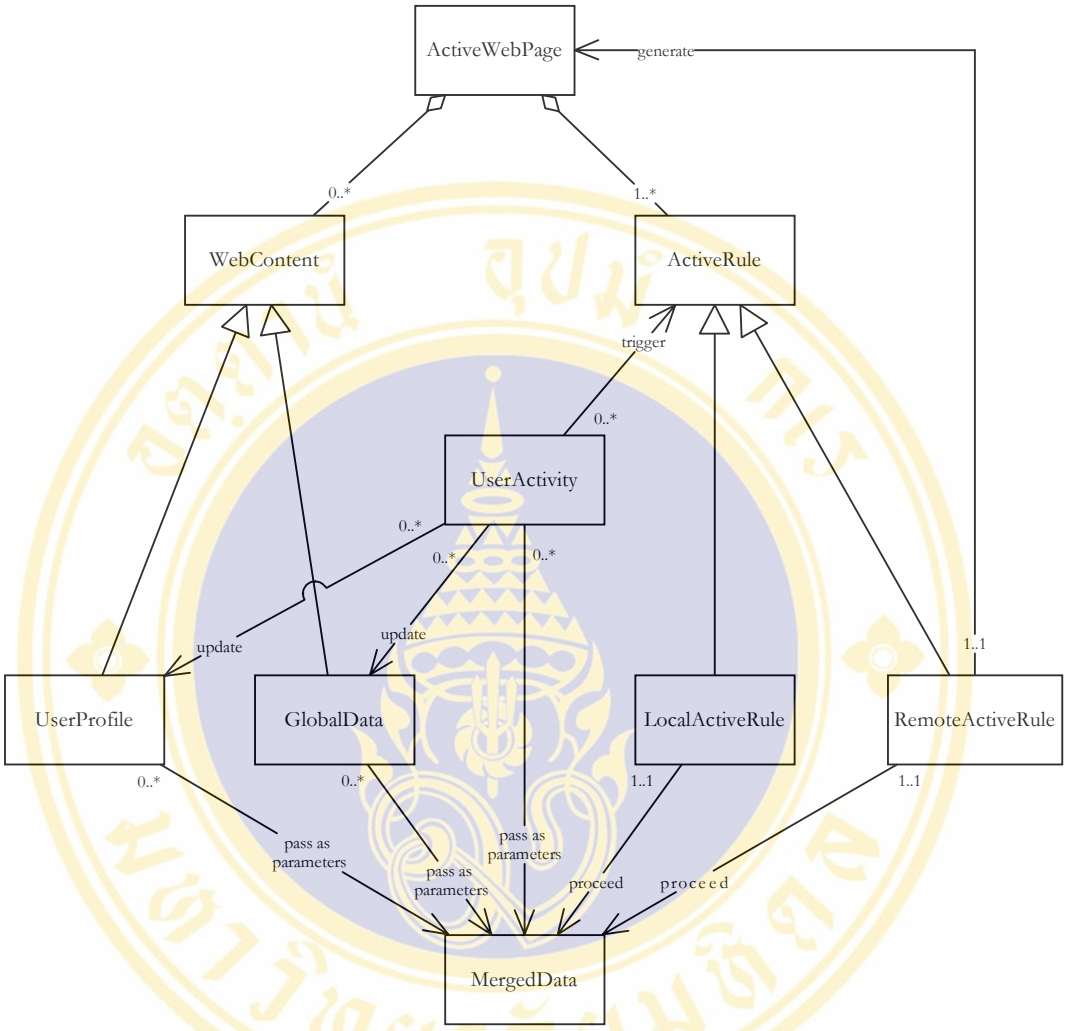


Figure 3-11 Conceptual Data Model of A²V

An *Active Web Page* serves as a small data source for an Active View construction process. It logically represents data, business logic, and presentation style for different users. Matching with what have been represented earlier, the Active Web Page consists of two non-trivial parts, which are *Web Content*s and *Active Rule*s. Firstly, a part of Web Contents provides data as input for business logic. This provides a dissimilar content of Active View for each user and also prevents a user's web browser from communicating to a web server every time to accomplish a user request for Active View construction. The Web Contents is divided into two types that are *User Profile* and *Global Data*. The User Profile collects personal details of a user that

Copyright by Mahidol University

were taken from a registration process, required for each e-service application at the first time use, whereas the Global Data collects transactional data. Both types of Web Contents are necessary to execute an Active Rule at a client side. Secondly, the Active Rules is another part of Active Web Page which provides for an instruction for constructing an Active View. There are possibly several Active Rules embedded in an Active Web Page, but at least an onLoad Active Rule is a must. The Active Rule in our framework is divided into two types which are a *Local Active Rule* and a *Remote Active Rule* whose details have been discussed earlier in Section 3.3. For an arrival of Active Web Page, the AV Client makes extraction of a Web Contents portion and an Active Rules portion from the body of Active Web Page and stores a Web Contents portion in the sharing workspace and an Active Rules portion in the Active Rules temporary storage on a user's machine.

An activity done by a user is *an event* which can fire an Active Rule that has been stored in the Active Rules temporary storage. Along with this event, some parameters, which are formatted in the form of *User Activity* document, might be passed to manipulate the existing User Profile and Global Data on a user's machine.

To execute an Active Rule, the AV Client consolidates items defined in a data binding part of the Active Rule from the three sources, which are the User Profile, the Global Data, and parameters in the User Activity document to form a single data source, namely *MergedData*. For an execution of a Local Active Rule, a MergedData is an input into the XSL instruction processing, whilst, for an execution of a Remote Active Rule, a MergedData is such parameters accompanied by a service call invocation to the AVML Processor.

3.7 A^2V Functionalities

Based on the A^2V architecture stated in the previous section, this section describes the functionalities of AVML processing units through the Use Case diagrams and the sequence diagram. In addition, it presents the sequence diagram which differentiates between the execution scheme of a generic web page/ a graphical

Copyright by Mahidol University

file, a Local Active Rule, and a Remote Active Rule. Also, the communication between the two AVML processing units is discussed in the end.

3.7.1 AV Client Functions

The operations of the AV Client are visualized in the Use Case diagram in Figure 3-12.

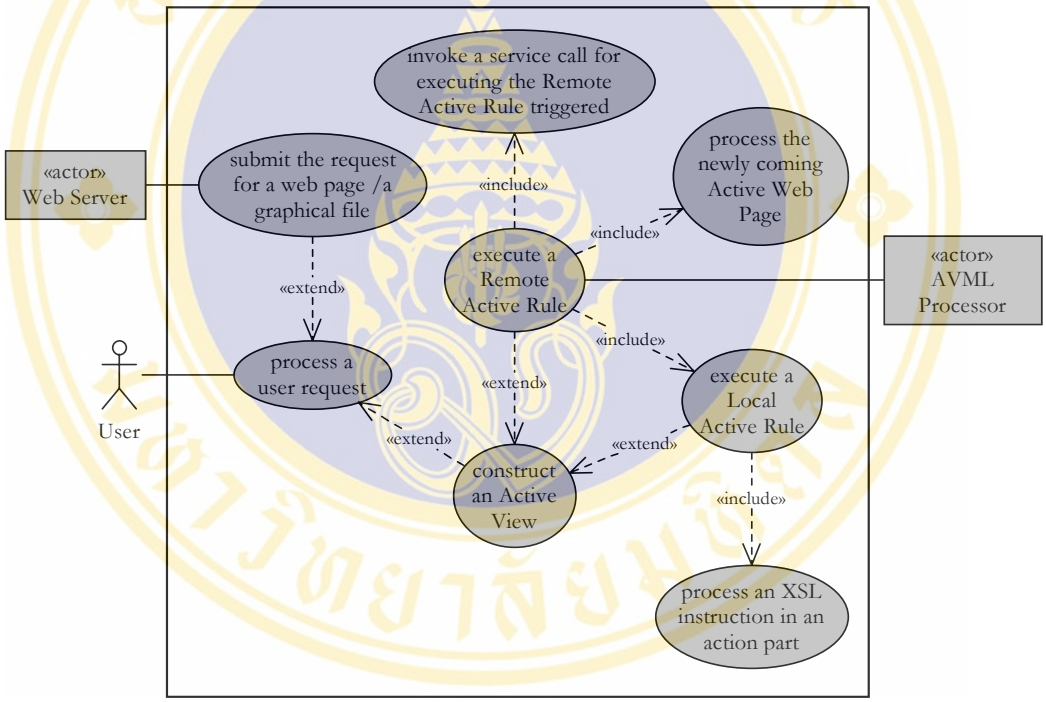


Figure 3-12 USE CASE of the AV Client

There are three actors, involved with the AV Client, which are a user, a web server, and the AVML Processor.

1) ***A User*** is an actor at a client side who connects the system *via a web browser*. A user indirectly interacts with the AV Client thru a web browser, by submitting *a* ***request for a web page or a graphical file*** as an input. This is specified in a content of HTTP header to ask for a desired HTML web page or a multimedia

Copyright by Mahidol University

format file (for example, the filename ending with the following extensions that are jpeg, gif, bmp, swf, and so forth).

2) ***A Web Server*** is an application system located on a server side, which processes user requests and finalizes response web pages in HTML-formatted and graphical multimedia files.

3) ***The AVML Processor*** is an Active Rule engine based on the Web Service development framework situated in a server side, which processes a service call for executing the triggered Remote Active Rule from the AV Client and then responds an appropriate Active Web Page to the AV Client.

The descriptions of the AV Client Use Cases and their relationships are noted as follows:

- **Submit a User Request Use Case** is responsible for handling a coming user request and analyzing what a user request is for – a request for a generic web page/a graphical file or a request for executing the Active Rule triggered. It works out by extending the two Use Cases, which are Process the Request for a Web Page/a Graphical File and Construct an Active View.

- **Process the Request for a Web Page or a Graphical File Use Case** is responsible for forwarding a user request to a web server for being in charge and in turn, receiving a response web page in HTML-formatted or a response graphical file to give back to a web browser for displaying. It is handled by the Proxy Adaptor described in Section 3.6.1.

- **Construct an Active View Use Case** is responsible for constructing an Active View according to the execution of the triggered Active Rule and outputting this Active View to display on a web browser. It also has the data items, accompanied by a User Activity document, updated to the Global Data and the User Profile on a user's machine. It is handled by the User Activity Evaluator described in Section 3.6.1

Copyright by Mahidol University

as well as extends the other two Use Cases, which are Execute a Local Active Rule and Execute a Remote Active Rule.

- **Execute a Local Active Rule Use Case** is responsible for executing the triggered Local Active Rule to construct an Active View. Also, it does bind the data items required for this triggered Active Rule with those which are in the Global Data and the User Profile on a user's machine. It is handled by the Active Rule Loader, the Parameter Binder described in Section 3.6.1, and includes the Process an XSL Instruction in the Action Part Use Case as well.

- **Process an XSL Instruction in the Action Part Use Case** is responsible for executing an XSL instruction in the action part of the triggered Local Active Rule by having the already-bound data items as inputs into an execution scheme. It yields a new Active View. It is handled by the Active View Generator described in Section 3.6.1.

- **Execute a Remote Active Rule Use Case** is responsible for executing the triggered Remote Active Rule to derive a new Active Web Page. It included the other three Use Cases which are Invoke a Service Call for Executing the Remote Active Rule triggered, Process the Newly Coming Active Web Page, and Execute a Local Active Rule. For more information, the Execute a Local Active Rule Use Case is included in an execution of a Remote Active Rule because an arrival of new Active Web Page automatically causes an onLoad event to trigger an onLoad Active Rule – such a Local Active Rule – embedded in an Active Web Page.

- **Invoke a Service Call for Executing the Remote Active Rule Triggered Use Case** is responsible for invoking a service call for executing the triggered Remote Active Rule to the AVML Processor for being in charge and in turn, receiving a response Active Web Page. It is handled by the Active Rule Loader described in Section 3.6.1.

Copyright by Mahidol University

- **Process the Newly Coming Active Web Page Use Case** is responsible for processing an incoming response Active Web Page, then executing an onLoad Active Rule to construct an Active View, and outputting a new Active View to display on a web browser. It is handled by the Lite Rule Processor described in Section 3.6.1.

3.7.2 AVML Processor Functions

The operations of the AVML Processor are depicted in the Use Case diagram in Figure 3-13.

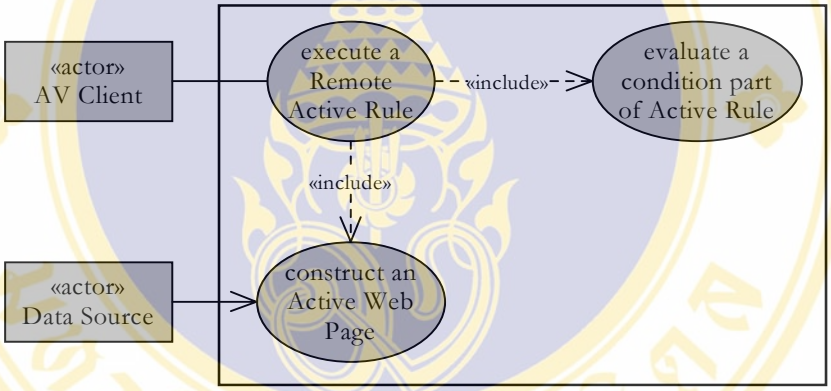


Figure 3-13 USE CASE of the AVML Processor

There are two actors, involved with the AVML Processor, which are the AV Client and the Data Source at a server side.

1) ***The AV Client***, one of two AVML processing units located on a client side, invokes a service call to the AVML Processor for executing the triggered Remote Active Rule and acquires a new Active Web Page resulted from the processing of the AVML Processor.

2) ***Data source***, a data provider situated in a server side – for instance, a database server and a data file, supplies data items that are required to be embedded in the forthcoming Active Web Page.

Copyright by Mahidol University

The descriptions of the AVML Processor Use Cases and their relationships are described as follows:

- **Execute a Remote Active Rule Use Case** is responsible for processing a service call request for executing the triggered Remote Active Rule from the AV Client and in turn returning a proper Active Web Page to the AV Client. It searches the Active Rules repository at a server side for the triggered Remote Active Rule identified by the two parameters accompanied by the service call – the Rule ID. and the referrer page name. It is handled by the AVML Controller and the Active Rule Loader and also includes the two Use Cases, which are Evaluate a Condition Part of Active Rule and Construct an Active Web Page.

- **Evaluate a Condition Part of Active Rule Use Case** is responsible for evaluating a condition part of the triggered Remote Active Rule by inputting the so-called *Service Call Parameters document*, which contains all parameters passed from the AV Client, into an evaluation scheme, if needed. In the process, it does bind the data items required in executing this triggered Remote Active Rule with that are in the Service Call Parameters document. In the end, it outputs the result of condition evaluation and comes out a proper implicit AVML document. It is handled by the Parameter Binder and the Condition Evaluator described in Section 3.6.2.

- **Construct an Active Web Page Use Case** is responsible for performing the materialization process to construct a complete Active Web Page. This is done by mapping the desired data items from various data sources into the Web Contents section and mapping the desired Active Rules from the Active Rules repository into the Active Rules section. It is handled by the Active Web Page Generator described in Section 3.6.2.

Copyright by Mahidol University

3.7.3 Communication between Client Side and Server Side

The Internet is an electronic public network that makes the client side components and the server side components communicable. In our framework, the AV Client handles all incoming and outgoing connections from/to a web browser. There are two pairs of communications between the AV Client and other components at a server side as follows:

- *The AV Client and a Web Server*: they connect each other via HTTP protocol.
- *The AV Client and the AVML Processor*: they link each other via SOAP protocol.

3.7.4 Materialization Process

The materialization process is a process which causes an implicit AVML document to become an explicit AVML document by mapping actual contents of Web Contents from a database server and of Active Rules from the Active Rules repository – as specified in the Web Contents section and the Active Rules section of the implicit AVML document respectively – into the implicit AVML document. Figure 3-14 illustrates how the materialization process works. When the AVML Processor finishes evaluating a condition part of the triggered Remote Active Rule, there is one implicit AVML document resulted no matter what an evaluation result is. The resulting implicit AVML document will then be an input into the materialization process. Once the materialization process has been done, an explicit AVML document or an Active Web Page is output and responded to the AV Client.

Copyright by Mahidol University

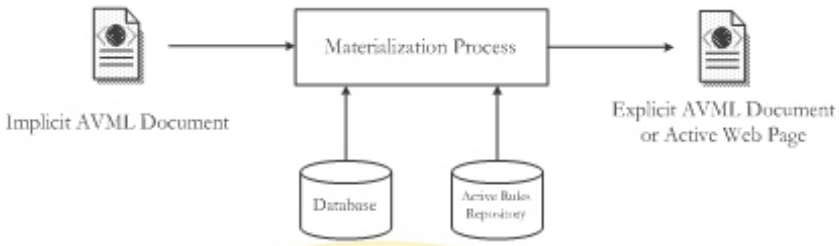


Figure 3-14 Conceptual View of Materialization Process

3.7.5 The Execution Scheme of a Generic Web Page and Active Rules

Let us see how different an execution of a generic web page/a graphical file, a Local Active Rule, a Remote Active Rule, which are mainly managed by the two AVML processing units, are. We illustrate how an interaction procedure between all relevant components goes in the sequence diagram as in Figure 3-15.

The step #1 to #5 are about to request a generic web page/a graphical file to a web server. Here are the details of each step:

1) A user browses a generic web page/a graphical file via a web browser.

2) The AV Client detects the incoming request and finds that this is such a request for a generic web page/a graphical file.

3) The AV Client just forwards this user request to a web server for taking into account.

4) The web server is prompted to proceed the user request and sends back a response web page in HTML-formatted or a response graphical file.

5) The AV Client monitors an arriving responded HTML/graphical file and detects that it is a HTML/a graphical file, so keeps posting to the web browser for displaying to the user.

Copyright by Mahidol University

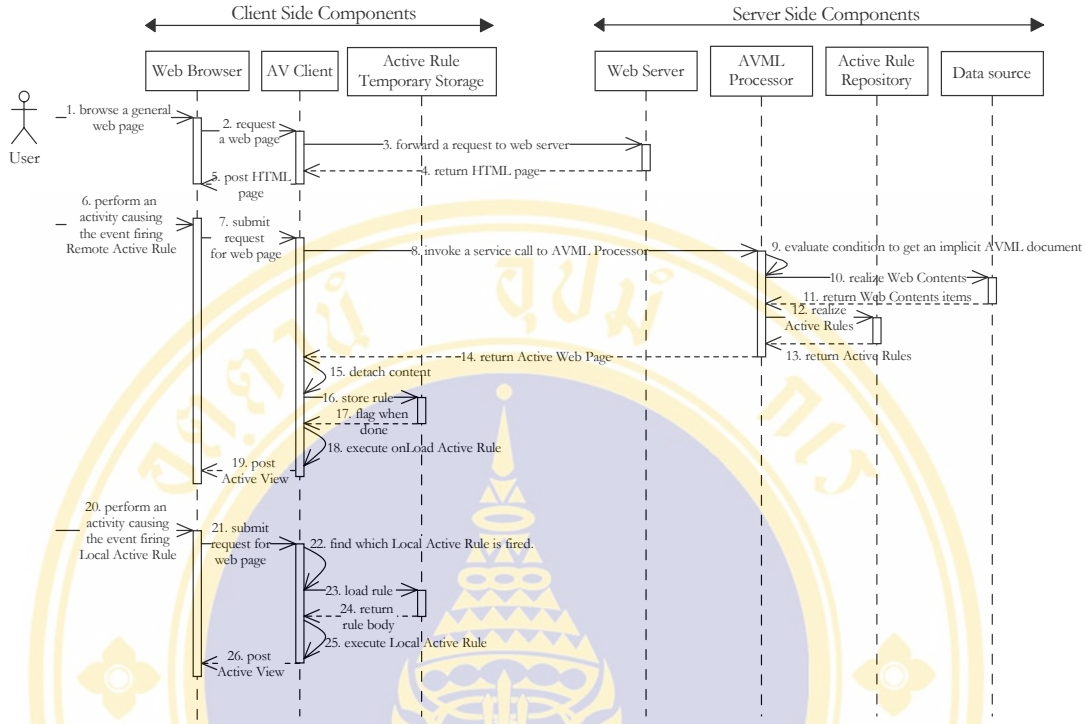


Figure 3-15 Sequence Diagram of Generic Web Page/Graphical File and Active Rules Execution Procedure

From the step #6 later, we assume that an Active Web Page had been responded to a client side and the AV Client executed an onLoad Active Rule in the Active Web Page, and a derived Active View was posted to the user's web browser. Afterward, the user starts performing any activities on the current Active View. Here are the details of each step:

6) We suppose that the user performs an activity on the current Active View causing a particular event that would fire a Remote Active Rule.

7) The AV Client receives a user request which contains the Rule ID. of the Remote Active Rule triggered and a User Activity document, and finds this triggered Remote Active Rule in the Active Rules temporary storage on the user's machine. Then, it does load a data binding part of the triggered Remote Active Rule, bind the data items, and pack them into the Service Call Parameters document.

8) The AV Client invokes a service call for executing the triggered Remote Active Rule, including the service call parameters document, to the AVML Processor.

Copyright by Mahidol University

After that, the AV Client pauses to await the result of the AVML Processor processing for the time being.

9) The AVML Processor starts processing the service call request by firstly reading the Rule Body part of the triggered Remote Active Rule from the Active Rules temporary storage into memory and secondly evaluating its condition part. The AVML Processor finalizes the evaluation result to get an appropriate implicit AVML document that is going to be returned.

10) Now, the materialization process begins. The AVML Processor acquires the actual values of data items, required in the implicit AVML document, from the data source such as a database server.

11) The data source gives back the values of requested data items.

12) Also, the AVML Processor cooperates with the Active Rules repository in obtaining the Active Rules required in the implicit AVML document.

13) The Active Rules repository gives back all the requested Active Rules.

14) At the present time, an explicit AVML document (or a new Active Web Page) is well-prepared. The AVML Processor returns it to the AV Client as the result of the service call processing.

15) The AV Client is resumed by the receiving Active Web Page and detaches its Web Contents section and its Active Rules section from the body of original Active Web Page.

16) The AV Client stores the detached Active Rules in the Active Rules temporary storage on the user's machine.

17) The AV Client is aware once the operation is done.

18) The AV Client starts to execute an onLoad Active Rule.

19) The AV Client sends back the resulting Active View to the web browser.

20) Assume that a few minutes later, the user performs an activity on the current Active View causing a particular event that would fire a Local Active Rule.

21) The AV Client receives a user request which contains the Rule ID. of the Local Active Rule triggered and a User Activity document and finds this triggered Local Active Rule in the Active Rules temporary storage on the user's machine.

Copyright by Mahidol University

22) Rather than asking the AVML Processor for processing, the AV Client just finds the Local Active Rule triggered already existing in the Active Rules temporary storage.

23) Therefore, the AV Client does read the desired Local Active Rule from the Active Rules temporary storage into memory.

24) The AV Client is prompted to receive the body of Local Active Rule returned.

25) The AV Client starts executing this Local Active Rule.

26) The resulting Active View is posted to the web browser.

3.8 A^2V Usage Guideline

The A^2V framework is not intentionally designed for using with the existing e-service applications, but for e-service applications built from scratch. However, this does not mean that the existing e-service applications can not comply with our framework, but the migration task is required. No matter what state of e-service applications is, to follow the A^2V framework, there are two tasks that need to be prepared as follows:

3.8.1 For Web Developers

This is the first task carried out by a web developer of an e-service application. The web developer has to analyze business constraints of the e-service application and list all the possible flows of web pages covering all constraints. The output at this stage is sort of a page transition diagram or an activity diagram. Later, step through the following sub-tasks orderly:

- *Step 1: Data Items Definition*: this is to define what the data will appear on an Active View of each web page and also where to obtain this data – for example, a database server.

Copyright by Mahidol University

- *Step 2: Active Rules Creation*: this is to define Active Rules that reflect the extracted business requirements, including the data required in an Active Rule execution, and store them in the Active Rules repository. To derive all Active Rules, we traverse the derived diagram by starting at a root node, in considering the following condition.

  - *For each web page (if a page transition diagram)/activity process (if an activity diagram) found*, represent by a Local Active Rule whose contents are introduced in Table 3-4.

Table 3-4 The Body of a Local Active Rule

Body of Active Rule	What Will Reside in
1. Data Binding	Put the items required in the web page which the Local Active Rule represents by following the simple type of XLink specification in term of the User Profile items or the Global Data items.
2. Condition-Action	Represent the source code of web page in the XSL specification.

  - *For each decision point found*, represent by a Remote Active Rule whose contents are introduced in Table 3-5.

Table 3-5 The Body of a Remote Active Rule

Body of Active Rule	What Will Reside in
1. Data Binding	Put the items required in a condition evaluation done by the AVML Processor by following the simple type of XLink specification in term of the User Profile items or the Global Data items.
2. Condition	Obtain from a decision point.
3. Action	Specify the desired two implicit AVML documents if a

Copyright by Mahidol University

Body of Active Rule	What Will Reside in
	condition is satisfied and unsatisfied respectively.

- *Step 3: Implicit AVML Documents Creation*: this is to map out references of data and Active Rules already defined. An implicit AVML document is typically descended from considering the diagram. This is done by keeping the following idea in mind.
  - *Start from a root of diagram and consider a decision point that will be coming.*
  - *If a decision point was found, it always came out with the two implicit AVML documents – one satisfied a condition, see Table 3-6 and the other one not satisfied a condition, see Table 3-7 .*

Table 3-6 The Body of an Implicit AVML Document Satisfying a Condition

Body of AVML Document	What Will Reside in
1. Local Active Rule(s)	Collect all Local Active Rule references describing the web pages next to the previous decision point until the next one.
2. Remote Active Rule(s)	Collect all Remote Active Rule references describing the next decision point.
3. Web Contents Item(s)	Specify Web Contents references required by Local Active Rules and Remote Active Rules as stated above. Section 3.4 has given the details of this.

Table 3-7 The Body of an Implicit AVML Document Not Satisfying a Condition

Body of AVML Document	What Will Reside in
1. Local Active Rule	Collect all Local Active Rule references describing the web page next to the previous decision point only.

Copyright by Mahidol University

Body of AVML Document	What Will Reside in
2. Remote Active Rule	Collect all Remote Active Rule references describing the current decision point only.
3. Web Contents Item(s)	Specify Web Contents references required by Local Active Rule and Remote Active Rule as stated above.

3.8.2 For Users

For a web end-user who wants to surf e-service applications implemented by the A^{2}V framework, he/she must perform the following steps:

- *Step 1*: A user starts visiting an e-service application through a web browser and finds a web page that provides for a hyperlink for downloading the AV Client application.
- *Step 2*: The user activates the hyperlink to download the AV Client application.
- *Step 3*: The user starts to install the AV Client application on his/her machine. From now onward, the AV Client is ready to work out.

In the next chapter, we will walk through an implementation of our A^{2}V framework in details.

Copyright by Mahidol University

CHAPTER IV
IMPLEMENTATION OF A^{2}V

This chapter describes how to develop our A^{2}V framework and gives the details for implementation. The prototype of A^{2}V will be used to prove whether an Active View construction conducted in a decentralized style could reduce the communication process between a client and the server. First, we introduce the modular design of our A^{2}V framework that is divided into two main modules which are the AV Client at a client side and the AVML Processor at a server side. We also discuss about the data and functional design for each of the modules. Next, the communication interface between these two main elements is addressed. In the end, the A^{2}V implementation environment is explained in terms of hardware and software requirements.

4.1 A^{2}V Data Design

This section describes the design of data in the A^{2}V framework, which includes Local Active Rule, Remote Active Rule, User Activity document, Merged Items document, Service Call Parameters document.

4.1.1 Implementation of Local Active Rule

A Local Active Rule is expressed in term of an XML-based structure. The AVML elements for Local Active Rule are earlier specified in Section 3.4 of Chapter III.

Copyright by Mahidol University

4.1.2 Implementation of Remote Active Rule

For the sake of simplicity, we use relational databases for storing Remote Active Rule. The schemas of such relational databases are corresponding to the AVML elements for Remote Active Rules which are specified earlier in Section 3.4 of Chapter III. Figure 4-1 shows the entity-relationship (E-R) diagram of a Remote Active Rule whilst the data dictionary of a Remote Active Rule are provided in Appendix A. In addition, for the search operation, an implementation by using relational databases might offer the better performance rather than utilizing an XPath.

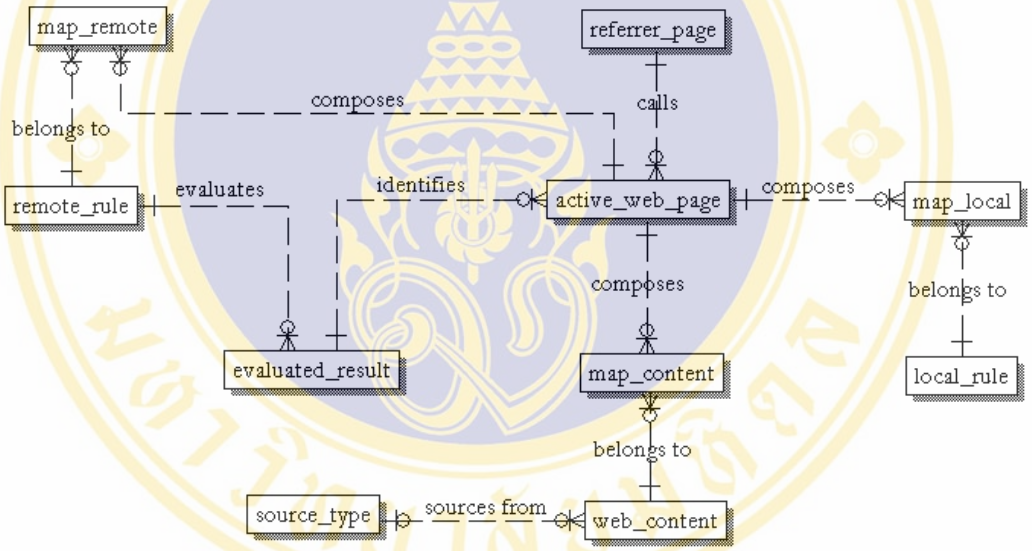


Figure 4-1 Entity-Relationship Diagram of Remote Active Rule

Figure 4-1 depicts the logical structural design of a Remote Active Rule through an E-R diagram. Having the result of condition evaluation of a Remote Active Rule represented by the relation “Remote Rule”, the AVML Processor has to match it with the proper one in the relation “Evaluated Result” including the consideration of the relation “Referrer Page” to derive the destined implicit AVML document in the relation “Active Web Page”. The relation “Active Web Page” describes general attributes and provides the mappings of an implicit AVML document as follows:

Copyright by Mahidol University

- The mapping of data items, which will be imbedded in the Web Contents section of this implicit AVML document, is represented by the relations “Map Content” and “Web Content”.
- The mapping of Local Active Rules, which will be imbedded in the Active Rules section of this implicit AVML document, is represented by the relation “Map Local” and “Local Rule”.
- The mapping of Remote Active Rules, which will be imbedded in the Active Rules section of this implicit AVML document, is represented by the relation “Map Remote” and “Remote Rule”.

4.1.3 User Activity Document

This is a XML-based document that carries data given when a user submits a request. It is obtained by the AV Client to update the existing Global Data items and/or the User Profile items in a user’s machine – we will further discuss in details in Section 4.2.1.2. A User Activity document is created by the JavaScript method [29] within an Active View and conforms to the specification depicted in the graphical representation in Figure 4-2. The XML Schema representation of a User Activity document will be provided in Section B.4 of Appendix B.

Copyright by Mahidol University

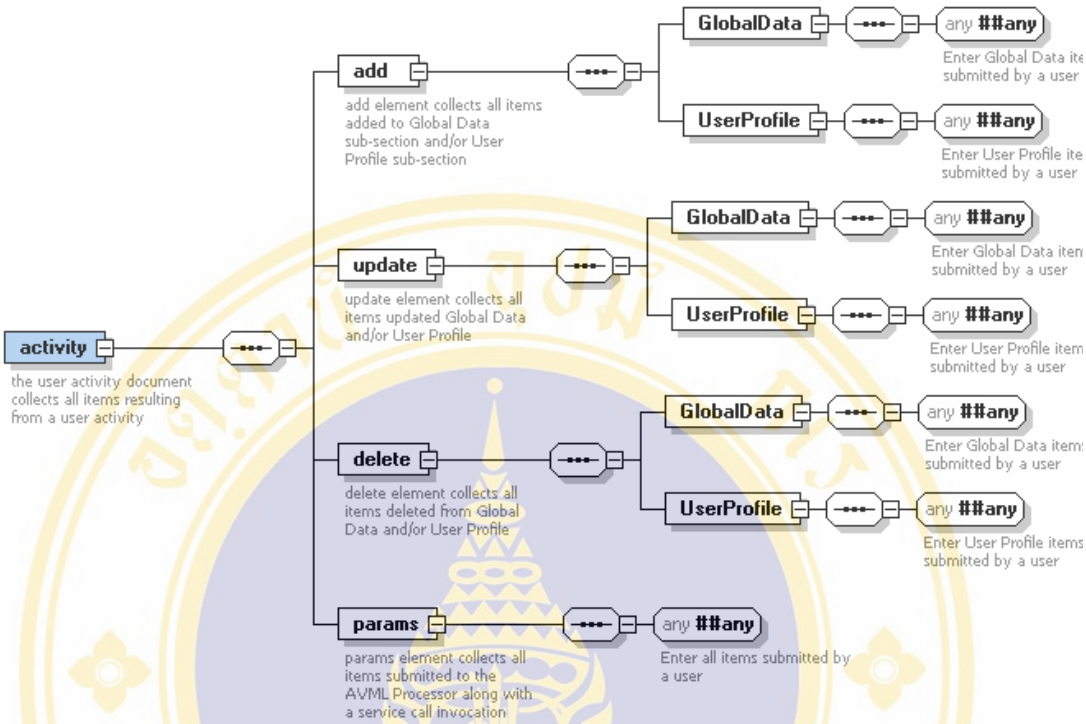


Figure 4-2 The Graphical Representation of a User Activity Document

4.1.4 Merged Items Document

This is an XML-based document data that comprises Global Data items and/or User Profile items that are really required in executing a Local Active Rule. It is created by the Parameter Binder sub-module, which will be further discussed in Section 4.2.1.3.2 and conforms to the specification in the graphical representation in Figure 4-3. The XML Schema representation of a Merged Items document will be provided in Section B.5 of Appendix B.

Copyright by Mahidol University

Figure 4-3 The Graphical Representation of a Merged Items Document

4.1.5 Service Call Parameters Document

This is an XML-based document that combines Global Data items and/or User Profile items and/or the parameters submitted thru a user's web browser. It is passed to the AVML Processor along with a service call invocation for executing a Remote Active Rule. Its definition is depicted in the graphical representation in Figure 4-4. The XML Schema representation of a Service Call Parameters document will be provided in Section B.6 of Appendix B.

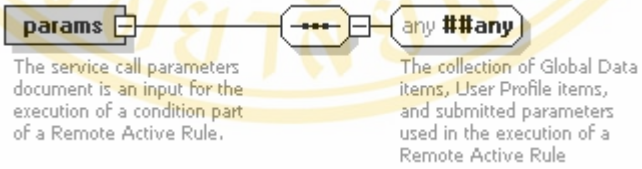


Figure 4-4 The Graphical Representation of a Service Call Parameters Document

Next, we go into the detail design of the A^2V components as discussed earlier in Section 3.5 of Chapter III.

Copyright by Mahidol University

4.2 A^{2}V Functional Design

As afore-mentioned in Section 3.5 of Chapter III, the A^{2}V framework is composed of the AV Client and the AVML Processor. The AV Client is seen as an Active View constructor whereas the AVML Processor serves as an Active Web Page provider. This section explores the details of these AVML processing units.

4.2.1 AV Client

The AV Client is comprised of Proxy Adaptor sub-module, User Activity Evaluator sub-module, and Lite Rule Processor sub-module. Additionally, the last one is featured by three sub-modules, Rule Loader, Parameter Binder, and Active View Generator.

4.2.1.1 Proxy Adaptor

Proxy Adaptor is a front-end interface sub-module accepting an incoming stream of data content sent from various sources and screening out its content prior to passing on to the AV Client core functions. In turn, it is in charge of serializing an outgoing content to the recipient components – a user's web browser or a web server.

As the class diagram displayed in Figure 4-5, the Proxy Adaptor sub-module is driven by ten classes which are grouped by their functions as noted here:

- *Proxy Handling classes* are the classes acting as a mini proxy and handling an incoming connection and an outgoing connection. Given the definition by Wikipedia [1], "a proxy server is a computer that offers a computer network service to allow clients to make indirect network connections to other network services. A client connects to the proxy server, then requests a connection, file, or other resource available on a different server. The proxy provides the resource either by connecting to the specified server or by serving it from a cache". As we found a fine source code of

Copyright by Mahidol University

simple Java Proxy Server, implemented by J. Plachy [30] which has been publicly listed on the Internet, we downloaded it and tailored upon our needs.

- *AV Client Handling classes* are the classes performing the AV Client functions. They are programmatically called within the proxy handling classes and are also under control of the proxy handling classes.

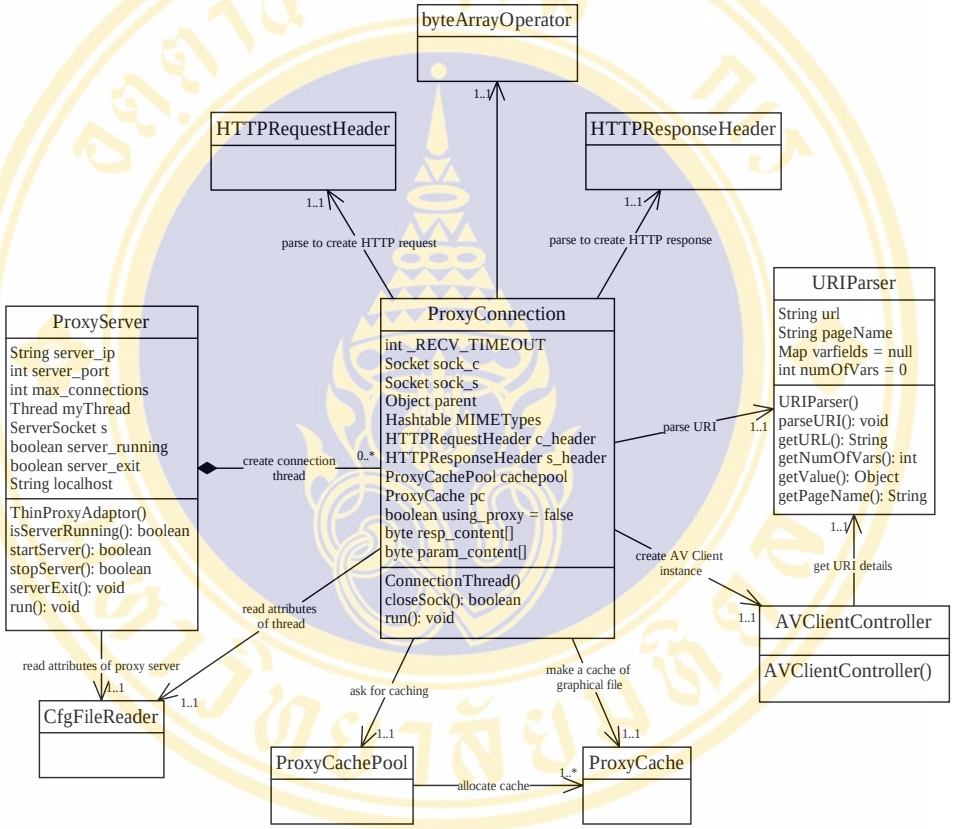


Figure 4-5 Class Diagram of Proxy Adaptor Module

The simplified proxy server [30] enables us to flexibly set its attributes through the configuration file named “proxy.cfg” according to our implementing environment. CfgFileReader class loads values of all properties from this file and makes them ready for being referenced. It is then called by ProxyServer class to initiate a proxy process that keeps on running if not manually be halted by a human intervention. To perform a proxy-like function, it keeps monitoring a coming connection from various sources. For an incoming connection detected, it creates a single thread of ProxyConnection class, the heart of Proxy Adaptor, to take into account. A stream of HTTP request

Copyright by Mahidol University

accompanied by the coming connection is validated against the given conditions and then carries on its way to the ending point. The header part of HTTP request stream is parsed into HTTPRequestHeader class so that the ProxyConnection class can easily refer to any tokens of HTTP request header – for example, a URI, a request method, and so forth. Besides, byteArrayOperator class supplies general utilities to operate the array of bytes structure since the data stream has originally been put on this structure.

For a request for a generic web page and a graphical files, their contents were responded from a web server after the ProxyConnection class has routed their request headers to a web server. Thenceforth, a response header received is parsed into HTTPResponseHeader class to be referenced by the ProxyConnection class whereas a content part is cached by ProxyCachePool class and ProxyCache class on a user's machine so that it would be at hand for the next time. Afterward, the content is posted to a web browser.

For a user request which needs to process an Active Rule, URIParser class and AVClientController class get involved. ProxyConnection class parses the requested URI into URIParser class which provides more methods to get details of URI including passing parameters. Especially, *the Rule ID.*, *the Rule Type*, and *the e-service application name* of the triggered Active Rule are the important arguments for AVClientController class. It is going to read the related User Profile and Global Data to memory and also happens to load the fired Active Rule identified by those three parameters from the Active Rules temporary storage. The ongoing Active Rule takes the User Profile and the Global Data as inputs in an execution phase.

The activity diagram in Figure 4-6 gives much more details what ProxyConnection class operates. It begins to read a coming content from an input socket and assign to an array of bytes frame. Here, we get bytes of request header as illustrated in Example 4-1 for the GET method and Example 4-2 for the POST method below and do parse them into HTTPRequestHeader class in order to be easily referenced. It then checks if a type of request method is GET or POST so that it knows how to acquire the parameters going with this coming request. Such these parameters

Copyright by Mahidol University

are vital for an Active Rule execution. If the request method is “GET”, the parameters will be contained in a URI string, otherwise – POST method, it gets to do further.

Example 4-1 Request’s Header by the GET method

```
1   GET
2   http://192.168.9.221:8080/ava/xml/index.xml?ruleid=1&ruletype=remote&ecAppName=www.amazon.com
2   HTTP/1.1
3   Host: 192.168.9.221:8080
4   Referrer: http://192.168.9.221:8080/ava/xml/index.xml
5   User-Agent: Mozilla/5.0 (compatible; WAPT; Windows)
6   Proxy-Connection: close
7   Accept: text/html, */*
8   Accept-Language: en-us, *
9   Accept-Encoding: compress, gzip, *
10  Pragma: no-cache
11
12
```

Example 4-2 Request’s Header by the POST method

```
1   POST http://192.168.9.221:8080/ava/xml/signin.xml HTTP/1.1
2   Content-Type: application/x-www-form-urlencoded
3   Content-Length: 283
4   Host: 192.168.9.221:8080
5   Referrer: http://192.168.9.221:8080/ava/xml/index.xml
6   User-Agent: Mozilla/5.0 (compatible; WAPT; Windows)
7   Proxy-Connection: close
8   Accept: text/html, */*
9   Accept-Language: en-us, *
10  Accept-Encoding: compress, gzip, *
11  Pragma: no-cache
12
13
```

In the case of the GET method, the URI string, the line #2 of Example 4-1, is parsed into URIParser class to find whether the parameters – Rule ID., Rule Type, and name of e-service name – are there or else it is categorized as a generic request for a HTML file or a graphical file. Pairs of parameters are next to the question mark (?) as a boldface. Each pair of parameters is delimited by an ampersand (&) and expressed in the form of “name=value”. In the line #2 of Example 4-1, there are three parameters

Copyright by Mahidol University

named “ruleid”, “ruletype”, and “ecAppName” and their values are “1”, “remote”, and “www.amazon.com” severally.

If the graphical file or the HTML file are detected, ProxyConnection class has to verify a type of requested data. If a graphical type is detected, it will look the matching cached content up in the specified directory in a user’s machine and streams the content to a user’s web browser. If the cache is not found or other types are discovered, it needs to forward the request to a web server. Also, in the case that al the three parameters are not seen in the URL, it will go the same. Otherwise, this would be classified as an activity a user has performed on a current Active View that causes the event potentially triggering an Active Rule. The ProxyConnection class designates AVClientController class to be in charge for constructing an Active View. Once a new Active View is in place, it will generate a new response header and then give the Active View back to a user’s web browser.

Copyright by Mahidol University

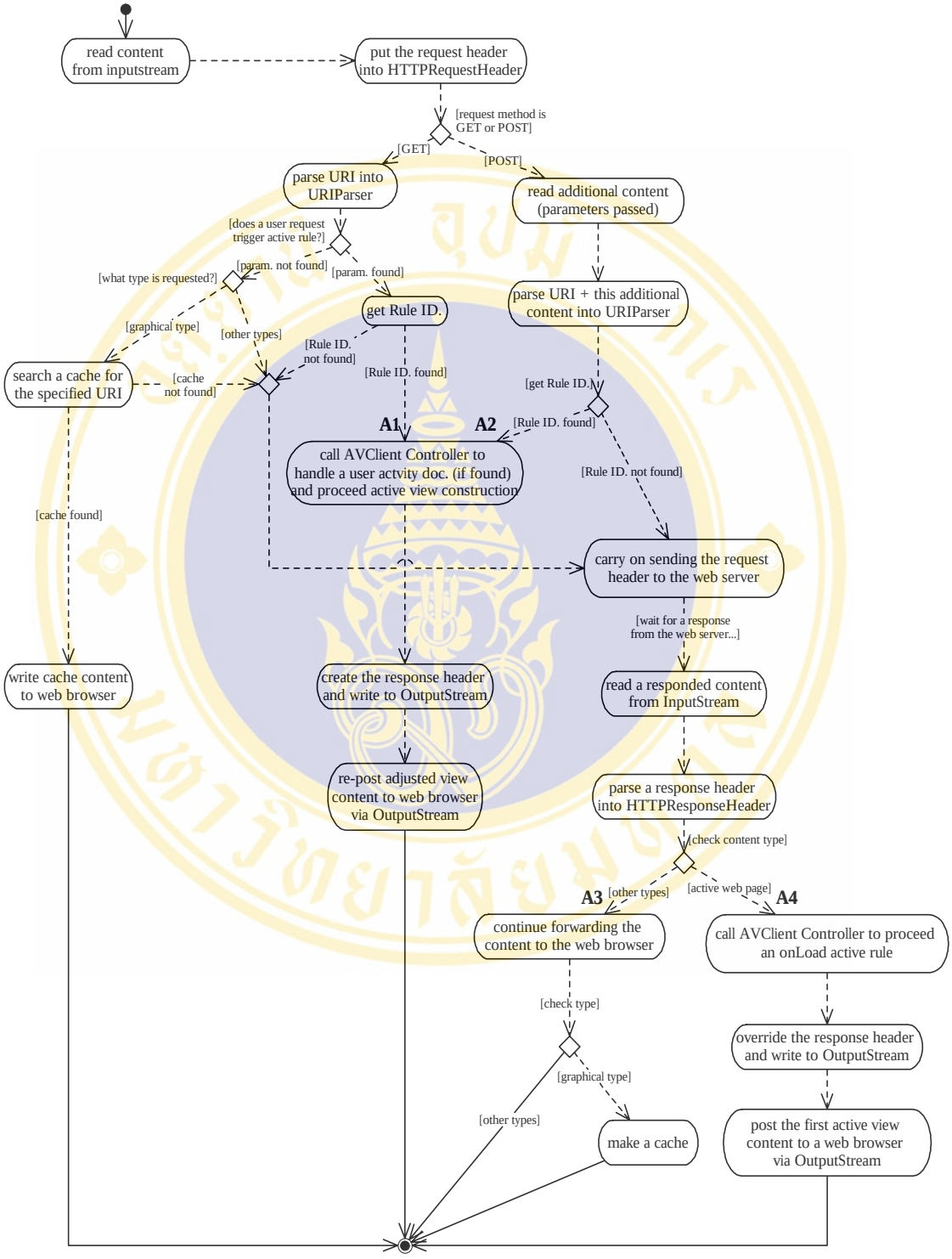


Figure 4-6 Activity Diagram of ProxyConnection Class

Copyright by Mahidol University

Example 4-3 An Example of Submitted Parameters

```
ruleid=2&ruletype=remote&ecAppName=www.amazon.com&activity=%3Cactivity%3E%3Cparams%3E%3
Cusername%3Eriam@gmail.com%3C%2Fusername%3E%3Cpassword%3Epassword%3C%2Fpassword%3E%3C%2
Fparams%3E%3C%2Factivity%3E&email=riam@gmail.com&action=sign-
in&password=password&Continue.x=125&Continue.y=4
```

On the other hand, in the case of the POST method, ProxyConnection class first reads additional contents – a parameters string – from an input pipeline as illustrated in Example 4-3. Like the GET method's parameters string, a pair of parameters of POST method is delimited by an ampersand and formed by "name=value". Second, ProxyConnection class combines the URI string (the boldface in the line #1 of Example 4-2) with this parameters string, parses them into URIParser class and then scans through the string to get the Rule ID. parameter. Third, if the Rule ID. is found, it will prompt AVClientController class to execute the Active Rule identified by this Rule ID. and ProxyConnection class will result the response Active View to a user's web browser. Alternatively, ProxyConnection class will treat the request like a generic web page/a graphical file as labeled A3 in Figure 4-6. In this case, it sends on to a web server. Upon a response for a requested web page/a requested graphical file from a web server, ProxyConnection class obtains the responded content from an input stream and parses it into HTTPResponseHeader class to check what the content type is. If the type goes to a plain HTML file or a graphical file, it will carry on sending the content to a user's web browser – in the case of the multimedia type, ProxyConnection class will need to make a cache prior to returning. Otherwise, the type is XML – we scope out an XML-related type to be an Active Web Page. *This is specific for the purpose of experiments conduction.* We create a dummy Active Web Page and place on a web server. The dummy Active Web Page holds a Remote Active Rule which results in another Active Web Page being yielded by the AVML Processor, once the action was made to the Active View. The dummy Active Web Page is directly called by that the user enters the URI to get it from a web server to a web browser's address bar. It is called once for each experimental testing as an introductory Active View that offers an activity inducing other new Active Web Pages brought out by the AVML Processor as usual. Therefore, when asked by a user, the

Copyright by Mahidol University

dummy Active Web Page follows the label A4 whereas all user requests are handled by the ProxyConnection class according to the labels A1 or A2 in Figure 4-6.

The sequence diagram in Figure 4-7 shows how to handle a coming user request which has been made on a current Active View appearing in a user's web browser. We are going through each step as follows:

1) ProxyServer class creates a new thread of ProxyConnection class upon an incoming user request received from a user's web browser.

2) ProxyConnection class reads a coming content – a content of request header – from an input stream.

3) ProxyConnection class parses the content into HTTPRequestHeader class.

4) A new instance of HTTPRequestHeader class is given back to the parent ProxyConnection class.

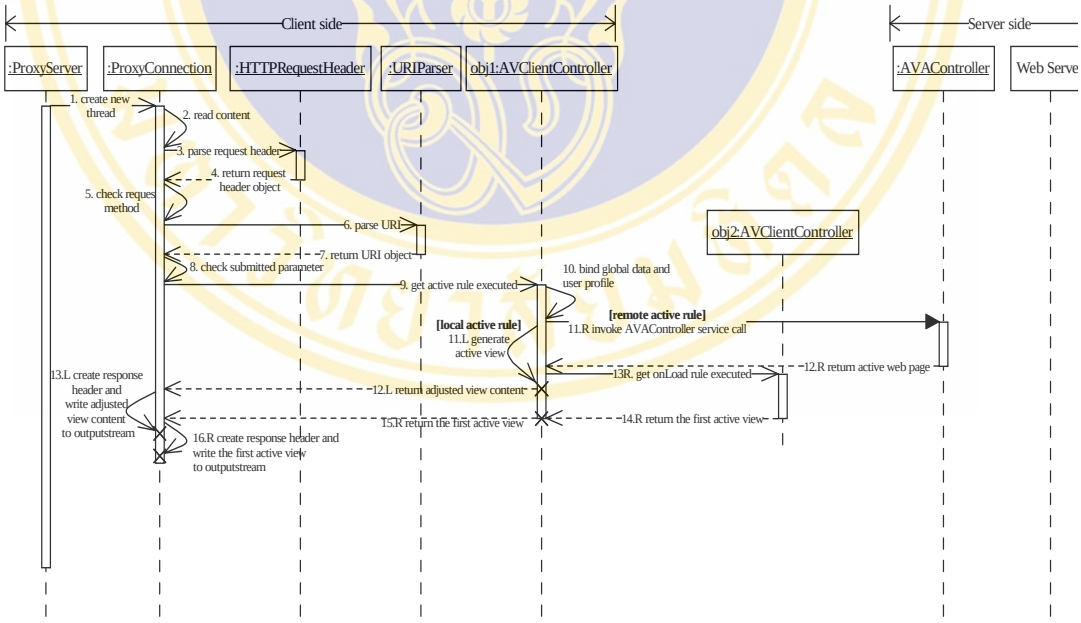


Figure 4-7 Sequence Diagram of Proxy Adaptor Module

5) ProxyConnection class determines what a request method is – GET or POST. If POST, it will need to read more contents (a parameters string) from the input stream.

Copyright by Mahidol University

6) ProxyConnection class refers to the HTTPRequestHeader instance to obtain a URI string and then parses it into URIParser class.

7) A new instance of URIParser class is returned to the parent ProxyConnection class.

8) ProxyConnection class refers to the URIParser instance to find the Rule ID. parameter.

9) ProxyConnection class creates a new instance of AVClientController class to sort out an Active Rule execution with respect to the specified Rule ID.

10) AVClientController class binds a User Profile file and a Global Data file in a user's machine (if existing) with the parameters received and prepares all data stated in the data binding part of Active Rule by taking from the User Profile and the Global Data. Also, it finds out what the type of Active Rule is.

For a Local Active Rule

11.L) AVClientController class proceeds an action part to construct an Active View.

12.L) AVClientController class returns the content of constructed Active View to its parent ProxyConnection class.

13.L) ProxyConnection class creates a response header and writes out either the response header or the constructed Active View content into the output stream.

For a Remote Active Rule

11.R) AVClientController class invokes a service call to AVAController class (the main class of the AVML Processor) along with all concerned parameters to execute a Remote Active Rule.

12.R) AVAController class returns an output of Remote Active Rule execution – a new Active Web Page to its caller class – obj1 instance.

13.R) Once the new Active Web Page arrives at obj1 instance, obj1 instance creates another instance of AVClientController class called obj2. obj2 finds an onLoad Active Rule (a kind of Local Active Rule) and gets it executed.

Copyright by Mahidol University

14.R) obj2 instance returns the content of the first Active View to its parent instance (obj1).

15.R) obj1 instance continually forwards the content received to ProxyConnection class.

16.R) proxyConnection class creates a response header and then writes out the response header and the first Active View content into the output stream.

The details of step#11 – step#15 will be discussed in the subsequent modules.

4.2.1.2 User Activity Evaluator

This module is **UAE** (short for User Activity Evaluator) which is responsible for handling the following coming inputs as follows:

- *Upcoming Active Web Page*: The AVML Processor has given back a new Active Web Page to the AV Client upon the condition part evaluation of Remote Active Rule. Once the Active Web Page is accepted, the UAE detaches a Web Contents section and an Active Rules section from the body of Active Web Page and stores them in the specified directory and the Active Rules temporary storage in a user's machine respectively. In addition, it prompts the Lite Rule Processor to proceed an onLoad Active Rule of this coming Active Web Page.

- *Upcoming User Request*: this results from that a user has performed some activities (e.g. button pressing, HTML link clicking) on the current Active View appearing on a user's web browser. Technically, a web developer adds in the Rule ID., Rule Type, and e-service application name parameters to be hidden input types (<input type="hidden"/>) of the HTML form element (<form></form>). When the HTML form is submitted, these three parameters are passed along with if any other parameters. There might be a parameter named "activity" passed alongside the afore-mentioned parameters. It is termed *User Activity document* containing variable names and their values to be added, updated, or deleted to the existing Global Data file and User Profile file on a user's machine. The UAE has to look for the "activity" parameter to determine what has to be done with the existing Global Data and User

Copyright by Mahidol University

Profile files. Afterward, it informs the Lite Rule Processor module of which Active Rule, belonging to the latest coming Active Web Page, will be executed as identified by the Rule ID., the Rule Type, and the e-service application name parameters.

We list all classes featuring in the UAE module including their relationship as shown in the class diagram in Figure 4-8.

For a coming Active Web Page received from the AVML Processor, AVClientController class, newly created by ProxyConnection class, refers to a content of Active Web Page for doing the following processes. First, AVClientController class passes the argument to ActiveWebPageImpl class which gets information out for further uses and creates a new ActiveWebPageBean class. ActiveWebPageImpl class gives SimpleXMLParser class, featured by JDOM [31], the content of Active Web Page to parse and make the details of Active Web Page available for being referenced by other classes.

After parsing, ActiveWebPageImpl class gets pieces of Active Web Page information by means of XPath mechanism through SimpleXPath class featured by Saxon [32]. Likewise, there is another class called SimpleXMLTransformer referenced by ActiveWebPageImpl class. SimpleXMLTranformer class departs the Active Rules section and some attributes – for instance, name of Active Web Page, e-service application name, referrer web page – from the original Active Web Page by means of XSLT transformation scheme also featured by Saxon [32]. ActiveWebPageImpl class stores the detached Active Rules section as an XML-formatted file in the Active Rules temporary storage that is indeed the specified directory on the user's machine. Each Active Web Page possesses its own Active Rules, so we constitute this XML filename by syntax of "<Active Web Page name>_ar.xml" (<Active Web Page name> will be substituted by the name of the ongoing Active Web Page). Example 4-4 shows an example of detached Active Rules section. With respect to the functions of SimpleXMLParser, SimpleXPath, and SimpleXMLTransformer classes, ActiveWebPageImpl class is prompted by a full of information to create ActiveWebPageBean class that is such a data entity bean. Other classes can obtain details of Active Web Page through this class.

Copyright by Mahidol University

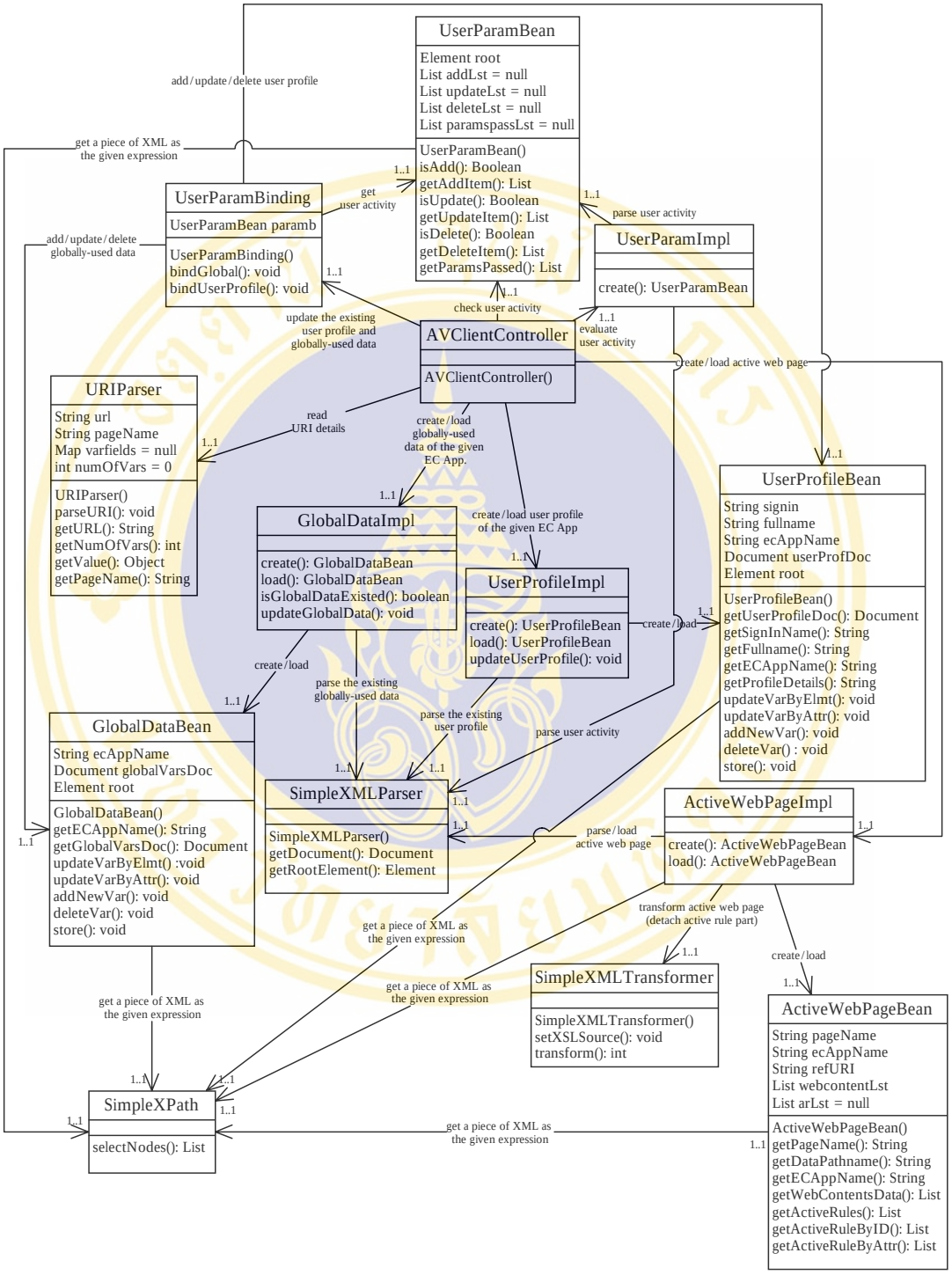


Figure 4-8 Class Diagram of User Activity Evaluator Module

Copyright by Mahidol University

Example 4-4 An Example of the Detached Active Rule

```
<?xml version="1.0" encoding="UTF-8"?>
<ActiveWebPage pageid="1" name="signin.xml" ecAppName="www.mybook.com"
lastUpdated="2005-07-05" referrerURI="http://localhost:8080/ava/xml/signin.xml">
 <ActiveRules>
  <ar id="1" name="signin-firstly" type="local" frequency="onLoad">
   <DataBinding>
    <global/>
    <UserProfile/>
   </DataBinding>
   <event/>
   <CondAction>
    <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
…
    </xsl:stylesheet>
   </CondAction>
  </ar>
</ActiveWebPage>
```

Second, AVClientController class goes through the Web Contents section. AVClientController class visits the Global Data sub-section and gives GlobalDataImpl class all items of this sub-section to store in an XML-based document, named "*globaldata.xml*", under the directory specified by the name of e-service application name – e.g. if the name of application name is www.mybook.com, we will get *www.mybook.com*\globaldata.xml – and store in GlobalDataBean class that contains all Global Data items. In the case of that the Global Data file has already existed, GlobalDataImpl class needs to load it onto GlobalDataBean class in being parsed by SimpleXMLParser class and keep it refreshed with the arrival items by using of SimpleXPath class addressing the destined items. Example 4-5 shows an example content of Global Data file.

Example 4-5 An Example of a Global Data File

```
<?xml version="1.0" encoding="UTF-8"?>
<global used="true">
 <product id="00000000001" title="The Emancipation of Mimi" author="Mariah Carey"
category="Audio CD" price="00000019.99" currency="$" discount="0.25" quantity="1"
latestadded="2005-07-05" unticked="false" />
```

Copyright by Mahidol University

```
 <product id="00000000002" title="The Hours (Widescreen Edition)" author="Stephen Daldry" category="DVD" price="00000009.99" currency="$" discount="0.25" quantity="1" latestadded="2005-07-05" unticked="false" />
 <product id="00000000003" title="Erin Brockovich" author="Steven Soderberg" category="DVD" price="00000014.98" currency="$" discount="0.20" quantity="1" latestadded="2005-07-05" unticked="false" />
</global>
```

Then, AVClientController class next explores the User Profile sub-section. Similarly, AVClientController class delivers all User Profile items to UserProfileImpl to store in an XML-based document, named "*userprof.xml*", in the directory specified by the name of e-service application name – e.g. *www.mybook.com*\userprof.xml – and to store in UserProfileBean class in any cases. If found, the existing User Profile file will be loaded onto UserProfileBean class by UserProfileImpl class in being parsed by SimpleXMLParser class. In that event, UserProfileImpl class has to update items in UserProfileBean class with the arrival items by using of SimpleXPath class addressing the destined item. Example 4-6 pictures an example content of User Profile file.

Example 4-6 An Example of a User Profile File

```
<?xml version="1.0" encoding="UTF-8"?>
<UserProfile used="true">
 <userinfo id="00001" username="riam@gmail.com" password="password" title="Ms." firstname="Riam" lastname="Samutrnava" fullname="Riam Samutrnava" type="2" age="27" occupation="Call Center Operator" region="TH" />
 <shippingaddress id="000003" recipient="Riam at PCC" addrno="142 4th Floor KBANK Bldg." road="Silom" subdistrict="Silom" district="Suriyawongse" city="Bangkok" country="Thailand" postcode="10500" selected="true" />
 <billingaddress id="000003" recipient="Riam at PCC" addrno="142 4th Floor KBANK Bldg." road="Silom" subdistrict="Silom" district="Suriyawongse" city="Bangkok" country="Thailand" postcode="10500" selected="true" />
 <paymentmethod id="51042" cardno="51042" cardtype="visa" expirymonth="12" expiryyear="2007" cardholder="Riam Samutrnava" selected="true" />
 <shippingoptions id="000001" shippingspeed="pri-intl-us-asia" shipsplitpreference="shipWhenever" />
</UserProfile>
```

Copyright by Mahidol University

For a coming user request which triggers one Active Rule in the Active Rules temporary storage, ProxyConnection class creates AVClientController class in passing with URIParser class and a request method. Firstly, AVClientController class signals ActiveWebPageImpl class to read the detached Active Rules of current Active Web Page, specified by the URI string gotten from URIParser class, from the Active Rules temporary storage and passes it to SimpleXMLParser class. Then, ActiveWebPageImpl class gets attributes of the Active Web Page and the contents of Active Rules from the parsed XML content through SimpleXPath class and builds up ActiveWebPageBean class.

At last, AVClientController class takes values of *User Activity* document expressed in an XML notion and passes them to UserParamImpl class to parse thru SimpleXMLParser class and construct the instance of UserParamBean class. The User Activity document is created by the JavaScript [29] in boldface in Example 4-12 at the time a user action is performed. In a designing view, a web developer has to put this JavaScript text in the condition-action part of Local Active Rule where a user would like the action, made on its Active View, to cause another Active Rule to be triggered. Later, AVClientController class determines if any items has to be added, updated, or deleted to either the existing Global Data file or the User Profile file. If this is a case, it will leave this task to UserParamBinding class which cooperates with GlobalDataImpl class and UserProfileImpl class in preparing GlobalDataBean class and UserProfileBean class for being refreshed by all the items from UserParamBean class. To achieve this, GlobalDataBean class and UserProfileBean class require SimpleXPath class to get access to items for each.

Example 4-7 An Example of a User Activity Document

```
<activity>
 <update>
  <UserProfile>
   <shippingaddress id="000003" selected="true"/>
  </UserProfile>
 </update>
</activity>
```

Copyright by Mahidol University

As the element “update” depicted in Example 4-7, UserParamBinding class has to *modify* the “shipping” item in the UserProfileBean class with `<shippingaddress id="000003" selected="true"/>`.

Next, we present actions done by the UAE in an activity diagram subjecting to two kinds of inputs from ProxyConnection class.

4.2.1.2.1 Upcoming Active Web Page

As the activity diagram shown in Figure 4-9, the UAE first creates a new Active Web Page object upon a content of Active Web Page received. From now on, any processes can get information from this Active Web Page object.

Second, the UAE starts verifying the “used” attribute of the Global Data section, indicating whether Global Data items are used by any embedded Active Rules of the Active Web Page. If its value goes to false, the UAE continuously will have to check if any new items are found in the Global Data sub-section of the receiving Active Web Page. If this is the case, the UAE will yet do another checking of an existence of a Global Data file on a user’s machine. If the file does not exist, a new Global Data file will be created in the directory noted earlier in the previous section and all the new items will be stored in this file as well. Otherwise, the file has existed, it will be read into a Global Data structure and all new items will be updated to this structure prior to storing in the flat file.

Let us back off to the case that the “used” attribute went to true, similarly the UAE performs checking of an existence of a Global Data file. If it is not present, a Global Data file will newly be created and stored. Otherwise, the UAE will read the file into the Global Data structure and then check if any new items are found. If this is the case, the new items will be updated to the Global Data structure which will be stored as a file.

Lastly, this turn is for updating of a User Profile. Like an updating of a Global Data, the UAE starts verifying the “used” attribute of the User Profile sub-section indicating whether User Profile items are required as inputs of an embedded Active Rule of an Active Web Page. If its value comes to false, the UAE continuously has to

Copyright by Mahidol University

check if any new items are found in the User Profile sub-section of the receiving Active Web Page. If this is the case, the UAE will do yet another checking of an existence of a User Profile file on a user's machine. If the file does not exist, a new User Profile file will be created in the directory noted earlier and all the new items will be stored in this file as well. Otherwise, the file has existed, it will be read into a User Profile structure and all new items will be updated to this structure prior to storing in the flat file.

Let us back off to the case that the "used" attribute went to true, similarly the UAE performs checking of an existence of a User Profile file. If it is not present, a User Profile file will newly be created and stored. Otherwise, the UAE will read the file into the User Profile structure and then check if any new items are found. If this is the case, the new items will be updated to the User Profile structure which will be stored as a file.

Copyright by Mahidol University

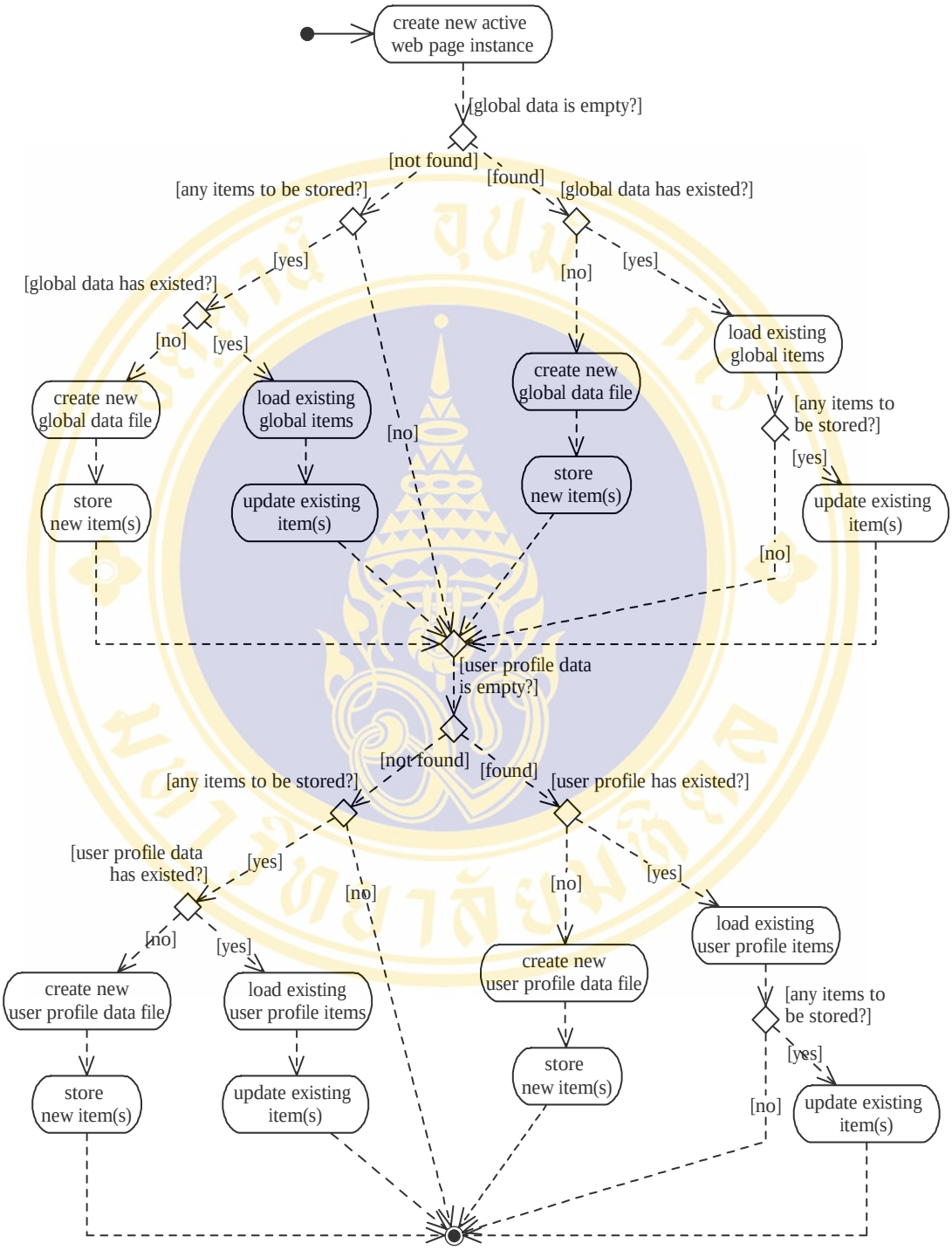


Figure 4-9 Activity Diagram of User Activity Evaluator Module Regarding an Upcoming Active Web Page

Copyright by Mahidol University

Subsequently, the sequence diagrams in Figure 4-10 and Figure 4-11 show how to sort out a coming Active Web Page received from the AVML Processor. The steps are described here:

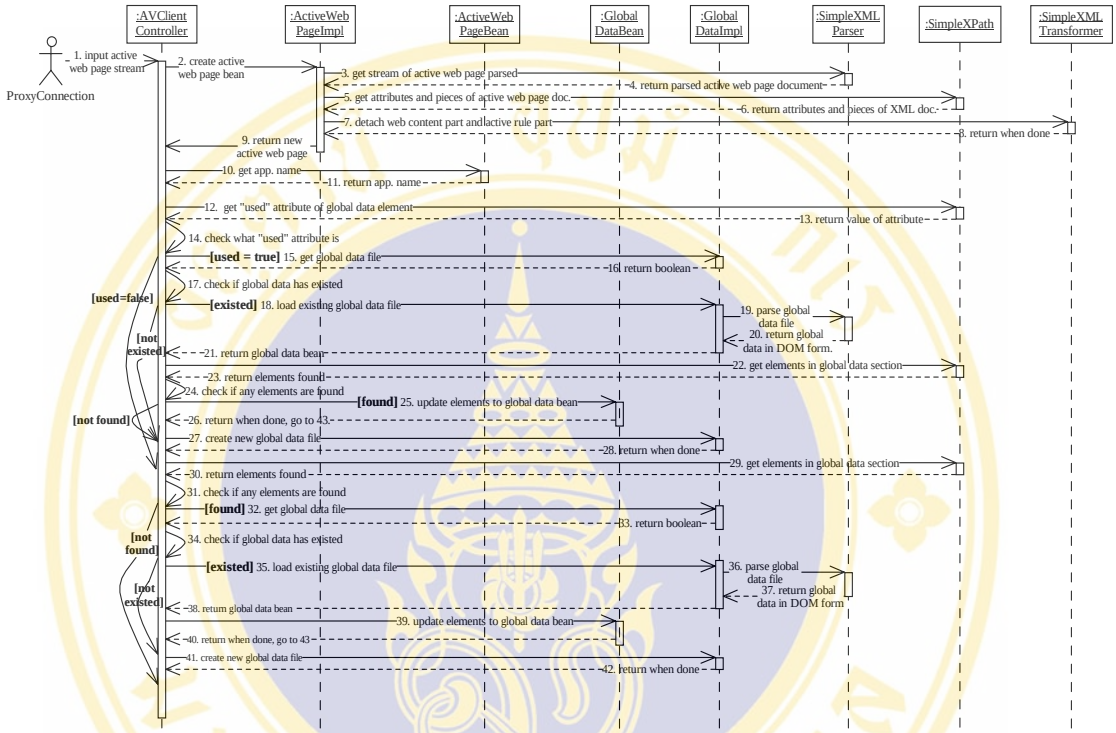


Figure 4-10 Sequence Diagram of User Activity Evaluator Module Regarding an Upcoming Active Web Page

1) ProxyConnection class creates an instance of AVClientController class with the input stream of Active Web Page content and the output stream where the first Active View of this coming Active Web Page is written as two incoming parameters.

2) AVClientController class calls ActiveWebPageImpl class by passing the stream of Active Web Page for dealing with a creation of Active Web Page object.

3) ActiveWebPageImpl class requests SimpleXMLParser class to parse the content of Active Web Page so that its information is available for further uses.

4) ActiveWebPageImpl class gets a response of DOM (short for Document Object model) content that represents the Active Web Page from SimpleXMLParser class.

5) ActiveWebPageImpl class requests SimpleXPath class to locate pieces of DOM content.

Copyright by Mahidol University

6) ActiveWebPageImpl class is prompted by the desired information of the Active Web Page from SimpleXPath class.

7) ActiveWebPageImpl class asks SimpleXMLTransformer class to detach an Active Rules section from the original Active Web Page. To accomplish this, we apply an XSL stylesheet to transform the DOM content to be like what is in Example 4-4.

8) ActiveWebPageImpl class is resumed when SimpleXMLTransformer class had done the transformation process. After that, ActiveWebPageImpl class collects other related information from the DOM structure to construct an instance of ActiveWebPageBean class.

9) AVClientController class gets a responded object of ActiveWebPageBean class from ActiveWebPageImpl class.

10) AVClientController class calls the method getECAppName() of ActiveWebPageBean object to get a name of e-service application where the Active Web Page came out.

11) The method returns the e-service application name string to AVClientController class.

12) AVClientController class requests SimpleXPath class to get a value of *used* attribute belonging to the Global Data sub-section.

13) AVClientController class gets the value of *used* attribute from SimpleXPath class.

14) AVClientController class checks what the used attribute is – true or false. If it appears true, it will continue on the step 15, otherwise, it will go to the step 30.

15) AVClientController asks GlobalDataImpl class to look for a Global Data file in the specified directory.

16) GlobalDataImpl class returns an existence status of Global Data file.

17) AVClientController class does check if the Global Data file is present with this returned status. If the file has already existed, it will continue on the next step, otherwise, it will skip down the step 27.

18) AVClientController class requests GlobalDataImpl class to read the existing Global Data file into memory.

19) GlobalDataImpl class calls SimpleXMLParser class for parsing the in-memory Global Data.

Copyright by Mahidol University

20) GlobalDataImpl class gets the Global Data in DOM form from SimpleXMLParser class and puts in Global Data bean structure.

21) GlobalDataImpl class returns the GlobalDataBean object to AVClientController class.

22) AVClientController class asks SimpleXPath class with passing the DOM structure of Active Web Page to get all the items under the Global Data sub-section.

23) SimpleXPath class returns all the items which would be found in the Global Data sub-section.

24) AVClientController class checks whether at least one item is found. If this is the case, it will continue on the next step, or else it will jump into the step 27.

25) AVClientController class requests the GlobalDataBean object to update the inside items with the arrival items.

26) The process is resumed on the step 43 once GlobalDataBean object has got the operation done.

27) AVClientController class calls GlobalDataImpl class to create a new Global Data file in the specified directory.

28) AVClientController class is prompted once GlobalDataImpl class got the creation of Global Data file done.

29) AVClientController class asks SimpleXPath class with passing the DOM structure of Active Web Page to get all items under the Global Data sub-section.

30) SimpleXPath class returns all items which would be found in the Global Data sub-section.

31) AVClientController class checks whether at least one item is found. If this is the case, it will continue on the next step, or else it will jump into the step 43.

32) AVClientController asks GlobalDataImpl class to look for a Global Data file in the specified directory.

33) GlobalDataImpl class returns an existence status of Global Data file.

34) AVClientController class does check if the Global Data file is present with this returned status. If the file has already existed, it will continue on the next step, otherwise, it will skip down the step 43.

35) AVClientController class requests GlobalDataImpl class to read the existing Global Data file into memory.

Copyright by Mahidol University

36) GlobalDataImpl class calls SimpleXMLParser class for parsing the in-memory Global Data.

37) GlobalDataImpl class gets the Global Data in DOM form from SimpleXMLParser class and puts in Global Data bean structure.

38) GlobalDataImpl class returns the GlobalDataBean object to AVClientController class.

39) AVClientController class requests the GlobalDataBean object to update the inside items with the arrival items.

40) The process is resumed on the step 43 once GlobalDataBean object has got the operation done.

41) In the case of that the Global Data file has not already existed, AVClientController class calls GlobalDataImpl class to create a new Global Data file in the specified directory.

42) AVClientController class is prompted once GlobalDataImpl class has got the creation of Global Data file done.

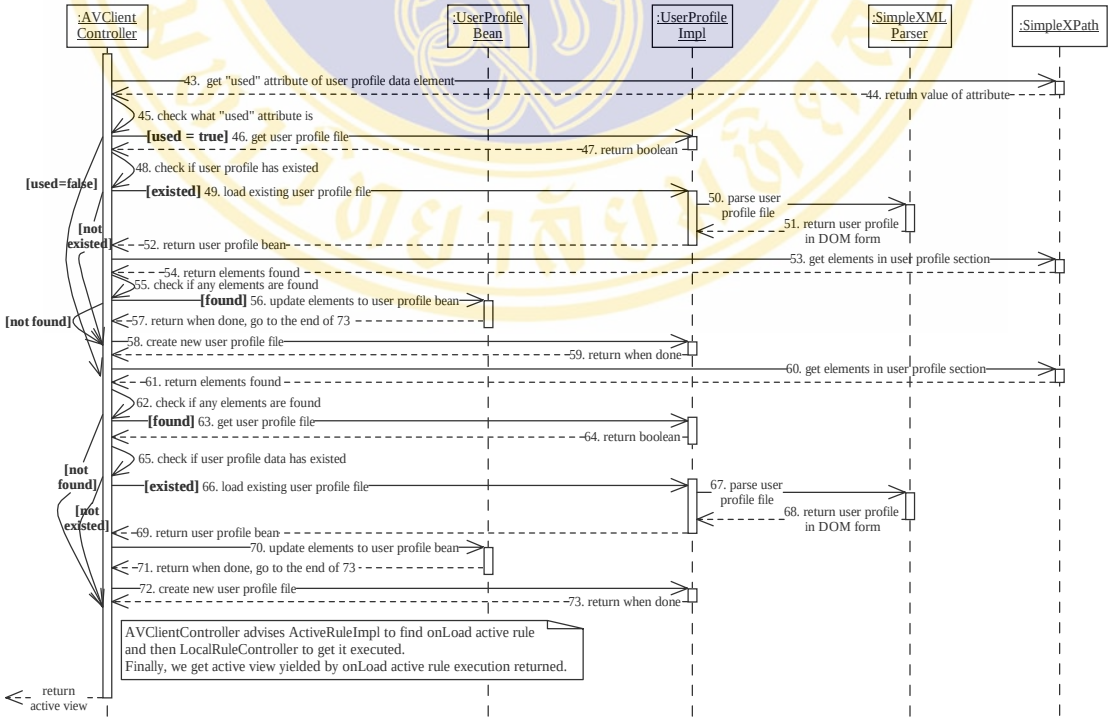


Figure 4-11 Sequence Diagram of User Activity Evaluator Module Regarding an Upcoming Active Web Page (cont.)

Copyright by Mahidol University

Continue in Figure 4-11, the following steps are for updating on the User Profile data. We get all steps appear to be similar to those of the Global Data.

43) AVClientController class requests SimpleXPath class to get a value of *used* attribute belonging to the User Profile sub-section.

44) AVClientController class gets the value of *used* attribute from SimpleXPath class.

45) AVClientController class checks what the used attribute is – true or false. If it appears true, it will continue on the step 46, otherwise, it will go to the step 61.

46) AVClientController asks UserProfileImpl class to look for a User Profile file in the specified directory.

47) UserProfileImpl class returns an existence status of User Profile file.

48) AVClientController class does check if the User Profile file is present through this returned status. If the file has already existed, it will continue on the next step, otherwise, it will skip down the step 58.

49) AVClientController class requests UserProfileImpl class to read the existing User Profile file into memory.

50) UserProfileImpl class calls SimpleXMLParser class for parsing the in-memory User Profile.

51) UserProfileImpl class gets the User Profile in DOM form from SimpleXMLParser class and puts in User Profile bean structure.

52) UserProfileImpl class returns the UserProfileBean object to AVClientController class.

53) AVClientController class asks SimpleXPath class with passing the DOM structure of Active Web Page to get all the items under the User Profile sub-section.

54) SimpleXPath class returns all the items would be found in the User Profile sub-section.

55) AVClientController class checks whether at least one item is found. If this is the case, it will continue on the next step, or else it will jump into the step 58.

56) AVClientController class requests the UserProfileBean object to update the inside items with the arrival items.

57) The process is resumed at the end of step 73 once UserProfileBean object has done the operation.

Copyright by Mahidol University

58) AVClientController class calls UserProfileImpl class to create a new User Profile file in the specified directory.

59) AVClientController class is prompted once UserProfileImpl class has got the creation of User Profile file done.

60) AVClientController class asks SimpleXPath class with passing the DOM structure of Active Web Page to get all the items under the User Profile sub-section.

61) SimpleXPath class returns all the items which would be found in the section of User Profile.

62) AVClientController class checks whether at least one item is found. If this is the case, it will continue on the next step, or else it will jump into the end of step 73.

63) AVClientController asks UserProfileImpl class to look for a User Profile file in the specified directory.

64) UserProfileImpl class returns an existence status of User Profile file.

65) AVClientController class does check if the User Profile file is present through this returned status. If the file has already existed, it will continue on the next step, otherwise, it will skip down the end of step 73.

66) AVClientController class requests UserProfileImpl class to read the existing User Profile file into memory.

67) UserProfileImpl class calls SimpleXMLParser class to parse the in-memory User Profile.

68) UserProfileImpl class gets the User Profile in DOM form from SimpleXMLParser class and puts in User Profile bean structure.

69) UserProfileImpl class returns the UserProfileBean object to AVClientController class.

70) AVClientController class requests the UserProfileBean object to update the inside items with the arrival items.

71) The process is resumed on the end of step 73 once UserProfileBean object has got the operation done.

72) In the case of that the User Profile file has not already existed, AVClientController class calls UserProfileImpl class to create a new User Profile file in the specified directory.

Copyright by Mahidol University

73) AVClientController class is prompted once UserProfileImpl class has got the creation of User Profile file done.

Right now, we could say that two sources of data – inputs for the Active Rule execution – are ready to pick up. From this onward, AVClientController class has to collaborate with the Lite Rule Processor, featuring ActiveRuleImpl class and ActiveRuleBean class. AVClientController class will pass two data sources to these classes to proceed out an Active View. We will leave further steps to be explained in the next module.

4.2.1.2.2 Upcoming User Request

As shown in Figure 4-12, the UAE begins loading the attributes and elements of Active Web Page as stated by a URI string. The UAE validates if the *activity* parameter, containing a User Activity document, exists in the URI string and also if the request method is POST. If the validation is satisfied, the UAE needs to scan through the value of activity parameter and remove some special characters found. Then, the UAE has the activity parameter value parsed and puts the parsed document in a parameter bean structure. The UAE verifies if items, listed on the document, are to manipulate (add, update, or delete which are enforced by the elements <add>, <update>, and <delete> respectively) the Global Data or the User Profile. If this is the case, the UAE will assign a parameter binding handler to do so. Next, the UAE determines to which these items are affected. First, for a Global Data, the UAE binds items in the Global Data with them. Second, UAE do bind items in the User Profile. Example 4-7 gives an example of an activity parameter. The item "*shippingaddress*" must *update* to the corresponding one in the *User Profile* file as specified by elements <*UserProfile*> and <*update*> respectively.

In addition, there is an additional tag, namely <params>, used for declaring all submitting items which are passed to the AVML Processor in invoking a service call.

Copyright by Mahidol University

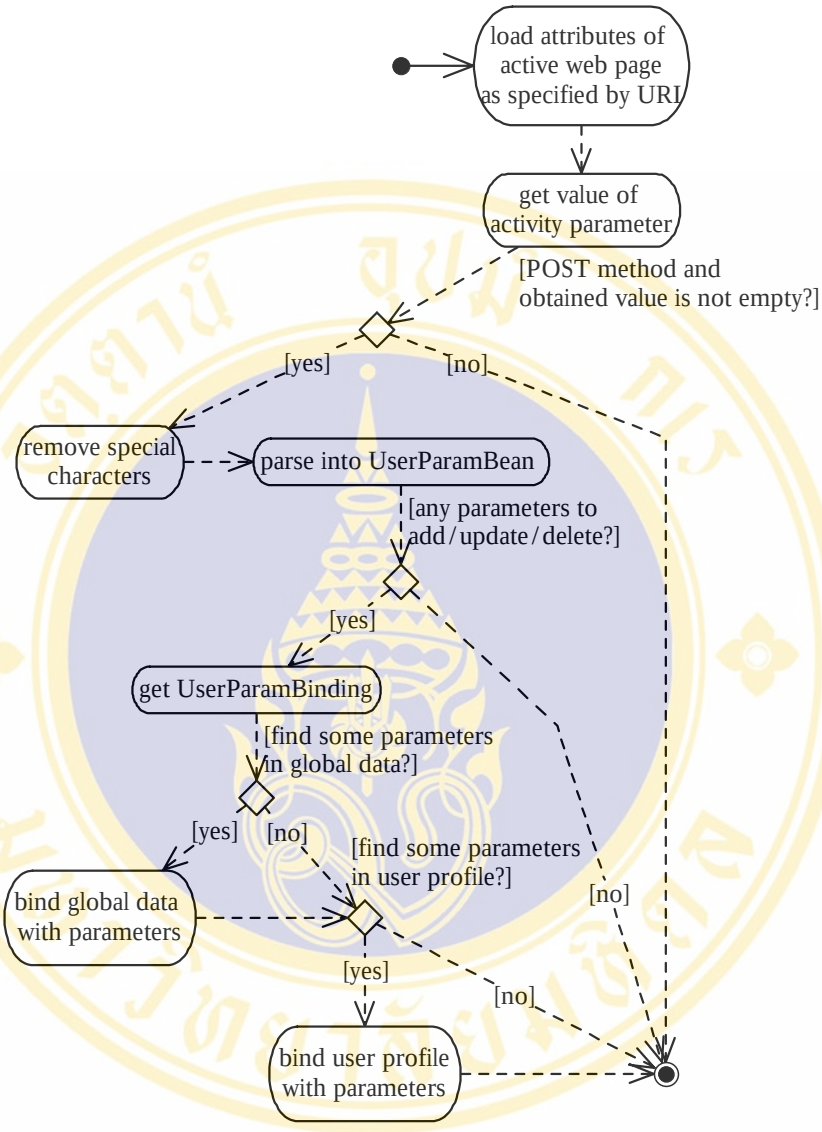


Figure 4-12 Activity Diagram of User Activity Evaluator Module Regarding an Upcoming User Request

Besides, let us see the sequence diagram in Figure 4-13 shows how to sort out the coming user request received from a user's web browser. The steps are described here:

1) ProxyConnection class creates an instance of AVClientController class with a URIParser object, a request method string and an output stream where the constructed Active View is written as three incoming parameters.

Copyright by Mahidol University

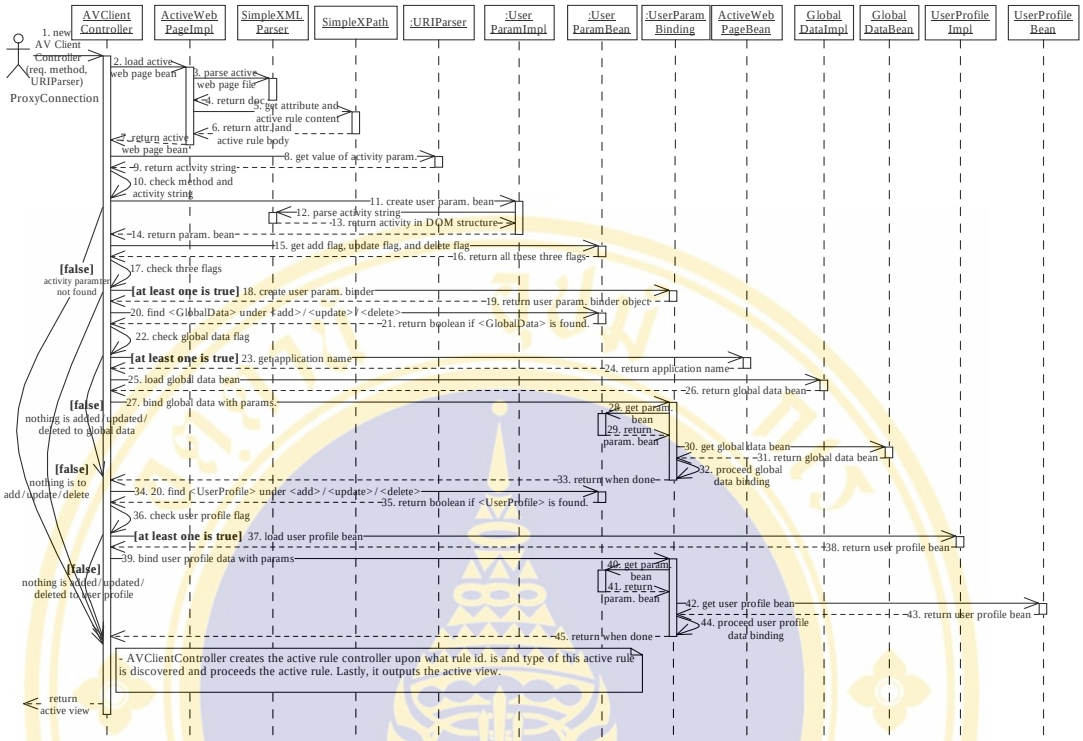


Figure 4-13 Sequence Diagram of User Activity Evaluator Module Regarding an Upcoming User Request

2) AVClientController class calls ActiveWebPageImpl class by passing the URIParser instance for dealing with a creation of Active Web Page object by reading of the existing detached Active Rule file.

3) ActiveWebPageImpl class requests SimpleXMLParser class to parse the detached Active Rule so that its information is available for further uses.

4) ActiveWebPageImpl class gets a response of DOM content that represents the Active Web Page from SimpleXMLParser class.

5) ActiveWebPageImpl class requests SimpleXPath class to locate pieces of DOM content – for instance, an e-service application name, and an Active Web Page name.

6) ActiveWebPageImpl class is prompted by the desired information of the Active Web Page from SimpleXPath class.

7) AVClientController class gets an object of ActiveWebPageBean class loaded by ActiveWebPageImpl class.

Copyright by Mahidol University

8) AVClientController class requests URIParser object to get a value of activity parameter.

9) AVClientController class gets back the activity parameter string expressed in XML-formatted from the URIParser object.

10) AVClientController class verifies if the request method is POST and the value of activity parameter is nothing. If this is a case, it will continue on the next step, or else go to the end of step 45.

11) AVClientController class calls UserParamImpl class to create a new UserParamBean object with the *activity* string.

12) UserParamImpl class requests SimpleXMLParser class to parse the *activity* string.

13) UserParamImpl class gets the activity in DOM form from SimpleXMLParser class and puts in a UserParamBean structure.

14) UserParamImpl class returns the newly built UserParamBean object to AVClientController class.

15) AVClientController class gets the add/update/delete flags marking what to be done toward the Global Data and the User Profile from the UserParamBean instance.

16) AVClientController class asks the UserParamBean object to obtain the add flag, update flag and delete flag so that it resolves if any submitting parameters are going to be added, updated, or deleted against the Global Data file or the User Profile.

17) AVClientController class examines the flags if any items are to affect the Global Data and the User Profile. If one of them is true, it will continue on the next step, otherwise, go to the end of step 45.

18) AVClientController class creates a UserParamBinding object to handler the data binding process.

19) The new UserParamBinding object is returned to AVClientController class.

20) AVClientController class asks the UserParamBean object to see if the element <GlobalData> appears under the elements <add>, <update>, or <delete>.

21).The UserParamBean object returns the boolean value to AVClientController class.

Copyright by Mahidol University

22) AVClientController class validates the Boolean value if any items in the activity parameter need to add, update, or delete toward the Global Data. If so, it will continue on the next step, or else leap to the step 34.

23) AVClientController class calls the method getECAppName() of ActiveWebPageBean object to get a name of e-service application where the Active Web Page came out.

24) The method returns the e-service application name string to AVClientController class.

25) AVClientController class requests GlobalDataImpl class to read Global Data items from the Global Data file into GlobalDataBean object.

26) AVClientController class gets back the instance of GlobalDataBean class.

27) AVClientController class asks the UserParamBinding object by passing the GlobalDataBean and UserParamBean objects to bind the existing Global Data items upon the add, update, or delete operations with the ones accompanied by the activity parameter.

28) The UserParamBinding object fetches items under the element <GlobalData> from the UserParamBean object.

29) The UserParamBean object returns the items to the UserParamBinding object.

30) The UserParamBinding object fetches items from the GlobalDataBean object.

31) The UserParamBean object returns the existing Global Data items to the UserParamBinding object.

32) The UserParamBinding object starts binding the Global Data items from two sources according to the add, update, or delete operations.

33) AVClientController class is prompted once the Global Data items binding operation was done.

34) AVClientController class asks the UserParamBean object to see if the element <UserProfile> appears under the elements <add>, <update>, or <delete>.

35).The UserParamBean object returns the Boolean value to AVClientController class.

Copyright by Mahidol University

36) AVClientController class validates the Boolean value if any items in the activity parameter need to add, update, or delete toward the User Profile data. If so, it will continue on the next step, or else leap to the step 34.

37) AVClientController class requests UserProfileImpl class to read User Profile items from the User Profile file and put into UserProfileBean object.

38) AVClientController class gets back the instance of UserProfileBean class.

39) AVClientController class asks the UserParamBinding object by passing the UserProfileBean and UserParamBean objects to bind the existing User Profile items upon the add, update, or delete operations with the ones accompanied by the activity parameter.

40) The UserParamBinding object fetches items under the element <UserProfile> from the UserParamBean object.

41) The UserParamBean object returns the items to the UserParamBinding object.

42) The UserParamBinding object fetches items from the UserProfileBean object.

43) The UserParamBean object returns the existing User Profile items to the UserParamBinding object.

44) The UserParamBinding object starts binding the User Profile items from two sources according to the adding, updating, or deleting operations.

45) AVClientController class is prompted once the User Profile items binding operation was done.

At this time, we keep the Global Data and the User Profile up-to-date with all data items in the User Activity document accompanied by the user request. Such these items engage in an Active Rule execution to construct an Active View. AVClientController class, from now on, has to get loaded of Active Rule identified by the Rule ID. and the Rule Type parameters from the Active Rules temporary storage. It certainly requires two data sources to construct an Active View. We soon step into that point.

Copyright by Mahidol University

4.2.1.3 Lite Rule Processor

Here we come to the last component of the AV Client that is such an *Active Rule engine*, namely Lite Rule Processor (or **ARE** in short). We state the ARE module working on the AV Client first and will then discuss about the other one on the AVML Processor later. *The ARE of the AV Client is responsible for an Active View Construction upon an arrival Active Web Page.* To accomplish this, the ARE has to execute an Active Rule which is recognized upon a user action.

Like the earlier module, the two sources from where the inputs came are a user's web browser, where a user request is generated and the AVML Processor, where a new Active Web Page was constructed. This differentiates what type of Active Rule will be fired, as briefed here.

- **An Approaching Active Web Page** always causes an onLoad Active Rule to be executed.
- **A User Request** possibly causes either a Local Active Rule or a Remote Active Rule to be executed upon which one is triggered by an action performed by a user.

Basically, the ARE is composed of the three sub-modules which are the Active Rule Loader, the Parameter Binder, and the Active View Generator. We are approaching to each in details *according to these two input patterns* later.

4.2.1.3.1 Active Rule Loader

The Active Rule Loader sub-module (RL in short) takes into account reading the triggered Active Rule from the Active Rules temporary storage into memory and defining the Active Rule controller toward the type of Active Rule fired in order to get executed. We list the classes nod for the Active Rule Loader sub-module in accepting a new Active Web Page from the AVML Processor and a user request from a user's web browser as shown in the class diagram Figure 4-14.

Copyright by Mahidol University

AVClientController class advises ActiveRuleImpl class with passing of ActiveWebPageBean class as mandatory parameters to load the body of Active Rule to ActiveRuleBean class. For a single parameter passed – only ActiveWebPageBean class (*this is for a new Active Web Page arriving*), ActiveRuleImpl class by default reads an onLoad Active Rule embedded in the ongoing Active Web Page mentioned in ActiveWebPageBean class to memory. Otherwise, there are two more optional parameters passed, that are the Rule ID. and the Rule Type (*this is for a user request detected*), ActiveRuleImpl class reads the Active Rule, identified by the Rule ID. and the Rule Type, of the Active Web Page mentioned in ActiveWebPageBean class to memory.

Once the desired Active Rule was loaded, ActiveRuleImpl class places it in ActiveRuleBean class. At last, ActiveRuleImpl class considers what type of Active Rule controller will be assigned for accomplishing an execution task.

- For an approaching Active Web Page, LocalRuleController class is responsible for handling the onLoad Active Rule execution;
- For a user request, it is dependent on what the Rule Type is. If a Local Active Rule is detected, LocalRuleController class will be appointed, or else, a Remote Active Rule is detected, RemoteRuleController class will be appointed.

Copyright by Mahidol University

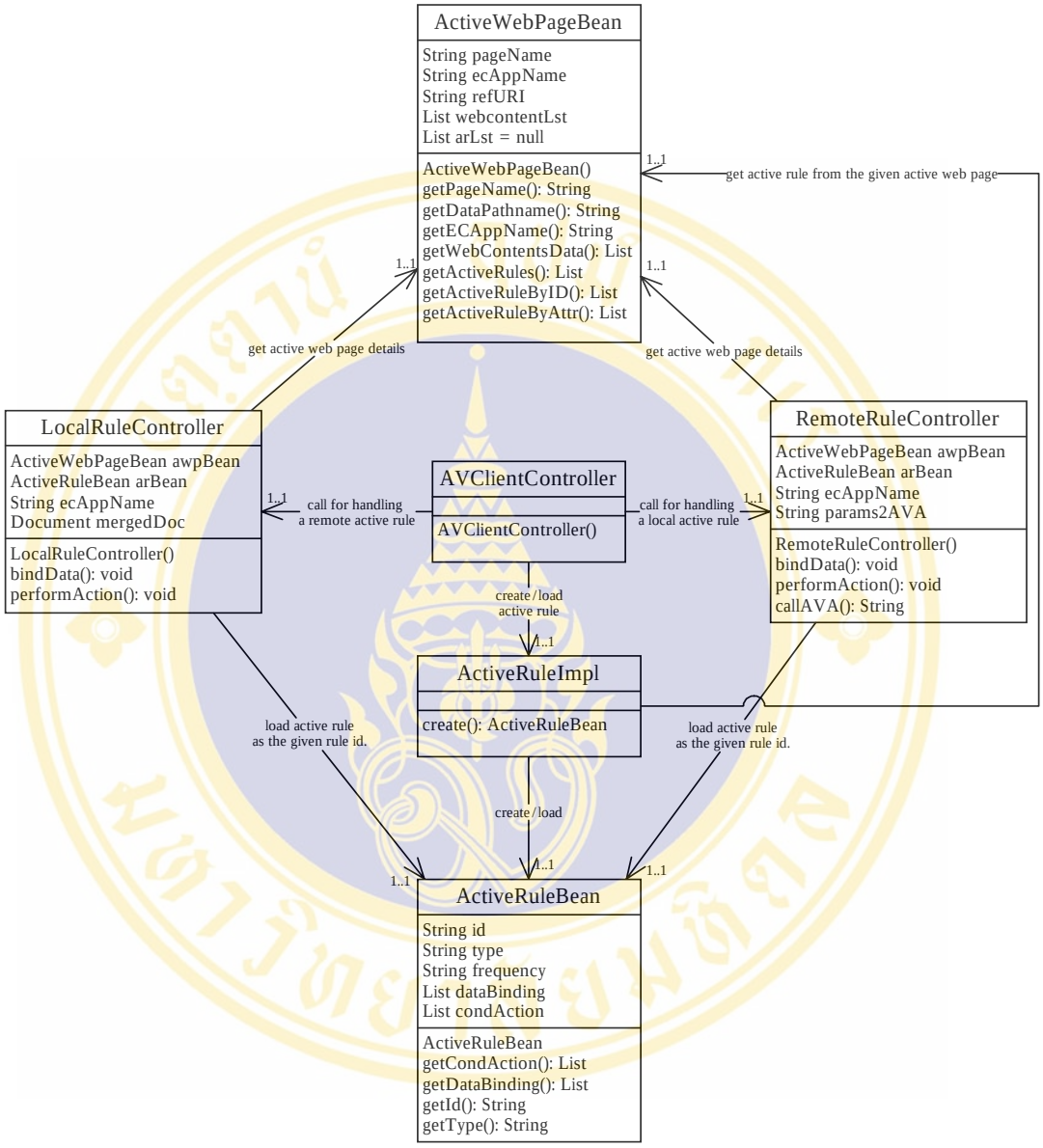


Figure 4-14 Class Diagram of Rule Loader Sub-module

Copyright by Mahidol University

We are coming to see how the ARE is going to work as follows.

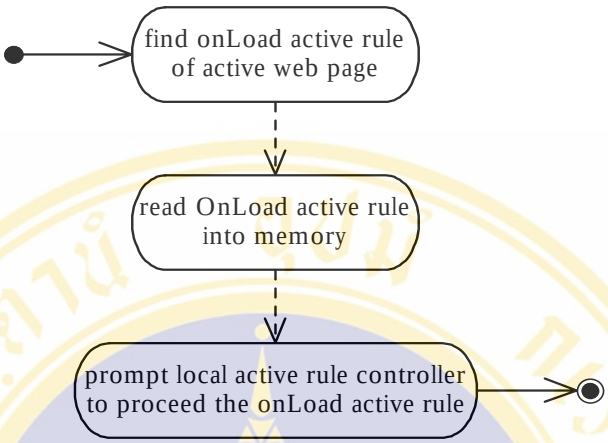


Figure 4-15 Activity Diagram of Rule Loader Sub-module Upon a Coming Active Web Page Received

For a coming Active Web Page received from the AVML Processor, we describe logical tasks done in this sub-module in the activity diagram in Figure 4-15. The RL reads an onLoad Active Rule among the others in the same Active Web Page into memory. Then, the RL asks a Local Active Rule controller to take up the in-memory onLoad Active Rule and proceed it.

Also, we supply the sequence diagram to this event in Figure 4-16 and explore each step as follows.

1) AVClientController class calls ActiveRuleImpl class to discover an onLoad Active Rule of the current Active Web Page.

2) ActiveRuleImpl class requests the existing ActiveWebPageBean instance that has been already created by the earlier module to address an onLoad Active Rule and pull out its body.

3) ActiveRuleImpl class is prompted by the body of onLoad Active Rule and creates a new object of ActiveRuleBean class with the onLoad Active Rule accepted.

4) The new instance of ActiveRuleBean class is returned to AVClientController class.

5) AVClientController class creates a new instance of LocalRuleController class and prompts it to conduct an execution of onLoad Active Rule.

Copyright by Mahidol University

6) The LocalRuleController object gains the name of e-service application from the ActiveWebPageBean instance.

7) The ActiveWebPageBean responds the application name to the issuer.

8) AVClientController class is resumed when the previous step was done.

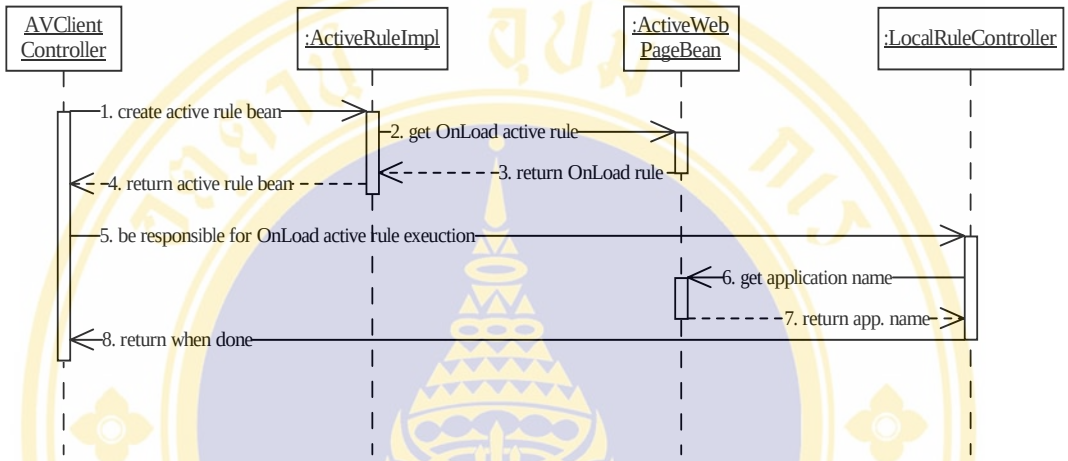


Figure 4-16 Sequence Diagram of Rule Loader Sub-module Upon a Coming Active Web Page Received

For a coming user request performed by a user, we describe the RL process, with respect to this situation, in the activity diagram in Figure 4-17. The RL starts identifying an Active Rule, amongst the others ranging in the same Active Web Page, in the Active Rules temporary storage by using a *Rule ID.* and a *Rule Type*. Then, the RL reads the body of Active Rule found into memory. Unlike the earlier input, the RE must resolve what its type is. The Local Active Rule controller will be assigned an execution task of Local Active Rule if the rule type appears “local”. Otherwise, the rule type appears “remote”, the Remote Active Rule controller will be appointed. From now onward, an Active Rule controller will go on an execution task.

Copyright by Mahidol University

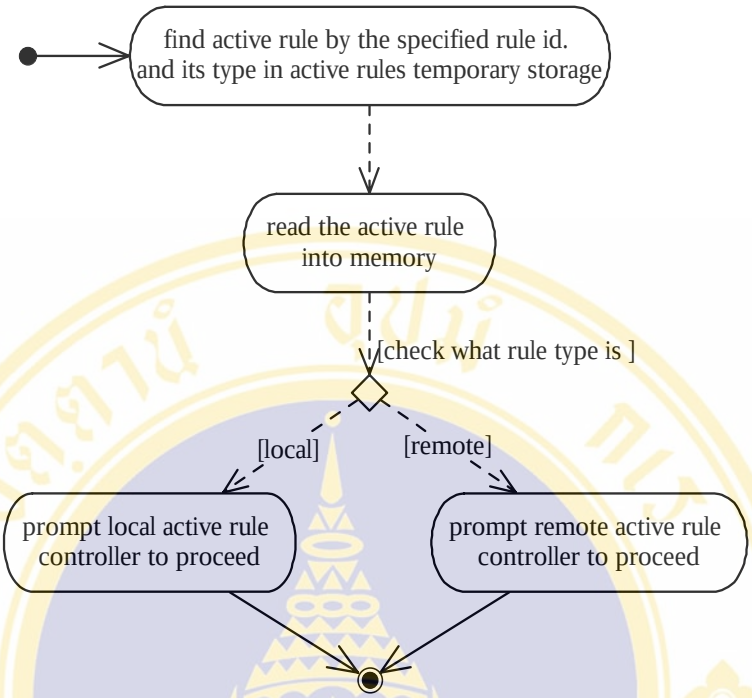


Figure 4-17 Activity Diagram of Rule Loader Sub-module
Upon a Coming User Request Detected

This context is exemplified by the sequence diagram in Figure 4-18 and each step will be disclosed as follows.

1) AVClientController class calls ActiveRuleImpl class to search the Active Rules temporary storage for the fired Active Rule as keyed by the Rule ID. and the Rule Type.

2) ActiveRuleImpl class requests the existing ActiveWebPageBean instance to address this Active Rule and retrieve its body.

3) ActiveRuleImpl class is prompted by the body and creates a new object of ActiveRuleBean class based on the content of destined Active Rule.

4) The new instance of ActiveRuleBean class is returned to AVClientController class.

5) AVClientController class asks the ActiveRuleBean object for the type of Active Rule.

6) AVClientController class is reacted by the rule type string.

Copyright by Mahidol University

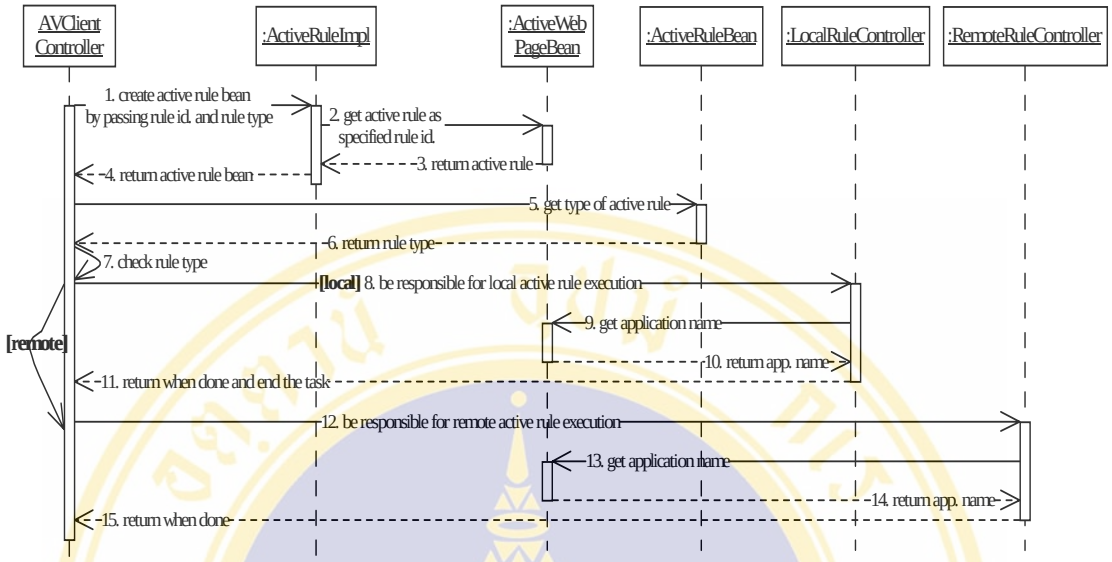


Figure 4-18 Sequence Diagram of Rule Loader Sub-module Upon a Coming User Request Detected

7) AVClientController class goes check the type. If the type turns out *local*, it will continue on the next step, or else, the type is *remote*, redirect to the step 12.

8) AVClientController class creates a new instance of LocalRuleController class to be responsible for a Local Active Rule execution happening soon.

9) The LocalRuleController object gains the name of e-service application from the ActiveWebPageBean instance.

10) The ActiveWebPageBean responds the application name to the issuer.

11) AVClientController class is prompted when the operation was done by the LocalRuleController object and ends the tasks of the sub-module.

12) AVClientController class creates a new instance of RemoteRuleController class to be responsible for a Remote Active Rule execution happening soon.

13) The RemoteRuleController object gains the name of e-service application from the ActiveWebPageBean instance.

14) The ActiveWebPageBean responds the application name to the issuer.

15) AVClientController class is prompted when the operation was done by the RemoteRuleController object and ends the operation of this sub-module.

Copyright by Mahidol University

4.2.1.3.2 Parameter Binder

The Parameter Binder sub-module (briefly PB) caters for obtaining value for items required in proceeding of the Active Rule. The items are involved in a condition part or an action part of Active Rule and distinguish Active View content and display for each user. To acquire their values, the PB has to bind the items cited for the Active Rule body with the corresponding ones possibly sourcing from *a Global Data, a User Profile, and a User Activity document*. Recall what has been done by the User Activity Evaluator module, we get the three data sources ready for being attained here.

To pursue the PB's goal, we come out the PB-classes-bait as structured in the class diagram in Figure 4-19. The PB takes slightly different actions upon the two kinds of Active Rule as described here.

To bind items for a Local Active Rule, DataBinding class is designated by LocalRuleController class to be in charge of items preparation. DataBinding class first gets a list of items used up in an execution phase from ActiveRuleBean class and creates an intermediate buck called MergedDataBean class to gather all items bound. The word "merge" implies a union of Global Data items and User Profile items. MergedDataBean class would become one of main inputs into the processing of Active View Generator. A list of items is formally declared under the element <DataBinding> of each Active Rule body. Second, DataBinding class checks if any of these items belong to the Global Data. If this is the case, DataBinding class will have to take GlobalDataBean class and obtain the Global Data DOM content. During a binding process of Global Data items, DataBinding class needs help from SimpleXPath class to get access to the target item in the Global Data DOM as keyed by the item id. and item name and then get its value appended to MergedDataBean class. Third, DataBinding class insures if any of these items are sort of the User Profile. If so, DataBinding class will associate with UserProfileBean class and obtain the User Profile DOM content. DataBinding class co-works with SimpleXPath class to address the target item in the User Profile DOM by use of the item id. and the item name as dual-identified keys and get its value appended to MergedDataBean class. In

Copyright by Mahidol University

the end, we get the output, the MergedDataBean which now contains the pair of item name and whose value varying in the two groups (Global Data and User Profile).

To bind items for a Remote Active Rule, Param2AVABinding class is designated by RemoteRuleController class to be in charge. Param2AVABinding class first gets a list of items used up of an execution phase from ActiveRuleBean class and creates an intermediate buck called Param2AVABean class to gather all items bound. Second, Param2AVABinding class checks if any of these items belong to the Global Data. If this is the case, Param2AVABinding class will have to obtain the Global Data DOM content from GlobalDataBean class. During the binding process of Global Data items, Param2AVABinding class needs help from SimpleXPath class to get access to the target item in the Global Data DOM and then get its value appended to Param2AVABean class. Third, Param2AVABinding class insures if any of these items are sort of User Profile group. If so, Param2AVABinding class will obtain the User Profile DOM content from UserProfileBean class. Param2AVABinding class co-works with SimpleXPath class to address the target item in the User Profile DOM and get its value appended to Param2AVABean class. Fourth, this step is incrementally raised so that the PB allows the parameters in the User Activity document submitted by a user to be passed to the AVML Processor along with the service call invocation. Param2AVABinding class validates if UserParamBean class has been set. If so, Param2AVABinding class will obtain the *params* element from UserParamBean class and append to Param2AVABean class. Refer to Figure 4-2, the element <params> is specified by the element <activity>. Finally, the PR finalizes the output in the form of Param2AVABean class which now contains the pair of item name and its value varying in the three groups (Global Data, User Profile, and parameters accompanied by the User Activity document).

Copyright by Mahidol University

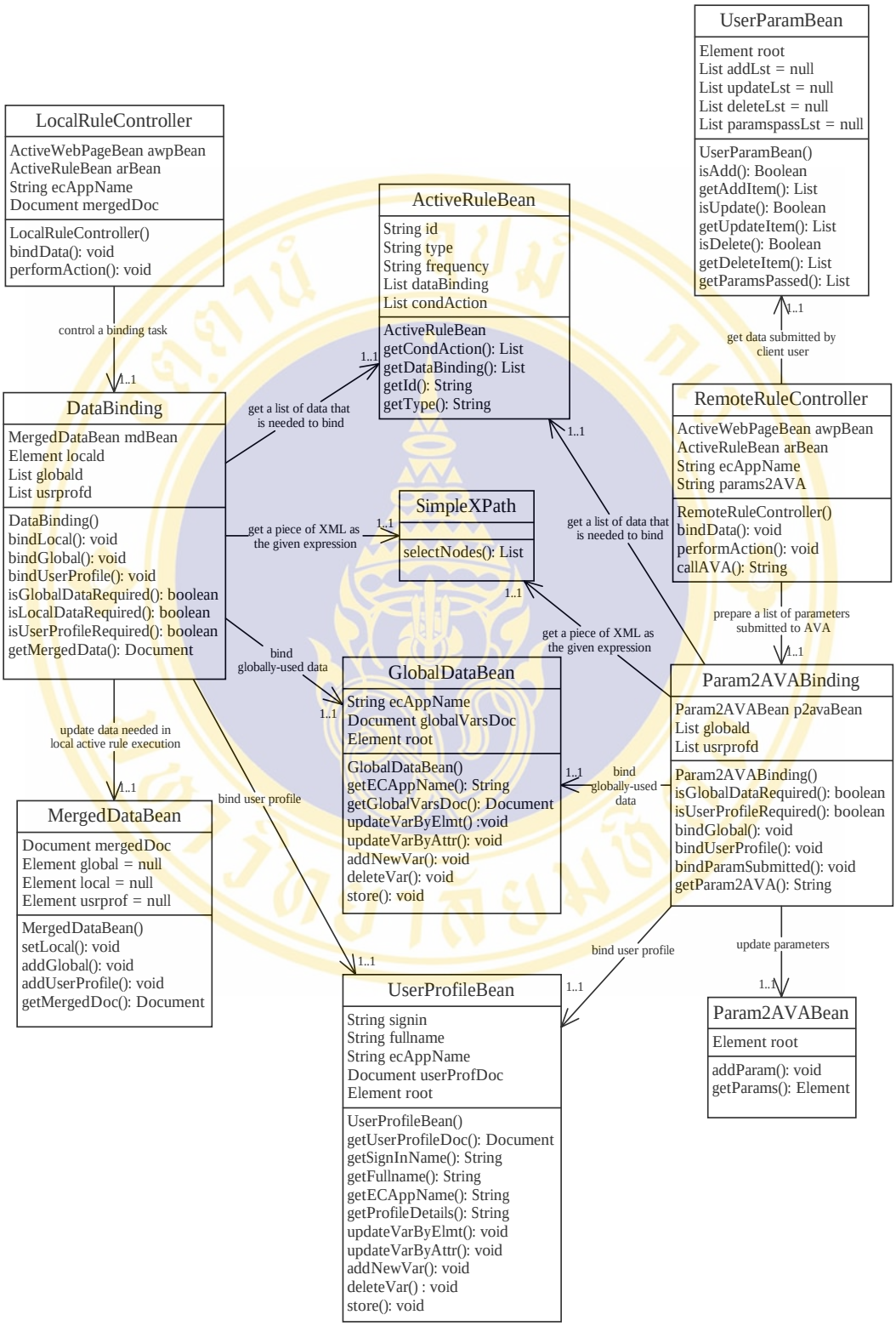


Figure 4-19 Class Diagram of Parameter Binder Sub-module

Copyright by Mahidol University

We separately explain the rest of this sub-module as a result of the differently binding tasks of two types of Active Rule.

4.2.1.3.2.1 Binding Process of Local Active Rule

We present generic processes in binding items for a Local Active Rule with the activity diagram as shown in Figure 4-20.

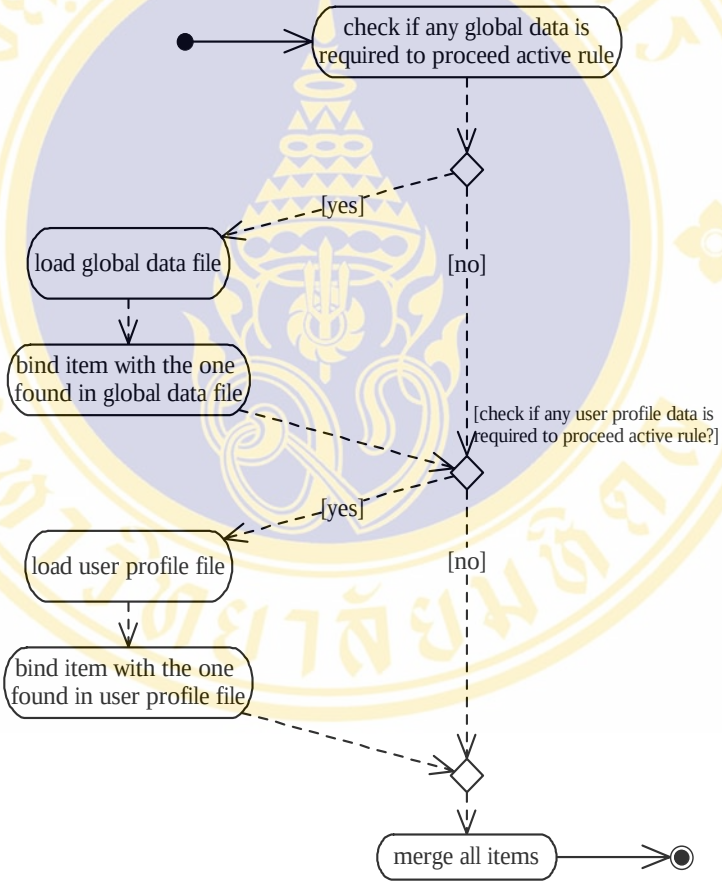


Figure 4-20 Activity Diagram of Parameter Binder Sub-module as to a Local Active Rule

First of all, the PB focuses on a data binding part of the ongoing Active Rule – for instance, this part is illustrated by the italic face in Example 4-8 – and affirms if the element <global> is found under the element <DataBinding>. If this is the case, the PB will refer to the Global Data file that should have been already read by the User

Copyright by Mahidol University

Activity Evaluator module and then start gaining the value of each item listed on the element <global> from the Global Data. The PB repeats the step of obtaining the value until the last item under the element <global> was reached.

Have a closer look at the attribute of <DataBinding> element. A noticeable effort is seen to apply XLink specification [20, 25] to specify *where* and *what* to obtain the value of each item. We, in particular adapt, a representation of a *simple XLink* to define where the source of item is and XPath expression to define what the item is. For example, at the line #4, this means that the PB has to select all items from the globaldata.xml file where the element name is *product* as matching with the term "*globaldata.xml#//product*". In fact, the value of attribute "xlink:href" should be defined in term of URI notion. On account of simplicity and file location locally stored in a user's machine, it would be better to override according to this concern. The selecting result is appended to the Merged Items document expressed in XML-formatted. The Merged Items document is structured in XML Schema representation in Figure 4-3. Example 4-9 and Example 4-10 exemplify the Merged Items document before (an empty merged data) and *after* the binding process for a Global Data item respectively.

Second, the PB checks if the element <UserProfile> is found below the element <DataBinding>. If this is the case, the PB will refer to the User Profile file that should have been read earlier and then start gaining the value of each item said in the element <UserProfile> from the User Profile.

Example 4-8 An Example of a Data Binding Part of a Local Active Rule

```
1  <ar id="6" name="order-summary" type="local" frequency="lazy">
2   <DataBinding xmlns:xlink="http://www.w3.org/1999/xlink">
3    <global>
4     <product xlink:type="simple" xlink:href="globaldata.xml#//product"
   xlink:actuate="onRequest"/>
5    </global>
6    <UserProfile>
7     <userid xlink:type="simple" xlink:href="userprof.xml#//userinfo"
   xlink:actuate="onRequest"/>
8     <shippingadress xlink:type="simple"
   xlink:href="userprof.xml#//shippingaddress[@selected='true']"
   xlink:actuate="onRequest"/>
```

Copyright by Mahidol University

```
9     <billingadress xlink:type="simple"
      xlink:href="userprof.xml#//billingaddress[@selected='true']"
      xlink:actuate="onRequest"/>
10    <paymentmethod xlink:type="simple"
      xlink:href="userprof.xml#//paymentmethod[@selected='true']"
      xlink:actuate="onRequest"/>
11    </UserProfile>
12   </DataBinding>
13   <CondAction>
14    ...
15   </CondAction>
16  </ar>
```

Example 4-9 Merged Items Document Before a Binding Process for a Global Data

```
<?xml version="1.0" encoding="UTF-8"?>
<merged>
 <global />
 <UserProfile />
</merged>
```

Example 4-10 Merged Items Document After a Binding Process for Global Data Items

```
<?xml version="1.0" encoding="UTF-8"?>
<merged>
 <global>
  <product id="cd1" title="The Emancipation of Mimi (Limited Edition)" author="Mariah
Carey" category="Audio CD" latestAdded="2005-04-26" price="19.99" currency="$"
discount="0.25" quantity="1" unticked="false"/>
  <product id="cd2" title="The Hours (Widescreen Edition)" author="Stephen Daldry"
category="DVD" latestAdded="2005-01-03" price="9.99" currency="$" discount="0"
quantity="1" unticked="false"/>
  <product id="cd3" title="Erin Brockovich" author="Steven Soderbergh" category="DVD"
latestAdded="2004-12-22" price="14.98" currency="$" discount="0.20" quantity="1"
unticked="false"/>
 </global>
 <UserProfile />
</merged>
```

In addition, the sequence diagram of the PB sub-module to prepare items for executing of a Local Active Rule is shown in Figure 4-21.

Copyright by Mahidol University

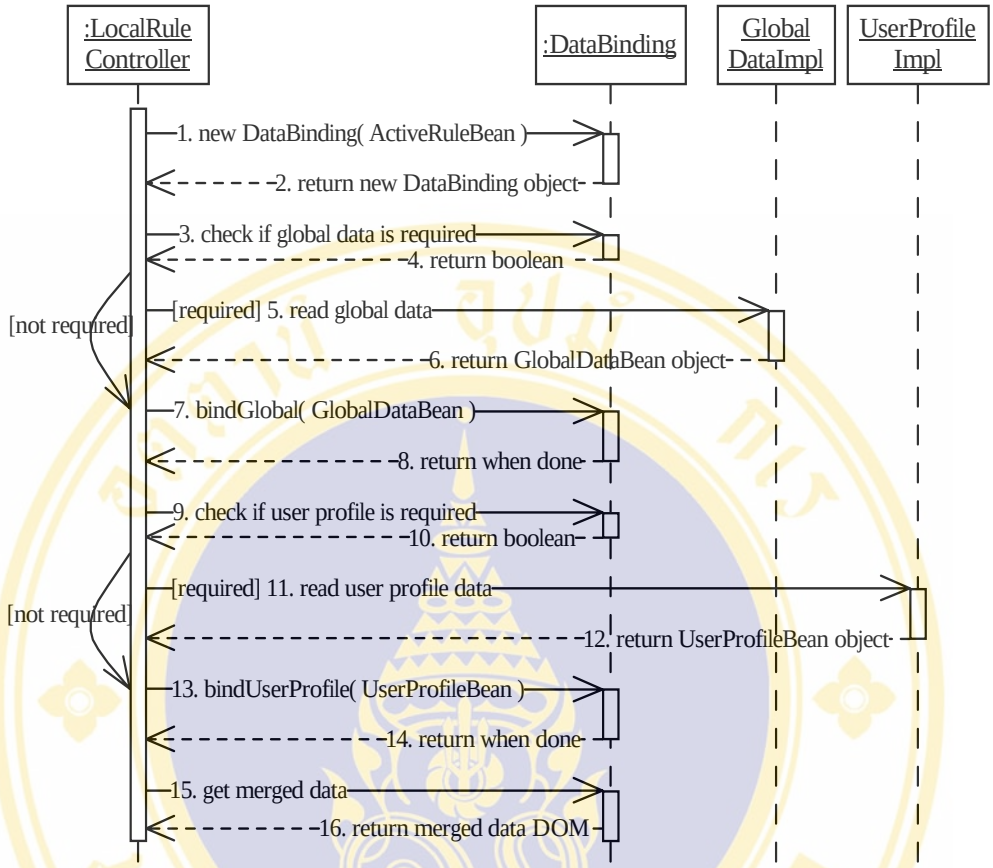


Figure 4-21 Sequence Diagram of Parameter Binder Sub-module in Binding Parameters of a Local Active Rule

1) The LocalRuleController object recently created on the last sub-module in the fact that a Local Active Rule was triggered calls the constructor method of DataBinding class to create the new instance with the passing ActiveRuleBean object.

2) The newly created DataBinding object is responded to the LocalRuleController object.

3) The LocalRuleController asks the DataBinding object to check if the sub-element <global> of the <DataBinding> element exists.

4) If it was present, the Boolean value would return true and continue on the next step, or else, the Boolean value would return false and direct to the step 7.

5) The LocalRuleController object asks GlobalDataImpl class to read the Global Data file into the GlobalDataBean class.

6) GlobalDataImpl class returns the GlobalDataBean object to the LocalRuleController object.

Copyright by Mahidol University

7-8) The LocalRuleController object asks the DataBinding object to bind items by obtaining the value from the GlobalDataBean object and placing in the MergedDataBean object. The next operation on the LocalRuleController object is resumed when the task here was done.

9) *Similar to the Global Data item binding*, the LocalRuleController object asks the DataBinding object to check if the sub-element <UserProfile> of the <DataBinding> element exists.

10) If it was present, the Boolean value would return true and continue on the next step, or else, the Boolean value would return false and direct to the step 13.

11) The LocalRuleController object asks UserProfileImpl class to read the User Profile file into the UserProfileBean class.

12) UserProfileImpl class returns the UserProfileBean object to the LocalRuleController object.

13-14) The LocalRuleController object asks the DataBinding object to bind items by obtaining the value from the UserProfileBean object and placing in the MergedDataBean object. The work will be resumed by the LocalRuleController object after the operation was done.

15) The LocalRuleController object requests the DataBinding object for merging *Global Data items and/or User Profile items*.

16) The merged data DOM is responded to the LocalRuleController object.

4.2.1.3.2.2 Binding Process of Remote Active Rule

For more information, the element <ar>, including its sub-element, *of a Remote Active Rule* is called *a data binding part* where its specification has explained in Section 3.4 of Chapter III. We present generic processes in binding items for a Remote Active Rule with the activity diagram as shown in Figure 4-22.

Copyright by Mahidol University

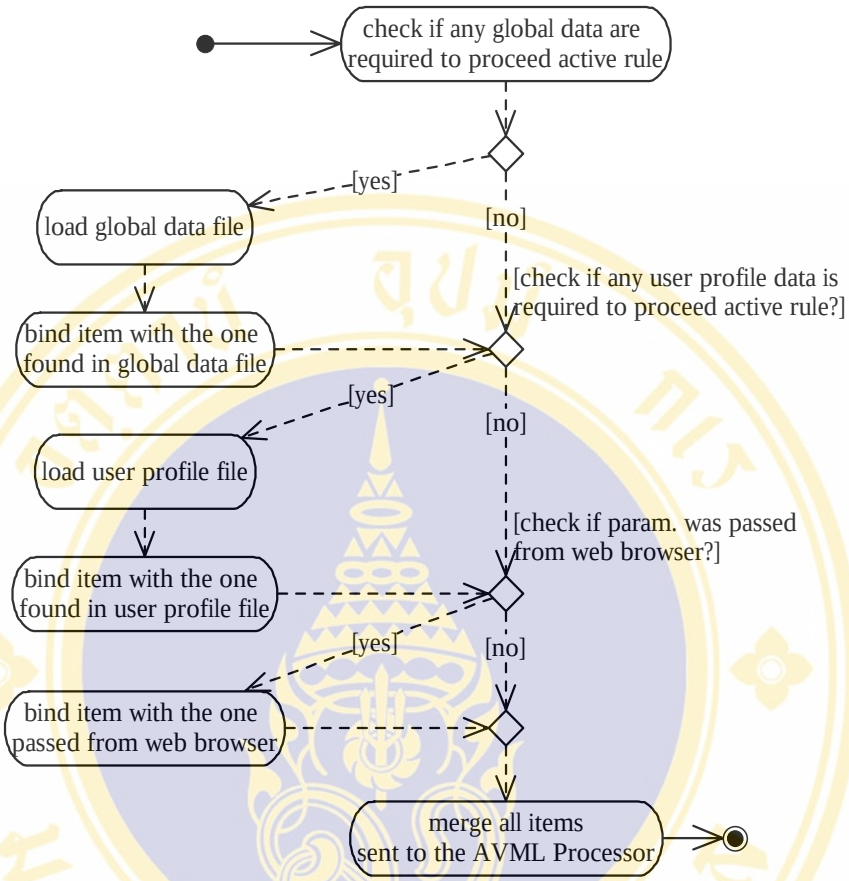


Figure 4-22 Activity Diagram of Parameter Binder Sub-module as to a Remote Active Rule

Similar to what is done for binding item of a Local Active Rule, the PB has to start binding the Global Data item and the User Profile item as indicated by the elements <global> and <UserProfile> regulated by the element <DataBinding>. The acquired item values are appended to a *Service Call Parameters document* which is represented in XML-formatted as structured by XML Schema in Figure 4-4. Moreover, there is an incremental step added. The PB checks if any parameter items were submitted by a user's web browser. If this is the case, all parameters found will be by default appended to the Service Call Parameters document as well. Ultimately, all items contained in the Service Call Parameters document like Example 4-11 will be submitted to the AVML Processor along with the service call invocation.

Copyright by Mahidol University

Example 4-11 An Example of a Service Call Parameters Document

```
<params>
 <username>riam@gmail.com</username>
 <password>password</password>
</params>
```

In addition, the sequence diagram of the PB, preparing parameter lists required for executing a Remote Active Rule, is shown in Figure 4-23.

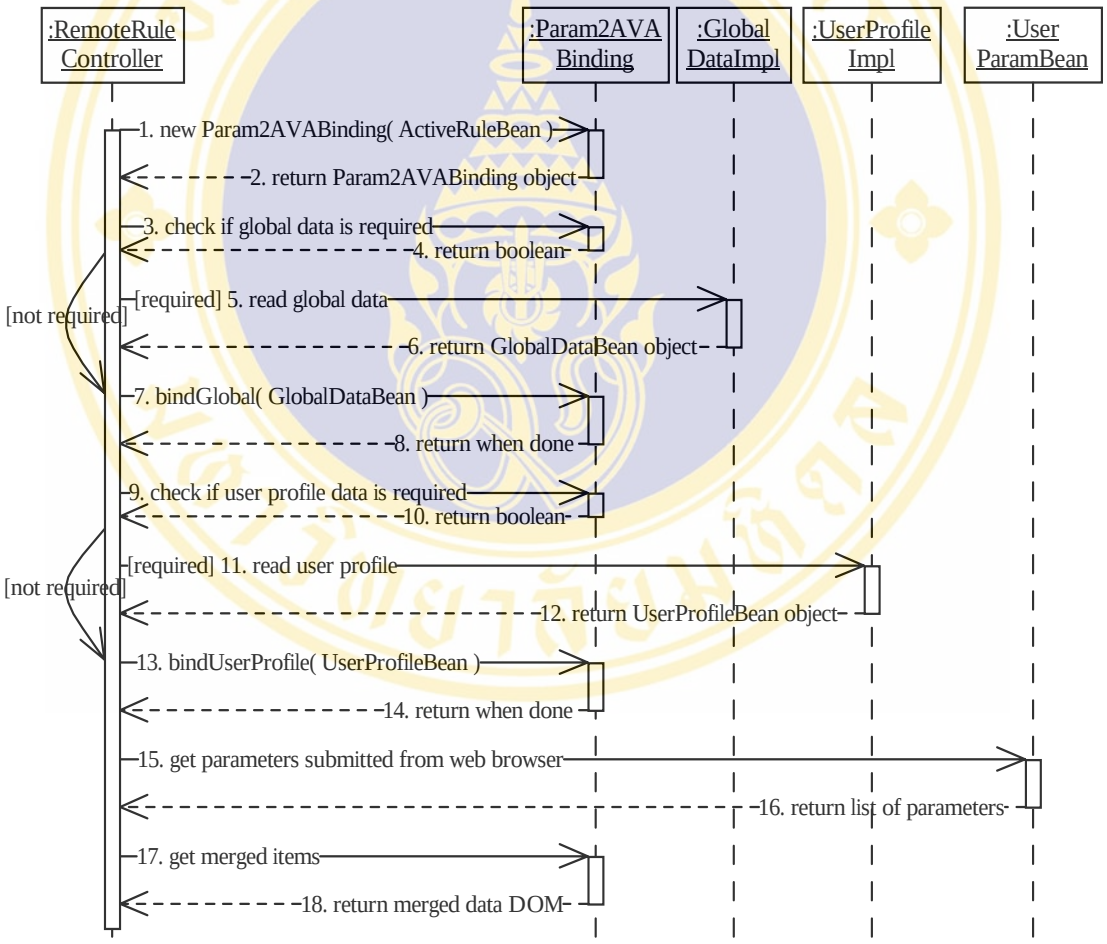


Figure 4-23 Sequence Diagram of Parameter Binder Sub-module in Binding Parameters of a Remote Active Rule

1) The RemoteRuleController object recently created on the last sub-module in the fact that the Remote Active Rule was fired calls the constructor method of

Copyright by Mahidol University

Param2AVABinding class to create the new instance with the passing ActiveRuleBean object.

2) The newly created Param2AVABinding object is responded to the RemoteRuleController object.

3) The RemoteRuleController asks the Param2AVABinding object to check if the sub-element <global> of the <DataBinding> element exists.

4) If it was present, the Boolean value would return true and continue on the next step, or else, the Boolean value would return false and direct to the step 7.

5) The RemoteRuleController object asks GlobalDataImpl class to read the Global Data file into the GlobalDataBean class.

6) GlobalDataImpl class returns the GlobalDataBean object to the RemoteRuleController object.

7-8) The RemoteRuleController object asks the Param2AVABinding object to bind items by obtaining the value from the GlobalDataBean object and placing in the Param2AVABean object. The next operation on the RemoteRuleController object is resumed when the task here was done.

9) *Similar to the Global Data item binding*, the RemoteRuleController object asks the Param2AVABinding object to check if the sub-element <UserProfile> of the <DataBinding> element exists.

10) If it was present, the Boolean value would return true and continue on the next step, or else, the Boolean would return false and direct to the step 13.

11) The RemoteRuleController object asks UserProfileImpl class to read the User Profile file into the UserProfileBean class.

12) UserProfileImpl class returns the UserProfileBean object to the RemoteRuleController object.

13-14) The RemoteRuleController object asks the Param2AVABinding object to bind items by obtaining the value from the UserProfileBean object and placing in the Param2AVABean object. The work will be resumed by the RemoteRuleController object after the operation was done.

15) The RemoteRuleController object acquires submitted parameters from the UserParamBean instance.

Copyright by Mahidol University

16) The RemoteRuleController object receives the User Activity DOM from the UserParamBean instance.

17) The RemoteRuleController object requests the Param2AVABinding object for merging *Global Data items, User Profile items, and/or User Activity document.*

18) The Merged Items DOM is responded to the RemoteRuleController object.

4.2.1.3.3 Active View Generator

We are going to introduce the last sub-module of the AV Client which plays a part of Active View launching. This sub-module takes the outputs coming out since the previous modules to be its inputs. We apply the XSLT transformation scheme [32] to complete the task. A look of Active View is portrayed by XSL stylesheet, which is in the form of the condition-action part of a Local Active Rule and XML-based document data that is a merging of Global Data items and User Profile items – namely a *Merged Items document*. With the same XSL, two users might get different look on account of different XML data and the constraints enforced in the XSL instruction as well.

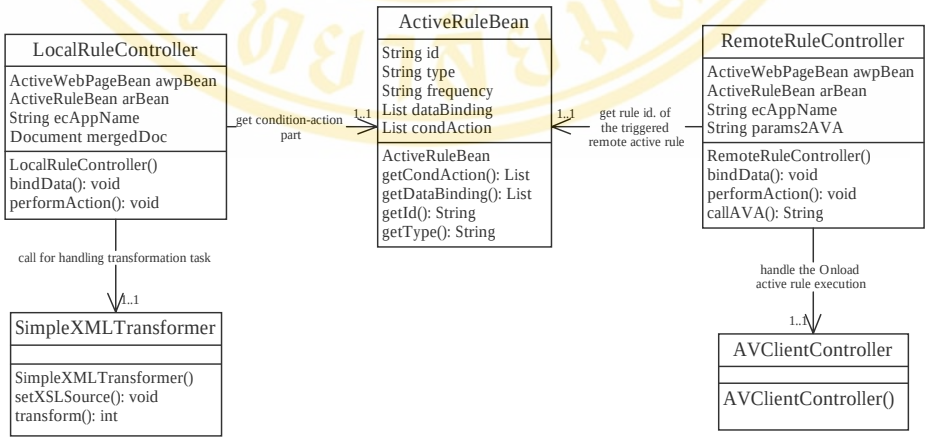


Figure 4-24 Class Diagram of Active View Generator Sub-module

The relevant classes of the Active View Generator are depicted in Figure 4-24. At this moment, it is known of which type of Active Rule is triggered. The Active

Copyright by Mahidol University

Rule engine sorts out a construction strategy of an Active View differently according to different types of Active Rules.

For a Local Active Rule, LocalRuleController class acquires a condition-action part of Local Active Rule under the element <CondAction> from ActiveRuleBean class. The specification of a condition-action part of Local Active Rule has mentioned in Section 3.4 in Chapter III and the instance of condition-action part is depicted in Example 4-12. The condition-action part happens to be an XSL which renders an Active View presentation whereas the Merged Items document happens to be such an XML-formateed document. LocalRuleController class associates with SimpleXMLTransformer class to develop a new Active View in regard to the XSLT transformation process in having these two data as inputs.

For a Remote Active Rule, RemoteRuleController class gets the attribute “Rule ID.” of Remote Active Rule from ActiveRuleBean class, the attribute “referrer page of Active Web Page”, and also the Service Call Parameters document. They are important arguments required to invoke the AVML Processor in order to get a new Active Web Page. RemoteRuleController class associates with AVClientController class to take care of an onLoad Active Rule execution when a new Active Web Page has arrived.

Example 4-12 An Example of A Condition-Action Part of a Local Active Rule

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
 <xsl:output method="html" media-type="text/html" indent="yes" />
 <!--main.html-->
 <xsl:template match="/">
   <html>
     <head>
       <title>Amazon.com Checkout: Payment</title>
       <script language="javascript1.2" type="text/javascript">var APOS = "'"; QUOTE =
'"'
var ESCAPED_QUOTE = {  }
ESCAPED_QUOTE[QUOTE] = '\"'
ESCAPED_QUOTE[APOS] = '\''

function formatAttributes( attributes )
{
var att_value;
```

Copyright by Mahidol University

```
var apos_pos, quot_pos;
var use_quote, escape, quote_to_escape;
var att_str;
var re;
var result = '';

for (var att in attributes) {
  att_value = attributes[att];

  apos_pos = att_value.indexOf(APOS);
  quot_pos = att_value.indexOf(QUOTE);

  if (apos_pos == -1 &amp;&amp; quot_pos == -1) {
      att_str = ' ' + att + "='" + att_value +  "'";
      result += att_str;
      continue;
  }

  if (quot_pos != -1 &amp;&amp; quot_pos &lt; apos_pos) {
      use_quote = APOS;
  }
  else {
      use_quote = QUOTE;
  }

  escape = ESCAPED_QUOTE[use_quote];

  re = new RegExp(use_quote,'g');
  att_str = ' ' + att + '=' + use_quote +
      att_value.replace(re, escape) + use_quote;
  result += att_str;
}
return result;
}

function element( name, content, attributes )
{
var att_str = '';
if ( attributes ) { // tests false if this arg is missing!
  att_str = formatAttributes( attributes );
}

var xml;
if ( !content ){
  xml = '&lt;' + name + att_str + '/&gt;';
}
else {
  xml = '&lt;' + name + att_str + '&gt;' + content + '&lt;/' + name + '&gt;';
```

Copyright by Mahidol University

```
}

return xml;
}

function firerule( form, image )
{
if ( image.name == 'submit' )
{

var atts = { id: ""+form.id.value,
          selected: "true" };

form.activity.value = element( 'activity', element( 'update', element( 'UserProfile',
element( 'billingaddress', '', atts ) ) ) );
form.submit();
}
}</script>
     </head>
     <body    bgcolor="#FFFFFF"    text="#000000"    link="#003399"    vlink="#003399"
alink="#FF9933">
       <table border="0" cellspacing="0" cellpadding="0" align="center">
         <tr>
           <td>
             <a name="top">
               <img       src="../images/checkout-spc-pay-banner.gif"       height="36"
alt="Amazon.com Checkout:  Payment" border="0" width="559" />
             </a>
           </td>
         </tr>
       </table>
       <br />
       <table border="0" cellspacing="0" cellpadding="0" width="90%" align="center">
         <tr>
           <td>
             <table    width="90%"    border="0"    align="center"    cellpadding="0"
cellspacing="0">
               <tr>
                 <td>
                   <b class="h1">Choose a Billing Address</b>
                   <br />
                   Your billing address must match the address that appears on your
monthly credit card or bank statement. You may select a billing address from your
address book below or enter a new billing address.  You'll  need to do this only once
for each credit card.
                 </td>
               </tr>
             </table>
```

Copyright by Mahidol University

```
            <br />
            <table    border="0"    cellspacing="0"    cellpadding="4"    width="100%"
align="center">
              <tr>
                <td class="headerCell" colspan="2">
                  <b>Address Book</b>
                </td>
              </tr>
              <tr>
                <xsl:apply-templates select="//billingaddress" />
              </tr>
              <tr>
                <td valign="top" width="50%">
                  <img    src="../images/green-pixel.gif"    alt="pixel"    height="1"
border="0" align="top" width="100%" vspace="6" />
                </td>
                <td valign="top" width="50%">
                  <img    src="../images/green-pixel.gif"    alt="pixel"    height="1"
border="0" align="top" width="100%" vspace="6" />
                </td>
              </tr>
            </table>
            <table     width="90%"     border="0"     align="center"     cellpadding="0"
cellspacing="0">
              <tr>
                <td>
                  <b class="h1">Or enter a new billing address</b>
                </td>
              </tr>
              <tr>
                <td>
                  <table border="0" cellpadding="2" cellspacing="2" width="100%">
                    <form action="" method="post">
                      <input type="hidden" name="creditCardID" value="qkoolxook" />
                      <tr>
                        <td align="right" nowrap="nowrap">
                          <b>Full Name: </b>

                        </td>
                        <td width="99%">
                          <input    type="text"    name="name"    value=""    size="50"
maxlength="60" />
                        </td>
                      </tr>
                      <tr>
                        <td align="right" nowrap="nowrap">
                          <b>Address Line1: </b>
```

Copyright by Mahidol University

```
                        </td>
                        <td width="%99">
                          <input   type="text"   name="address1"   value=""   size="50"
maxlength="60" />
                          <br />
                          <span class="tiny">Street address, P.O. box, company name,
c/o</span>
                        </td>
                      </tr>
                      <tr>
                        <td align="right" nowrap="nowrap">
                          <b>Address Line2: </b>

                        </td>
                        <td width="%99">
                          <input   type="text"   name="address2"   value=""   size="50"
maxlength="60" />
                          <br />
                          <span class="tiny">Apartment, suite, unit, building, floor,
etc.</span>
                        </td>
                      </tr>
                      <tr>
                        <td align="right" nowrap="nowrap">
                          <b>City: </b>

                        </td>
                        <td width="%99">
                          <input type="text" name="city" value="" size="25" />
                        </td>
                      </tr>
                      <tr>
                        <td align="right" nowrap="nowrap">
                          <b>State/Province/Region: </b>

                        </td>
                        <td width="%99">
                          <input type="text" name="state" value="" size="15" />
                        </td>
                      </tr>
                      <tr>
                        <td align="right" nowrap="nowrap">
                          <b>ZIP/Postal Code: </b>
                        </td>
                        <td width="%99">
                          <input type="text" name="zip" value="" size="20" />
                        </td>
                      </tr>
```

Copyright by Mahidol University

```
                        <tr>
                          <td align="right" nowrap="nowrap">
                            <b>Country: </b>

                          </td>
                          <td width="%99">
                            <select name="countryName">
                            ...
                            </select>
                          </td>
                        </tr>
                        <tr>
                          <td align="right" nowrap="nowrap">
                            <b>Phone Number: </b>

                          </td>
                          <td width="%99">
                            <input type="text" name="voice" value="" size="15" />
                          </td>
                        </tr>
                        <tr>
                          <td> </td>
                          <td>
                            <input type="image" src="../images/continue.gif" height="22"
alt="Continue" border="0" width="86" name="continue" />
                          </td>
                        </tr>
                      </form>
                    </table>
                  </td>
                </tr>
              </table>
              <hr noshade="true" size="1" />
              <p class="tiny" align="center">
                <a href="" target="AmazonHelp" onclick="">Conditions of Use</a>
                <a href="" target="AmazonHelp" onclick="">Privacy Notice</a>
                  ? 1996-2005, Amazon.com, Inc.
              </p>
            </td>
          </tr>
        </table>
      </body>
    </html>
  </xsl:template>
  <!--billing address-->
  <xsl:template match="billingaddress">
    <td valign="top" width="50%">
      <form name="billaddr" method="post" action="viewcart.xml">
```

Copyright by Mahidol University

```
        <input type="image" src="../images/use-this-address.gif" height="23" border="0"
width="125" name="submit" onclick="firerule( billaddr, this )" />
        <input type="hidden" name="ruleid" value="6" />
        <input type="hidden" name="ruletype" value="local" />
        <input type="hidden" name="ecAppName" value="www.amazon.com" />
        <input type="hidden" name="activity" value="" />
        <input type="hidden" name="id">
          <xsl:attribute name="value">
            <xsl:value-of select="@id" />
          </xsl:attribute>
        </input>
        <br />
        <b>
          <xsl:value-of select="/@recipient" />
        </b>
        <br />
        <xsl:value-of select="@addrno" />

        <xsl:value-of select="@road" />
        ,
        <xsl:value-of select="@subdistrict" />
        <br />
        <xsl:value-of select="@district" />
        ,
        <xsl:value-of select="@city" />

        <xsl:value-of select="@postcode" />
        <br />
        <xsl:value-of select="@country" />
        <br />
      </form>
    </td>
  </xsl:template>
</xsl:stylesheet>
```

An interaction of considerable classes, which relates to a construction of Active View by a Local Active Rule controller and the Remote Active Rule controller, is shown in the sequence diagrams in Figure 4-25 and Figure 4-26 respectively.

Copyright by Mahidol University

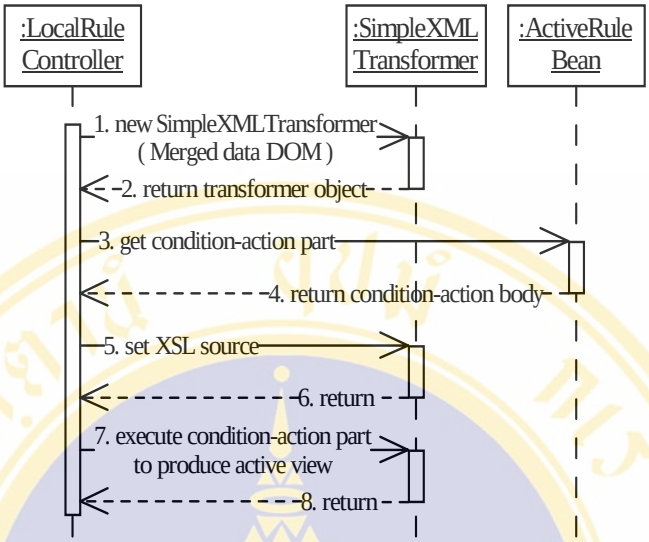


Figure 4-25 Sequence Diagram of LocalRuleController Class in Constructing an Active View

1) The LocalRuleController object calls the constructor method of SimpleXMLTransformer class in passing the merged data DOM as gained from the last sub-module.

2) The new object of SimpleXMLTransformer class is returned to the LocalRuleController object.

3) The LocalRuleController object asks the ActiveRuleBean object for the condition-action part of ongoing Local Active Rule.

4) The ActiveRuleBean object gives back the condition-action DOM to the issuer.

5-6) The LocalRuleController object sets an XSL for the SimpleXMLTransformer object the returned condition-action DOM and returns after done.

7-8) The LocalRuleController object calls the SimpleXMLTransformer object to produce a new Active View by transforming the merged data DOM applied by the condition-action instruction.

Copyright by Mahidol University

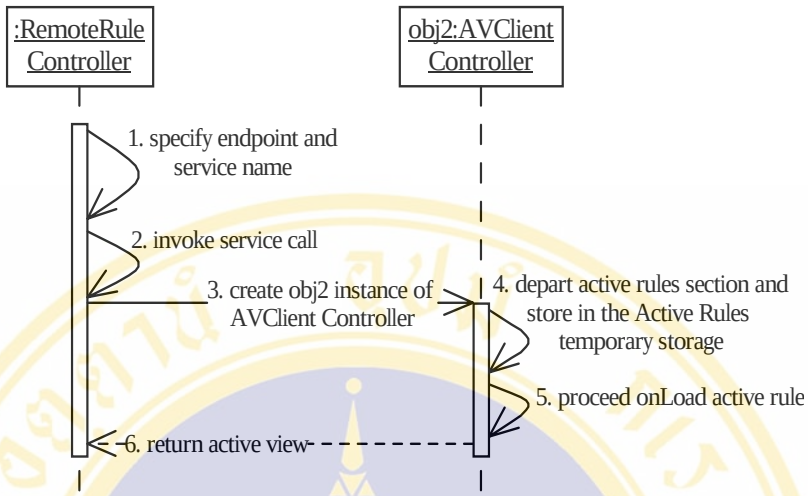


Figure 4-26 Sequence Diagram of RemoteRuleController Class in Constructing an Active View

1) The RemoteRuleController object specifies the endpoint where the AVML Processor is running and the AVML Processor's Web Service name.

2) The RemoteRuleController object invokes a service call to the AVML Processor in passing the three parameters which are the Rule ID. of firing Remote Active Rule, the referrer page, and the parameter string converted from its DOM form. The RemoteRuleController object waits for a new Active Web Page responded from the AVML Processor.

3) Once a new Active Web Page has arrived, the RemotRuleController object creates the new instance of AVClientController class to sort out the afore-mentioned tasks.

4) Let us recall the tasks that need to complete again. The new AVClientController object firstly departs the Active Rules section from the coming Active Web Page and then stores it in the Active Rules temporary storage.

5) The AVClientController object starts executing an onLoad Active Rule.

6) Finally, it turns out the first Active View of the Active Web Page and returns it to the RemoteRuleController object that recurrently returns to the original AVClientController object.

Copyright by Mahidol University

4.2.2 AVML Processor

Now, we check out the second tier which is the AVML Processor, another Active Rule engine at a server side. As discussed in Section 3.5.2 of Chapter III, the AVML Processor is comprised of the AVML Controller, and the Rule Processor. Similar to the AV Client's Lite Rule Processor, the one hereby consists of four sub-modules which are Active Rule Loader sub-module, Parameter Binder, Condition Evaluator, and Active Web Page Generator. We are about to start exploring each one of them later.

4.2.2.1 AVML Controller

The AVML Controller is a front module listening to a service call request for executing the triggered Remote Active Rule from the AV Client, fulfilling the request, and returning the resulting Active Web Page to the AV Client. It is logically equivalent to the Proxy Adaptor of the AV Client.

The AVML Controller is featured by Apache AXIS Web Service [33]. The Java class "AVAController" working for this module must be deployed in the AVML Processor Web Service Registry. Let us see all the classes concerning this module in the class diagram in Figure 4-27.

Copyright by Mahidol University

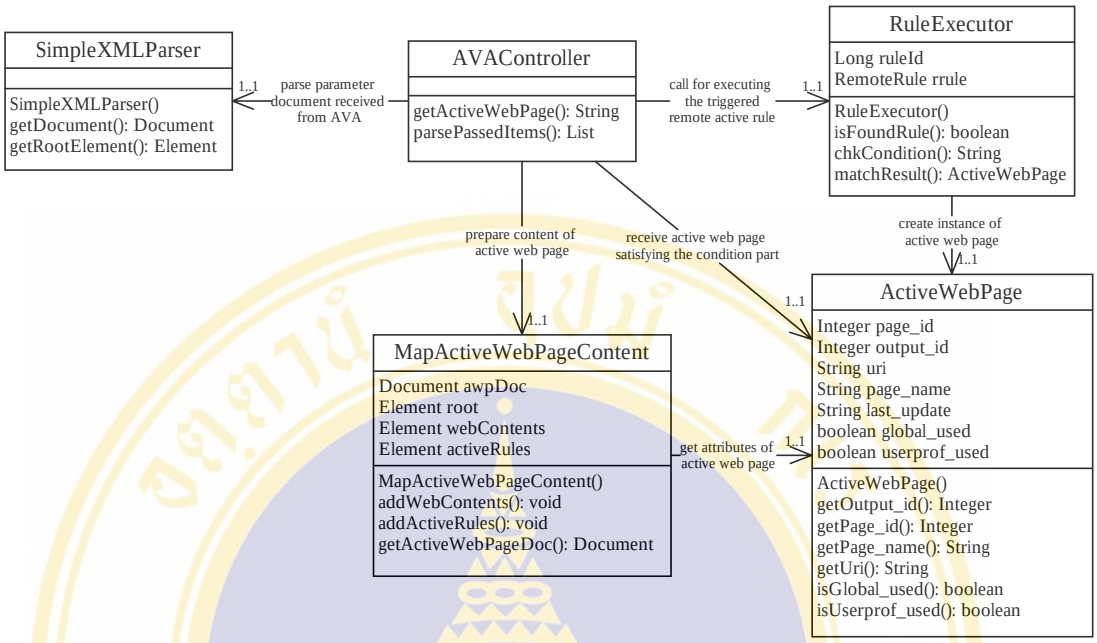


Figure 4-27 Class Diagram of AVML Controller Module

AVAController class is a central class of the module. The method *getActiveWebPage()* is invoked by the AV Client. As explained before, one argument needed by this method is a Service Call Parameters document. Therefore, AVAController class requires SimpleXMLParser class to parse the Service Call Parameters document and get back with its DOM form. Then, AVAController class interacts with RuleExecutor class to read the triggered Remote Active Rule from the Active Rules repository as specified by the Rule ID. The part of Remote Active Rule that will be read is the *rule body part*, where its definition is shown in Section 3.4 of Chapter III. RuleExecutor class controls a condition evaluation process that might require the Service Call Parameters DOM. When an evaluation result comes out, it has to find an appropriate Active Web Page fitting the result and its attributes loaded onto ActiveWebPage class Later, AVAController class cooperates with MapActiveWebPageContent class in materializing the items for the Web Contents section of this Active Web Page, Local Active Rules, and Remote Active Rules for the Active Rules section of this Active Web Page.

The sequence diagram in Figure 4-28 well gives an overview of AVML Controller work by quickly touring the main module called AVML Controller.

Copyright by Mahidol University

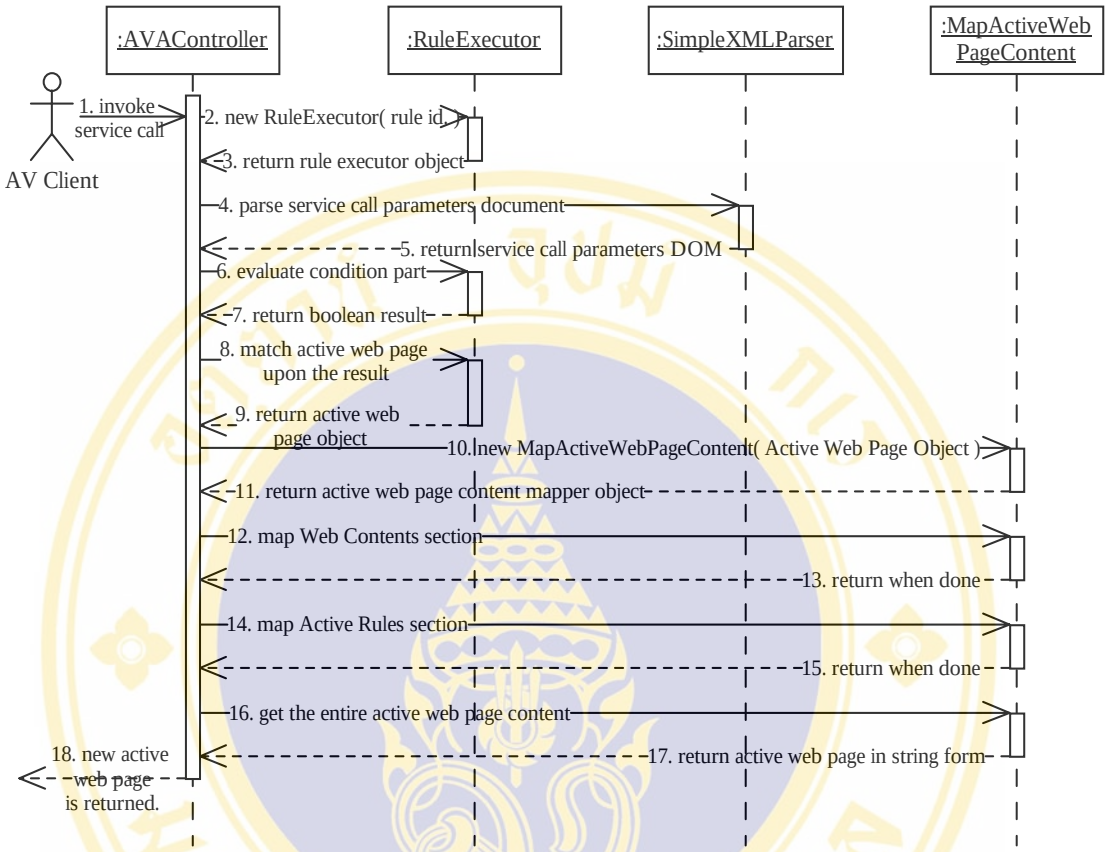


Figure 4-28 Sequence Diagram of AVML Controller Module

1) This is the only one entrance to input data for the AVML Processor. The AV Client invokes a service call for the Remote Active Rule triggered at a client side to the AVML Processor. The AV Client needs to supply the AVML Processor for the three parameters which are the *Rule ID.* of the fired Remote Active Rule, a *Service Call Parameters document* containing all items required in the Remote Active Rule execution here, and the *referrer web page*. AVAController class, a Java Web Service, has to fulfill this request.

2) AVAController class informs RuleExecutor class – the Remote Active Rule execution handler – to start executing the Remote Active Rule identified by the Rule ID. The constructor method of RuleExecutor class has to read its rule body part from the Active Rules repository.

3) The instance of RuleExecutor class is returned to AVAController class.

Copyright by Mahidol University

4) AVAController class calls SimpleXMLParser class to parse the Service Call Parameters document.

5) SimpleXMLParser class returns the content of Service Call Parameters document in the form of DOM structure to AVAController class.

6) AVAController class calls the RuleExecutor object to evaluate the condition part of the Remote Active Rule.

7) The RuleExecutor object gives back the Boolean result of condition evaluation – true or false – to AVAController class. The result comes out with the desired Active Web Page.

8) AVAController class requests the RuleExecutor class to find the target Active Web Page upon the Boolean value.

9) The RuleExecutor object forms the Active Web Page attributes into ActiveWebPage class and gives back to AVAController class.

10) AVAController class calls the constructor method of MapActiveWebPageContent class to map out the Web Contents items and the Active Rules required by the resulting Active Web Page.

11) The newly made instance of MapActiveWebPageContent class is given back to AVAController class.

12) AVAController class asks the RuleExecutor object to join the items belonging to the desired Active Web Page to the Web Contents section.

13) AVAController class is prompted when the items of Web Contents section have been mapped.

14) AVAController class asks the RuleExecutor object to join the Active Rules belonging to the desired Active Web Page to the Active Rules section.

15) AVAController class is prompted when the Active Rules of Active Rules section have been mapped.

16) AVAController class acquires the complete Active Web Page content from the MapActiveWebContent object.

17) AVAController class is prompted by the Active Web Page content in string form.

18) AVAController class returns the Active Web Page content as the result of AVAController web service call to the AV Client.

Copyright by Mahidol University

Later, we explore the Rule Processor supporting the AVML Controller.

4.2.2.2 Rule Processor

Since an Active View construction process is partly shifted to the AV Client, the *Rule Processor* here is just responsible for producing a new Active Web Page corresponding to the result of condition part evaluation toward the coming arguments.

4.2.2.2.1 Active Rule Loader

The Active Rule loader reads a rule body part and attributes of Remote Active Rule, identified by the parameter Rule ID. accompanied by a service call from the AV Client, from the Active Rules repository. All related classes, featuring this sub-module, are shown in the class diagram in Figure 4-29.

RuleExecutor class is the primary class that handles the Remote Active Rule execution task. At first, RuleExecutor class co-works with RemoteRuleImpl class to read the triggered Remote Active Rule from a database. To achieve this, RemoteRuleImpl class associates with PoolingDB class to manage the database operations. Second, once the rule body part and attributes of triggered Remote Active Rule are retrieved, RemoteRuleImpl class has them to build an instance of RemoteRule class. The content of Remote Active Rule is ready to be referenced by RuleExecutor class.

Copyright by Mahidol University

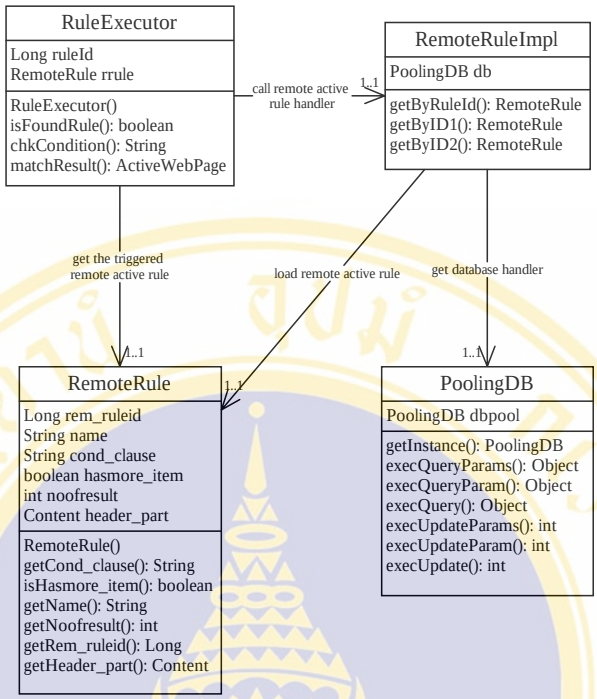


Figure 4-29 Class Diagram of Active Rule Loader Sub-module

The sequence diagram in Figure 4-30 shows a reaction between these classes.

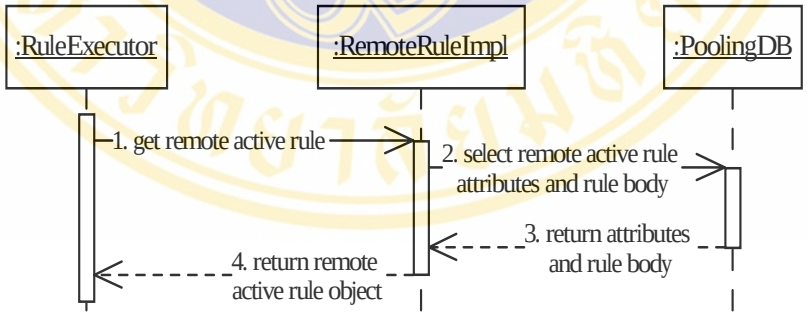


Figure 4-30 Sequence Diagram of Active Rule Loader Sub-module

1) RuleExecutor class asks RemoteRuleImpl class to read the body part of ongoing Remote Active Rule from the Active Rules repository.

2) RemoteRuleImpl class connects the database thru PoolingDB class to select the body and also its attributes of Remote Active Rule.

3) The record of Remote Active Rule details is returned to RemoteRuleImpl class. It creates the new instance of RemoteRule class based on the details.

4) The RemoteRule object is returned to RuleExecutor class.

Copyright by Mahidol University

4.2.2.2.2 Parameter Binder and Condition Evaluator

By the nature of their working relationship, we combine the Parameter Binder and the Condition Evaluator to be explained together.

The *Parameter Binder* is responsible for binding items required to judge a condition part of Remote Active Rule with those which are accompanied by a Service Call Parameters document. We give an example of the condition part in Example 4-13. We are looking for the question mark (?) that hints how many items need to be bound. In Example 4-13, it gets three items to fill regarding the three occurrences of question mark. Due to the fact that an order of '?' appearing on a condition part is the same in the Service Call Parameters DOM, it does the straight matching of them. The condition part after the binding process was done looks like Example 4-14.

Example 4-13 An Example of A Condition Part of a Remote Active Rule

```
<if>
 <predicate type="SQL">
  select 1 from user_profile where username = ? and password = ?
 </predicate>
 <AND/>
 <predicate type="conditional">
  ? > 5
 </predicate>
</if>
```

Example 4-14 A Condition Part After the Binding Operation

```
<if>
 <predicate type="SQL">
  select 1 from user_profile where username = 'riam' and password = 'password'
 </predicate>
 <predicate type="conditional">
  6 > 5
 </predicate>
</if>
```

Continuously, the *condition evaluator* does evaluate a condition part to the Boolean value. If a condition part is satisfied – the Boolean *true* is returned, one

Copyright by Mahidol University

Active Web Page will be resulted. Otherwise, a condition part is not satisfied – the Boolean false is returned, this causes another different Active Web Page to be resulted.

We present the class nod of these two sub-modules in the class diagram in Figure 4-31.

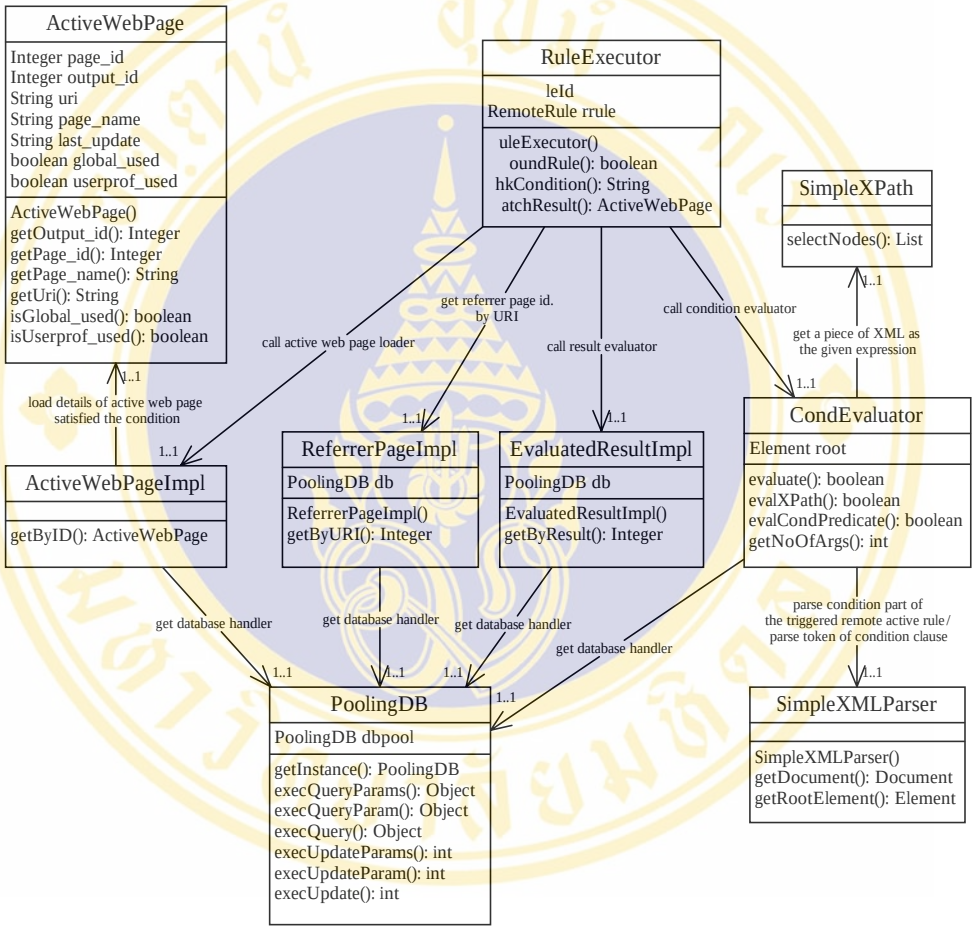


Figure 4-31 Class Diagram of Parameter Binder and Condition Evaluator Sub-modules

RuleExecutor class associates with CondEvaluator class to control a condition evaluation and a new implicit AVML document derivation. CondEvaluator class first has SimpleXMLParser class to get a condition part parsed. A condition part in DOM form is ready to be picked up. CondEvaluator class second analyzes a condition part and sees if any items of this part need to obtain their values. If this is the case, it has to match them with the conforming ones in the Service Call Parameters DOM. Third,

Copyright by Mahidol University

CondEvaluator class start evaluating each predicate found in a condition part toward its predicate type. In this circumstance, CondEvaluator might tie in SimpleXPath class or PoolingDB class to succeed. We explain the details of condition evaluation process next. Besides of CondEvaluator class, in the event any classes want to associate with a database server, they are through a supporting of PoolingDB class. CondEvaluator class will finalize an evaluation result. Fourth, RuleExecutor class asks EvaluatedResultImpl class to find an output identifier that meets the Boolean result from a database and also joins ReferrerPageImpl class to gain an identifier of referring URI passed from the AV Client. RuleExecutor class makes use of these two identifiers to retrieve details of desired Active Web Page from a database via ActiveWebPageImpl class. Lastly, ActiveWebPageImpl class returns the details in the form of ActiveWebPage class.

Figure 4-32 illustrates how CondEvaluator class evaluates a condition part. First, let us study an element of condition part. As in Example 4-14, the element <if> indicates a condition part and its sub-element <predicate> is a criteria expression. At least a predicate is a must, but more predicates are allowed under the element “if”; notwithstanding, they must be delimited by the *conditional* operators AND or OR. Besides, the predicate type needs to be specified with the value of attribute “type”. There are three predicate types allowed in our approach as follows:

1) ***SQL***: It is written in a SELECT form as in Example 4-14. This means CondEvaluator class needs to have this SQL statement executed in a database. If an execution yields nothing, an evaluation result is false, otherwise, an evaluation result is true;

2) ***XPath***: It is written in an XPath form. This means CondEvaluator class needs to have this XPath expression evaluated on the target XML document. If a evaluation yields nothing, an evaluation result is false, otherwise – a piece of XML is returned, an evaluation result is true;

Copyright by Mahidol University

3) ***Conditional***: it is written in an infix notation, which means that the *relational* operators appear *between* its operands like *op1 [=, !=, >, <, >=, <=] op2*. An evaluation of this conditional expression goes to *true* or *false* as summarized in Table 4-1.

Table 4-1 List of Relational Operators [34]

Operator	Use	Description
=	op1 = op2	Returns true if op1 and op2 are equal
!=	op1 != op2	Returns true if op1 and op2 are not equal
>	op1 > op2	Returns true if op1 is greater than op2
<	op1 < op2	Returns true if op1 is less than op2
>=	op1 >= op2	Returns true if op1 is greater than or equal to op2
<=	op1 <= op2	Returns true if op1 is less than or equal to op2

As presented in Figure 4-32, CondEvaluator class begins to read the block of condition part governed by the element <if>. It gets a predicate at a time and checks what its type is classified. It scans through a content of predicate to find a question mark and replace with the item value from the Service Call Parameters DOM no matter what its type (SQL, XPath, and Conditional) is. If the type goes to SQL, it will make inquiry over a database, or else, the type goes to XPath or Conditional, CondEvaluator class will need to assess an expression and save a result as the guideline stated above in an array of boolean results. However, if the conditional operators AND or OR are found, CondEvaluator class will just save them in the same array. It continues reading the next predicate until no more is found. At last, we get the array of each predicate's result like Example 4-15.

Copyright by Mahidol University

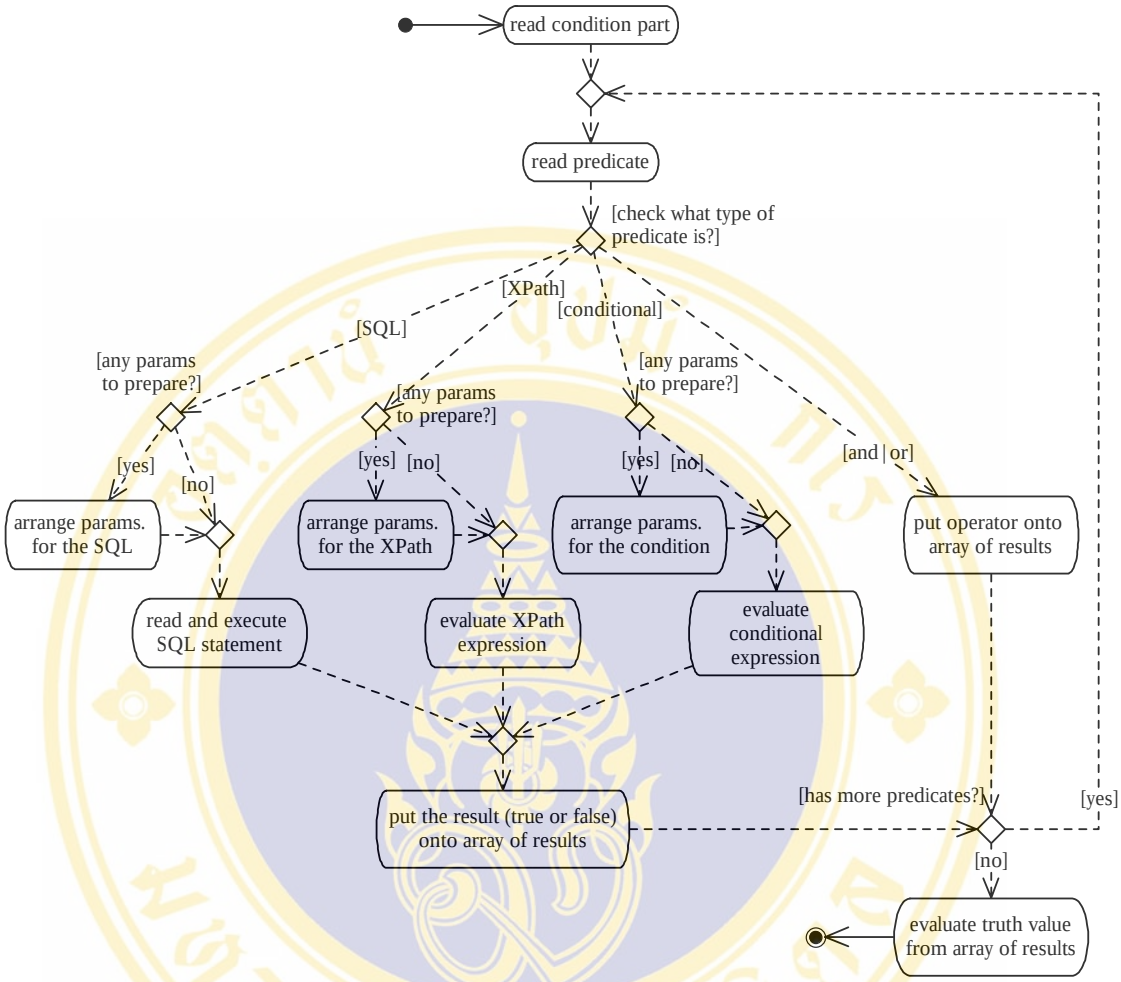


Figure 4-32 Activity Diagram of Condition Evaluator Sub-module

Example 4-15 Array of Evaluation Results of Each Predicate

```
result[0] = true
result[1] = OR
result[2] = false
result[3] = AND
result[4] = true
```

In the case of that there is a single predicate under the element <if>, an evaluation result of a condition part will be an evaluation result of a predicate. On the other hand, in Example 4-15, CondEvaluator class needs to save each predicate's evaluation result in memory and then read a pair of them and operate it by its original

Copyright by Mahidol University

conditional operator at a time till a last pair. This would output an evaluation result of condition part.

In Example 4-15, firstly, the entry [0] operates the entry [2] with an operator in the entry [1], so we get *true*. Then, we put *true* operating the entry [4] with the operator in the entry [3], so we get *true* as a final result.

Figure 4-33 and Figure 4-34 show the continuous sequence diagrams covering all the aspects of condition evaluation scheme.

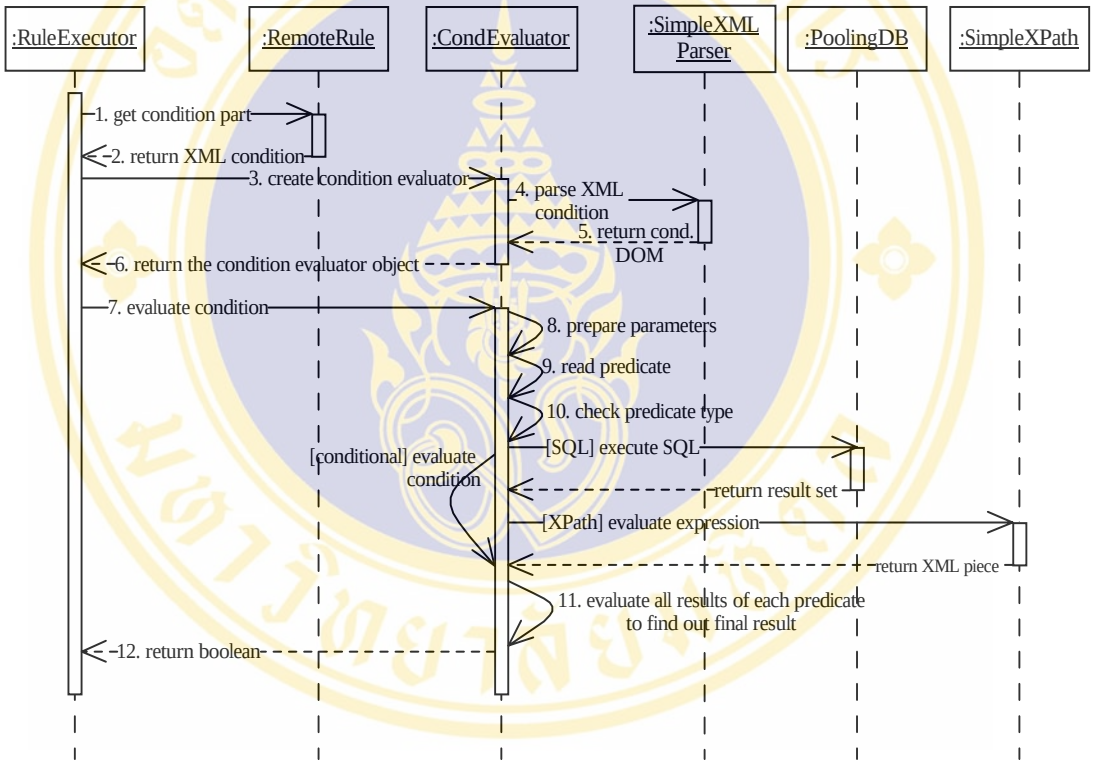


Figure 4-33 Sequence Diagram of Condition Evaluation

1) The RuleExecutor object which has been already created on the prior sub-module gets a condition part from the RemoteRule object.

2) A condition content in XML notion (specified by the element <if>), where its definition has been discussed in Section 3.4 of Chapter III, is returned to the RuleExecutor object.

3) The RuleExecutor object creates an instance of CondEvaluator class in passing a condition content as an argument.

Copyright by Mahidol University

4) In a constructor method of CondEvaluator class, it has to call SimpleXMLParser class to parse a condition content to get a DOM structure of condition content.

5) The instance of CondEvaluator class is built and returned to the RuleExecutor object.

6) The new instance of CondEvaluator class is returned to the RuleExecutor object.

7) The RuleExecutor object asks the CondEvaluator object to evaluate the holding condition.

8) The CondEvaluator object prepares parameters accompanied by a service call invocation that might be required in binding the items.

9) The CondEvaluator object explores each predicate of a condition part.

10) The CondEvaluator object checks which the type of each predicate is.

- If its type is ***SQL* statement**, it will ask PoolingDB class to get the SQL statement run. The number of row returned indicates what Boolean result is determined. If there is nothing returned, an evaluation for this predicate will go to false, or else, an evaluation will go to true;

- if its type is ***XPath* expression**, it will ask SimpleXPath class to find the piece of XML data in the Service Call Parameters DOM as to the XPath expression. Any pieces of data found indicate what Boolean result is determined. If there is nothing found, an evaluation for this predicate goes to false, or else, an evaluation goes to true;

- if its type is ***Conditional* expression**, it will resolve a conditional expression to get truth values – true or false.

The CondEvaluator object has to find the Boolean result of each predicate till the last one and arrange the Boolean result into an array of Boolean.

11) Later, the CondEvaluator object will bring out the final Boolean result by checking out each entry of Boolean array and having it operated by the operands like

Copyright by Mahidol University

AND or *OR* regarding the specification of condition part of Remote Active Rule in Section 3.4 in Chapter III.

12) The CondEvaluator object returns the finalized Boolean value to the RuleExecutor object.

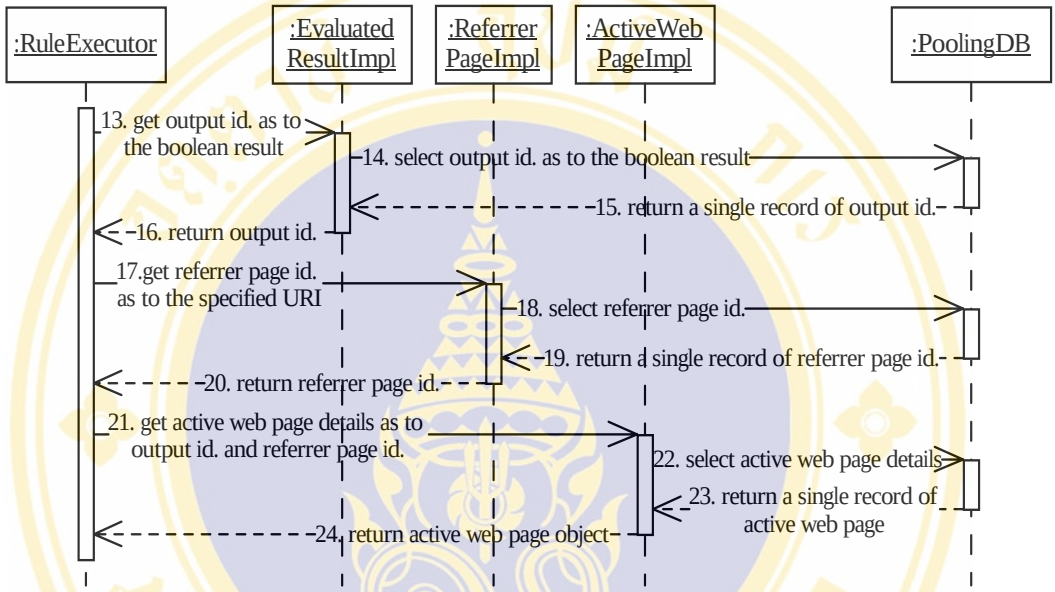


Figure 4-34 Sequence Diagram of Condition Evaluation (cont.)

13) The RuleExecutor object calls EvaluatedResultImpl class to get an *output id.* corresponding to the Boolean result derived previously.

14) EvaluatedResultImpl class would ask PoolingDB class to select the output id. introduced by this Boolean result from a database.

15) EvaluatedResultImpl class is returned by the output id. record from PoolingDB class.

16) EvaluatedResultImpl class carries the output id. to the RuleExecutor object.

17) The RuleExecutor object calls ReferrerPageImpl class to get a *page id.* corresponding to the referrer web page argument.

18) ReferrerPageImpl class asks PoolingDB class to select the page id. introduced by the referrer page id. from the Service Call Parameters DOM.

Copyright by Mahidol University

19) ReferrerPageImpl class is returned by the record of page id. from PoolingDB class.

20) ReferrerPageImpl class carries the page id. to the RuleExecutor object.

At this moment, we have the output id. and the page id. in the bag. They are important to derive an output – a new implicit AVML document.

21) The RuleExecutor instance requests ActiveWebPageImpl class to find the destined implicit AVML document.

22) ActiveWebPageImpl class asks PoolingDB class to retrieve the details of implicit AVML document from a database.

23) ActiveWebPageImpl class is prompted by the record of Active Web Page details that provides the attributes of Active Web Page and what items and Active Rules are required. It creates a new instance of ActiveWebPage based on these details.

24) The ActiveWebPage object is given back to the RuleExecutor object.

4.2.2.2.3 Active Web Page Generator

This sub-module is responsible for acquiring contents of a Web Contents section and a Active Rules Section of a resulting implicit AVML document. Generally, an *implicit* AVML document describes what are required in these sections; nevertheless, it does not save the real data in the context of implicit AVML document permanently by virtue of the nature of data dynamically changed upon different views of users. The implicit AVML document has even no attribute and empty Web Contents section and Active Rules section as illustrated in Section 3.4 of Chapter III. However, an *explicit* AVML document or an Active Web Page is full of attributes and two sections. There are a lot of classes acting for the sub-module as shown in Figure 4-35.

Copyright by Mahidol University

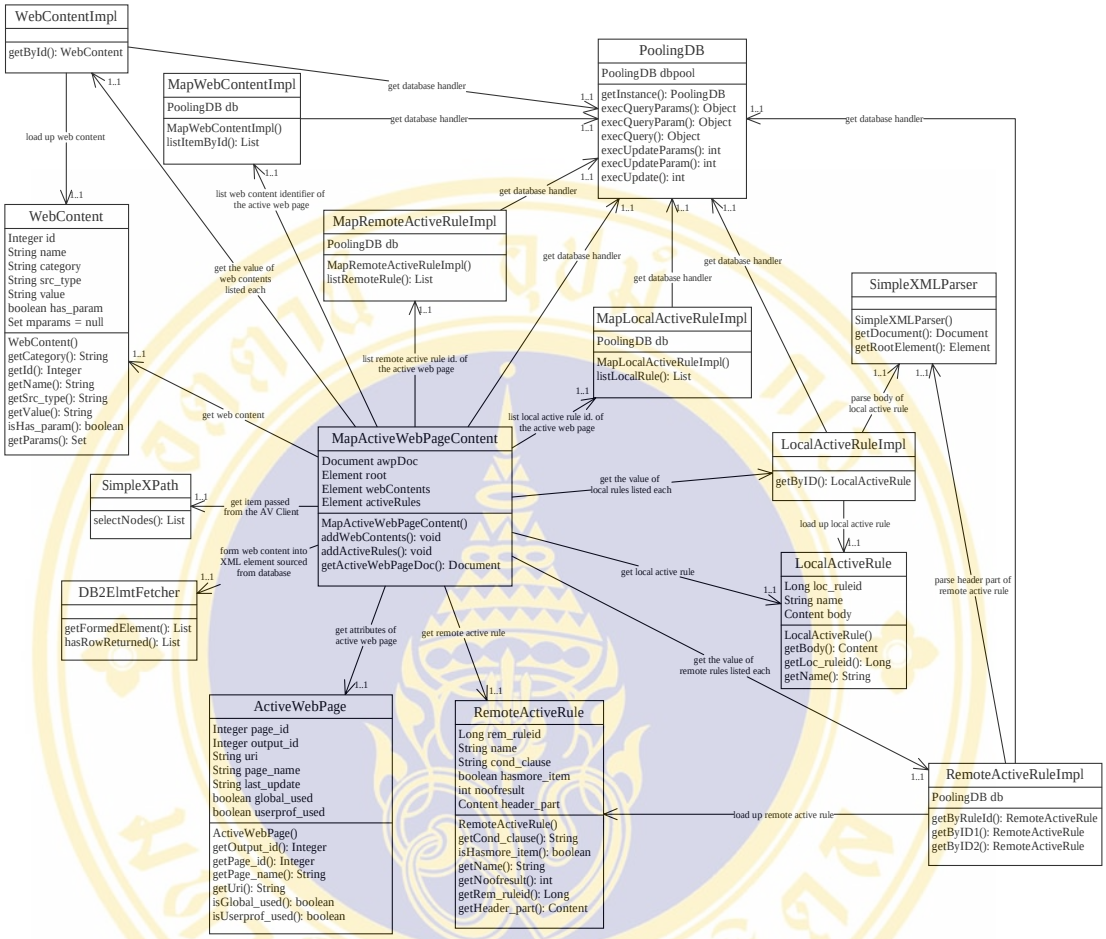


Figure 4-35 Class Diagram of Active Web Page Generator Sub-module

Likewise, PoolingDB class is seen as an arbitrated class that provides other classes for an easy and convenient way to control database operations.

Firstly, MapActiveWebPageContent class gets attributes of Active Web Page from ActiveWebPage class and adds them under the sub elements <global> and/or <UserProfile> of the element <ActiveWebPage>. Secondly, it cooperates with MapWebContentImpl class in listing all item identifiers belonging to the Web Contents section of this Active Web Page. After receiving the list of item id., MapActvieWebPageContent class cooperates with WebContentImpl class in fetching details of each item and elaborating WebContent class. Then, MapActiveWebPageContent class creates its well-formed of it via DB2ElmtFetcher class and adds it to the Web Contents section.

Copyright by Mahidol University

Thirdly, MapActiveWebPageContent class cooperates with MapLocalActiveRuleImpl class in listing all Local Active Rule identifiers belonging to the Active Rules section of this implicit AVML document. After receiving the lists of Local Active Rule id., MapLocalActiveRuleImpl class cooperates with LocalActiveRuleImpl class in fetching a body of Local Active Rule and elaborating LocalActiveRule class. Then, MapActiveWebPageContent class takes the body from LocalActiveRule class and adds it to the Active Rules section.

In the end, MapActiveWebPageContent class cooperates with MapRemoteActiveRuleImpl class in listing all the Remote Active Rule identifiers belonging to the Active Rules section of this implicit AVML document. After receiving the lists of Remote Active Rule id., MapRemoteActiveRuleImpl class cooperates with RemoteActiveRuleImpl class in fetching a data binding part of Remote Active Rule and elaborating RemoteActiveRule class. Then, MapActiveWebPageContent class takes the data binding part from RemoteActiveRule class and adds it to the Active Rules section.

Figure 4-36 and Figure 4-37 display the sequence diagrams capturing the mapping scheme of Web Contents items and two types of Active Rules respectively.

1) As created by AVAController class earlier, the MapActiveWebPageContent object gets the attributes in need from the ActiveWebPage object.

2) The ActiveWebPage object returns all attributes of Active Web Page to the MapActiveWebPageContent object.

3) The MapActiveWebPageContent object manages to prepare a layout where to place Active Web Page attributes, Web Contents attributes and items, and Active Rules.

4) The MapActiveWebPageContent object asks MapWebContentImpl class to list item id(s). belonging to the implicit AVML document in passing the output id. and page id. as parameters.

5) MapWebContentImpl class calls PoolingDB class to select item id(s). which are specified by the output id. and page id. from a database.

6) PoolingDB class returns a result set of item id(s). to MapWebContentImpl class.

Copyright by Mahidol University

7) MapWebContentImpl class carries the list of item ids. to the MapActiveWebPageContent object.

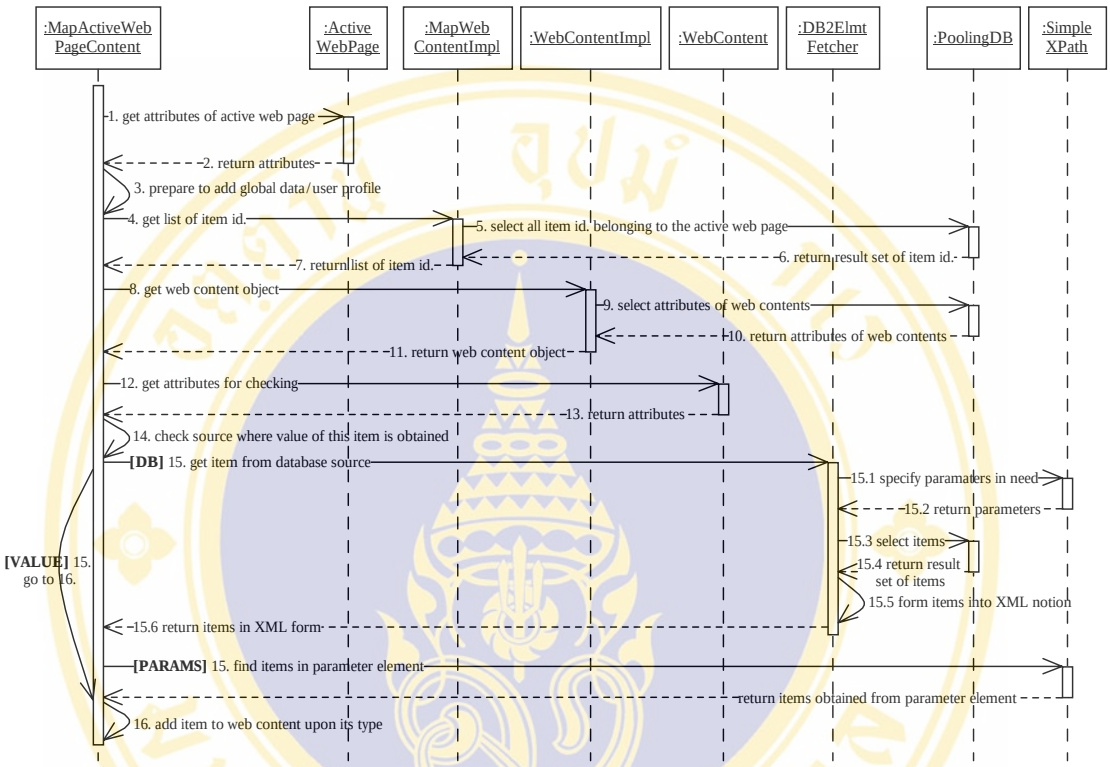


Figure 4-36 Sequence Diagram of Active Web Page Generator Sub-module

8) The MapActiveWebPageContent object prompts WebContentImpl class to obtain the detail *for each* item id. and form it into the WebContent object.

9) WebContentImpl class asks PoolingDB class to retrieve details of item from a database first.

10) The record of item details is returned to WebContentImpl class that will get them formed into the new object of WebContent class.

11) The WebContent object is given back to the MapActiveWebPageContent object.

12) The MapActiveWebPageContent object gains the source type of item indicating where to obtain the value of item from the WebContent object.

13) The WebContent object responds the source type as requested.

14) The MapActiveWebPageContent object verifies what the type of source is.

Copyright by Mahidol University

15) The MapActiveWebPageContent object does different actions against the following sources:

- If the type of source goes to ***DB***, the item acquisition will be described in SQL style. The MapActiveWebPageContent object will ask DB2ElmtFetcher class to obtain the value of item. In this regard, DB2ElmtFetcher class needs to know if any parameters are bound with this SQL statement before sending it to execute. *(labeled 15.1-15.2)* In this case, DB2ElmtFetcher class will find from the Service Call Parameters DOM. *(labeled 15.3-15.4)* Then, DB2ElmtFetcher class has the SQL executed by PoolingDB class and waits for the row returned. *(labeled 15.5)* DB2ElmtFetcher class constitutes the XML form of the record returned, then sends back to the MapActiveWebPageContent object and goes on the step 16;

- if the type of source goes to ***VALUE***, the value of item will be obtained at once because its exact value has been already given;

- if the type of source goes to ***PARAMS***, it will be defined in the form of XPath expression whose the target XML is the Service Call Parameters DOM. The MapActiveWebPageContent object asks SimpleXPath class to simply address the value in the Service Call Parameters DOM.

16) The MapActiveWebPageContent object adds the XML item to the Web Contents section of Active Web Page layout.

17) The MapActiveWebPageContent object gets the output id. and the page id. from the ActiveWebPage object.

18) The ActiveWebPage object responds them to the MapActiveWebPageContent object as requested.

19) The MapActiveWebPageContent object urges MapLocalActiveRuleImpl class in passing the output id. and the page id to prepare a list of Local Active Rule id. belonging to this Active Web Page.

20) MapLocalActiveRuleImpl class asks PoolingDB class to select Local Active Rule id(s). which correspond to the output id. and the page id. from a database.

21) PoolingDB class gives back the result set of Local Active Rule ids to its caller.

Copyright by Mahidol University

22) MapLocalActiveRuleImpl class keeps forwarding this list to the MapActiveWebPageContent object.

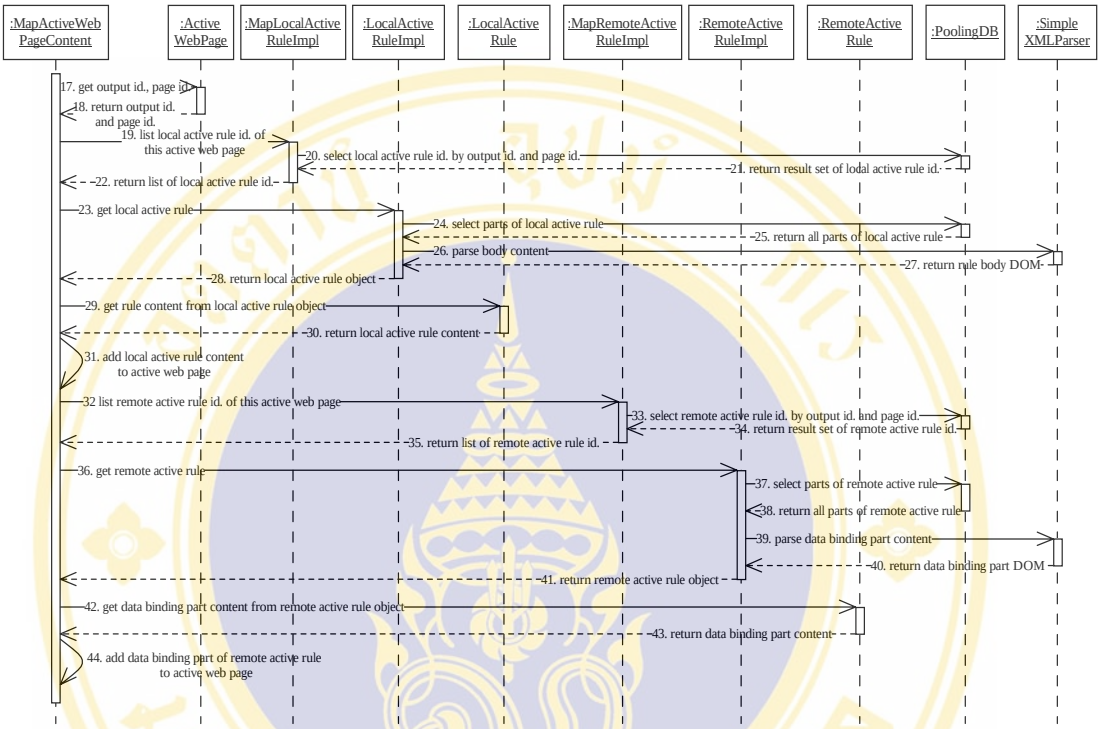


Figure 4-37 Sequence Diagram of Active Web Page Generator Sub-module (cont.)

From now on, the MapActiveWebPageContent object needs to collect the content of *all* the Local Active Rules appearing in the list returned.

23) In Practice, the MapActiveWebPageContent object has to get details of Local Active Rule from the LocalActiveRule object for further actions. To accomplish this, the MapActiveWebPageContent object asks LocalActiveRuleImpl class to complete.

24) LocalActiveRuleImpl class calls PoolingDB class to select details of Local Active Rule from the database.

25) PoolingDB class returns the record of Local Active Rule details to the issuer class. One of details is a body content expressed in XML-formatted.

26) Thus, LocalActiveRuleImpl class continuously requests SimpleXMLParser class to parse a body content.

Copyright by Mahidol University

27) SimpleXMLParser class returns the DOM structure of rule body to LocalActiveRuleImpl class which forms it and any other details to the new LocalActiveRule object.

28) The LocalActiveRule object is given back to the MapActiveWebPageContent object.

29) The MapActiveWebPageContent object requests the LocalActiveRule object for a body content of Local Active Rule.

30) At this moment, the rule body DOM is ready to be served to the MapActiveWebPageContent object.

31) The MapActiveWebPageContent object takes the rule body DOM to be added to the Active Rules section of Active Web Page.

From this onward, the MapActiveWebPageContent object has to do similarly with the Remote Active Rule.

32) The MapActiveWebPageContent object urges MapRemoteActiveRuleImpl class in passing the output id. and the page id to prepare lists of Remote Active Rule id. belonging to this Active Web Page.

33) MapRemoteActiveRuleImpl class asks PoolingDB class to select Remote Active Rule id(s). which correspond to the output id. and the page id. from a database.

34) PoolingDB class gives back the result set of Remote Active Rule id(s). to its caller.

35) MapRemoteActiveRuleImpl class keeps forwarding this result set to the MapActiveWebPageContent object.

Next, the MapActiveWebPageContent object needs to collect the *data binding* part of *all* the Remote Active Rules appearing in the list returned.

36) In Practice, the MapActiveWebPageContent object has to get details of Remote Active Rule from the RemoteActiveRule object for further actions. To accomplish this, the MapActiveWebPageContent object asks RemoteActiveRuleImpl class to complete.

37) RemoteActiveRuleImpl class calls PoolingDB class to select details of Remote Active Rule from a database.

Copyright by Mahidol University

38) PoolingDB class returns the record of Remote Active Rule details to the issuer class. One of details is a data binding part expressed in XML-formatted.

39) Thus, RemoteActiveRuleImpl class continuously requests SimpleXMLParser class to parse a data binding part content.

40) SimpleXMLParser class returns the DOM structure of data binding part to RemoteActiveRuleImpl class which forms it and any other details to the new RemoteActiveRule object.

41) The RemoteActiveRule object is given back to the MapActiveWebPageContent object.

42) The MapActiveWebPageContent object requests the RemoteActiveRule object for the data binding part content of Remote Active Rule.

43) At this moment, the data binding part DOM is ready to be served to the MapActiveWebPageContent object.

44) The MapActiveWebPageContent object takes the data binding part DOM to be added to the Active Rules section of Active Web Page.

Now, we get the complete Active Web Page materialized with the item(s) in a Web Contents section and Local Active Rules and/or Remote Active Rules in an Active Rules section. The complete Active Web Page is returned as a result of the service call processing by the AVML Processor to the AV Client.

4.3 Communication Interface Design

This section discusses a specification of communication between the main components of the A^2V framework. That is, we are talking about an interconnection between a client side and a server side. As illustrated in Figure 3-2 of Chapter III, both client side and server side are tied to the Internet network. Primarily, there are two network protocols employed in our framework as follows:

Copyright by Mahidol University

4.3.1 SOAP Protocol

This protocol is used for a communication between the AV Client at a client side and the AVML Processor at a server side. The reason why this protocol is engaged in this regime is that the AVML Processor is such a Web Service that is activated via only a SOAP protocol. *A SOAP envelope is created by the AV Client and submitted to* the AVML Processor *when a user request caused a Remote Active Rule to be triggered.* When the AVML Processor has carried out a service call, it generates a new SOAP message containing the resulting of Active Web Page and gives back to the AV Client.

4.3.2 HTTP Protocol

This protocol is used for the following pairs of components:

- *Communication between a user's web browser and its AV Client which are together at a client side*: This takes place when a user submits a request for a new generic web page/a graphical file to a web server or performs any activities on a current Active View that causes the specified event to trigger an Active Rule belonging to its original Active Web Page. The AV Client accepts these two possible ways of request and executes them to produce an Active View and then posts to a user's web browser via HTTP protocol.

- *Communication between the AV Client at a client side and a web server at a server side*: This happens when the AV Client accepts the request from a user's web browser and realizes that it is a request *for a generic web page/a graphical file*. The AV Client is going to forward it to a web server to deal with and also receives a response HTML web page/a response graphical file from a web server via HTTP protocol.

Copyright by Mahidol University

4.4 Output Design

This section describes an output specification of our framework. Easily, we divide an output form into two forms as follows:

4.4.1 Explicit Output

It is an Active View present on a user's web browser. It results from an Active View construction processing of the AV Client in cooperating with the AVML Processor and a web server. Since the AV Client transparently runs as the background process on a user's machine monitoring all incoming connections from either a user's web browser or a server side, a user does not see any interface screens of the AV Client. Instead, a user just sees only an Active View display.

4.4.2 Application Log

However, the purpose of our framework is to propose an alternative method employing a decentralized control structure to improve a performance of Active View construction. It is required to have an application log of the AV Client so that we do analyze log contents and find an appropriate conclusion. We provide an analysis of logs in details in Chapter VI.

4.5 Implementation Environment

This last section outlines the hardware, software, and network requirement to implement the prototype of A^2V framework. There are two main processing units of A^2V framework which are the AV Client and the AVML Processor.

In our development environment, the AV Client and the AVML Processor are implemented and work on the same machine. In the production environment, the AV

Copyright by Mahidol University

Client and the AVML Processor are however supposed to be at a client side and a server side respectively. The AV Client runs as a Java application whereas the AVML Processor is deployed as a Web Service under control of AXIS Web Service framework.

We provide hardware and software specifications of the AV Client and the AVML Processor and a network specification connecting them as follows:

4.5.1 AV Client Specification

- CPU: Intel Centrino Mobile Technology 1.6 GHz
- Minimum 512MB Main Memory
- Minimum Hard Disk Space: 400-500 MB
- Network Card 10/100 Mbps
- Operating System: Microsoft Windows XP Professional or 2000 Server/Professional
- Web Browser: Microsoft Internet Explorer version 6.0 SP1
- Development Tools:
    - J2SE version 1.4.1_02
    - Eclipse Platform version 3.0 RC
    - Proxy Server Java Class

4.5.2 AVML Processor Specification

- CPU: Intel Centrino Mobile Technology 1.6 GHz
- Minimum 512MB Main Memory
- Minimum Hard Disk Space: 400-500 MB
- Network Card 10/100 Mbps
- Operating System: Microsoft Windows XP Professional or 2000 Server/Professional

Copyright by Mahidol University

- Web Server & Servlet Engine: Apache Jakarta Tomcat version 4.1.31
- DBMS: MySQL Database Server version 4.1.12
- Web Browser: Microsoft Internet Explorer version 6.0 SP1
- Development Tools:
    - J2SE version 1.4.1_02
    - Eclipse Platform version 3.0 RC
    - Web Service Framework: Apache Axis version 1.2

4.5.3 Network Requirement

- Network Cable 10 or 100 Mbps

In the next chapter, we will describe the demonstrated case study used for conducting an experiment through this prototype.

Copyright by Mahidol University

CHAPTER V
DEMONSTRATED CASE STUDY

In this chapter, we present an ordering processing e-service application case study that we will apply our approach to, its business processes, and web interfaces. Also, we are going to describe how to convince these examples of web application to be featured by the A^2V framework.

Ordering Process e-Service Application Case Study

This case study simplifies an ordering process of the Amazon web site [35] in dividing into three steps which are a signing-in step, items browsing step, an order summarizing steps whose details are provided in the next section.

5.1 Process Description of Ordering Process

The activity diagram in Figure 5-1 illustrates a flow of the customized ordering process. There are three sub-processes of

- *Sign-in sub-process*

This is to let a user to enter the sign-in name and password, and submit them to a web application so that it validates this information toward the correct ones in the database (a user authentication). If a user is certified, a user's profile will be fetched from a database and recognized who a user is and which data or operations are allowed to a user. In the activity diagram, this sub-process is composed of two web pages which are an index page in Figure 5-2 and a sign-in page in Figure 5-3.

Copyright by Mahidol University

- *Cart browsing sub-process*:

This is to display items and their details in a user's cart such as an item name, its category, a price per unit, and a quantity. A user is allowed to add more items to a user's cart, change a quantity, and review all the items a user would like to buy prior to proceeding to check out. This sub-process has only one web page that is a cart viewing page as depicted in Figure 5-4.

- *Order checking-out sub-process*:

This is when a user is set for placing a new order. A user has to specify additional information which is required by a web application. Such are a shipping address, a shipping option, a payment method, and a billing address. The dissimilar user profile that has been known since the first sub-process dominates a content remained in a view of web page. For instance, in the shipping option choosing page, a view shows sensible options if the system detects a residence where a user lives. Another example is in the order reviewing page. If a user bought a DVD item, there will be an alert informing a DVD player might not accept some DVD encoding formats.

This sub-process is composed of five web pages which are a shipping address specifying page, a shipping option selection page, a payment method selection page, a billing address specifying page, and an order reviewing page as illustrated in Figure 5-5 – Figure 5-10 severally.

Copyright by Mahidol University

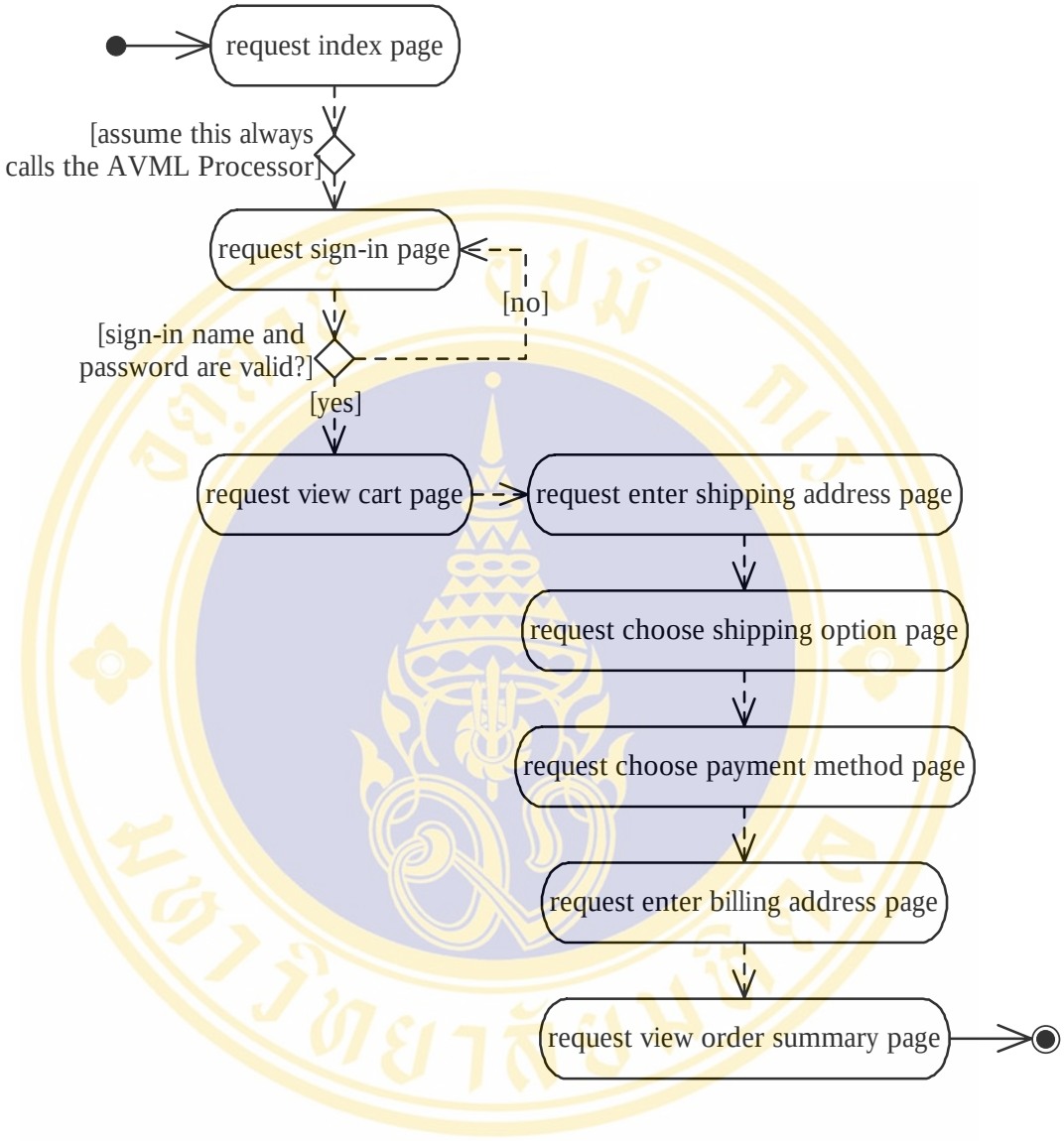


Figure 5-1 Activity Diagram of Ordering Process

Copyright by Mahidol University

5.2 Web Interface of Ordering Process

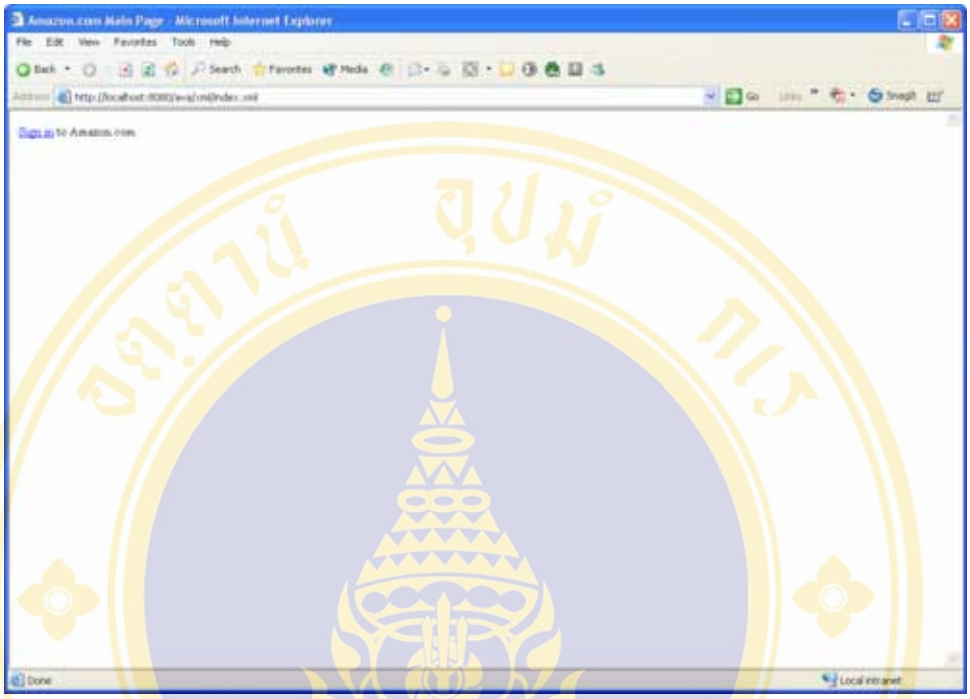


Figure 5-2 First screen (dummy screen)

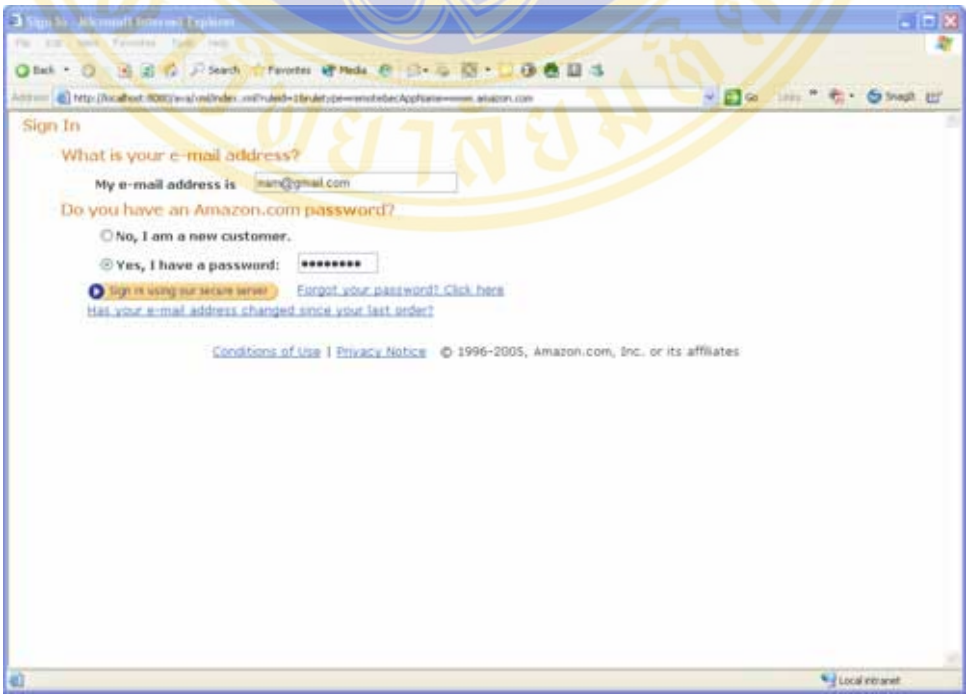


Figure 5-3 Sign-in Screen

Copyright by Mahidol University

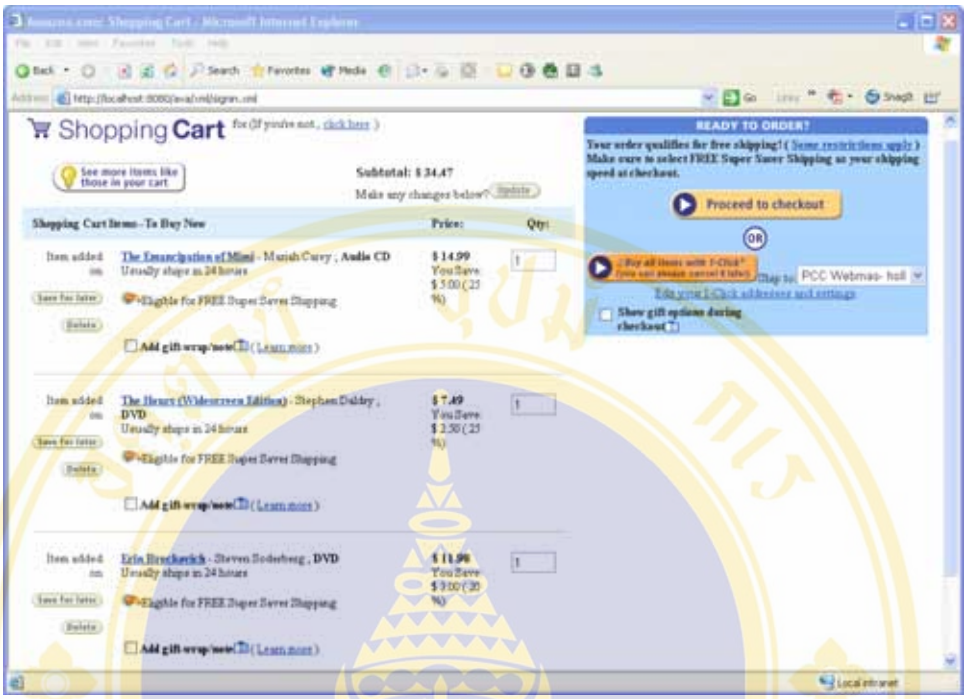


Figure 5-4 Cart Viewing Screen

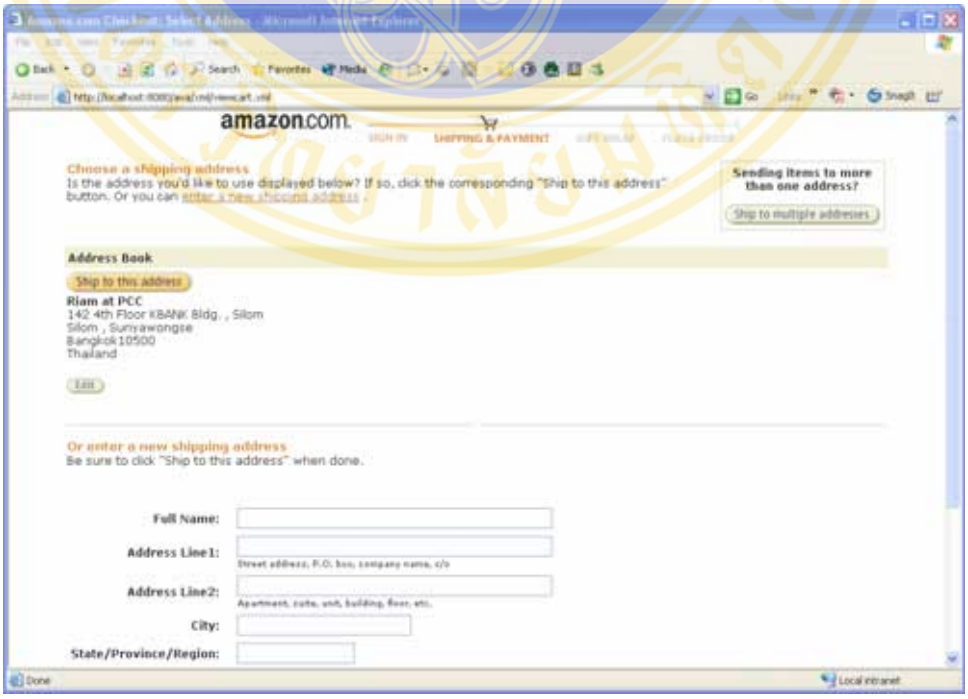


Figure 5-5 Choose a Shipping Address Screen

Copyright by Mahidol University

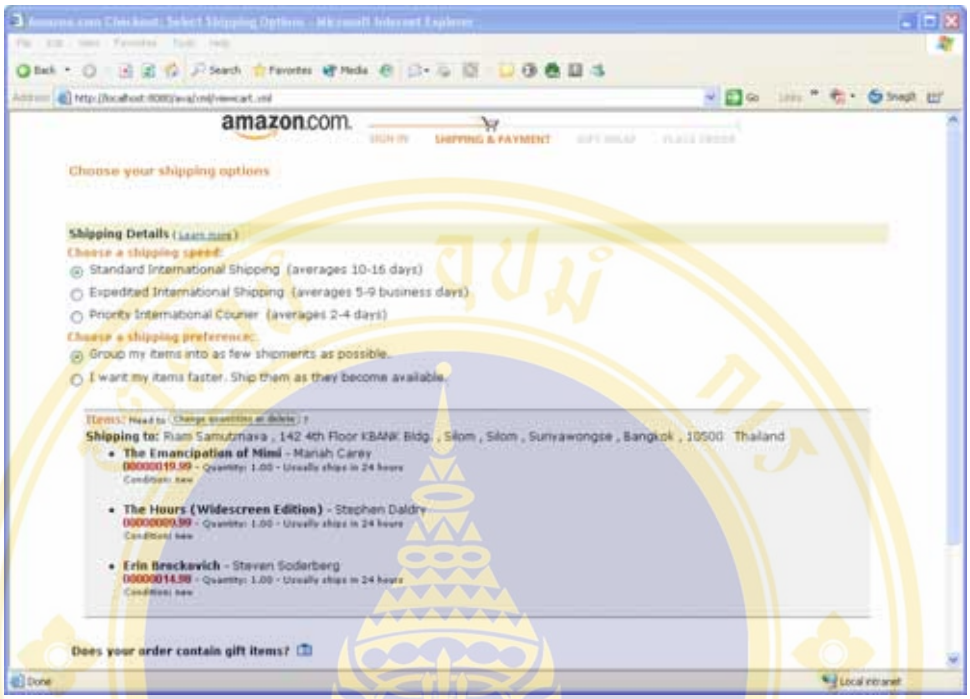


Figure 5-6 Choose a Shipping Option Screen

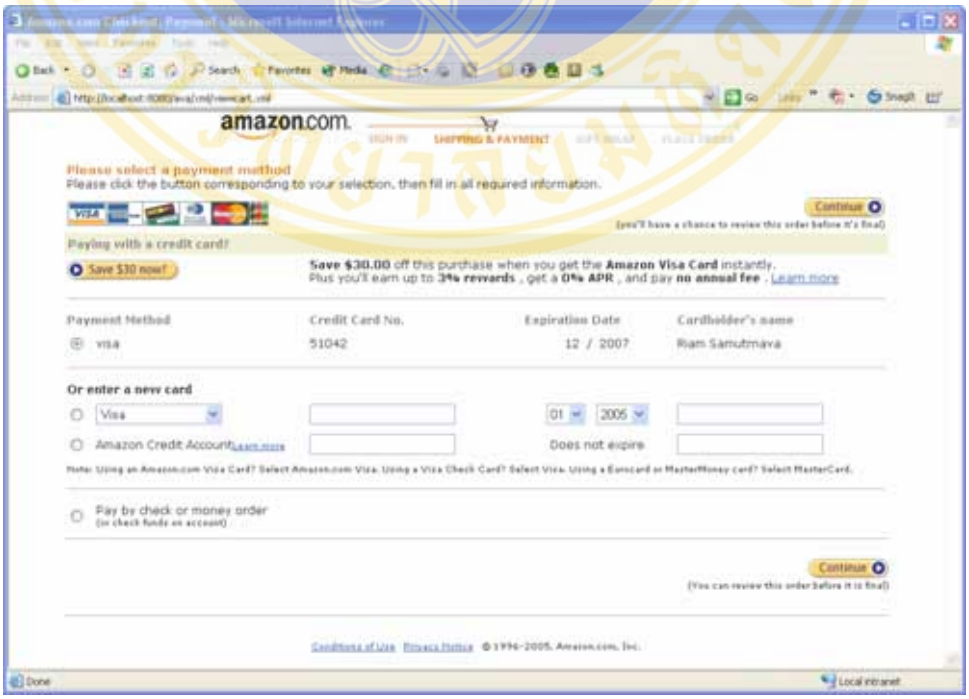


Figure 5-7 Select a Payment Method Screen

Copyright by Mahidol University

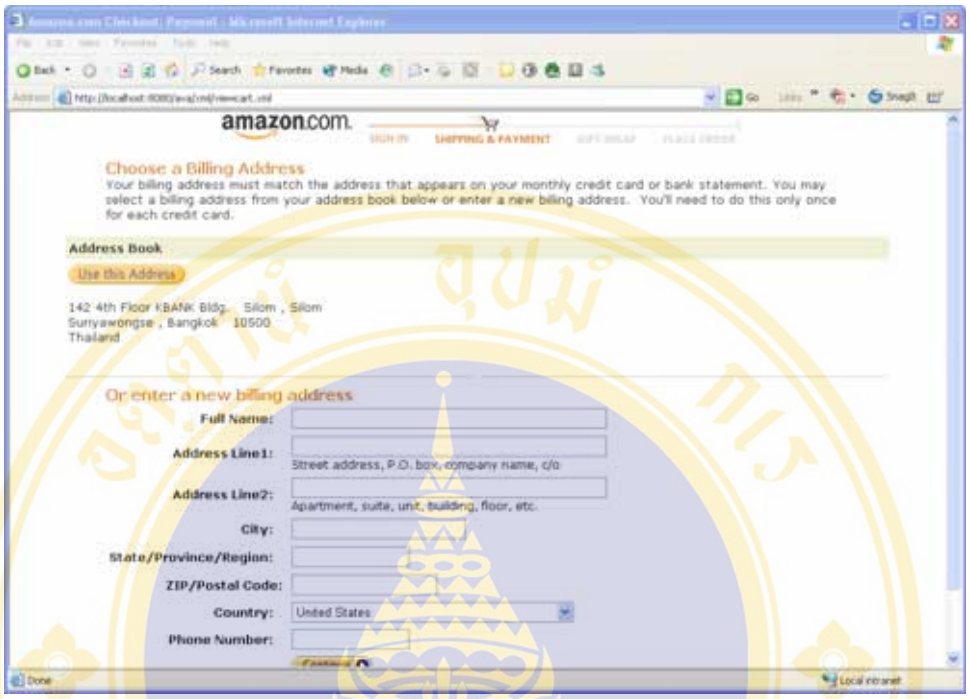


Figure 5-8 Choose a Billing Address Screen

Figure 5-9 Review Order Screen

Copyright by Mahidol University

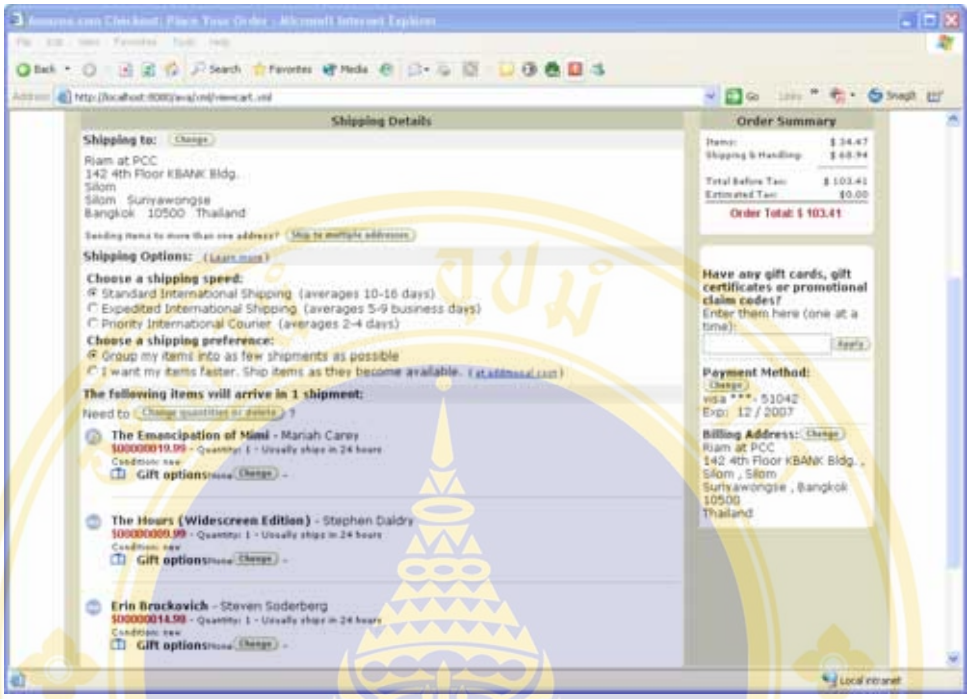


Figure 5-10 Review Order Screen (cont.)

5.3 Represent Ordering Process in the A^{2}V Framework

In this section, we will represent the ordering process in the A^{2}V framework. A web developer, who is responsible for a conversion scheme, takes an ordering flow and existing web pages as inputs and transforms them to the outputs, the implicit AVML documents. To transform the web pages, we reference the guideline of defining the A^{2}V knowledge in Section 3.8 of Chapter III as follows:

1) *Data Items Definition of Ordering Process*

This is to explore each existing web page and gather its data items as well as where to acquire. Also, it should categorize these items as a Global Data item, a User Profile, or a static item (which adheres to its original web page).

Copyright by Mahidol University

- Index web page

This web page is just a *dummy page* using for introducing other following web pages. The presence of index page is to behave a general web page. There is no item required in this page except the hyperlink linking to the next page.

- Sign-in web page named WP1

There is no Global Data item and no User Profile item in the sign-in page, but only static items as descriptive texts seen in Figure 5-3. These static items are then encoded in XSL notion.

- Cart viewing web page named WP2

This page will be reached if the sign-in name and password are correct. It requires user information (*User Profile item*), item(s) and its attributes which a user has already added to the cart (*Global Data item*) as shown in Table 5-1. Apart from that, there are either descriptive texts or graphical items (*static item*) as seen in Figure 5-4.

Table 5-1 Data Member of Cart Viewing Web Page

ID.	**Description**	General information of user given by a user on the first time registration
I1	**Category**	User Profile
	Source Type	DB
	Value	select cast(userid as char) "id", username, password, title, firstname, lastname, fullname, cast(type as char) "type", cast(age as char) "age", occupation, region from user_profile where username = ?
	More Parameters?	Yes
	Parameter	//username
ID.	**Description**	Items added to the cart already
I2	**Category**	Global
	Source Type	DB
	Value	select cast(c.id as char) "id", c.title, author, category, cast(price as char) "price", currency, cast(discount as char) as "discount", cast(quantity as char) as "quantity", date_format(latestadded, '%Y-%m-%d') "latestadded", unticked

Copyright by Mahidol University

		from cart a, order_line b, product c, user_profile d where b.order_id = a.order_id and b.id = c.id and a.userid = d.userid and d.username = ?
	More Parameters?	Yes
	Parameter	//username

Remark:

- ID.: An identifier of this item
- Description: What is this item for?
- Category: Select one from Global Data item and User Profile item
- Source Type: Select one from DB, Value, and Params
- Value: This is up to what a source is. It tells how to acquire an item from the source specified by the Source Type. All possible source types are mentioned in Section 4.2.2.2.3 of Chapter IV.
- More Parameters?: This tells whether more parameters are required according to the specified Value. If so, put Yes, otherwise, put No.
- Parameter: To bind item, this tells how to address an item in the Service Call Parameters document.

- Shipping address entering web page named WP3

This page contains general user information (*User Profile item*) recently listed on the previous page – we omit to repetitively define. Also, another user profile type is required, that is, shipping addresses (*User Profile item*), listed on Table 5-2, which have been given on the earlier visits of a user. The shipping address information was stored in a productive database. Besides, there are such labels, HTML form in the case of that a user has more shipping addresses, and graphical items (*static item*) as shown in Figure 5-5.

- Shipping option choosing web page named WP4

This page has items in the cart (*Global Data item*) as noted in Table 5-1, selected shipping address (*User Profile item*), and also descriptive labels and graphical items as seen in Figure 5-6 (*static item*).

Copyright by Mahidol University

Table 5-2 Data Member of Shipping Address Entering Web Page

ID.	**Description**	Shipping address information given by user on the previous visits
I3	**Category**	User Profile
	Source Type	DB
	Value	select cast(a.addrid as char) "id", c.recipient, c.addrno, c.road, c.subdistrict, c.district, c.city, c.country, c.postcode, 'false' as "selected" from shipping_address a, user_profile b, address c where a.userid = b.userid and a.addrid = c.addrid and b.username = ?
	More Parameters	Yes
	Parameter	//username

- Payment method choosing web page named WP5

This page is composed of the payment method that has been chosen in the past visits of the user – assume that the user had paid by a credit card (as shown in Table 5-3) whose information was stored in the database (*User Profile item*). Still, there are labels, HTML form in case of that the user prefers other methods provided to the last selected one, and graphical items (*static item*) as seen in Figure 5-7.

Table 5-3 Data Members of Payment Method Choosing Web Page

ID.	**Description**	Credit Card information given by user on the previous visits
I4	**Category**	User Profile
	Source Type	DB
	Value	select a.cardno "id", a.cardno, a.cardtype, a.expirymonth, a.expiryyear, a.cardholder, 'false' as "selected" from payment_method a, user_profile b where a.userid = b.userid and b.username = ?
	More Parameters	Yes
	Parameter	//username

- Billing address entering web page named WP6

This page contains the billing address that has been registered in the past visits of a user (as shown in Table 5-4) whose information was stored in a database (*User*

Copyright by Mahidol University

Profile item). Still, there are labels, HTML form in the case of that a user has more billing addresses and graphical items (*static item*) as seen in Figure 5-8.

Table 5-4 Data Member of Billing Address Choosing Web Page

ID.	**Description**	Billing address information given by user at the first time registration
I5	**Category**	User Profile
	Source Type	DB
	Value	select cast(a.addrid as char) "id", c.recipient, c.addrno, c.road, c.subdistrict, c.district, c.city, c.country, c.postcode, 'false' as "selected" from billing_address a, user_profile b, address c where a.userid = b.userid and a.addrid = c.addrid and b.username = ?
	More Parameters	Yes
	Parameter	//username

- Order reviewing web page named WP7

This page is the last page of the ordering flow which has the information specified in the previous pages that are user information, the shipping address, the shipping option, the payment method, and the billing address (*User Profile item*.) Also, there are the added items (*Global Data item*), labels and graphical items (*static item*) as seen in Figure 5-9 and Figure 5-10 respectively.

2) *Active Rules Forming of Ordering Process*

When acquiring data items required inside each web page, we first have to form up Active Rules that represent each web page mentioned. As the activity diagram shown in Figure 5-1, we first traverse its root node by keeping the following rule in mind *"For each web page found, we convert it to be represented by a Local Active Rule. For each decision point, we insert a new Remote Active Rule."* Hence, we get *seven Local Active Rules* and *two Remote Active Rules* regardless of the index page.

Then, we build Local Active Rules. At least, it needs to have an onLoad Active Rule descended from the web page next to a decision point of the tree diagram. A condition-action part of a Local Active Rule is derived from the transformation of the

Copyright by Mahidol University

source code of web page to be expressed in XSL instruction. A data binding part of a Local Active Rule comes from representing items required in this rule (in this page) in XLink syntax [20, 25].

In the end, we build a data binding part of a Remote Active Rule by listing all the items passed to the AVML Processor for an execution of a Remote Active Rule. Also, its rule body part is from having a conditional term as a condition part and implicit AVML documents corresponding to the results of condition evaluation as an action part.

We give a brief summary of a Local Active Rule and a Remote Active Rule attributes reflecting the ordering process in Table 5-5 and Table **5-6** respectively. For a condition-action part of a Local Active Rule, it is encoded in XSL being converted from original source codes of each web page.

Table 5-5 Local Active Rules Representing the Ordering Process

Web Page	Rule ID.	What presents in the Data Binding part	What presents in the Condition-Action part
WP1	L1	-	XSL producing the view of sign-in page
WP2	L2	- User information (I1) - Items in the cart and their details (I2)	XSL producing the view of cart viewing page
WP3	L3	- User information (I1) - Shipping address (I3)	XSL producing the view of shipping address entering page
WP4	L4	- Shipping address (I3) - Items in the cart and their details (I2)	XSL producing the view of shipping option choosing page
WP5	L5	- Payment method (Credit Card information) (I4)	XSL producing the view of payment method choosing page
WP6	L6	- Billing address (I5)	XSL producing the view of billing address entering page
WP7	L7	- User information (I1) - Shipping address chosen - Shipping option chosen - Payment method chosen - Billing address chosen - Items in the cart and their details (I2)	XSL producing the view of order reviewing page

Copyright by Mahidol University

Table 5-6 Remote Active Rules Representing the Ordering Process

Rule ID.	Data Binding Part	Rule Body Part	
		Condition Part	**Action Part**
R1	-	Not available.	IA1 will be returned.
R2	-	Check if the sign-in name and password are correct by validating against the existing ones in a database	If the condition evaluates true, IA2 will be returned; Else the condition evaluates false, IA3 will be returned.

In Table 5-6, for the sake of simplicity, we put IAx, where IA stands for an **i**mplicit **A**VML document and **x** is a running number in an action part column. We are going to depict the Web Contents items, Local Active Rules, and Remote Active Rules of the derived implicit AVML documents accordingly.

3) *Implicit AVML Document Forming of Ordering Process*

This is to think of which a Web Contents item, a Local Active Rule, and a Remote Active Rule should be mapped into each implicit AVML document declared in the previous section. We call the mapping idea mentioned in Section 3.8 of Chapter III to mind. Thus, let us see the results in Table 5-7.

Noticeably, there is likely *less* Web Contents item required in the succeeded implicit AVML documents– for example, IA3 – despite of these items still in need for them, this is because in the implicit AVML document IA2, most items, gathered up there, have been already stored in the Global Data file and the User Profile file in a user's machine at a client side. They are retrievable for being used in IA3. In fact, it could logically be said that if the implicit AVML documents early in the flow (referring to the diagram in Figure 5-1) included the items that could be called in the subsequent implicit AVML documents, it would need not remain these items in the following implicit AVML document repetitively.

Copyright by Mahidol University

Table 5-7 Implicit AVML Documents Representing the Ordering Process

IA ID.	Web Contents Section		Active Rules Section	
	Global Data (Item ID.)	**User Profile (Item ID.)**	**Local (Rule ID.)**	**Remote (Rule ID.)**
IA1	-	-	**L1**	R2
IA2	I2	I1, I3, I4, I5	**L2**, L3, L4, L5, L6, L7	-
IA3	-	-	**L1**	R2

Furthermore, the Rule IDs. of Local Active Rules in boldface appearing on Table 5-7 mean the onLoad Active Rules of each implicit AVML document.

Copyright by Mahidol University

CHAPTER VI
EXPERIMENTS

This chapter presents a proof-of-concept of the A^2V framework by using our A^2V prototype. Due to the difficulties in acquiring and setting up the existing Active View approaches as reviewed in Chapter II, we can only performed *hands-on experiments* on testing the estimated performance of our A^2V framework against traditional web applications development approaches. Nevertheless, we also provided a *critical analysis* for verifying the performance of the A^2V framework against the centralized model of the existing Active View approaches. Finally, we also provide the summary of both the hands-on experiments and the critical analysis at the end of this chapter.

6.1 Experimental Objectives

- To demonstrate that the A^2V framework, which is based on the principle of decentralized control, has the better performance of Active View construction process for e-service applications than that of other existing Active View approaches based on the principle of centralized control.
- To measure the performance of the A^2V framework against traditional web applications development approaches in order to proof the efficiency of using Active Views constructed by our A^2V framework over the traditional approaches.

6.2 Experimental Methods

Due to the difficulties in acquiring and setting up existing Active View approaches as mentioned earlier, we conduct our experiments in two aspects:

Copyright by Mahidol University

1) Critical Analysis of the A^2V Framework against the Existing Active View Construction Approaches

2) Performance Measurement of the A^2V Framework against the Traditional Web Applications Development Approaches

For both hands-on experiments and critical analysis, we use response time to determine and compare the performance of Active View construction of different Active View approaches, including our A^2V framework. We define *the response time as the duration from when a request is sent by a web browser to when the last byte of Active View content is received by a web browser*. Thus, *the response time includes the processing time of relevant client side components and server side components and the communication time between all relevant components.*

6.3 Critical Analysis of A^2V Performance against Existing Active View Approaches

This section provides a proof of our idea that an improvement of response time for constructing an Active View will be achieved if some communication processes between clients and servers are eliminated.

Figure 6-1 depicts all possible communication times and processing times that occur in a centralized control framework of the existing Active View approaches as reviewed in Chapter II, while Figure 6-2 depicts all possible communication times and processing times that occur in the A^2V framework.

From Figure 6-1, given that

- c_{i1} is the **c**ommunication time, in seconds, consumed for an **i**nbound communication over the Internet when a web browser submits a request to an Active Rule engine;

- c_{o1} is the **c**ommunication time, in seconds, consumed for an **o**utbound communication over the Internet when a web browser receives a response Active View from an Active Rule engine;

Copyright by Mahidol University

- c_{i2} is the **c**ommunication time, in seconds, consumed for an **i**nbound communication within a server side when an Active Rule engine submits a request to a web server;

- c_{o2} is the **c**ommunication time, in seconds, consumed for an **o**utbound communication within a server side when an Active Rule engine receives a response web page from a web server;

- p_{ar} is the **p**rocessing time, in seconds, consumed for all processes done by an **A**ctive **R**ule engine;

- p_{ws} is the **p**rocessing time, in seconds, consumed for all processes done by a **w**eb **s**erver.

From Figure 6-2, given that

- c_{i3} is the **c**ommunication time, in seconds, consumed shortly for an **i**nbound communication within a user's machine when a web browser submits a request to the AV Client;

- c_{o3} is the **c**ommunication time, in seconds, consumed shortly for an **o**utbound communication within a user's machine when a web browser receives a response Active View from the AV Client;

- c_{i4} is the **c**ommunication time, in seconds, consumed for an **i**nbound communication over the Internet when the AV Client submits a request to a web application;

- c_{o4} is the **c**ommunication time, in seconds, consumed for an **o**utbound communication over the Internet when the AV Client receives a response web page from a web server;

- c_{i5} is the **c**ommunication time, in seconds, consumed for an **i**nbound communication over the Internet when the AV Client invokes a service call to the AVML Processor;

- c_{o5} is the **c**ommunication time, in seconds, consumed for an **o**utbound communication over the Internet when the AV Client receives a response Active Web Page from the AVML Processor.

- p_{ap} is the **p**rocessing time, in seconds, consumed for all processes done by the **A**VML **P**rocessor;

Copyright by Mahidol University

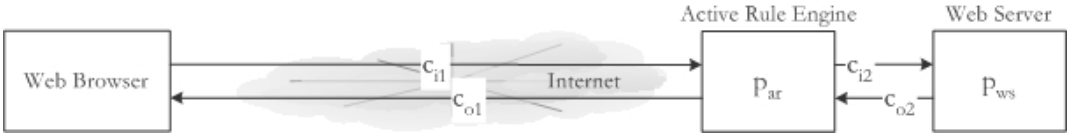


Figure 6-1 Conceptual Structure of Existing Active View Approaches

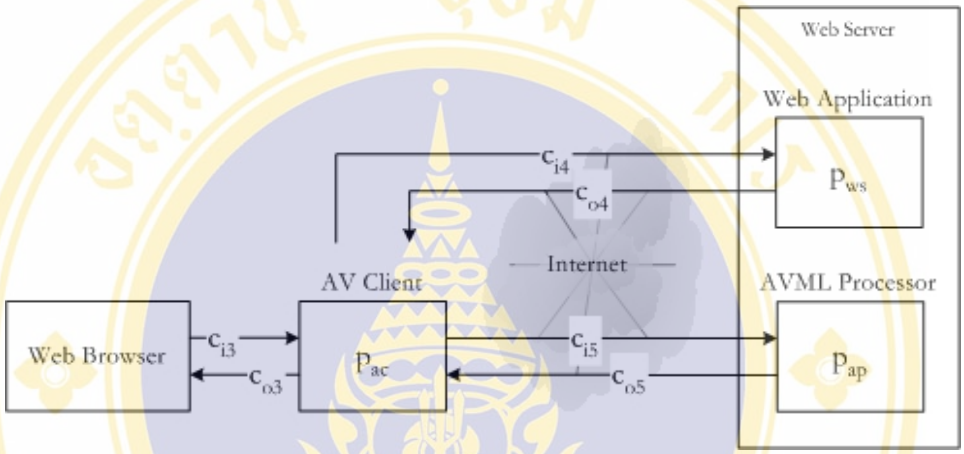


Figure 6-2 Conceptual Structure of the A^2V Framework

Based on the configuration as seen in Figure 6-1 and Figure 6-2, we can derive the following equations.

$$c_{i3} \rightarrow 0 \quad \text{.....(1)}$$
$$c_{o3} \rightarrow 0 \quad \text{.....(2)}$$
$$c_{i1} + c_{i2} \geq c_{i3} + c_{i4} \quad \text{.....(3)}$$
$$c_{o1} + c_{o2} \geq c_{o3} + c_{o4} \quad \text{.....(4)}$$
$$c_{i1} + c_{i2} \geq c_{i3} + c_{i5} \quad \text{.....(5)}$$
$$c_{o1} + c_{o2} \geq c_{o3} + c_{o5} \quad \text{.....(6)}$$
$$p_{ar} + p_{ws} \geq p_{ac} + p_{ws} \quad \text{.....(7)}$$
$$p_{ar} + p_{ws} \geq p_{ac} + p_{ap} \quad \text{.....(8)}$$

The equations (1) and (2) represent the fact that both c_{i3} and c_{o3} approach zero because they represent the communication time between the processes that reside on the same machine.

Copyright by Mahidol University

The equations (3), (4), (5), (6) are hold between Figure 6-1 and Figure 6-2 because of the following facts. In the A^2V framework, the communication between the AV Client at a client side and the server side components can be considered as one hop due to the facts from the equations (1) and (2). On the other hand, the communication between a client and a web server in the existing Active View approaches can be either one hop or two hops, depending on whether their Active Rule engines will be locally tied together with the web server in an internal network. If the number of communication hops of A^2V is less than those of the existing Active View approaches, the A^2V will definitely yield the better performance than the existing Active View approaches. If the number of communication hops of A^2V is equal to those of the existing Active View approaches, the A^2V will still give the better performance. This is because, in the existing Active View approaches, their server side components have to handle the number of all coming user sessions alone, while, in the A^2V framework, some user sessions can be remotely and completely handled by a client side component, the AV Client. Thus, the equation (3), (4), (5), and (6) are held.

While the afore-mentioned equations (1), (2), (3), (4), (5), and (6) are considered from the *communication times*, consumed amongst the relevant components, which are one variable of the formulated response time, the equations (7) and (8) are considered from the *processing times* consumed by the relevant components.

The equation (7) is hold between Figure 6-1 and Figure 6-2 because, in existing Active View approaches, there is the process on an Active Rule engine which checks whether any Active Rules for the requested web page are triggered for all time that it is requested by a user. In contrast to the existing Active View approaches, in the A^2V framework, the AV Client treats differently between a request requiring a generic web page/a graphical file and a request requiring to process an Active Rule. Hence, the AV Client needs not always check the Active Rule which is potentially to be triggered. However, in existing Active View approaches, if an Active Rule engine found that there was no Active Rule to be triggered after checking, the processing time by p_{ar} and p_{ws} would be approximately equal to that of by p_{ac} and p_{ws}. In summary, we can conclude that the processing time by p_{ar} and p_{ws} is greater than or equal to that of by p_{ac} and p_{ws}.

Copyright by Mahidol University

The equation (8) is hold between Figure 6-1 and Figure 6-2 because, in existing Active View approaches, an execution of Active Rules always involves both a web server and an Active Rule engine. Contrary to the existing Active View approaches, in the A^2V framework, the AV Client handles differently between an execution of Local Active Rules and an execution of Remote Active Rules. To execute Remote Active Rules, the AV Client has to always collaborate with the AVML Processor to carry out the execution process, whereas to execute Local Active Rules, the AV Client can totally complete the execution process itself. In other words, the processing time p_{ap} will be included in the equation (8) in the case that Local Active Rules are triggered. Thus, if user requests mostly trigger Local Active Rules, the processing time by p_{ar} and p_{ws} would be approximately greater than that of by p_{ac} and p_{ap} because p_{ap} is always zero. Otherwise, user requests mostly trigger Remote Active Rules, the processing time by p_{ar} and p_{ws} would be approximately equal to that of by p_{ac} and p_{ap}. In summary, we can logically conclude that the processing time by p_{ar} and p_{ws} is greater than or equal to that of by p_{ac} and p_{ap}.

Note that the equation $p_{ar} + p_{ws} \geq p_{ac} + p_{ap} + p_{ws}$ is not held in the A^2V framework according to the architectural and functional details of two AVML processing units which have been provided in Section 3.5 and Section 3.7 in Chapter III. Hence, the equations (7) and (8) will independently and separately occur from each other.

To compare the performance of the A^2V framework against the existing Active View approaches, consider the following cases:

Case I – When all Active Views, ***n***, are executed at a server side

The response time, t_1, of centralized construction process of existing Active View approaches is

$$t_1 = n(c_{i1} + p_{ar} + c_{i2} + p_{ws} + c_{o2} + c_{o1}) \quad \text{.....(9)}$$

Where n is the number of Active Views to be constructed.

Copyright by Mahidol University

For the A^2V framework, the response time for executing all Active Views at a server side depends on the types of request as discussed in Chapter III. Suppose that *i* is the number of requests for a typical web pages, *j* is the number of requests for remote rules execution at the AVML Processor, and $i + j = n$. Therefore the response time, t_2, of the A^2V framework is

$$t_2 = j(c_{i3} + p_{ac} + c_{i4} + p_{ws} + c_{o4} + c_{o3}) + i(c_{i3} + p_{ac} + c_{i5} + p_{ap} + c_{o5} + c_{o3})$$

From the assumptions (1) and (2), therefore

$$t_2 = j(p_{ac} + c_{i4} + p_{ws} + c_{o4}) + i(p_{ac} + c_{i5} + p_{ap} + c_{o5}) \text{.....(10)}$$

From the formulas (9) and (10), the difference between t_1 and t_2 for the case I is

$$t_1 - t_2 = n(c_{i1} + p_{ar} + c_{i2} + p_{ws} + c_{o2} + c_{o1}) - j(p_{ac} + c_{i4} + p_{ws} + c_{o4}) - i(p_{ac} + c_{i5} + p_{ap} + c_{o5})$$

From the assumptions (3), (4), (7), and (8), so $t_1 - t_2 \geq 0$.....(11)

From (11), we can conclude that when all Active Views, *n*, are executed at a server side, the response time of the existing Active View approaches is approximately longer than that of the A^2V framework. More importantly, we can obviously see from the equations that the more number, *n*, of Active Views that we want to generate, the longer the response time of the existing Active View approaches will be than that of the A^2V framework. In other words, the differences between the response time of the existing Active View approaches and that of the A^2V framework is varied according to the value of *n*.

Copyright by Mahidol University

Case II – When some Active Views, ***m***, can be distributed to be remotely constructed by the AV Client at a client side, and some Active Views, ***n***, are constructed by a collaboration of the AV Client at a client side and the AVML Processor at a server side

Since in this case all Active Views of the existing Active View approaches are still required to be executed at a server side, the response time, $t_{3,}$ of centralized construction process of existing Active View approaches in this case is

$$t_3 = n(c_{i1} + p_{ar} + c_{i2} + p_{ws} + c_{o2} + c_{o1}) + m(c_{i1} + p_{ar} + c_{i2} + p_{ws} + c_{o2} + c_{o1})$$
$$= (n+m)(c_{i1} + p_{ar} + c_{i2} + p_{ws} + c_{o2} + c_{o1}) \text{.....(12)}$$

Whereas, the response time, $t_{4,}$ of distributed construction process of the A^2V framework in this case is

$$t_4 = n(c_{i3} + p_{ac} + c_{i5} + p_{ap} + c_{o5} + c_{o3}) + m(c_{i3} + p_{ac} + c_{o3})$$

From the assumptions (1) and (2), we can derive that

$$t_4 = n(p_{ac} + c_{i5} + p_{ap} + c_{o5}) + m(p_{ac}) \text{.....(13)}$$

From the formulas (12) and (13), the difference between t_3 and t_4 for the case II is

$$t_3 - t_4 = (n+m)(c_{i1} + p_{ar} + c_{i2} + p_{ws} + c_{o2} + c_{o1})$$
$$- n(p_{ac} + c_{i5} + p_{ap} + c_{o5}) - m(p_{ac})$$

From the assumptions (5), (6), (8), and if m > n, we can derive that

$$t_3 - t_4 \geq 0 \text{.....(14)}$$

Similar to case I, from (14), we can conclude that when some Active Views, *m*, can be distributed to be remotely constructed by the AV Client at a client side, and some Active Views, *n*, are constructed by a collaboration of the AV Client at a client

Copyright by Mahidol University

side and the AVML Processor at a server side, the response time of the existing Active View approaches is approximately longer than that of the A^2V framework, especially when the proportion of *m* is greater than *n*. The higher m is, the more noticeable the difference between t_3 and t_4 is. If n = 0, this means all Active Views are able to be distributed to be totally constructed at a client side. Therefore, there will be no communication between clients and servers.

In summary, from (11) and (14), we conclude that the decreasing of the response time resulted from the A^2V framework is in direct proportion to the number of Active Views which can be remotely constructed by the AV Client at a client side. This is because the Active Views can be completely constructed by the AV Client, eliminating the communication time to a server side. More obviously, for the case II, the response time by existing Active View approaches is approximately higher than that of the A^2V framework when Active Views are remotely constructed at a client side. This is because, in the A^2V framework, there is no communication time calculated in the formula, but only p_{ac} is considered. As a result, we conclude that the performance of an Active View construction by the A^2V framework is on average better than that of existing Active View approaches because some Active Views can be totally constructed by the AV Client at a client side.

6.4 Performance Measurement of A^2V

To demonstrate the efficiency of using the A^2V framework, this section explains how we have tested the use of the A^2V in comparison to traditional web applications. Apart from that, we have tested the use of the AV Client in processing multiple user requests concurrently. In so doing, we have implemented a demonstrated case study as explained in Chapter V and run the case study.

Based on the demonstrated case study provided in Chapter V, we conduct a set of hands-on experiments by using our A^2V prototype and traditional web applications synchronously. We set up the experiments in different experimental environments to measure which one yields the more efficient results.

Copyright by Mahidol University

The demonstrated case study contains seven Active Views – the signing-in page, the cart viewing view, the shipping address entering view, the shipping option choosing view, the payment method choosing view, the billing address entering view, and the order reviewing view. In our experiments, we choose to test on two scenarios. The first is when a user signs in to use the demonstrated web application at the first time and the second is when a user enters his/her user name and password to access cart viewing page. This is to compare the performance of different Active View approaches in two facets. The former scenario requires that all Active Views are constructed at a server side, while the later requires that some Active Views can be distributed to be constructed by the AV Client at a client side and some Active Views are collaboratively constructed by the AV Processor at a server side and the AV Client at a client side. The response time is calculated on the sum of accumulated time consumed for each web page in the flow of all concurrent sessions of each model.

Also, we also conduct the experiments by having a single AV Client process multiple user requests concurrently at a time on a single machine to measure the response time and the memory consumed and evaluate a usability of the AV Client toward multiple accessing to web pages in real scenarios.

In our performance measurement, we apply a web application performance testing tool named WAPT [36] to measure the response times of our experiments in different experimental environments. This is to avoid human error in measuring the response time manually and synchronously for both experiments on our A^2V prototype and traditional web applications. The details of how we use WAPT to run the demonstrated case study in our experiments will be discussed further in Section 6.4.2.

6.4.1 Test Environment

We set up various experimental environments as we would like to see the various result. Therefore, we could determine which one yields the effective performance.

Copyright by Mahidol University

1) **Server Specification**:

- CPU: Intel Centrino Mobile 1.6 GHz
- Main Memory: 512MB
- Hard Disk Space: 400-500 MB
- Operating System: Microsoft Windows XP Professional SP1

2) **Client Specification**:

- For duo machines testing

- CPU: Mobile Intel Pentium 4 – M 1.70GHz
- Minimum 256MB Main Memory
- Minimum Hard Disk Space: 200-300 MB
- Network Card 10/100 Mbps
- Operating System: Microsoft Windows XP Professional SP1

- For multiple machines testing

- CPU: ranged from x86 Family 6 Model 7 minimum to Intel Pentium 4 3.60GHz
- Minimum 164MB – 512MB Main Memory
- Minimum Hard Disk Space: 100-200 MB
- Network Card 10/100 Mbps
- Operating System: MS Windows 2000 Professional/Server, MS Windows XP Professional SP1

Copyright by Mahidol University

6.4.2 Log Capture Implementation

To compare our approach with traditional web applications development approaches in term of a performance in constructing an Active View, we concern their response time. This sub-section describes how to WAPT helps run the demonstrated case study synchronously for both experiments on our A^2V prototype and traditional web applications and how to acquire the log files to extract the response time from them after running the experiment so that we have them compared and evaluated.

WAPT is a client-side component and has to be installed on the computer that represents a client side. In practice, there has to be two WAPT programs running on the client machine, but their port numbers are set differently. The port number of one WAPT is set to the listening port number of the A^2V prototype and the other one's port number is set to the Log Capturer's listening port number in traditional web applications. Also, the sequence of a request for web page and the parameters required for the transition of web pages must have already been prepared for each particular WAPT. Both have been scheduled to start browsing the first web page of the flow simultaneously.

Thus, there are at least two concurrent sessions per a client machine – one is WAPT for the A^2V and the other one is WAPT for the Log Capturer of traditional web applications. Once WAPTs had finished running the demonstrated case study, we collect the log files of either A^2V prototype or the Log Capturer and then run the log extractor to get the output in the MS Excel format for further analyses.

Thereafter, we gradually increase by one concurrent session up to twenty sessions and also keep the Excel-formatted output for each increment. As mentioned above, the two concurrent sessions of two approaches generally requires two WAPTs, we need to run two more WAPTs for each new session increased.

To acquire the transactional log files from traditional web applications, we insert the Log Capturer into a client side as shown in Figure 6-3. It is an additional component which is a Java-based application that we newly developed for capturing an interaction between a user's web browser and a web server. It does such a small debugger.

Copyright by Mahidol University

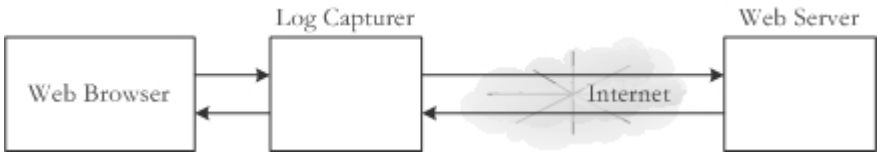


Figure 6-3 Conceptual Structure of Traditional Web Applications Development Approaches

For an implementation of the A^2V prototype, we employ the log4j API [37, 38, 39] to capture the runtime information of the AVML processing units. We just place the debugging statement like *debug.print(…)* at the beginning and the end of the AV Client operation (or when the new thread is newly created for handling the *in* connection and the *out* connection is at last written to the client user's web browser).

Whilst we is running the demonstrated case study, the runtime details of the A^2V prototype and traditional web applications are written to the log file. Once halting, we have timestamps of the beginning-process and the ending-process already there. It then needs to get the time consumed for each incoming connection by finding the difference between the ending time and the beginning time from the log file – here it is called a *log capture*. However, more simply and conveniently, we developed another Java-based application, called a *log extractor*, featuring Excel Java API [40] dealing with this regard. The captured output will be appeared on the Microsoft Excel file (.xls file) for further calculation and analysis. This step is called a *log analysis*.

Essentially, it is noted that the handling of a graphical file request is beyond our scope. Our log content is only interested in a general web page and Active Web Page retrieval.

6.4.3 Experimental Results

Table 6-1, Table 6-2, and Table 6-3 show the accumulated response times of all web pages in the flow of all concurrent sessions for each model in different aggregated views – the minimum result, the maximum result, and the average result – to complete all web pages in the flow of the case study in the different test environments. All records of resulting accumulated response times from the

Copyright by Mahidol University

experiments of the A^{2}V prototype and traditional web applications grouped by the test environments above are provided in Appendix C.

Table 6-4 shows the average accumulated response times and the memory consumed of a single AV Client in concurrently processing multiple user requests (up to twenty) on a single machine.

Table 6-1 Comparison of Cumulative Response Time on Single Machine between A^{2}V and Traditional Web Applications Development Approaches

No. of concurrent session(s)	A^{2}V			Traditional Web Applications Development Approaches		
	Min. accum. response time (sec.)	Max. accum. response time (sec.)	Avg. accum. response time (sec.)	Min. accum. response time (sec.)	Max. accum. response time (sec.)	Avg. accum. response time (sec.)
1	5.969	5.969	5.969	145.097	145.097	145.097
2	8.353	8.524	8.4385	143.998	144.478	144.238
4	9.785	12.778	11.4393	144.295	144.698	144.5098
5	11.387	14.8	13.2476	144.589	145.57	145.0568
6	10.846	19.058	16.559	144.276	145.43	144.8183
8	16.802	27.059	23.3871	44.161	146.12	110.0066
10	21.331	33.969	28.5207	1.362	146.22	84.239
12	20.589	39.927	32.964	2.134	147.054	64.38308
14	32.647	87.747	48.7364	3.026	147.024	54.62614
15	25.377	69.699	43.9038	3.275	146.062	47.95833
16	-	-	-	5.177	145.96	45.06156
18	-	-	-	6.849	158.338	44.21668
20	-	-	-	8.863	146.23	38.95245

Summary of the Results on a Single Machine

In Table 6-1, some entries in the A^{2}V column have no value (the ones with dash) due to incomplete executions. Note that only the successful responses would be used for our critical analysis. Nevertheless, this phenomenon is explainable and the reason behind this will be further discussed in the next section.

The average accumulated response times of the A^{2}V framework and traditional web applications development approaches on *a single machine* in Table 6-1 are represented in a graph in Figure 6-4. From the graph in Figure 6-4, a horizontal axis

Copyright by Mahidol University

stands for the number of concurrent sessions running and a vertical axis stands for the average time, in seconds, consumed for browsing the same web pages. From the graph, there are two lines plotted against each other – one with diamond shapes resulted from traditional web applications development approaches and the other one with rectangle shapes resulted from the A^2V prototype.

From the graph, the A^2V line appears to be shorter than the other one and tends to be higher. The average accumulated response times of the traditional web applications development approaches from one to six concurrent sessions seem high and tend to be lower thenceforth. It will be applicable if the numbers of concurrent sessions per a client side vary from one to fifteen sessions. The results by the concurrent sessions beyond this number are quite meaningless. By overall, the average accumulated response times of the A^2V framework on a single machine are approximately better than that of traditional web applications development approaches although they are compared up to only fifteen concurrent sessions.

Copyright by Mahidol University

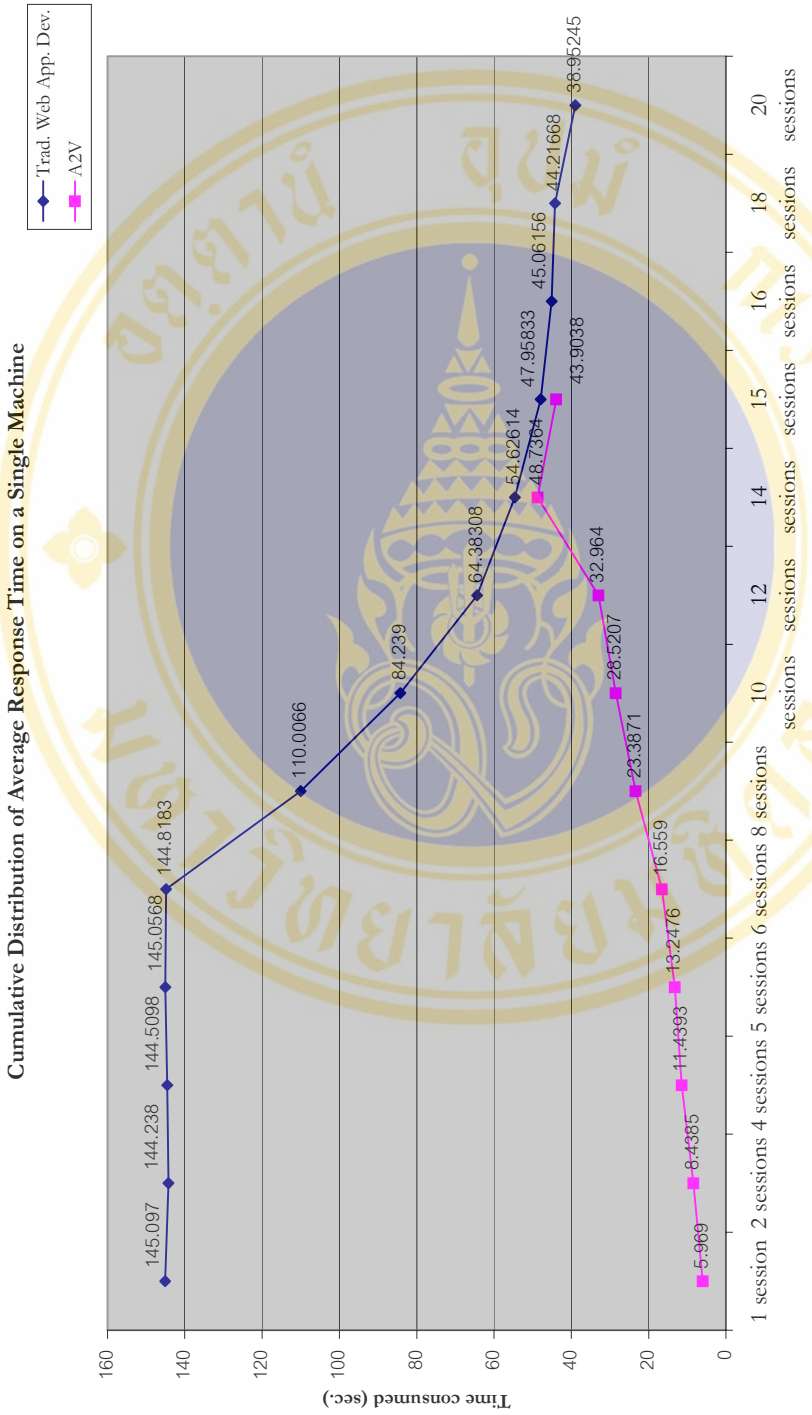


Figure 6-4 Comparison Chart of Average Response Time on a Single Machine between A^{2}V and Traditional Web Applications Development Approaches

Copyright by Mahidol University

Table 6-2 Comparison of Cumulative Response Time on Duo Machines between A^2V and Traditional Web Applications Development Approaches

No. of concurrent session(s)	A^2V			Traditional Web Applications Development Approaches		
	Min. accum. response time (sec.)	Max. accum. response time (sec.)	Avg. accum. response time (sec.)	Min. accum. response time (sec.)	Max. accum. response time (sec.)	Avg. accum. response time (sec.)
1	6.079	6.079	6.079	144.769	144.769	144.769
2	7.741	13.355	10.548	145.188	145.209	144.989
4	12.588	22.112	18.2083	144.859	144.978	144.921
5	23.263	24.785	24.0926	144.939	145.138	145.047
6	17.706	31.997	27.8522	144.899	145.21	145.056
8	26.587	78.002	53.473	1.543	145.397	106.796
10	34.66	245.338	126.576	1.353	145.529	82.2704
12	118.06	336.589	224.527	1.343	145.797	69.0383
14	-	-	-	44.172	106.633	53.9071
15	-	-	-	2.934	144.939	50.9215
16	-	-	-	1.812	87.266	48.0474
18	-	-	-	4.977	144.948	58.5392
20	-	-	-	50.394	153.782	94.7614

Summary of the Results on Duo Machines

Similar to Table 6-1, some entries in Table 6-2 have no value. This is because of the incomplete execution as described earlier, so, like the results on a single machine, the length of the A^2V line happens to be shorter than the other one.

The average accumulated response times of the A^2V framework against traditional web applications development approaches on *duo machines* in Table 6-2 are represented in a graph in Figure 6-5.

Similar to a single machine scenario, in Figure 6-5, the average accumulated response times of the traditional web applications development approaches from one to six concurrent sessions seem high and tend to be lower afterward though.

The results of the A^2V framework tend to be higher than traditional web applications development approaches when there are more concurrent sessions. Especially, for the nine concurrent sessions up above, the average cumulative response times of the A^2V framework is far longer than that of traditional web applications

Copyright by Mahidol University

development approaches. From the graph, it will be applicable if the numbers of concurrent sessions per a client side vary from one to nine sessions. Even so, when comparing the concurrent sessions between one up to nine around, we can summarize that the average accumulated response times of the A^2V framework on duo machines are still approximately better than that of traditional web applications development approaches.

Copyright by Mahidol University

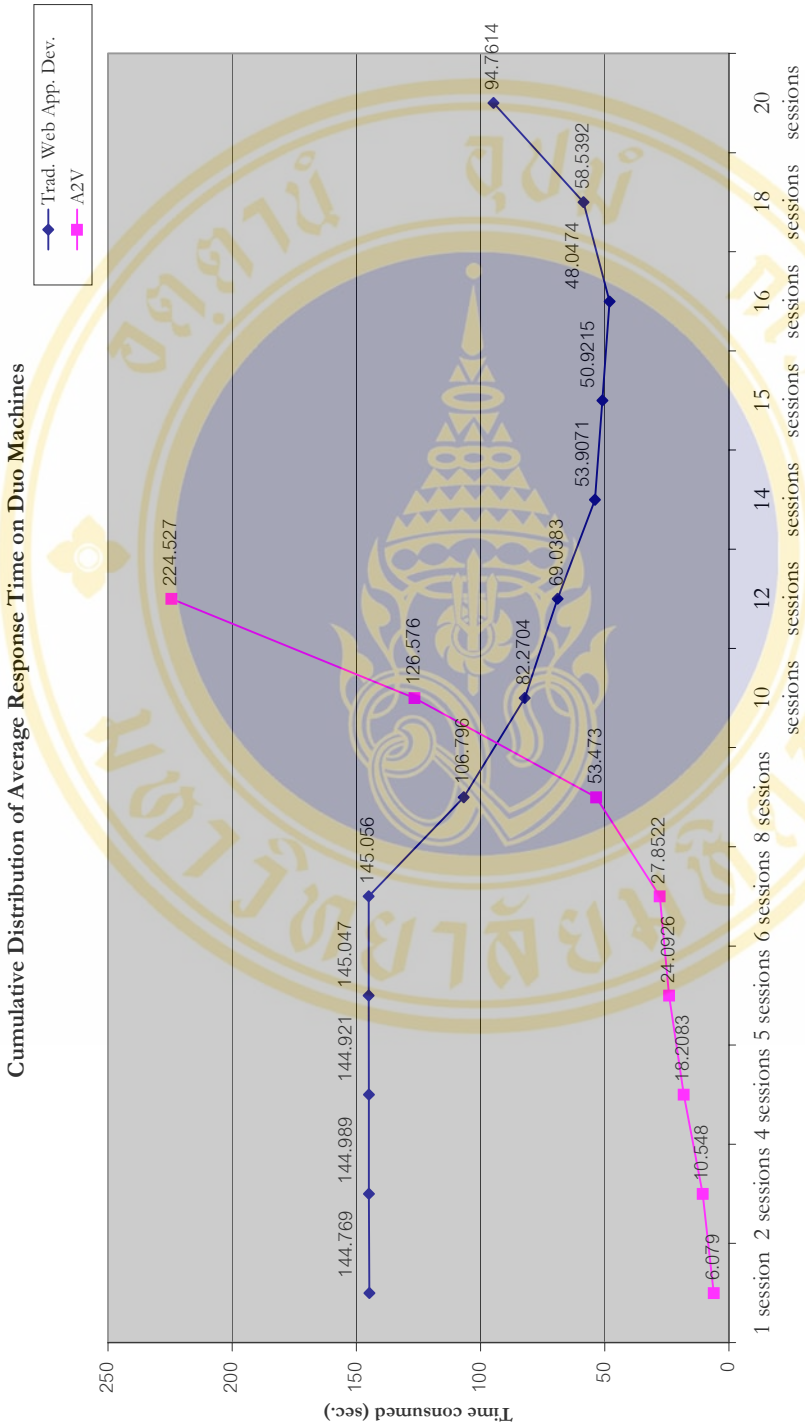


Figure 6-5 Comparison Chart of Average Response Time on Duo Machines between A^2V and Traditional Web Applications Development Approaches

Copyright by Mahidol University

Table 6-3 Comparison of Cumulative Response Time on Multiple Machines (15 machines) between A^2V and Traditional Web Applications Development Approaches

No. of concurrent session(s)	A^2V			Traditional Web Applications Development Approaches		
	Min. accum. response time (sec.)	Max. accum. response time (sec.)	Avg. accum. response time (sec.)	Min. accum. response time (sec.)	Max. accum. response time (sec.)	Avg. accum. response time (sec.)
1	6.079	6.079	6.079	144.769	144.769	144.769
2	6.39	6.488	6.439	145.188	145.532	145.36
4	6.289	18.567	10.38967	144.749	146.011	145.21
5	5.588	17.705	11.50025	144.736	146.332	145.249
6	5.868	16.824	10.5334	144.877	146.013	145.455
8	6.077	17.185	9.9865	144.764	145.641	145.186
10	5.609	16.644	9.085143	64.312	145.327	122.054
12	3.031	17.065	8.335	22.792	144.875	106.85
14	4.375	17.354	9.200333	23.547	145.108	95.5869
15	3.358	16.794	8.7	2.844	147.938	88.8895
16	3.422	17.385	9.081091	2.103	146.251	84.4734
18	3.438	17.786	9.060583	1.467	144.891	73.8277
20	3.751	17.055	9.607846	3.255	145.531	58.5748

Summary of the Results on Multiple Machines

In Table 6-3, since there is a sufficient resource to run the AV Client properly, all the entries are filled up. One resource is fully occupied by one session of each AV client and each log capturer.

The average accumulated response times of the A^2V framework and traditional web applications development approaches on *multiple machines* in Table 6-3 are represented in a graph in Error! Reference source not found.Figure 6-6. The difference of response time by the A^2V framework and traditional web applications development approaches is clearly insightful. The A^2V line is relatively precise while the line of traditional web applications development approaches seems high, even tends to be low thereafter. However, if we sum up the cumulative response times of all concurrent sessions of both A^2V framework and traditional web applications development approaches at the same *n* sessions and compare against each other, we will get that the

Copyright by Mahidol University

results of traditional web applications development approaches are greater than that of the A^2V framework.

By overall, at all levels of concurrent sessions conducted in this test environment, we can conclude that the average accumulated response times of the A^2V framework on multiple machines are apparently better than that of traditional web applications development approaches.

Copyright by Mahidol University

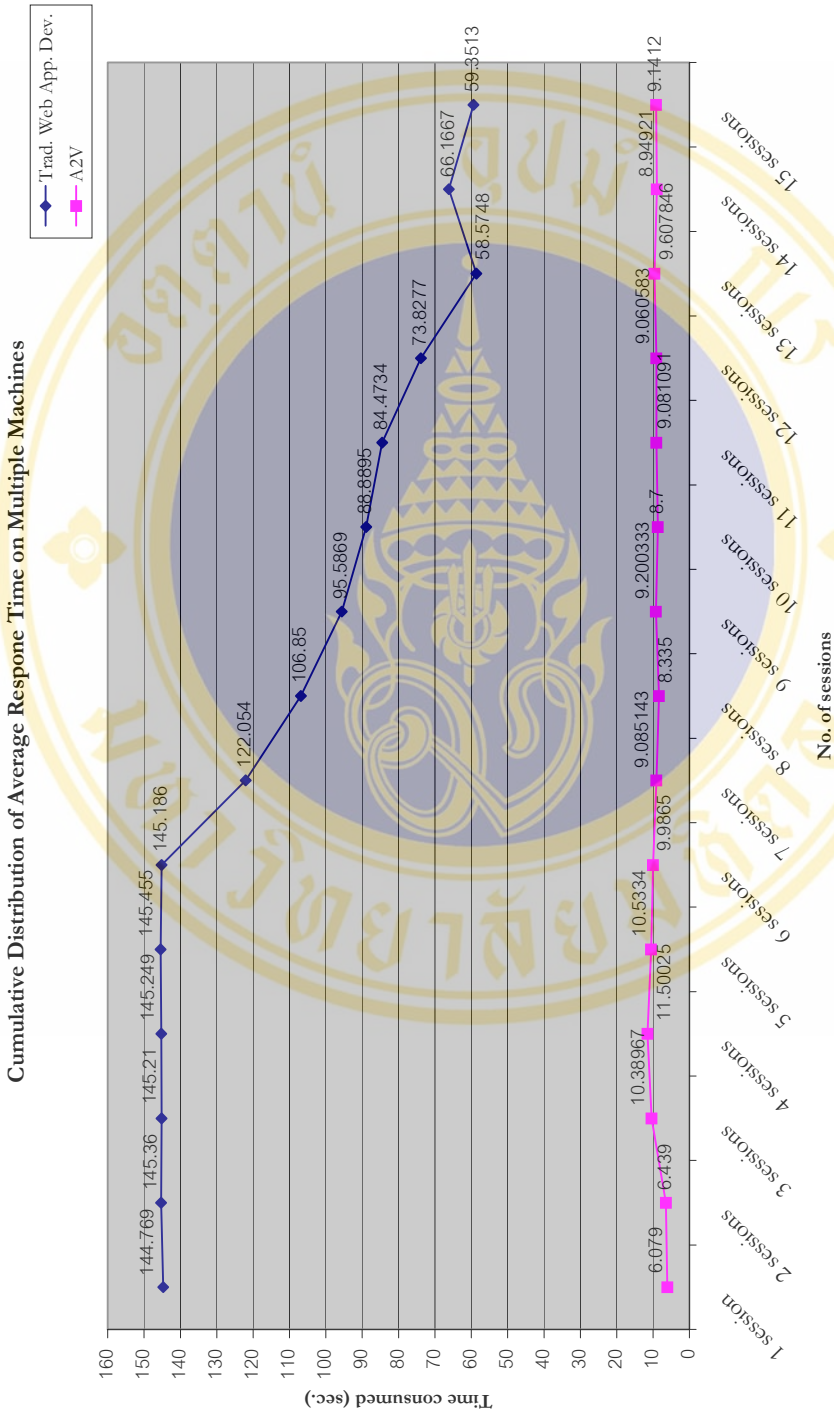


Figure 6-6 Comparison Chart of Average Response Time on Multiple Machines between A^2V and Traditional Web Applications Development Approaches

Copyright by Mahidol University

Table 6-4 Average Accumulated Response Times and Average Memory Consumed by a Single AV Client in Processing Multiple User Requests

No. of concurrent session(s)	Average Accumulated Response Times (sec.)	Average Memory Consumed (MB)
2	5.133	23
4	8.475	35
6	10.855	41
8	14.331	32
10	16.133	46
12	18.005	45
14	20.902	62
16	24.481	25
18	27.723	33
20	26.749	51

Summary of the Results by a Single AV Client toward Multiple User Requests

The average accumulated response times and the average memory consumed of the single AV Client in processing the different number of user requests in the test environment on a single machine in Table 6-4 are represented in graphs in Figure 6-7 and Figure 6-8 respectively. From the graphs, a horizontal axis stands for the number of concurrent user requests running and vertical axes in Figure 6-7 and Figure 6-8 stand for the average time in seconds consumed for constructing seven continuous Active Views.

From the graph in Figure 6-7, the average response times tend to be higher (longer) when the number of concurrent user requests goes up. Similarly, from the graph in Figure 6-8, the overall memory consumed goes up as the number of concurrent user requests rises. However, the results at some high levels of the number of concurrent user requests (for example, at eight, ten, and sixteen) fall. In summary, a single AV Client can process concurrent user requests.

Copyright by Mahidol University

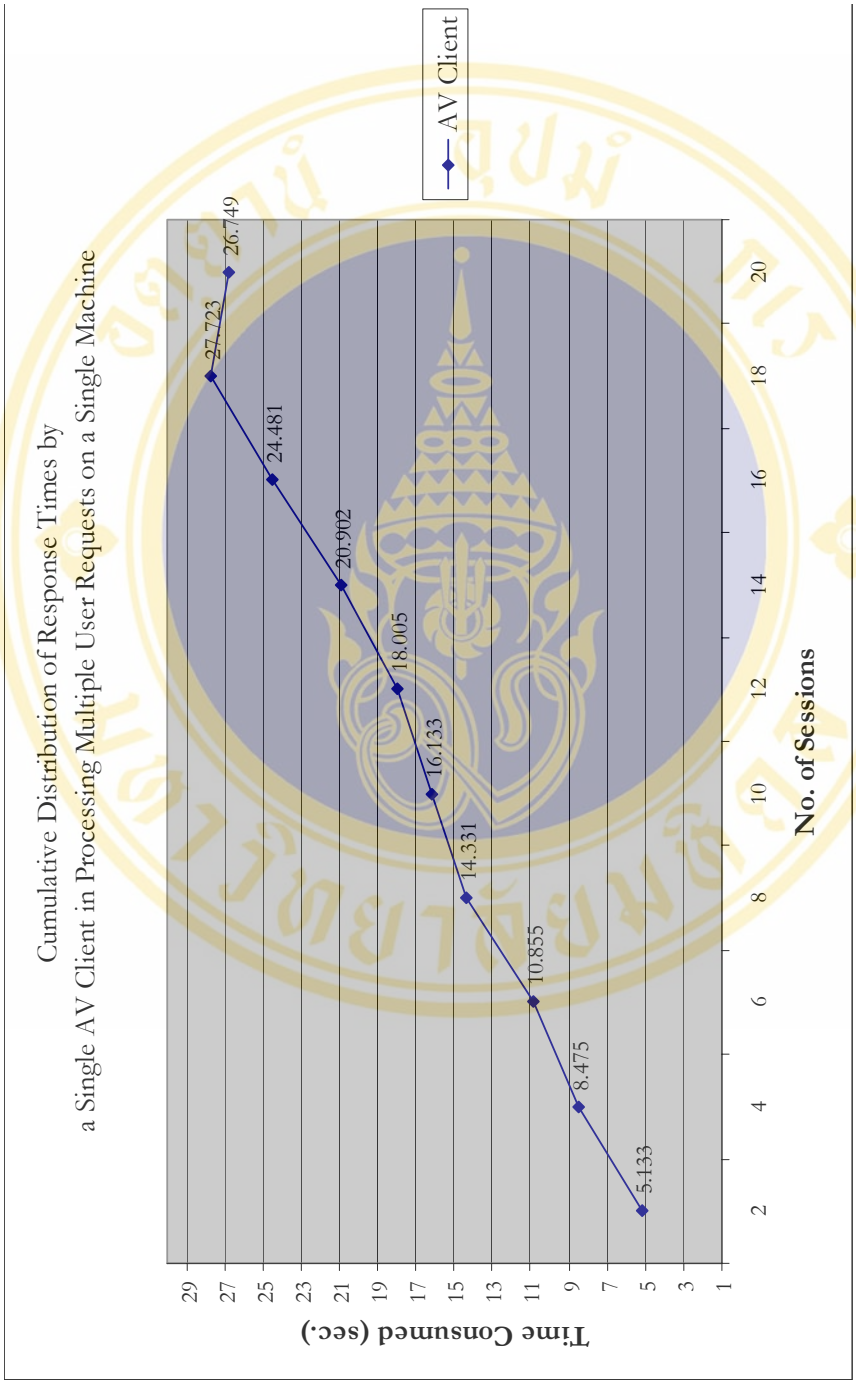


Figure 6-7 Cumulative Distribution of Average Response Times by a Single AV Client in Processing Multiple User Requests

Copyright by Mahidol University

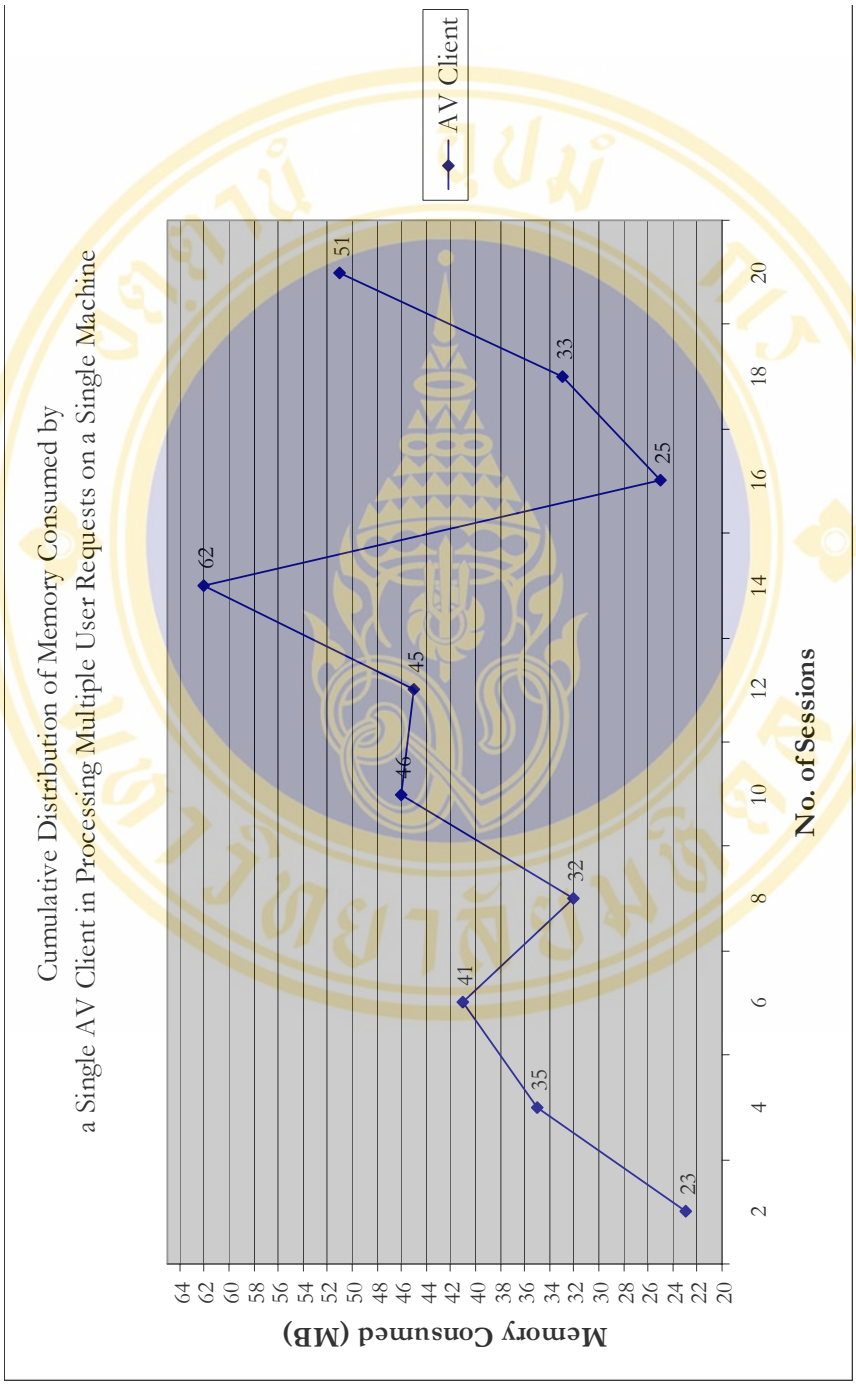


Figure 6-8 Cumulative Distribution of Average Memory Consumed by a Single AV Client in Processing Multiple User Requests

Copyright by Mahidol University

6.4.4 Critical Analysis

1) Critical Analysis of the Results on a Single Machine, Duo Machines, and Multiple Machines

As stated earlier, on a *single machine*, some sessions were incompletely executed. The reason behind this is that there were too many concurrent sessions running on the same machine at the same time that caused the system run out of memory. They were sharing the limited system resource. In addition, the AV Client was implemented by Java-based platform, which is notably well known for high resource consumption. In addition, on *duo machines*, the number of incomplete session obviously went higher since the hardware specification set up for being stimuli of clients has a lower performance. Therefore, when the number of concurrent sessions are high, the average cumulative response times of the A^2V framework is far longer that of traditional web applications development approaches – due to the shortage of available system resource. In practice, it will never be like this due to the fact that one user session is highly recommended for one AV Client – the test result representing this fact will be present in Figure 6-6.

Furthermore, even though the real cause of that the average accumulated response times of the traditional web applications development approaches tend to be lower is yet to be known, it is possible that the concurrent sessions tend to be fully utilized by the processing of a web server. As a consequence, some got the response output in a very short time whilst some wait longer to receive the response output.

On *multiple machines*, the overall workload is shared among the concurrent sessions efficiently because each AV Client and each Log Capturer were able to fully utilize the system resource on their own machines. Figure 6-6 clearly happens to be a realistic scenario.

Copyright by Mahidol University

2) Critical Analysis of the Nature of Active Rules Executions toward the Demonstrated Case Study

According to the demonstrated case study earlier discussed in Chapter V, we have derived the seven Active Views which are represented by two Remote Active Rules and seven Local Active Rules. Thus, considering the number of Local Active Rules and Remote Active Rules in the demonstrated case study, the number of Active Views represented by Remote Active Rules is less than the number of Active Views represented by Local Active Rules. In other words, most of business logic are appropriately represented by Local Active Rules and also the data items used by these Local Active Rules are retrievable at a client side. The two Active Views, represented by Remote Active Rules, are constructed by having the communications to a server side to acquire a new set of Active Rules and data items, whereas the rest of Active Views, represented by Local Active Rules can be remotely and completely constructed at a client side.

Conclusively, the more business logic and data items are expressed in term of Local Active Rules, the less the number of communication between a client side and a server side is. As a result, this helps enhance the performance of Active View construction. However, some business logic and some data items are too risk to be brought to process at a client side. Web developers have to determine business requirements of their e-services to balance the better performance of Active View construction against vulnerability of data.

3) Critical Analysis of Capability of Single AV Client in Processing Multiple User Requests

From the graph in Figure 6-8, the memory consumed at some high levels of number of concurrent user requests unexpectedly drops. The possible reason why this is the case is that some user requests could not be processed to construct complete Active Views due to the fact that CPU was fully used up and it ran out of available system resources, as well as an access violation in mutually reading the same file. Hence, there are less user requests to be processed.

Copyright by Mahidol University

According to the experimental results of response time and memory consumed, a single AV Client is capable of processing multiple user requests concurrently. However, the more the number of user requests increases, the more the time consumed and the memory consumed for constructing Active Views for each user request is. For each two additional concurrent requests (up to twenty), the response time of a single AV Client in processing user requests rises by 46.79% and the memory consumed rises by 13.53% (which is 8.03% of a capacity of main memory (512MB)).

From the experiments earlier conducted in the different test environments, a single AV Client needed to process a single user request at a time for each experiment as a one-to-one basis, whereas this experiment was conducted by having a single AV Client processing multiple user requests concurrently. The performance of a single AV Client does deteriorate when the number of user requests goes up. However, by comparison, the average accumulated response time of a traditional web application development on a single machine running twenty concurrent sessions approaches 38.952 seconds (see Figure 6-4). The response times of 26.749 seconds in this experiment with the same number of concurrent user requests is shorter. Therefore, this experiment with the AV Client supports the real scenarios in which most users often opens multiple browsers to have access to multiple web pages at a time.

6.5 Experiment Summary

In conclusion, to prove the efficiency of an Active View construction performance by our A^2V framework, we have conducted our experiments in two aspects which are the critical analysis and the hands-on experiments.

First, the critical analysis aims to give the reasonable evidence that the Active Views which are able to be distributively constructed at a client side could improve the performance of an Active View construction. The critical analysis has been performed through the A^2V framework against existing Active View approaches. To demonstrate which approach gives the better performance, it has formulated the response time consumed for constructing an Active View of each model with respect to the two scenarios. Also, it has compared the response times by calculating the difference of the

Copyright by Mahidol University

response time between the A^2V framework and existing Active View construction approaches. From the critical analysis, we see that the performance of an Active View construction by the A^2V framework is logically better than that of existing Active View approaches because some Active Views can be distributively constructed at a client side completely. In addition, the better performance will be even more obvious if there are more Active Views which are remotely constructed at a client side.

Second, we have conducted the hands-on experiments which aim to measure the performance of the A^2V framework against traditional web applications development approaches. In experiments, we compared the total accumulated response times of each model at the different level of number of concurrent sessions. In the experiment, we ran the demonstrated case study through the two models' prototypes and collected the transactional log files for doing analysis of the response time. Based on the average experimental results according to the three different experimental environments, it reveals that the response times measured from the A^2V framework is about 91.39% better than that of traditional web applications development approaches. Furthermore, it has convinced that having the AV Client distributively installed on a user machine by one-to-one happens to be the appropriate and outstanding scenario which efficiently enhances the performance of an Active View construction for e-service applications. Additionally, we have conducted the experiments which support an applicability of a single AV Client in being capable of processing multiple user requests concurrently.

Next chapter will provide the discussion and conclusion on our research. Some assumptions and limitations of our work will also be included.

Copyright by Mahidol University

CHAPTER VII
DISCUSSION AND CONCLUSION

This chapter discusses and gives the conclusion of our A^2V framework. Section 7.1 discusses how the A^2V framework alternatively and efficiently enhances the performance of Active View construction. Next, Section 7.2 presents the limitations of A^2V implementation and experiment conduction. Finally, Section 7.3 guides the further research and development of the A^2V framework.

7.1 Discussion and Conclusion

As stated in the introductory chapter, the construction process of Active Views conducted in a centralized control manner at a sever side may lead to low performance of the process and the process may even run into a bottleneck state when concurrent user sessions increase. To resolve such problems, this research work has proposed an alternative model, namely A^2V, which is based on the decentralized control architecture as adapted from the Active XML framework. In the A^2V framework, we enhance the performance of Active View construction process for e-service applications by relocating part of Active View construction process to be distributively executed at a client side. To accomplish this, we propose a new markup language, namely AVML (short for Active Views Markup Language), which is capable of be properly operated in a distributed environment.

AVML works with two processing units, the AV Client and the AVML Processor. The structure of an AVML document is divided into two parts, Web Contents and Active Rules. The Web Contents part contains a set of transactional data, while the Active Rules part contains a set of Active Rules, which are used to control the access of data and activities on each particular Web page. Some Active Rules,

Copyright by Mahidol University

referred to as Local Active Rules, can be entirely executed by the AV Client at a client side, whereas some, referred to as Remote Active Rules, need to be cooperatively executed by the AV Client and the AVML Processor, which is located on a server side. We also termed an AVML-formatted web page or an AVML document as an *Active Web Page*. Both local Active Rules and remote Active Rules are embedded in Active Web Pages and they make part of the Active Web Pages be actively executed to construct Active Views.

According to the materialization process as described in Section 3.7.4 of Chapter III, we also classify the life cycle of AVML documents into two states, an implicit state and an explicit state. An *implicit AVML document* is an AVML document *before* realized by the materialization process whilst an *explicit AVML document* is an AVML document *after* enriched by the materialization process. Basically, implicit AVML documents are pre-defined by web developers using our *A²V* guidelines as described in Section 3.8 of Chapter III and the documents are stored at a server side. After passing the materialization process, an implicit AVML document will be transformed into an explicit AVML document, so-called Active Web Pages as discussed earlier. The AVML documents in explicit state are the responses that will be returned to the AV Client for the construction of Active Views.

Note that a security issue might be raised when the AV Client invokes a service call for executing the triggered Remote Active Rule to the AVML Processor. This is because we did not apply any authentication scheme to verify incoming requests from the different AV Clients. As a result, unauthorized applications fraudulently impersonate the authoritative AV Client to enter the system easily.

Based on the collaborative execution of the AV Client and the AVML Processor in the A²V framework, we can see that an improvement of a response time used for constructing an Active View will be achieved when part of the construction process is left to be executed at a client side, thereby eliminating some communication processes between clients and servers. We have provided a critical analysis and some hands-on experiments as a proof-of-concept to that. Our critical analysis has shown that the response time of the existing Active View construction approaches is typically longer than that of the A²V framework while the hands-on experiments have shown that the A²V framework can achieve the state of our research objectives by reducing

Copyright by Mahidol University

approximately 91.39% of time consumed in the Active View construction in comparison to traditional Web application development approaches. Therefore, we can conclude that the A^2V framework has the better performance of Active View construction process for e-service applications than that of other existing Active View construction approaches based on the principle of centralized control. It is also more efficient to use the A^2V framework to create Web Access Control than using traditional Web application development approaches. Furthermore, our hands-on experiments have shown that the A^2V framework can handle multiple user requests concurrently at a time, so this ensures an applicability of the A^2V framework which is able to support the real scenarios in which most users often opens multiple browsers to have access to multiple web pages at a time.

7.2 Limitations

There are some limitations of our A^2V development as follows:

- Due to the difficulties in finding the source codes of the existing Active View construction approaches, we cannot set up the real experiments to compare the existing approaches with ours. Instead, we find some paper discussing on the communication process between a client side and a server side to help prove the applicability of our A^2V framework by using a theoretical proof.
- Unwittingly, the efforts are required to apply the A^2V framework to be used in existing web applications and also e-service applications built from scratch since we do not provide the tool which facilitates a schema creation for implicit AVML documents of the A^2V framework. In practice, a web developer needs to follow our A^2V guideline as mentioned earlier to study business flows of a web application and derive the flows to a page transition diagram or an activity diagram. Then he/she has to manually declare Local Active Rules or Remote Active Rule, and new implicit AVML documents by using the information obtained from those diagrams.
- Due to the simplicity and immaturity of XQuery API, we have stored Remote Active Rules by using relational databases. Yet, there is a cost of query

Copyright by Mahidol University

operations to be taken into account, for example inputting/outputting overheads. The more the attributes and contents of a Remote Active Rule are, the longer the time we require to read or write the rule to the databases.

7.3 Further Research

Further research can be developed in order to resolve the above limitations as follows:

- To promote the usability of A^2V, an interface adaptor that is able to convert existing e-service applications or e-service applications built from scratch to be represented by Active Rules and implicit AVML documents should be developed and finally combined into the AVML Processor component. This utility should make the conversion process easier and more convenient for web developers.
- To promote the capability of the AV Client in processing multiple user requests efficiently, file sharing techniques should be studied more and applied to modify the AV Client to resolve the problem of file access violation when processing concurrent user requests.
- As stated in the earlier section, it would be useful to seek other methods to implement the Remote Active Rule rather than the relational database. So far, XPath and the indexing techniques would be interesting choices, or even XQuery will be an outstanding choice if the API stability and its ease-of-use are favored by the developers.
- By virtue of the limitation of data sources where to obtain items for the Web Contents section of AVML document – they are strictly allowed for a database server or a service call parameters document; it should be extended to other sources – for example, other databases, XML repository, computer programs, or backend systems. XLink could be considered regarding this.
- It would be beneficial to discover new features and competence of XSL, XLink, and XQuery in detail to enrich the display of Active Views.
- It would be value-added for the A^2V framework to apply public-key encryption schemes to the AVML Processor for authenticating incoming service call

Copyright by Mahidol University

requests for executing the triggered Remote Active Rule from various AV Clients. For example, each AV Client might be asked to issue a valid code to the AVML Processor in order to check out its validity prior to continuing further processes.

- To seek a new alternative that can reduce the communication cost between a client side and a server side. So far, we have heard of the new arrival AJAX [41] which is the framework that enables higher performance, more responsive, more interactive by replacing the “click, wait, and refresh” user interaction and the synchronous request/response characteristic in the conventional web applications development framework by the partial screen update and asynchronous communication techniques featured by Java and XML. It is worth studying whether we can apply AJAX to replace the process of the AV Client.

Copyright by Mahidol University

REFERENCES

1. The Wikipedia Homepage. Available from: http://en.wikipedia.org.
2. S. Abiteboul, B. Amann, S. Cluet, A. Eyal, L. Mignet. and T. Milo. **Active Views for Electronic Commerce**. In Int. Conf. on Very Large Databases (VLDB), Edinburgh, Scotland, pages 138-149, 1999.
3. K. Sakulwong. **Active View Agent System for Electronic Commerce**. Master Thesis, Computer Science Department, Mahidol University, 2002.
4. H. Kiyomitsu, A. Takeuchi. and K. Tanaka. **ActiveWeb: XML-Based Active Rules for Web View Derivations and Access Control**. In IEEE Proc, page 31-39, 2001.
5. S. Abiteboul et. al. **Active XML Primer**. Available from: ftp://ftp.inria.fr/INRIA/Projects/gemo/gemo/GemoReport-307.pdf.
6. S. Abiteboul, O. Benjelloun and T. Milo. **The Active XML Project: An Overview**. November 17, 2005. Available from: ftp://ftp.inria.fr/INRIA/Projects/gemo/gemo/GemoReport-331.pdf.
7. S. Abiteboul et. al. **Active XML: A Data-Centric Perspective on Web Services**. BDA 2002. Available from: ftp://ftp.inria.fr/INRIA/Projects/gemo/gemo/GemoReport-213.ps.
8. The Active XML Homepage. Available from: http://activexml.net/.
9. S. Abiteboul et. al. **Active XML**. Slide Presentation. Available from: http://activexml.net/axml-10-2004.pdf.
10. A. Bonifati, S. Ceri. and S. Paraboschi. **Active Rules for XML: A New Paradigm for E-Services**. VLDB Journal, 10(1):39-47, 2001.
11. A. Bonifati, S. Ceri. and S. Paraboschi. **Pushing Reactive Services to XML Repositories Using Active Rules**. In Proc. 10th World-Wide-Web Conference, 2001.
12. N.W. Paton and O. Diaz. **Active Database Systems**. ACM Computing Surveys, Vol. 31 No. 1, March 1999.

Copyright by Mahidol University

13. The XML Protocol Working Group. **SOAP Version 1.2 Part 1: Messaging Framework**. W3C Recommendation (24 June 2003). Available from: http://www.w3.org/TR/soap12-part1/.
14. J. Baumgarten. **Active XML User Guide**. April 27, 2004. Available from: http://forge.objectweb.org/projects/activexml/.
15. The Active XML Team. **Active XML User's Guide**. September 2005. Available from: http://activexml.net/AXML%20Guide.pdf.
16. I. J. Taylor. **Lecture 1: Taxonomy for Distributed Systems**. Available from: http://users.cs.cardiff.ac.uk/Ian.J.Taylor/DistributedSystems/Notes/lecture1.pdf.
17. I. J. Taylor. **Lecture 2: Peer to Peer Systems**. Available from: http://www.cs.cf.ac.uk/user/I.J.Taylor/Dributedsystems/Notes/lecture2.pdf, Distributed Systems, 2003.
18. B. Buehling. **Enabling Peer-to-Peer (P2P) Applications with XML**. Available from: http://www.gca.org/papers/xmleurope2001/papers/html/s08-1.html.
19. Networking/Telecom Section. **Is P2P Ready to Do Business?**. May 23, 2001. Available from: http://www.computerweekly.com/Article102171.htm.
20. D. Hunter et. al. **Beginning XML 2nd Edition**. Wrox Press, August 2002.
21. K. Cagle. et. al. **Professional XSL**. Wrox Press, 2001.
22. S. Bonneau. et al. **XML Design Handbook**. 1st Edition, Wrox Press, March 2003.
23. C. M. Sperberg-McQueen. and H. Thompson. **XML Schema**. March 10, 2006. Available from: http://www.w3.org/XML/Schema.
24. F. Arciniegas A. **XML for Developer's Guide**. 1st Edition, McGraw-Hill, December 21, 2000.
25. A. Moller and M. I. Schwartzbach. **XML Tutorial: XLink, XPointer, and XPath**. Available from: http://www.brics.dk/~amoeller/XML/linking/
26. James Clark, editor. **XSL Transformation (XSLT) Version 1.0**, W3C Recommendation (16 November 1999). Available from: http://www.w3.org/TR/xslt.

Copyright by Mahidol University

27. James Clark, editor. **XML Path Language (XPath) Version 1.0**. W3C Recommendation (16 November 1999). Available from: http://www.w3.org/TR/xpath.
28. T. Berners-Lee. **When to use XLink – Design Issues**. Available from: http://www.w3.org/DesignIssues/XLink.html.
29. M. Fitzgerald. **Create Well-Formed XML with JavaScript**. Available from: http://hacks.oreilly.com/pub/h/2127.
30. J. Plachy. **Java Proxy Server Source Code**. Available from: http://www.c-lab.de/~jindent.
31. JDOM Homepage, Available from: http://www.jdom.org.
32. M. Kay. **About SAXON**. Available from: http://saxon.sourceforge.net/saxon7.4/index.html.
33. Apache AXIS Web Services. Available from: http://ws.apache.org/axis/.
34. The Java™ Tutorial. **Relational and Conditional Operators**. Available from: http://java.sun.com/docs/books/tutorial/java/nutsandbolts/relational.html.
35. Amazon Homepage. Available from: http://www.amazon.com.
36. WAPT – Web Application Load, Stress and Performance Testing Homepage. Available from: http://www.loadtestingtool.com.
37. Logging Services. Available from: http://logging.apache.org.
38. C. Gulcu. **Short Introduction to log4j**. Available from: http://logging.apache.org/log4j/docs/manual.html.
39. A. J.S Mills. **Log4J**. The University of Birmingham, 2005. Available from: http://supportweb.cs.bham.ac.uk/documentation/tutorials/docsystem/build/tutorials/log4j/log4j.html.
40. A. Khan. **Java Excel API Tutorial**. Available from: http://www.andykhan.com/jexcelapi/tutorial.html#reading.
41. C. K. Wei. **AJAX: Asynchronous Java + XML?**. Intranet Journal, August 23, 2005. Available from: http://www.intranetjournal.com/articles/200508/pij_08_23_05a.html.
42. MySQL Homepage. Available from: http://www.mysql.com.

Copyright by Mahidol University

APPENDIX

Copyright by Mahidol University

APPENDIX A
RELATIONAL DATABASE SCHEMA OF REMOTE ACTIVE RULE

This appendix gives an explanation of a Remote Active Rule in representing by a relational database schema.

Data Dictionary of Remote Active Rule

This section presents the data dictionary of each of the relational database schema of a Remote Active Rule, which is earlier specified in Section 4.1.2 of Chapter IV. Note that its data type is subjecting to our chosen database management system, MySQL DBMS [42].

A.1 Database Table: remote_rule

Description: Remote Active Rule entity

Name	Description	Data Type	Allow Null (Y/N)	Key
rem_ruleid	Remote Active Rule identifier	int(11) unsigned zerofill auto_increment	N	PK
name	Name of Remote Active Rule in short	varchar(35)	Y	
cond_clause	Its Condition part	text	Y	
hasmore_item	The flag guiding if more items are bound prior to evaluating the condition part	char(1)	Y	
noofresult	The number of possible evaluated results (true/false)	tinyint(2) unsigned	Y	
header_part	Its data binding part	text	Y	

Copyright by Mahidol University

A.2 Database Table: evaluated_result

Description: Evaluated Result of a condition part of a Remote Active Rule

Name	Description	Data Type	Allow Null (Y/N)	Key
output_id	Output identifier – one of composite key identifying an Active Web Page	smallint(6) unsigned zerofill auto_increment	N	PK
rem_ruleid	Remote Active Rule identifier	int(11) unsigned zerofill	N	FK
result	The possible truth values (true/false) resulting from the condition evaluation of the Remote Active Rule	varchar(50)	N	

A.3 Database Table: referrer_page

Description: Previous Web Page causing an execution of Remote Active Rule

Name	Description	Data Type	Allow Null (Y/N)	Key
page_id	Web Page identifier	tinylint(3) unsigned zerofill auto_increment	N	PK
uri	Its URI	varchar(150)	Y	
page_name	Its short name	varchar(35)	Y	

Copyright by Mahidol University

A.4 Database Table: active_web_page

Description: Implicit AVML document entity

Name	Description	Data Type	Allow Null (Y/N)	Key
page_id	Referrer Web Page identifier	tinylint(3) unsigned zerofill auto_increment	N	PK
output_id	Output identifier	smallint(6) unsigned zerofill auto_increment	N	PK
uri	Its URI	varchar(150)	Y	
page_name	Name of Active Web Page in short	varchar(35)	Y	
last_update	Its last modified date	Date	Y	
global_used	The flag guiding the AV Client if the Global Data item is needed for executing any active rules of this implicit AVML document	char(1)	Y	
userprof_used	The flag guiding the AV Client if the User Profile is needed for executing any active rules of this implicit AVML document	char(1)	Y	

A.5 Database Table: source_type

Description: Data Source where the data item comes from

Name	Description	Data Type	Allow Null (Y/N)	Key
src_type	Type code that could possibly be of DB, VALUE, and PARAMS	varchar(10)	N	PK
desc	Its description in brief	varchar(35)	Y	

Copyright by Mahidol University

A.6 Database Table: web_content

Description: List of Data Items put in the Web Contents section of an implicit AVML document

Name	Description	Data Type	Allow Null (Y/N)	Key
id	Item identifier	smallint(6) unsigned zerofill auto_increment	N	PK
name	Item name	varchar(35)	Y	
category	Which type it is classified into (Global/User Profile)	varchar(10)	Y	
src_type	Where to obtain its value as to the type of source	varchar(10)	Y	
value	How to get its value: - SQL if type is DB; - Exact value if type is VALUE; - XPath expression if type is PARAMS;	text	Y	
has_param	The flag guiding if any parameters are needed for	char(1)	Y	
params	Where to obtain parameter	text	Y	

A.7 Database Table: map_content

Description: Mapping of Data Item belonging to an implicit AVML document

Name	Description	Data Type	Allow Null (Y/N)	Key
page_id	Referrer Web Page identifier	tinyint(3) unsigned zerofill	N	PK, FK
output_id	Output identifier	smallint(6) unsigned zerofill	N	PK, FK
id	Local Active Rule identifier	smallint(6) unsigned zerofill	N	PK, FK

Copyright by Mahidol University

A.8 Database Table: local_rule

Description: Local Active Rule entity

Name	Description	Data Type	Allow Null (Y/N)	Key
loc_ruleid	Local Active Rule identifier	int(11) unsigned zerofill auto_increment	N	PK
name	Name of Local Active Rule in short	varchar(35)	Y	
body	The Condition-Action part	mediumtext	Y	

A.9 Database Table: map_local

Description: Mapping of Local Active Rule belonging to an implicit AVML document

Name	Description	Data Type	Allow Null (Y/N)	Key
page_id	Referrer Web Page identifier	tinyint(3) unsigned zerofill	N	PK, FK
output_id	Output identifier	smallint(6) unsigned zerofill	N	PK, FK
loc_ruleid	Local Active Rule identifier	int(11) unsigned zerofill	N	PK, FK

A.10 Database Table: map_remote

Description: Mapping of Remote Active Rule belonging to an implicit AVML document

Name	Description	Data Type	Allow Null (Y/N)	Key
page_id	Referrer Web Page identifier	tinyint(3) unsigned zerofill	N	PK, FK
output_id	Output identifier	smallint(6) unsigned zerofill	N	PK, FK
rem_ruleid	Remote Active Rule identifier	int(11) unsigned zerofill	N	PK, FK

Copyright by Mahidol University

APPENDIX B
XML SCHEMA REPRESENTATION OF A^{2}V KNOWLEDGE

This appendix provides the XML Schema representation of all A^{2}V data that is an AVML document, two types of Active Rule, a User Activity document, a Merged Items document used for a Local Active Rule execution, and a Service Call Parameters document used for the AVML Processor processing.

B.1 AVML Document

Table B-1 depicts the XML schema structure of an *implicit* AVML document.

Copyright by Mahidol University

Table B-2 XML Schema Representation for an Implicit AVML Document

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xlink="http://www.w3.org/1999/xlink" elementFormDefault="qualified"
attributeFormDefault="unqualified">
 <xsd:element name="ActiveWebPage">
  <xsd:annotation>
   <xsd:documentation>This is an implicit AVML document</xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element name="WebContents">
     <xsd:annotation>
      <xsd:documentation>This is the Web Contents section</xsd:documentation>
     </xsd:annotation>
     <xsd:complexType>
      <xsd:sequence>
       <xsd:element name="global">
        <xsd:annotation>
         <xsd:documentation>This is the Global Data
section</xsd:documentation>
        </xsd:annotation>
        <xsd:complexType>
         <xsd:sequence minOccurs="0">
          <xsd:any namespace="Global Data-related items" minOccurs="0"
maxOccurs="unbounded">
           <xsd:annotation>
            <xsd:documentation>These are the Global Data
references</xsd:documentation>
           </xsd:annotation>
          </xsd:any>
         </xsd:sequence>
         <xsd:attribute name="used" type="xsd:boolean" use="optional"/>
        </xsd:complexType>
       </xsd:element>
       <xsd:element name="UserProfile">
        <xsd:annotation>
         <xsd:documentation>This is the User Profile
section</xsd:documentation>
        </xsd:annotation>
        <xsd:complexType>
         <xsd:sequence minOccurs="0">
          <xsd:any namespace="User Profiles-related items" minOccurs="0"
maxOccurs="unbounded">
           <xsd:annotation>
            <xsd:documentation>These are the User Profile references
</xsd:documentation>
           </xsd:annotation>
          </xsd:any>
         </xsd:sequence>
         <xsd:attribute name="used" type="xsd:boolean" use="optional"/>
        </xsd:complexType>
       </xsd:element>
      </xsd:sequence>
     </xsd:complexType>
    </xsd:element>
    <xsd:element name="ActiveRules">
     <xsd:annotation>
      <xsd:documentation>This is the Active Rules section</xsd:documentation>
     </xsd:annotation>
     <xsd:complexType>
      <xsd:sequence>
```

Copyright by Mahidol University

```
       <xsd:element name="ar" maxOccurs="unbounded">
        <xsd:annotation>
         <xsd:documentation>These are the references referring to the real
contents of Active Rules in the Active Rules repository.</xsd:documentation>
        </xsd:annotation>
        <xsd:complexType>
         <xsd:attribute name="id" type="xsd:ID" use="required"/>
         <xsd:attribute name="name" type="xsd:string" use="optional"/>
         <xsd:attribute name="type" type="xsd:string" use="required"/>
         <xsd:attribute name="frequency" type="xsd:string" use="required"/>
        </xsd:complexType>
       </xsd:element>
      </xsd:sequence>
     </xsd:complexType>
    </xsd:element>
   </xsd:sequence>
   <xsd:attribute name="name" type="xsd:string" use="optional"/>
   <xsd:attribute name="ecAppName" type="xsd:string" use="required"/>
   <xsd:attribute name="referrerURI" type="xsd:string" use="required"/>
   <xsd:attribute name="lastUpdated" type="xsd:string" use="optional"/>
  </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

B.2 Local Active Rule

Table B-3 depicts the XML Schema structure of a Local Active Rule.

Table B-3 XML Schema Representation for a Local Active Rule

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:annotation>
  <xsd:documentation xml:lang="en">
    XML Schema Representation for Local Active Rule
  </xsd:documentation>
 </xsd:annotation>
 <xsd:element name="ar" type="ServiceCallType"/>
 <xsd:complexType name="ServiceCallType">
  <xsd:sequence>
   <xsd:element name="DataBinding">
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element name="global" minOccurs="0">
       <xsd:complexType>
        <xsd:sequence>
         <xsd:any namespace="##any" processContents="skip" minOccurs="0"
maxOccurs="unbounded">
          <!--xsd:attribute name="xlink:type" type="xsd:string">
          <xsd:simpleType>
           <xsd:restriction base="xsd:string">
            <xsd:enumeration value="simple"/>
           </xsd:restriction>
          </xsd:simpleType>
```

Copyright by Mahidol University

```
         </xsd:attribute>
         <xsd:attribute name="xlink:href" type="xsd:string">
          <xsd:annotation>
           <xsd:documentation xml:lang="en">
             This is of the format "globaldata.xml#" followed by XPath
expression to access the target element in the globaldata.xml
            Ex. globaldata.xml#//product refers the "product" elements in the
global data file.
            </xsd:documentation>
          </xsd:annotation>
         </xsd:attribute>
         <xsd:attribute name="xlink:actuate" type="xsd:string">
          <xsd:simpleType>
           <xsd:restriction base="xsd:string">
            <xsd:enumeration value="onRequest"/>
           </xsd:restriction>
          </xsd:simpleType>
         </xsd:attribute-->
         </xsd:any>
        </xsd:sequence>
       </xsd:complexType>
      </xsd:element>
      <xsd:element name="UserProfile" minOccurs="0">
       <xsd:complexType>
        <xsd:sequence>
         <xsd:any namespace="##any" processContents="skip" minOccurs="0"
maxOccurs="unbounded">
          <!--xsd:attribute name="xlink:type" type="xsd:string">
          <xsd:simpleType>
           <xsd:restriction base="xsd:string">
            <xsd:enumeration value="simple"/>
           </xsd:restriction>
          </xsd:simpleType>
         </xsd:attribute>
         <xsd:attribute name="xlink:href" type="xsd:string">
          <xsd:annotation>
           <xsd:documentation xml:lang="en">
             This is of the format "userprof.xml#" followed by XPath
expression to access the target element in the userprof.xml
            Ex. userprof.xml#//paymentmethod refers the "paymentmethod"
element in the user profile file.
            </xsd:documentation>
          </xsd:annotation>
         </xsd:attribute>
         <xsd:attribute name="xlink:actuate" type="xsd:string">
          <xsd:simpleType>
           <xsd:restriction base="xsd:string">
            <xsd:enumeration value="onRequest"/>
           </xsd:restriction>
          </xsd:simpleType>
         </xsd:attribute-->
         </xsd:any>
        </xsd:sequence>
       </xsd:complexType>
      </xsd:element>
     </xsd:sequence>
    </xsd:complexType>
   </xsd:element>
   <xsd:element name="CondAction">
    <xsd:annotation>
     <xsd:documentation xml:lang="en">
       This is XSL representation that is input by the data as mentioned in
the DataBinding element
       and formats the Active View.
      </xsd:documentation>
```

Copyright by Mahidol University

```
    </xsd:annotation>
    <xsd:complexType>
     <xsd:sequence>
      <xsd:any namespace="xsl:stylesheet"/>
     </xsd:sequence>
    </xsd:complexType>
   </xsd:element>
  </xsd:sequence>
  <xsd:attributeGroup ref="service-call-definition"/>
 </xsd:complexType>
 <xsd:attributeGroup name="service-call-definition">
  <xsd:attribute name="id" use="required">
   <xsd:annotation>
    <xsd:documentation xml:lang="en">
    For the pattern format, refer to the Unicode Regular Expression Guidelines
http://www.unicode.org/unicode/reports/tr18/
          </xsd:documentation>
   </xsd:annotation>
   <xsd:simpleType>
    <xsd:restriction base="xsd:ID">
     <xsd:pattern value="[0-9 A-F]{8}-[0-9 A-F]{4}-[0-9 A-F]{4}-[0-9 A-F]{4}-
[0-9 A-F]{12}"/>
    </xsd:restriction>
   </xsd:simpleType>
  </xsd:attribute>
  <xsd:attribute name="name" type="xsd:string" use="optional"/>
  <xsd:attribute name="type" use="required">
   <xsd:annotation>
    <xsd:documentation xml:lang="en">local indicates this Active Rule is a
type of Local</xsd:documentation>
   </xsd:annotation>
   <xsd:simpleType>
    <xsd:restriction base="xsd:string">
     <xsd:enumeration value="local"/>
    </xsd:restriction>
   </xsd:simpleType>
  </xsd:attribute>
  <xsd:attribute name="frequency" use="required">
   <xsd:annotation>
    <xsd:documentation xml:lang="en">
      The frequency can, so far, be of 2 formats. The first one is "onLoad"
which says that the service call has to be called only once the Active Web
Page is received from the AVML Processor.
      The second one is "lazy" which says that the service call has to be
called only upon the user action.
     </xsd:documentation>
    <xsd:documentation xml:lang="en">frequency = "onLoad"</xsd:documentation>
    <xsd:documentation xml:lang="en">frequency = "lazy"</xsd:documentation>
   </xsd:annotation>
   <xsd:simpleType>
    <xsd:restriction base="xsd:string">
     <xsd:enumeration value="onLoad"/>
     <xsd:enumeration value="lazy"/>
    </xsd:restriction>
   </xsd:simpleType>
  </xsd:attribute>
 </xsd:attributeGroup>
</xsd:schema>
```

Copyright by Mahidol University

B.3 Remote Active Rule

Table B-4 depicts the XML Schema structure of a Remote Active Rule.

Table B-4 XML Schema Representation for a Remote Active Rule

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
 <xsd:element name="ar">
  <xsd:annotation>
   <xsd:documentation>
    XML Schema Representation for a Remote Active Rule
   </xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element name="DataBinding">
     <xsd:annotation>
      <xsd:documentation>This is the data binding part.  It is the only one
part which will be embedded in an Active Web Page returned to a client
side.</xsd:documentation>
     </xsd:annotation>
     <xsd:complexType>
      <xsd:sequence>
       <xsd:element name="global" minOccurs="0">
        <xsd:complexType>
         <xsd:sequence>
          <xsd:any namespace="##any" minOccurs="0" maxOccurs="unbounded">
           <!--xsd:attribute name="xlink:type" type="xsd:string">
          <xsd:simpleType>
           <xsd:restriction base="xsd:string">
            <xsd:enumeration value="simple"/>
           </xsd:restriction>
          </xsd:simpleType>
         </xsd:attribute>
         <xsd:attribute name="xlink:href" type="xsd:string">
          <xsd:annotation>
           <xsd:documentation xml:lang="en">
             This is of the format "globaldata.xml#" followed by XPath
expression to access the target element in the globaldata.xml
            Ex. globaldata.xml#//product refers the "product" elements in the
global data file.
           </xsd:documentation>
          </xsd:annotation>
         </xsd:attribute>
         <xsd:attribute name="xlink:actuate" type="xsd:string">
          <xsd:simpleType>
           <xsd:restriction base="xsd:string">
            <xsd:enumeration value="onRequest"/>
           </xsd:restriction>
          </xsd:simpleType>
         </xsd:attribute-->
          </xsd:any>
         </xsd:sequence>
        </xsd:complexType>
       </xsd:element>
       <xsd:element name="UserProfile" minOccurs="0">
```

Copyright by Mahidol University

```
        <xsd:complexType>
         <xsd:sequence>
          <xsd:any namespace="##any" minOccurs="0" maxOccurs="unbounded">
           <!--xsd:attribute name="xlink:type" type="xsd:string">
          <xsd:simpleType>
           <xsd:restriction base="xsd:string">
            <xsd:enumeration value="simple"/>
           </xsd:restriction>
          </xsd:simpleType>
         </xsd:attribute>
         <xsd:attribute name="xlink:href" type="xsd:string">
          <xsd:annotation>
           <xsd:documentation xml:lang="en">
             This is of the format "userprof.xml#" followed by XPath
expression to access the target element in the userprof.xml
            Ex. userprof.xml#//paymentmethod refers the "paymentmethod"
element in the user profile file.
           </xsd:documentation>
          </xsd:annotation>
         </xsd:attribute>
         <xsd:attribute name="xlink:actuate" type="xsd:string">
          <xsd:simpleType>
           <xsd:restriction base="xsd:string">
            <xsd:enumeration value="onRequest"/>
           </xsd:restriction>
          </xsd:simpleType>
         </xsd:attribute-->
          </xsd:any>
         </xsd:sequence>
        </xsd:complexType>
       </xsd:element>
       <xsd:element name="params" minOccurs="0">
        <xsd:complexType>
         <xsd:sequence>
          <xsd:any namespace="##any" minOccurs="0" maxOccurs="unbounded"/>
         </xsd:sequence>
        </xsd:complexType>
       </xsd:element>
      </xsd:sequence>
     </xsd:complexType>
    </xsd:element>
    <xsd:element name="if" minOccurs="0">
     <xsd:annotation>
      <xsd:documentation>This is the condition part. It adheres to the action
part and is always stored in the Active Rule repository at a server
side.</xsd:documentation>
     </xsd:annotation>
     <xsd:complexType>
      <xsd:sequence minOccurs="0" maxOccurs="unbounded">
       <xsd:element name="predicate">
        <xsd:annotation>
         <xsd:documentation>If type is SQL, the content of predicate element
is a SELECT statement. Otherwise, type is Value, the content of predicate
element is expressed in term of "variable_name =
expression"</xsd:documentation>
        </xsd:annotation>
        <xsd:complexType>
         <xsd:attribute name="type" use="required">
          <xsd:simpleType>
           <xsd:restriction base="xsd:string">
            <xsd:enumeration value="SQL"/>
            <xsd:enumeration value="Value"/>
           </xsd:restriction>
          </xsd:simpleType>
         </xsd:attribute>
```

Copyright by Mahidol University

```
        </xsd:complexType>
       </xsd:element>
       <xsd:sequence minOccurs="0" maxOccurs="unbounded">
        <xsd:choice>
         <xsd:element name="and">
          <xsd:annotation>
           <xsd:documentation>And operation</xsd:documentation>
          </xsd:annotation>
         </xsd:element>
         <xsd:element name="or">
          <xsd:annotation>
           <xsd:documentation>Or operation</xsd:documentation>
          </xsd:annotation>
         </xsd:element>
        </xsd:choice>
        <xsd:element name="predicate">
         <xsd:annotation>
          <xsd:documentation>If type is SQL, the content of predicate element
is a SELECT statement. Otherwise, type is Value, the content of predicate
element is expressed in term of "variable_name =
expression"</xsd:documentation>
         </xsd:annotation>
         <xsd:complexType>
          <xsd:attribute name="type" use="required">
           <xsd:simpleType>
            <xsd:restriction base="xsd:string">
             <xsd:enumeration value="SQL"/>
             <xsd:enumeration value="Value"/>
            </xsd:restriction>
           </xsd:simpleType>
          </xsd:attribute>
         </xsd:complexType>
        </xsd:element>
       </xsd:sequence>
      </xsd:sequence>
     </xsd:complexType>
    </xsd:element>
    <xsd:element name="action">
     <xsd:annotation>
      <xsd:documentation>This is the action part. It adheres to the condition
part and is always stored in the Active Rule repository at a server
side.</xsd:documentation>
     </xsd:annotation>
     <xsd:complexType>
      <xsd:sequence>
       <xsd:element name="then" maxOccurs="unbounded">
        <xsd:annotation>
         <xsd:documentation>If the condition is satisfied</xsd:documentation>
        </xsd:annotation>
        <xsd:complexType>
         <xsd:sequence>
          <xsd:element name="ActiveWebPage">
           <xsd:annotation>
            <xsd:documentation>This is a reference of an implicit AVML
document</xsd:documentation>
           </xsd:annotation>
           <xsd:complexType>
            <xsd:attribute name="name" type="xsd:string" use="optional"/>
            <xsd:attribute name="ecAppName" type="xsd:string" use="required"/>
            <xsd:attribute name="referrerURI" type="xsd:string"
use="required"/>
            <xsd:attribute name="lastUpdated" type="xsd:string"
use="optional"/>
           </xsd:complexType>
          </xsd:element>
```

Copyright by Mahidol University

```
          </xsd:sequence>
          <xsd:attribute name="referrerPage" type="xsd:string" use="required"/>
         </xsd:complexType>
        </xsd:element>
        <xsd:element name="otherwise" minOccurs="0" maxOccurs="unbounded">
         <xsd:annotation>
          <xsd:documentation>If the condition is not
satisfied</xsd:documentation>
         </xsd:annotation>
         <xsd:complexType>
          <xsd:sequence>
           <xsd:element name="ActiveWebPage">
            <xsd:annotation>
             <xsd:documentation>This is a reference of an implicit AVML
document</xsd:documentation>
            </xsd:annotation>
            <xsd:complexType>
             <xsd:attribute name="name" type="xsd:string" use="optional"/>
             <xsd:attribute name="ecAppName" type="xsd:string" use="required"/>
             <xsd:attribute name="referrerURI" type="xsd:string"
use="required"/>
             <xsd:attribute name="lastUpdated" type="xsd:string"
use="optional"/>
            </xsd:complexType>
           </xsd:element>
          </xsd:sequence>
         </xsd:complexType>
        </xsd:element>
       </xsd:sequence>
       <xsd:attribute name="referrerPage" type="xsd:string" use="required"/>
      </xsd:complexType>
     </xsd:element>
    </xsd:sequence>
    <xsd:attribute name="id" use="required">
     <xsd:annotation>
      <xsd:documentation>this is rule identifier</xsd:documentation>
     </xsd:annotation>
     <xsd:simpleType>
      <xsd:restriction base="xsd:ID">
       <xsd:pattern value="[0-9 A-F]{8}-[0-9 A-F]{4}-[0-9 A-F]{4}-[0-9 A-F]{4}-
[0-9 A-F] {12}"/>
      </xsd:restriction>
     </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="name" type="xsd:string" use="required"/>
    <xsd:attribute name="type" type="xsd:string" use="required"
fixed="remote"/>
   </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

B.4 User Activity Document

Table B-5 depicts the XML Schema structure of a User Activity document that carries all the items submitted by a user thru a web browser.

Copyright by Mahidol University

Table B-5 XML Schema Representation for a User Activity Document

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
 <xsd:element name="activity">
  <xsd:annotation>
   <xsd:documentation>the user activity document collects all items resulting
from a user activity</xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element name="add">
     <xsd:annotation>
      <xsd:documentation>add element collects all items added to Global Data
and/or User Profile</xsd:documentation>
     </xsd:annotation>
     <xsd:complexType>
      <xsd:sequence>
       <xsd:element name="GlobalData">
        <xsd:complexType>
         <xsd:sequence>
          <xsd:any namespace="##any" processContents="skip">
           <xsd:annotation>
            <xsd:documentation>Enter Global Data items submitted by a
user</xsd:documentation>
           </xsd:annotation>
          </xsd:any>
         </xsd:sequence>
        </xsd:complexType>
       </xsd:element>
       <xsd:element name="UserProfile">
        <xsd:complexType>
         <xsd:sequence>
          <xsd:any namespace="##any" processContents="skip">
           <xsd:annotation>
            <xsd:documentation>Enter User Profile items submitted by a
user</xsd:documentation>
           </xsd:annotation>
          </xsd:any>
         </xsd:sequence>
        </xsd:complexType>
       </xsd:element>
      </xsd:sequence>
     </xsd:complexType>
    </xsd:element>
    <xsd:element name="update">
     <xsd:annotation>
      <xsd:documentation>update element collects all items updated Global
Data and/or User Profile</xsd:documentation>
     </xsd:annotation>
     <xsd:complexType>
      <xsd:sequence>
       <xsd:element name="GlobalData">
        <xsd:complexType>
         <xsd:sequence>
          <xsd:any namespace="##any" processContents="skip">
           <xsd:annotation>
            <xsd:documentation>Enter Global Data items submitted by a
user</xsd:documentation>
           </xsd:annotation>
          </xsd:any>
         </xsd:sequence>
```

Copyright by Mahidol University

```
        </xsd:complexType>
       </xsd:element>
       <xsd:element name="UserProfile">
        <xsd:complexType>
         <xsd:sequence>
          <xsd:any namespace="##any" processContents="skip">
           <xsd:annotation>
            <xsd:documentation>Enter User Profile items submitted by a
user</xsd:documentation>
           </xsd:annotation>
          </xsd:any>
         </xsd:sequence>
        </xsd:complexType>
       </xsd:element>
      </xsd:sequence>
     </xsd:complexType>
    </xsd:element>
    <xsd:element name="delete">
     <xsd:annotation>
      <xsd:documentation>delete element collects all items deleted from
Global Data and/or User Profile</xsd:documentation>
     </xsd:annotation>
     <xsd:complexType>
      <xsd:sequence>
       <xsd:element name="GlobalData">
        <xsd:complexType>
         <xsd:sequence>
          <xsd:any namespace="##any" processContents="skip">
           <xsd:annotation>
            <xsd:documentation>Enter Global Data items submitted by a
user</xsd:documentation>
           </xsd:annotation>
          </xsd:any>
         </xsd:sequence>
        </xsd:complexType>
       </xsd:element>
       <xsd:element name="UserProfile">
        <xsd:complexType>
         <xsd:sequence>
          <xsd:any namespace="##any" processContents="skip">
           <xsd:annotation>
            <xsd:documentation>Enter User Profile items submitted by a
user</xsd:documentation>
           </xsd:annotation>
          </xsd:any>
         </xsd:sequence>
        </xsd:complexType>
       </xsd:element>
      </xsd:sequence>
     </xsd:complexType>
    </xsd:element>
    <xsd:element name="params">
     <xsd:annotation>
      <xsd:documentation>params element collects all items submitted to the
AVML Processor along with a service call invocation</xsd:documentation>
     </xsd:annotation>
     <xsd:complexType>
      <xsd:sequence>
       <xsd:any namespace="##any" processContents="skip">
        <xsd:annotation>
         <xsd:documentation>Enter all items submitted by a
user</xsd:documentation>
        </xsd:annotation>
       </xsd:any>
      </xsd:sequence>
```

Copyright by Mahidol University

```
      </xsd:complexType>
     </xsd:element>
    </xsd:sequence>
   </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

B.5 Merged Items Document

Table B-6 depicts the XML Schema structure of a Merged Items document containing all Global Data items and/or User Profile items required to execute the triggered Local Active Rule.

Table B-6 XML Schema Representation for a Merged Items Document

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
 <xsd:element name="merged">
  <xsd:annotation>
   <xsd:documentation>The merged items document is an input of the execution
of a condition-action part of a Local Active Rule.</xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element name="global" minOccurs="0">
     <xsd:annotation>
      <xsd:documentation>There can be any Global Data-related elements within
this tag as needed by the Active Rule execution.</xsd:documentation>
     </xsd:annotation>
     <xsd:complexType>
      <xsd:sequence>
       <xsd:any namespace="##any" processContents="skip">
        <xsd:annotation>
         <xsd:documentation>The Global Data items used in the execution of a
Local Active Rule </xsd:documentation>
        </xsd:annotation>
       </xsd:any>
      </xsd:sequence>
     </xsd:complexType>
    </xsd:element>
    <xsd:element name="UserProfile" minOccurs="0">
     <xsd:annotation>
      <xsd:documentation>There can be any User Profile-related elements
within this tag as needed by the Active Rule execution</xsd:documentation>
     </xsd:annotation>
     <xsd:complexType>
      <xsd:sequence>
       <xsd:any namespace="##any" processContents="skip">
        <xsd:annotation>
         <xsd:documentation>The User Profile items used in the execution of a
Local Active Rule</xsd:documentation>
        </xsd:annotation>
```

Copyright by Mahidol University

```
        </xsd:any>
       </xsd:sequence>
      </xsd:complexType>
     </xsd:element>
    </xsd:sequence>
   </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

B.6 Service Call Parameters Document

Table B-7 depicts the XML Schema structure of a Service Call Parameters document containing all items being passed to the AVML Processor along with a service call invocation.

Table B-7 XML Schema Representation for a Service Call Parameters Document

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
 <xsd:element name="params">
  <xsd:annotation>
   <xsd:documentation>The service call parameters document is an input for
the execution of a condition part of a Remote Active
Rule.</xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
   <xsd:sequence>
    <xsd:any namespace="##any" processContents="skip">
     <xsd:annotation>
      <xsd:documentation>The collection of Global Data items, User Profile
items, and submitted parameters used in the execution of a Remote Active Rule
</xsd:documentation>
     </xsd:annotation>
    </xsd:any>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

Copyright by Mahidol University

APPENDIX C
PERFORMANCE TEST RESULTS

This appendix gives all performance test results from the hands-on experiments between the A^2V prototype and the traditional web applications, as described in Section 6.4 of Chapter VI. The results are grouped by the different test environments which are a single machine, duo machines, and multiple machines.

The following test results are the response times of Active Views construction of the A^2V prototype and the traditional web applications for each session according to the demonstrated case study in Chapter V.

C.1 A Single Machine

1) 1 Session

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
signin	1		2	974		20	660
viewcart			1	903		21	491
shippingaddress				210		20	609
shippingoption				211		20	569
paymentmethod				250		20	579
billingaddress				170		20	599
ordersummary				251		20	590
Total			5	969	2	25	97

Copyright by Mahidol University

2) 2 Concurrent Sessions

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
signin	1		4	86		20	599
viewcart			2	294		20	559
shippingaddress				401		20	580
shippingoption				491		20	560
paymentmethod				460		20	530
billingaddress				531		20	570
ordersummary				261		20	600
Total			8	524	2	23	998
signin	2		4	196		20	610
viewcart			2	404		20	590
shippingaddress				371		20	770
shippingoption				271		20	609
paymentmethod				450		20	529
billingaddress				321		20	690
ordersummary				340		20	680
Total			8	353	2	24	478

3) 4 Concurrent Sessions

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
signin	1		4	66	0	20	589
viewcart			6	89	0	20	620
shippingaddress				781	0	20	550
shippingoption				550	0	20	550
paymentmethod				782	0	20	620
billingaddress				230	0	20	710
ordersummary				280	0	20	940
Total			12	778	2	24	579
signin	2		3	625	0	20	780
viewcart			2	324	0	20	589
shippingaddress				711	0	20	569
shippingoption				571	0	20	569
paymentmethod				872	0	20	559
billingaddress				590	0	20	579
ordersummary			1	92	0	20	650
Total			9	785	2	24	295
signin	3		3	365	0	20	719
viewcart			4	16	0	20	619

Copyright by Mahidol University

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
shippingaddress				791	0	20	560
shippingoption				561	0	20	650
paymentmethod				811	0	20	550
billingaddress				511	0	20	580
ordersummary				481	0	20	789
Total			10	536	2	24	467
signin	4		4	727	0	20	640
viewcart			4	507	0	20	670
shippingaddress			1	41	0	20	569
shippingoption				691	0	20	559
paymentmethod				240	0	20	610
billingaddress				701	0	20	730
ordersummary				751	0	20	920
Total			12	658	2	24	698

4) 5 Concurrent Sessions

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
signin	1		5	888		20	710
viewcart			4	496		20	730
shippingaddress			1	302		20	800
shippingoption				851		20	670
paymentmethod				391		20	700
billingaddress				871		20	940
ordersummary			1	1		21	20
Total			14	800	2	25	570
signin	2		4	216		20	679
viewcart			6	430		20	760
shippingaddress				601		20	840
shippingoption				771		20	830
paymentmethod				390		20	570
billingaddress				661		20	560
ordersummary				551		20	730
Total			13	620	2	24	969
signin	3		4	917		20	630
viewcart			4	686		20	720
shippingaddress			1	211		20	670
shippingoption				861		20	709
paymentmethod				351		20	609
billingaddress			1	191		20	780
ordersummary				811		21	291

Copyright by Mahidol University

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
Total			14	25	2	25	409
signin	4		5	27		20	740
viewcart			5	237		20	690
shippingaddress				580		20	549
shippingoption				360		20	549
paymentmethod				321		20	559
billingaddress				440		20	860
ordersummary				441		20	800
Total			12	406	2	24	747
signin	5		2	704		20	770
viewcart			6	840		20	630
shippingaddress				661		20	810
shippingoption				461		20	560
paymentmethod				270		20	560
billingaddress				181		20	570
ordersummary				270		20	689
Total			11	387	2	24	589

5) 6 Concurrent Sessions

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
signin	1		8	132		20	570
viewcart			6	259		20	580
shippingaddress				901		20	719
shippingoption				601		20	569
paymentmethod			1	172		20	599
billingaddress				771		20	830
ordersummary			1	222		21	171
Total			19	58	2	25	38
signin	2		4	827		20	650
viewcart			4	336		20	800
shippingaddress				291		20	609
shippingoption				310		20	710
paymentmethod				391		20	750
billingaddress				160		20	931
ordersummary				531		20	980
Total			10	846	2	25	430
signin	3		5	758		20	569
viewcart			8	332		20	559
shippingaddress			1	212		20	600
shippingoption				762		20	540

Copyright by Mahidol University

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
paymentmethod				250		20	590
billingaddress			1	212		20	670
ordersummary			1	41		21	170
Total			18	567	2	24	698
signin	4		6	399		20	760
viewcart			8	1		20	780
shippingaddress			1	422		20	670
shippingoption				481		20	630
paymentmethod				370		20	540
billingaddress			1	102		20	630
ordersummary			1	251		20	639
Total			19	26	2	24	649
signin	5		5	117		20	639
viewcart			7	581		20	619
shippingaddress			1	32		20	640
shippingoption				641		20	560
paymentmethod				291		20	570
billingaddress				871		20	569
ordersummary				921		20	679
Total			16	454	2	24	276
signin	6		4	617		20	580
viewcart			7	501		21	10
shippingaddress			1	412		20	740
shippingoption				891		20	540
paymentmethod				221		20	530
billingaddress				501		20	680
ordersummary				260		20	740
Total			15	403	2	24	820

6) 8 Concurrent Sessions

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
signin	1		6	409		20	710
viewcart			5	759		20	780
shippingaddress			1	402			130
shippingoption			1	362			120
paymentmethod				721			90
billingaddress			2	413			30
ordersummary			1	753			301
Total			19	819		44	161
signin	2		8	783		20	850

Copyright by Mahidol University

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
viewcart			14	70		20	780
shippingaddress				871		20	860
shippingoption				601		20	720
paymentmethod				881		20	890
billingaddress				451		20	840
ordersummary				541		21	180
Total			26	198	2	26	120
signin	3		7	771		20	770
viewcart			4	626		20	780
shippingaddress			1	703		20	770
shippingoption			1	242		20	619
paymentmethod				621		20	780
billingaddress			2	193		21	20
ordersummary			2	844		21	161
Total			21	0	2	25	900
signin	4		12	678		20	800
viewcart			6	660		20	779
shippingaddress			1	442		20	790
shippingoption			1	572		21	201
paymentmethod			1	122		20	590
billingaddress			1	442		20	719
ordersummary			2	143			481
Total			27	59	2	5	360
signin	5		10	635		20	640
viewcart			11	687		20	800
shippingaddress			1	462		20	890
shippingoption				601		21	30
paymentmethod				490		20	630
billingaddress				872		20	730
ordersummary				741		21	221
Total			26	488	2	25	941
signin	6		10	595		20	750
viewcart			13	510		20	860
shippingaddress				801		20	710
shippingoption				381		20	649
paymentmethod				751			110
billingaddress				360			150
ordersummary				451			321
Total			26	849	1	23	550
signin	7		7	751		20	590
viewcart			7	361		20	560
shippingaddress			1	812		20	570
shippingoption			1	412		20	560
paymentmethod				420		20	620

Copyright by Mahidol University

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
billingaddress			2	504		20	600
ordersummary			1	622		21	320
Total			22	882	2	24	820
signin	8		3	615		20	900
viewcart			6	739		20	740
shippingaddress			1	692			80
shippingoption			1	161			100
paymentmethod				561			80
billingaddress			1	743			20
ordersummary			1	291			281
Total			16	802		44	201

7) 10 Concurrent Sessions

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
signin	1		9	93		20	610
viewcart			6	319		20	580
shippingaddress			2	273		20	619
shippingoption			1	672		20	620
paymentmethod				541		20	580
billingaddress			3	395		20	720
ordersummary			2	523		21	922
Total			25	816	2	25	651
signin	2		11	526		20	679
viewcart			11	556		20	640
shippingaddress			2	734		20	710
shippingoption				842		20	700
paymentmethod				430		20	599
billingaddress			2	384		20	740
ordersummary			1	832		22	152
Total			31	304	2	26	220
signin	3		9	2		20	770
viewcart			12	899		20	730
shippingaddress			1	882		20	880
shippingoption			1	592			180
paymentmethod			1	232			581
billingaddress			2	133			130
ordersummary			1	292			361
Total			30	32	1	3	632
signin	4		12	568		20	720
viewcart			12	57		20	690

Copyright by Mahidol University

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
shippingaddress			2	914		20	991
shippingoption			2	83			190
paymentmethod			1	522			241
billingaddress			1	583			90
ordersummary			1	242			490
Total			33	969	1	3	412
signin	5		6	760		20	580
viewcart			6	79		20	570
shippingaddress			2	43		20	620
shippingoption			1	51		20	620
paymentmethod			1	893		20	529
billingaddress			1	682		20	569
ordersummary			1	823		21	491
Total			21	331	2	24	979
signin	6		12	107		20	650
viewcart			13	109		20	740
shippingaddress			1	912		20	809
shippingoption			1	302		20	880
paymentmethod			1	532		20	870
billingaddress			2	93			270
ordersummary			1	442			651
Total			33	497	1	44	870
signin	7		17	185		20	609
viewcart			6	820		20	810
shippingaddress			1	623		20	830
shippingoption			1	452		20	810
paymentmethod				741			320
billingaddress			2	924			371
ordersummary			2	3			500
Total			32	748	1	24	250
signin	8		9	243		20	970
viewcart			11	737		20	920
shippingaddress			1	291			240
shippingoption			1	402			411
paymentmethod				881			150
billingaddress			2	484			150
ordersummary			1	101			341
Total			28	139		43	182
signin	9		5	448			381
viewcart			11	867			210
shippingaddress			1	703			160
shippingoption			1	582			110
paymentmethod				271			90
billingaddress			1	782			120

Copyright by Mahidol University

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
ordersummary			1	553			291
Total			24	206		1	362
signin	10		6	610		20	780
viewcart			10	425		20	880
shippingaddress			1	943			350
shippingoption			1	402			251
paymentmethod				360			160
billingaddress			1	923			100
ordersummary			1	502			311
Total			24	165		44	832

8) 12 Concurrent Sessions

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
signin	1		14	180		20	720
viewcart			12	618			821
shippingaddress			1	923			331
shippingoption			1	632			310
paymentmethod			3	155			231
billingaddress			1	81			120
ordersummary			2	344			451
Total			36	933		22	984
signin	2		10	836		20	580
viewcart			16	363		20	750
shippingaddress			1	803		20	700
shippingoption			2	3		21	51
paymentmethod				580		20	660
billingaddress			2	534		21	21
ordersummary			2	113		22	292
Total			35	232	2	27	54
signin	3		13	529		20	599
viewcart			16	394		20	699
shippingaddress			2	664		20	740
shippingoption				931		20	710
paymentmethod				681		20	860
billingaddress			2	63		20	720
ordersummary			1	522		21	772
Total			37	784	2	26	100
signin	4		11	427		20	760
viewcart			18	667		20	630
shippingaddress			1	902		21	110

Copyright by Mahidol University

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
shippingoption			1	272		20	940
paymentmethod				281			291
billingaddress			1	331			651
ordersummary				812		1	31
Total			35	692	1	25	413
signin	5		16	223		20	830
viewcart			15	122		21	211
shippingaddress			2	523		20	600
shippingoption			1	532		20	680
paymentmethod				982		20	880
billingaddress			2	133			240
ordersummary			1	412		1	783
Total			39	927	1	46	224
signin	6		14	891			271
viewcart			11	727			320
shippingaddress			1	903			431
shippingoption			1	562			291
paymentmethod				651			170
billingaddress			1	963			200
ordersummary			2	644			451
Total			35	341		2	134
signin	7		11	607		20	799
viewcart			16	363		20	740
shippingaddress			2	343		20	680
shippingoption			1	292		20	730
paymentmethod				490		20	760
billingaddress			2	424		20	790
ordersummary			1	442		22	412
Total			35	961	2	26	911
signin	8		10	605		20	980
viewcart			9	744			471
shippingaddress			1	852			330
shippingoption			1	532			240
paymentmethod			2	434			251
billingaddress				921			140
ordersummary			2	754			441
Total			29	842		22	853
signin	9		12	808		20	840
viewcart			5	307			651
shippingaddress				601			460
shippingoption				371			301
paymentmethod				551			240
billingaddress				300			221
ordersummary				651			480

Copyright by Mahidol University

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
Total			20	589		23	193
signin	10		16	153			601
viewcart			6	239			321
shippingaddress				701			260
shippingoption				270			281
paymentmethod				821			120
billingaddress				531			140
ordersummary				942			451
Total			25	657		2	174
signin	11		8	482		20	730
viewcart			16	894		21	120
shippingaddress			1	652			801
shippingoption			1	472			351
paymentmethod				471			390
billingaddress			1	352			271
ordersummary			1	1			470
Total			31	324		44	133
signin	12		7	571		21	120
viewcart			19	609			831
shippingaddress			1	462			281
shippingoption				781			230
paymentmethod				470			220
billingaddress				852			211
ordersummary				541			531
Total			31	286		23	424

9) 14 Concurrent Sessions

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
signin	1		14	641		20	910
viewcart			20	960		21	220
shippingaddress			1	852			300
shippingoption			3	946			551
paymentmethod			3	585			221
billingaddress			2	764			691
ordersummary			2	83			811
Total			49	831		44	704
signin	2		15	572		21	60
viewcart			21	981		1	512
shippingaddress			2	714			441
shippingoption			3	84			310

Copyright by Mahidol University

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
paymentmethod			11	457			221
billingaddress			5	778			600
ordersummary				611			461
Total		1	1	197		24	605
signin	3		12	878		21	40
viewcart			10	715		1	412
shippingaddress			2	23			360
shippingoption			2	604			210
paymentmethod			1	72			401
billingaddress			3	384			170
ordersummary			2	344			882
Total			35	20		24	475
signin	4		16	63		20	990
viewcart			18	787		21	31
shippingaddress			1	993		20	920
shippingoption			4	186		1	212
paymentmethod			1	112			741
billingaddress			3	205			140
ordersummary			2	363			711
Total			47	709	1	5	745
signin	5		15	773		20	760
viewcart		1	65	824		20	900
shippingaddress				942		20	890
shippingoption			2	514		21	0
paymentmethod				271		1	362
billingaddress			1	502			270
ordersummary				921			932
Total		1	27	747	1	25	114
signin	6		16	623		20	810
viewcart			10	755			340
shippingaddress			1	953			371
shippingoption				792			941
paymentmethod			1	952			311
billingaddress			1	722			100
ordersummary			3	886			721
Total			37	683		23	594
signin	7		16	273		20	890
viewcart			11	797		21	130
shippingaddress			1	803			360
shippingoption				801			411
paymentmethod			2	123			631
billingaddress			3	375			220
ordersummary			2	764			862
Total			38	936		44	504

Copyright by Mahidol University

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
signin	8		15	231		20	720
viewcart			21	832		20	760
shippingaddress			3	54		20	890
shippingoption			7	791		21	30
paymentmethod				431		20	990
billingaddress			2	664			490
ordersummary				450		1	713
Total			51	453	1	44	593
signin	9		14	641		21	71
viewcart			21	120			651
shippingaddress			2	63		1	291
shippingoption			3	85			220
paymentmethod			5	428			110
billingaddress			5	738			30
ordersummary				992			681
Total			53	67		24	54
signin	10		14	140		20	630
viewcart			10	595		20	740
shippingaddress			2	283		20	720
shippingoption			1	2		20	830
paymentmethod			1	31		21	11
billingaddress			2	344		21	41
ordersummary			2	253		22	52
Total			33	648	2	27	24
signin	11		11	306		20	589
viewcart			12	238		20	699
shippingaddress			1	663		20	580
shippingoption			1	783		20	710
paymentmethod			1	121		20	690
billingaddress			2	945		20	850
ordersummary			2	303		22	282
Total			33	359	2	26	400
signin	12		14	90		20	860
viewcart			9	223		1	252
shippingaddress			1	843			751
shippingoption			1	102			210
paymentmethod			1	512			40
billingaddress			1	372			70
ordersummary			3	505			661
Total			32	647		23	844
signin	13		10	715			300
viewcart			22	752			311
shippingaddress			2	964			260
shippingoption			20	810			561

Copyright by Mahidol University

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
paymentmethod			1	552			50
billingaddress				321			30
ordersummary				391			691
Total			59	505		3	26
signin	14		6	649			391
viewcart			22	823			421
shippingaddress			1	953			801
shippingoption			18	927			260
paymentmethod				401			581
billingaddress			8	402			50
ordersummary			1	352			580
Total		1	0	507		3	84

10) 15 Concurrent Sessions

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
signin	1		17	295		20	700
viewcart			11	807		20	920
shippingaddress			3	676		21	531
shippingoption			1	72			901
paymentmethod				701			301
billingaddress			2	693			380
ordersummary			2	33		1	603
Total			39	277	1	6	336
signin	2		17	75		21	631
viewcart			13	399		21	20
shippingaddress			3	15			361
shippingoption			3	365			270
paymentmethod				871			441
billingaddress			3	555			270
ordersummary			2	224		1	352
Total			43	504		45	345
signin	3		14	471		20	910
viewcart			24	975		20	880
shippingaddress				390		21	291
shippingoption			2	614		21	361
paymentmethod				360		1	82
billingaddress			4	347			220
ordersummary				500		1	432
Total			47	657	1	27	176
signin	4		18	36		20	769

Copyright by Mahidol University

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
viewcart			19	248		20	950
shippingaddress			1	122		21	131
shippingoption			1	592		21	371
paymentmethod			1	622			481
billingaddress			1	112			581
ordersummary			2	143		1	752
Total			44	875	1	27	35
signin	5		17	415			932
viewcart			19	628			721
shippingaddress			1	662			721
shippingoption			1	962			701
paymentmethod			1	853			130
billingaddress				661			211
ordersummary			1	923			661
Total			45	104		4	77
signin	6		16	163		21	141
viewcart			12	608		21	50
shippingaddress			1	62			841
shippingoption			2	333			721
paymentmethod			1	893			211
billingaddress			3	205			230
ordersummary			1	602		1	72
Total			38	866		45	266
signin	7		16	153		21	240
viewcart			21	371			520
shippingaddress				370			170
shippingoption				841			440
paymentmethod			1	42			732
billingaddress				210			290
ordersummary				351			911
Total			40	338		24	303
signin	8		14	60		21	10
viewcart			8	753		20	940
shippingaddress				511		21	270
shippingoption				451			872
paymentmethod				350			180
billingaddress				321			641
ordersummary				931		1	352
Total			25	377	1	6	265
signin	9		14	261		20	930
viewcart			20	289		20	840
shippingaddress			1	943		21	411
shippingoption			1	101			591
paymentmethod				321			290

Copyright by Mahidol University

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
billingaddress			2	23			511
ordersummary			2	203		1	242
Total			42	141	1	5	815
signin	10		16	13		20	610
viewcart			21	831		20	570
shippingaddress			1	693		20	530
shippingoption			4	316		20	540
paymentmethod				581		20	610
billingaddress				411		20	930
ordersummary			2	594		22	272
Total			47	439	2	26	62
signin	11		16	123		1	492
viewcart			23	374			801
shippingaddress			1	182			310
shippingoption			7	952			280
paymentmethod				130			160
billingaddress				270			60
ordersummary				751			691
Total			49	782		3	794
signin	12		15	613		21	60
viewcart			12	618		21	80
shippingaddress			1	963			460
shippingoption			2	103		1	252
paymentmethod			1	222			220
billingaddress			1	952			481
ordersummary			3	285			962
Total			38	756		45	515
signin	13		14	942			991
viewcart			15	552		1	192
shippingaddress			2	253			791
shippingoption			2	944			260
paymentmethod				511			221
billingaddress			2	83			140
ordersummary			1	762			601
Total			40	47		4	196
signin	14		18	526		21	641
viewcart			19	298		1	332
shippingaddress				881			240
shippingoption			1	672			831
paymentmethod			1	723			170
billingaddress			1	602			131
ordersummary			1	993			570
Total			45	695		24	915
signin	15		15	532		1	252

Copyright by Mahidol University

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
viewcart			24	345			711
shippingaddress				521			211
shippingoption			25	977			350
paymentmethod			2	353			130
billingaddress				321			81
ordersummary				650			540
Total		1	9	699		3	275

11) 16 Concurrent Sessions

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
signin	1					20	790
viewcart						1	932
shippingaddress							621
shippingoption							191
paymentmethod							280
billingaddress							170
ordersummary							731
Total						24	715
signin	2					21	261
viewcart						1	2
shippingaddress							841
shippingoption							321
paymentmethod							180
billingaddress							250
ordersummary							932
Total						24	787
signin	3					21	241
viewcart						1	182
shippingaddress							261
shippingoption							210
paymentmethod							601
billingaddress							250
ordersummary							921
Total						24	666
signin	4					21	221
viewcart						20	920
shippingaddress						1	62
shippingoption							791
paymentmethod							210
billingaddress						1	282

Copyright by Mahidol University

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
ordersummary							471
Total						45	957
signin	5					21	181
viewcart							892
shippingaddress							481
shippingoption							671
paymentmethod							651
billingaddress							160
ordersummary							761
Total						24	797
signin	6					21	551
viewcart							601
shippingaddress							571
shippingoption							821
paymentmethod							190
billingaddress							191
ordersummary							701
Total						24	626
signin	7					2	194
viewcart							540
shippingaddress							240
shippingoption							270
paymentmethod						1	212
billingaddress							170
ordersummary							551
Total						5	177
signin	8					20	590
viewcart						20	639
shippingaddress						20	590
shippingoption						20	640
paymentmethod						20	590
billingaddress						20	920
ordersummary						21	991
Total					2	25	960
signin	9					21	130
viewcart						21	140
shippingaddress							981
shippingoption						1	92
paymentmethod							240
billingaddress							180
ordersummary							842
Total						45	605
signin	10					20	780
viewcart						21	120

Copyright by Mahidol University

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
shippingaddress							471
shippingoption							731
paymentmethod							170
billingaddress							601
ordersummary							992
Total						44	865
signin	11					20	610
viewcart						20	580
shippingaddress						20	570
shippingoption						20	580
paymentmethod						20	600
billingaddress						20	629
ordersummary						21	210
Total					2	24	779
signin	12					21	732
viewcart							671
shippingaddress							581
shippingoption						1	202
paymentmethod							751
billingaddress							140
ordersummary							471
Total						25	548
signin	13					21	90
viewcart						20	980
shippingaddress							401
shippingoption							861
paymentmethod							310
billingaddress							281
ordersummary							511
Total						44	434
signin	14					21	251
viewcart							862
shippingaddress						1	272
shippingoption							220
paymentmethod							130
billingaddress							101
ordersummary							871
Total						24	707
signin	15					21	110
viewcart						21	0
shippingaddress						1	71
shippingoption							220
paymentmethod							782
billingaddress							170

Copyright by Mahidol University

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
ordersummary							551
Total						44	904
signin	16					21	141
viewcart						1	292
shippingaddress							591
shippingoption							731
paymentmethod							721
billingaddress							101
ordersummary							881
Total						25	458

12) 18 Concurrent Sessions

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
signin	1					20	830
viewcart						21	40
shippingaddress							381
shippingoption							230
paymentmethod							200
billingaddress							281
ordersummary							180
Total						43	142
signin	2					20	860
viewcart						20	850
shippingaddress							461
shippingoption							300
paymentmethod							170
billingaddress							150
ordersummary							220
Total						43	11
signin	3						941
viewcart						1	1
shippingaddress							992
shippingoption							991
paymentmethod							922
billingaddress						1	1
ordersummary						1	2
Total						6	850
signin	4						951
viewcart							981
shippingaddress							992

Copyright by Mahidol University

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
shippingoption							991
paymentmethod							961
billingaddress							992
ordersummary							981
Total						6	849
signin	5					20	980
viewcart						20	940
shippingaddress						20	699
shippingoption							130
paymentmethod							301
billingaddress							370
ordersummary							721
Total					1	4	141
signin	6						971
viewcart							992
shippingaddress							991
shippingoption							991
paymentmethod							971
billingaddress							972
ordersummary							961
Total						6	849
signin	7					20	830
viewcart						20	850
shippingaddress						21	200
shippingoption							200
paymentmethod							280
billingaddress							561
ordersummary							70
Total					1	3	991
signin	8					1	1
viewcart						1	2
shippingaddress						1	1
shippingoption						1	2
paymentmethod							991
billingaddress							981
ordersummary							992
Total						6	970
signin	9					20	549
viewcart						20	770
shippingaddress						20	609
shippingoption						20	649
paymentmethod						20	610
billingaddress						33	18
ordersummary						21	191

Copyright by Mahidol University

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
Total					2	37	396
signin	10					20	900
viewcart						21	381
shippingaddress							210
shippingoption							120
paymentmethod							250
billingaddress							61
ordersummary							210
Total						43	132
signin	11						961
viewcart						1	22
shippingaddress							981
shippingoption						1	1
paymentmethod							991
billingaddress							972
ordersummary							961
Total						6	889
signin	12						971
viewcart							992
shippingaddress							991
shippingoption							991
paymentmethod						1	1
billingaddress							962
ordersummary							971
Total						6	879
signin	13					20	660
viewcart						33	38
shippingaddress						20	920
shippingoption						20	650
paymentmethod						20	540
billingaddress						20	700
ordersummary						21	762
Total					1	38	270
signin	14					20	800
viewcart						20	890
shippingaddress						21	170
shippingoption							230
paymentmethod							180
billingaddress							251
ordersummary							110
Total					1	3	631
signin	15						971
viewcart						1	41
shippingaddress							952

Copyright by Mahidol University

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
shippingoption							991
paymentmethod						1	51
billingaddress							952
ordersummary							971
Total						6	929
signin	16					20	980
viewcart						20	820
shippingaddress							471
shippingoption							180
paymentmethod							210
billingaddress							141
ordersummary							240
Total						43	42
signin	17					20	620
viewcart						20	710
shippingaddress						20	609
shippingoption						20	549
paymentmethod						20	710
billingaddress						33	68
ordersummary						22	72
Total					2	38	338
signin	18						951
viewcart							992
shippingaddress							991
shippingoption							991
paymentmethod							951
billingaddress						1	1
ordersummary						1	2
Total						6	879

13) 20 Concurrent Sessions

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
signin	1						570
viewcart						1	493
shippingaddress							641
shippingoption						2	554
paymentmethod						1	452
billingaddress						1	422
ordersummary							731
Total						8	863

Copyright by Mahidol University

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
signin	2					20	750
viewcart						1	222
shippingaddress							641
shippingoption							771
paymentmethod						2	173
billingaddress						3	245
ordersummary							691
Total						29	493
signin	3					21	812
viewcart						20	950
shippingaddress						6	870
shippingoption							921
paymentmethod							571
billingaddress							201
ordersummary							551
Total						51	876
signin	4					20	890
viewcart						21	90
shippingaddress						1	963
shippingoption							751
paymentmethod						2	514
billingaddress						2	704
ordersummary							721
Total						50	633
signin	5					21	1
viewcart						21	241
shippingaddress							321
shippingoption						1	292
paymentmethod							671
billingaddress						2	553
ordersummary							952
Total						48	31
signin	6					1	993
viewcart						2	884
shippingaddress						2	323
shippingoption						2	704
paymentmethod							470
billingaddress							211
ordersummary							480
Total						11	65
signin	7					21	911
viewcart						22	182
shippingaddress							430
shippingoption						3	124

Copyright by Mahidol University

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
paymentmethod							341
billingaddress						2	353
ordersummary							882
Total						51	223
signin	8					21	501
viewcart						2	273
shippingaddress							461
shippingoption							981
paymentmethod						2	494
billingaddress						2	463
ordersummary							882
Total						31	55
signin	9					21	331
viewcart						2	353
shippingaddress							911
shippingoption						2	703
paymentmethod							290
billingaddress						1	532
ordersummary						1	612
Total						30	732
signin	10					20	670
viewcart						20	610
shippingaddress						20	629
shippingoption						20	610
paymentmethod						21	20
billingaddress						20	739
ordersummary						21	952
Total					2	26	230
signin	11					20	700
viewcart						21	110
shippingaddress						2	363
shippingoption						2	844
paymentmethod							281
billingaddress						1	562
ordersummary						1	502
Total						50	362
signin	12					1	242
viewcart						2	63
shippingaddress							531
shippingoption							300
paymentmethod						2	244
billingaddress						2	103
ordersummary						1	342
Total						9	825

Copyright by Mahidol University

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
signin	13					20	710
viewcart						21	291
shippingaddress						1	762
shippingoption							771
paymentmethod						2	513
billingaddress							501
ordersummary						2	874
Total						50	422
signin	14					21	50
viewcart						2	33
shippingaddress						2	984
shippingoption						2	494
paymentmethod							340
billingaddress							191
ordersummary							520
Total						29	612
signin	15					21	201
viewcart						2	52
shippingaddress							470
shippingoption							651
paymentmethod							560
billingaddress						2	184
ordersummary						1	782
Total						28	900
signin	16					21	100
viewcart						22	432
shippingaddress							361
shippingoption							681
paymentmethod						2	734
billingaddress							330
ordersummary						3	34
Total						50	672
signin	17					21	120
viewcart						2	464
shippingaddress						2	703
shippingoption							902
paymentmethod							951
billingaddress						1	703
ordersummary							550
Total						30	393
signin	18					20	971
viewcart						3	215
shippingaddress						4	316
shippingoption						1	823

Copyright by Mahidol University

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
paymentmethod							50
billingaddress							60
ordersummary							491
Total						30	926
signin	19						721
viewcart						1	382
shippingaddress							491
shippingoption						3	345
paymentmethod						1	903
billingaddress							560
ordersummary							551
Total						8	953
signin	20					21	160
viewcart						1	923
shippingaddress							591
shippingoption						3	225
paymentmethod						1	752
billingaddress							551
ordersummary							581
Total						29	783

C.2 Duo Machines

1) 1 Session

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
signin	1		2	454		20	600
viewcart			1	942		20	780
shippingaddress				351		20	559
shippingoption				321		20	670
paymentmethod				330		20	710
billingaddress				270		20	720
ordersummary				411		20	730
Total			6	79	2	24	769

Copyright by Mahidol University

2) 2 Concurrent Sessions

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
signin	1		2	754		20	630
viewcart			2	994		20	730
shippingaddress				531		20	920
shippingoption				321		20	730
paymentmethod				380		20	700
billingaddress				251		20	719
ordersummary				510		20	780
Total			7	741	2	25	209
signin	2		4	837		20	630
viewcart			3	705		20	770
shippingaddress				411		20	560
shippingoption				350		20	660
paymentmethod				371		20	710
billingaddress				260		20	729
ordersummary				421		20	710
Total			10	3355	2	24	769

3) 4 Concurrent Sessions

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
signin	1		8	593		20	739
viewcart			7	561		20	800
shippingaddress			1	111		20	569
shippingoption				971		20	720
paymentmethod				391		20	690
billingaddress				951		20	680
ordersummary				701		20	739
Total			20	279	2	24	937
signin	2		8	222		20	640
viewcart			8	722		20	760
shippingaddress			1	272		20	580
shippingoption				871		20	700
paymentmethod				411		20	730
billingaddress			1	182		20	709
ordersummary			1	432		20	740
Total			22	112	2	24	859
signin	3		4	526		20	670
viewcart			4	757		20	790

Copyright by Mahidol University

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
shippingaddress				491		20	569
shippingoption				801		20	760
paymentmethod				361		20	700
billingaddress				861		20	730
ordersummary				791		20	759
Total			12	588	2	24	978
signin	4		5	898		20	640
viewcart			7	50		20	820
shippingaddress			1	311		20	570
shippingoption				671		20	720
paymentmethod				471		20	700
billingaddress			1	71		20	730
ordersummary			1	382		20	730
Total			17	854	2	24	910

4) 5 Concurrent Sessions

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
signin	1		8	452		20	600
viewcart			9	583		20	749
shippingaddress			1	913		20	570
shippingoption			1	142		20	700
paymentmethod				971		20	720
billingaddress			1	192		20	820
ordersummary			1	532		20	970
Total			24	785	2	25	129
signin	2		7	201		20	649
viewcart			11	46		20	740
shippingaddress			1	872		20	569
shippingoption				832		20	720
paymentmethod				590		20	970
billingaddress			1	252		20	790
ordersummary				922		20	700
Total			23	715	2	25	138
signin	3		8	312		20	590
viewcart			7	130		21	0
shippingaddress			1	472		20	560
shippingoption				821		20	649
paymentmethod				901		20	690
billingaddress			2	434		20	720
ordersummary			2	193		20	730

Copyright by Mahidol University

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
Total			23	263	2	24	939
signin	4		9	564		20	629
viewcart			7	731		20	820
shippingaddress			1	32		20	809
shippingoption			1	152		20	640
paymentmethod				801		20	720
billingaddress			1	993		20	740
ordersummary			2	343		20	710
Total			24	616	2	25	68
signin	5		7	30		20	580
viewcart			10	805		20	880
shippingaddress			1	783		20	639
shippingoption			1	512		20	700
paymentmethod				461		20	710
billingaddress			1	222		20	730
ordersummary			1	271		20	720
Total			24	84	2	24	959

5) 6 Concurrent Sessions

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
signin	1		10	345		20	679
viewcart			12	418		20	850
shippingaddress			2	353		20	589
shippingoption			3	24		20	680
paymentmethod				922		20	700
billingaddress				320		20	790
ordersummary				451		20	740
Total			29	833	2	25	28
signin	2		11	286		20	910
viewcart			12	488		20	951
shippingaddress			1	843		20	560
shippingoption			1	752		20	749
paymentmethod				371		20	660
billingaddress			1	152		20	700
ordersummary				590		20	680
Total			29	482	2	25	210
signin	3		12	388		20	580
viewcart			11	897		20	980
shippingaddress				901		20	670
shippingoption			1	743		20	670

Copyright by Mahidol University

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
paymentmethod			1	833		20	700
billingaddress				490		20	739
ordersummary			1	623		20	730
Total			30	875	2	25	69
signin	4		9	413		20	630
viewcart			10	485		20	790
shippingaddress			2	204		20	559
shippingoption			1	192		20	760
paymentmethod			1	812		20	759
billingaddress				681		20	690
ordersummary			1	433		20	730
Total			27	220	2	24	918
signin	5		11	587		20	590
viewcart			12	348		20	920
shippingaddress			2	83		20	760
shippingoption			1	713		20	660
paymentmethod			1	412		20	710
billingaddress				501		20	880
ordersummary			2	353		20	690
Total			31	997	2	25	210
signin	6		7	260		20	580
viewcart			6	349		20	890
shippingaddress				962		20	620
shippingoption				771		20	679
paymentmethod				802		20	720
billingaddress				580		20	710
ordersummary				982		20	700
Total			17	706	2	24	899

6) 8 Concurrent Sessions

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
signin	1		13	239		21	171
viewcart			14	851		20	870
shippingaddress			1	262		20	570
shippingoption			5	37		20	690
paymentmethod				340		20	709
billingaddress			4	677		20	750
ordersummary			1	682			311
Total			41	88	2	5	71
signin	2		21	632		20	870

Copyright by Mahidol University

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
viewcart			51	804		20	730
shippingaddress				571		20	639
shippingoption			2	83		20	690
paymentmethod				751		20	720
billingaddress				450		20	710
ordersummary				711		20	720
Total		1	18	2	2	25	79
signin	3		12	949		20	640
viewcart			15	242		20	790
shippingaddress			1	412		20	560
shippingoption			39	507		20	670
paymentmethod				490		20	740
billingaddress			5	368		20	800
ordersummary				901		21	30
Total		1	15	869	2	25	230
signin	4		15	452		21	91
viewcart			5	37		20	880
shippingaddress				991		20	570
shippingoption				931		20	710
paymentmethod				501		20	720
billingaddress				831		20	750
ordersummary			2	844			290
Total			26	587	2	5	11
signin	5		15	413		20	619
viewcart			36	81		20	750
shippingaddress				671		20	569
shippingoption			3	666		20	699
paymentmethod				450		20	710
billingaddress				972		21	231
ordersummary				621		20	819
Total			57	874	2	25	397
signin	6		13	960		20	610
viewcart			13	439		20	870
shippingaddress			1	312		20	610
shippingoption			4	307		20	690
paymentmethod				420		20	710
billingaddress			1	612		20	799
ordersummary				722		20	770
Total			35	772	2	25	59
signin	7		19	839		20	670
viewcart			11	787			500
shippingaddress				591			50
shippingoption			3	996			150
paymentmethod				230			190

Copyright by Mahidol University

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
billingaddress				661			200
ordersummary				521			221
Total			37	625		21	981
signin	8		13	569			240
viewcart			56	731			421
shippingaddress				501			50
shippingoption			1	832			221
paymentmethod				591			190
billingaddress			1	2			210
ordersummary				741			211
Total		1	14	967		1	543

7) 10 Concurrent Sessions

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
signin	1		19	749		20	840
viewcart		1	86	735		21	60
shippingaddress				781			531
shippingoption			2	914			230
paymentmethod				561			261
billingaddress				310			400
ordersummary				802			331
Total		2	51	852		43	653
signin	2		17	485		20	840
viewcart			17	836		20	980
shippingaddress			3	65			451
shippingoption		1	106	483			300
paymentmethod			33	839			351
billingaddress			3	485			300
ordersummary			3	45			271
Total		4	5	238		43	493
signin	3		18	777		20	579
viewcart						21	531
shippingaddress			7	731			631
shippingoption			2	444			271
paymentmethod				280			410
billingaddress				551			301
ordersummary				731			280
Total						44	3
signin	4		20	600		20	589
viewcart		1	118	240		20	920

Copyright by Mahidol University

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
shippingaddress			14	91		20	589
shippingoption			3	965		20	679
paymentmethod				451		20	910
billingaddress			14	721		20	740
ordersummary			2	23		20	720
Total		3	54	91	2	25	147
signin	5		14	691		20	659
viewcart			10	325		21	80
shippingaddress			1	452		20	560
shippingoption			2	634		20	750
paymentmethod				481		20	710
billingaddress				941		20	980
ordersummary			4	136		20	740
Total			34	660	2	25	479
signin	6		21	321		20	639
viewcart						21	110
shippingaddress			6	279		20	940
shippingoption			1	713		20	680
paymentmethod				430		20	700
billingaddress				471		20	710
ordersummary				501		20	750
Total					2	25	529
signin	7		20	39		20	569
viewcart			26	418		20	780
shippingaddress			6	950		20	570
shippingoption			4	456		20	710
paymentmethod			5	638		20	720
billingaddress			1	782		1	111
ordersummary			23	174			271
Total		1	28	457	1	44	731
signin	8		17	425		21	381
viewcart			15	512		20	890
shippingaddress			1	533		1	212
shippingoption			12	37			170
paymentmethod			2	734			350
billingaddress			2	604			361
ordersummary			6	789			210
Total			58	634		44	574
signin	9		15	492		20	620
viewcart			46	6		20	700
shippingaddress			1	382		20	710
shippingoption			6	179		20	730
paymentmethod			36	282		20	690
billingaddress			6	750			871

Copyright by Mahidol University

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
ordersummary				811			421
Total		1	52	902	1	44	742
signin	10		16	534			231
viewcart			34	149			320
shippingaddress			5	208			50
shippingoption			5	898			171
paymentmethod			2	153			180
billingaddress				971			200
ordersummary			1	862			201
Total		1	6	775		1	353

8) 12 Concurrent Sessions

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
signin	1		38	355		20	570
viewcart						20	870
shippingaddress						20	650
shippingoption						20	740
paymentmethod			13	58		20	750
billingaddress			1	121		20	759
ordersummary			1	1		20	810
Total					2	25	149
signin	2		41	690		20	709
viewcart			44	4		20	730
shippingaddress			3	24		20	559
shippingoption			18	977		20	730
paymentmethod			3	906		20	660
billingaddress			1	863		21	21
ordersummary			4	596		20	770
Total		1	58	60	2	25	179
signin	3		37	964			761
viewcart							581
shippingaddress							40
shippingoption							681
paymentmethod			20	830			220
billingaddress			7	962			261
ordersummary				971			270
Total						2	814
signin	4		37	634			931
viewcart							470
shippingaddress							240

Copyright by Mahidol University

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
shippingoption							531
paymentmethod							200
billingaddress							211
ordersummary							260
Total						2	843
signin	5		35	601		20	930
viewcart			45	85		21	100
shippingaddress			2	925		20	599
shippingoption			13	800		20	760
paymentmethod			3	705		1	102
billingaddress		1	84	691			531
ordersummary			18	16			220
Total		4	23	823	1	25	242
signin	6		36	652		20	619
viewcart			45	466		21	551
shippingaddress			4	717			691
shippingoption			20	560			230
paymentmethod		1	116	137			301
billingaddress			28	972			450
ordersummary			24	85			191
Total		5	36	589		44	33
signin	7		41	730		20	710
viewcart						21	261
shippingaddress							661
shippingoption							250
paymentmethod			59	105			421
billingaddress			1	21			270
ordersummary			1	141			311
Total						43	884
signin	8		39	817		20	620
viewcart						21	531
shippingaddress						20	610
shippingoption						20	680
paymentmethod			15	352		20	700
billingaddress				631			731
ordersummary				952			220
Total					1	45	92
signin	9		35	501		20	630
viewcart			41	129		21	360
shippingaddress			3	946		20	549
shippingoption			19	248		20	719
paymentmethod			3	826		20	700
billingaddress			28	461		21	120
ordersummary			5	177		20	719

Copyright by Mahidol University

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
Total		2	17	288	2	25	797
signin	10		34	820		20	830
viewcart						20	990
shippingaddress						20	570
shippingoption						21	121
paymentmethod						20	690
billingaddress							741
ordersummary							350
Total					1	45	292
signin	11		25	857			340
viewcart			46	66			451
shippingaddress			4	186			100
shippingoption			20	19			270
paymentmethod		1	76	730			260
billingaddress			23	194			161
ordersummary			10	825			210
Total		4	26	877		1	792
signin	12						191
viewcart							310
shippingaddress							40
shippingoption							191
paymentmethod			7	61			190
billingaddress				580			210
ordersummary			2	463			211
Total						1	343

9) 14 Concurrent Sessions

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
signin	1					20	800
viewcart						21	901
shippingaddress						20	850
shippingoption						20	730
paymentmethod						20	679
billingaddress							661
ordersummary							601
Total					1	46	222
signin	2					21	241
viewcart						21	541
shippingaddress							360
shippingoption							621

Copyright by Mahidol University

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
paymentmethod						1	112
billingaddress							270
ordersummary							180
Total						45	325
signin	3					21	471
viewcart						21	441
shippingaddress							941
shippingoption							250
paymentmethod							732
billingaddress							270
ordersummary							300
Total						45	405
signin	4					21	200
viewcart						21	561
shippingaddress						20	570
shippingoption						20	700
paymentmethod						20	990
billingaddress							701
ordersummary							911
Total					1	46	633
signin	5					21	581
viewcart						21	50
shippingaddress							691
shippingoption							511
paymentmethod							190
billingaddress							291
ordersummary							160
Total						44	474
signin	6					20	640
viewcart						22	793
shippingaddress							160
shippingoption							531
paymentmethod							721
billingaddress							431
ordersummary							210
Total						45	486
signin	7					20	660
viewcart						22	352
shippingaddress							581
shippingoption							561
paymentmethod							281
billingaddress							571
ordersummary							410
Total						45	416

Copyright by Mahidol University

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
signin	8					21	331
viewcart						21	511
shippingaddress							570
shippingoption							310
paymentmethod							621
billingaddress							400
ordersummary							191
Total						44	934
signin	9					20	739
viewcart						21	691
shippingaddress							300
shippingoption							431
paymentmethod							170
billingaddress							471
ordersummary							370
Total						44	172
signin	10					21	140
viewcart						21	180
shippingaddress						1	172
shippingoption							551
paymentmethod							241
billingaddress						1	91
ordersummary							441
Total						45	816
signin	11					21	381
viewcart						21	621
shippingaddress							490
shippingoption							340
paymentmethod							611
billingaddress							721
ordersummary							281
Total						45	445
signin	12					20	860
viewcart						21	400
shippingaddress						1	151
shippingoption							280
paymentmethod							361
billingaddress							280
ordersummary							781
Total						45	113
signin	13					21	321
viewcart						22	151
shippingaddress							701
shippingoption							501

Copyright by Mahidol University

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
paymentmethod							220
billingaddress							491
ordersummary							260
Total						45	645
signin	14					21	351
viewcart						21	120
shippingaddress							380
shippingoption							611
paymentmethod							611
billingaddress							220
ordersummary							321
Total						44	614

10) 15 Concurrent Sessions

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
signin	1					20	569
viewcart						21	240
shippingaddress							992
shippingoption							401
paymentmethod							260
billingaddress							291
ordersummary							240
Total						43	993
signin	2					1	573
viewcart							520
shippingaddress							50
shippingoption							180
paymentmethod							181
billingaddress							200
ordersummary							230
Total						2	934
signin	3					20	920
viewcart						1	211
shippingaddress							180
shippingoption							231
paymentmethod							360
billingaddress							561
ordersummary							521
Total						23	984
signin	4					20	759

Copyright by Mahidol University

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
viewcart						20	890
shippingaddress						20	579
shippingoption						20	720
paymentmethod							891
billingaddress							270
ordersummary							231
Total					1	24	340
signin	5					20	579
viewcart						21	430
shippingaddress						1	2
shippingoption							490
paymentmethod							741
billingaddress							181
ordersummary							210
Total						44	633
signin	6					1	572
viewcart						1	482
shippingaddress							530
shippingoption							331
paymentmethod							190
billingaddress							210
ordersummary							201
Total						4	516
signin	7					20	590
viewcart						21	581
shippingaddress							180
shippingoption							361
paymentmethod							220
billingaddress							290
ordersummary							241
Total						43	463
signin	8					20	580
viewcart						20	850
shippingaddress						20	549
shippingoption							931
paymentmethod							150
billingaddress							451
ordersummary							300
Total					1	3	811
signin	9					20	810
viewcart						20	850
shippingaddress						20	570
shippingoption						20	991
paymentmethod						20	750

Copyright by Mahidol University

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
billingaddress						20	890
ordersummary							801
Total					2	5	662
signin	10					20	980
viewcart						1	532
shippingaddress							270
shippingoption							261
paymentmethod							380
billingaddress							621
ordersummary							391
Total						24	435
signin	11					20	719
viewcart						20	780
shippingaddress						20	579
shippingoption						20	760
paymentmethod							861
billingaddress							270
ordersummary							251
Total					1	24	220
signin	12					20	860
viewcart						21	881
shippingaddress							660
shippingoption							531
paymentmethod							331
billingaddress							200
ordersummary							300
Total						44	763
signin	13					20	580
viewcart						20	930
shippingaddress						20	620
shippingoption						20	699
paymentmethod						20	700
billingaddress						20	730
ordersummary						20	680
Total					2	24	939
signin	14					21	101
viewcart						1	542
shippingaddress							50
shippingoption							290
paymentmethod							300
billingaddress							671
ordersummary							551
Total						24	505
signin	15					1	232

Copyright by Mahidol University

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
viewcart							571
shippingaddress							140
shippingoption							691
paymentmethod							210
billingaddress							541
ordersummary							240
Total						3	625

11) 16 Concurrent Sessions

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
signin	1					20	850
viewcart						21	711
shippingaddress						20	810
shippingoption						2	254
paymentmethod							901
billingaddress							591
ordersummary							291
Total					1	7	408
signin	2					20	630
viewcart						21	151
shippingaddress							991
shippingoption						1	542
paymentmethod							641
billingaddress							650
ordersummary							490
Total						46	95
signin	3					20	619
viewcart						22	623
shippingaddress						20	570
shippingoption						20	960
paymentmethod						1	372
billingaddress							852
ordersummary							270
Total					1	27	266
signin	4					21	611
viewcart						21	271
shippingaddress						20	609
shippingoption						2	804
paymentmethod							341
billingaddress							681

Copyright by Mahidol University

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
ordersummary							280
Total					1	7	597
signin	5					20	970
viewcart						20	960
shippingaddress						1	191
shippingoption							361
paymentmethod							411
billingaddress							781
ordersummary							401
Total						45	75
signin	6					20	670
viewcart						21	511
shippingaddress						20	710
shippingoption						1	492
paymentmethod						1	973
billingaddress							531
ordersummary							491
Total					1	7	378
signin	7					21	261
viewcart						2	163
shippingaddress						1	102
shippingoption							761
paymentmethod							942
billingaddress						1	61
ordersummary							681
Total						27	971
signin	8					20	649
viewcart						22	132
shippingaddress						20	580
shippingoption						20	970
paymentmethod						1	1
billingaddress						1	142
ordersummary							340
Total					1	26	814
signin	9					20	590
viewcart						22	472
shippingaddress						1	31
shippingoption							350
paymentmethod						1	522
billingaddress							440
ordersummary							251
Total						46	656
signin	10					20	720
viewcart						22	613

Copyright by Mahidol University

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
shippingaddress						1	422
shippingoption							982
paymentmethod						1	2
billingaddress							230
ordersummary							992
Total						47	961
signin	11					20	970
viewcart						2	413
shippingaddress						1	12
shippingoption							541
paymentmethod							721
billingaddress							771
ordersummary							521
Total						26	949
signin	12					21	10
viewcart						2	163
shippingaddress							501
shippingoption							971
paymentmethod							962
billingaddress							581
ordersummary						1	462
Total						27	650
signin	13					21	450
viewcart						1	933
shippingaddress						1	42
shippingoption							341
paymentmethod						1	272
billingaddress							661
ordersummary						1	51
Total						27	750
signin	14					20	610
viewcart						21	981
shippingaddress							571
shippingoption						1	362
paymentmethod							220
billingaddress							351
ordersummary							782
Total						45	877
signin	15					21	371
viewcart						21	721
shippingaddress							901
shippingoption							932
paymentmethod						1	962
billingaddress							510

Copyright by Mahidol University

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
ordersummary						1	102
Total						48	499
signin	16						170
viewcart							661
shippingaddress							60
shippingoption							351
paymentmethod							140
billingaddress							220
ordersummary							210
Total						1	812

12) 18 Concurrent Sessions

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
signin	1					20	921
viewcart						4	476
shippingaddress							210
shippingoption							271
paymentmethod							360
billingaddress							391
ordersummary							431
Total						27	60
signin	2					1	282
viewcart						1	132
shippingaddress							110
shippingoption						1	572
paymentmethod							271
billingaddress							410
ordersummary							200
Total						4	977
signin	3					20	720
viewcart						21	761
shippingaddress							922
shippingoption						1	121
paymentmethod							741
billingaddress							231
ordersummary							550
Total						46	46
signin	4					21	451
viewcart						1	833
shippingaddress							961

Copyright by Mahidol University

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
shippingoption							691
paymentmethod							441
billingaddress							160
ordersummary							691
Total						26	228
signin	5					20	820
viewcart						2	674
shippingaddress							50
shippingoption							291
paymentmethod							901
billingaddress							541
ordersummary							811
Total						26	88
signin	6					20	800
viewcart						3	185
shippingaddress							50
shippingoption							271
paymentmethod						1	261
billingaddress							370
ordersummary							631
Total						26	568
signin	7					20	589
viewcart						21	632
shippingaddress							831
shippingoption						1	352
paymentmethod							681
billingaddress							831
ordersummary							581
Total						46	497
signin	8					20	670
viewcart						21	110
shippingaddress						20	891
shippingoption							741
paymentmethod						1	483
billingaddress							311
ordersummary							520
Total					1	5	726
signin	9					20	569
viewcart						21	491
shippingaddress						20	950
shippingoption						21	131
paymentmethod							751
billingaddress							992
ordersummary							321

Copyright by Mahidol University

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
Total					1	26	205
signin	10					20	619
viewcart						21	331
shippingaddress							881
shippingoption							451
paymentmethod						1	663
billingaddress							361
ordersummary							681
Total						45	987
signin	11					20	780
viewcart						22	412
shippingaddress							861
shippingoption							461
paymentmethod						1	82
billingaddress							341
ordersummary							420
Total						47	357
signin	12					20	609
viewcart						21	90
shippingaddress						20	570
shippingoption						22	262
paymentmethod							491
billingaddress						1	442
ordersummary							491
Total					1	26	955
signin	13					20	921
viewcart						2	623
shippingaddress							170
shippingoption							371
paymentmethod							901
billingaddress							591
ordersummary							621
Total						26	198
signin	14					20	679
viewcart						2	714
shippingaddress							150
shippingoption							170
paymentmethod							541
billingaddress						1	12
ordersummary							711
Total						25	977
signin	15					20	770
viewcart						21	751
shippingaddress						20	570

Copyright by Mahidol University

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
shippingoption						21	591
paymentmethod						1	993
billingaddress							791
ordersummary							191
Total					1	27	657
signin	16					20	629
viewcart						22	402
shippingaddress						20	570
shippingoption						21	290
paymentmethod						1	71
billingaddress						1	663
ordersummary							681
Total					1	28	306
signin	17					20	619
viewcart						20	810
shippingaddress						20	610
shippingoption						20	760
paymentmethod						20	700
billingaddress						20	730
ordersummary						20	719
Total					2	24	948
signin	18					20	649
viewcart						20	880
shippingaddress						20	559
shippingoption						20	669
paymentmethod						20	710
billingaddress						20	730
ordersummary						20	729
Total					2	24	926

13) 20 Concurrent Sessions

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
signin	1					24	305
viewcart						23	164
shippingaddress						20	570
shippingoption						21	490
paymentmethod						20	650
billingaddress						20	890
ordersummary						20	840
Total					2	31	909

Copyright by Mahidol University

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
signin	2					20	670
viewcart						22	843
shippingaddress						1	543
shippingoption						1	202
paymentmethod							531
billingaddress							560
ordersummary						3	45
Total						50	394
signin	3					6	299
viewcart						21	852
shippingaddress						20	550
shippingoption						20	750
paymentmethod						21	91
billingaddress						21	80
ordersummary						20	700
Total					2	12	322
signin	4					6	590
viewcart						2	584
shippingaddress						21	571
shippingoption						20	910
paymentmethod						20	960
billingaddress						20	750
ordersummary						20	739
Total					1	54	104
signin	5					22	292
viewcart						5	408
shippingaddress						2	284
shippingoption							180
paymentmethod						2	23
billingaddress						20	690
ordersummary						20	930
Total					1	13	807
signin	6					21	622
viewcart						21	310
shippingaddress						20	600
shippingoption						20	679
paymentmethod						20	700
billingaddress						21	671
ordersummary						20	940
Total					2	27	522
signin	7					20	599
viewcart						22	853
shippingaddress						2	674
shippingoption							521

Copyright by Mahidol University

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
paymentmethod							851
billingaddress						1	822
ordersummary						1	361
Total						50	681
signin	8					20	610
viewcart						22	262
shippingaddress						20	870
shippingoption						23	13
paymentmethod							961
billingaddress							491
ordersummary						1	262
Total					1	29	469
signin	9					22	353
viewcart						21	461
shippingaddress						21	751
shippingoption						4	46
paymentmethod						1	21
billingaddress						2	694
ordersummary							311
Total					1	13	637
signin	10					21	10
viewcart						23	694
shippingaddress						20	760
shippingoption						21	451
paymentmethod						2	744
billingaddress						2	93
ordersummary							811
Total					1	32	563
signin	11					20	600
viewcart						22	813
shippingaddress						22	82
shippingoption						21	251
paymentmethod						4	15
billingaddress							791
ordersummary						1	272
Total					1	32	824
signin	12					21	421
viewcart						4	66
shippingaddress						1	933
shippingoption							812
paymentmethod						1	552
billingaddress						2	904
ordersummary						21	120
Total						53	808

Copyright by Mahidol University

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
signin	13					20	640
viewcart						24	846
shippingaddress						4	456
shippingoption							821
paymentmethod							211
billingaddress							170
ordersummary						1	622
Total						52	766
signin	14					26	67
viewcart						2	734
shippingaddress						2	994
shippingoption							491
paymentmethod						1	302
billingaddress						2	554
ordersummary						20	729
Total						56	871
signin	15					21	792
viewcart						21	891
shippingaddress						21	401
shippingoption						4	826
paymentmethod						2	774
billingaddress						1	972
ordersummary							241
Total					1	14	897
signin	16					21	842
viewcart						3	735
shippingaddress						1	422
shippingoption						1	341
paymentmethod						1	613
billingaddress						2	483
ordersummary						21	231
Total						53	667
signin	17					21	110
viewcart						23	794
shippingaddress							812
shippingoption							331
paymentmethod							250
billingaddress						20	820
ordersummary						20	709
Total					1	27	826
signin	18					23	964
viewcart						25	777
shippingaddress						20	560
shippingoption						21	21

Copyright by Mahidol University

Active View	Session#	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
paymentmethod						20	910
billingaddress						20	780
ordersummary						20	770
Total					2	33	782
signin	19					24	14
viewcart						25	36
shippingaddress						20	560
shippingoption						20	790
paymentmethod						21	50
billingaddress						21	521
ordersummary						20	689
Total					2	33	660
signin	20					8	612
viewcart						25	757
shippingaddress						20	810
shippingoption						20	660
paymentmethod						20	940
billingaddress						21	0
ordersummary						20	940
Total					2	18	719

C.3 Multiple Machines

1) 1 Session

Active View	IP Addr. 192.168.9.	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
signin	82	0	2	454		20	600
viewcart		0	1	942		20	780
shippingaddress				351		20	559
shippingoption				321		20	670
paymentmethod				330		20	710
billingaddress				270		20	720
ordersummary				411		20	730
Total			6	79	2	24	769

Copyright by Mahidol University

2) 2 Concurrent Sessions

Active View	IP Addr. 192.168.9.	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
signin	82		1	812		20	870
viewcart			2	13		20	800
shippingaddress				351		20	600
shippingoption				300		20	729
paymentmethod				211		20	710
billingaddress				380		20	700
ordersummary				421		20	779
Total			6	488	2	25	188
signin	143		2	281		20	562
viewcart			2	172		21	63
shippingaddress				375		20	562
shippingoption				343		20	703
paymentmethod				250		20	688
billingaddress				422		21	219
ordersummary				547		20	735
Total			6	390	2	25	532

3) 3 Concurrent Sessions

Active View	IP Addr. 192.168.9.	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
signin	82		2	364		20	610
viewcart			2	213		20	750
shippingaddress				370		20	559
shippingoption				300		20	820
paymentmethod				220		20	710
billingaddress				411		20	710
ordersummary				411		20	710
Total			6	289	2	24	869
signin	143		1	922		20	562
viewcart			2	485		20	687
shippingaddress				390		20	547
shippingoption				344		20	672
paymentmethod				390		20	688
billingaddress				282		20	781
ordersummary				500		20	812
Total			6	313	2	24	749
signin	67		6	489		21	231

Copyright by Mahidol University

Active View	IP Addr. 192.168.9.	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
viewcart			6	570		21	61
shippingaddress			1	132		20	600
shippingoption				961		20	699
paymentmethod				701		20	730
billingaddress			1	332		20	730
ordersummary			1	382		20	960
Total			18	567	2	26	11

4) 4 Concurrent Sessions

Active View	IP Addr. 192.168.9.	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
signin	82		1	833		20	560
viewcart			2	83		20	820
shippingaddress				360		20	560
shippingoption				301		20	680
paymentmethod				210		20	689
billingaddress				401		20	710
ordersummary				400		20	760
Total			5	588	2	24	779
signin	143		1	891		20	563
viewcart			2	438		20	704
shippingaddress				390		20	562
shippingoption				344		20	687
paymentmethod				390		20	672
billingaddress				282		20	782
ordersummary				500		20	766
Total			6	235	2	24	736
signin	67		4	867		20	660
viewcart			6	149		20	740
shippingaddress			1	132		21	972
shippingoption				922		20	680
paymentmethod				731		20	720
billingaddress			1	321		20	810
ordersummary			1	351		20	750
Total			16	473	2	26	332
signin	52		6	159		20	599
viewcart			6	48		20	840
shippingaddress			1	142		20	600
shippingoption				981		20	760
paymentmethod			1	132		20	679

Copyright by Mahidol University

Active View	IP Addr. 192.168.9.	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
billingaddress				901		20	880
ordersummary			1	342		20	790
Total			17	705	2	25	148

5) 5 Concurrent Sessions

Active View	IP Addr. 192.168.9.	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
signin	82		2	183		20	569
viewcart			1	973		20	770
shippingaddress				370		20	710
shippingoption				310		20	679
paymentmethod				341		20	710
billingaddress				270		20	700
ordersummary				421		20	739
Total			5	868	2	24	877
signin	143		2	140		21	15
viewcart			2	250		20	688
shippingaddress				406		20	563
shippingoption				328		20	703
paymentmethod				266		20	766
billingaddress				406		20	719
ordersummary				484		20	719
Total			6	280	2	25	173
signin	67		4	887		22	483
viewcart			6	149		20	780
shippingaddress			1	152		20	600
shippingoption				931		20	730
paymentmethod				701		20	720
billingaddress			1	252		20	870
ordersummary			1	342		20	830
Total			16	414	2	26	13
signin	52		5	227		20	619
viewcart			6	169		20	770
shippingaddress			1	132		21	561
shippingoption				952		20	690
paymentmethod			1	141		20	720
billingaddress				861		20	800
ordersummary			1	342		20	710
Total			16	824	2	25	870
signin	246		2	516		20	625

Copyright by Mahidol University

Active View	IP Addr. 192.168.9.	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
viewcart			2	609		20	719
shippingaddress				438		20	562
shippingoption				437		20	703
paymentmethod				406		20	656
billingaddress				312		21	265
ordersummary				563		20	812
Total			7	281	2	25	342

6) 6 Concurrent Sessions

Active View	IP Addr. 192.168.9.	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
signin	82		1	912		20	590
viewcart			3	225		20	780
shippingaddress				391		20	550
shippingoption				300		20	780
paymentmethod				220		20	749
billingaddress				381		20	730
ordersummary				411		20	710
Total			6	840	2	24	889
signin	143		1	937		20	562
viewcart			2	297		20	687
shippingaddress				406		20	594
shippingoption				344		20	687
paymentmethod				250		20	734
billingaddress				406		20	719
ordersummary				500		20	781
Total			6	140	2	24	764
signin	67		5	418		20	670
viewcart			6	429		21	311
shippingaddress			1	162		20	610
shippingoption				921		20	790
paymentmethod				711		20	700
billingaddress			1	242		20	710
ordersummary			1	302		20	850
Total			17	185	2	25	641
signin	52		4	957		20	700
viewcart			6	179		20	860
shippingaddress			1	171		20	630
shippingoption				941		20	760
paymentmethod			1	132		20	800

Copyright by Mahidol University

Active View	IP Addr. 192.168.9.	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
billingaddress				911		20	900
ordersummary			1	542		20	720
Total			16	833	2	25	370
signin	246		2	78		21	188
viewcart			2	625		20	687
shippingaddress				422		20	578
shippingoption				438		20	672
paymentmethod				421		20	766
billingaddress				329		20	735
ordersummary				531		20	750
Total			6	844	2	25	376
signin	242		2	125		20	593
viewcart			2	203		20	922
shippingaddress				390		20	625
shippingoption				313		20	687
paymentmethod				328		20	672
billingaddress				281		20	718
ordersummary				437		20	859
Total			6	77	2	25	76

7) 7 Concurrent Sessions

Active View	IP Addr. 192.168.9.	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
signin	82		2	123		20	640
viewcart			2	124		20	840
shippingaddress				360		20	639
shippingoption				301		20	640
paymentmethod				230		20	710
billingaddress				391		20	720
ordersummary				420		20	740
Total			5	949	2	24	929
signin	143		2	172		20	578
viewcart			2	281		20	781
shippingaddress				391		20	594
shippingoption				328		20	687
paymentmethod				391		20	703
billingaddress				281		20	687
ordersummary				485		20	734
Total			6	329	2	24	764
signin	67		4	777		21	140

Copyright by Mahidol University

Active View	IP Addr. 192.168.9.	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
viewcart			6	129		20	990
shippingaddress			1	112		21	10
shippingoption				951		20	910
paymentmethod				691		20	740
billingaddress			1	211		20	840
ordersummary			1	773		1	242
Total			16	644	2	6	872
signin	52		4	737		20	660
viewcart			6	148		21	160
shippingaddress			1	132		21	311
shippingoption				942			380
paymentmethod				721			301
billingaddress			1	292			240
ordersummary			1	452			260
Total			16	424	1	4	312
signin	246		2	187		20	578
viewcart			2	453		20	766
shippingaddress				422		20	562
shippingoption				438		20	687
paymentmethod				266			235
billingaddress				454			250
ordersummary				546			234
Total			6	766	1	23	312
signin	242		1	672		20	578
viewcart			2	235		20	797
shippingaddress				375		20	562
shippingoption				312		20	719
paymentmethod				328		20	640
billingaddress				250		21	219
ordersummary				437		20	812
Total			5	609	2	25	327
signin	210		2	953		20	625
viewcart			1	687		20	781
shippingaddress				250		20	641
shippingoption				219		20	672
paymentmethod				266		20	703
billingaddress				188		20	734
ordersummary				312		20	703
Total			5	875	2	24	859

Copyright by Mahidol University

8) 8 Concurrent Sessions

Active View	IP Addr. 192.168.9.	A²V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
signin	82		2	83		20	710
viewcart			2	193		20	740
shippingaddress				360		20	549
shippingoption				301		20	760
paymentmethod				340		20	740
billingaddress				261		20	689
ordersummary				410		20	670
Total			5	948	2	24	858
signin	143		2	47		20	563
viewcart			2	625		20	718
shippingaddress				391		20	547
shippingoption				344		20	687
paymentmethod				375		20	657
billingaddress				281		20	703
ordersummary				500		20	703
Total			6	563	2	24	578
signin	67		5	328		21	201
viewcart			6	319		20	780
shippingaddress			1	172			140
shippingoption				932			271
paymentmethod				711			280
billingaddress			1	261			270
ordersummary			1	342			321
Total			17	65		43	263
signin	52		5	167		20	609
viewcart			6	149			320
shippingaddress			1	142			832
shippingoption				951			310
paymentmethod			1	132			230
billingaddress				841			191
ordersummary			1	362			300
Total			16	744		22	792
signin	246		2	266		20	656
viewcart			2	516		20	765
shippingaddress				422		20	579
shippingoption				422			656
paymentmethod				281			281
billingaddress				438			421
ordersummary				547		1	594
Total			6	892	1	4	952

Copyright by Mahidol University

Active View	IP Addr. 192.168.9.	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
signin	242		1	765		20	578
viewcart			2	250		20	719
shippingaddress				359		20	562
shippingoption				328		20	703
paymentmethod				437		20	703
billingaddress				265		20	687
ordersummary				454		20	687
Total			5	858	2	24	639
signin	210		1	360		20	547
viewcart			2	0		20	703
shippingaddress				250		20	562
shippingoption				219		20	750
paymentmethod				250		20	657
billingaddress				188		20	734
ordersummary				312		20	922
Total			4	579	2	24	875
signin	72		1	515		20	563
viewcart			1	594		20	797
shippingaddress				203		20	562
shippingoption				156		20	672
paymentmethod				203		20	688
billingaddress				125		20	734
ordersummary				235		20	828
Total			3	31	2	24	844

9) 9 Concurrent Sessions

Active View	IP Addr. 192.168.9.	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
signin	82		1	983		20	619
viewcart			1	913		20	820
shippingaddress				360		20	559
shippingoption				321		20	860
paymentmethod				330		20	690
billingaddress				261		20	820
ordersummary				410		20	740
Total			5	578	2	25	108
signin	143		1	954		20	562
viewcart			2	296		20	781
shippingaddress				391		20	547
shippingoption				328		20	657

Copyright by Mahidol University

Active View	IP Addr. 192.168.9.	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
paymentmethod				250		20	718
billingaddress				406		20	781
ordersummary				484		20	718
Total			6	109	2	24	764
signin	67		5	758		21	280
viewcart			6	179			400
shippingaddress			1	152			681
shippingoption				951			291
paymentmethod				701			310
billingaddress			1	211			280
ordersummary			1	402			331
Total			17	354		23	573
signin	52		4	717		20	840
viewcart			6	28		23	374
shippingaddress			1	122			140
shippingoption				941			451
paymentmethod			1	162			250
billingaddress				891			290
ordersummary			1	342			241
Total			16	203		45	586
signin	246		2	500		1	594
viewcart			2	562			328
shippingaddress				438			109
shippingoption				453			219
paymentmethod				281		20	766
billingaddress				438			281
ordersummary				531			250
Total			7	203		23	547
signin	242		1	891		20	594
viewcart			2	953		20	735
shippingaddress				375		20	563
shippingoption				312		20	703
paymentmethod				454		20	704
billingaddress				266		20	813
ordersummary				469		20	922
Total			6	720	2	25	34
signin	210		1	468		20	563
viewcart			1	704		20	734
shippingaddress				250		20	578
shippingoption				219		20	688
paymentmethod				156		20	719
billingaddress				265		20	766
ordersummary				313		20	922

Copyright by Mahidol University

Active View	IP Addr. 192.168.9.	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
Total			4	375	2	24	970
signin	72		4	313		20	578
viewcart			1	312		20	797
shippingaddress				203		20	562
shippingoption				172		20	735
paymentmethod				109		20	687
billingaddress				203		20	860
ordersummary				250		20	719
Total			6	562	2	24	938
signin	230		4	136			170
viewcart			4	647		20	820
shippingaddress				761			100
shippingoption				781		20	700
paymentmethod				852			381
billingaddress				591			210
ordersummary				931			381
Total			12	699		42	762

10) 10 Concurrent Sessions

Active View	IP Addr. 192.168.9.	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
signin	82		1	873		20	619
viewcart			2	3		20	730
shippingaddress				360		20	549
shippingoption				301		20	710
paymentmethod				320		20	740
billingaddress				271		20	679
ordersummary				420		20	720
Total			5	548	2	24	747
signin	143		1	953		20	687
viewcart			2	313		20	719
shippingaddress				391		20	547
shippingoption				344		20	735
paymentmethod				391		20	703
billingaddress				297		20	672
ordersummary				484		20	890
Total			6	173	2	24	953
signin	67		5	128		20	630
viewcart			6	149		20	910
shippingaddress			1	141			150

Copyright by Mahidol University

Active View	IP Addr. 192.168.9.	A^{2}V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
shippingoption				961			341
paymentmethod			1	112			621
billingaddress				961		20	740
ordersummary			1	342		20	950
Total			16	794	1	24	342
signin	52		5	157			701
viewcart			6	129			381
shippingaddress			1	142			190
shippingoption				952			391
paymentmethod			1	131		20	670
billingaddress				852			250
ordersummary			1	332			300
Total			16	695		22	883
signin	246		2	156			656
viewcart			2	906			219
shippingaddress				438		20	563
shippingoption				453		1	313
paymentmethod				422			750
billingaddress				328			938
ordersummary				547			187
Total			7	250		24	626
signin	242		2	31		20	719
viewcart			2	172		20	937
shippingaddress				375		20	906
shippingoption				312		20	859
paymentmethod				453		20	688
billingaddress				266		20	703
ordersummary				484		20	766
Total			6	93	2	25	578
signin	210		1	469		20	563
viewcart			2	890		20	750
shippingaddress				266		20	547
shippingoption				219		20	719
paymentmethod				266		20	688
billingaddress				187		23	546
ordersummary				313		21	125
Total			5	610	2	27	938
signin	72		1	172		20	562
viewcart			1	297		20	782
shippingaddress				187		20	578
shippingoption				172		20	688
paymentmethod				109		20	813
billingaddress				187		20	828

Copyright by Mahidol University

Active View	IP Addr. 192.168.9.	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
ordersummary				234		20	718
Total			3	358	2	24	969
signin	230		4	46			170
viewcart			4	647			321
shippingaddress				771			90
shippingoption				781		1	482
paymentmethod				821			251
billingaddress				581			260
ordersummary				941			270
Total			12	588		2	844
signin	123		3	422		20	593
viewcart			1	875		3	203
shippingaddress				344			63
shippingoption				328			250
paymentmethod				375			219
billingaddress				204			265
ordersummary				343		1	422
Total			6	891		26	15

11) 11 Concurrent Sessions

Active View	IP Address 192.168.9.	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
signin	82		1	862		20	640
viewcart			1	983		20	700
shippingaddress				351		20	560
shippingoption				300		20	690
paymentmethod				321		20	649
billingaddress				260		20	710
ordersummary				431		20	590
Total			5	508	2	24	539
signin	143		2	375		20	547
viewcart			2	250		20	672
shippingaddress				390		20	547
shippingoption				344		20	734
paymentmethod				375			172
billingaddress				282		20	734
ordersummary				484			203
Total			6	500	1	43	609
signin	67		5	188		20	630
viewcart			6	409		20	830

Copyright by Mahidol University

Active View	IP Address 192.168.9.	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
shippingaddress			1	132			221
shippingoption				961			320
paymentmethod			1	182			801
billingaddress				931			231
ordersummary			1	372		1	41
Total			17	175		44	74
signin	52		5	779		1	662
viewcart			6	169			551
shippingaddress			1	161			130
shippingoption				951			251
paymentmethod			1	142			290
billingaddress				851		1	102
ordersummary			1	332		20	780
Total			17	385		24	766
signin	246		3	719			
viewcart			2	593			
shippingaddress				438			
shippingoption				500			
paymentmethod				437			
billingaddress				313			
ordersummary				546			
Total			9	546			
signin	242		1	907		20	875
viewcart			2	188		21	954
shippingaddress				375		20	578
shippingoption				313		20	656
paymentmethod				437		20	719
billingaddress				266		20	719
ordersummary				453		20	750
Total			5	939	2	26	251
signin	210		1	312		20	562
viewcart			3	78		22	219
shippingaddress				250		23	563
shippingoption				219			187
paymentmethod				281			203
billingaddress				203			312
ordersummary				328			188
Total			5	671	1	7	234
signin	72		1	125		20	578
viewcart			1	390		20	687
shippingaddress				188		20	547
shippingoption				172		20	750
paymentmethod				109		20	719

Copyright by Mahidol University

Active View	IP Address 192.168.9.	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
billingaddress				203		20	812
ordersummary				235		20	703
Total			3	422	2	24	796
signin	230		3	585			581
viewcart			4	607			230
shippingaddress				771			90
shippingoption				791			310
paymentmethod				831			261
billingaddress				581			260
ordersummary				941			371
Total			12	107		2	103
signin	123		1	140		20	562
viewcart			2	438		20	750
shippingaddress				343			203
shippingoption				312			203
paymentmethod				438			328
billingaddress				187			266
ordersummary				328			266
Total			5	186		42	578
signin	144		3	203		20	610
viewcart			4	687		21	94
shippingaddress				719		20	578
shippingoption				594		20	704
paymentmethod				438		20	766
billingaddress				937		20	766
ordersummary				875			266
Total			11	453	2	4	784

12) 12 Concurrent Sessions

Active View	IP Address 192.168.9.	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
signin	82		1	943		20	610
viewcart			2	303		20	689
shippingaddress				361		20	560
shippingoption				300		20	670
paymentmethod				221		20	740
billingaddress				380		20	730
ordersummary				431		20	760
Total			5	939	2	24	759
signin	143		2	47		20	594

Copyright by Mahidol University

Active View	IP Address 192.168.9.	A²V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
viewcart			2	469		20	766
shippingaddress				406		20	610
shippingoption				328		20	687
paymentmethod				375		20	687
billingaddress				282		20	734
ordersummary				484		20	735
Total			6	391	2	24	813
signin	67		4	957			360
viewcart			6	289		1	22
shippingaddress			1	132			120
shippingoption				952			301
paymentmethod				731			260
billingaddress			1	251			221
ordersummary			1	342			320
Total			16	654		2	604
signin	52		6	279			501
viewcart			5	999			641
shippingaddress			1	142			121
shippingoption				951			340
paymentmethod				732			240
billingaddress			1	311			281
ordersummary			1	372			290
Total			17	786		2	414
signin	246		2	328			516
viewcart			2	516			546
shippingaddress				422		20	563
shippingoption				437		20	687
paymentmethod				282		20	703
billingaddress				453			359
ordersummary				562			281
Total			7	0	1	3	655
signin	242		1	890		20	578
viewcart			2	469		20	781
shippingaddress				375		20	657
shippingoption				312		20	687
paymentmethod				453		20	703
billingaddress				266		20	719
ordersummary				469		20	766
Total			6	234	2	24	891
signin	210		1	437		20	593
viewcart			2	407		20	750
shippingaddress				250		20	578
shippingoption				219		20	781

Copyright by Mahidol University

Active View	IP Address 192.168.9.	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
paymentmethod				265		20	672
billingaddress				188		20	703
ordersummary				312			219
Total			5	78	2	4	296
signin	72		1	31		20	703
viewcart			1	500		20	766
shippingaddress				188		20	562
shippingoption				156			204
paymentmethod				125			296
billingaddress				203		20	938
ordersummary				235			406
Total			3	438	1	23	875
signin	230		4	516			170
viewcart			4	517			360
shippingaddress				781			90
shippingoption				801			260
paymentmethod				851			241
billingaddress				581			260
ordersummary				942			761
Total			12	989		2	142
signin	123		1	391			62
viewcart			1	781			343
shippingaddress				344			62
shippingoption				312			266
paymentmethod				344			250
billingaddress				203			203
ordersummary				328			281
Total			4	703		1	467
signin	144		5	15		20	610
viewcart			4	360		20	766
shippingaddress				672			407
shippingoption				563			562
paymentmethod				406		3	297
billingaddress				766		20	718
ordersummary				828			219
Total			12	610	1	6	579
signin	231		3	422		21	125
viewcart			3	687		20	735
shippingaddress				578		20	562
shippingoption				500		20	687
paymentmethod				609		20	671
billingaddress				422			329
ordersummary				687			328

Copyright by Mahidol University

Active View	IP Address 192.168.9.	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
Total			9	905	1	44	437

13) 13 Concurrent Sessions

Active View	IP Address 192.168.9.	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
signin	82		2	243		20	610
viewcart			1	913		21	210
shippingaddress				351			80
shippingoption				300			230
paymentmethod				331			181
billingaddress				280		20	709
ordersummary				431			180
Total			5	849	1	3	200
signin	143		2	0		20	563
viewcart			2	578		20	688
shippingaddress				391		20	641
shippingoption				343		20	687
paymentmethod				375		20	703
billingaddress				281		21	391
ordersummary				484		20	672
Total			6	452	2	25	345
signin	67		5	148			310
viewcart			6	399			281
shippingaddress			1	152		1	102
shippingoption				962			761
paymentmethod			1	141			250
billingaddress				861			230
ordersummary			1	392			321
Total			17	55		3	255
signin	52		5	488			671
viewcart			6	9		1	842
shippingaddress			1	132			130
shippingoption				971			571
paymentmethod			1	182			251
billingaddress				851			280
ordersummary			1	322			220
Total			16	955		3	965
signin	246		2	313		2	297
viewcart			5	281		1	891
shippingaddress				437			62

Copyright by Mahidol University

Active View	IP Address 192.168.9.	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
shippingoption				438			219
paymentmethod				265			188
billingaddress				453			218
ordersummary				547			407
Total			9	734		5	282
signin	242		2	156		20	594
viewcart			3	47		21	94
shippingaddress				375		20	562
shippingoption				329		20	719
paymentmethod				437		20	719
billingaddress				265		20	954
ordersummary				453			266
Total			7	62	1	4	908
signin	210		1	437		20	640
viewcart			2	750		20	688
shippingaddress				250		20	593
shippingoption				219		20	672
paymentmethod				281		20	890
billingaddress				187			766
ordersummary				313		1	469
Total			5	437	1	45	718
signin	72		1	157			375
viewcart			1	609		20	875
shippingaddress				219			234
shippingoption				188			156
paymentmethod				125		20	703
billingaddress				203		20	828
ordersummary				250		1	421
Total			3	751	1	4	592
signin	230		4	186			271
viewcart			4	676			220
shippingaddress				771		2	944
shippingoption				791			341
paymentmethod				891			190
billingaddress				581			181
ordersummary				942		1	452
Total			12	838		5	599
signin	123		1	157			78
viewcart			2	969		1	797
shippingaddress				344			46
shippingoption				297		2	219
paymentmethod				438			203
billingaddress				187			172

Copyright by Mahidol University

Active View	IP Address 192.168.9.	A²V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
ordersummary				313		1	407
Total			5	705		5	922
signin	144		3	16		20	578
viewcart			4	672		20	750
shippingaddress				688		1	140
shippingoption				562			484
paymentmethod				406			500
billingaddress				782			266
ordersummary				813			266
Total			10	939		43	984
signin	231		2	969		20	625
viewcart			3	594		20	859
shippingaddress				578		20	578
shippingoption				500		20	735
paymentmethod				360		20	860
billingaddress				656			312
ordersummary				766			203
Total			9	423	1	44	172
signin	201		4	922		20	641
viewcart			4	765		21	281
shippingaddress				844		20	688
shippingoption				687		20	687
paymentmethod				859		20	734
billingaddress				625		20	797
ordersummary			1	0		20	703
Total			13	702	2	25	531

14) 14 Concurrent Sessions

Active View	IP Address 192.168.9.	A²V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
signin	82		2	23		21	41
viewcart			2	113		20	770
shippingaddress				360		20	559
shippingoption				301		20	709
paymentmethod				220		20	690
billingaddress				391		20	710
ordersummary				410		20	940
Total			5	818	2	25	419
signin	143		2	250		20	562
viewcart			2	203		20	704

Copyright by Mahidol University

Active View	IP Address 192.168.9.	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
shippingaddress				391		20	578
shippingoption				343		20	703
paymentmethod				266		20	672
billingaddress				406		20	844
ordersummary				500		20	672
Total			6	359	2	24	735
signin	67		5	257		1	161
viewcart			6	389			461
shippingaddress			1	141			120
shippingoption				951			541
paymentmethod				711			991
billingaddress			1	252			261
ordersummary			1	312			350
Total			17	13		3	885
signin	52		5	37			221
viewcart			6	139			370
shippingaddress			1	131			580
shippingoption				941		2	264
paymentmethod			1	152		20	770
billingaddress				841			311
ordersummary			1	332			300
Total			16	573		24	816
signin	246		2	516			687
viewcart			2	703		1	594
shippingaddress				437			203
shippingoption				437			297
paymentmethod				422			266
billingaddress				313			203
ordersummary				562		20	688
Total			7	390		23	938
signin	242			547		20	609
viewcart			1	437		20	782
shippingaddress				203		23	672
shippingoption				219			188
paymentmethod				328			266
billingaddress				250			203
ordersummary				297			281
Total			3	281	1	6	1
signin	210		1	344		20	547
viewcart			5	46		20	734
shippingaddress				250		20	547
shippingoption				218		20	734
paymentmethod				266		20	719

Copyright by Mahidol University

Active View	IP Address 192.168.9.	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
billingaddress				187		20	782
ordersummary				313		20	766
Total			7	624	2	24	829
signin	72		2	688		20	593
viewcart			1	328		20	719
shippingaddress				203		20	563
shippingoption				172		28	47
paymentmethod				109			187
billingaddress				203			203
ordersummary				235		1	422
Total			4	938	1	31	734
signin	230		3	585			180
viewcart			4	697			271
shippingaddress				771			290
shippingoption				781			341
paymentmethod				491			250
billingaddress				871			371
ordersummary				942		1	412
Total			12	138		3	115
signin	123		1	265		20	578
viewcart			1	828			204
shippingaddress				328			63
shippingoption				328			203
paymentmethod				406			203
billingaddress				188			156
ordersummary				313			250
Total			4	656		21	657
signin	144		3	281		20	610
viewcart			4	375		21	297
shippingaddress				688		20	641
shippingoption				562			329
paymentmethod				406			406
billingaddress				782			344
ordersummary				796			328
Total			10	890	1	3	955
signin	231		2	875		20	640
viewcart			4	0		24	187
shippingaddress				562			140
shippingoption				500			328
paymentmethod				640			343
billingaddress				422			297
ordersummary				703			172
Total			9	702		46	107

Copyright by Mahidol University

Active View	IP Address 192.168.9.	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
signin	201		4	157		20	579
viewcart			4	781		20	985
shippingaddress				843		20	578
shippingoption				704		20	734
paymentmethod				547		20	704
billingaddress				906		20	766
ordersummary				969		20	719
Total			12	907	2	25	65
signin	154		2	297			79
viewcart			1	969			94
shippingaddress				359			297
shippingoption				313			47
paymentmethod				234			250
billingaddress				250			187
ordersummary				578			203
Total			6	0		1	78

15) 15 Concurrent Sessions

Active View	IP Address 192.168.9.	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
signin	82		2	163		20	649
viewcart			3	756		20	700
shippingaddress				360		20	550
shippingoption				301		20	710
paymentmethod				320			240
billingaddress				270			761
ordersummary				421		1	632
Total			7	591	1	25	242
signin	143		2	343		20	578
viewcart			2	969		20	672
shippingaddress				390		20	578
shippingoption				344		20	672
paymentmethod				375		20	735
billingaddress				282		20	703
ordersummary				500		21	172
Total			7	203	2	25	110
signin	67		5	107			841
viewcart			6	399			591
shippingaddress			1	162		2	183
shippingoption				951			381

Copyright by Mahidol University

Active View	IP Address 192.168.9.	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
paymentmethod			1	142		20	779
billingaddress				851			260
ordersummary			1	342			371
Total			16	954		25	406
signin	52		4	757			551
viewcart			6	29			501
shippingaddress			1	131			180
shippingoption				971			320
paymentmethod			1	132			281
billingaddress				851		20	740
ordersummary			1	332			381
Total			16	203		22	954
signin	246		2	188		2	204
viewcart			2	672			500
shippingaddress				422			641
shippingoption				437			422
paymentmethod				266			406
billingaddress				438			328
ordersummary				547			313
Total			6	970		4	814
signin	242		2	329		20	578
viewcart			2	968		20	734
shippingaddress				375		20	563
shippingoption				313		20	703
paymentmethod				453		20	734
billingaddress				266		20	843
ordersummary				438		21	219
Total			7	142	2	25	374
signin	210		1	922		20	625
viewcart			3	47		20	703
shippingaddress				250		20	829
shippingoption				219			219
paymentmethod				265			282
billingaddress				188			203
ordersummary				328			907
Total			6	219	1	3	768
signin	72		1	484			125
viewcart			3	360			250
shippingaddress				188			188
shippingoption				156			219
paymentmethod				109			203
billingaddress				203			219
ordersummary				234		1	938

Copyright by Mahidol University

Active View	IP Address 192.168.9.	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
Total			5	734		3	142
signin	230		4	216			271
viewcart			4	646		3	705
shippingaddress				762		20	590
shippingoption				791			310
paymentmethod				831			280
billingaddress				581			271
ordersummary				941			260
Total			12	768		25	687
signin	123		1	359			360
viewcart			1	657			547
shippingaddress				344			219
shippingoption				344			594
paymentmethod				406		2	578
billingaddress				187			203
ordersummary				313		1	312
Total			4	610		5	813
signin	144		3	453		20	594
viewcart			4	703		1	281
shippingaddress				672		1	656
shippingoption				563			484
paymentmethod				391			375
billingaddress				781		1	297
ordersummary				797			188
Total			11	360		25	875
signin	231		3	641		1	485
viewcart			4	47		1	454
shippingaddress				578			641
shippingoption				485			219
paymentmethod				360			188
billingaddress				640			343
ordersummary				719		20	672
Total			10	470		25	2
signin	201		4	250		20	609
viewcart			5	671		20	750
shippingaddress				859		20	718
shippingoption				703		20	766
paymentmethod				531		20	735
billingaddress				906		20	781
ordersummary				969		21	15
Total			13	889	2	25	374
signin	154		1	750			453
viewcart			2	609		20	563

Copyright by Mahidol University

Active View	IP Address 192.168.9.	A^2V			Trad. Web Apps.		
		min.(s)	sec.(s)	ms.(s)	min.(s)	sec.(s)	ms.(s)
shippingaddress				359			421
shippingoption				312			46
paymentmethod				235			188
billingaddress				265			188
ordersummary				578			203
Total			6	108		21	609
signin	221		1	602		20	700
viewcart			1	612		20	690
shippingaddress				261		20	649
shippingoption				210		20	580
paymentmethod				451		20	640
billingaddress				380		20	570
ordersummary				381		21	271
Total			3	897	2	25	100

Copyright by Mahidol University

BIOGRAPHY

NAME Mr. Chainarong Kongchoke

DATE OF BIRTH 3 November 1977

PLACE OF BIRTH Bangkok, Thailand

INSTITUTIONS ATTENDED Thammasat University, 1995-1998:
Bachelor of Science (Computer Science)

Mahidol University, 2000-2006:
Master of Science (Computer Science)

POSITION&OFFICE 1999-2005, Technology and Software Solution Co., Ltd.
195 Empire Tower 12th Floor Unit 1214, South Sathorn Road, Yannawa, Sathorn, Bangkok, Thailand.
Position : Senior System Analyst

2005-Present, INFOMAX System Solutions & Services Co., Ltd.
195 Empire Tower 12th Floor Unit 1214, South Sathorn Road, Yannawa, Sathorn, Bangkok, Thailand.
Position : Senior System Analyst

Copyright by Mahidol University