FARAMARZ SAFI ESFAHANI DOCTOR OF PHILOSOPHY 2011

ADAPTABLE DECENTRALIZED ORCHESTRATION ENGINE FOR BLOCK STRUCTURED NON-TRANSACTIONAL WORKFLOW IN SERVICE ORIENTED ARCHITECTURE

FARAMARZ SAFI ESFAHANI

DOCTOR OF PHILOSOPHY

UNIVERSITI PUTRA MALAYSIA

2011

ADAPTABLE DECENTRALIZED ORCHESTRATION ENGINE FOR BLOCK STRUCTURED NON-TRANSACTIONAL WORKFLOW IN SERVICE ORIENTED ARCHITECTURE

By

FARAMARZ SAFI ESFAHANI

Thesis Submitted to the School of Graduate Studies, Universiti Putra Malaysia, in Fulfillment of the Requirements for the Degree of Doctor of Philosophy

April 2011

Abstract of thesis presented to the Senate of Universiti Putra Malaysia in fulfillment of the requirement for the degree of Doctor of Philosophy

ADAPTABLE DECENTRALIZED ORCHESTRATION ENGINE FOR BLOCK STRUCTURED NON-TRANSACTIONAL WORKFLOW IN SERVICE ORIENTED ARCHITECTURE

By

FARAMARZ SAFI ESFAHANI

April 2011

Chairman: Masrah Azrifah Azmi Murad, PhD

Faculty: Computer Science and Information Technology

In the Service Oriented Architecture (SOA), business processes are executed by non-scalable centralized orchestration engines. Nonetheless proliferation of business process applications in organizations raises scalability requirements. Decentralized orchestration engines are applied to address the scalability by decentralizing a process into design-time static fragments without considering runtime requirements. The fragments are then encapsulated into runtime components such as agents.

The SOA orchestration layer suffers from the lack of adaptability with runtime environment in decentralization of business processes. Accordingly, three aspects of runtime adaptability in decentralization are studied in this thesis. The first aspect is *frequent-path* adaptability, which is equal to detecting closely-interrelated activities and encapsulating them in the same fragment. Another aspect is *proportional-fragment* adaptability, which is analogous to the proportionality of produced fragments with number of machines. The last aspect is *available-bandwidth*

adaptability, which is process fragmentation based on current circumstances of communication media.

An ever-changing runtime environment along with the mentioned adaptability aspects raises the following research problems: 1) there is no framework to support architectures, decentralization methods, and a feedback loop from runtime environment; 2) current decentralization methods do not consider the frequent-path and proportional-fragment adaptability aspects in creating fragments; 3) there is no method to map runtime circumstances to a suitable decentralization method in order to satisfy the available-bandwidth adaptability. Accordingly, the following research objectives are considered: first, to propose a framework including architectures, decentralization methods, and a feedback loop from runtime environment; second, to improve response-time and throughput of decentralized business processes applying the frequent-path and proportional-fragment adaptability aspects; third, to improve bandwidth-usage of decentralized business processes applying the available-bandwidth adaptability.

The contributions of this research are also as follows: i) An Adaptable and Decentralized Workflow Execution Framework (ADWEF) is introduced that proposes an abstraction of a runtime adaptable decentralization in the SOA orchestration layer; ii) two architectures Type-1 and Type-2 are presented for the ADWEF that are able to support the execution of dynamically created fragments; iii) three aspects of runtime adaptability in decentralization namely frequent-path, proportional-fragment and available-bandwidth are introduced; iv) two decentralization methods called Hierarchical Process Decentralization (HPD) and Hierarchical and Intelligent Process Decentralization (HIPD) are presented, which are capable of providing various fragments. The latter considers the frequent-path

adaptability and both of them together satisfy both frequent-path and proportional-fragment adaptability aspects; v) A Fuzzy Decentralization Decision Making unit (FDDM) is presented based on the fuzzy logic to choose a suitable method of decentralization that satisfies the three adaptability aspects frequent-path, proportional-fragment and available-bandwidth; and, vi) an algorithm is introduced for wiring of dynamic fragments.

Evaluations of the three adaptability aspects in the ADWEF demonstrate that the frequent-path adaptability greatly improves response-time, throughput, and bandwidth-usage of decentralized business processes. The proportional-fragment adaptability proves that number of fragments must be proportional to the number of workflow engines machines. The available-bandwidth adaptability which is realized by the FDDM unit unifies the mentioned adaptability aspects and reduces the number of exchanged messages compared to other methods.

Abstrak tesis yang dikemukakan kepada Senat Universiti Putra Malaysia sebagai memenuhi keperluan untuk ijazah Doktor Falsafah

PENGAGIHAN BOLEH SUAI ENJIN ORKESTRASI UNTUK ALIR KERJA BERSTRUKTUR BLOK TANPA URUS NIAGA DI DALAM SENI BINA BERASASKAN SERVIS

Oleh

FARAMARZ SAFI ESFAHANI

April 2011

Pengerusi: Masrah Azrifah Azmi Murad, PhD

Fakulti: Sains Komputer dan Teknologi Maklumat

Dalam Seni Bina Berasaskan Servis (SOA), proses niaga tertentu BPEL dilaksanakan oleh enjin orkestrasi terpusat yang ketidakboleh skala. Perkembangan aplikasi proses niaga dalam organisasi meningkatkan keperluan skalabiliti. Enjin orkestrasi teragih diterapkan untuk mengatasi skalabiliti dengan mengagihkan proses BPEL menjadi serpihan statik pada waktu reka bentuk tanpa mempertimbangkan keperluan masa jalanan yang tidak mempunyai maklumat pada masa reka bentuk tersebut. Serpihan-serpihan kemudian dikemas ke dalam komponen masa jalanan seperti agen.

Lapisan orkestrasi SOA menderita daripada kekurangan kebolehsuaian dengan persekitaran masa jalanan di dalam pengagihan proses niaga. Oleh kerana itu, tiga aspek kebolehsesuaian masa jalanan dalam pengagihan dipelajari di dalam tesis ini. Aspek pertama adalah kebolehsuaian laluan-kerap bertujuan untuk mengesan aktiviti yang relevan dan melingkupinya dalam serpihan yang sama. Aspek lain adalah

kebolehsuaian perkadaran serpihan yang analog dengan perkadaran serpihan yang dihasilkan dengan jumlah mesin. Aspek yang terakhir adalah kebolehsuaian dengan lebar jalur yang tersedia di mana proses serpihan adalah berdasarkan keadaan semasa bagi media komunikasi.

Persekitaran masa jalanan yang selalu berubah seiring dengan aspek kebolehsuaian yang disebutkan menimbulkan beberapa masalah kajian berikut: i) tiada rangka kerja untuk menyokong seni bina, kaedah pengagihan dan gelung maklum balas daripada sekitaran masa jalanan; ii) kaedah pengagihan terkini tidak mempertimbangkan aspek-aspek perkadaran serpihan dan laluan-kerap di dalam pembentukan serpihan-serpihan; dan iii) tiada unit untuk memetakan keadaan masa jalanan kepada kaedah pengagihan yang sesuai untuk memuaskan kebolehsuaian lebar jalur yang tersedia. Untuk menyelesaikan masalah-masalah yang dinyatakan di atas, beberapa objektif kajian berikut dipertimbangkan: pertama, untuk mencadangkan satu rangka kerja bagi menyokong pelaksanaan penghasilan serpihan-serpihan secara dinamik; kedua, untuk menambahbaik masa sambutan dan daya pemprosesan bagi proses niaga teragih dengan mengaplikasikan aspek-aspek kebolehsuaian laluan-kerap dan perkadaran serpihan; dan akhir sekali, untuk menambahbaik penggunaan lebar jalur proses niaga teragih dengan mengaplikasikan kebolehsuaian lebar jalur yang tersedia.

Selain daripada itu, sumbangan kajian ini adalah seperti berikut: i) satu rangka kerja pelaksanaan alir kerja teragih dan boleh suai (ADWEF) diperkenalkan yang mencadangkan suatu peniskalaan bagi pengagihan boleh suai masa jalanan dalam lapisan orkestrasi SOA; ii) dua seni bina Jenis-1 dan Jenis-2 digambarkan untuk ADWEF yang menyokong pelaksanaan penghasilan serpihan-serpihan secara dinamik; iii) tiga aspek kebolehsuaian masa jalanan dalam pengagihan termasuk

laluan-kerap, perkadaran serpihan dan lebar jalur yang tersedia diperkenalkan; iv) dua kaedah pengagihan HPD dan HIPD dibentangkan yang mampu menyediakan pelbagai serpihan. Kaedah pengagihan yang kedua mempertimbangkan kebolehsuaian laluan-kerap dan kedua-dua kaedah memuaskan aspek-aspek kebolehsuaian laluan-kerap dan perkadaran serpihan; v) satu unit pembuat keputusan pengagihan kabur (FDDM) dibentangkan untuk memilih kaedah yang sesuai bagi pengagihan yang dapat memuaskan aspek-aspek laluan-kerap, perkadaran serpihan dan lebar jalur yang tersedia; dan vi) satu unit diperkenalkan untuk pendawaian serpihan dinamik.

Penilaian terhadap tiga aspek kebolehsuaian di dalam ADWEF menunjukkan bahawa kebolehsuaian laluan-kerap meningkatkan masa sambutan, daya pemprosesan dan penggunaan lebar jalur bagi proses niaga teragih. Kebolehsuaian perkadaran serpihan membuktikan bahawa jumlah serpihan mestilah berkadar dengan jumlah enjin alir kerja. Kebolehsuaian lebar jalur tersedia yang direalisasikan oleh unit FDDM menyatukan aspek kebolehsuaian tersebut dan mengurangkan jumlah pertukaran mesej jika dibandingkan dengan kaedah yang lain.

ACKNOWLEDGEMENTS

First and foremost, my sincere gratitude and thanks to God for giving me the strength, patience, courage and determination in completing this work.

I would like to express my sincere thankfulness to my supervisor Dr. Masrah Azrifah Azmi Murad for giving me an opportunity to commence this project. Through the course of my study, I have had the great fortune to get to know and interact with her. Her comments and supports for further development as well as her assistance during writing this thesis are invaluable to me. Her teachings also provided me an opportunity to do research in my favorite area.

I would like to express my sincere thanks and appreciation to the supervisory committee members Associate Professor Dr. Md. Nasir B. Sulaiman and Dr. Nur Izura Udzir for their guidance, valuable suggestions and advice throughout this work in making this a success.

I cordially appreciate my wife Ms. Sanaz Gilani for her beautiful love and my warmest gratefulness for her support, patience and encouragement that paved the way for finishing my study. I also owe my parents for their kindness, love, encouragement and affirmation, which made it possible for me to achieve this work. My deepest gratitude is to them for devoting their life to their children. Finally, I appreciate my father and mother in low for their infinite patience and support during this study.

APPROVAL

I certify that a Thesis Examination Committee met on **28th April 2011** to conduct the final examination of **Faramarz Safi Esfahani** on his **Doctor of Philosophy** thesis entitled "**Adaptable Decentralized Orchestration Engine For Block-Structured And Non-Transactional Workflows In Service Oriented Architecture**" in accordance with Universiti Pertanian Malaysia (Higher Degree) Act 1980 and Universiti Pertanian Malaysia (Higher Degree) Regulations 1981. The Committee recommends that the candidate be awarded the relevant degree. Members of the Examination Committee were as follows:

Madya Dr. Ramlan Mahmod, PhD
Associate Professor
Faculty of Computer Science and Information Technology
Universiti Putra Malaysia
(Chairman)

Abdul Azim Abd Ghani, PhD
Associate Professor
Faculty of Computer Science and Information Technology
Universiti Putra Malaysia
(Internal Examiner)

Rodziah Atan, PhD
Senior Lecturer
Faculty of Computer Science and Information Technology
Universiti Putra Malaysia
(Internal Examiner)

Schahram Dustdar, PhD
Full Professor
Information Systems Institute
Vienna University of Technology
Austria
(External Examiner)

BUJANG KIM HUAT, PhD
Professor and Deputy Dean
School of Graduate Studies
Universiti Putra Malaysia

Date:

This thesis was submitted to the Senate of Universiti Putra Malaysia and has been accepted as fulfillment of the requirement for the degree of Doctor of Philosophy. The members of the Supervisory Committee were as follows:

Masrah Azrifah Azmi Murad, PhD
Senior Lecturer
Faculty of Computer Science and Information Technology
Universiti Putra Malaysia
(Chairman)

Md. Nasir B. Sulaiman, PhD
Associate Professor
Faculty of Computer Science and Information Technology
Universiti Putra Malaysia
(Member)

Nur Izura Udzir, PhD
Senior Lecturer
Faculty of Computer Science and Information Technology
Universiti Putra Malaysia
(Member)

HASANAH MOHD GHAZALI, PhD
Professor and Dean
School of Graduate Studies
Universiti Putra Malaysia

Date:

DECLARATION

I declare that the thesis is my original work except for quotations and citations which have been duly acknowledged. I also declare that it has not been previously, and is not concurrently, submitted for any other degree at University Putra Malaysia or other institution.

FARAMARZ SAFI ESFAHANI

Date: 28 April 2011

TABLE OF CONTENTS

ABSTRACT iii
ABSTRAK vi

ACKNOWLEDGEMENTS ix

APPROVAL x

TABLE OF CONTENTS xiii

LIST OF TABLES xvi

Table Page xvi

LIST OF FIGURES xviii

Figure Page xviii

LIST OF ABBREVIATIONS xxiii

1 INTRODUCTION 1
- 1.1 Motivation and Background 1
- 1.2 Problem Statement 5
- 1.3 Hypotheses 6
- 1.4 Research Objectives 6
- 1.5 Research Scope and Constraints 8
- 1.6 Research Contributions 10
- 1.7 Organization of Thesis 12
- 1.8 Summary 13

2 BACKGROUND 14
- 2.1 Introduction 14
- 2.2 Web Service Definitions 14
- 2.3 Web Services Architectures 15
- 2.4 Service Composition, Choreography and Orchestration 18
- 2.5 Business Process Execution Language (BPEL) 27
- 2.6 Summary 35

3 LITERATURE REVIEW 36
- 3.1 Introduction 36
- 3.2 Open-world Software Paradigm 36
- 3.3 Business Process Fragmentation, Enactment and Adaptability 40
  - 3.3.1 Centralized Process Fragmentation 42
  - 3.3.2 Static Fragmentation Applying Third-Party Middleware Enactment 42
  - 3.3.3 Dynamic Fragmentation Based on Workflow Circumstances 51
  - 3.3.4 Static Decentralization Using Process Embedded Communication Facilities 54
  - 3.3.5 Compiler Based Decentralization Approach 59
  - 3.3.6 Comparison of Most Relevant Decentralization Methods 62
- 3.4 Runtime Reconfiguration of Business Processes 62
- 3.5 Using Mining Approach in Business Process Decentralization 64
  - 3.5.1 Process Mining and Frequent Path Detection 64

3.6 Overview of Fuzzy Logic 65
3.7 Summary 68

4 RESEARCH METHODOLOGY 69
4.1 Introduction 69
4.2 Research Overview 69
4.3 Research Steps 70
4.4 Referenced Business Processes 75
4.5 Experiment Platform 75
4.5.1 Applying a Multi-agent System to Implement ADWEF 75
4.5.2 WADE/JADE Multi-agent Platform 76
4.6 Experimental Setup 78
4.7 Runtime Evaluation Metrics of Workflows 79
4.7.1 Bandwidth-usage 80
4.7.2 Response-time 80
4.7.3 Throughput 80
4.7.4 Adaptability 81
4.8 Model Verification and Validation 82
4.8.1 Boundary-condition Experiments 82
4.8.2 Frequent-path and Proportional-fragment Experiments 87
4.8.3 Fuzzy Decentralization Decision Making Unit Experiments 94
4.9 Summary 98

5 ADAPTABLE DECENTRALIZED WORKFLOW EXECUTION FRAMEWORK (ADWEF) 100
5.1 Introduction 100
5.2 Adaptable Decentralized Workflow Execution Framework 100
5.3 Adaptable SOA Orchestration Layer 103
5.4 ADWEF Architectures 106
5.4.1 ADWEF Architecture Formalization 106
5.4.2 ADWEF Architecture Types 109
5.5 DPDC Implementation Policies 113
5.6 Hierarchical Process Decentralization (HPD) 113
5.7 Hierarchical Intelligent Process Decentralization (HIPD) 116
5.7.1 HIPD-Based DPDC 117
5.7.2 HIPD Formalization 119
5.7.3 HIPD-Based DPDC Algorithms 121
5.7.4 HIPD Decentralization of Loan-process 125
5.8 Wiring of Fragments in ADWEF 127
5.9 Creating Dynamic Fragments Based on WADE/JADE Facilities 130
5.9.1 Relations of Workflow Decentralization Methods 131
5.9.2 Decentralizing Block-structured vs. Graph-structured BPEL 133
5.10 Adaptable Decentralization Decision Making Unit 134
5.10.1 Runtime-adaptable Decentralization 134
5.10.2 Fuzzy Decentralization Decision Making Unit (FDDM) 136
5.11 Miscellaneous Aspects of ADWEF Architecture 138
5.12 Summary 139

6 RESULTS AND DISCUSSION 140

6.1 Introduction 140
6.2 Fragmenting Basic and Complex Business Processes 140
6.3 Boundary-condition Experiments 141
6.3.1 Experiment-1 (evaluating response-time, variable-request-rate, constant-message-size) 141
6.3.2 Experiment-2 (evaluating throughput, variable-request-rate, constant-message-size) 143
6.3.3 Experiment-3 (evaluating response-time, constant-request-rate, variable-message-size) 144
6.3.4 Experiment-4 (evaluating throughput, constant-request-rate, variable-message-size) 145
6.3.5 Experiment-5 (evaluating bandwidth-usage, variable-request-rate, constant-message-size) 147
6.3.6 Boundary-condition Evaluation Discussion 148
6.4 Frequent-path and Proportional-fragment Evaluation Experiments 148
6.4.1 If-process Experiment-6 to Experiment-10 150
6.4.2 While-process Experiment-6 to Experiment-10 162
6.4.3 Flow-process Experiment-6 to Experiment-10 174
6.4.4 Loan-process Experiment-6 to Experiment-13 187
6.4.5 Frequent-path and Proportional-fragment Evaluation Discussion 225
6.5 Available-bandwidth Experiments 226
6.5.1 Experiment-14 (evaluating available-bandwidth adaptability, HPD decentralization, random bandwidth) 227
6.5.2 Experiment-15 (evaluating available-bandwidth adaptability, HPD decentralization, exponential bandwidth) 229
6.5.3 Experiment-16 (evaluating available-bandwidth, HPD/HIPD decentralization, random bandwidth) 232
6.5.4 Experiment-17 (evaluating available-bandwidth adaptability, HPD/HIPD decentralization, exponential bandwidth) 235
6.5.5 Available-bandwidth Evaluation Discussion 237
6.6 Discussion 239
6.7 Summary 240

7 SUMMARY, CONCLUSION AND RECOMMENDATIONS FOR FUTURE RESEARCH 241
7.1 Research Summary 241
7.2 Conclusion 243
7.3 Recommendations for Future Research 243

REFERENCES 245

APPENDIX A 255

BIO DATA OF STUDENT 263

LIST OF PUBLICATIONS 264

LIST OF TABLES

Table	Page
2.1 Summary of BPEL Activities	28
3.1 Brief Comparison of Most Relevant Business Process Decentralization Methods	61
4.1 Experiment-1 Configurations	85
4.2 Experiment-2 Configurations	86
4.3 Experiment-3 Configurations	86
4.4 Experiment-4 Configurations	86
4.5 Experiment-5 Configurations	86
4.6 Experiment-6 Configurations	90
4.7 Experiment-7 Configurations	90
4.8 Experiment-8 Configurations	91
4.9 Experiment-9 Configurations	91
4.10 Experiment-10 Configurations	92
4.11 Experiment-11 Configurations	92
4.12 Experiment-12 Configurations	93
4.13 Experiment-13 Configurations	93
4.14 Experiment-14 Configurations	97
4.15 Experiment-15 Configurations	97
4.16 Experiment-16 Configurations	98
4.17 Experiment-17 Configurations	98
6.1 Average Response-time Improvement of If-process Fragments Compared to FPD	153
6.2 Average Throughput Improvement of If-process Fragments Compared to FPD	155
6.3 Average Response-time Improvement of If Fragments Compared to FPD	158
6.4 Average Throughput Improvement of If Fragments Compared to FPD	161
6.5 Average Response-time Improvement of While Fragments Compared to FPD	166
6.6 Average Throughput Improvement of While Fragments Compared to FPD	168
6.7 Average Response-time Improvement of While Fragments Compared to FPD	170
6.8 Average Throughput Improvement of While Fragments Compared to FPD	173
6.9 Average Response-time Improvement of Flow Fragments Compared to FPD	178
6.10 Average Throughput Improvement of Flow Fragments Compared to FPD	180

6.11 Average Response-time Improvement of Flow Fragments Compared to FPD 183

6.12 Average Throughput Improvement of Flow Fragments Compared to FPD 185

6.13 Average Response-time Improvement of Loan Fragments Compared to FPD 189

6.14 Average Throughput Improvement of Loan Fragments Compared to FPD 192

6.15 Average Response-time Improvement of Loan Fragments Compared to FPD 195

6.16 Average Throughput Improvement of Loan Fragments Compared to FPD 198

6.17 Average Percentage of Response-time Improvement by Loan Fragments in Architecture Type-1 203

6.18 Average Percentage of Throughput Improvement by Loan Fragments in Architecture Type-1 204

6.19 Average Percentage of Response-time Improvement by Loan Fragments in Architecture Type-2 207

6.20 Average Percentage of Throughput Improvement by Loan Fragments in Architecture Type-2 208

6.21 Average Percentage of FPD Response-time Improvement by HPD and HIPD methods, Type-1, Constant-Request-Rate, Variable-Message-Size 213

6.22 Loan Average Percentage of FPD Response-time Improvement, by HPD and HIPD, Type-2, Constant-Request-Rate, Variable-Message-Size 216

6.23 Average Percentage of FPD Throughput Improvement, by HPD and HIPD methods, Type-1, Constant-Request-Rate, Variable-Message-Size 221

6.24 Average Percentage of FPD Throughput Improvement, by HPD and HIPD methods, Type-2, Constant-Request-Rate, Variable-Message-Size 224

6.25 Average Exchange Message Improvement, HPD, Random Bandwidth 229

6.26 Average Exchange Message Improvement, HPD, Exponential Bandwidth 231

6.27 Average Exchange Message Improvement, HPD/HIPD, Random Bandwidth 234

6.28 Average Exchange Message Improvement, HPD/HIPD, Exponential Bandwidth 237

6.29 Average FPD Exchange Message Improvement 238

LIST OF FIGURES

Figure **Page**

Figure 1.1: Scope of Research 7
Figure 2.1: WSDL Compilation to Stubs and Skeletons (Alonso et al., 2004) 15
Figure 2.2: WSDL Service Specification (Alonso et al., 2004) 16
Figure 2.3: Supplier Process Specified by Activity Diagram (Alonso et al., 2004) 20
Figure 2.4: Supplier Process Specified by Statechart (Alonso et al., 2004) 20
Figure 2.5: Activity Hierarchy Specification of Supplier Process (Alonso et al., 2004) 21
Figure 2.6: Supplier BPEL Process Specified by BPEL (Alonso et al., 2004) 30
Figure 2.7: Definition of Partners in BPEL (Alonso et al., 2004) 33
Figure 2.8: Exception Handling in BPEL (Alonso et al., 2004) 34
Figure 2.9: Instance Routing using Correlation (Alonso et al., 2004) 35
Figure 3.1: Layers and Aspects in Service-based Applications (Nitto et al., 2008) 38
Figure 3.2: Three Aspects of Workflow Decentralization 41
Figure 3.3: NINOS Distributed Business Process Execution Architecture 49
Figure 3.4: Dynamic Fragmentation of A Sample Process 52
Figure 3.5: Sugeno Style Rule Evaluation 67
Figure 3.6: Sugeno-Style Defuzzification 67
Figure 4.1: Research Steps Highlights 71
Figure 4.2: Three Basic Processes, One Complex Process 74
Figure 4.3: Experiment Environment Configuration 78
Figure 4.4: Four Boundary-condition Configurations 82
Figure 4.5: Evaluation Method of Boundary-condition Configurations 84
Figure 4.6: Evaluation Method of Frequent-path and Proportional-fragment Adaptability 88
Figure 4.7: Evaluation Method of Available-bandwidth Adaptability 95
Figure 5.1: Adaptable Decentralized Workflow Execution Framework (ADWEF) 101
Figure 5.2: Traditional Decentralization 102
Figure 5.3: Runtime Based Decentralization 103
Figure 5.4: Recommended Abstraction for SOA Orchestration Layer 105
Figure 5.5: ADWEF Architecture Setup 108
Figure 5.6: Architecture Type-1 110
Figure 5.7: Implementing Architecture Type-1 Using WADE/JADE Facilities 110
Figure 5.8: Architecture Type-2 112
Figure 5.9: Implementing Architecture Type-2 Using WADE/JADE Facilities 113
Figure 5.10: Hierarchical Process Decentralization for Blocked-structured Processes 114
Figure 5.11: HPD Decentralization of Loan Application Business Process 115
Figure 5.12: Hierarchical Intelligent Process Decentralization (HIPD) 116
Figure 5.13: DPDC Structure for HIPD 117
Figure 5.14: Sequential Path Detection Algorithm 122
Figure 5.15: HIPD Fragmentation Algorithm 122
Figure 5.16: Node Frequency Calculation Algorithm 123
Figure 5.17: List of Nodes in *Level-n* 123
Figure 5.18: Determining Fragments Based on Frequent/Infrequent Paths 124
Figure 5.19: Loan Application Process after Calculating Frequency of Nodes 125
Figure 5.20: HIPD Decentralization of Loan Application Business Process 126
Figure 5.21: Workflow Fragment Building Sections 128
Figure 5.22: Wiring of Fragments by Traversing Original Business Process 129
Figure 5.23: Decentralization Policies Relations 132

Figure 5.24: Choosing a Suitable Decentralization Method Based on BPEL Type 133
Figure 5.25: Three Adaptability Aspects Meet Together 135
Figure 5.26: Adaptable Decentralization Decision Making Unit 135
Figure 5.30: Fuzzy Decentralization Decision Making Unit 137
Figure 6.1: Boundary–condition Experiment, Response-time Comparison , Variable-Request-Rate, Constant-Message-Size 142
Figure 6.2: Boundary-condition Experiment, Throughput Comparison, Variable-Request-Rate, Constant-Message-Size 143
Figure 6.3: Boundary-condition Experiment, Response-time Comparison, Constant-Request-Rate, Variable-Message-Size 145
Figure 6.4: Boundary-condition Experiment, Throughput Comparison, Constant-Request-Rate, Variable-Message-Size 146
Figure 6.5: Boundary-condition Experiment, Comparison of Exchanged Messages Number 147
Figure 6.6: If-process Decentralization Patterns 151
Figure 6.7: If-process Response-time Comparison, Type-1, Variable-Request-Rate, Constant-Message-Size, Min-Max distributions 152
Figure 6.8: If Response-time Comparison, Type-2, Variable-Request-Rate, Constant-Message-Size, Min-Max distributions 153
Figure 6.9: Type-1 and Type-2, If-process Response-time Improvement Comparison 153
Figure 6.10: Type-1 If-process Throughput Comparison in Min-Max distributions 155
Figure 6.11: Type-2 If-process Throughput Comparison in Min-Max distributions 155
Figure 6.12: Type-1 and Type-2, Throughput Improvement Comparison 155
Figure 6.13: If-process Response-time Comparison, Type-1, Constant-Request-Rate, Variable-Message-Size, Min-Max distributions 156
Figure 6.14: If-process Response-time Comparison, Type-2, Constant-Request-Rate, Variable-Message-Size, Min-Max distributions 157
Figure 6.15: Type-1 and Type-2, If-process Response-time Improvement with Request Rates 50 and 500 158
Figure 6.16: If-process Throughput Comparison, Type-1, Constant-Request-Rate, Variable-Message-Size, Min-Max distributions 160
Figure 6.17: If-process Throughput Comparison, Type-2, Constant-Request-Rate, Variable-Message-Size, Min-Max distributions 160
Figure 6.18: Type-1 and Type-2, If-process Throughput Improvement with Request Rates 50 and 500 161
Figure 6.19: Exchanged Message Comparison of If-process Fragments 162
Figure 6.20: While-process Decentralization Patterns 163
Figure 6.21: While-process Response-time Comparison, Type-1, Variable-Request-Rate, Constant-Message-Size, Min-Max distributions 164
Figure 6.22: While-process Response-time Comparison, Type-2, Variable-Request-Rate, Constant-Message-Size, Min-Max distributions 165
Figure 6.23: Type-1 and Type-2, While-process Response-time Improvement Comparison 165
Figure 6.24: While-process Throughput Comparison, Type-1 , Variable-Request-Rate, Constant-Message-Size, Min-Max distributions 167
Figure 6.25: While-process Throughput Comparison, Type-2, Variable-Request-Rate, Constant-Message-Size, Min-Max distributions 167
Figure 6.26: Type-1 and Type-2, While-process Throughput Improvement Comparison 167
Figure 6.27: While-process Response-time Comparison, Type-1, Constant-Request-Rate, Variable-Message-Size, Min-Max distributions 169
Figure 6.28: While-process Response-time Comparison, Type-2, Constant-Request-Rate, Variable-Message-Size, Min-Max distributions 169
Figure 6.29: Type-1 and Type-2, While-process Response-time Improvement with Request Rates 50 and 500 170

Figure 6.30: While-process Throughput Comparison, Type-1, Constant-Request-Rate, Variable-Message-Size, Min-Max distributions 171
Figure 6.31: While-process Throughput Comparison, Type-2, Constant-Request-Rate, Variable-Message-Size in Min-Max distributions 172
Figure 6.32: Type-1 and Type-2, While-process Throughput Improvement with Request Rates 50 and 500 172
Figure 6.33: Exchanged Message Comparison of While-process Fragments 174
Figure 6.34: Flow-process Decentralization Patterns 175
Figure 6.35: Flow-process Response-time Comparison, Type-1, Variable-Request-Rate, Constant-Message-Size, Min-Max distributions 177
Figure 6.36: Flow-process Response-time Comparison, Type-2, Variable-Request-Rate, Constant-Message-Size, Min-Max distributions 177
Figure 6.37: Type-1 and Type-2, Flow-process Response-time Improvement Comparison 178
Figure 6.38: Flow-process Throughput Comparison, Type-1, Variable-Request-Rate, Constant-Message-Size, Min-Max distributions 179
Figure 6.39: Flow-process Throughput Comparison, Type-2, Variable-Request-Rate, Constant-Message-Size, Min-Max distributions 179
Figure 6.40: Type-1 and Type-2, Flow-process Throughput Improvement Comparison 180
Figure 6.41: Flow-process Response-Time Comparison, Type-1, Constant-Request-Rate, Variable-Message-Size, Min-Max distributions 181
Figure 6.42: Flow-process Response-Time Comparison, Type-2, Constant-Request-Rate, Variable-Message-Size, Min-Max distributions 182
Figure 6.43: Type-1 and Type-2, Flow-process Response-time Improvement with Request Rates 50 and 500 182
Figure 6.44: Flow-process Throughput Comparison, Type-1, Constant-Request-Rate, Variable-Message-Size, Min-Max distributions 184
Figure 6.45: Flow-process Throughput Comparison, Type-2, Constant-Request-Rate, Variable-Message-Size, Min-Max distributions 184
Figure 6.46: Type-1 and Type-2, Flow-process Throughput Improvement with Request Rates 50 and 500 185
Figure 6.47: Exchanged Message Comparison of Flow-process Fragments 186
Figure 6.48: Loan-process Response-time Comparison, Type-1, Variable-Request-Rate, Constant-Message-Size, Min-Max distributions 188
Figure 6.49: Loan-process Response-time Comparison, Type-2, Variable-Request-Rate, Constant-Message-Size, Min-Max distributions 189
Figure 6.50: Type-1 and Type-2, Loan-process Response-time Improvement 189
Figure 6.51: Loan-process Throughput Comparison, Type-1, Variable-Request-Rate, Constant-Message-Size, Min-Max distributions 190
Figure 6.52: Loan-process Throughput Comparison, Type-2, Variable-Request-Rae, Constant-Message-Size, Min-Max distributions 191
Figure 6.53: Type-1 and Type-2, Loan-process Throughput improvement 191
Figure 6.54: Loan-process Response-Time Comparison, Type-1, Constant-Request-Rate, Variable-Message-Size, Min-Max distributions 193
Figure 6.55: Loan-process Response-Time Comparison, Type-2, Constant-Request-Rate, Variable-Message-Size, Min-Max distributions 194
Figure 6.56: Type-1 and Type-2, Loan-process Response-time Improvement with Request Rates 50 and 500 194
Figure 6.57: Loan-process Throughput Comparison, Type-1, Constant-Request-Rate, Variable-Message-Size, in Min-Max distributions 196
Figure 6.58: Loan-process Throughput Comparison, Type-2, Constant-Message-Size, Variable-Message-Sizes, Min-Max distributions 197
Figure 6.59:Type-1 and Type-2, Loan-process Throughput Improvement, Variable-Message-Size, Constant-Request-Rate 50 and 500 197

Figure 6.60: Exchanged Message Comparison of Loan-process Fragments 198
Figure 6.61: Fragment Proportionality of Loan fragments in terms of Response-time and Throughput, Type-1, Variable-Request-Rate, Constant-Message-Size 202
Figure 6.62: Average Response-time Improvement, Type-1, Variable-Request-Rate, Constant-Message-Size 203
Figure 6.63: Average Throughput Improvement, Type-1, Variable-Request-Rate, Constant-Message-Size 204
Figure 6.64: Loan-process Fragment Proportionality in terms of Response-time and Throughput, Type-2, Variable-Request-Rate, Constant-Message-Size 206
Figure 6.65: Average Response-time Improvement in Architecture Type-2, Variable-Request-Rate, Constant-Message-Size 207
Figure 6.66: Average Throughput Improvement in Architecture Type-2, Variable-Request-Rate, Constant-Message-Size 208
Figure 6.67: Loan-process Fragment Proportionality in terms of Response-time, Type-1, Constant-Request-Rate, Variable-Message-Size 212
Figure 6.68: Average Percentage of FPD Response-time Improvement by HPD and HIPD, Type-1, with Constant-Request-Rate, Variable-Message-Size 213
Figure 6.69: Fragment Proportionality of Loan Fragments in terms of Response-time, Type-2, Constant-Request-Rate, Variable-Message-Size 215
Figure 6.70: Average Percentage of FPD Response-time Improvement by HPD and HIPD, Type-2, Constant-Request-Rate, Variable-Message-Size 216
Figure 6.71: Fragment Proportionality of Loan Application Process in terms of Throughput, Type-1 with Constant-Request-Rate, Variable-Message-Size 220
Figure 6.72: Average Percentage of FPD Throughput Improvement, by HPD and HIPD methods, Type-1, Constant-Request-Rate, Variable-Message-Size 221
Figure 6.73: Fragment Proportionality of Loan Application Process in terms of Throughput, Type-2, Constant-Request-Rate, Variable-Message-Size 223
Figure 6.74: Average Percentage of FPD Throughput Improvement, by HPD and HIPD, Type-2, Constant-Request-Rate, Variable-Message-Size 224
Figure 6.75: Reaction of FDDM unit to Fragment Proportionality, Random Bandwidth, HPD Decentralization 227
Figure 6.76: FPD and FDDM Number of Exchanged Message Comparison, Considering Fragment Proportionality, Random Bandwidth, HPD Decentralization 228
Figure 6.77: Exchanged Message Improvement of FPD, Applying HPD, Random Bandwidth 229
Figure 6.78: Reaction of FDDM Unit to Exponential Bandwidth and HPD Decentralization 230
Figure 6.79: FPD and FDDM Number of Exchanged Message Comparison, Exponential Bandwidth and HPD Decentralization 230
Figure 6.80: Exchanged Message Improvement of FPD Applying HPD and Exponential Bandwidth 231
Figure 6.81: Reaction of FDDM unit to Random Bandwidth, HPD and HIPD Decentralization 232
Figure 6.82: FPD and FDDM Number of Exchanged Message Comparison, Random Bandwidth, HPD and HIPD Decentralization 233
Figure 6.83: Exchanged Message Improvement of FPD Applying HPD/HIPD, Random Bandwidth 234
Figure 6.84: Reaction of FDDM Unit to Exponential Bandwidth, HPD and HIPD Decentralization 235
Figure 6.85: FPD and FDDM Number of Exchanged Message Comparison, Considering Exponential Bandwidth, HPD and HIPD Decentralization 236
Figure 6.86: Exchanged Message Improvement of FPD Applying HPD/HIPD, Exponential Bandwidth 237
A.1 Using If Activity in Two Methods 262
A.2 Using While Activity in two Methods 264
A.3 Using Flow Activity in Two Methods 264

A.4 Using Pick Activity in Two methods 266
A.5 Scope, Compensate and Fault 267

LIST OF ABBREVIATIONS

Phrase	Description
2PC	Two Phase Commitment protocol is used to coordinate components involved in a transaction.
ACID	A transaction is a piece of code that has four fundamental features including Atomicity, Consistency, Isolation, and Durability.
ADWEF	An Adaptable and Decentralized Workflow Execution Framework presented in this thesis to support dynamic execution, decentralization methods, a feedback loop from runtime environment to be considered in business process decentralization.
Available-bandwidth adaptability	The ability of a fragmentation method in adapting the created fragments with the available bandwidth.
Basic agent (Basic WADE /JADE agent)	It is an agent created in WADE/JADE environment that has only basic functionalities of agents including movement, and ability to execute workflow metaphors. It is also called Micro-engine, Performer agent, and workflow engine agent (WFE).
Basic Flow-process	It is a *basic process* that its central complex activity is a Flow activity.
Basic fragment	It is a fragment that includes one or two activity which is able to call another activity inside the same fragment or another fragment.
Basic If-process	It is a basic process that its central complex activity is an If activity.
Basic process (Basic BPEL process)	In this thesis three business processes If-process, While-process, and Flow-process are introduced in which there is only one complex activity If, While, and Flow, respectively. The main goal of introducing such processes is to study the behavior of their complex activity.
Basic While-process	It is a basic process that its central complex activity is a While activity.
BPEL	Business Process Execution Language.

Phrase	Description
BPMN	Business Process Modeling Notation (BPMN) provides businesses with the capability of understanding their internal business procedures in a graphical notation and gives organizations the ability to communicate these procedures in a standard manner.
Business Process	Groups of activities which are related together using a standard descriptive language such as BPEL.
Closely-Interrelated Activities	It refers to activities of a business process, which are frequently executed together.
Complex process	In this thesis the loan application business process is called Complex process because it contains several simple as well as complex activities.
CORBA	Common Object Request Broker Architecture (CORBA) enables separate pieces of software written in different languages and running on different computers to work with each other like a single application or set of services.
FDDM	Fuzzy Decentralization Decision Making is an unit introduced in this thesis to implement three aspects of adaptability including the frequent-path, proportional-fragment, and available-bandwidth.
FPD	Fully Process Decentralization is a method of decentralizing a business process to its building activity-level fragments.
Frequent-path adaptability	An adaptability aspect that encapsulates high relevant, juxtaposed, closely interrelated activities of a workflow in one fragment.
FTP	File Transfer Protocol (FTP) is a standard network protocol used to copy a file from one host to another over a TCP-based network, such as the Internet.
HIPD	Hierarchical and Intelligent Process Decentralization, which at first detects frequently executed paths of a business process applying a mining approach and then fragments the frequent paths using the HPD decentralization method.
HPD	Hierarchical Process Decentralization method, which decomposes a block-structured business process based on the levels of a business process tree, hierarchically.

Phrase	Description
HTTP	Hyper Text Transfer Protocol (HTTP) is the underlying protocol used by the World Wide Web. It defines how messages are formatted and transmitted, and what actions Web servers and browsers should take in response to various commands.
IDL	Interface Description Language or alternatively, Interface Definition Language (IDL) is a specification language used to describe a software component's interface. IDLs describe an interface in a language-neutral way, enabling communication between software components that do not share a language.
JADE	Java Agent DEvelopment Framework (JADE) is a software Framework fully implemented in Java language. It simplifies the implementation of multi-agent systems through a middle-ware and through a set of graphical tools that supports the debugging and deployment phases.
Linda	Linda is a model of coordination and communication among several parallel processes operating upon objects stored in and retrieved from shared, virtual, associative memory.
Process fragment	Set of activities, which are grouped together to make a logical unit called fragment.
Proportional-fragment adaptability	An adaptability aspect, which stresses on the proportionality of number of process fragments and number of workflow engine machines.
RPC	Remote Procedure Call (RPC) is an inter-process communication that allows a computer program to cause a subroutine or procedure to execute in another address space (commonly on another computer on a shared network) without the programmer explicitly coding the details for this remote interaction.
SMTP	Simple Mail Transfer Protocol (SMTP) is an Internet standard for e-mail transmission across Internet Protocol networks.
SOA Stack	The Service Oriented Architecture (SOA) contains several layers such as messaging, coordination, orchestration, choreography etc. that are called SOA stack.
SOAP	Simple Object Access Protocol is a messaging standard in

Phrase	Description
	the Service Oriented Architecture.
UDDI	Universal Description and Discovery Interface is a repository through which registering, advertising, and discovering of web services are possible.
UML	Universal Modeling Language is a standard and graphical language to model software systems.
URI	Uniform Resource Identifier (URI) is a string of characters used to identify a name or a resource on the Internet. Such identification enables interaction with representations of the resource over a network using specific protocols.
URL	Uniform Resource Locator (URL) is a Uniform Resource Identifier (URI) that specifies where a known resource is available and the mechanism for retrieving it. The World Wide Web is a good example of URL addressing.
WADE	Workflow and Agent Development Environment (WADE) is a software platform based on JADE that provides support for the execution of tasks defined according to the workflow metaphor.
WFMC	Workflow Management Coalition (WfMC) is a global organization of adopters, developers, consultants, analysts, as well as university and research groups engaged in workflow and business process management.
Workflow	A set of activities related based on a workflow metaphor. In this thesis a workflow is referred as a runtime instance of a business process specified by BPEL.
Workflow fragment	The runtime version of each process fragment.
WSDL	Web Service Definition Language.
WSFL	The Web Services Flow Language is an XML language for the description of Web Services compositions as part of a business process definition.
XLANG	XLANG is an XML-based extension of Web Services Description Language (WSDL). It provides both the model of an orchestration of services as well as collaboration contracts between orchestrations.
XML	Extensible Markup Language (XML) is a set of rules for

Phrase	Description
	encoding documents in machine-readable form.
XPath	XPath is a language for addressing parts of an XML document, designed to be used by both XSLT and XPointer.
XQuery	A Query language that uses the structure of XML intelligently and is able to express queries across all these kinds of data stored or viewed as XML.

CHAPTER 1

INTRODUCTION

1.1 Motivation and Background

The applications of business processes in industries such as chain management, online retail, and healthcare are discussed in (Li et al., 2010). They consist of complex interactions among a large set of geographically distributed services deployed and maintained by various organizations. In addition, an electronic manufacturer is also reported by (Li et al., 2010) that employs business processes to conduct its operations including component stocking, manufacturing, warehouse administration, order management, and sales forecasting. There exist geographically distributed parties such as a number of suppliers, several organizational departments, a dozen of sales centers, and many retailers. It implicitly mentions that any process in this organization is likely to receive a large number of requests from different parties. Such number of concurrent requests is a natural fit to this thesis; particularly, for department-level processes because inter-department processes are mostly considered in the SOA choreography layer.

In addition, new software paradigms introduced in the cloud computing (Dimitris, 2011; Syed A. Ahson & llyas, 2011) such as Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS) are supposed to receive a huge number of requests. Particularly, in the case of orchestration engine as a service, a huge number of workflow instances might be deployed and requested from various clients all around the world. Handling such number of requests results in an ever-changing runtime environment and demands for different resources. The

resources such as number of cloud machines and/or available bandwidth of communication media are usually subject to change.

On the other hand, the idea of the open-world paradigm (Baresi, Nitto et al., 2006) mentions that software executes in an ever-changing environment; therefore, static design time metrics will not be applicable at runtime. In addition, authors in (Dustdar & Papazoglou, 2008; Papazoglou et al., 2007, 2008) bring the same idea to the service oriented architecture. The idea defines the adaptability as an ability of system components to avoid approaching and violating system thresholds. While the runtime environment changes continuously, it is the software that must be self-adapted to keep the whole system in a safe side. In the open-world paradigm, evolving the software at runtime is necessary due to the fact that most of the time changes in runtime environment are not predictable at design time. Software, therefore, needs to continuously and automatically adapt itself with runtime environment and react to the changes. Software systems will need to operate correctly despite of unexpected changes in factors such as environmental conditions, user requirements, technology, legal regulations, and market opportunities. The adaptability with runtime environment can be considered in the business process decentralization as well. This work also brings an example of applying the open-world concept to decentralize the orchestration layer of service oriented applications.

The SOA orchestration layer enacts BPEL specified business processes. Centralized orchestration engines are usually applied to execute business processes, which suffer from the lack of scalability. On the other hand, clustering of the centralized engines cannot be a final solution for the scalability issue either (Li et al., 2010). Alternatively, decentralized orchestration engines are applied to decentralize BPEL processes and encapsulate produced fragments into runtime components to improve

system non-functional quality factors. The decentralized methods can be classified into design-time and runtime categories. A number of decentralization methods are considered in the design-time category while a few of them are classified in the runtime category.

The runtime based decentralization can be based on either workflow conditions or environment circumstances. An instance of decentralization based on workflow conditions is presented by (Tan & Fan, 2007) in which a process is step-by-step decentralized at runtime. In some situations a workflow can be decentralized while in some situations it is not the case. This kind of fragmentation based on dynamic criteria results in creating different versions of a workflow at runtime. This work does not reveal any architecture to support dynamic decentralization of workflows either. On the other hand, no research study has been introduced to decentralize a process based on runtime environment requirements to create runtime-adaptable fragments.

NINOS (Li et al., 2010) is classified in the design-time category of decentralization and focuses on decentralizing and executing business processes in the service oriented architecture applying a decentralized orchestration engine. It considers the adaptability of decentralization with runtime environment; although, static fragmentation is used in this engine. The mentioned research, decentralizes business processes in an activity level, which is analogous to decomposing a business process into its building activities. Each activity is considered as a fragment and encapsulated in an agent whose role is running this kind of activity and the agent is eventually deployed into runtime environment. Simultaneously, another research (Muthusamy et al., 2009) focuses on runtime reconfiguration of agents to keep the system status consistent with Service Level Agreement (SLA) constraints. The adaptability is

realized by reconfiguration (locating/relocating) of agents to suitable places, where they are able to satisfy the SLA requirements.

This thesis is also based on the NINOS (Li et al., 2010) view regarding business process decentralization. However, there are several shortcomings in the NINOS system that come about due to lack of adaptability of created fragments with runtime circumstances:

- The first aspect is encapsulating inappropriate process activities in fragments. If some activities are closely interrelated; therefore, they must be encapsulated in one fragment to omit intra-fragment communication costs among the activities.
- The second aspect is the lack of proportionality between number of machines and number of fragments. In the finest activity-level fragmentation applied in the NINOS, number of created fragments in case of a workflow engine with one machine is equal to when several machines are allocated for the engine. A suitable solution should be able to produce only one fragment in the case of one machine or proper number of fragments in the case of more machines. It increases consumption and degradation of system resources. Memory consumption due to number of the fragments and bandwidth wastage because of number of messages exchanged among the fragments are instances of wasting system resources. It negatively results in increasing response-time and decreasing throughput of workflows.
- The third aspect is ignoring communication media conditions in producing fragments. In busy network media, producing a number of fragments results in a large amount of exchanging messages. Indeed, decentralization process must provide less number of fragments (coarser granular fragments) to refrain from

approaching media thresholds. Fragments with softer granularity can be produced to use the maximum scalability when more bandwidth is available.

1.2 Problem Statement

In order to classify the problems of previous research attempts in business process decentralization, three adaptability aspects in decentralization are introduced in this thesis. The first aspect is the *frequent-path* adaptability, which is equal to detecting closely-interrelated activities and encapsulating them in the same fragment to omit the communication cost among the activities. The *proportional-fragment* adaptability is another aspect, which is analogous to the proportionality of produced fragments with number of machines to extenuate internal communication among fragments on the same machine. The last one is the *available-bandwidth* adaptability, which means process fragmentation based on the bandwidth fluctuations of a communication medium.

An ever-changing runtime environment along with the mentioned adaptability aspects may result in producing variety versions of a business process at runtime. Considering such dynamicity in workflow execution raises problems, which can be stated as the following research problems:

- There is no framework to support architectures to execute dynamically created fragments, decentralization methods in order to create adapted fragments, and a feedback loop to consider runtime circumstances in decentralization.
- Current decentralization methods do not consider the frequent-path and proportional-fragment adaptability aspects in creating fragments, which negatively affects response-time and throughput of decentralized workflows.

- There is no method to map runtime circumstances to a suitable decentralization method in order to satisfy the available-bandwidth adaptability.

1.3 Hypotheses

The stated research problems in this thesis are hypothesized as follows:

- Hypothesis-1: an adaptable and decentralized workflow execution framework supports architectures to execute dynamic created fragments, decentralization methods, and a feedback loop to consider runtime circumstances in business process decentralization.
- Hypothesis-2: applying the frequent-path and proportional-fragment adaptability aspects improve response-time, throughput, and bandwidth-usage of decentralized business processes.
- Hypothesis-3: applying the available-bandwidth adaptability aspect balances scalability and adaptability of a decentralized workflow engine and improves bandwidth-usage.

1.4 Research Objectives

The focus of this study is on developing a decentralized orchestration engine, which is adaptable with ever-changing runtime requirements. In other words, this research aims at the following research objectives:

- To propose a framework including architectures to support execution of dynamically created fragments, decentralization methods to create adapted fragments, and a feedback loop from runtime environment to consider runtime circumstances in business process decentralization.

- To improve response-time and throughput of decentralized business processes applying the frequent-path and proportional-fragment adaptability aspects.
- To improve bandwidth-usage of decentralized business processes applying the available-bandwidth adaptability.

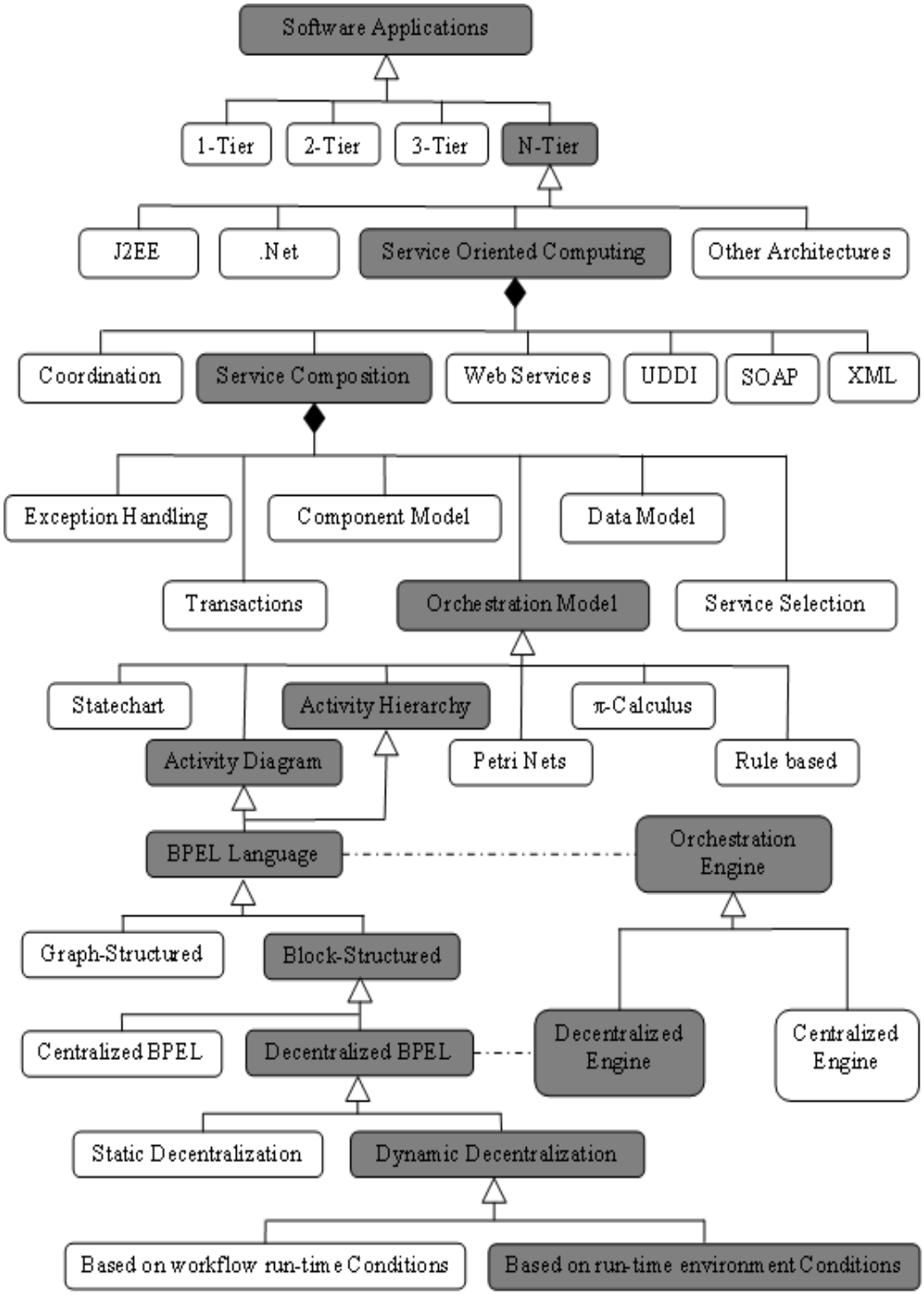


Figure 1.1: Scope of Research

1.5 Research Scope and Constraints

Software applications are categorized to 1-tier (Centralized), 2-tier (Client/Server), 3-tier (User Interface Logic/Business logic/Data Logic) and n-tier from a distribution perspective. In this thesis, the web service computing as an instance of n-tier applications would be focused. The shaded area of Figure 1.1 shows the position of this thesis among variety concepts introduced by the service oriented computing.

Business Process Execution Language (BPEL) is the most recent language in describing orchestration of web services and is originated from activity chart and activity hierarchy models. The BPEL language can be implemented as graph-structured, which is symbolized by notions of graph or can be implemented as block-structured, which is symbolized by notions of block. The block-structured and graph-structured BPEL are analogous in modeling of service composition (Garcia-Banuelos, 2008; Kopp et al., 2008; Mendling et al., 2008); however, there are advantages and disadvantages for each one. The block-structured BPEL is more common compared to the graph-structured one so that several pioneer companies have implemented only this model (Active-Endpoints, 2011). The focus of this thesis is also on the block-structured BPEL; although, several discussions in this thesis may cover the graph-structured processes.

Basically, BPEL specified business processes are usually enacted by centralized orchestration engines, which suffer from the lack of scalability. There are several attempts to decentralize BPEL processes to overcome the scalability issue. Business processes are decentralized applying static or dynamic criteria. Dynamic process decentralization can be achieved based on workflow circumstances or runtime environment conditions. Here, the focus would be on dynamic criteria, which gather

feedback data from runtime environment and make decision on how to decentralize a business process based on the feedback data.

Indeed, two methods are introduced to decentralize workflows namely HPD and HIPD. The former considers the hierarchy of activities while the latter considers not only the hierarchy of activities but also applies a mining approach using the log data of previously executed workflows. Both methods are also used by a fuzzy decentralization decision making unit in order to determine a suitable decentralization method based on the current number of machines and available bandwidth of a network. Both HPD and HIPD decentralization methods fully decentralize a business process in the last level of its process tree, which is called Fully Process Decentralization (FPD). Only (Li et al., 2010) considers the FPD method for decentralization and along with (Muthusamy et al., 2009), they consider runtime adaptability by reconfiguration of agents based on feedbacks from runtime environment. They apply a multi-agent system to execute process fragments. Indeed, the whole experiments and comparisons in this thesis consider the Centralized and FPD methods.

The presented decentralization methods are evaluated using three basic BPEL processes each of which includes only one structured activity. Activities such as While, Flow and Switch (If) are considered due to their common usage and other structured BPEL activities are ignored in that they are simple and straightforward enough to be overlooked.

There are also some more constraints, which are worth mentioning here. Several aspects of workflow management systems are excluded from this research such as governance of fragments, inter-organizational workflows and security, managing

runtime state of workflow fragments, runtime workflow reconfiguration and long running workflows, transaction, exception handling and sharing workflow interior/exterior variables. These are normally either implemented by a distributed middleware (Atluri et al., 2007; Cai et al., 1996; Li et al., 2010; Tan & Fan, 2007) or may demand for more attention from dynamic process decentralization view in future work.

1.6 Research Contributions

The main contribution of this research is to support the implementation of an adaptable decentralized workflow engine, which is able to adapt workflow decentralization with runtime environment circumstances and provide more suitable workflow fragments instead of static design-time fragmentation. The prominent contributions can be mentioned as follows:

i. An Adaptable and Decentralized Workflow Execution Framework (ADWEF) is introduced, which proposes an abstraction of runtime adaptable decentralization in the SOA orchestration layer.
ii. Two types of architectures namely Type-1 and Type-2 are presented for the ADWEF, which support the execution of dynamic process decentralization.
iii. Three aspects of decentralization adaptability are introduced namely frequent-path, proportional-fragment, and available-bandwidth adaptability.
iv. Two decentralization methods are presented called HPD and HIPD, which are capable of providing various fragments. The latter considers frequent-path adaptability and both of them satisfy both frequent-path and proportional-fragment adaptability aspects.

v. A Fuzzy Decentralization Decision Making unit (FDDM) is presented based on the fuzzy logic to choose a suitable method of decentralization that satisfies the three adaptability aspects frequent-path, proportional-fragment and available-bandwidth.

vi. An algorithm is introduced for wiring of dynamic fragments.

The findings of this thesis by considering the mentioned research hypotheses are classified as follows:

i. The Hypothesis-1 was substantiated by two ADWEF architectures Type-1 and Type-2, the decentralization methods HPD and HIPD, and a feedback loop from runtime environment. The architectures were proved during performing the experiments in that they could execute dynamically created fragments. Both decentralization methods could provide customized fragments which are applied to satisfy the three adaptability aspects. The feedback loop from runtime environment was established by (1) a data repository, for realizing the frequent-path adaptability; (2) number of workflow engine machines, for realizing the proportional-fragment adaptability; and (3) available bandwidth, for realizing the available-bandwidth adaptability.

ii. The frequent-path adaptability was substantiated by Hypothesis-2 in that the encapsulation of closely-interrelated activities in one fragment resulted in great improvements in response-time, throughput, and bandwidth-usage. Moreover, the fragment proportionality proved Hypothsis-2 in that when number of fragments was proportional to number of workflow engine machines, response-time and throughput were improved.

iii. Hypothesis-3 was substantiated by the available-bandwidth adaptability so that scalability and adaptability were balanced and bandwidth-usage was improved.

1.7 Organization of Thesis

The remaining parts of this thesis are organized according to the following chapters:

Chapter 2 provides background information about web service concept as well as service oriented architectures and clarifies orchestrate layer of the service oriented stack. The common aspects of business process languages are studied as well.

Chapter 3 focuses on the literature of business process decentralization from different aspects. In addition, it categorizes the presented methods and determines the position of this thesis among many. In addition, it determines how this thesis is related to the open-world concept. Furthermore, the concepts of workflow mining, block/graph-oriented business processes and the fuzzy logic are discusses as well.

Chapter 4 presents the methodology of this research and introduces benchmark business processes, design of experiments and evaluation metrics for two decentralization methods and a fuzzy decentralization decision making unit.

Chapter 5 presents two architectures to execute different decentralized versions of a business process. In addition to the architectures, two decentralization methods for business process decentralization are presented as well. Finally, a fuzzy unit is presented to show how the process decentralization can be adapted with workflow frequent paths, number of machines and the available bandwidth of a network.

Chapter 6 draws conclusions and outlines prospective activities in the sequel of this thesis based on the experiences achieved throughout this research.

1.8 Summary

This chapter introduced the mainstream of this thesis. Motivation and background information of this research attempt were presented. Later, statements of research problems and hypotheses were articulated. Following that, research objectives, scopes, contributions and thesis organization were detailed.

CHAPTER 2

BACKGROUND

2.1 Introduction

In this chapter several aspects of service orientation are discussed. First the concept of web service and its architecture are introduced. Most importantly, the orchestration of web services is elaborated by paying more attention to BPEL as a language for web service composition. Furthermore, the content of this chapter is based on the books (Alonso et al., 2004; Singh & Huhns, 2005); however, each section is elaborated by referring to more related works.

2.2 Web Service Definitions

The term web service is used in different meaning nowadays and several definitions are presented to clarify it. UDDI consortium (UDDI-Consortium, 2001) characterizes web services more precisely as “self-contained, modular business applications that have open, Internet-oriented, standard-based interfaces”. Alternatively, World Wide Web consortium and specifically the group involved in the web service activity says “a software application identified by a URI, whose interfaces and bindings are capable of being defined, described, and discovered as XML artifacts. A Web service supports direct interactions with other software agents using XML-based message exchange via Internet-based protocols” (W3C, 2011a). Webopedia defines web services as “a standardized way of integrating Web-based applications using XML, SOAP, WSDL, and UDDI open standards over the Internet protocol backbone. XML is used to tag the data, SOAP is used to transfer the data, WSDL is used for describing the services available, and UDDI is used for listening what services are available” (Jupitermedia-Corporation, 2011).

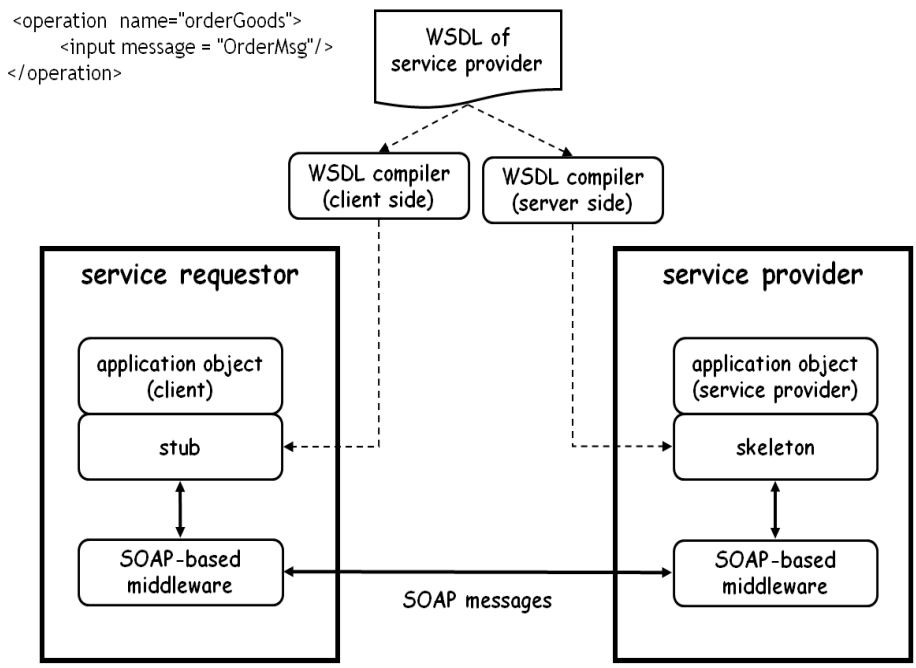


Figure 2.1: WSDL Compilation to Stubs and Skeletons (Alonso et al., 2004)

2.3 Web Services Architectures

There are two salient aspects in web services architectures. The first aspect is the way of exposing internal operations so as they can be accessible through the web. Hence, a mechanism is required to receive the web service invocations from the web and send them to software applications, which are interacting through an internal middleware. The second view regarding web service architecture is a middleware infrastructure to integrate various web services, which can be referred to as an external middleware.

The core function of a web service infrastructure must cover some minimal requirements according to Figure 2.1 . First, XML style has been accepted as a unique format to specify data, interfaces, communications, etc. Second, SOAP message format is used by a RPC-based communication between clients and servers. However, SOAP is not supported in most cases and other exchange message protocols such as HTTP, SMTP, etc. are used to convey SOAP messages. Third,

WSDL has been taken into account to describe web services functionality. This interface identifies methods implemented by a web service, input and output parameters. Client is able to call server methods as a local procedure call. Client request and server reply are converted to SOAP messages. The web service infrastructure uses WSDL to make client and server proxies (stub and skeleton).

According to Figure 2.1, the next step after establishing communication among services is to apply the most basic middleware to register the services for being able to be looked up. Indeed, service requesters will be able to look for their favorite services through a standard registry. This registry in web service context is called Universal Description, Discovery, and Integration (UDDI), which acts as a name server.

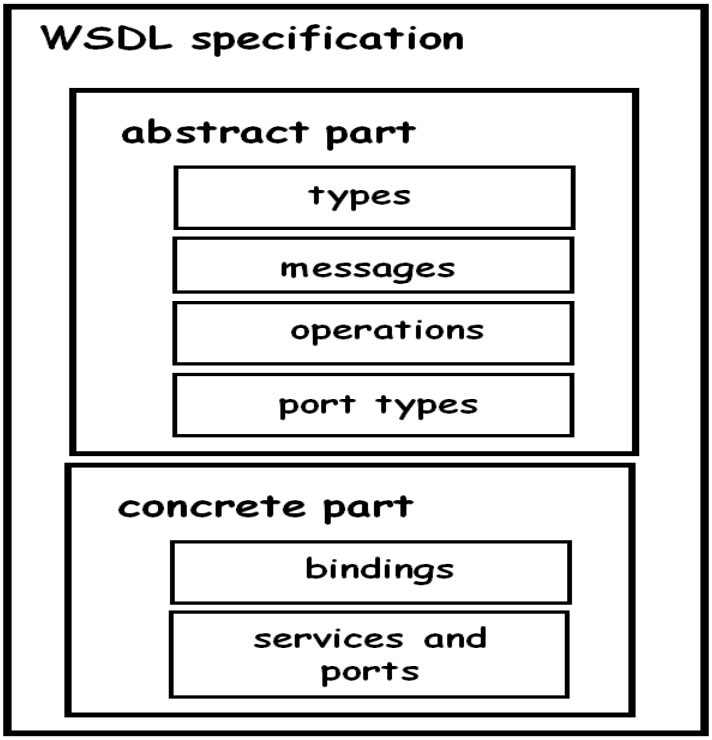


Figure 2.2: WSDL Service Specification (Alonso et al., 2004)

Web services are specified by an XML-styled document called WSDL (W3C, 2011b). Based on Figure 2.2, a WSDL document is divided into the abstract and the

concrete parts. The abstract part includes *port type*, which is analogous to conventional IDL concept in traditional distributed system middleware in that it separates specification from implementation. The port type identifies a set of operations and each operation is defined by exchangeable messages. Any communication with a web service is handled by messages, which include data required by web services. WSDL supports a type system to define data so that it is understandable by both parts of communication and thanks to the built-in basic data types and ability of XML schema in allowing user-defined data types. Based on the XML schema, data structures required in exchanged messages can be defined. Each message is divided into *parts* and each part has its own name and type according to the XML schema. For instance, in order to make a message to invoke a web service method including an integer and float, two parts must be included each of which is described by the respective XML schema.

Then it is time to define operations according to four operation types including *one-way, notification, request-response, and solicit-response*. One-way and notification types are similar in that they use just one message; however, in the one-way type a client invokes a service by sending a message and in the notification a service sends a message. The request-response and solicit-response are also the same as they exchange two messages; however, in request-response client starts sending message and solicit-response service initiate sending message. In the former, two methods are involved for synchronous interactions while in the latter two methods are engaged for asynchronous communication. The final step is the grouping of operations into port types and even extending one port type from another port type is possible. This grouping allows binding of each group of operations individually.

The concrete part of WSDL includes *Interface Binding, ports, and services*. The binding is responsible for defining message encoding and protocol bindings of all operations and messages. Determining the type of operation, which is *RPC-style* or *Document-style,* can be done in this part. In addition, it is the right place of identifying communication and transport protocol of web service messages. For instance, SOAP as communication and HTTP or SMTP as transport protocols. The next section is Ports, which is also known as *Endpoints* and allocates the Interface Binding a URI network address. This address determines where the service implementation is located. Services are grouping of ports that implicitly mean there might be several implementation of this service in different locations or even by logically grouping of related ports they are likely to be at the same address; and there might be different bindings for the same port type, which is analogous to accessing a service using several communication and transport protocols.

2.4 Service Composition, Choreography and Orchestration

Composition of web services tackles the implementation of web services whose business logic involves the invocation of other web services operations. Usually a *composite web service* is a web service whose functionality is combining the functionality provided by other web services. *Service composition* is the process of composing web services through a separate web service.

Web service composition had previously been implemented by clients in traditional system; however, it is currently analogous to what happens in workflow systems where the business logic of an application is implemented by composing other applications. The only difference is that service composition and applications are emerging as web services in service oriented computing. Figure 2.3 shows a sample

of supplier activity diagram, which orchestrated customer, warehouse and supplier local service.

Web service composition can be studied from different aspects such as Component model, Orchestration model, Data and data access model, Service selection model, Transactions and Exception handling.

Component model determines which components make a composition. The most general form of composition can be made based on a very simple assumption such as using RPC-like communication and using XML as a facility to manage data and communication. In addition, a more specific composition model can be based on a number of standards such as using HTTP/SOAP protocols for communication, WSDL standard, which determines the interface of web services, etc.

Orchestration model of composition is an abstract layer, which is usually described by descriptive languages. These languages determine in which order and under which circumstances a web service must be invoked. Even exceptional cases might be managed in a very controlled manner. Several orchestration models have been drawn in (Aiello et al., 2002; Cass et al., 2000; IBM, 2011a, 2011b; Sheng et al., 2002), which are mostly based on Activity diagram, State chart, Petri-nets, π-calculus, Activity hierarchies, and Rule-based orchestration.

Activity diagrams have been formed based on the flowchart concept and are able to determine the A-to-Z of web service composition using flowchart-like notations. The similarity of human being thinking to the way activity diagrams model the world is a success factor of activity diagrams. Activity diagrams are used to model the business logic among several web services according to the sample shown in Figure 2.3.

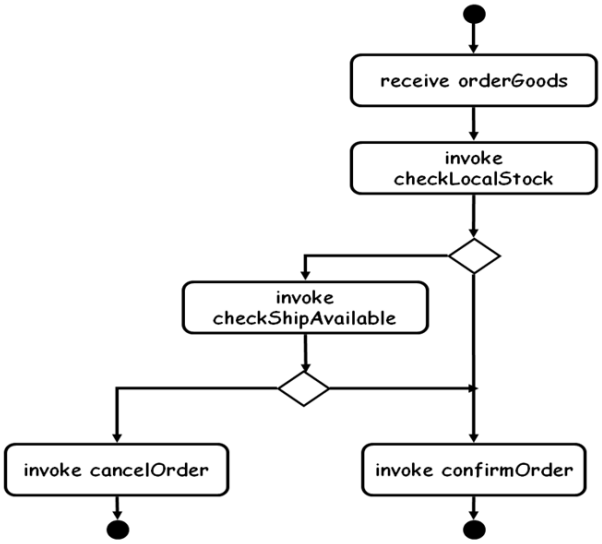


Figure 2.3: Supplier Process Specified by Activity Diagram (Alonso et al., 2004)

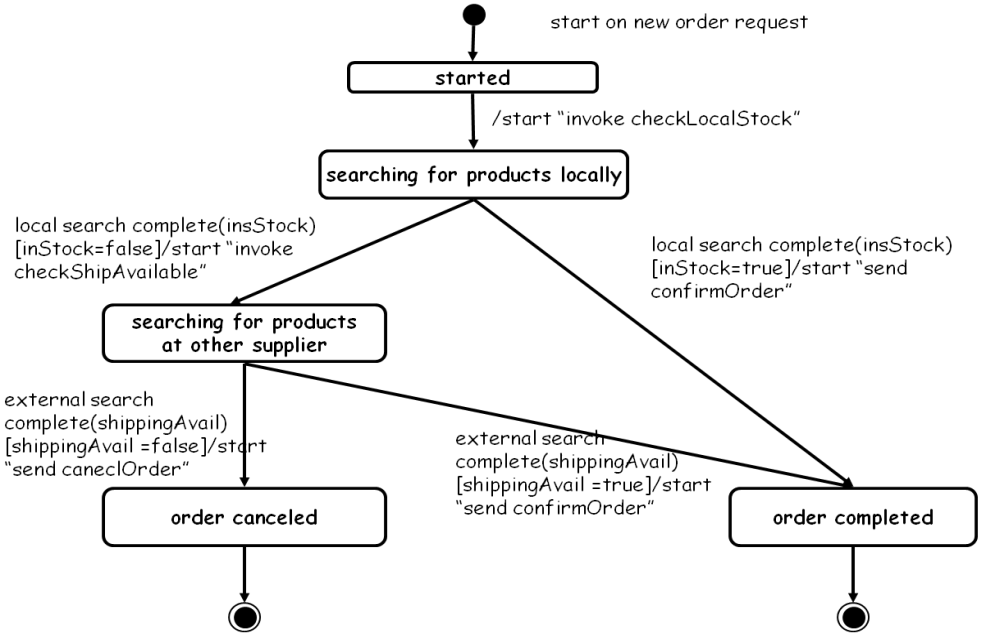


Figure 2.4: Supplier Process Specified by Statechart (Alonso et al., 2004)

Statecharts are based on state machines and specifies how activities can be executed based on entering, existing and exiting a specific state. The relation of different states can be determined applying several notations such as transitions, conditions and events. By giving meaningful names to each state, a bulk of useful information about

system is obtained at a glance. A sample state chart diagram is shown in Figure 2.4, which models the supplier process.

Petri-nets is a combination of activity and state chart diagrams, which are fully supported by many analytical tools to refrain from deadlocks and many other erroneous situations. This is a very well-known workflow description language and has been fully studied from different aspects.

Π-calculus applies algebraic notations to define features of a workflow and has inspired some modern composition languages such as XLANG (Microsoft, 2001) and BPEL. This language also focuses on the formalism of workflows such as what Petri-nets does. This language offers sequential, parallel and conditional constructs to composition of activities.

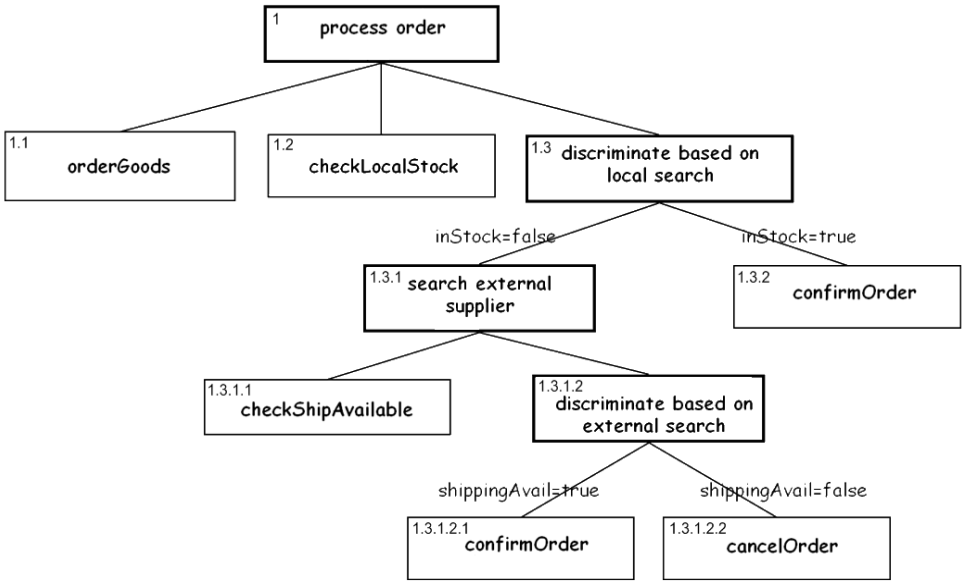


Figure 2.5: Activity Hierarchy Specification of Supplier Process (Alonso et al., 2004)

Activity Hierarchies approach has a top-down view to activity composition so that leaf nodes are real activities while middle nodes contain their subsidiaries and are actually control constructs, which force several constraints on the leaf nodes

according to the supplier sample in Figure 2.5. A notable point regarding this kind of activity composition is that companies mostly prefer it compared to the prior models. Several advantages can be counted on this modeling. First, abstraction of orchestration level is possible through this model so that more abstract layers are on the top and more real layers are at the bottom; second, this model is intrinsically modularized; and third, this model is conveniently mapped to usual programming languages. A language based on this approach is Little-JIL (Cass et al., 2000).

Rule-based orchestration specifies the orchestration of services by a set of rules. Especially, event based systems are mostly matched with rule-based systems. Once an event happens an action is performed to handle the situation. Hence, a system can be modeled by a set of <event-action> pairs. Another paradigm also recommended for rule-based orchestration called event-condition-action (ECA), which includes conditions in handling rules by evaluating a Boolean predicate when an event occurs. Accordingly, rule-based composition engines are based on reaction to events such as receiving messages from clients and services, evaluating the whole conditions and considering actions such as sending messages to other parties. All the rules can be triggered when a service is invoked by others. It makes rule-based orchestration less structured compared to the activity-hierarchy based orchestration. Also, considering an order number among services is not convenient applying rules. In addition, a large number of rules make the system behavior difficult to figure out and sensitive to errors. Triggering rules based on conditions makes a system not predictable and hard to manage; although, it makes the system more compatible with systems made based on exception handling logic.

Data and Data Transfer Model of service composition is analogous to common programming languages data access and manipulation. Data types, data definition

and passing data among activities as well as web services are of salient importance in web service composition.

Generally, data from service composition point of view can be classified into *application-specific* as well as *control-flow* data. The former is the data, which is sent or received as messages among services and the latter is the data used to control the composition of services through control structures such as While, Switch, Flow, etc. Application data is mostly richer than control data and includes structured data such as Tree, Array, Record, Class, etc. and in the web service context they are mainly mapped to XML structures. The *control-flow* data are mostly basic structures, which can also be mapped to XML structures in advanced service composition cases.

There are two prominent methods in order to access *application-specific* data. One approach considers data as a black box, where accessing to it can be done by pointers from activity to activity. All data formats such as documents and images can be given a URL to refer them. Indeed, the composition logic does not intervene in exchanging complex data among services; however, in some cases the composition logic may need to access real data not just its reference. Another approach is definition of data as part of composition logic, which is the same as what happened to application *control-flow* data.

In addition to the data access, the definition of data types for variables and application specific data is of high importance. In the context of web services, XML has found a great position so that it can model all types of data (Hewlett-Packard Company, 2011). Furthermore, XML is a pretty well-formed structure and can be standardized conveniently and sounds well for *application–specific* data, particularly.

However, XML demands a lot of parsing and data conversions and also conveys a large amount of data due to text definition of data.

The transfer model of data is passing data from one web service operation to the next and can essentially be done through *blackboard* and *explicit data flow* approaches. The blackboard approach is inspired by common programming language way of managing data. A collection of data is considered for each scope of a program called blackboard, where is a right place for activities to take/put their input/output data. A mapping schema maps the in/out messages data to the blackboard data. Hence, when a web service method is invoked, a message is made by taking required variables from the blackboard and is then mapped to message. Correspondingly, when a message is received as output of a web service method, data in the massage is mapped to the variables in the blackboard. In addition, accessing to the variables can be managed by the blackboard and authors in (Hewlett-Packard Company, 2011; IBM, 2011a) use the same approach as well. Dataflow approach is based on explicit making of data flow between activities. It can be imagined as wiring of activities so that a designer may wire input data of an activity to output data of another activity and (Bausch et al., 2003; IBM, 2011b) are sound examples of using dataflow approach in the web service context.

The blackboard, compared to the dataflow approach, is more suitable for programmers due to its similarity with common programming logic. In addition, accessing to the shared data in the blackboard must also be controlled. However, dataflow approach is more flexible and naturally more complex. As an example of complexity is that the completion of a source activity must be controlled because it must provide data required by a destination activity.

Service Selection is the policy of determining a destination service, which must be called. The information about target services is typically hardcoded to a composition schema. It is worth mentioning that according to (Singh & Huhns, 2005), *Uniform Resource Identifier* (*URI*) is presented to unique addressing of resources in the web scale. The URI is based on several schemes such as *http*, *ftp*, etc. *Uniform Resource Locator* (*URL*) is a kind of URI, which gives address/location of each resource. Port type is a concept mostly used in composition languages to determine the target service and is resolved to real port addresses at runtime. There are at least four methods for resolving a target service including *static binding*, *dynamic binding by reference*, *dynamic binding by lookup*, and *dynamic operation selection*. In *static binding*, which is the simplest method, a URL is hardcoded in a composition schema and is typically determined by composition designers and there is actually no service selection at runtime. An immediate addressing of the mentioned issue is determining the service URI at runtime through which URI is specified by composition logic not by process designer and this method is called *dynamic binding by reference*. Usually URI is searched by an extra activity from UDDI repository beforehand. This service selection method is recommended by majority of composition languages as well. *Dynamic binding by lookup* is usually supported by composition middleware and is realized by adding query facilities through which the composition logic is able to make query to find a suitable service registered by the UDDI. Alternatively, CORBA (OMG, 2011b) offers an approach through which not only service selection can be dynamic but also selecting service operation is also possible. This method provides highly flexible service composition particularly when methods provided by various service providers for the same purpose are different; although, implementing robust applications based on it is unpredictably erroneous.

Transactions are essential in some cases and are supposed to be fully implemented by middleware without involving composition designers. In the web service context, WS-Transaction specifies the transaction features of web services and service composition. On the other hand, wide usage of transactions is continuously blamed due to long-term locking of system resources and ACID features of transactions are not applicable in general. In order to support transactions in services and service composition schema, *atomic region* and *compensation* are introduced. Atomic region can be implemented by the 2PC protocol with high complexity. *Compensation* is a mechanism to undo committed services by executing other operations. In service composition, compensation concept is also implemented by initiating protocols to compensate the executed activities to rollback some parts of atomic region partially.

Exception Handling means managing any exception in behavior of composed services. It usually happens by any failure in servers, services, communication, etc. Sometimes exceptions are handled by transactions and it makes exception handling rather complex due to firing transactions. There are several alternative methods and the most prominent ones are flow-based, try-catch-throw, and rule-based approaches.

Flow-based approaches can be considered as a substitute for other methods of exception handling. In this approach, at the end of each operation the results are tested for errors, and based on the error type a proper action is taken. This idea can be considered in service composition by testing the result of each activity. It is worth noting that exceptions can be thrown explicitly to signal an erroneous situation by activities or web services. On the other hand, designers are able to define exception-handling logic to handle specific situations with more flexibility.

Try-catch-throw approach is very similar to the Java way of managing exceptions and is adapted to service composition as well. When an exception occurs it will be handled by one or more activities. The exception handling mechanism listens to some erroneous situations and then by receiving an erroneous signal the exception handling part of the code is executed. It is the exception mechanism that transfers the control to process execution logic where the next activity is located, or terminates the business process. This mechanism is also well suited to composition logic, which is based on the hierarchy of activities. The *try-catch-throw* mechanism separates the normal and exception logic clearly. Usually the normal behavior of composition is designed and then exceptional cases are added by designers.

Rule-based approaches are based on the means of the previously mentioned ECA rules (Event-Condition-Action), where an event defines an exceptional case and can be captured in the form of message. Each event is handled by a Boolean expression and for each event, which satisfies a Boolean expression there is an action that reacts. Rules-based approaches also separate normal and exception behavior of composition logic. However, rule definition languages are different from composition schema definition languages. The complexity of managing a large number of rules is a real cumbersome for developers as well.

2.5 Business Process Execution Language (BPEL)

BPEL (Active-Endpoints, 2011; Vanhatalo et al., 2009) is a language, which supports the specification of composition schema as well as coordination protocols and acts as a framework for both of them. BPEL is also used to define the external behavior of a service by defining an abstract process and internal service implementation through an executable process. From now on, the term process is

used for both abstract and executable process, which shows the composition of web services. BPEL is a standard language that defines different roles in exchanging message with a process; the port types and their relations with roles and the process itself; the orchestration logic of the process; correlating the messages and the target composition instance of a business process.

Table 2.1: Summary of BPEL Activities

	Activity Name	Description
Basic	Assign	Manipulate state variables.
	Compensate	Handle a fault or exception.
	Invoke	Synchronous or asynchronous Web service call.
	Receive	Blocking wait for a message to arrive.
	Reply	Respond to a synchronous operation.
	Terminate	Terminate a process instance.
	Throw	Indicate a fault or exception.
	Wait	Delay execution for a duration or deadline.
Structured	Flow	Concurrent execution of activities.
	Pick	Conditional execution based on events.
	Sequence	Sequential execution of a set of activities.
	Switch	Conditional execution based on instance state.
	While	Looping constructs.

From the component model perspective, BPEL has fine-grained activities. Referring to Table 2.1, key BPEL basic activities, which are real components, can be classified as basic activities to perform some primitive operations and structured activities to define the control flow. In BPEL, the key basic activities are Assign, Compensate, Invoke, Receive, Reply, Terminate, Throw, and Wait, whereas the key structured activities are Flow, Pick, Sequence, Switch (If) and looping constructs such as While.

Invoke activity represents the invocation of a request/reply or a one-way operation presented by a service. *Receive* activity is to receive a message from a client while *reply* activity is to send a message in response to an invoked operation. *Assign* activity is also used to assign data to variables. *Wait* activity is used to block the execution of a process for a certain amount of time or until a specific date/time reaches. In addition, both BPEL and web services are described by WSDL abstract.

From the orchestration point of view, BPEL combines activity diagram and the activity hierarchy model such as the sample illustrated in Figure 2.6. Structured activities are used to show the hierarchy of activities and are able to group a set of basic as well as structured activities and determine the ordering of constraints among them. Structured BPEL activities are *flow*, *for-each*, *switch (if)*, *pick*, *repeat-until*, s*cope*, *sequence* and *while*, respectively. *Sequence* activity shows the order of activities to be executed sequentially. *Switch* is similar to conditional structure in common programming languages. *Pick* includes a set of events associated with an activity such as receiving a message or expiration of an alarm. When an event occurs, a specific activity associated with the event is executed and the pick is considered completed. *While* and *for-each* are loop activities, which are executed repeatedly until the exit condition comes true. *Flow* is to run different groups of

activities in parallel, where the activity diagram and activity-hierarchy of BPEL are combined together. Flow activity uses the concept of *links*, which are similar to arc in the activity diagram. A link *l* connects one and only one source activity *S* to one and only one target activity T; so that target activity T cannot start until the source activity *S* completed. Links can also be associated with transition conditions, which are evaluated when the source activity is completed. Evaluating the source link activities determines whether the target activity must be started. Most importantly, links also cannot be used to make loop and it means they can be used only for generating acyclic graphs. Loops must be created using loop structures and links can also be used in more complex structures. For instance, joining links for the cases in which an activity has more than one incoming links is also possible.

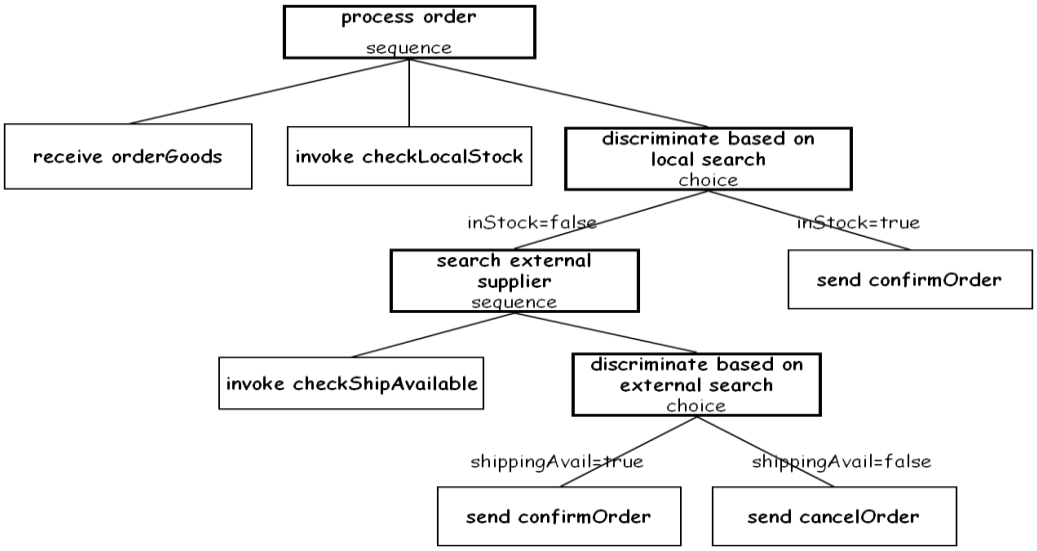


Figure 2.6: Supplier BPEL Process Specified by BPEL (Alonso et al., 2004)

Workflow languages can be classified in terms of graph or block notions. In block-structured languages, control flow is defined similar to existing programming languages by using block structures such as *If* or *While* in *C* language. This kind of modeling; however, provides clear semantics to every modeler familiar with basic computer programming languages. Furthermore, since the representation is already

in a form similar to a real program, transformation into executable code typically is easier to achieve.

Alternatively, workflow languages can be classified according to notions of graphs (Mendling et al., 2008). Process control flow in graph-structured workflow languages is defined through explicit control links between activities and is very similar to business process modeling languages such as BPMN (OMG, 2011a). Based on (Garcia-Banuelos, 2008; Kopp et al., 2008; Mendling et al., 2008), all graph-structured models can be mapped to block-structured models and vice versa. Several research studies have tried to convert the graph-structured specified business processes to their block-structured equivalents (Garcia-Banuelos, 2008; Kopp et al., 2008; Mendling et al., 2008; Vanhatalo et al., 2009). Accordingly, WS-BPEL is essentially a hybrid language derived from a block-structured ancestor XLANG (Microsoft, 2001) and a graph-structured ancestor WSFL (IBM, 2011b). It allows users to choose between both approaches and even possible to mix both concepts freely (Kopp et al., 2008). Nonetheless, due to the simplicity of block-structured BPEL and its similarities to programming languages several pioneer companies such as Oracle and Sun Microsystems (ORACLE, 2011a) do not support graph-structured BPEL (Active-Endpoints, 2011). Indeed, blocked-structured BPEL is a good starting point to adaptable and dynamic business process decentralization.

From data types and data transfer aspect of BPEL, it must be noted that an executive instance of a BPEL process maintains its state information as variable data, which are very similar to what is happened to usual programming languages. Variables in BPEL language are declared by name and types, which are specified using WSDL message types, XML schema simple types, or XML schema elements. After defining variables, they can be used as input or output parameters in operation

invocation or can be used in switch conditions or links, and loops. There are some constraints on definition of variables; for instance variables that are used as input or output parameters must be of WSDL message type. Input parameters are sent through a message to an invoked web service while output of a method invocation is sent by message from the invoked web service to the invoker. In order to access a particular value in a variable, query languages such as XPath (Kay, 2007) and XQuery (Walmsley, 2009) can be applied.

BPLE is also based on the blackboard approach and simultaneous access to variables must be controlled by a synchronization mechanism. It means concurrent access of several activities to a set of variables will have the same result such as sequential access of activities to the variables and this is ensured by BPEL.

Service Selection aspect of BPEL is realized by the concept of *partner link type*, *partner links* and *endpoint references*. Partner link types include two exchanging message roles during process execution and also WSDL port types, which must be implemented by services playing these roles. According to Figure 2.7, a partner link type orderLT shows there is an interaction between customer web service, which implements customerPT port type and supplier web service, which implements supplierPT port type. Partner link types are not created just for service composition and they are able to show relationships between any pair of web services. Roles, relationship and port types are defined by partner link types, which identify services invoked during the execution of a process. The invoked services called endpoints can be referred to statically or dynamically. Partner link definition references a partner link type, the roles of process and other party according to Figure 2.7.

In deployment time of a composite service an endpoint reference can be associated to the partner link. References can dynamically be associated to partner links at runtime based on a discovery mechanism. It means the partner link types and the roles remain the same, but the actual web service can be different from time to time. During the execution of a BPEL activity where partner links are referred to, composition/orchestration engine will be aware of the target service to which a request must be sent.

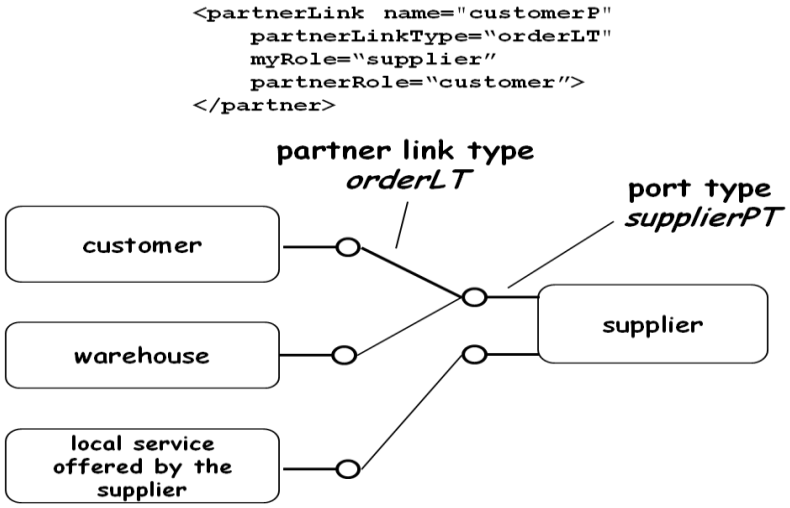


Figure 2.7: Definition of Partners in BPEL (Alonso et al., 2004)

Exception and Transactions are two important aspects of BPEL language. Exceptions are essentially handled by try-catch-throw approach inspired from the fact that BPEL has a structured orchestration model and activities are organized as nested. Everything is based on the *scope* idea so that each structured or basic activity in BPEL implicitly has its own scope. For managing groups of activities, BPEL allows explicit definition of scopes. Each scope may include *fault handlers* and they apply catch elements to determine which faults can be managed when a fault occurs. Faults may occur by orchestration engines and activities implicitly or explicitly.

When faults occur, all the activities inside the scope are suspended and control will be transferred to a specific fault handler. Another useful construct that is similar to exception handling, is *event handler*, which is always active when its container scope is running. This mechanism is able to monitor a specific event such as receiving a message, approaching a deadline, etc. and may be executed several times. Pick activity may also be used in the same way; however, it is not always active. Figure 2.8 shows a fault that occurs in the invoke activity and is propagated up until *searchExternal* activity, where respective handler resides.

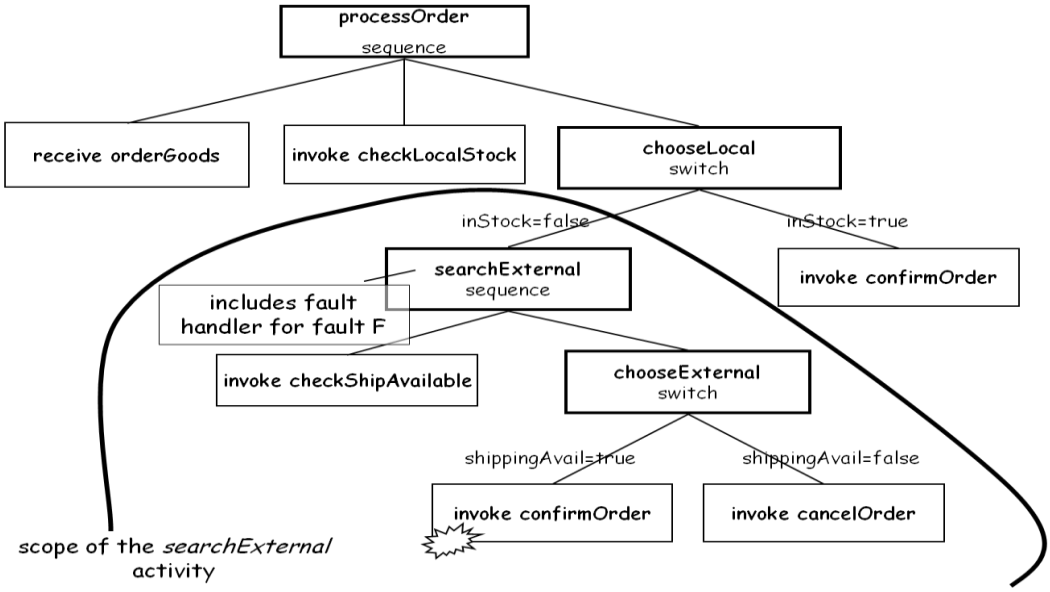


Figure 2.8: Exception Handling in BPEL (Alonso et al., 2004)

Transactions are also handled through the idea of scope. Each scope may also include compensation handlers, consisting of a single activity, which knows how to rollback the operations of this specific scope. The compensation logic of each scope can be executed once after completing the normal scope operations. It can be activated by a fault handler automatically or can be called by the scope activities explicitly. When a compensate activity is on top of a composite process, it means this process can be rolled back entirely.

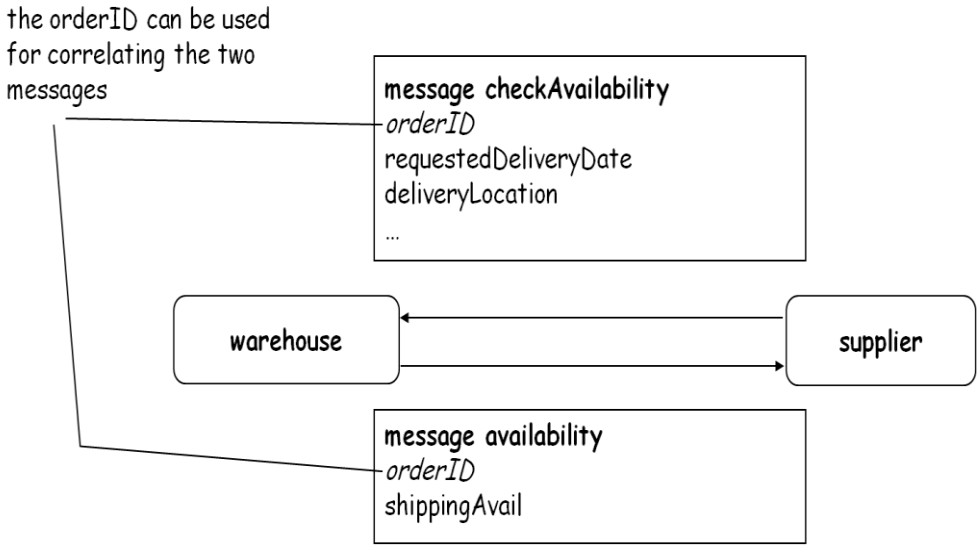


Figure 2.9: Instance Routing using Correlation (Alonso et al., 2004)

Instance Routing is mapping a message received by an orchestration engine to a right instance of a composed process. It is managed by *correlation sets* and so-called correlating messages to composition instances. Correlation sets can be associated with any incoming/outgoing messages from/to composition engine. In other words, correlation sets can be associated to invoke, reply or receive BPEL activities. Designers are also able to associate a correlation with several messages to dictate the engine that the messages belong to a specific composition instance. Correlation can be done based on designers' opinion on how to correlate messages and composition instances. Figure 2.9 shows that an orderID can be a correlation item based on which messages are sent to a specific process instance.

2.6 Summary

The main focus of this chapter was on introducing the background of service orientation. Web service standard terms were defined and web service concepts were elaborated. Later, different aspects of web services and orchestration were detailed. Finally, the BPEL language and its characteristics were encompassed.

CHAPTER 3

LITERATURE REVIEW

3.1 Introduction

The main goal of this chapter is to present relevant research works based on which the theory of our work is shaped. This study is inspired from: first, the open-world software paradigm (Baresi, Nitto et al., 2006; Dustdar & Papazoglou, 2008; Papazoglou et al., 2007, 2008), which draws the idea of dynamic and adaptive software; second, the orchestration layer of the service oriented architecture, decentralized orchestration engines; and third, dynamic business process decentralization.

This chapter is organized based on the above mentioned research activities. Accordingly, in Section 3.2 the idea of the open-world paradigm is overviewed. Section 3.3, studies the business process decentralization from three aspects including fragmentation, enactment and adaptability. Section 3.5, focuses on using a mining approach in business process decentralization. Finally, an overview of the fuzzy logic and its application in this thesis is brought at the end of the chapter.

3.2 Open-world Software Paradigm

Authors in (Baresi, Nitto et al., 2006) open a new view towards software development by introducing the idea of open-world software paradigm, which has attracted much attention nowadays. This idea is matched to the service oriented architecture by (Dustdar & Papazoglou, 2008; Nitto et al., 2008; Papazoglou et al., 2007, 2008) as well. According to this idea, in an open-world paradigm, software executes in an ever-changing environment; therefore, static design time metrics will not be responsive at runtime. Although the runtime environment changes

continuously, it is the software itself, which has to be self-adapted to keep the whole system in a safe side (Baresi, Nitto et al., 2006; Dustdar & Papazoglou, 2008; Papazoglou et al., 2007, 2008). In other words, evolving the software at runtime is necessary due to the fact that changes in runtime environment are not predictable at design time. Software thus needs to continuously and automatically adapt itself and react to the changes. Systems will need to operate correctly despite of unexpected changes in factors such as environmental conditions, user requirements, technology, legal regulations, and market opportunities. In addition, this thesis brings an example of applying open-world idea to the service oriented orchestration layer.

Authors in (Dustdar & Papazoglou, 2008; Nitto et al., 2008; Papazoglou et al., 2007, 2008) also bring the idea of open-world paradigm to service oriented applications. A chronologically overview of developing software applications is particularly introduced in (Nitto et al., 2008) that includes four phases:

- Genesis: rigid processes, static, and monolithic applications, which do not consider any change in requirements, development process and runtime environment.
- Enlightenment: agile process and dynamic system structures, which consider iterative software development process and consider stakeholders requirements;
- Industrialization: automation of software development and distributed systems, which was along with high demands for Computer Aided Software Engineering (CASE) tools, Model Driven Architecture (MDA), and increasing processing power of computers.
- Globalization: covers shared ownership and adaptive systems concepts.

Software must reach high levels of dynamism as the real world is dynamic and evolves. Simultaneously, service concept and service oriented paradigm (Erl, 2005) emerged, which composes software services while ownership of the services might belong to different parties. Indeed, the level of dynamism increases due to distribution of software among parties. In these systems, changing requirements and dynamic contexts are extremely increased and systems must be able to dynamically adapt themselves with ever-changing world. A future trend can also be software systems, which are able to predict problems and proactively solve them beforehand. Potential erroneous behaviors, deviation from standard routines, approaching software degradation situations and thresholds are some examples of problematic situations.

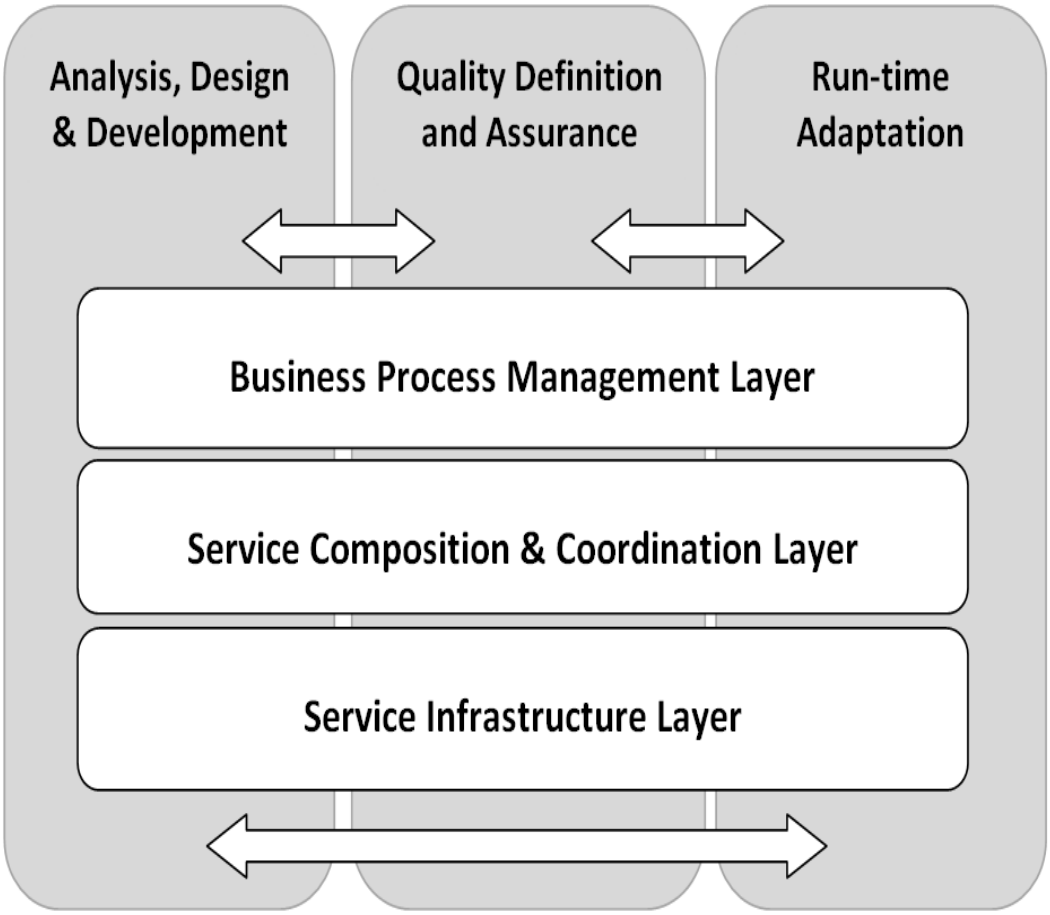


Figure 3.1: Layers and Aspects in Service-based Applications (Nitto et al., 2008)

According to Figure 3.1 from (Nitto et al., 2008), different layers of service oriented architecture can be considered with three main activities including: first, analysis, design and development; second, quality definition and assurance as well as negotiation; third, runtime adaptation. The last one can be seen from many aspects such as handling mismatches among services, repairing faults, reconfiguring the application in a way, which leads to better requirement meet. The last one is also analogous to the concept of self-adaptation, which means automatically, autonomously, and dynamically adapt to changes. This thesis also brings runtime adaptation to business process decentralization as well.

Open-world idea has been applied in several researches, which is worth mentioning to have a look at them briefly; although, none of them used it for business process decentralization. In (Perez-Palacin et al., 2010), the main focus is on performance analysis of open-world systems. This work is motivated by the fact that the performance of open-world software relies on the performance of third-party services. Indeed, performance prediction based on the current performance analysis is no longer valid. Performance must be considered not only at design time, but also at runtime. Indeed, architecture is presented to facilitate the prediction of performance. In this thesis, performance is also studied from response-time and throughput points of view based on adaptive workflow fragmentation methods.

Authors in (Baresi et al., 2010) focus on monitoring aspects of service oriented architecture when open-world requirements are considered. A general monitoring specification language is presented and methods in order to data collection, computation and analysis are presented. In addition, (Baresi & Guinea, 2010) encompasses evolving of service composition, which may come about due to the operation of third-party services or service composition itself. Predicting such

unexpected behaviors is not convenient when a service composition is designed. However, service composition must supervise itself and react to changes in order to find solutions for them. Supervising can be described in terms of monitoring and recovery. The monitoring means the system must be continuously probed while the recovery is process reaction to recover from the erroneous situation.

By and large, a few works has been performed on self-adapting of service-oriented applications, especially from service composition point of view. However, none of them considers self-adaptation of business process decentralization with runtime environment. This thesis draws the idea of open-world paradigm into business process composition layer from decentralization point of view. Consistently with changes happen in runtime environment, a decentralized workflow engine is required to be adequately flexible, dynamic and adaptive to handle the changes by providing adaptable fragments. The focus of this work is on a decentralized workflow engine, which makes adaptable fragments from a business process based on runtime environment feedbacks. The adaptability comes about in terms of first, relation among business process activities called frequent-path adaptability; second, number of dedicated machines to a workflow engine called proportional-fragment adaptability; third, choosing a suitable decentralized method which is more commensurate with available bandwidth called available-bandwidth adaptability.

3.3 Business Process Fragmentation, Enactment and Adaptability

From this study point of view and according to Figure 3.2, business process decentralization can be studied from fragmentation, enactment and adaptability. The fragmentation is actually decentralization of business processes, which results in producing a number of process fragments. Fragmentation methods are different in

that various criteria are used in fragmenting business processes. The criteria can be considered by designers at design/compile time statistically or can be determined based on runtime environments feedback data dynamically. Enactment methods address the way of realizing decentralization at runtime. Some methods use embedded communication facilities of business process to wire the fragments while several methods apply third-party platforms such as agent-based middlewares to implement fragments. In addition, adaptability of decentralization is of high importance considering the concept of open-world software paradigm. It may come about by locating/relocating of fragments to/from workflow engine machines or by producing fragments considering runtime environment circumstances, and a hybrid method can naturally be considered as well.

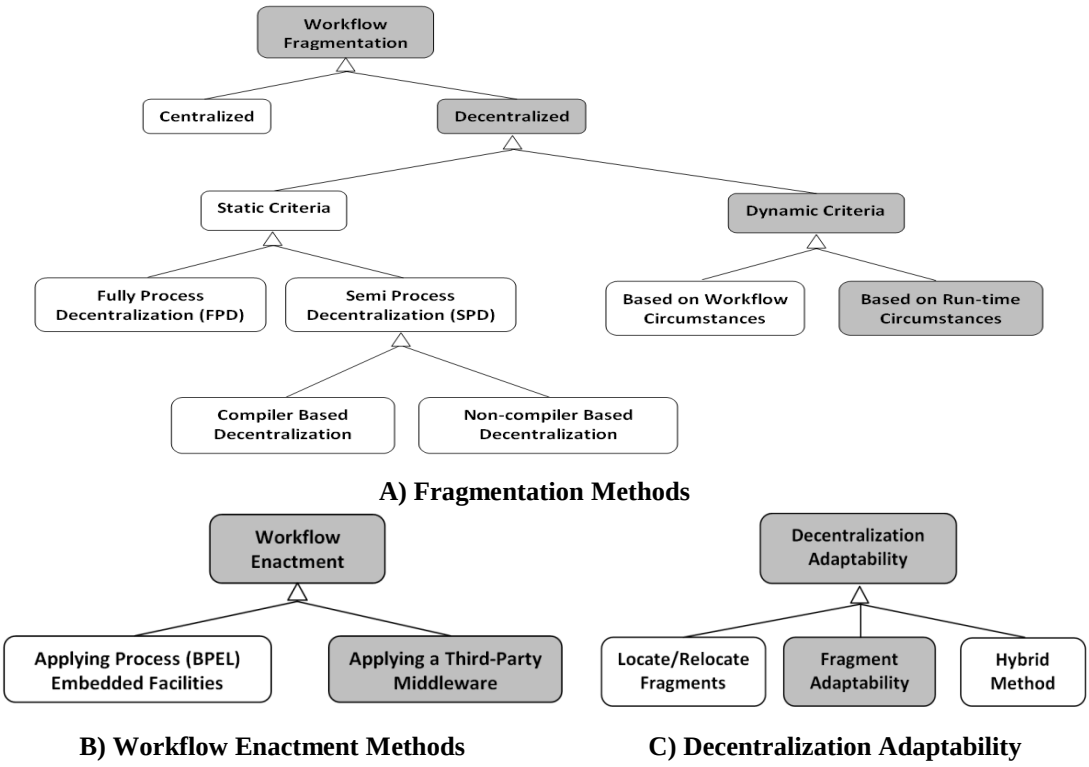


Figure 3.2: Three Aspects of Workflow Decentralization

Although all the related works can be studied from different aspects according to Figure 3.2; from now on several decentralization classes are considered and each work is classified in the best fitted category. In addition, more focus is on the related

works classified in the shaded areas and the others are referred to concisely. To be more specific, more interested areas are workflow engines, which use dynamic fragmentation, implemented by a third-party middleware and consider adaptability of workflow systems with runtime environment.

3.3.1 Centralized Process Fragmentation

Workflow Management Coalition (WMFC) (WFMC, 2011) manages workflow standards. It explains standards and methods on how to implement centralized workflow management systems. From decentralization point of view, a centralized workflow can be considered as a single fragment while parallel parts run as concurrent threads.

3.3.2 Static Fragmentation Applying Third-Party Middleware Enactment

There are several methods to enact workflows without using the business process communication facilities. These methods apply other applications to manage the fragments without embedding them in a decentralized process.

In (Mohan et al., 1995), a process is fully fragmented to the finest level of granularity, which is in activity level and the fragments are then distributed to a number of nodes. In this paper, a distributed architecture, called Exotica/FMQM, is introduced for workflow systems, which does not keep process state information in a centralized database; however, a kind of message called persistent is introduced to store the state information of business processes and it is expected to enhance the system reliability. This work can be a good model for future implementation of workflow state management in this thesis, and our current focus is on dynamic workflow fragmentation based on runtime circumstances.

METEOR project (Miller et al., 1996) proposes a distributed workflow decentralization architecture. It mostly focuses on system architecture and implementation technique based on specific communication mechanisms without paying attention to fragmentation issue. METEOR2 (Das et al., 1996; Sheth et al., 1996) also introduces a method of process scheduling among various tasks. The scheduling information is distributed among different task managers. Each task manager passes the control to its immediate neighbor once the task it controls terminates. Task manager along with task activation, task invocation, error handling and recovery components all are implemented based on CORBA components. In these approaches, a workflow is pre-partitioned in a central server, thus the partitions are made statically in the central server and distributed to each execution agent. Our work is different from this work in that a fragmentation is based on runtime feedbacks and cannot be static at all.

INCAs (Barbara et al., 1996) applies an object called INCA to workflow decentralization. An INCA contains the context of a workflow instance including execution state of the process and distribution of the object among processing stations. The INCA is then routed among the stations according to the control flow specification stored in the INCA as well as the processing station. INCA object in each station is executed as a centralized workflow and lack of scalability is a drawback of INCA while in our method there is a dynamic fragmentation and in some cases it might also be possible to create a centralized workflow based on system circumstances.

In ObjectFlow (Hsu & Kleissner, 1996), which is a workflow engine, steps are run by agents coordinated by a distributed workflow engine and the state of workflows is maintained by a centralized database. In addition, there is a flow definition model

through which workflows are specified using a graph-based specification language. This system is one of the primary workflow management systems, which considers a new generation of workflows, which are able to interact with each other. This system uses static fragmentation, which is determined at design time and workflows are realized by software agents as well. This system is different from our work in that it tries to provide a general platform for workflow execution. Runtime adaptability is neglected in this work while in our work adaptability of workflow fragments with runtime requirements is targeted.

WF-net model is extended by (Alast, 2000) to an inter-organizational paradigm based on a top-down view and a global workflow model is made up of several local ones. By transferring an inter-organizational workflow into the WF-net, the correctness issue can be solved easily. In fact, this work can be considered in the choreography level of service oriented architecture while this thesis focuses on the orchestration layer, where several machines are locally dedicated to a decentralized workflow engine.

In (Mittasch et al., 2000), a framework for business process automation is introduced, which is decentralized based on the CORBA (Common Object Request Broker Architecture) (OMG, 2011b). In this framework so-called BPA Frame, several configuration and deployment methods for workflows are implemented applying the CORBA platform and the identified approach is discussed and analyzed in terms of performance. However, this work is different from our work in that it does not have any discussion on the fragmentation models and adaptation of workflows. It does not consider any standard workflow description language such as BPEL and the CORBA framework is applied to implement workflows. In this thesis,

WADE/JADE platform is used as a multi-agent system to enact workflow fragments created based on runtime circumstances.

An integration method for the existent business processes is introduced in (Discenza., 2001) to make a larger workflow. It defines several event points in a business process where events can be received or sent. A publish/subscribe model is used to handle the interaction of business processes handled by events. This work actually introduces a button up view to decentralization where fragments are already exists and are connected to each other through a messaging mechanism. This work gives good comments on connecting workflow fragments through a messaging middleware while in this thesis, fragments do not exist previously and are produced dynamically.

SELF-SERV (Benatallah et al., 2003; Sheng et al., 2002) presents a method to implement inter-organizational workflows. In this method, the invocation of composed services is implicitly encoded within processes. State diagrams with transitions and specific rules called ECA are used to model a workflow. In addition, lightweight coordinators manage the state transition, i.e. when to enter or to finish the state. The composite service state diagram is translated into XML document, and control information of each state of the composite state chart is extracted into a routing table, which is uploaded to hosts of component services. The routing table is used by each of the coordinators as a knowledge base at runtime and contains information of location, peers and control flow routing policies. In comparison to our work, the scope of this work is inter-organizational level; where workflows already exists and must be connected to make choreography.

MOBIS (Baresi et al., 2004; Maurino & Modafferi, 2005) applies UML activity diagram instead of BPEL4WS, which is transformed to a new activity diagram

equipped with control delegation rules to coordinate process controllers. The new diagram is divided into sub-diagrams that are distributed to mobile devices to be run. This method also uses a static fragmentation before deciding where the fragments get run; however, our work focuses on runtime based decentralization.

In (Li Guo et al., 2008; L. Guo et al., 2005), a multi-agent platform to enact decentralized BPEL business processes is introduced as well. The key difference of this work with the similar multi-agent workflow decentralization is direct conversion of a business process to a multi-agent system, which results in reducing the validation and verification of interaction protocol. Coordination of agents is implemented through message passing. A protocol so-called LCC is used to interpret the specification of BPEL to agents and elaborates on how to relate the basic BPEL activities with the LCC protocol. In relation to this thesis, the LCC protocol can be an alternate way of communicating among fragment agents. Furthermore, fragmentation is not discussed accurately and seems to be in the finest activity level. In contrast, our opinion towards decentralization is to consider runtime feedbacks in fragment creation time.

In (Reese et al., 2006), an agent based approach is applied to inter-organizational workflow enactment and a Petri-net formalization is considered to describe workflows. Decentralization of a workflow into inter-organizational fragments is handled through a semi-automated process. The fragments are then encapsulated in agents and are activated using CAPA agent framework. However, this method shows the decentralization without showing any empirical result to prove it. In contrast to our work, the fragments produced in this work are considered at compile time without considering runtime environment conditions.

In (Fortino et al., 2006a, 2006b), an architecture is introduced to implement distributed workflow enactment so-called Agent-based Workflow Enactment Framework (AWEF) using JADE (Caire et al., 2008) platform. AWEF is a multi-agent system instantiated on the basis of workflow metaphor and creates a specific workflow engine for each workflow. The workflow engine mainly consists of a hierarchy of *ManagerAgents*, which is responsible for enacting of a specific workflow schema. A *ManagerAgent* runs several *TaskAgents* to execute specific workflow tasks. The hierarchical structure of *ManagerAgents* and *TaskAgents* lets convenient managing of workflows. This work provides a novel implementation of workflow engines; however, the policy of decomposing a business process to several sub-workflows is not clear and seems to be based on designers' point of view. Decentralization methods presented in this thesis can also be implemented by this new engine. This engine neglects the adaptability feature of workflow engines with runtime environment.

Several partitioning rules for decentralizing business processes have been introduced by (Baresi, Maurino et al., 2006). Partitioning logic must take into account both execution flow and data distribution. Execution flow partitioning is based on designer decisions at design time by applying several decentralization rules. Data distribution is also done based on the flow decentralization so that for each partition a set of variables is produced including local and global variables, which are accessible by the respective partitions. However, data dependency must be according to some rules and in some special situations there is no way to partitions the workflows. Based on a process control flow and the specified rules, data dependency graphs are also created. There also exist several rules to manage exception handlers among partitions. In comparison to our work, this work considers static fragments at

design time while our work is based on runtime fragmentation of workflows. However, this work gives good comments on sharing variables among fragments and managing exception handling as well.

In (Mirkov Viroli, 2007), a conceptual platform is used to wrap each activity of a BPEL process in an autonomous agent and the Linda tuplespace concept (Carriero & Gelernter, 1989) is applied to make ReSpecT objects with aim of mediating the communication of agents. The main feature of this architecture is isolating interaction, correlation and workflow concepts. A formal description of BPEL is introduced as well to make mapping of BPEL to the mentioned separated concepts conveniently. Due to the fact that Fully Process Decentralization (FPD) is considered as a fragmentation method in this work, it cannot support dynamic fragmentation of workflows. However, the mentioned architecture can be extended to create dynamic workflows at runtime.

Authors in (Wutke et al., 2008) overcome deficiency of centralized orchestration engines by presenting an architecture model for executing workflows as a set of self-coordinating components. The components are specified formally and Tuplespace concept of Linda is applied to mediate interaction of component. The presented system is in abstract level and enactment of workflows is independent from the fragmentation model. Then, it can be considered for implementing the dynamicity of the decentralization methods presented in this thesis.

NINOS (Li et al., 2010) is a distributed workflow engine, which applies PADRES publish/subscribe middleware (Fidler et al., 2005) to handle the communications of a workflow system. As shown in Figure 3.3, a business process is fragmented to the lowest activity level granularity and then each fragment type is encapsulated in a

dedicated agent. The fragment agents are deployed to NINOS runtime environment, afterwards. The PADRES publish/subscribe middleware is used to wiring the agents as well. Business processes are invoked through a client whose role is translating the invocations into the NINOS service requests.

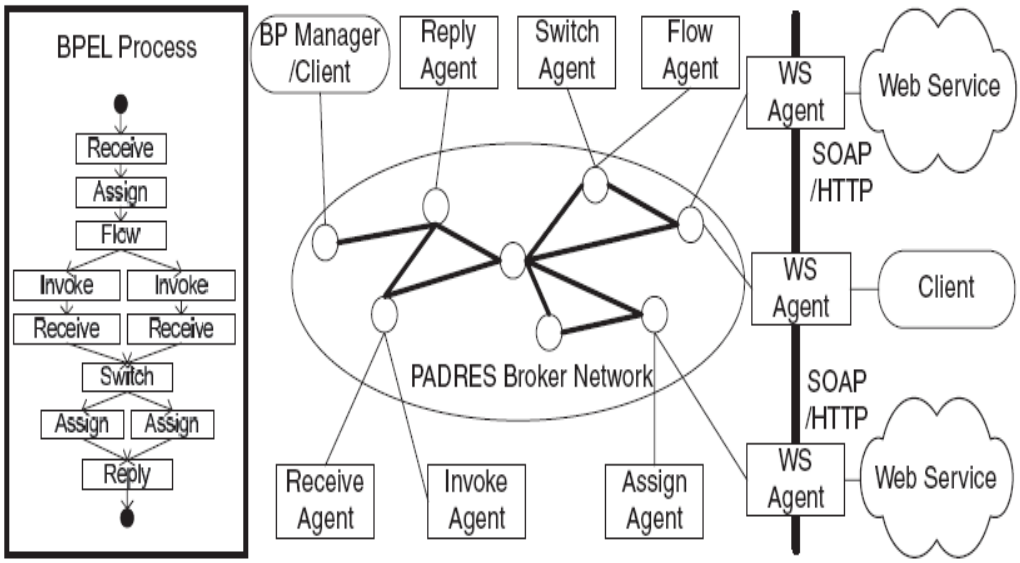


Figure 3.3: NINOS Distributed Business Process Execution Architecture

As the system is based on the finest granularity level; therefore, it is considerably flexible from different aspects. Agents can be deployed to any node across the network. In this system, accumulating the whole agents on one machine is considered equal to the Centralized method and alternatively the system is able to distribute them on different machines. Monitoring of agents is also possible through the publish/subscribe broker. Cross-organizational support of business processes can be another benefit of this system through migration of respective agents to the target organization. In addition, decomposition of a process to the finest level of granularity makes the system capable of managing load balancing and replication needs. Even activities that cause bottleneck at runtime can be duplicated to divide the load among two or more agents.

This thesis is also based on the NINOS (Li et al., 2010) view regarding business process decentralization and management. However, there are several aspects, which can be considered as shortcomings of the NINOS system and this work is targeted at finding solutions for them. First, the finest activity level of fragmentation may result in producing a number of fragments. It may increase degradation of system resources such as memory consumption due to number of fragments or waste of bandwidth due to number of messages exchanged among agents; second, as the number of fragments increases, the communication of fragments is increased and it depends on communication resources of system. Immediate effects would be increasing response-time and decreasing throughput of workflows; third, provided workflow fragments are not adaptable with current runtime environment. For instance, if just a machine is dedicated to a workflow engine, the same number of fragments is produced compared to the time when a number of machines are dedicated to the engine. A suitable solution should be able to produce only one fragment in case of one machine and consider proper number of fragments when more numbers of machines are available. Alternatively, when some activities are closely interrelated; they can be encapsulated in one fragment to omit the communication cost among the fragments.

In this thesis, a workflow system is presented which is able to provide fragments adaptable with the current runtime environment. Considering runtime requirements in decentralizing a business process does not mean that the flexibilities of the NINOS have gone. In fact, it can be a value added to the NINOS system. Simultaneously, another research work (Muthusamy et al., 2009) brings adaptability with runtime environment to the NINOS. In order to implement adaptability, SLA documents (Lamanna et al., 2003) are applied through which the management of system

thresholds becomes more convenient for administrators. Afterwards, several formulas are used to make a decision on moving/removing or locating/relocating of workflow agents from/to right machines. Several policies can be considered here such as cloning and moving agents to other machines in case of high load. Even, by incrementing or decrementing the number of machines, the system is able to move/remove agents to/from other machines. Due to the fact that the fragmentation of workflows in NINOS is based on the FPD, there are a number of fragments collaborating to enact a business process. Although this work brings flexibility to a runtime environment, it neglects the adaptation of fragments with runtime environment. This thesis addresses the fragment adaptation from the following aspects:

1) Frequent-path adaptability: by considering relations among workflow activities. In this case, a mining approach discovers closely interrelated activities of a business process by monitoring the behavior of previously executed workflows.
2) Proportional-fragment adaptability: by considering the proportionality of number of fragments with number of orchestration engine machines. The fragment adaptation can be complementary to (Li et al., 2010; Muthusamy et al., 2009) and the same policies for locating/relocating of agents can be applied consequently.
3) Available-bandwidth adaptability: by choosing the most suitable fragments, which are more commensurate with available bandwidth.

3.3.3 Dynamic Fragmentation Based on Workflow Circumstances

Dartflow project (Cai et al., 1996) has shown a usage of mobile agents in distributed workflow execution. In the Dartflow, the workflow model is fragmented

dynamically, and the partitions are carried by mobile agents and sent to different sites, which are responsible for them. This work establishes good points for dynamicity of error handling and data sharing among agents in a dynamic workflow system; however, it focuses on the system architecture and does not detail the fragmentation model, adequately.

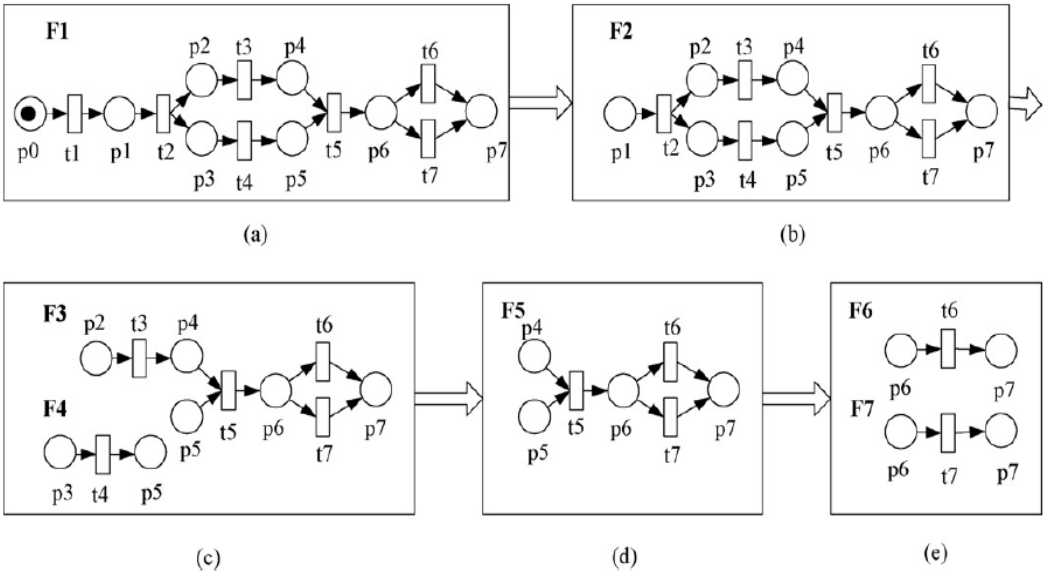


Figure 3.4: Dynamic Fragmentation of A Sample Process

An abstract and conceptual dynamic workflow fragmentation method is shown by (Tan & Fan, 2007), which applies Petri-nets formalism. The presented method partitions a centralized process into fragments step by step while the process is executed. The created fragments are able to migrate to proper servers, where tasks are performed and new fragments are created and forwarded to other servers (i.e. using mobile agents) to be executed. Figure 3.4 shows the on-the-fly creation of fragments. Process starts as a single fragment *F1* and after finishing *P0* a new fragment *F2* is created. Then there is a potential to decompose the flow branches into two fragments *F3* and *F4*. Fragment *F5* has to wait for the execution of *F4* to continue. The same policy may be considered to create fragments *F6* and *F7* where there is another Flow activity.

Dynamism in decentralization is a prominent aspect of this work. However, a natural result of this dynamism is the time cost of creating fragments at runtime. It can be mitigated by creating fragments beforehand in a fragment pool and even can be overlooked by evolving of processors. This work brings a considerable dynamicity to decentralization; although, it does not encompass any architecture to support this dynamicity. The fragmentation method of this work is different from our work in that it considers workflow runtime condition to decentralize a business process while runtime environment circumstances are applied for the same purpose in this thesis. Nonetheless, this method is similar to our work in that the fragments must be prepared beforehand in a fragment pool or must be built on the fly at runtime.

In (Atluri et al., 2007), a decentralized workflow model is presented for inter-organizational workflow decentralization, where inter-task dependencies are enforced without requiring to have a centralized WFMS. In the presented model, a workflow is divided into partitions called self-describing workflows, and handled by a light weight workflow management component, called workflow stub, located at each organization. This approach partitions a workflow as it progresses with its execution. As a result, the partitions are sent to an agent only when it is needed, whereas other approaches send the partitions to the execution agents even if they are not executed at all. Finally the presented decentralization algorithms are compared with a centralized model for several cases. This work is different from our work in that it considers a criterion to decentralize a business process from inter-organizational point of view, which is fitted to choreography layer of service oriented architecture. The produced fragments can be considered as inputs of our work for dynamic decentralization. This work is also an inter-organizational version of the research (Tan & Fan, 2007), which partitions the workflows on the fly. It is also

different from our work in that a workflow is partitioned based on workflow runtime conditions while this thesis is targeted to use runtime environment circumstances for decentralization purposes.

In (Gong et al., 2009), the ever-changing legislation of governments, customer needs and other changes in environment of business processes are introduced as the main reasons of implementing various forms of business process applications in different organizations. This work looks for a way of providing highest level of flexibility as well as adaptability to the changes in runtime environment. Although this work is interested in an abstract level, the ways of separating knowledge from business process and applying ECA (Event-Condition-Action) mechanism to determine the next step in a workflow execution at runtime is unique. By sticking to this approach static development of processes proceeds to a dynamic level. The decentralization methods in this thesis require flexible architectures to support dynamic fragmentation, which execute different execution forms of a business process a runtime.

3.3.4 Static Decentralization Using Process Embedded Communication Facilities

Decentralization methods classified in this category fragment a business process based on communication facilities embedded into process itself and are referred to as Semi Process Decentralization (SPD). These methods of decentralization also tend to decentralize a business process without considering a third party to handle interactions. In addition, they are more suitable to realize choreography of business processes where using a third-party middleware is not convenient. Due to using static criteria to decentralization, produced fragments are not compatible with ever-changing runtime environment.

The idea of fragmenting distributed relational databases is applied to decentralize Petri-net business processes in (Kirsten et al., 1998). A particular version of Petri-nets called predicate/transition model is considered in this work, which includes behavior and object-related aspects of workflows in a relational scheme. Fragmentation is inspired by the fragmentation model in distributed databases. Even similar to distributed databases, some fragments are replicated to improve the performance. Three kinds of workflow fragmentation are introduced in this work similar to relational distributed databases including horizontal, vertical, and diagonal. This method assumes that fragmentation must be predicated by designers. Fragmentation is started by splitting the predicates and assigning the split parts to the fragments and several fragmentation patterns are also introduced. In comparison to our work, fragments here are produced based on designers attitude at design time while fragmentation at runtime is considered in this thesis.

In (Yan et al., 2003), a conceptual peer-to-peer approach to decentralized workflow enactment is presented. At first, machines involved in executing business processes are grouped in virtual communities and process activities are then assigned to each community. During process deployment each task is deployed to respective nodes in a relevant community. The allocation of nodes to the communities and assigning tasks to each community are based on designers' opinion. However, distributing tasks to the nodes at runtime is an on-the-fly decision making based on the load of each node. In comparison to our work, this work uses static fragmentation at design time while our method is based on dynamic fragmentation. Also, allocation of tasks to machines involved in executing the processes is made at runtime. Our method is a basis to reconfiguration of the processes at runtime, which is neglected in the mentioned work.

Authors in (Chafle et al., 2004; Contenti et al., 2005) encompass business and government processes as composite services, which are executed based on web service orchestration. In addition, architecture of applying BPEL processes to address the problems of inter-organizational business processes is described. Although their focus is on decentralization of business processes; however, it implicitly implements choreography of business processes scattered among organizations running on different orchestration engines. The experimental work shows decentralized orchestration of composite web services takes less response-time in terms of number of requests and higher throughput compared to the centralized execution. Our study includes dynamic partitioning along with performance analysis of workflows inside an organization, not from inter-organization point of view.

A static fragmentation algorithm is presented by (Tan & Fan, 2005) so that each fragment in a process is considered to be executed in a predetermined server. Being designated activities to different sites, a process is decomposed to fragments and each fragment is executed by its own server. The presented algorithm duplicates shared places by transitions on different machines. In order to describe the workflows CWF-net and Petri-net formalizations are used. This work is also based on designers' decentralization at design time and does not consider adaptability of decentralization with runtime environment.

Authors in (Jung et al., 2006) introduce a methodology for decentralizing data-intensive scientific workflows. The decentralization part of this method includes three steps. In the first step, for each scientific experiment a server is assigned because the scientific workflows are intrinsically distributed among scientific laboratories. The second step is converting the experiments to workflows in each

server. Then, the workflows in different servers are wired to make a global workflow in step three. This work is different from our work in that it considers the choreography of workflows running on different machines while our method is classified in orchestration level, where activities of workflows collaborate to enact a workflow.

BPEL decentralization is formalized in an abstract layer by (Khalaf & Leymann, 2006). It partitions a business process so that each partition can be enacted by a different participant. In fact, it disconnects the partitioning itself from the design of a business process. Furthermore, it uses SOA messaging protocols and WSDL to wire decomposed components. Designers are able to split a process by defining partitions including a set of activities. Control links are split by applying more extra activities in the created partitions. Actually, a new type of BPEL is presented, which uses data links to model data flows. This work is extended by (Khalaf et al., 2008), which brings both control and data flow partitioning into the standard BPEL. These studies are fitted to our work in that the partitions created by designers at design time can be considered for dynamic decentralization based on the current environment circumstances at runtime, subsequently.

In (Sadiq et al., 2006), an approach has been presented to support automatic generation and linking of distributed process models from an integrated process definition in an abstract level. The integrated model is utilized for global monitoring and visualization while the distributed models are executable and each model runs in a local site. The objective in developing concepts described in this work is to devise a top down approach for integrated process modeling, which provides the support for automatic generation and linking of distributed process models. It does not consider the data dependency of process activities either. Our approach compared to this

method considers those business processes, which must be executed in a workflow engine and are not out of scope of a specific workflow engine.

BUST introduced in (Liu & Deters, 2007) applies an approach for implementing a scalable service oriented orchestration engine. It considers the service orientation as processing of inbound and generation of outbound messages. A general snippet template is considered to model the interaction of a basic service component and environment. This snippet is introduced as a disassembling point for orchestration. Four message exchange points (MEP) are introduced based on which it is ensured that there is no idle period inside a snippet. The snippets are implemented independently and are executed without a centralized control. By disassembling an orchestration to snippets, the actual execution of a business process is transformed to the snippets. The whole system is modeled by a virtual machine, which can be described by a state diagram. Actually, orchestration is message passing among the snippets. BUST orchestration state information means storing the state of snippets in a persistent storage. The BPEL is extended by some extra features to manage state information. As a matter of fact, this idea is different from our work in that the produced fragments or snippets are completely static. The produced snippets have to communicate with each other using message passing, although the communication model is improved by event-driven state transitions. This work also produces good comments on managing a process state and the introduced communication models can be used in our work as future work.

In (Yildiz & Godart, 2007), an abstract method has been introduced to nested decentralization of business processes. This work formalizes a method to transform a business process to several sub-processes and the total execution of the business process is achieved by enabling sub-processed to exchange data. Authors in (Yildiz

& Godart, 2007) also introduce another conceptual method to draw peer processes from a centralized specification. The criterion to detect peer processes is decentralizing based on P2P interactions with underlying web services and then the peer processes are wired. In comparison to our work, the created processes can be considered as input of dynamic decentralization.

An approach for process decentralization based on producing dependency tables of business process activities is introduced by (Fdhila et al., 2009). The presented technique analyzes current structure of a business process and extracts process activities and maps the structures to dependency tables. By considering the dependency of the activities, transitive dependency tables are created. The activities, which use similar web services, can be an example of such dependencies. Based on these dependencies sub-processes are extracted from a centralized business process specification. This work does not consider changing runtime circumstances and uses the dependency of the activities based on the process metaphor at design time. As a matter of fact, the sub-processes created by this method can also be decentralized at runtime based on the current runtime circumstances.

3.3.5 **Compiler Based Decentralization Approach**

Compiler based decentralization approaches improve business processes using compiler-inspired policies by parallelizing workflows activities. Generally, the compiler-inspired methods can be complementary to this thesis in that; 1) after analyzing and improving a business process, it can be used in our method to dynamic decentralization; 2) the produced fragments in our work can be improved by the compile-inspired methods to better runtime execution.

Authors in (Wodtke et al., 1996) transform a centralized workflow-like activity diagram into several fragments and (P. Muth et al., 1998) realizes it. The main goal is parallelizing the execution of different activities based on compiler-inspired methods. The focus of this work is partitioning flow activities into parallel partitions. They analytically show that how this method ameliorates system quality factors. This work can be complementary to our work in that an original version of a business process is used in this research; however, an improved process can also be used with aim of dynamic decentralization.

In (M. G. Nanda et al., 2004; Mangala Gowri Nanda & Karnik, 2003), a compiler-inspired method based on program analysis has been introduced. A Control Flow Graph (CFG) is used to automatic partitioning of a BPEL process similar to program partitioning in multiprocessors. In fact, a BPEL program is partitioned into parallel routines. Partitioning is done using data analysis on BPEL to gather required information for partitioning. Finally, the introduced method is proved by implementing several empirical experiments applying a loan application business process and then throughput and response-time are analyzed. The main objective of this work is decreasing load on servers which host the partial workflows. After optimizing a business process, it can be decentralized by the method presented in this thesis at runtime. In (Chafle et al., 2004), minimizing interactions among and maximizing parallelism of the executing components is emphasized. Advantages of adopting Java Messaging Service (JMS) (Richards et al., 2009) instead of HTTP to engine-to-engine communications are also highlighted.

Table 3.1: Brief Comparison of Most Relevant Business Process Decentralization Methods

	Process Representation	Decentralized Engine	Static/Dynamic Fragmentation	Adaptability with runtime Environment
Exotica (Alonso et al., 1995)	Compiled Process	Distributed Node Manager	Static Fragmentation (FPD)	No
Meteor (Das et al., 1997; Miller et al., 1996; Sheth et al., 1996)	Compiled Process	Task Manager	Static Fragmentation	No
Mentor (D. Wodtke, 1996; P. Muth et al., 1998)	State Chart	Workflow Server Engine (for each node)	Static Fragmentation	No
INCA (Barbara et al., 1996)	CORBA Component	Processing Agent	No Fragmentation	No
SELF-SERV (Benatallah et al., 2003; Sheng et al., 2002)	State Chart (Composite Web Service)	Coordinator at each peer	No Fragmentation	No
Chafle et al (Chafle et al., 2004; Nanda et al., 2004)	Graph-structured BPEL	Web service host	Static Fragmentation	No
MOBIS (Baresi et al., 2004)	UML activity chart	Mobile Peer	Static Fragmentation	No
Atluri et al (Atluri et al., 2007)	Graph-structured BPEL	Task Agent	Dynamic Fragmentation Based on workflow runtime Circumstances	No
NINOS (Li et al., 2010; (Muthusamy et al., 2009)	Graph-structured BPEL	Activity Agent	Static Fragmentation (FPD)	Yes
Wei Tan et al (Tan and Fan, 2007)	Petri Nets	--	Dynamic Fragmentation Based on workflow runtime circumstances	No
Viroli et al (Mirkov Viroli, 2007)	Graph-structured BPEL	Linda Objects	Static Fragmentation (FPD)	No
Fortino et al (Fortino et al., 2006a, 2006b)	Graph-structured BPEL	JADE Performer Agent	Static Fragmentation (FPD)	No
Our Approach	Block-structured BPEL	WADE/JADE Performer Agent	Dynamic Fragmentation based on runtime environment feedback	Yes

3.3.6 Comparison of Most Relevant Decentralization Methods

This thesis introduces three customizable workflow decentralization methods implemented by WADE/JADE multi-agent platform to support dynamic fragmentation. Authors in (Li et al., 2010) uses the FPD method and executes the fragments at runtime using a third-party multi-agent platform and consider adaptability at runtime by a complementary research work (Muthusamy et al., 2009). Indeed, the FPD and Centralized methods are considered in our experiments for evaluation purposes because: 1) the FPD is a subset of our decentralization methods; 2) The FPD is a basis for current dynamic systems, which use a third-party platform for implementation and reconfiguration and; 3) the Centralized method is also known as a traditional method and is a subset of our method. A brief comparison of the most relevant methods is shown in Table 3.1 in terms of process representation, decentralized engine, fragmentation method and adaptability.

3.4 Runtime Reconfiguration of Business Processes

Although reconfiguration of workflows is considered as future work, studying some reconfiguration-based workflow engines may enlighten the current idea of this thesis. Reconfiguration based on current runtime circumstances may come about due to two reasons. First, when new/suspended workflows are requested to be executed/re-executed; and second, a system monitoring agent shows that non-functional metrics in a system are being violated. In the former case, the requested workflows are re-fragmented based on the current system conditions while in the latter case the executing workflows are suspended, fragmented and executed again one by one until the system returns back to a normal situation.

In (Cao et al., 2006), a reconfigurable workflow engine is presented to overcome tremendous amount of concurrent request for workflows. In this approach, an enactment service is responsible to execute business processes. When demands for workflows are very high, a new workflow enactment service is created and submitted to new machines to handle some user requests partially. Through this approach, the performance and availability of engine is warranted. In addition, based on the system circumstances, the workflow services may be added or removed, which supports reconfiguration ability of the system. In comparison to our work, this system does not support decentralization of workflows at runtime and is equal to the Type-1 architecture, when a fragment includes the whole business process. This system is more similar to clustering methods. Previously, (Li et al., 2010) showed that decentralization outweighed clustering through several experiments.

An approach to migrate asynchronous concurrent portions of an old instance of a process to a new process model is shown in (Cicirelli et al., 2010). In this work, static reconfiguration means once a case is started, any change in a process is allowed only after completing the process. Dynamic reconfiguration can be made when a process is running without stopping or removing the previous processes. In order to evaluate this study, PN-Engine is used, which allows the versioning of Petri-nets specified business processes, dynamically. This method considers reconfiguration of workflows at runtime; however, it does not detail the management of workflow states and considers the new state information after reconfiguration. This work is similar to our work in that in our work different shapes of a workflow are created based on runtime conditions. It can be interpreted as if new versions of a workflow have been created while the older workflows are still running.

Work (Tsai et al., 2010) focuses on applying parallel servers to improve the performance of centralized engines. Due to ever-changing rate of user requests, allocating a static number of servers is not efficient. Indeed, the intention of this work is to use dynamic resource provisioning so that a dynamic number of servers is allocated based on varied number of requests in a specific time. This work is different from our work in that clustering execution of workflows is focused on; however, workflow decentralization based on current system circumstances is given attention in this thesis.

3.5 Using Mining Approach in Business Process Decentralization

In this section, at first the history of using process mining as well as frequent path detection in business processes is studied and later some motivator examples of applying a mining approach in decentralizing business processes are presented. It is worth mentioning that there is no novelty in process mining and frequent path detection in this section; however, using a mining approach in business process decentralization is introduced for the first time in this thesis.

3.5.1 Process Mining and Frequent Path Detection

Several research studies have been conducted toward building models without a priori knowledge of process specification, called Process Mining. Using process mining, one can look for the presence or absence of certain patterns and deduce some process models from it (Dustdar & Gombotz, 2006; Van der Aalst et al., 2003). Although a variety of process mining algorithms have already been developed; none of them considered mining approaches to business process decentralization and the adaptability of business process with runtime environment. In (Dubouloz & Toklu, 2005), an algorithm is presented to detect the most frequent paths of graph-structured

BPEL processes. It is similar to our work in that both of them know the business process description and each record of the process log file addresses an activity execution in a workflow. In our work there is no novelty in BPEL mining and any frequent path detection algorithm can be used; nonetheless, an algorithm is presented to decentralize a block-structured business process considering both frequent path and granularity. In addition, the mining of graph-structured BPEL processes covers the block-structured BPEL; however, the focus is on both decomposing and mining of processes. Furthermore, in the Appendix A, several motivating examples of using mining approach in business process decentralization are analytically discussed. In this current study, both HPD and HIPD decentralization methods decompose a business process based on levels of a process tree and in addition to it, the HIPD method mines process log data to detect frequent paths of business processes. Indeed, more attention is paid to the block-structured BPEL and the graph-structured model will be focused in future work.

3.6 Overview of Fuzzy Logic

The possibility theory was extended by (Zadeh, 1965) into a formal system of mathematical logic to bring the concepts of working with fuzzy natural language terms to engineering and mathematic. The author of (Zadeh, 1965) invented a branch of logic that uses degrees of membership in sets rather than a strict true/false membership.

In fuzzy logic, fuzzy terms in natural languages are referred to as linguistic or fuzzy variables. Terms such as hot, cold, height etc. are some examples of such linguistic variables. Fuzzy logic also brings a new reasonable concept to traditional sets. It assigns membership values between 0 and 1 that reflect more naturally a member's

association with the set. A linguistic variable carries with it the concept of fuzzy set qualifiers, called hedges. They include adverbs such as very, somewhat, quite, more or less and slightly.

Fuzzy logic treats a fuzzy set as a fuzzy proposition. A fuzzy proposition is a statement that asserts a value for some given linguistic variables such as "*height is tall*" and it can be presented as *X is A* where A is a fuzzy set on the universe of discourse X. A fuzzy rule relates two fuzzy propositions in the form *IF X is A THEN y is B*; where x and y are linguistic variables; and *A and B* are linguistic values, determined by fuzzy sets on the universe of discourses *X and Y*. For instance, IF height is high THEN weight is light.

In a fuzzy system, all rules fire to some degree, or in other words they fire partially. If the antecedent is true to some degree of membership, then the consequent is also true to that same degree. In (Mamdani & Assilian, 1975), four steps for a fuzzy system are recommended including 1) fuzzification of the input variables; 2) rule evaluation; 3) aggregation of the rule outputs; and 4) defuzzification.

Alternatively, (Sugeno & Kang, 1988) suggests using a single spike, a singleton, as the membership function of the rule consequent. A singleton or more precisely a fuzzy singleton is a fuzzy set with a membership function that is unity at a single particular point on the universe of discourse and zero everywhere else. Sugeno-style fuzzy inference is very similar to the Mamdani method. Sugeno changed only a rule consequent and used a mathematical function of the input variable instead of a fuzzy set. IF *(x is A)* AND *(y is B)* THEN *z is f (x; y)*; is an example of a Sugeno-style fuzzy rule, where x, y and z are linguistic variables; *A and B* are fuzzy sets on universe of discourses *X and Y*, and *f(x; y)* is a mathematical function. The most

commonly used zero-order Sugeno fuzzy model applies fuzzy rules whose THEN part takes the form THEN *z is k*, where *k* is a constant. In this case, the output of each fuzzy rule is constant. All consequent membership functions are represented by singleton spikes. The evaluation of the Sugeno-style fuzzy rules is according to Figure 3.5.

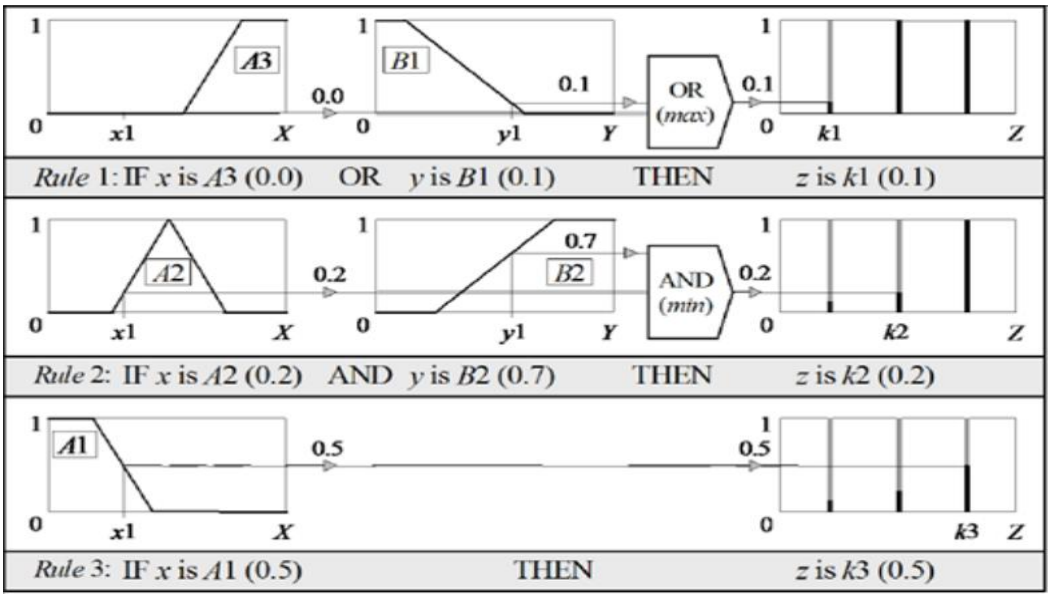
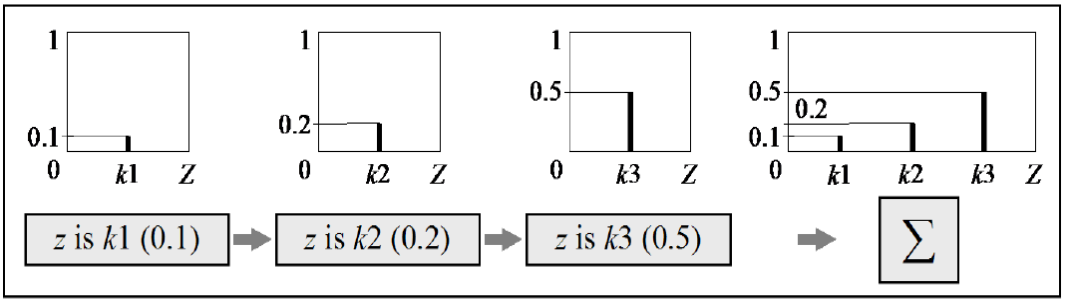


Figure 3.5: Sugeno Style Rule Evaluation



WA = (0.1*20)+(0.2*50)+(0.5*80)/(0.1+0.2+0.5) = 65

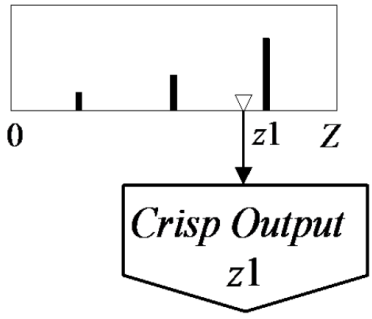


Figure 3.6: Sugeno-Style Defuzzification

The weighted average can be calculated according to Figure 3.6. In addition, the Sugeno method is computationally efficient and works well with optimization and adaptive techniques, which makes it very attractive in control problems, particularly for dynamic nonlinear systems. The mentioned aspects of the fuzzy logic are later applied in this thesis to implement a fuzzy decentralization decision making unit.

3.7 Summary

In this chapter, a brief history of decentralizing business processes was introduced. First, the idea of open-world paradigm was overviewed. The idea referred to the ability of software to react to its environment. This thesis applied the idea to bring adaptability to business process decentralization. Later, the business process decentralization was studied from three aspects fragmentation, enactment and adaptability. The most relevant research studies reviewed in the literature were briefly compared in a table at the end of the section. Follow that a mining approach was introduced in business process decentralization. Finally, this chapter ended up with an overview of the fuzzy logic and its application in our work.

CHAPTER 4

RESEARCH METHODOLOGY

4.1 Introduction

This chapter presents the quantitative methodology adopted in this research. Section 4.2 reviews the research problem and objectives. An outline of research steps is introduced in Section 4.3. Benchmarks and reference business processes are presented in Section 4.4 followed by the experiment platform in Section 4.5. Experimental setup is shown in Section 4.6 and runtime evaluation metrics are introduced in Section 4.7. Section 4.8 encompasses model verification and validation of decentralization approaches and an adaptable decentralization decision making unit. In the end, Section 4.9 summarizes the chapter.

4.2 Research Overview

The focus of this study is on decentralizing workflows by creating fragments adaptable with runtime environment circumstances. Three aspects of runtime adaptability in decentralization are studied in this thesis. The first aspect is the *frequent-path* adaptability (frequent paths), which is equal to detecting high relevant activities and encapsulating them in the same fragment to omit the communication cost among the activities. The *proportional-fragment* adaptability (fragment proportionality) is another adaptability aspect, which is analogous to the proportionality of produced fragments with number of machines to extenuate the internal communication among fragments on the same machine. The last one is the *available-bandwidth* adaptability, which means process fragmentation based on current circumstances of a communication medium. An ever-changing runtime environment along with the mentioned adaptability aspects may result in producing a

variety of business process fragments at runtime. Considering such dynamicity in workflow execution raises several issues, which can be classified in three categories. The first problem is lack of framework to support architectures to execute dynamically created fragments, decentralization methods in order to fragment workflows based on runtime circumstances, and methods to map runtime conditions to decentralization methods. The next problem is that current decentralization methods do not consider the frequent-path and proportional-fragment adaptability aspects in creating fragments, which negatively affects response-time and throughput of decentralized workflows. Finally, current decentralization methods do not consider the available-bandwidth adaptability in creating fragments, which negatively affects bandwidth-usage of decentralized workflows.

In addition, the main objectives of this research are first, to propose a framework including architectures to support execution of dynamically created fragments, decentralization methods to create adapted fragments, and methods to map runtime conditions to decentralization methods; second, to improve response-time and throughput of decentralized business processes applying the frequent-path and proportional-fragment adaptability aspects; third, to improve band-width usage of decentralized business processes applying available-bandwidth adaptability.

4.3 Research Steps

This research focuses on dynamic fragmentation of business processes based on current system circumstances. Accordingly, the following steps have been taken to satisfy the research objectives as shown in Figure 4.1.

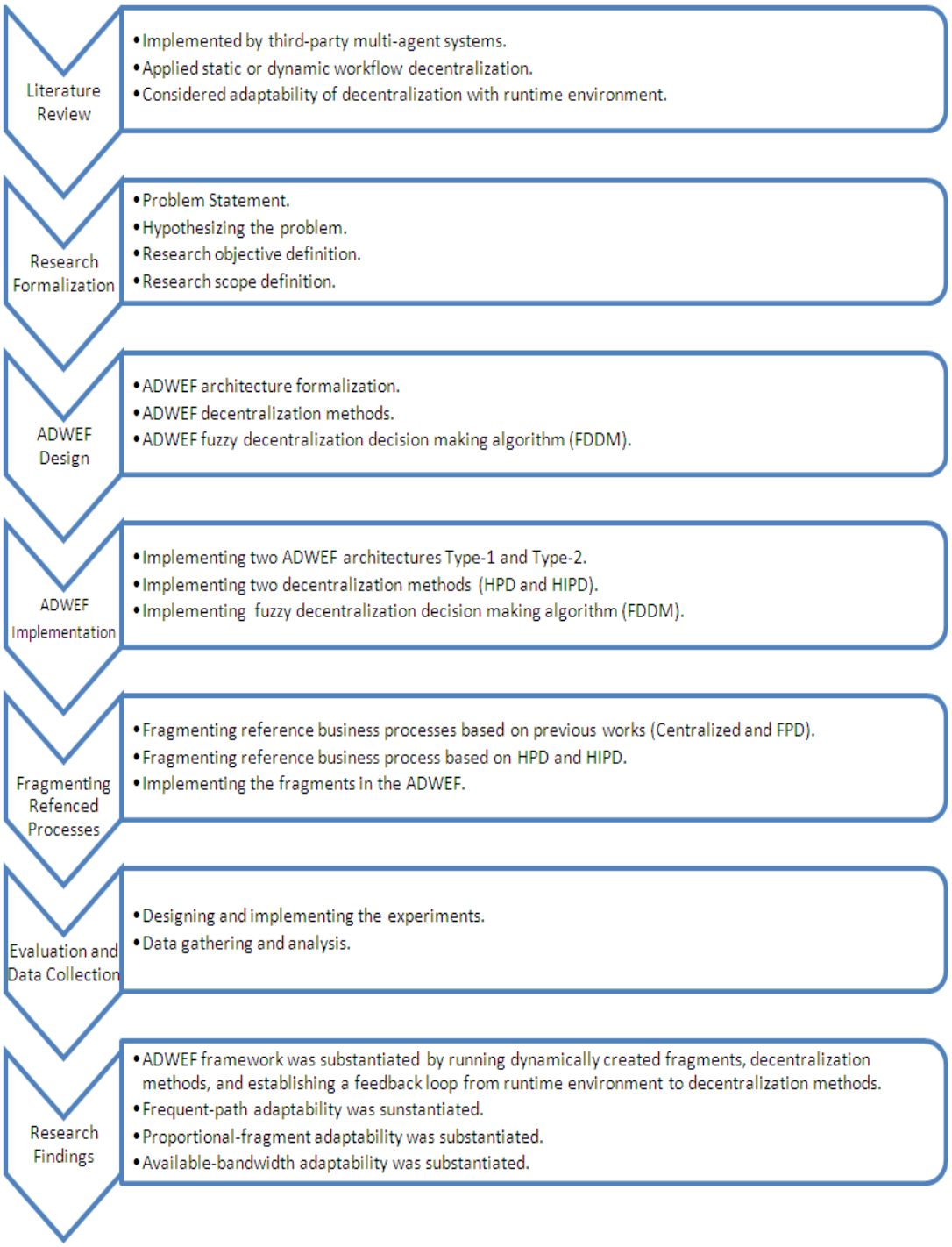


Figure 4.1: Research Steps Highlights

1. **Review of Business Process Decentralization**: The service oriented architecture was studied with more focus on the orchestration and choreography layers. The BPEL language and several orchestration engines such as Oracle (ORACLE, 2011a) and ActiveBPEL (Active-Endpoints, 2011) engines were studied to find out how

BPEL specified business processes are designed and enacted by these centralized engines. The literature of workflow decentralization was reviewed as well. More stress was on studying decentralized workflow engines with the following characteristics:

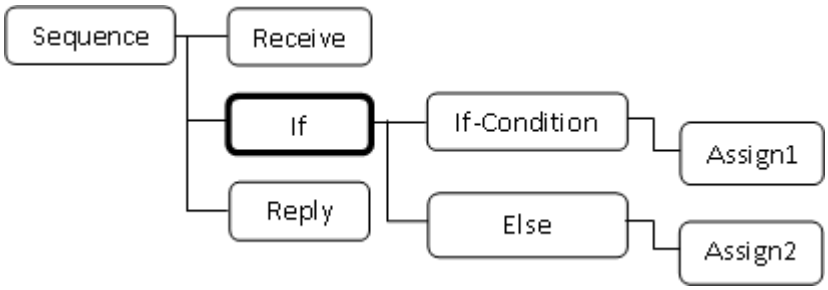
- Implemented by third-party multi-agent systems.
- Applied static or dynamic workflow fragmentation.
- Considered adaptability of decentralization with runtime environment.

Research works (Li et al., 2010; Muthusamy et al., 2009) were specifically fitted to the above areas. They recommended a Fully Process Decentralization (FPD) approach, which statically fragmented a business process in an activity level, implemented by a multi-agent system and considered adaptability with runtime environment.

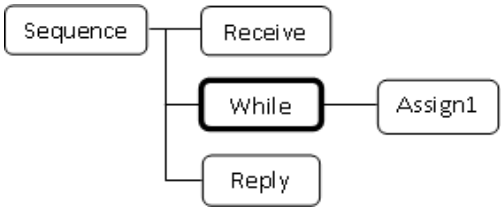
2. **Research Formalization:** in this step problems of the research were identified and hypothesized. Research objectives were accordingly introduced and then scope and constraints of research were presented.
3. **ADWEF Design**: A framework was designed with the aim of decentralizing business processes considering current runtime circumstances. The purpose was to produce process fragments dynamically and more commensurate with current runtime circumstances. Indeed, two architectures were recommended to support dynamically produced fragments. Two decentralization methods were presented, which were customizable enough to produce various fragments. The first method decomposed a block-structured business process based on hierarchy of activities (HPD) while the second method (HIPD) was based on the HPD method along with a mining approach, which decomposed a business process to frequent and infrequent paths. The mentioned methods were evaluated against the traditional

Centralized and Fully Process Decentralization (FPD) methods. A Fuzzy decentralization decision making unit (FDDM) was presented to choose a suitable decentralization method based on current number of machines and available bandwidth.

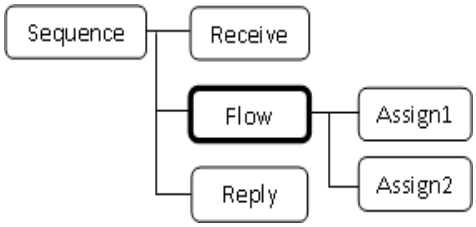
4. **ADWEF Implementation**: the designed architectures were realized by two implementations namely Type-1 and Type-2 based on WADE/JADE multi-agent platform. Moreover, the decentralization methods and FDDM unit were implemented applying the Java language.
5. **Fragmenting Reference Processes**: Several business processes including three basic as well as a complex loan application business process were applied in this research. The processes were fragmented based on these methods along with the implementation of the fully process decentralization (FPD) and the Centralized models.
6. **Evaluation and Data Collection**: Several experiments were designed and implemented to study first, boundary conditions of platform; second, the frequent-path, proportional-fragment, and available-bandwidth adaptability aspects of decentralization methods.
7. **Research Findings**: After analyzing the collected data, the three aspects of adaptability were studied. By running the experiments, the abilities of the ADWEF framework in running dynamically created fragments, applying decentralization methods, and establishing a feedback loop from runtime environment to decentralization methods were proved. In addition to this, the results substantiated the adaptability of fragmentation from frequent paths, fragment proportionality and available bandwidth views.



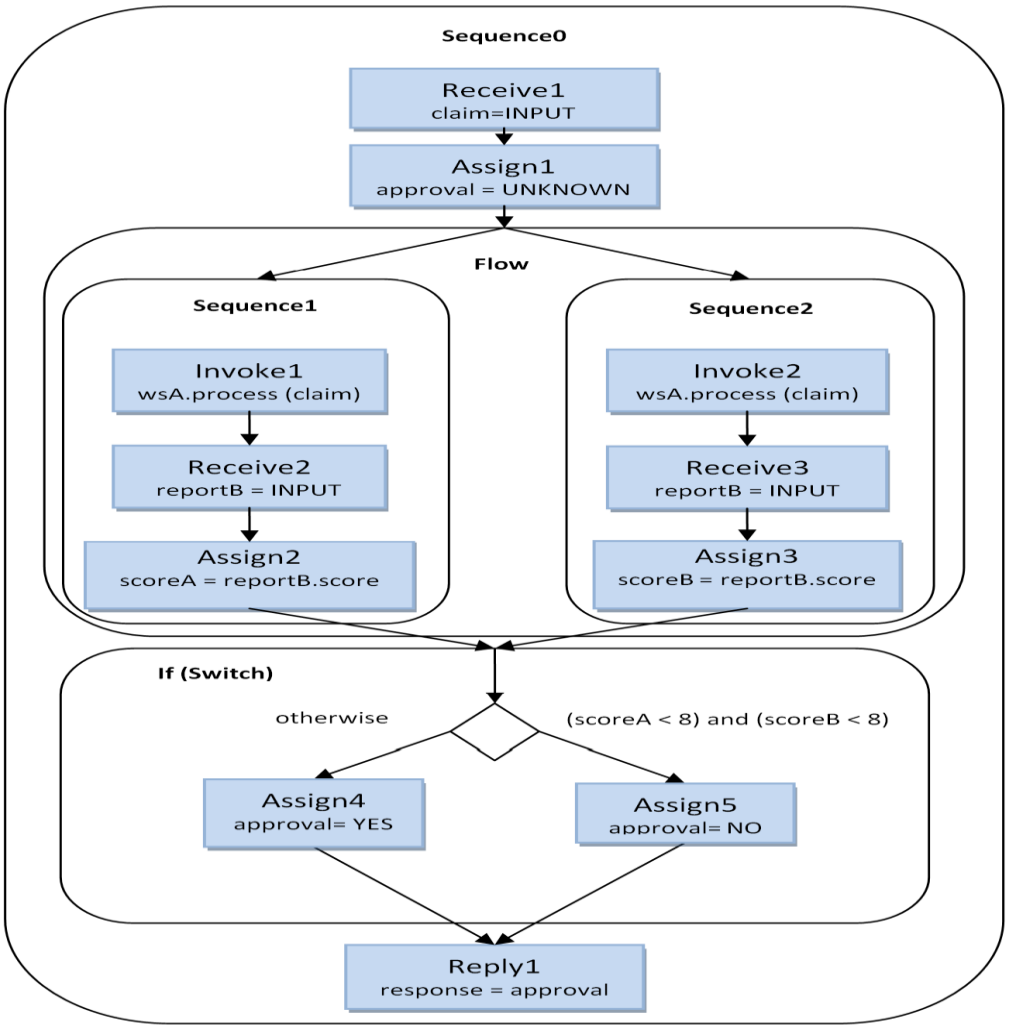
A) Basic If-process



B) Basic While-process



C) Basic Flow-process



D) BPEL View of Loan-process As A Complex Process

Figure 4.2: Three Basic Processes, One Complex Process

4.4 Referenced Business Processes

As a matter of fact, there is no standard business process to benchmark BPEL processes. Nonetheless, (Chafle et al., 2004; Khalaf & Leymann, 2006; Li et al., 2010; Muthusamy et al., 2009) have used a loan application BPEL process in their work as a reference model. In this study, several BPEL processes are applied as illustrated in Figure 4.2(A-D). The BPEL benchmark processes are selected with purpose of studying basic processes illustrated in Figure 4.2(A-C) and complex processes depicted in Figure 4.2(D). The basic BPEL processes are used to evaluate the behavior of their central activity such as If (Switch), While and Flow. On the other hand, the complex BPEL process paves the way for studying several simple and structured activities together as it is the case for the loan application process.

The Loan-process includes several structured as well as simple activities such as Sequence, Flow, If (Switch), Receive, Invoke, Assign and Reply. The Loan-process consults with two external web services, which send a credit report for the loan applicant. In order to refrain from approbating risky loans, the loan request is accepted, provided that both web service reports confirm the applicant's credit.

4.5 Experiment Platform

In this section, at first reasons of applying a multi-agent middleware are explained, and then WADE/JADE platform is introduced, which is used in implementing the whole experiments in this thesis.

4.5.1 Applying a Multi-agent System to Implement ADWEF

An agent is essentially a special software component that has autonomy that provides an interoperable interface to an arbitrary system and/or behaves like a human agent, working for some clients in pursuit of its own agenda. Multi-agent systems (MAS)

can model complex systems and introduce the possibility of agents having common or conflicting goals. These agents may interact with each other both indirectly (by acting on the environment) or directly (via communication and negotiation). Agents may decide to cooperate for mutual benefit or may compete to serve their own interests (Bellifemine et al., 2007).

In this thesis, agents cooperate to execute fragments of a business process while the fragments are produced considering the three aforementioned adaptability aspects. Agents encapsulate fragments and are then deployed into network machines. In extensions of this thesis, business processes will be expected to be re-decentralized, re-encapsulated, and re-deployed based on runtime circumstances. A fragment inside a business process might be overloaded that is needed to be replicated and moved to other machines. Agents are also required to be moved from one machine to another machine in case of a machine crash or when an agent is required to closely interact with another agent on another machine. The needs to implement re-decentralization, re-deployment, replication, and re-location of agents in the extensions of this thesis motivated this research to be implemented by a multi-agent system. In addition, based on Figure 3.2 presented in Section 3.3, several decentralized workflow engines are implemented applying a multi-agent third-party middleware. These are the reason why this research was motivated to apply a multi-agent system.

4.5.2 **WADE/JADE Multi-agent Platform**

JADE (Bellifemine et al., 2007; Bellifemine et al., 1999, 2001) is a popular open-source middleware conceived to facilitate the development of distributed applications based on an agent-oriented paradigm. It also provides a distributed runtime environment based on the “agent” and “behavior” abstractions, P2P communication

among agents, basic agent lifecycle management and discovery mechanisms. JADE is also efficient enough to implement distributed agent-based applications (Bellifemine et al., 2007; Chmiel et al., 2005). WADE (Workflow and Agent Development Environment) (Caire et al., 2008) represents a main evolution of JADE as well. WADE adds to JADE the support for the execution of tasks defined according to a workflow metaphor and a number of mechanisms that help managing the complexity of distribution in terms of both administration and fault tolerance. WOLF (Caire et al., 2008) is also an Eclipse (Chmiel et al., 2005) plug-in and allows WADE developers to exploit full power of the Eclipse IDE plus additional WADE-specific features.

According to (Caire et al., 2008), WADE provides an extension of the basic agent class of the JADE library called *WorkflowEngineAgent* (*WFE*) that embeds a small and lightweight workflow engine (sometimes called micro-workflow engine or performer agent). A *workflow behavior* is a Java class that inherits from a workflow class and can be executed by a WADE workflow-engine agent, which plays a performer role. It means a WFE runs special behaviors called workflows. Basically, it is possible to run one or several workflows under the same performer. In addition, workflow activities are mapped to methods of a workflow class. Among these activities there is a sub-flow activity that calls another workflow running under the same or different performer agent. In this thesis, WADE/JADE-based agents are also applied to realize the runtime components in all experiments.

In order to count the number of exchanged messages, a JADE plug-in is used called Java Sniffer (Bellifemine et al., 2007; Bellifemine et al., 2008; Bellifemine et al., 2001), which listens and logs exchanged messages among agents. According to WADE (Bellifemine et al., 2007; Bellifemine et al., 2001), to invoke a sub-flow in a

regular workflow, three messages are exchanged including REQUEST, AGREE (workflow correctly loaded) and INFORM (workflow execution completed). In the case of transactional workflows, five messages are exchanged including CFP (transactional request), AGREE (workflow correctly loaded), PROPOSE (workflow execution completed), ACCEPT_PROPOSAL (commit) and INFORM (commit OK) (Caire et al., 2008). In this work, workflows are entirely run in a non-transactional mode in all experiments.

4.6 Experimental Setup

The configuration of experiment environment is shown in Figure 4.3. In order to implement experiments, WADE was run on Sun's JDK 1.6.0 to manage the performer agents. On the client side, graphical WADE/JADE remote agent management console were run. Moreover, client and server sides executed workflow agents by means of *StartBootDaemon* process. All machines were also of type Intel(R) Core(TM) 2 Duo CPU 2.40 GHz with 3.23 GB RAM memory.

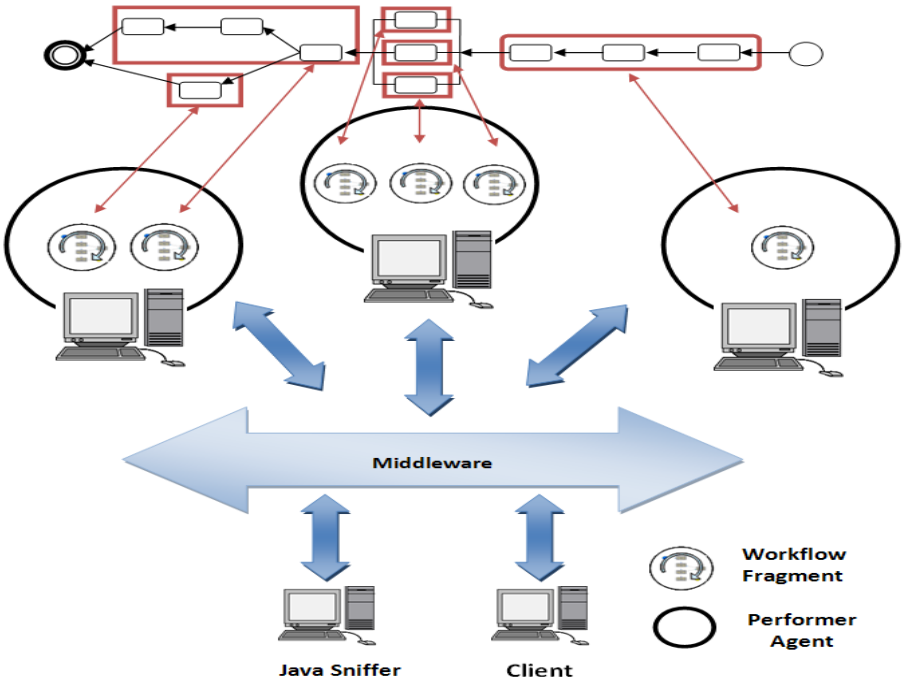


Figure 4.3: Experiment Environment Configuration

In all the experiments, the client was a multi-thread WADE/JADE agent running on a Linux Suse Real-time version (Novell, 2011) evolved with Java Real-time version (ORACLE, 2011b). The client ran a number of client threads, which called the server side workflows. Requests with different speeds were sent to server machines by the client threads. Each client request resulted in creating a thread in the server side for each workflow fragment. Receiving replies from the fragments, the client measured response-time and throughput using gathered data.

In order to address the experiments conveniently, each experiment was named according to the following naming convention: *<decentralization method name><granularity level>*'_'*<number of fragments>*'_'*<number of server machines>*. For instance, HPD3_4_2 reveals that a business process is decentralized by the HPD policy in the third level of the process tree (G=3) and four fragments are created and executed by two server machines.

4.7 Runtime Evaluation Metrics of Workflows

This section introduces evaluation metrics applied by experiments in the following sections. Four metrics are introduced including bandwidth-usage, response-time, throughput, and adaptability. Among the mentioned metrics, the bandwidth-usage is an independent variable in a distributed workflow engine in that it effects on response-time, throughput and adaptability while it may be decreased or increased by other applications which use network media. The response-time and throughput are variable that depend on request rate and size of messages which implicitly reveals their dependence to communication media and bandwidth-usage. The adaptability is also a dependent variable and is affected by response-time and throughput.

4.7.1 **Bandwidth-usage**

In a workflow management system, response-time and throughput of workflows are affected by several factors such as processing power, available memory, workflow behavior and most importantly available bandwidth, which determines capacity of communication media. In a distributed workflow system, where different components have to communicate through the communication media, protecting the consistency of the media is of high importance. Accordingly, a right metric can be the number of exchanged messages. When the number of exchanged messages is high, more bandwidth will be used and eventually more delay will appear in communication that finally results in higher response-time and less throughput. The *number of exchanged messages* among business process fragments is analogous to the amount of bandwidth usage. In order to measure it, the number of exchange messages among distributed components is counted.

4.7.2 **Response-time**

Response-time is the time difference between invoking a workflow agent until receiving a reply from it. The response-time is expected to be as shorter as possible and close to a real-time response. The response-time can be affected by the request rate as well as message size. In order to omit the effect of request rate on response-time, requests are sent to servers with constant rate while message size varies. The omission of message size takes effect when the size each message is zero while requests with various rates are sent to servers.

4.7.3 **Throughput**

Throughput is the number of completed requests per time unit divided by the total number of requests. It is usually shown in terms of percentage and illustrates the

power of system in processing requests during a time unit which is the reason why it is expected to be as high as possible. The throughput can be affected by the request rate as well as message size. In order to omit the effect of request rate on throughput, requests are sent to servers with constant rate while message size varies. The omission of message size takes effect when the size of each message is zero while = requests with various rates are sent to servers.

4.7.4 **Adaptability**

Adaptability of each decentralization method is a dependent metric and can be measured by improvement of response-time, throughput and bandwidth-usage. In this thesis, the runtime adaptability of fragments is studied from three aspects; first, the *frequent-path* adaptability, which is equal to detecting high relevant activities and encapsulating them in the same fragment to omit the communication cost among the activities; second, the *proportional-fragment adaptability*, which is analogous to the proportionality of produced fragments with number of machines to extenuate the internal communication among fragments on the same machine; third, the *available-bandwidth* adaptability, which means producing the process fragments based on ever-changing circumstances of a communication medium.

The mentioned adaptability aspects are evaluated individually based on the above metrics. The frequent-path adaptability is evaluated by comparing number of exchanged messages, response-time and throughput of HIPD fragments with fragments of other methods. The proportional-fragment adaptability is evaluated by the comparison of response-time and throughput of HPD and HIPD fragments applying different number of machines. The available-bandwidth adaptability, which is implemented by the FDDM unit, is evaluated by counting the number of

exchanged messages with absence/presence of the frequent paths along with the fragment proportionality and a simulated bandwidth.

4.8 Model Verification and Validation

This section presents the quantitative methodology adopted to collect data for evaluating the mentioned metrics. Several experiments are presented to evaluate the fragmentation methods and fuzzy decentralization decision making unit. Evaluation metrics and method for each experiment are subsequently presented.

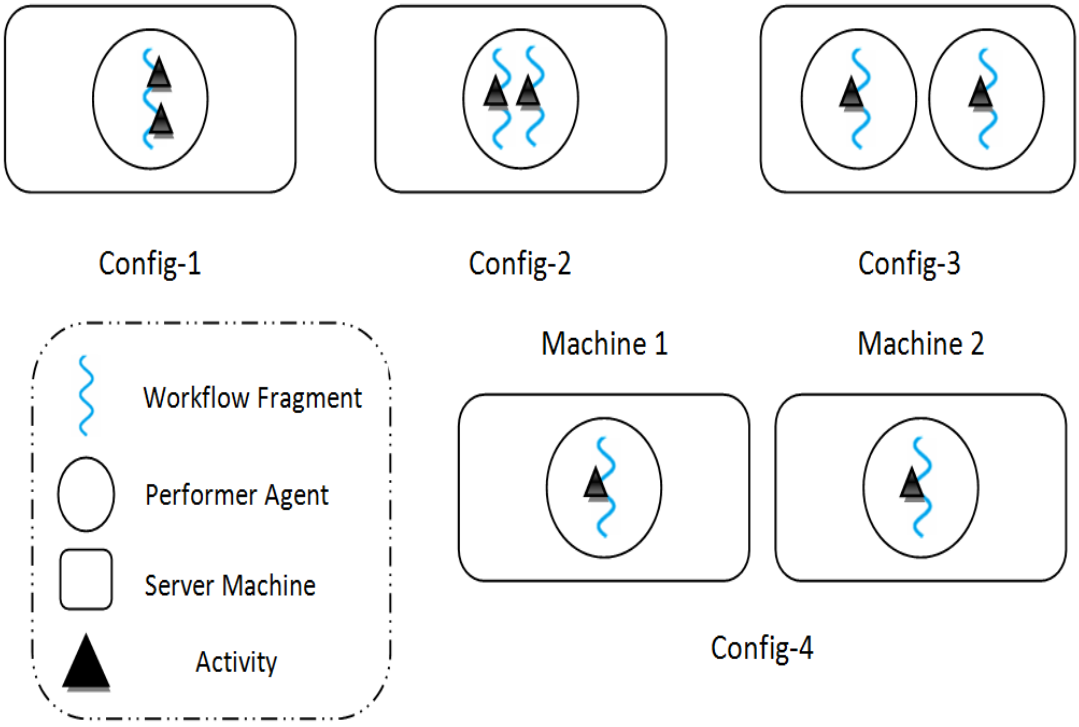


Figure 4.4: Four Boundary-condition Configurations

4.8.1 Boundary-condition Experiments

The main goal of boundary-condition experiments is to study the behavior of basic fragments running on basic agents. In this section, four boundary-condition configurations are introduced. Evaluation metrics and methods are later introduced and finally four experiments and their configurations are presented.

Four Boundary-condition Configurations

Four possible configurations to evaluate basic fragments are shown in Figure 4.4. In Config-1, which is equal to the Centralized method, the whole process is considered as one fragment. Process activities are also implemented as regular programming methods. The process may run the methods as threads communicating using shared variables without message passing.

The Config-1 shows one fragment including two activities. The activities are implemented as two simple functions and the first activity calls the second one ***n*** times. In Config-2, -3, and -4 each fragment includes one activity, which is analogous to a regular programming method. Being received a request from the client, the first fragment calls the second fragment ***n*** times. In Config-2, the fragments are run on the same performer agent and communicate internally. The Config-3 executes the fragments on different performer agents; however, on the same machine and communicate internally. In Config-4, the fragments are executed on different performer agents and machines so that they communicate externally. The comparison of the execution results envisages the costs of the executing fragments in the Centralized model without message passing, in the distributed model using internal message passing, and again in the distributed model applying external message passing.

Evaluation Metrics

Three metrics response-time, throughput and number of exchanged messages are used to evaluate basic agents in WADE/JADE platform to execute two basic fragments.

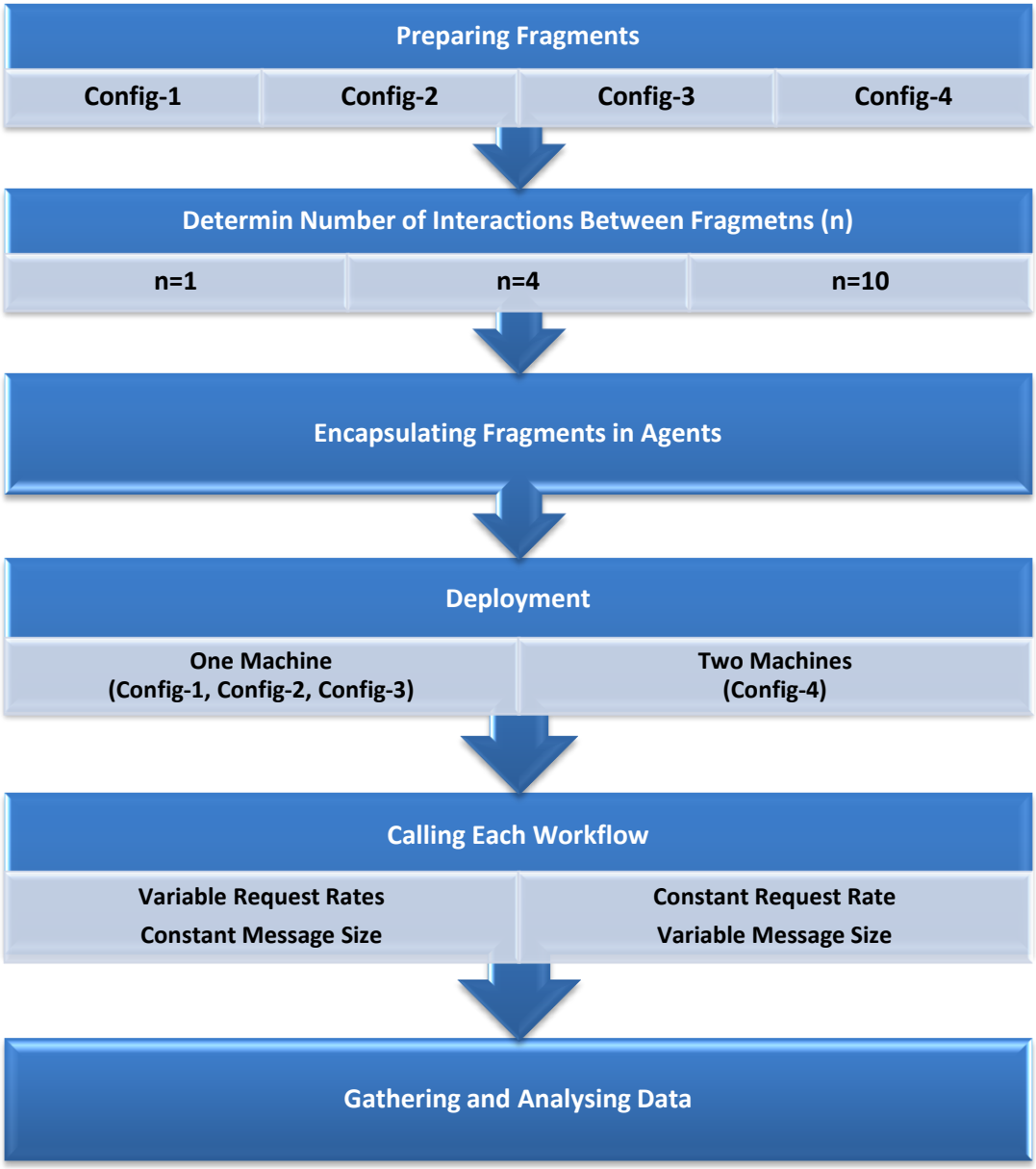


Figure 4.5: Evaluation Method of Boundary-condition Configurations

Evaluation Method

Figure 4.5 shows the evaluation of the four boundary-condition configurations. Based on the selected configuration, one or two fragments are produced. The fragments are encapsulated in agents and the agents are deployed to one or two machines based on the selected configuration. The workflows are called by a client in two different ways: 1) variable-request-rate, constant-message-size; 2) constant-request-rate, variable-message-size. The response-time and throughput are affected

by the request rate and size of messages passed through process activities. Thus, two categories of experiments are designed. The first category considers variable-request-rate and constant-message-size to omit the effect of message size. The second category considers constant-request-rate in presence of message size with varied values 50KB, 100KB, 150KB, 200KB and 250KB.

All experiments are targeted at measuring response-time, throughput and number of exchanged messages and they apply basic fragments executed by basic agents in WADE/JADE platform based on the four configurations. Experiment-1 and Experiment-2 evaluate response-time and throughput of basic fragments with variable-request-rate and constant-message-size, respectively. Experiment-3 and Experiment-4 evaluate the same metrics with constant-request-rate and variable-message-size. Experiment-5 measures the bandwidth-usage of fragments in four configurations.

Experiment-1 (evaluating response-time, variable-request-rate, constant-message-size)

The main goal of this experiment is to evaluate the effect of request rate on response-time of basic fragments executed by basic agents with variable-request-rate, constant-message-size. The experiment is configured according to Table 4.1.

Table 4.1: Experiment-1 Configurations

Number of Machines	Number of Iterations (k)	Request Rate	Messages Size	Configuration Type	Evaluation Parameter
1,2	1, 4, 10	Variable 1-600 req/sec	Constant ~zero	Config1, 2, 3, 4	Response-time

Experiment-2 (evaluating throughput, variable-request-rate, constant-message-size)

The main goal of this experiment is to evaluate the effect of request rate on throughput of basic fragments executed by basic agents with variable-request-rate, constant-message-size. The experiment is configured according to Table 4.2.

Table 4.2: Experiment-2 Configurations

Number of Machines	Number of Iterations k)	Request Rate	Messages Size	Configuration Type	Evaluation Parameter
1,2	1, 4, 10	Variable 1-600 req/sec	Constant ~ zero	Config-1, 2, 3, 4	Throughput

Experiment-3 (evaluating response-time, constant-request-rate, variable-message-size)

The main goal of this experiment is to evaluate the effect of request rate on response-time of basic fragments executed by basic agents with constant-request-rate and variable-message-size. The experiment is configured according to Table 4.3.

Table 4.3: Experiment-3 Configurations

Number of Machines	Number of Iterations (k)	Request Rate	Messages Size	Configuration Type	Evaluation Parameter
1,2	1, 4, 10	Constant 50, 500 req/min	Variable 50, 100, 150, 200, 250 KB	Config-1, 2, 3, 4	Response-time

Experiment-4 (evaluating throughput, constant-request-rate, variable-message-size)

The main goal of this experiment is to evaluate the effect of request rate on throughput of basic fragments executed by basic agents with constant-request-rate, variable-message-size. The experiment is configured according to Table 4.4.

Table 4.4: Experiment-4 Configurations

Number of Machines	Number of Iterations (n)	Request Rate	Messages Size	Configuration Type	Evaluation Parameter
1,2	1, 4, 10	Constant 50, 500 req/min	Variable 50, 100, 150, 200, 250 KB	Config-1, 2, 3, 4	Throughput

Experiment-5 (evaluating bandwidth-usage, variable-request-rate, constant-message-size)

The main goal of this experiment is to evaluate the bandwidth usage by measuring the number of exchanged messages among basic fragments executed by basic agents in WADE/JADE environment with variable-request-rate and constant-message-size. The experiment is configured according to Table 4.5.

Table 4.5: Experiment-5 Configurations

Number of Machines	Number of Iterations (n)	Request Rate	Messages Size	Configuration Type	Evaluated Parameter
1,2	1, 4, 10	Variable 1-600 req/sec	Constant ~ zero	Config-1, 2, 3, 4	Bandwidth-usage

4.8.2 Frequent-path and Proportional-fragment Experiments

The main goal of the frequent-path and proportional-fragment experiments is to study the behavior these two aspects of adaptability in business process decentralization. In this section, the evaluation method of HPD and HIPD decentralization approaches, experimental setup, evaluation metrics and methods are presented. The evaluation of the frequent-path and proportional-fragment adaptability are performed through measuring response-time, throughput and bandwidth usage of fragments. In business process decentralization, process activities and their relations are under focus; therefore, no extra code is considered inside process activities. Furthermore, the time cost of calling external web services is considered equal to zero due to the same reason. The fragments created by both HPD and HIPD methods are then compared by the Centralized and FPD methods.

Evaluation Metrics

Three metrics are used to evaluate decentralization methods including, response-time, throughput, and number of exchanged messages.

Evaluation Method

The main steps of all experiments are shown in Figure 4.6; however, each experiment considers different goals. A business process specified by the BPEL language is considered for decentralization. There are two decentralization methods including HPD and HIPD represented in this thesis. Both of them decentralize block-structured business processes in terms of process tree levels; however, the HIPD recognizes frequent and infrequent paths of previously executed workflows prior to decentralizing new processes. The produced fragments based on the type of

architecture are encapsulated in agents and then they are deployed into runtime environment.

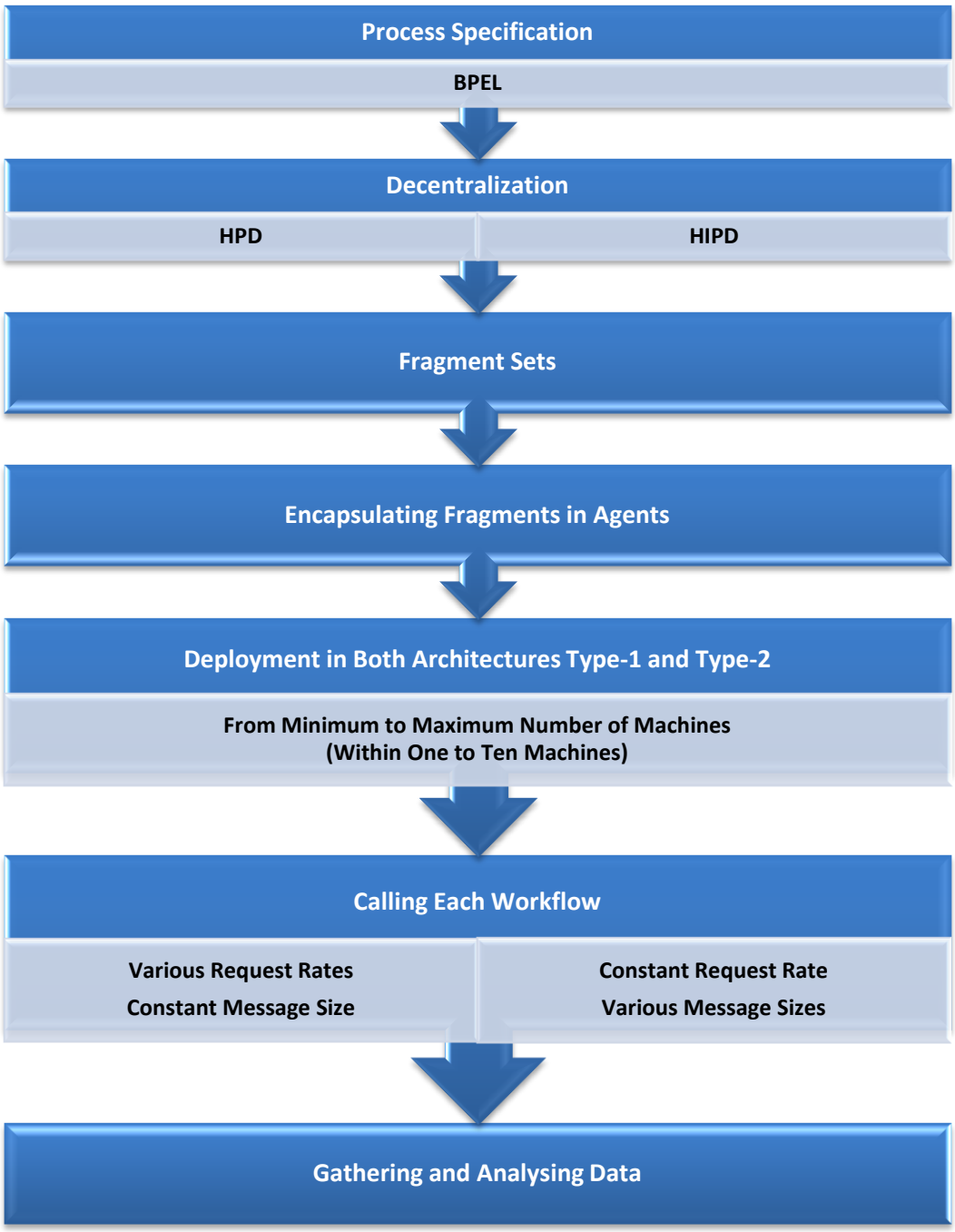


Figure 4.6: Evaluation Method of Frequent-path and Proportional-fragment Adaptability

The experiments are performed on *minimum and maximum (Min-Max) distributions of* fragments. The *minimum distribution* applies one machine and in the architecture Type-1, it is equal to applying only one performer agent to execute process fragment

while in the architecture Type-2, it is analogous to running a dedicated performer agent for each process fragment. The *maximum distribution* for the architecture Type-1 means to allocate exactly one machine to each performer agent so that they are able to run different process fragments at the same time. For the architecture Type-2, it means allocating exactly one performer agent for each process fragment and each performer agent is executed by a dedicated machine.

The response-time and throughput are affected by the request rate and size of messages passed through process activities. Thus, two categories of experiments are designed. The first category considers variable-request-rate and constant-message-size to omit the effect of message size. The second category considers constant-request-rate in presence of message size with varied values 50KB, 100KB, 150KB, 200KB and 250KB.

All experiments are targeted at measuring response-time, throughput, and number of exchanged messages. Experiment-6 and Experiment-7 examine Hypothesis-2 and evaluate response-time and throughput with variable-request-rate, constant-message-size. Experiment-8 and Experiment-9 examine Hypothesis-2 and evaluate response-time and throughput with constant-request-rate, variable-message-size. In addition, Experiment-10 also examines Hypothesis-2 and evaluates the effect of request rate on bandwidth-usage of fragments.

Experiment-6 (frequent-path adaptability, response-time evaluation, variable-request-rate/constant-message-size)

The main goal of this experiment is to substantiate Hypothesis-2 by evaluating response-time with variable-request-rate and constant-message-size. The experiment is configured according to Table 4.6.

Table 4.6: Experiment-6 Configurations

Architecture Type	Distribution	Number of Machines				Request Rate	Messages Size	Evaluated Parameter
		If -process	While-process	Flow-process	Loan -process			
Type-1	Min	1	1	1	1	Variable 1-600 req/sec	Constant ~ zero	Response-time
	Max	6	5	6	16	Variable 1-600 req/sec	Constant ~ zero	Response-time
Type-2	Min	1	1	1	1	Variable 1-600 req/sec	Constant ~ zero	Response-time
	Max	6	5	6	16	Variable 1-600 req/sec	Constant ~ zero	Response-time

Experiment-7 (frequent-path adaptability, throughput evaluation, variable-request-rate/constant-message-size)

The main goal of this experiment is to substantiate Hypothesis-2 by evaluating throughput with variable-request-rate and constant-message-size. The experiment is configured according to Table 4.7.

Table 4.7: Experiment-7 Configurations

Architecture Type	Distribution	Number of Machines				Request Rate	Messages Size	Evaluated Parameter
		If -process	While-process	Flow-process	Loan -process			
Type-1	Min	1	1	1	1	Constant 1-600 req/sec	Variable ~zero	Throughput
	Max	6	5	6	16	Constant 1-600 req/sec	Variable ~zero	Throughput
Type-2	Min	1	1	1	1	Constant 1-600 req/sec	Variable ~zero	Throughput
	Max	6	5	6	16	Constant 1-600 req/sec	Variable ~zero	Throughput

Experiment-8 (frequent-path adaptability, response-time evaluation, constant-request-rate/variable-message-size)

The main goal of this experiment is to substantiate Hypothesis-2 by evaluating response-time with constant-request-rate and variable-message-size. The experiment is configured according to Table 4.8.

Table 4.8: Experiment-8 Configurations

Architecture Type	Distribution	Number of Machines				Request Rate	Messages Size	Evaluated Parameter
		If -process	While-process	Flow-process	Loan -process			
Type-1	Min	1	1	1	1	Constant 50, 500 req/min	Variable 50, 100, 150, 200, 250 KB	Response-time
	Max	6	5	6	16	Constant 50, 500 req/min	Variable 50, 100, 150, 200, 250 KB	Response-time
Type-2	Min	1	1	1	1	Constant 50, 500 req/min	Variable 50, 100, 150, 200, 250 KB	Response-time
	Max	6	5	6	16	Constant 50, 500 req/min	Variable 50, 100, 150, 200, 250 KB	Response-time

Table 4.9: Experiment-9 Configurations

Architecture Type	Distribution	Number of Machines				Request Rate	Messages Size	Evaluated Parameter
		If -process	While-process	Flow-process	Loan -process			
Type-1	Min	1	1	1	1	Constant 50, 500 req/min	Variable 50, 100, 150, 200, 250 KB	Throughput
	Max	6	5	6	16	Constant 50, 500 req/min	Variable 50, 100, 150, 200, 250 KB	Throughput
Type-2	Min	1	1	1	1	Constant 50, 500 req/min	Variable 50, 100, 150, 200, 250 KB	Throughput
	Max	6	5	6	16	Constant 50, 500 req/min	Variable 50, 100, 150, 200, 250 KB	Throughput

Experiment-9 (frequent-path adaptability, evaluating throughput, constant-request-rate, variable-message-size)

The main goal of this experiment is to substantiate Hypothesis-2 by evaluating throughput with constant-request-rate and variable-message-size. The experiment is configured according to Table 4.9.

Experiment-10 (frequent-path adaptability, evaluating bandwidth-usage, variable-request-rate/constant-message-size)

The main goal of this experiment is to substantiate the frequent-path adaptability in Hypothesis-2 by evaluating bandwidth-usage with variable-request-rate/constant-message-size. The experiment is configured according to Table 4.10. Different decentralization methods produce various numbers of fragments.

Table 4.10: Experiment-10 Configurations

Number of Machines	Request Rate	Messages Size	Evaluated Parameter
N/A	Variable 1-600 req/sec	Constant ~zero	Bandwidth-usage

Experiment-11 (proportional-fragment adaptability, response-time and throughput evaluation, variable-request-rate/constant-message-size)

The main goal of this experiment is to substantiate the proportional-fragment adaptability in Hypothesis-2 by evaluating response-time and throughput of process fragments with variable-request-rate/constant-message-size. The experiment is configured according to Table 4.11.

Table 4.11: Experiment-11 Configurations

	Number 0f Machines	**Request Rate**	**Messages Size**	**Evaluated Parameters**
Type-1	**Within 1-10 Machines for Loan-process**	Variable 1-600 req/sec	Constant ~zero	Response-time, Throughput
Type-2		Variable 1-600 req/sec	Constant ~zero	Response-time, Throughput

Experiment-12 (proportional-fragment adaptability, response-time evaluation, constant-request-rate/variable-message-size)

The main goal of this experiment is to substantiate the proportional-fragment adaptability in Hypothesis-2 by evaluating response-time with constant-request-rate/variable-message-size. This experiment figures out what number of fragments must be provided in presence of different number of machines. The experiment is configured according to Table 4.12.

Table 4.12: Experiment-12 Configurations

	Number of Machines	Request Rate	Messages Size	Evaluated Parameter
Type-1	Within 1-10 Machines for Loan-process	Constant 50, 500 req/min	Variable 50, 100, 150, 200, 250 KB	Response-time
		Constant 50, 500 req/min	Variable 50, 100, 150, 200, 250 KB	Response-time
Type-2		Constant 50, 500 req/min	Variable 50, 100, 150, 200, 250 KB	Response-time
		Constant 50, 500 req/min	Variable 50, 100, 150, 200, 250 KB	Response-time

Table 4.13: Experiment-13 Configurations

	Number of Machines	Request Rate (req/min)	Messages Size	Evaluated Parameter
Type-1	Within 1-10 Machines for Loan-process	Constant 50, 500 req/min	Variable 50, 100, 150, 200, 250 KB	Throughput
		Constant 50, 500 req/min	Variable 50, 100, 150, 200, 250 KB	Throughput
Type-2		Constant 50, 500 req/min	Variable 50, 100, 150, 200, 250 KB	Throughput
		Constant 50, 500 req/min	Variable 50, 100, 150, 200, 250 KB	Throughput

Experiment-13 (proportional-fragment adaptability, throughput evaluation, constant-request-rate/variable-message-size)

The main goal of this experiment is to substantiate the proportional-fragment adaptability in Hypothesis-2 by evaluating throughput with constant-request-rate/variable-message-size. This experiment figures out what number of fragments

must be provided in presence of different number of machines. The experiment is configured according to Table 4.13.

4.8.3 Fuzzy Decentralization Decision Making Unit Experiments

The main goal of the available-bandwidth experiments is to study the behavior of this adaptability aspect in balancing scalability and adaptability. In this section, the evaluation method of the available-bandwidth adaptability, experimental setup, evaluation metrics and methods are presented. The evaluation of the available-bandwidth adaptability is performed through measuring bandwidth-usage of a decentralized process which is done by counting number of exchanged messages among process fragments.

Workflow decentralization policies must not degrade system functionalities. It must be commensurate with runtime environment circumstances to avoid any malfunction. The Fuzzy Decentralization Decision Making unit (FDDM) introduced in this thesis applies the fuzzy logic for making decision on how to decentralize a business process. The unit receives the number of machines and available bandwidth from runtime environment and chooses a suitable decentralization method so that it satisfies the frequent-path, proportional-fragment and available-bandwidth adaptability aspects.

Evaluation Metrics

The number of exchanged messages is used to evaluate the available-bandwidth adaptability of the Fuzzy Decentralization Decision Making unit (FDDM) against the FPD method.

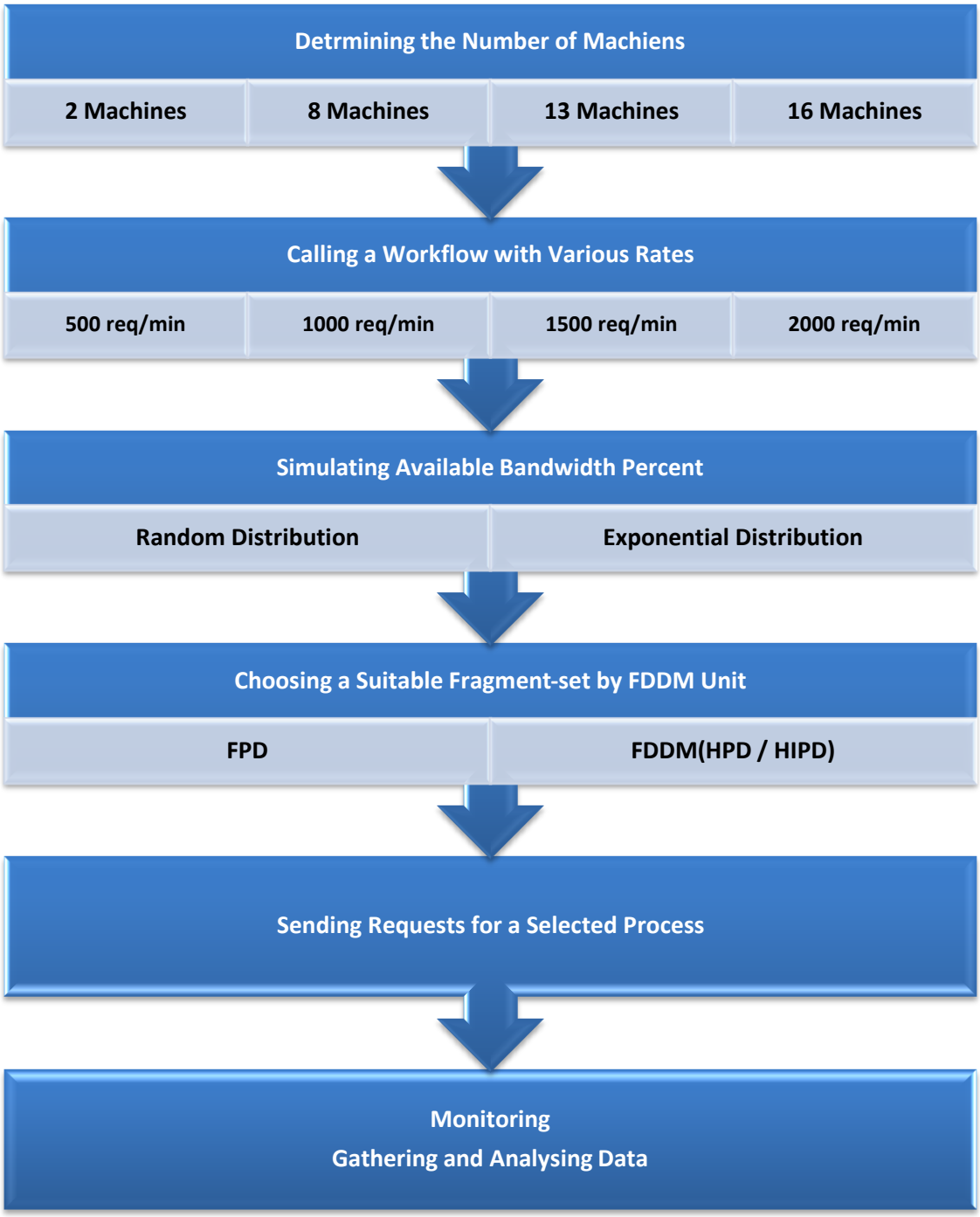


Figure 4.7: Evaluation Method of Available-bandwidth Adaptability

Evaluation Method

The main steps of experiments are specified in Figure 4.7; however, each experiment focuses on a different goal. This evaluation is independent of the type of the applied architecture in that the number of exchanged messages among the fragments is the same in both architecture Type-1 and Type-2.

The FDDM unit may be equipped with only proportional-fragment adaptability or both frequent-path adaptability and proportional-fragment adaptability aspects. Applying only the HPD approach in the FDDM unit means that the unit is equipped with the proportional-fragment adaptability aspect while using both HPD and HIPD methods is analogous to equipping the FDDM unit with both proportional-fragment and frequent-path adaptability aspects.

Each experiment is performed in two sections (A) FDDM behavior study, and (B) available-bandwidth adaptability. The section A studies the behavior of the FDDM unit in recommending a suitable level of granularity (fragment-set) in a business process tree. The experiment is repeated for two, eight, thirteen, and sixteen machines. A client sends requests with constant-request-rate of 500 requests per minute. The available bandwidth is simulated randomly/exponentially (between 0-100%). The FDDM unit is equipped with either HPD or HPD/HIPD decentralization methods. During the experiment, the Java Sniffer software records messages exchanged among the agents. In the section B, the available-bandwidth adaptability is evaluated. Indeed, the experiment is repeated for two, eight, thirteen, and sixteen machines. A client sends requests with specific rates of 500, 1000, 1500, and 2000 requests per minute. On producing a client request, a bandwidth number (between 0-100%) is randomly/exponentially generated, which simulates the percentage of available bandwidth. The available bandwidth and number of machines are sent to the FDDM unit, which recommends a suitable fragment-set. The FDDM unit is equipped with either HPD or HPD/HIPD decentralization methods. The same experiments are repeated without presence of the FDDM unit, which is analogous to applying the FPD method. During the experiment, the Java Sniffer software records messages exchanged among the agents. Experiments 14-17 examine Hypothesis-3

and evaluate the bandwidth-usage of the FDDM in comparison to the FDDM method.

Experiment-14 (evaluating available-bandwidth adaptability, HPD decentralization, random bandwidth)

The main goal of this experiment is to substantiate Hypothesis-3 by evaluating bandwidth-usage of the FDDM unit applying the HPD decentralization method and a random simulation of bandwidth. The experiment is conducted in two sections A and B which are configured according to Table 4.14.

Table 4.14: Experiment-14 Configurations

Section	Number of Machines	Request Rate	Available Bandwidth Simulation	Decentralization Method	Evaluated Parameter
A	2, 8, 13, 16	Constant 500 req/min	Random	HPD	FDDM Recommended Granularity Level
B	2, 8, 13, 16	Constant 500, 1000,1500, 2000 req/min	Random	HPD	Bandwidth-usage

Experiment-15 (evaluating available-bandwidth adaptability, HPD decentralization, exponential bandwidth)

The main goal of this experiment is to substantiate Hypothesis-3 by evaluating bandwidth-usage of the FDDM unit applying the HPD decentralization method and an exponential simulation of bandwidth. The experiment is conducted in two sections A and B which are configured according to Table 4.15.

Table 4.15: Experiment-15 Configurations

Section	Number of Machines	Request Rate	Available Bandwidth Simulation	Decentralization Method	Evaluated Parameter
A	2, 8, 13, 16	Constant 500 req/min	Exponential	HPD	FDDM Recommended Granularity Level
B	2, 8, 13, 16	Constant 500, 1000,1500, 2000 req/min	Exponential	HPD	Bandwidth-usage

Experiment-16 (evaluating available-bandwidth, HPD/HIPD decentralization, random bandwidth)

The main goal of this experiment is to substantiate Hypothesis-3 by evaluating bandwidth-usage of the FDDM unit applying both HPD and HIPD decentralization methods and a random simulation of bandwidth. The experiment is conducted in two sections A and B which are configured according to Table 4.16.

Table 4.16: Experiment-16 Configurations

Section	Number of Machines	Request Rate	Available Bandwidth Simulation	Decentralization Method	Evaluated Parameter
A	2, 8, 13, 16	Constant 500 req/min	Random	HPD/HIPD	FDDM Recommended Granularity Level
B	2, 8, 13, 16	Constant 500, 1000,1500, 2000 req/min	Random	HPD/HIPD	Bandwidth-usage

Experiment-17 (evaluating available-bandwidth adaptability, HPD/HIPD decentralization, exponential bandwidth)

The main goal of this experiment is to substantiate Hypothesis-3 by evaluating bandwidth-usage of the FDDM unit applying both HPD and HIPD decentralization methods and an exponential simulation of bandwidth. The experiment is conducted in two sections A and B which are configured according to Table 4.17.

Table 4.17: Experiment-17 Configurations

Section	Number of Machines	Request Rate	Available Bandwidth Simulation	Decentralization Method	Evaluated Parameter
A	2, 8, 13, 16	Constant 500 req/min	Exponential	HPD/HIPD	FDDM Recommended Granularity Level
B	2, 8, 13, 16	Constant 500, 1000,1500, 2000 req/min	Exponential	HPD/HIPD	Bandwidth-usage

4.9 Summary

This chapter overviewed the adopted research of this thesis, articulated the research problem and presented the steps taken to achieve the research objectives. In order to evaluate and verify the proposed model, reference business processes as well as

experiment platform and set up were introduced. In addition, numerous experiments were established and the incorporated metrics and methods to implement each experiment were encompassed.

CHAPTER 5

ADAPTABLE DECENTRALIZED WORKFLOW EXECUTION FRAMEWORK (ADWEF)

5.1 Introduction

This chapter aims at proposing a model for dynamic business processes decentralization based on runtime circumstances. Accordingly, an Adaptable Decentralized Workflow Execution Framework (*ADWEF*) is presented in Section 5.2 followed by the definition of an adaptable orchestration layer in the service oriented stack in Section 5.3. Later, two architectures are introduced to support the dynamicity of fragmentation in Section 5.4. In Section 5.5, two decentralization methods are then introduced, which are dynamic enough to satisfy adaptability aspects of decentralization. An algorithm for wiring of fragments is represented as well. Section 5.10 presents an adaptable decentralization decision making unit, which applies the fuzzy logic to choose a right method of decentralization based on runtime circumstances. Finally this chapter ends with miscellaneous aspects of the ADWEF in Section 5.11 and a summary in Section 5.12.

5.2 Adaptable Decentralized Workflow Execution Framework

A runtime environment is subject of many changes during the execution of software applications. The adaptability of software with requirements of its execution environment is important in that it helps the runtime environment to refrain from catastrophic events, which is the result of violating system thresholds. Distributed systems are always used to resolve the scalability issues of the centralized systems. However, distribution may result in performance bottlenecks due to communication among system components and unexpected changes in a distributed environment.

Adaptability of a distributed system with its environment is of high importance to refrain from approaching system thresholds and bypassing bottlenecks.

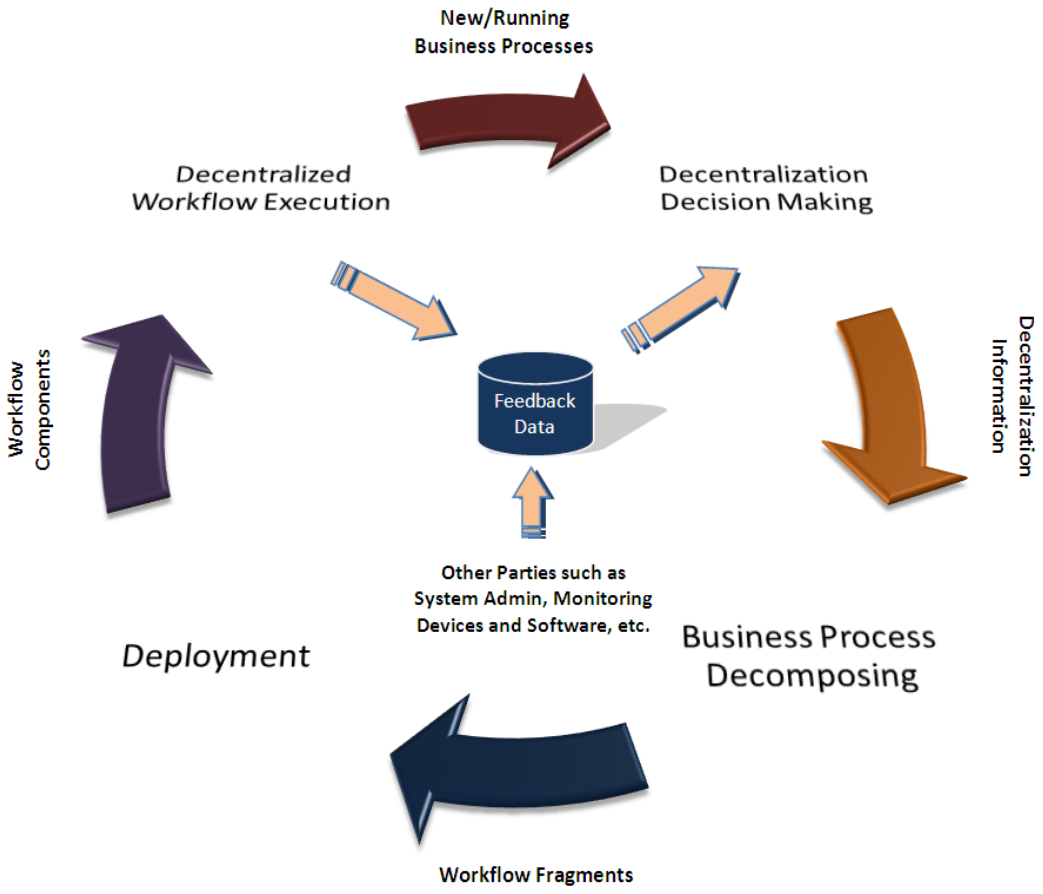


Figure 5.1: Adaptable Decentralized Workflow Execution Framework (ADWEF)

Figure 5.1 shows the main phases of an *Adaptable Decentralized Workflow Execution Framework (ADWEF)* to support adaptable decentralization of business processes. The central part of the figure is a feedback data repository, which can be initialized by different parties such as system administrator, distributed workflow engine, configuration files, monitoring software and devices, etc. Based on the data provided in the feedback repository a decentralization decision maker may be able to decentralize/re-decentralize a new/running business process. Re-decentralization may occur due to violating system thresholds. Making decision on how to decentralize a business process, the decision maker submits required information to the business

process decomposer, which is able to fragment a business process to workflow fragments. Through a process of deployment, the produced fragments are encapsulated into runtime components (i.e. agents) and will be deployed into the machines dedicated to a workflow engine.

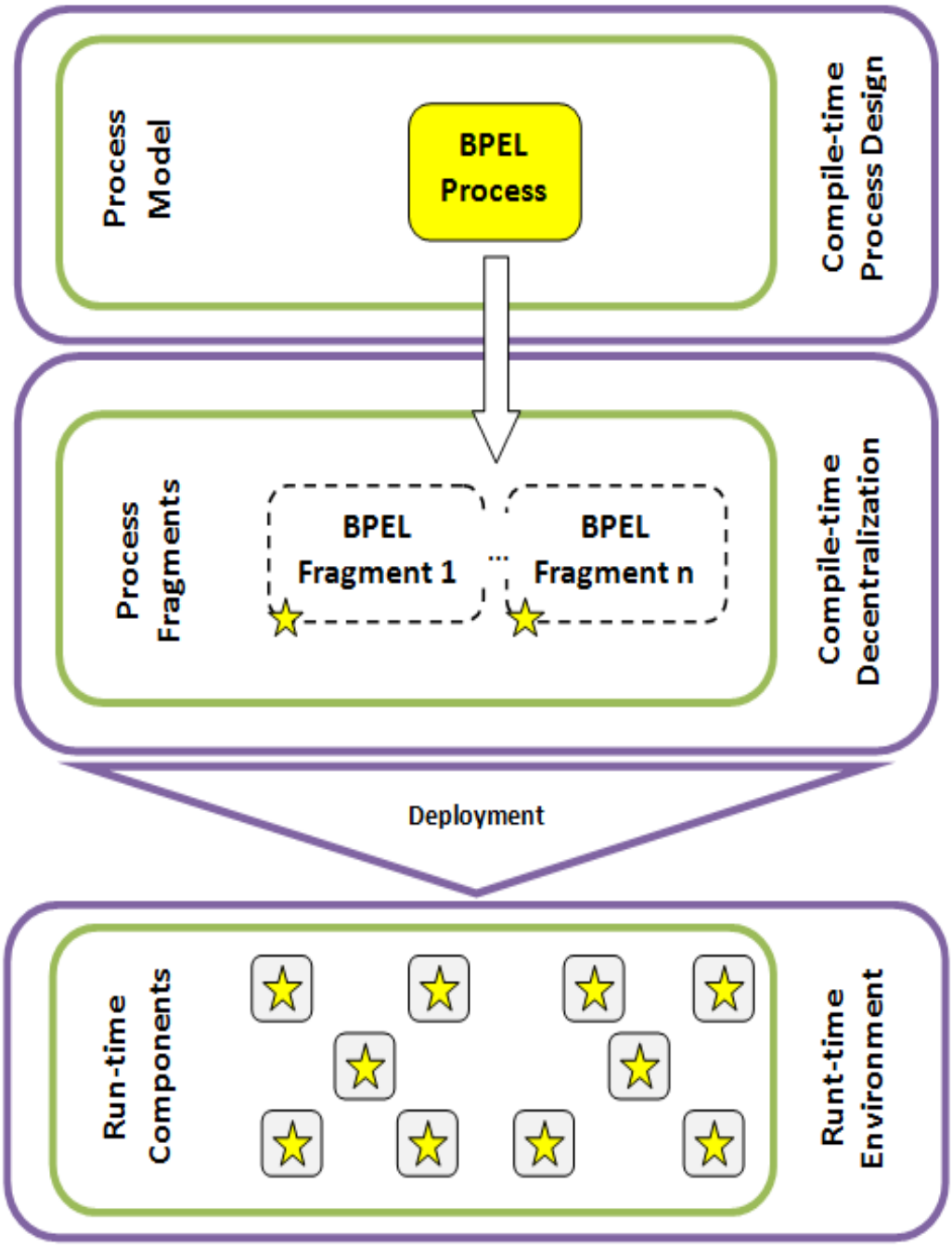


Figure 5.2: Traditional Decentralization

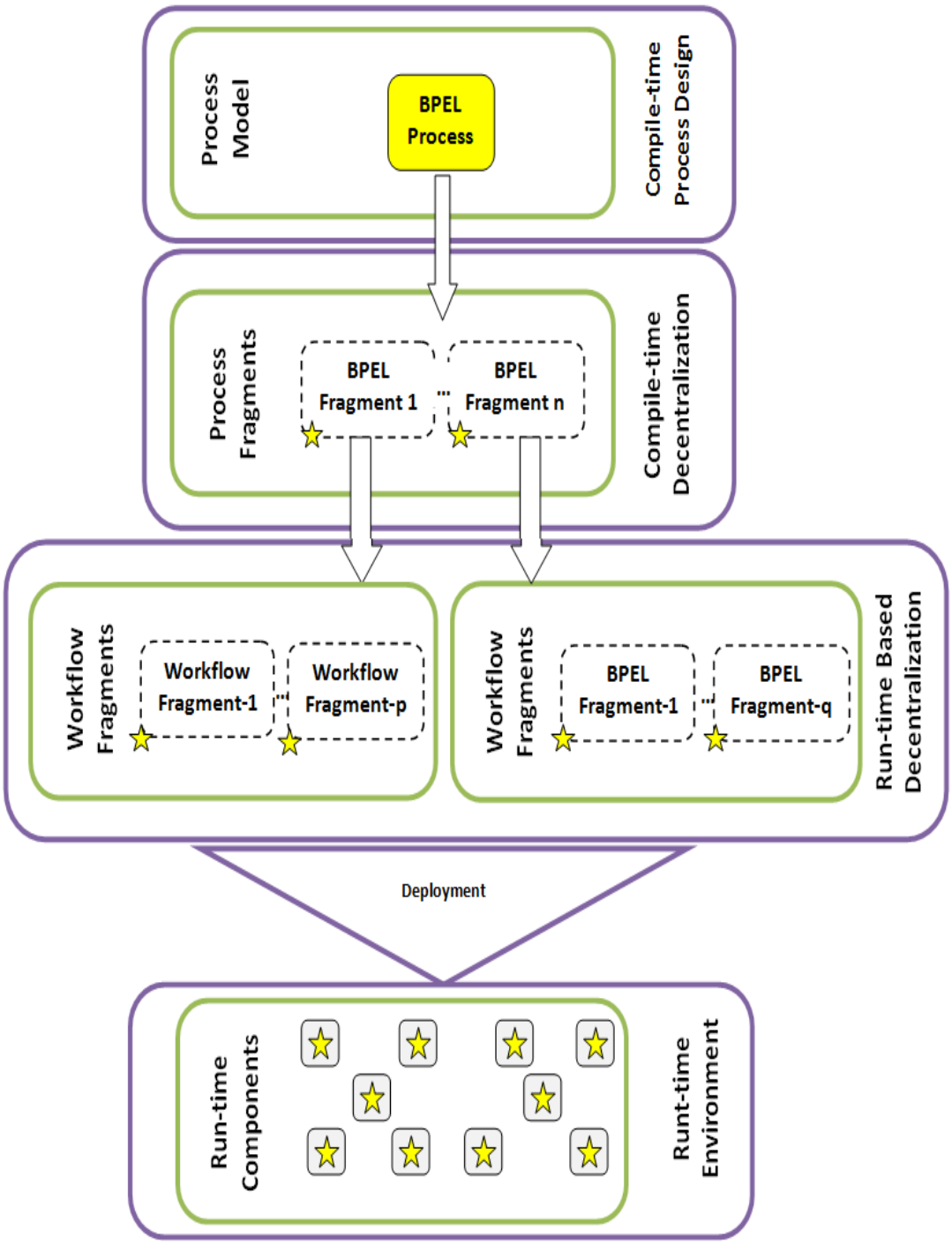


Figure 5.3: Runtime Based Decentralization

5.3 Adaptable SOA Orchestration Layer

According to the literature, business processes are mostly decentralized into several fragments, called *process fragments*. The decentralization is performed based on a number of criteria that are considered by business process designers. After

fragmenting a business process, the fragments are encapsulated in runtime components such as web services, agents, etc. as shown in Figure 5.2.

Generally, business process designers do not have any prior knowledge on runtime circumstances and consequently they are not able to make right decentralization decisions at design time. As runtime circumstances change, a workflow engine has to adapt itself with this ever-changing environment and current decentralization methods suffer from a lack of runtime based workflow decentralization. Indeed, design-time and runtime process decentralization have to be isolated to extenuate designers' role in compile-time. It naturally makes decentralization of a business process more convenient.

On the other hand, considering the open-world software paradigm (Baresi, Nitto et al., 2006) and the concept of self-adaptability in the service oriented architecture introduced by (Dustdar & Papazoglou, 2008; Papazoglou et al., 2007, 2008), the procedure of business process decentralization may also incorporate the idea of self-

adaptability with runtime environment. The process of decentralization may be configured so that the produced fragments become adaptable with different aspects of runtime environment. For example, when a number of business process activities are closely interrelated, they can be fragmented together, or fragments can be provided based on the number of available machines allocated for a workflow engine. In case of one machine, creating one fragment suffices instead of producing a number of fragments on the same machine while more number of fragments can be generated in case of more machines.

Fragments might be shrunk or expanded, which can be imagined as fusion/fission of workflow fragments based on environment conditions to realize adaptability. In other

words, different running versions of a business process can be conceived of its avatars that are able to be transformed to each other. For instance while a version of a process workflow lives in a single machine, another version of the process may be fragmented and scattered on a number of nodes in a network. They may toggle their shape from the scattered to the centralized version and vice versa based on environment circumstances.

Figure 5.3 shows the recommended runtime based decentralization in which each *process fragment* is converted to several workflow fragments based on runtime circumstances. *Workflow fragment* is different from *process fragment* in that the former is created based on runtime environment feedbacks while the latter is decentralized based on designers' opinion. Workflow fragments will eventually be encapsulated in runtime components such as web services, agents, etc.

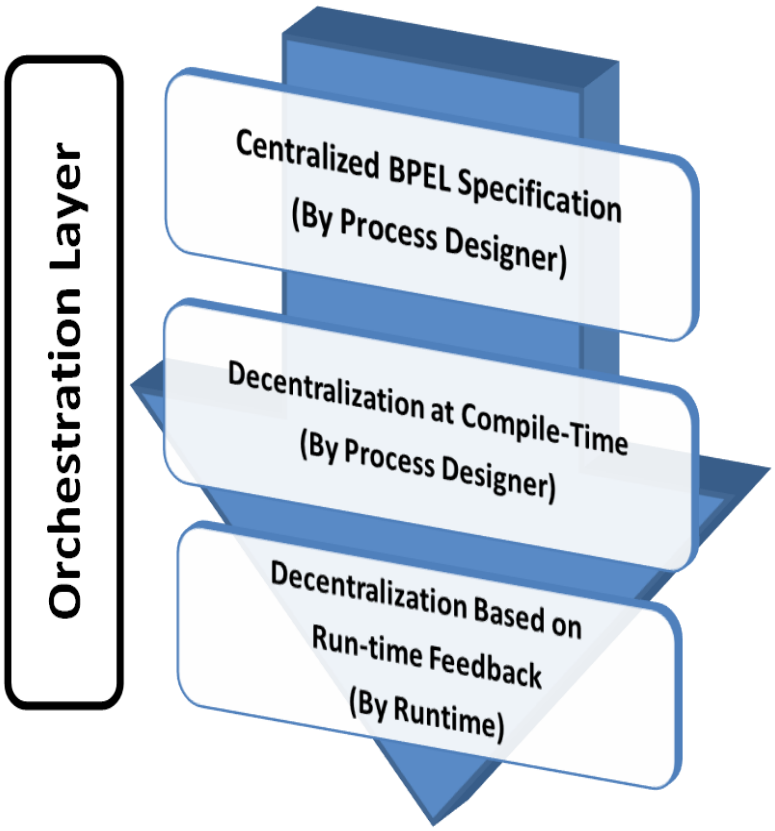


Figure 5.4: Recommended Abstraction for SOA Orchestration Layer

The independency of compile-time and runtime decentralization can be realized through a three-layered architecture recommended for the orchestration layer of service oriented architecture as shown in Figure 5.4. The top layer is the specification of BPEL processes, which is specified by process designers. The middle layer is a compile-time fragmentation, which is done by process designers or tools. The last layer, which is the originality of this study, is introduced as an abstract layer based on which compile-time and runtime decentralization of workflows become independent. This layer can also be equipped with a number of intelligent/unintelligent techniques to make more effective decentralization decisions based on current circumstances of runtime environment, which is analogous to decentralization adaptability.

5.4 ADWEF Architectures

This section presents two architectures to support dynamic fragment execution. At first, an execution environment is formally described and then two architectural types namely Type-1 and Type-2 are encompassed.

5.4.1 ADWEF Architecture Formalization

The *ADWEF* architecture can be formalized as shown in definition (5.1). In this framework, $P = \{p_1, \dots, p_i, \dots, p_m\}$ is a set of business processes $p_i \in P$; $F = \{f_1, \dots, f_i, \dots, f_m\}$ is also a set of *fragment sets* $f_i \in F$ made from processes $p_i \in P$, so that each fragment set $f_i = \{pf_1, \dots, pf_i, \dots, pf_n\}$ contains a set of process fragments $pf_i \in f_i$.

$$ADWEF = (P, F, WFC, M, \Phi, \Sigma, \Delta) \qquad (5.1)$$

For instance each business process can be encapsulated in one fragment as a Centralized process that means $|f_{Centralized}| = 1$. Alternatively, a process with *k* activities can be fully decentralized and provide *k* fragments; therefore, $|f_{FPD}| = k$.

Workflow components (called performer or micro-workflow engine) are small size workflow engines, which are able to execute *process fragments* called *workflow fragments* at runtime. *WFC* is also a set of *workflow components* to run workflow fragments. *M* is a set of machines dedicated to an orchestration engine as well. The workflow components are able to communicate with each other through a middleware. However, distinctions among workflow execution platforms are determined by Φ, Σ and Δ sets. Set $\Phi = \{\varphi_{HPD}, \varphi_{HIPD}, \varphi_{Centralized}, \varphi_{FPD}, \varphi_{IPD}\}$ is a set of fragment functions φ_i, which convert a process to a set of fragments based on a specific policy and is defined as $\varphi_i: P \rightarrow F$. Functions φ_{HPD} and φ_{HIPD} presented in the following sections, fragment business processes applying a hierarchical and a mining approach, respectively. Two functions $\varphi_{Centralized}$ and φ_{FPD} are considered to model the previous works; the former translates the whole business process to one fragment and the latter translates it to a set of fragments so that each fragment includes just one process activity. Function φ_{IPD}, which is discussed more in the following sections, is actually the intersection of both block and graph-structured business processes from mining based decentralization view.

Set Σ includes encapsulate functions, which assign a workflow fragment to a workflow component. In fact, a workflow component can be an agent, a web service, etc., which is intrinsically implemented as a thread in runtime environment. Similarly, workflow fragments are implemented as runtime threads in themselves, which are assigned to runtime components. Consequently, creating workflow

components and encapsulating workflow fragments in them results in creating several threads in runtime environment.

```
name: ADWEF_Setup
input: P, F, WFC, M, Φ, Σ, Δ
output:-;
begin
  | 0 ≤ |p|; ∀ wfc_i ∈ WFC; wfc_i ← wfc_i ∪ createWorkflowComponent();
  | ∃p_i ∈ P; ∃f_i ∈ F; f_i ← f_i ∪ Φ(p_i);
  | ∀wf_i ∈ f_i; ∀ wfc_j ∈ WFC; wfc_i ← encapsulate(wf_i); |encapsulate ∈ Σ;
  | deploy(WFC, M); |deploy ∈ Δ;
end;
```

Figure 5.5: ADWEF Architecture Setup

Set Σ includes encapsulate functions, which assign a workflow fragment to a workflow component. In fact, a workflow component (also called performer or micro-workflow engine) can be an agent, a web service, etc., which is intrinsically implemented as a thread in runtime environment. Similarly, workflow fragments are also runtime threads in themselves, which are assigned to runtime components. Consequently, creating workflow components and encapsulating workflow fragments in them result in creating a number of threads in runtime environment.

Moreover, assigning the fragments to workflow components is based on the type of selected architecture. In addition, Δ is a deployment/distributor function, which maps workflow components to network machines and is defined as $\Delta: WFC \rightarrow M$, which is also dependent on the type of architecture. The implementation of Δ methods is a subject of load balancing as well.

Figure 5.5 shows the setup instruction of the ADWEF architecture using its basic elements. Accordingly, ADWEF has to create at least one workflow component to run workflow fragments dependent on the type of architecture. In addition, each process p_i has to be fragmented and the fragments are stored in the respective

fragment set f_i. Workflow fragments must be encapsulated in workflow components applying different policies. Furthermore, various deployment policies can be applied to send all $wfc_i \in WFC$ to workflow engine machines M.

5.4.2 ADWEF Architecture Types

Two ADWEF architectures are presented in Figure 5.6 and Figure 5.8 called Type-1 and Type-2, which support dynamic decentralization and execution of dynamic fragments. However, several aspects of workflow management systems are not included in this research such as governance of workflow fragments, inter-organizational workflows, managing runtime state of workflow fragments, sharing interior/exterior variables among workflow fragments, runtime workflow reconfiguration, transactions and exception handling. These are normally either implemented by a distributed middleware or may demand more attention from dynamic process decentralization view in future work.

Both architectures include a ***runtime environment*** containing a set of ***machines*** each of which is responsible for running ***workflow components***. Workflow components are containers in which ***workflow fragments*** are executed and can be realized by agents, web services, etc. The activity of workflows is logged by the runtime environment and all the log data are stored in a ***log repository***. The log repository can intrinsically be a shared file located on a rapid disk; although, a dozen of mining methods discuss on mining distributed log files (Bhaduri et al., 2011). ***Dynamic process decentralization component*** **(*DPDC*)** is a software component and decentralizes business processes based on feedback data from runtime environment and/or stakeholders. After making decision on the type of decentralization, fragments are built. Workflow fragments might be made on-the-fly at runtime or can be taken

from a ***fragment repository***, which contains pre-compiled workflow fragments. As soon as fragments become available, they are encapsulated into workflow components and deployed into the runtime environment. The whole components in both architectures are able to communicate through a ***middleware*** as well.

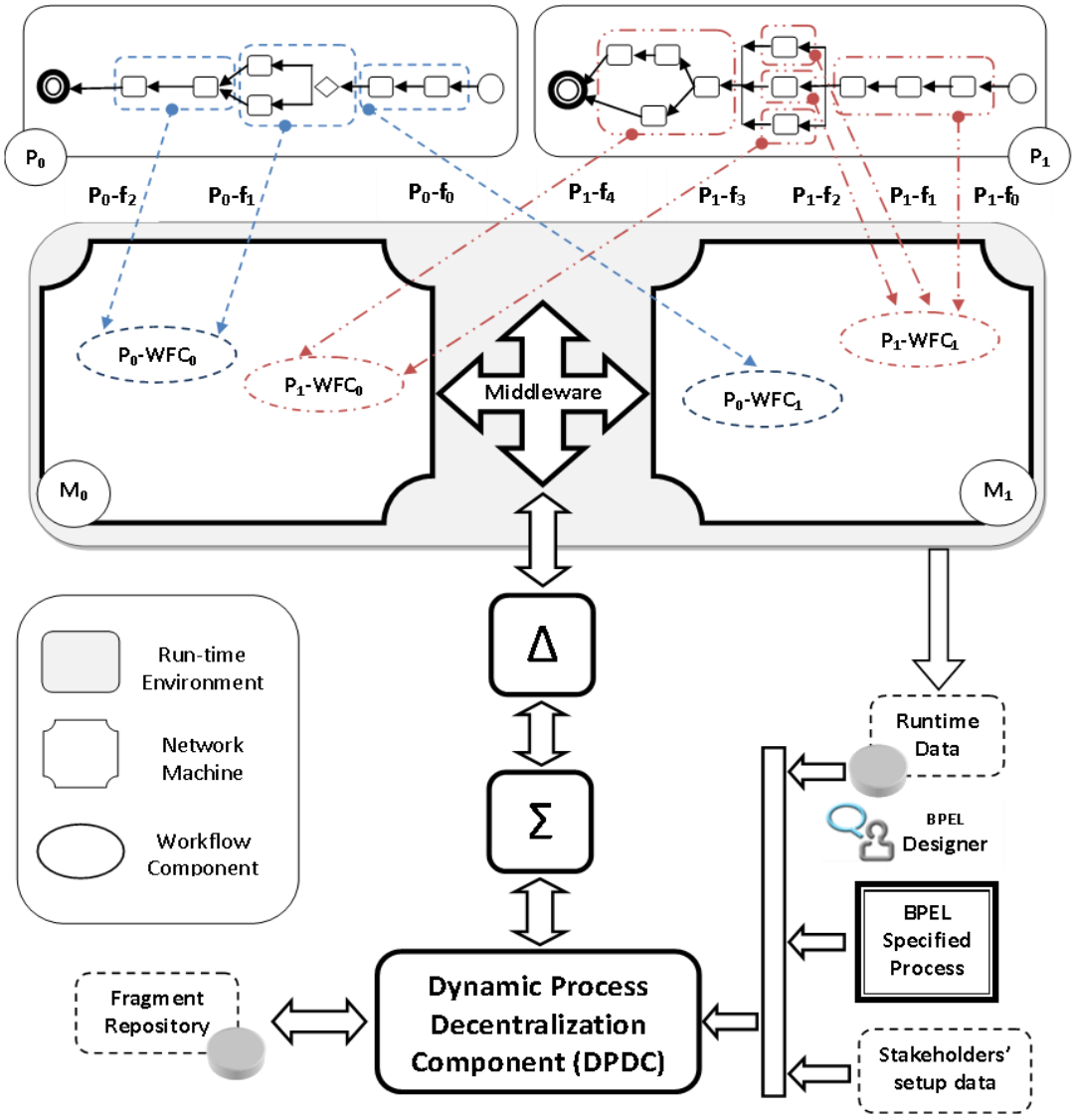


Figure 5.6: Architecture Type-1

1:	For each machine create a performer agent with a unique name;
2:	Compile each workflow fragment to its equivalent subflow;
3:	∑: each subflow must be assigned to a performer agent (Round-Robin Policy);
4:	Δ: distribute the performer agents on network machines (for each machine a performer agent);
5:	Setup input parameters of the subflow;
6:	Invoke the subflow;

Figure 5.7: Implementing Architecture Type-1 Using WADE/JADE Facilities

The architecture Type-1 aims at providing a general container for each workflow running on each machine. For instance as shown in Figure 5.6, workflow components $P_0 - WFC_0$ and $P_0 - WFC_1$ created on both machines M_0 and M_1 for the process P_0. Encapsulation functions Σ, are able to assign fragments to workflow components applying random, round-robin, etc. policies and in the case of this thesis round-robin policy is applied. In case of P_0, the fragments created for process P_0 including $P_0 - f_0$, $P_0 - f_1$ and $P_0 - f_2$ are encapsulated into the dedicated workflow components located on each machine.

The Deployment functions Δ are quite simple in the architecture Type-1, due to the existence of a default workflow component for each process on each machine. In this architecture, the management of workflow components would be more convenient, because they are able to execute all workflow fragments of the process at runtime. On the other hand, implementing such a general workflow component might consume a bulk of system resources. From runtime view, each workflow component is a single thread and each workflow fragment runs as a new thread as well. Figure 5.7 shows the algorithm for implementing architecture Type-1.

The architecture Type-2 as illustrated in Figure 5.8, allocates a dedicated workflow component to each fragment. The implementation of Σ functions is different from Type-1 in that each fragment is encapsulated in its own workflow component. For example process P_0 has three fragments; therefore, three workflow components $P_0 - WFC_0$, $P_0 - WFC_1$ and $P_0 - WFC_2$ are created for the fragments $P_0 - F_0$, $P_0 - F_1$, $P_0 - F_2$. Deployment function Δ determines, which workflow component must be sent to which machine. It can be based on load balancing; although, some simple distributions policies such as random, round-robin, etc. are also applicable

and in the case of this thesis a random distribution is applied. In this architecture, workflow components are simpler compared to a general workflow component in Type-1, because each workflow component executes just one workflow fragment. It consequently results in consuming fewer system resources. Normally, for each workflow component a new thread and for each fragment running on each workflow component an extra thread is created. Due to the fact that dynamic fragmentation creates a number of fragments from one business process; therefore, a number of threads will be created, which results in creating a number of workflow components. Figure 5.9 shows the algorithm for implementation of the architecture Type-2.

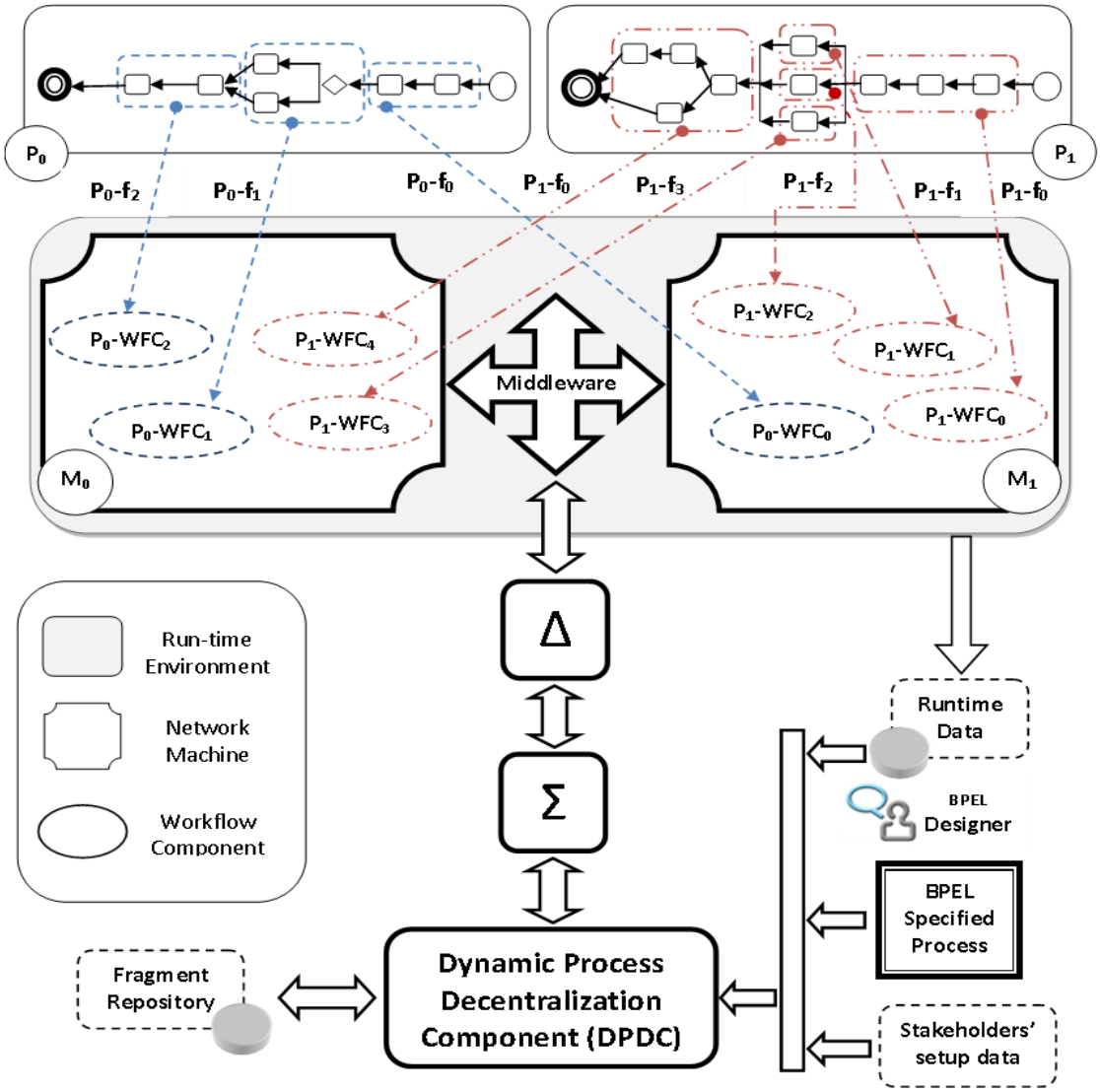


Figure 5.8: Architecture Type-2

```
1:  For each workflow fragment create a new performer agent with a unique name;
2:  Compile each workflow fragment to its equivalent subflow;
3:  ∑: each subflow must be assigned to the dedicated performer agent created in Step1;
4:  Δ: distribute the performer agents on network machines (Random Policy);
5:  Setup input parameters of the subflow;
6:  Invoke the subflow;
```

Figure 5.9: Implementing Architecture Type-2 Using WADE/JADE Facilities

5.5 DPDC Implementation Policies

In this section, the implementation of Hierarchical Process Decentralization (φ_{HPD}) and Hierarchical Intelligent Process Decentralization (φ_{HIPD}) are introduced, which are applied in the DPDC component. The HPD has a hierarchical view to business process decentralization while the HIPD is an intelligent method based on a mining approach and along with HPD, it provides more fragments. According to the naming convention of the experiments in Section 4.6, number of created fragments by a decentralization method in each granularity is shown as follows: <*decentralization method name*><*granularity level*>'_'<*number of fragments*>. For instance, HPD1_4 means that with applying the HPD method in level one of the process tree, four fragments are created.

5.6 Hierarchical Process Decentralization (HPD)

In this section, the implementation of Hierarchical Process Decentralization (φ_{HPD}) is introduced, which are applied in the DPDC component. A block-structured BPEL is a hierarchy of structured and unstructured activities nested in logical blocks, which can be modeled as a tree (Ouyang et al., 2006). A BPEL tree can be decentralized into its activities based on the level of decentralization, which is called Hierarchical Process Decentralization (φ_{HPD}). Fragmentation algorithm for a HPD-based DPDC component can be started from any level in a workflow tree applying a Breadth First Search/Traverse algorithm (Knuth, 1997).

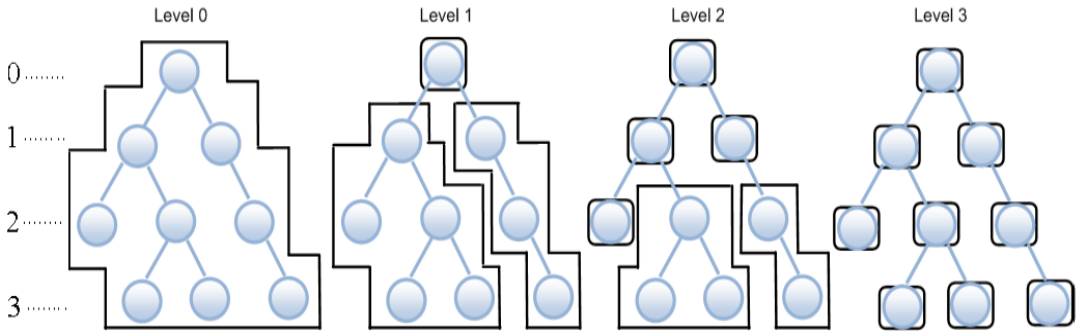


Figure 5.10: Hierarchical Process Decentralization for Blocked-structured Processes

Figure 5.10 shows the HPD decentralization of a process based on the levels of a sample BPEL tree. A fragmentation, which starts from *level-0* of the process tree, results in producing only one fragment that includes the whole tree. The fragmentation in *level-1* results in three fragments including root node, left sub-tree and right sub-tree, respectively. Similarly, the fragmentation that starts from *level-2* provides six fragments among which four fragments include one activity. Obviously, a *level-3* fragmentation provides nine fragments with the finest granularity while the *level-0* fragmentation provides the coarsest one. The coarser granularity fragments, the less number of fragments and the softest granularity fragments, the more number of fragments. The activities inside each fragment are logically related with each other according to the process specification.

In the HPD decentralization, leaves are simple activities while structured activities are in the middle and intrinsically keep the most relevant activities together in a single fragment. The coarsest fragment is equal to the traditional Centralized method and the softest granularity is equal to the FPD fragmentation.

Figure 5.11 illustrates the loan application process, which is decentralized with the HPD in four levels of granularity $0 \leq g < 4$. There exist four execution versions of the loan business process at runtime and in each version fragments are marked by an asterisk icon.

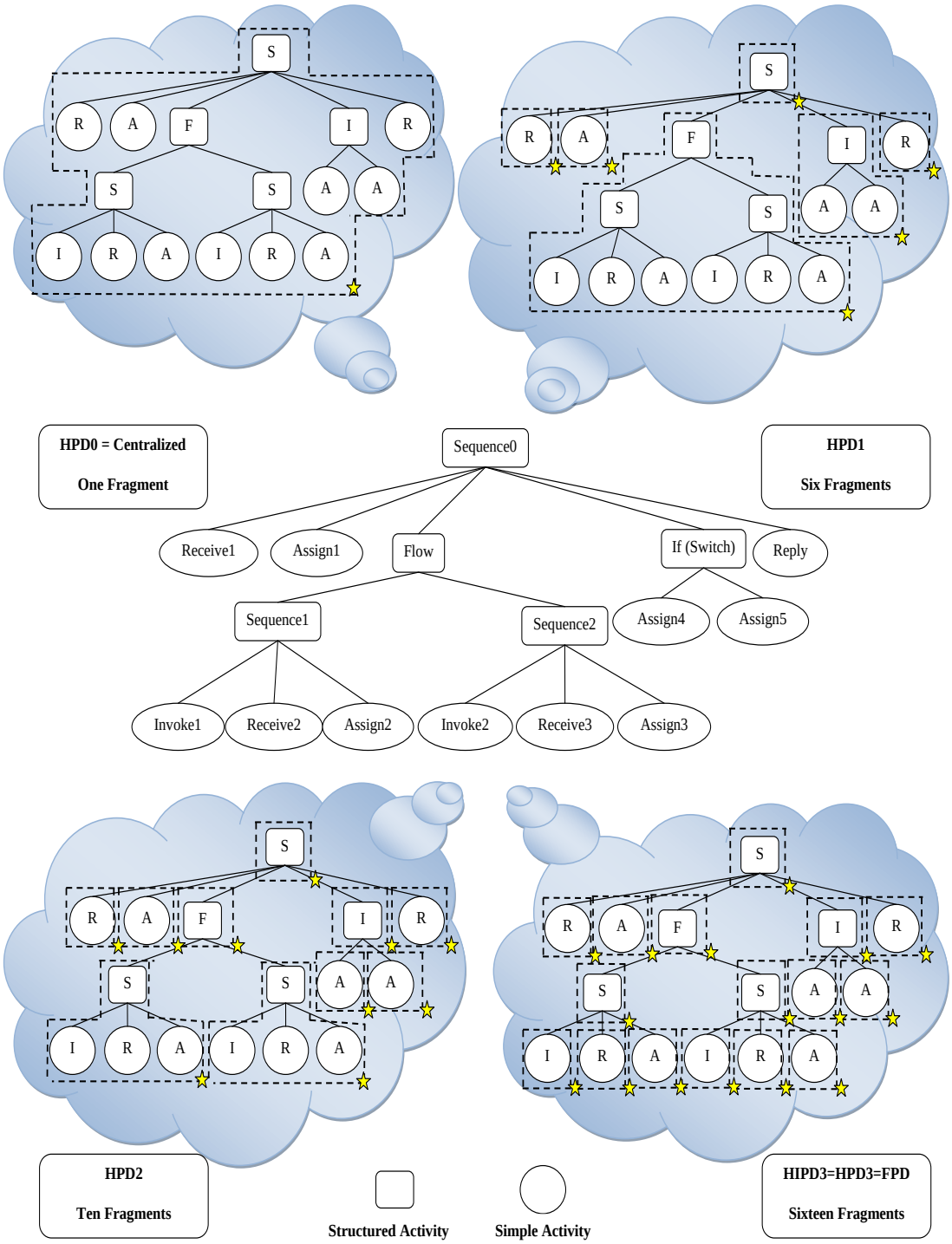


Figure 5.11: HPD Decentralization of Loan Application Business Process

From HPD point of view, the process can be fragmented in four levels. HPD0_1, which is fragmentation in *level-0* wraps the whole process tree and is equal to the Centralized method. HPD1_6, HPD2_10 and HPD3_16 are also subsequent levels of fragmentation; however, HPD3_16 is decentralization in the last level, which is equal to the FPD.

5.7 Hierarchical Intelligent Process Decentralization (HIPD)

This section introduces a decentralization approach called Hierarchical Intelligent Process Decentralization (HIPD). As a matter of fact, the aforementioned HPD is a general method used when mining data is not available and has a top down view to business process decentralization. HIPD applies HPD and a frequent path mining together; when mining patterns are available.

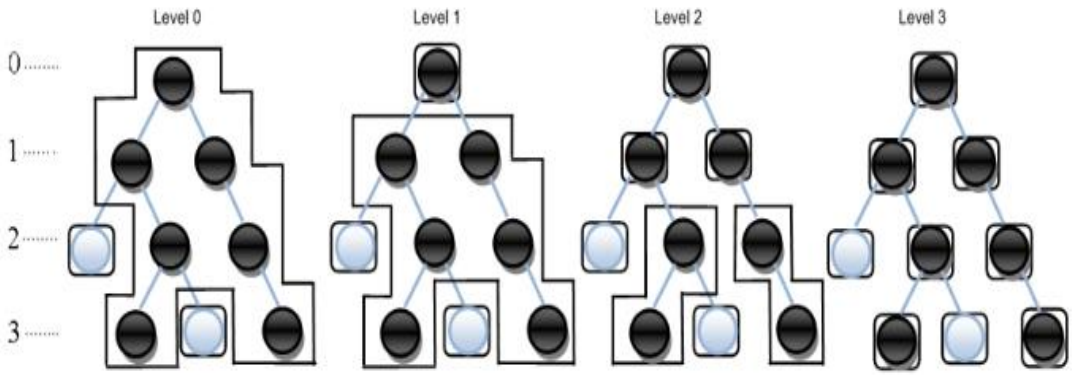


Figure 5.12: Hierarchical Intelligent Process Decentralization (HIPD)

Figure 5.12 is based on the assumption that dark nodes of a process tree are detected as frequent nodes. Basically, just frequent nodes are considered in frequent paths and infrequent ones are excluded. By incorporating the HPD view to decentralization, there are three fragments in *level-0* among which two fragments include only one activity and one fragment includes a frequent path. Accordingly, similar policies can be applied for subsequent levels.

The main purpose of the HIPD is to decentralize BPEL processes intelligently by applying a mining approach. In fact, a HIPD-based DPDC is able to apply any frequent path mining algorithm such as (Dubouloz & Toklu, 2005) to decentralize a block/graph-structured workflow; nonetheless, block-structured workflows are stressed in this work. Particularly, this method is expected to work well for those kinds of workflows in which there is a high frequent mainstream to distribute tasks among web services and sub-workflows. The mining approach can be used for both

kinds of block-structured and graph-structured processes. The coarsest level of granularity in the HIPD method (HIPD0) is called IPD, because it is a common point between block-structured and graph-structured processes from frequent path mining view.

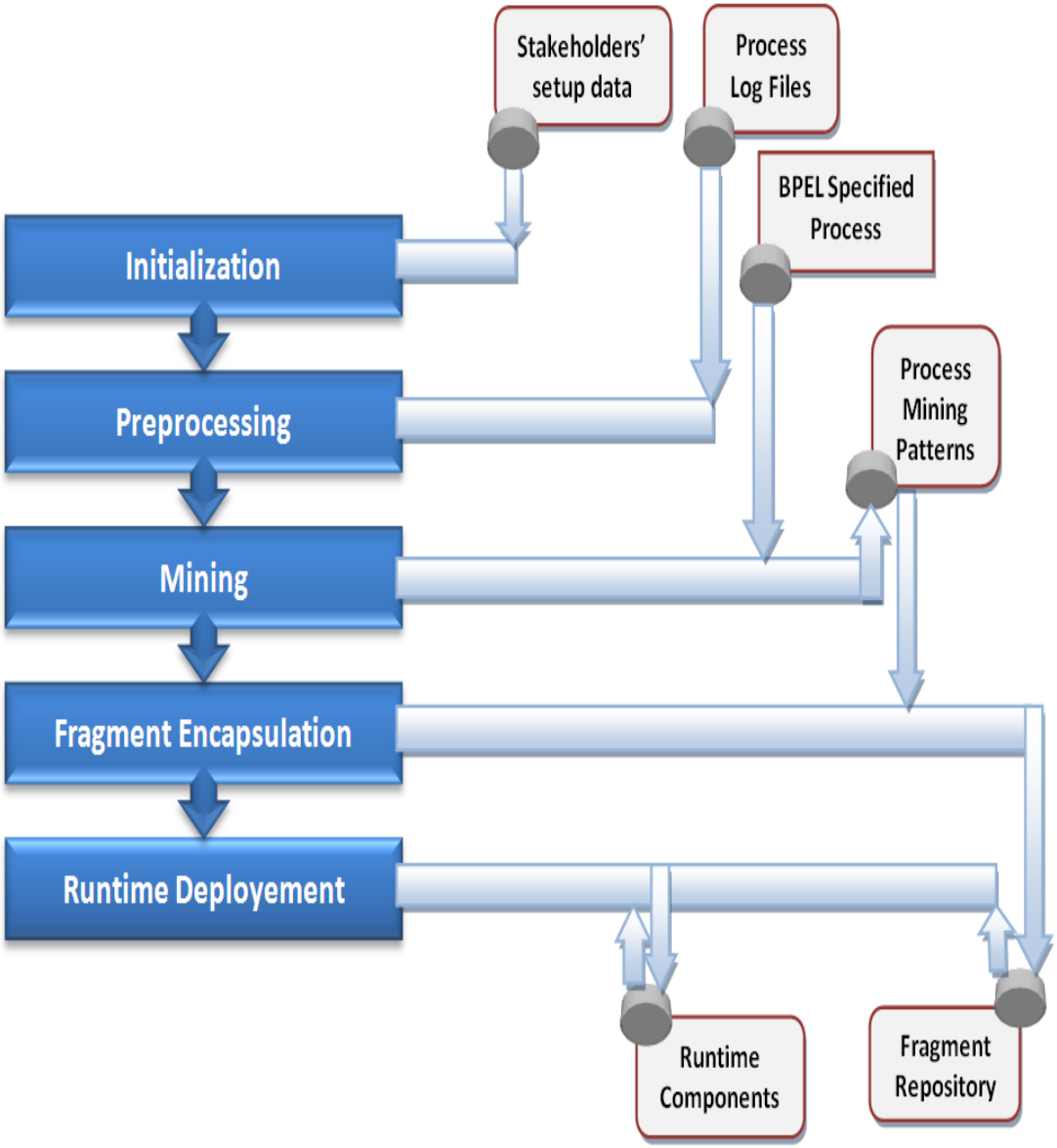


Figure 5.13: DPDC Structure for HIPD

5.7.1 HIPD-Based DPDC

In order to make a mining-based DPDC, five phases are recommended by Figure 5.13 including initialization, preprocessing, mining, fragment encapsulation, and runtime deployment which are introduced, respectively.

Phase1 (Initialization): this phase initializes the frequent path mining based on the information provided by system stakeholders that are: 1) the minimum frequency or minimum support of all activities; 2) the level of granularity for each workflow.

Phase2 (Preprocessing): after merging the log files, a preprocessor removes all noise data as is the case for all data mining applications. The noise data are extra information in log files that must be removed prior to starting the mining process. The number of distributed process log files is based on the number of machines involved in executing workflows so that each machine may have at least one log file. Process activities are executed and their execution is logged in log files. Most importantly, it is not a matter whether or not the log file is produced by a centralized instance of a process or a decentralized one. In addition, the log file can be located in a shared rapid disk to provide better performance. Alternatively, several mining approaches discuss on simultaneous processing of distributed log files as well (Bhaduri et al., 2011).

Phase3 (Mining): In the beginning, no mining information is available. DPDC embeds a decentralization decision maker that commences with the HPD method and recommends a suitable level of granularity based on current system circumstances. When the system is warmed up and the common behavior of processes is revealed, the process compiler uses up-to-date mining patterns to decentralize processes. Workflow Miner, which is the kernel part of the system mines the process log files and maintains the extracted decentralization patterns in a process mining pattern repository. The key steps of mining process as follows:

1. **BPEL process tree construction**: a BPEL *process tree* is constructed according to BPEL specification. Leaf nodes are naturally basic BPEL activities while non-

leaf nodes are structured ones. Indeed, the children of non-leaf nodes are in their immediate sub-level. In addition, the ordering of the children of each node from left to right shows the sequence of activities.

2. ***Synchronizing BPEL process tree and workflow tree****: workflow tree,* which is by definition all the executed paths in a process tree, must be synchronized with the process tree. It is achieved by marking the executed activities according to workflow log file information so as the visited counter of each node in workflow tree is incremented after each visit.
3. **Detecting frequent and infrequent paths**: It is worth mentioning that there is no stress on any frequent path detection and mining algorithm in this work. All previously presented frequent path detection algorithms can be applied in block/graph-structured workflows. Nonetheless, a granular frequent path detection algorithm is implemented in this section to granulate detected frequent paths.

Phase4 (Fragmentation): In this phase, a BPEL compiler translates the mined processes to workflow fragments and wires them so as they can communicate through a middleware. The fragments are then stored in a fragment repository.

Phase5 (Deployment): At deployment time the fragments are taken from the fragment repository, instantiated and sent to the runtime components.

5.7.2 HIPD Formalization

The idea of HIPD is formulated in this section. Set $A= \{a\}$ is introduced as a set of activities and set E as a set of edges that are used to make a tree of activities. Based on A and E, $BT= (A, E)$ stands for a block-structured process tree, which is described

by BPEL language. Furthermore, ET is considered as a tree built from a BPEL execution log file and definitely $ET \subseteq BT$.

A path is defined as a sequence of activities started from a process root to a leaf. A path is called *frequent path* when all of its activities are frequent. An activity is also called a frequent activity when its frequency value exceeds min_sup value (Han & Kamber, 2001). The min_sup is the minimum frequency value that a frequent activity must gain to be known as a frequent activity. In order to categorize activities and paths in a process tree several sets are required including *Frequent Activity (FA)*, *Frequent Path (FP)*, *Infrequent Activity (IA)* and *Infrequent Path (IP)* as follows.

FA is a set of frequent activities, which their frequency is bigger than *min_sup* according to Equation (5.2). The frequency of each activity is also calculated based on the type of activity as illustrated in Equation (5.3). If an activity is leaf; its frequency is equal to its iteration number. Otherwise, the frequency of the activity is equal to the frequency of the child with maximum frequency number.

$$FA = \{a \in A \wedge frequency(a) \geq \text{min}_sup\} \tag{5.2}$$

$$frequency(a) = \begin{cases} if\ chilren(a) = \Phi, return\ activity\ iteration\ number; \\ else, return \max\left(frequency\left(children(a)\right)\right); \end{cases} \tag{5.3}$$

There exist only one frequent path from root to leaves of a business process tree and set *FP* includes activities of this frequent path. Granulating this frequent path is important due to decentralization of a business process and providing a variety of process fragments. It comes about by considering levels of a process tree or granularity degree *G*, which is defined in Equation (5.4).

$$G = \{G_i | G_{depth-1} > \cdots > G_i > \cdots > G_{root} = G_0\} \tag{5.4}$$

Accordingly, FP_G illustrated in Equation (5.5) is defined as a set of granular frequent paths fp_G started from *level-G* of the tree to the last level of process tree (*depth-1*).

In order to formalize the granular frequent paths, function $level(G)$ is applied, which returns the nodes in *level-G* of the tree. If a path includes just one activity; it is not considered as a frequent path.

$$FP_G = \{fp_G\} = \{fp(a_i) | 0 \leq G \leq depth - 1, \forall a_i \in level(G), |fp(a_i)| > 1\} \quad (5.5)$$

Function $fp(a_i)$ in Equation (5.6) searches for subsequent frequent activities of the activity a_i and insert them to set fp_{ai}. If the activity a_i is a leaf node, then it is right added to the frequent path fp_{ai}. Otherwise, its subsequent children are checked for the same condition.

$$fp(a_i) = \begin{cases} if(children(a_i) = \varphi), fp_{ai} = fp_{ai} \cup a_{i;} \\ else\ , fp_{ai} = fp_{ai} \cup fp\big(children(a_i)\big); \end{cases} \quad (5.6)$$

Infrequent activities are defined in set IA shown in Equation (5.7) and includes those activities whose frequency is less than *min_sup*.

$$IA = \{a_i | a_i \in A, frequency(a_i) < \text{min_}sup\} \quad (5.7)$$

In addition, IP is the union of infrequent activities as well as frequent paths with the cardinality value equal to one as shown in Equation (5.8). The frequent paths with only one activity are considered as an infrequent path as well. It is worth mentioning that all paths in sets *IP* and *FP*, are encapsulated in runtime components, afterwards.

$$IP = IA \cup \{fp(a_i) | \forall a_i \in A, |fp(a_i)| = 1\} \quad (5.8)$$

5.7.3 **HIPD-Based DPDC Algorithms**

In this section the core algorithms applied in a HIPD-based DPDC are explained. The main algorithm applies three routines including *calculateNodeFrequency*, *determineSequentialPaths* and *FrequentPathFragmentation*, which are elaborated, respectively.

```
name: determineSequentialPaths;
input: node;
output: -;
begin
  for (counter = 0; counter < depth-1; counter ++ )
    for each node ε levelNth(root, counter)
      if ((node.frequency > min_sup) and (node. type = 'Sequence'))
        for each stNode ε sub_tree(node)
          stNode.frequency = min_sup;//is flagged as frequent
end;
```

Figure 5.14: Sequential Path Detection Algorithm

```
name: HIPD_Fragmentation;
input: root, min_sup, granularity;
output: root;
begin
  calculateNodeFrequency(root);
  determineSequentialPaths(root);
  frequentPathFragmentation (root, min_sup, granularity);
end;
```

Figure 5.15: HIPD Fragmentation Algorithm

HIPD Fragmentation: the HIPD fragmentation algorithm illustrated in Figure 5.15 aims at marking fragments in a process tree based on the information obtained by mining the log files. Three parameters are applied here including 1) the process tree; 2) the degree of granularity (*G*); and 3) the minimum support (*min_sup*) of the activities that shows the minimum frequency required for an activity to be included in a frequent path. In fact, the HIPD fragmentation algorithm can be specified in three phases. First) the frequency of each node in the process tree is calculated; second) sequential paths in the process tree are determined; and third) by considering frequent as well as infrequent paths each activity of the business process is marked by the fragment it belongs to.

Calculating Frequency of Activities: In order to detect frequent paths of a business process, execution frequency of each activity is required. The starting point to detect the frequency of activities is counting the number of visits for each activity. In order

to reach this goal, a workflow log file is used to calculate the number of visits for each node. According to Figure 5.16, the frequent path detection is based on the presumption that if one simple activity is visited *n* times, then its frequency is *n*. A complex activity contains a number of simple and complex activities; therefore, its frequency is equal to the frequency of the most frequent child.

```
name: calculateNodeFrequency;
input: node;
output: node frequency;
begin
  if (node.childrenNumber > 0)
  begin
    maxFrequency = 0;
    for each child ε node.children()
    begin
      tempFrequency=calculateNodeFrequency(child);
      if (tempFrequency > maxFrequency)
        maxFrequency = tempFrequency;
    end;
    node.frequency = maxFrequency;
  end;
end;
```

Figure 5.16: Node Frequency Calculation Algorithm

```
name: levelNth;
input: node, required level number;
output: A list containing nodes of Nth level;
begin
  traverse the tree using Breadth First algorithm;
  return a list containing the nodes of level-n;
end;
```

Figure 5.17: List of Nodes in *Level-n*

Sequential Path Detection: Simply, a sequential path is a sequence of activities, which run sequentially and the sequence activity in the BPEL language embraces them. In order to access the nodes in *level-n* of process tree, method *levelNth* is applied according to Figure 5.17. Detecting sequential paths starts from the root node (*level-0*) of the tree to level *depth-1*, because the Sequence is a structured activity and

cannot be a leaf node. If a Sequence node is a frequent node then all of its children are considered as frequent nodes. Figure 5.14 shows the sequential path detection algorithm as well.

```
name: frequentPathFragmentation;
input: root, min_sup, granularity;
output:-;
begin
  f = 0; //fragment counter
  depth = 0; //tree depth counter
  while (0≤ depth ≤ granularity)
  begin
    for each node ε levelNth(root, depth)
    begin
      node.fragment = fragment_f ;
      f++;
      if (depth = granularity)
        for each stNode ε (node ∪ sub_tree(node))
      begin
        if (stNode.frequency >= min_sup)
          stNode.fragment = node.fragment;
        else begin
          stNode.fragment = fragment_f ;
          f++;
        end;
      end;
    end;
    depth++
  end;
end;
```

Figure 5.18: Determining Fragments Based on Frequent/Infrequent Paths

Fragmentation Applying Frequent/Infrequent Paths: an algorithm is shown in Figure 5.18, which shows a way of marking the activities of the business process tree by the fragments they are belonged to. The algorithm requires the business process tree, minimum support (min_sup) and granularity level (G) parameters. Briefly, all the process activities, which are in the levels upper than G are marked as distinct fragments. Those activities in level-G along with their children with the frequency number larger than or equal to min_sup are marked with the same fragment name (fragmentf). Other activities are marked as separate fragments as well.

5.7.4 HIPD Decentralization of Loan-process

In order to perform HIPD fragmentation, a workflow execution dataset (log file) was provided. Precisely, to prepare the dataset, *ActiveBPEL* open-source workflow engine version 2.1 (Active-Endpoints, 2011) and tomcat servlet container version 5.0.28 (Tomcat, 2011) were used to run the Loan-process.

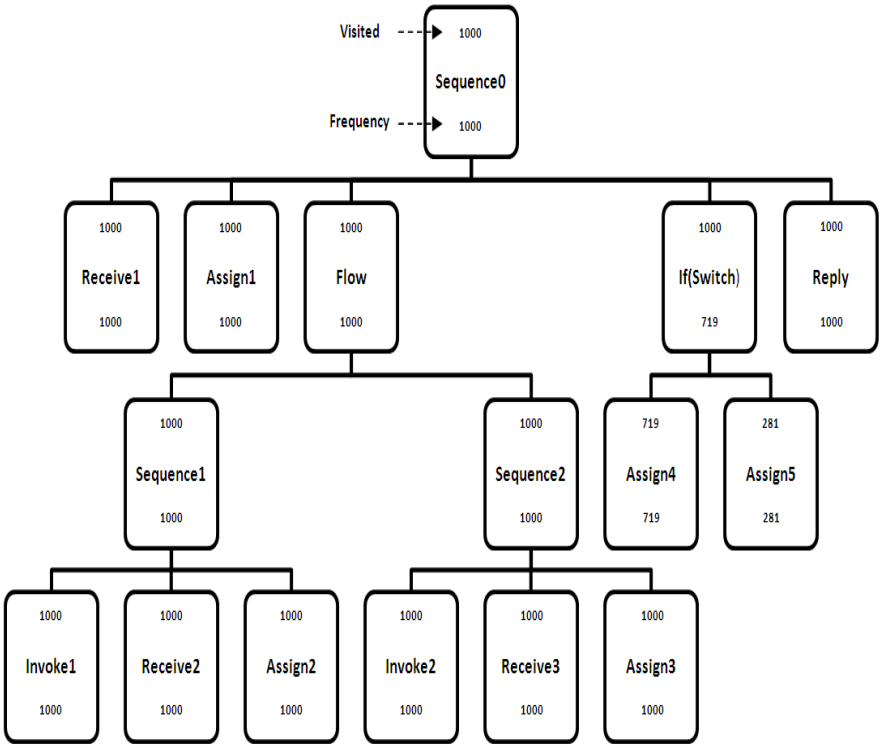


Figure 5.19: Loan Application Process after Calculating Frequency of Nodes

After preprocessing the log repository, the mining algorithm came into action. According to the *calculateNodeFrequency* algorithm shown in Figure 5.16, for each node there exist *visited* and *frequency* fields and their values are respectively shown on the top and bottom of the nodes in Figure 5.19. The *visited* and *frequency* counter for all leaf nodes are similar; however, the frequency of non-leaf nodes is equal to maximum frequency number of their children. After detecting the sequential paths of the process tree by *determineSequentialPaths* method in Figure 5.14, the tree is sent to *frequentPathFragmentation* algorithm in Figure 5.18. The output is a fragmented business process so that each process activity is categorized in a specific fragment.

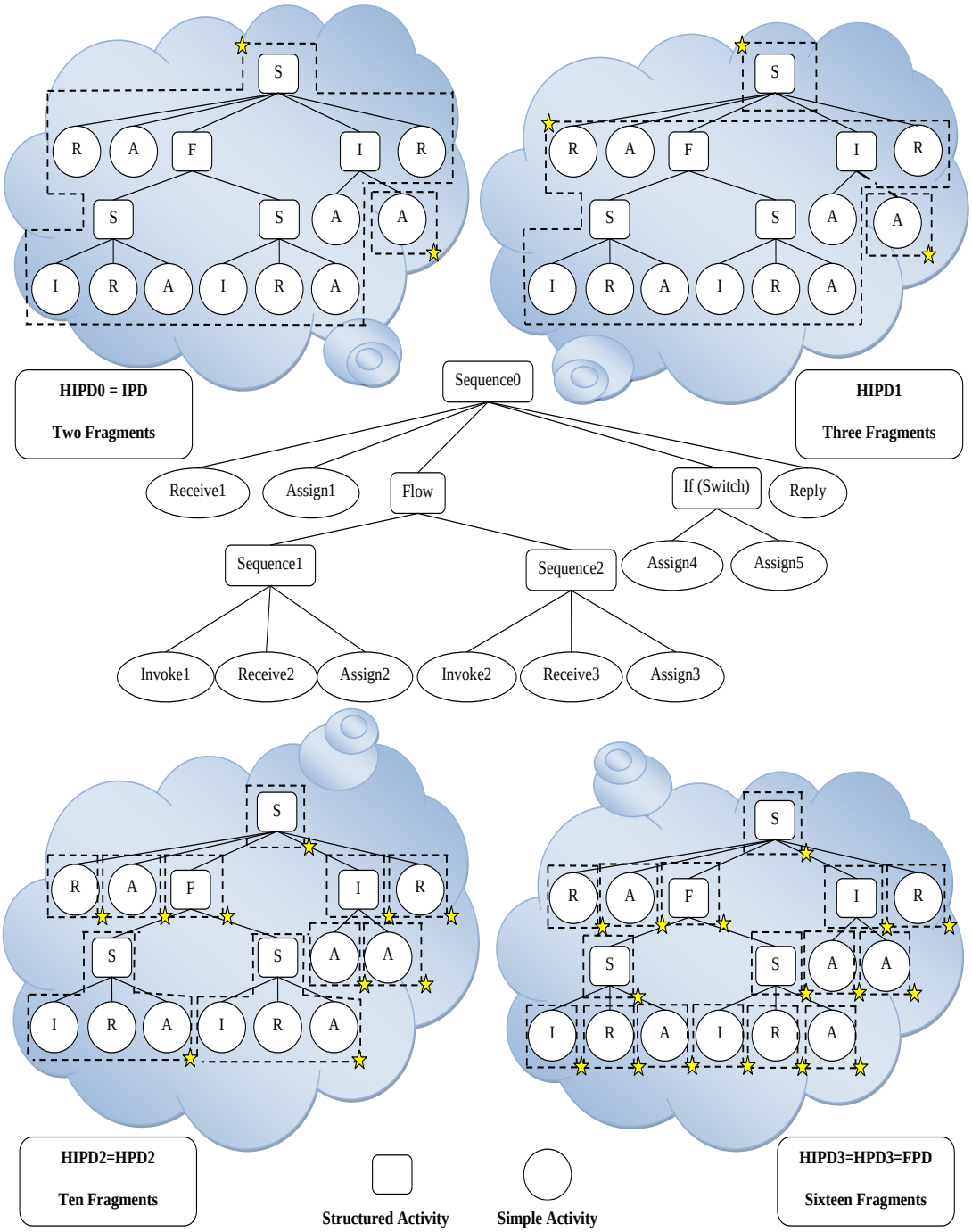


Figure 5.20: HIPD Decentralization of Loan Application Business Process

The execution result of the algorithm is depicted in Figure 5.20, where fragments are marked by an asterisk sign. The HIPD0, which is also called IPD, is the first level of decentralization based on detecting a frequent path. In this experiment, it is assumed that requests for the loan are mostly accepted and path [Sequence0 (Receive1, Assign1, Flow (Sequence1 (invoke1, Receive2, Assign2) || Sequence2 (Invoke2,

Receive3, Assign3)), If (Assign4), Reply1] is a frequent path and [Assign5] is infrequent, which is excluded from the path. In order to make the above path frequent, the client was set to invoke the Loan-process 1000 times so that 70% of invocations resulted in approbating and 30% in rejecting the loan application. Minimum support was also considered equal to 3% as well. In addition, there exists two overlaps among the HPD and HIPD fragmentation of the Loan-process, which are HPD2 = HIPD2 and HPD3=HIPD3=FPD.

5.8 Wiring of Fragments in ADWEF

Wiring of workflow fragments means realizing links connecting activities inside the fragments at runtime. In fact, when a business process is targeted at being decentralized, the connecting links must be substituted by communication facilities. Generally, a process fragment may consist of several incoming and outgoing links and for each sort of links a specific policy must be applied. Process fragments are made from an original business process; therefore, the addressing of activities and connecting them is possible by having the original process specification. In order to implement the input/output links, different policies might be applied by a business process compiler. Hereafter, at first a general structure for the workflow fragments is recommended and then an algorithm is presented to produce a fragment based on the fragment structure and original business process specification.

Figure 5.21 recommends four building sections for a business process including input-parameters, link-activity-mapping, body and output-parameters. When other parties call a fragment as a subflow, a list of parameters is sent to the fragment. The input-parameter section is responsible for extracting and managing the parameters. Following the input-parameter is the link-activity-mapping section, which maps the

incoming links to the respective functions in the body section. When other parties targeted at invoking the fragment, the link between two activities is translated to a subflow invocation and the target function is included in the invocation as a parameter. The link-activity-mapping section subsequently calls a relevant function as soon as receiving the function name.

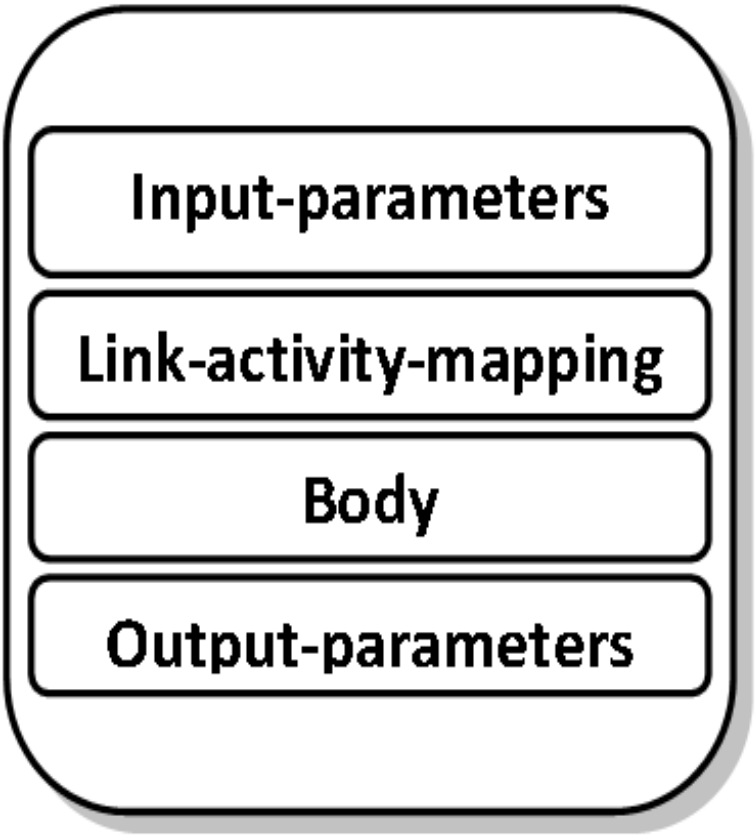


Figure 5.21: Workflow Fragment Building Sections

The body section is actually the repository of the functions corresponding to the business process activities. It means that each business process activity must be compiled to its equivalent function in the body section. In addition, the outgoing links of each activity must be translated to either internal function calls or external subflow invocations in the cases of inter-fragment or intra-fragment outgoing links. When a fragment is called, a list of parameters is returned back to the caller as a reply to the invocation. The output-parameter section accumulates the parameters and provides the final reply.

Figure 5.22 illustrates an algorithm to wire fragments of a business processes. Each business process activity assumed to have a number of incoming and outgoing links. The algorithm traverses the business process activities and after coming across activities, incoming and outgoing links, their corresponding parts are created in the relevant sections in the fragment shown in Figure 5.21. Each of the fragment sections is considered as an object, which is accessible through its methods.

```
name: fragmentWiring;
input: Set of activities;
output: -;
Begin
  FragmentList fList;
  i = 0;
  for each activity ε activities
  begin
    if (activity.fragment is not in fList )
    begin
      fragment = new FragmentSection();
      fList[++i] = fragment;
    end;
    else fragment = findFragment(fList, activity);
    function = fragment.body.createSpecificFunction(activity);
    for each link ε activity.incomingLinks()
    begin
      if (activity.fragment ≠ link.source.fragment)
      begin
        parameters = link.getParameters();
        fragment.link-activity-mapping.addInvocation(link.activity, parameter);
      end;
    end;
    for each link ε activity.outgoingLinks()
    begin
      parameter = link.getParameters();
      if (activity.fragment = link.destination.fragment)
        function.addInvocation(link.destination, parameter);
      else
        function.createSubflowInvocation(link.destination, parameter);
    end;
  end;
end;
```

Figure 5.22: Wiring of Fragments by Traversing Original Business Process

The algorithm assumes that each activity of a business process has already been marked by a fragmentation method and fragment name of the activity is accessible through *activity.fragment* field. An equivalent fragment is created for each activity unless it has already been created. For each specific activity its equivalent function is created in the fragment body section and its input/output variables are set. After that it is the turn of activity incoming/outgoing links to be implemented. Each incoming link is simultaneously an outgoing link of another activity; therefore, it will be turn to an outgoing link implementation. The only exception is for the incoming links, which come from other fragments. In this case, the links are actually incoming links of the whole fragment. Indeed, they have already been substituted by a subflow call in the caller fragment. In the called fragment, the link must be mapped to its analogous function in the fragment body section, which is a simple local function call. The same view is for the activity outgoing links so that the links among the activities are translated to the local function calls inside a fragment while the links among the fragments are translated to their corresponding subflow calls.

5.9 Creating Dynamic Fragments Based on WADE/JADE Facilities

At least two methods can be considered to create workflows dynamically at runtime including *on-the-fly class creation* and *on-the-fly object creation*. Statically created classes have to be compiled and got alive by connecting to the workflow state object(s). Alternatively, in the on-the-fly object creation method, every element of a workflow has to be created at runtime and then connected to the state object(s) to get alive.

For the former method, normally a workflow is an instance of a class that extends a *WorkflowBehaviour* class in WADE/JADE. Hence, a new workflow class can be

created by putting everything in Java classes at runtime. By creating an instance of the class a new workflow is realized at runtime.

For the latter method, an instance of the *WorkflowBehaviour* class must be directly created and then activities and transitions are added to it by means of the *registerActivity()* and *registerTransition()* methods. Typically, data produced by an activity are inputs of a successive activity. This can be done by means of binding mechanism so that each activity has a *BindingManager* (that can be retrieved by means of the *getBindingManager*() method). The *BindingManager* allows the output of an activity get bounded with the input of another one. Once a *WorkflowBehaviour* was created, a *WFE* agent must execute it. It can be done in two ways based on WADE/JADE facilities. The first method is serializing the *WorkflowBehaviour* object into a *WFE* agent. The second method is sending a *REQUEST* message to the agent that has to execute the workflow. Possibly, it can be the same agent that created and instructed the *WorkflowBehaviour* object as described. Then it requests the *ExecuteWorkflow* action and specifies a *WorkflowDescriptor* with a serialized *WorkflowBehaviour* object as an input parameter.

5.9.1 Relations of Workflow Decentralization Methods

There exist relations among different decentralization functions, which are shown in Figure 5.23. In some cases, fragment sets produced by different methods may have some intersections. As mentioned before, $\varphi_{Centralized}$ translates a business process to only one fragment $|\varphi_{Centralized}| = 1$, which contains the whole process and runs as a multi-thread object on one machine. In addition, φ_{FPD} fragments a business process to the finest level of granularity so that each fragment contains one process

activity. Indeed, the FPD method for an n-activity process results in n fragments, which means $|\varphi_{FPD}| = n$.

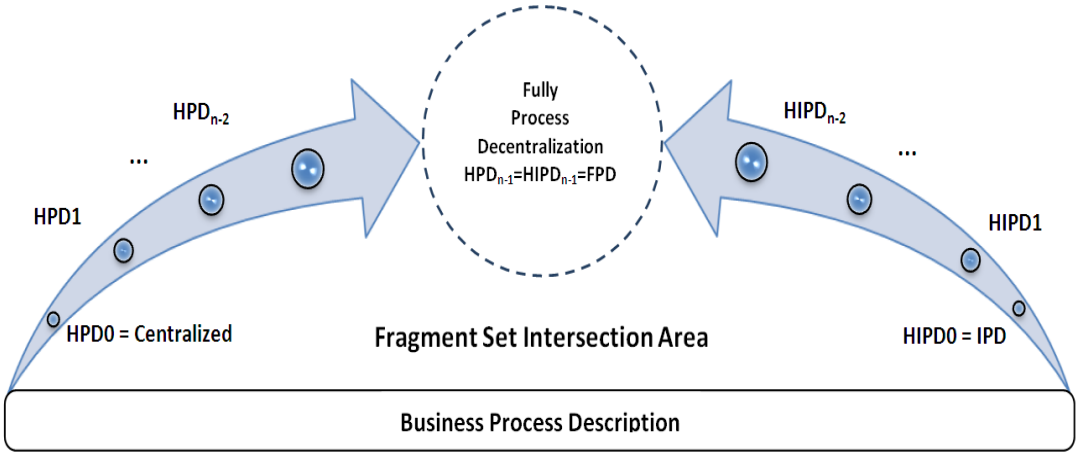


Figure 5.23: Decentralization Policies Relations

Furthermore, φ_{IPD} fragments a business process to one frequent path as well as infrequent paths. These functions work well for both block/graph-structured business processes. The HIPD and HPD methods are based on the hierarchy of activities in a business process tree; therefore, the assumption here is that business processes are block-structured.

In order to show a set of fragments in granularity g, the method name along with the granularity level g is considered. For instance, HPD_g (where $0 \leq g < depth$) is a set of fragments produced by φ_{HPD} for level g of the process tree. Both HPD and HIPD methods for each process $p_i \in P$ have at least one similar fragment, $|\varphi_{HPD}(p_i) \cap \varphi_{\mathrm{HIPD}}(p_i)| \geq 1$ which is φ_{FPD}. In addition, both IPD and HIPD methods have exactly one similar fragment, $|\varphi_{IPD}(p_i) \cap \varphi_{\mathrm{HIPD}}(p_i)| = 1$, which is HIPD_0. The IPD for block-structured BPEL processes detects a frequent path from a process root to process leaves. In the case of the loan application process, there exists two overlaps among the HPD and HIPD fragmentation methods, which are HPD2 = HIPD2 and HPD3=HIPD3=FPD.

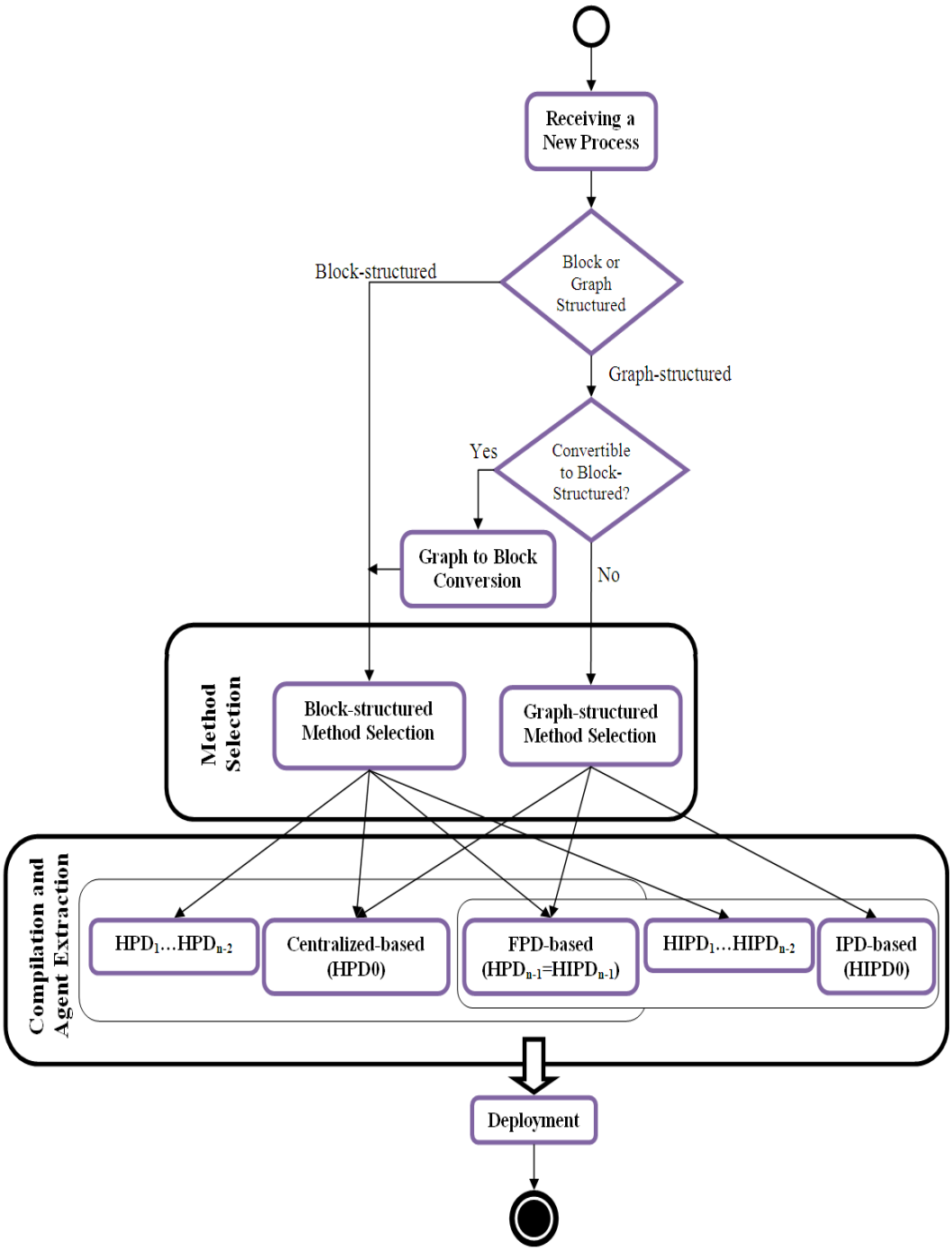


Figure 5.24: Choosing a Suitable Decentralization Method Based on BPEL Type

5.9.2 Decentralizing Block-structured vs. Graph-structured BPEL

The dependency of process activities is determined by workflow metaphors, which can be block or graph structured. According to the block/graph-structured part of the literature review, the block-structured BPEL is more popular compared to the graph-structured one which is the reason why this study focuses on the block-structured

BPEL. Nonetheless, a decentralized workflow engine should be able to run different types of a business process. Based on the aforementioned methods of executing a business process, different methods can be taken into account to execute block/graph-structured business processes.

According to Figure 5.24, a decentralized workflow engine is able to execute graph-structured processes by compiling them to the Centralized, IPD or FPD models. In contrast, block-structured processes can be executed as fragments of the HIPD, HPD, IPD, FPD and Centralized methods. It is worth mentioning that the IPD is equal to the HIPD0 for block-structured business processes and it decomposes block/graph-structured business processes to frequent and infrequent paths.

5.10 Adaptable Decentralization Decision Making Unit

This section elaborates on an adaptable decentralization decision making unit. Adaptability can be studied from different aspects such as adaptability with memory consumption, bandwidth usage, throughput, etc. and mostly having all of them together is impossible due to confliction of goals. This chapter also opens discussions on the adaptability of workflow fragmentation by considering runtime circumstances in business process decentralization. Different methods of decision making such as intelligent or unintelligent methods may go to action to provide more accurate decisions.

5.10.1 **Runtime-adaptable Decentralization**

As mentioned in the prior sections, the frequent-path adaptability is realized by the HIPD decentralization method. In addition, the fragments provided by both HPD and HIPD decentralization methods provide a variety of fragments. They pave the way for satisfying the frequent-path and proportional-fragment aspects of adaptability.

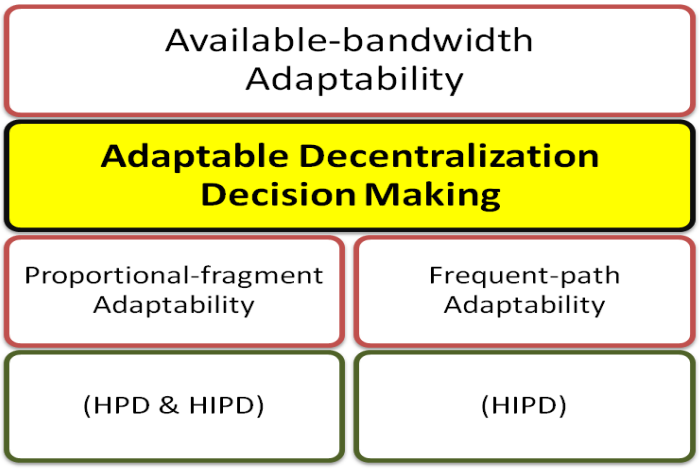


Figure 5.25: Three Adaptability Aspects Meet Together

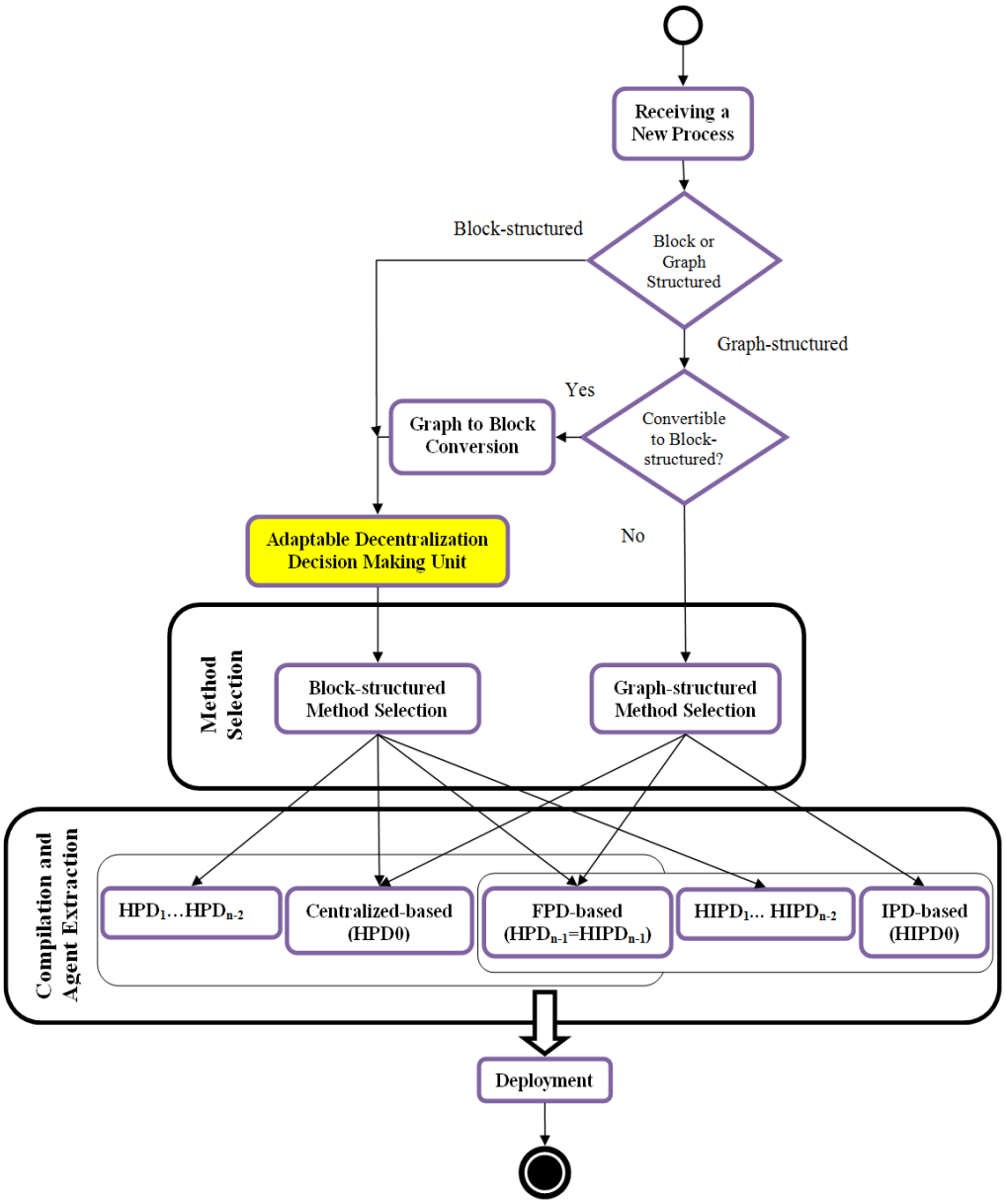


Figure 5.26: Adaptable Decentralization Decision Making Unit

These two adaptability aspects are the basis of the available-bandwidth adaptability. As shown in Figure 5.25, the three adaptability aspects meet together by means of the adaptable decision making unit. To put it in another way, Figure 5.24 is promoted to Figure 5.26 which is equipped with the adaptable decision making unit, to make a decision on the method and level of decentralization for each requested process. This thesis applies the fuzzy logic for implementing this unit as well.

5.10.2 **Fuzzy Decentralization Decision Making Unit (FDDM)**

Bandwidth plays a major role in performance of distributed systems in that it is independent from other parameters such as response-time and throughput; however, they are highly affected by the available bandwidth. A busy communication media may increase the latency of communication among components, which results in increasing response-time and decreasing throughput.

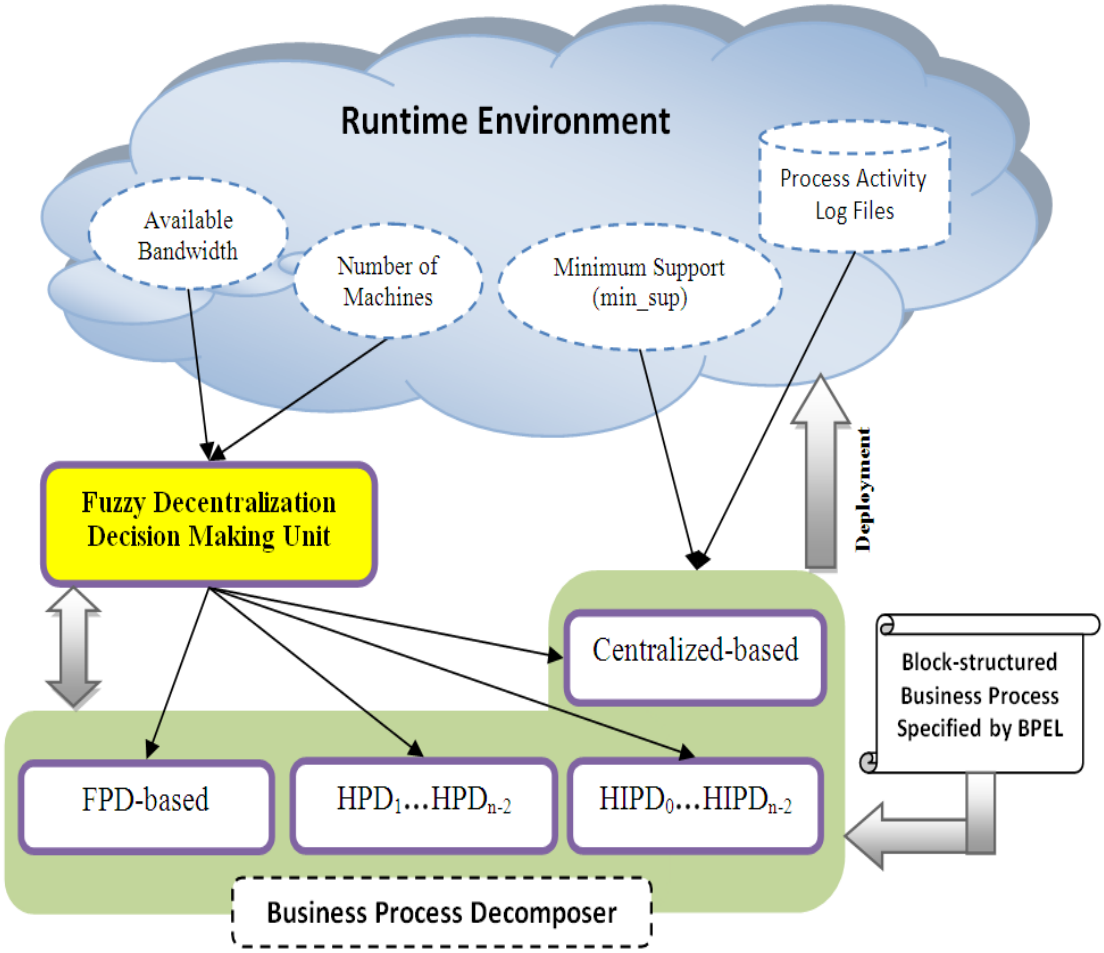


Figure 5.27: Fuzzy Decentralization Decision Making Unit

The FDDM unit presented in this section satisfies the available-bandwidth adaptability by offering a suitable method of decentralization based on current runtime circumstances. As a matter of fact, the FDDM is established on the frequent-path and proportional-fragment adaptability and offers a suitable fragmentation set, which is more commensurate with current runtime. Figure 5.27 illustrates the fuzzy decentralization decision making unit. The process specification, available bandwidth and number of machines are inputs of the unit while the level of granularity and selected decentralization method are outputs of the algorithm that are used by business process decomposer. The number of machines is used to calculate fragment proportionality and process activity log files and min_sup are used in workflow decomposer to produce HIPD fragments, which are frequent-path

adaptable. When process mining data is not available, the algorithm only applies the HPD fragments; otherwise, it uses both HPD and HIPD fragments together.

5.11 Miscellaneous Aspects of ADWEF Architecture

There are several aspects of the ADWEF architecture, which will not be focused on in this thesis. Nonetheless, it is worth giving a brief explanation about how these issues can be handled. **Governance of workflows** determines which workflow fragments must be governed by which organizational unit. These decisions are usually made by designers at design time so that the designers mark an activity or a set of activities as a fragment and assign them to a specific unit to govern. There should be some facilities in a workflow engine to mark activities to determine which of them must be fragmented, or not, or can be fragmented and run only by specific machines inside/outside an organizational unit. **Managing state of workflows, transactions, and exception handling** are also the common issues of decentralized workflows, which are usually studied together. Although a database can be used to manage the state and eventually transaction and exception handling (Mohan et al., 1995), research attempts such as (Mirkov Viroli, 2007; Wutke et al., 2008) used to implement these aspects through the tuplespace concept. There are even some commercial implementations such as JavaSpaces (Freeman et al., 1999; Philip & Nigel, 2003) and Camel (Apache-Camel, 2011) to handle state, transaction and exception handling in distributed systems. In case of this thesis, shared variables are managed by a shared tuplespace (Carriero & Gelernter, 1989). A variety of design patterns can be considered for managing workflow states. The dynamic fragmentation needs to adopt a unique ID for each workflow activity, which is usually the case, and make it independent from the fragment in which it is encapsulated. In other words, if a workflow requires re-fragmentation/re-

configuration then it must be able to retrieve its last state in a new fragment after being re-fragmented. Fragmentation/re-fragmentation algorithm can be activated when 1) a new workflow is requested; 2) a suspended long-running process goes alive; and 3) system thresholds are violated. Each of these reasons requires individual/batch re-fragmentation of workflows based on current runtime circumstances.

5.12 Summary

This chapter represented the proposed model of this thesis for Adaptable and Decentralized Workflow Execution Framework (ADWEF) along with two architectures to realize it. After that two decentralization policies were introduced called HPD and HIPD. The former decentralized business processes based on a hierarchical view while the latter combined the hierarchical view and a mining approach to decentralize processes. Later, a general algorithm for fragmenting business processes was presented. Finally, a fuzzy decentralization decision making unit was represented that realized the available-bandwidth adaptability based on the frequent-path and proportional-fragment aspects of adaptability.

CHAPTER 6

RESULTS AND DISCUSSION

6.1 Introduction

This chapter focuses on the evaluation of the ADWEF framework. Several experiments are presented and discussed according to the aforementioned methodology presented in Chapter 4 to study different aspects of the ADWEF framework introduced in Chapter 5. Section 6.2 introduces the decentralization of the basic BPEL processes based on both HPD and HIPD fragmentation methods. Section 6.3 introduces several boundary conditions experiments to evaluate the behavior of the most basic workflows implemented by the WADE/JADE agents. Later, in Section 6.4, several experiments are shown to evaluate the decentralization methods; finally, in Section 6.5 the Fuzzy Decentralization Decision Making unit (FDDM) is examined.

6.2 Fragmenting Basic and Complex Business Processes

This section introduces the HPD and HIPD decentralized versions of the reference BPEL processes introduced in Chapter 4. Three very basic business processes as well as one complex process are introduced. The study of the basic business processes paves the way for examining their central activity such as IF, While and Flow and clarifies how different decentralization policies may have their own specific impact on the basic process execution. A complex loan application business process also shows how combination of different activities behaves in different decentralization situations.

6.3 Boundary-condition Experiments

Boundary conditions experiments are expected to show the response-time, throughput and number of exchanged messages among two WADE/JADE elementary agents, when they contain elementary fragments with the least functionality. The whole experiments were based on the Section 4.8.1, which specifies how a very basic business process can be implemented in four configurations as shown in Figure 4.4. In order to evaluate the behavior of such a simple workflow, they were implemented using WADE/JADE performer agents. A client invoked the workflow with variable-request-rate/constant-message-size and the workflow was tested by different iteration values $n = 1, 4, 10$, which determined the number of communications between two fragments and then response-time, throughput and the number of exchanged messages were calculated using the gathered data.

6.3.1 Experiment-1 (evaluating response-time, variable-request-rate, constant-message-size)

This experiment was conducted according to Experiment-1 in boundary conditions specified in Section 4.8.1. Figure 6.1 shows that increasing the number of interactions (k) between two fragments resulted in sharp rising of response-time in four configurations. Briefly, Config-1 outweighed other configurations due to the low number of message passing. It means exchanging messages is a serious constraint in decentralization of workflows. Config-1 and Config-3 substantiated that keeping closely interrelated activities together resulted in better response-time. Config-3 also outweighed Config-4, despite of using two machines. Config-4, which was completely a distributed case suffered from the external message passing for communication of fragments. Although, Config-2 used an internal communication, it

showed the worst response-time. In fact, it showed a limitation of WADE/JADE performer agents to execute different types of fragments, simultaneously.

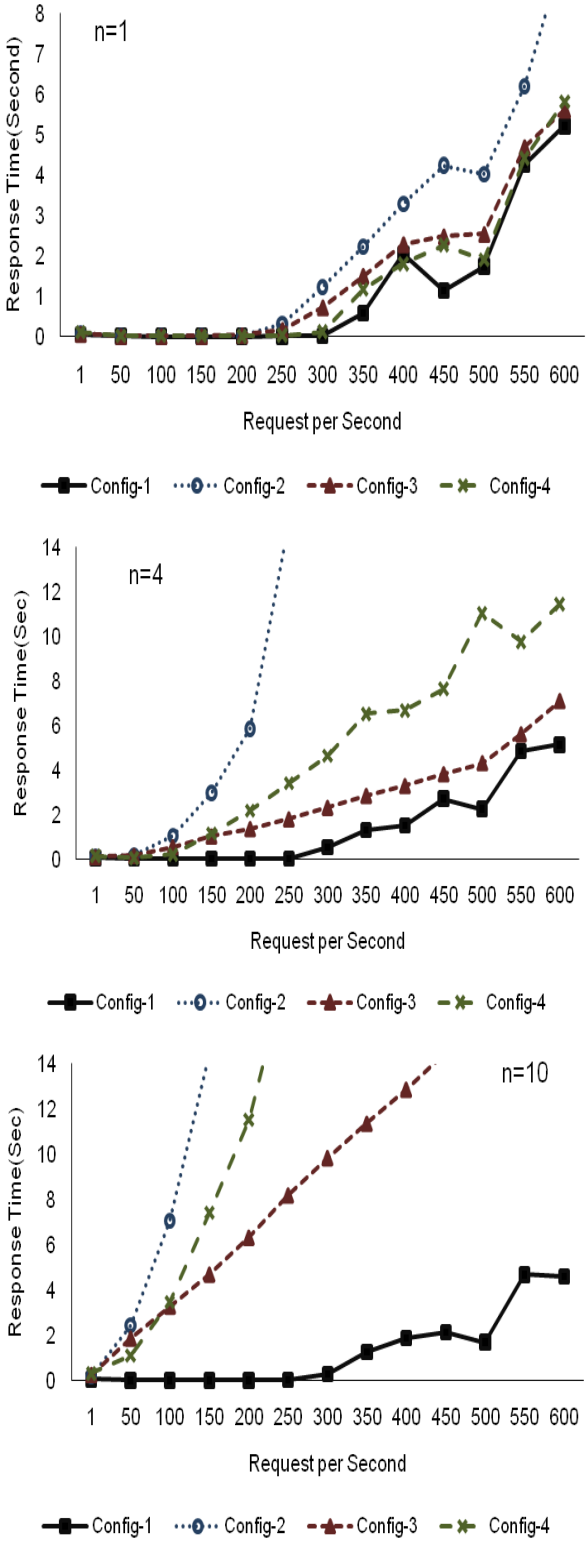


Figure 6.1: Boundary–condition Experiment, Response-time Comparison , Variable-Request-Rate, Constant-Message-Size

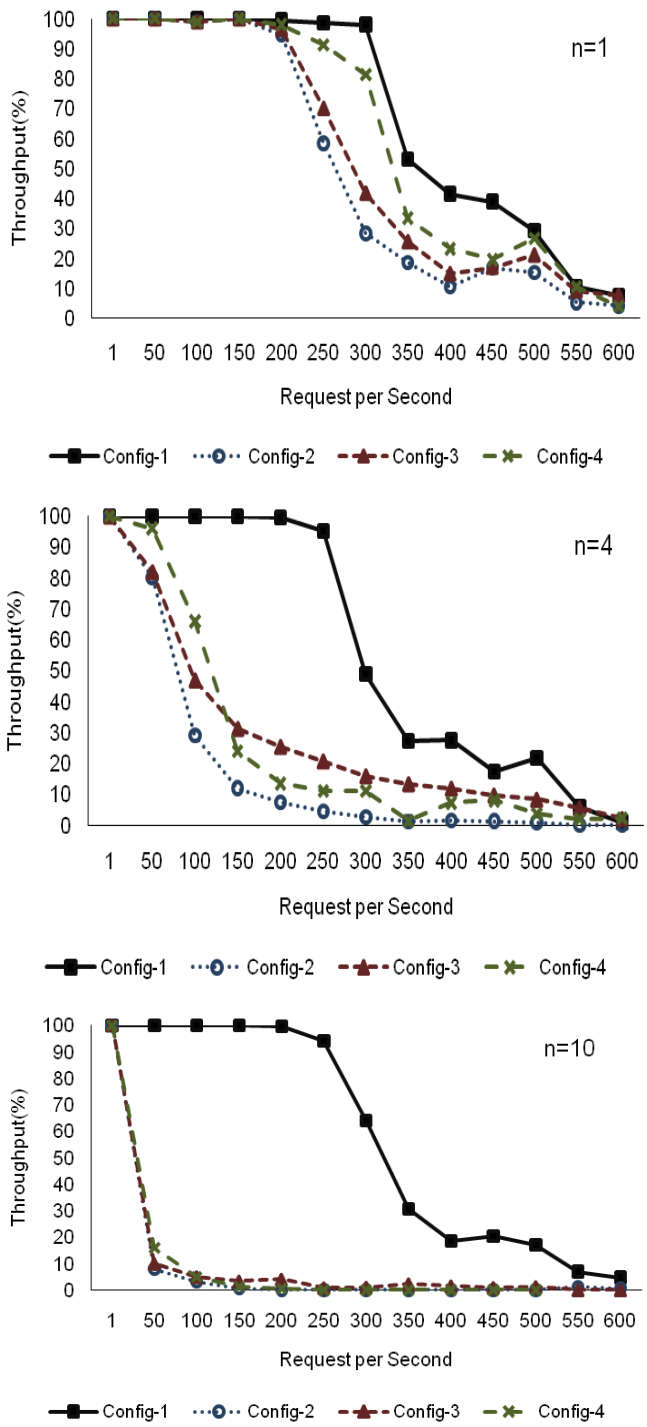


Figure 6.2: Boundary-condition Experiment, Throughput Comparison, Variable-Request-Rate, Constant-Message-Size

6.3.2 Experiment-2 (evaluating throughput, variable-request-rate, constant-message-size)

This experiment was conducted according to Experiment-2 in boundary conditions specified in Section 4.8.1. It is obviously shown in Figure 6.2 that by increasing

number of iterations (*n*), the number of exchanged messages between two activities was increased and the throughput in Config-2 sharply plummeted. Config-3 and Config-4 showed similar behavior as well. In Config-3, an internal message passing was applied and it showed better results compared to Config-2. Config-4, which was distributed between two machines, was outweighed by Config-1 and Config-3 after increasing *n*. Finally, Config-1 due to the low number of message passing outperformed other configurations and it showed that message passing was a serious constraint in decentralization of workflows. Config-1 and Config-3 substantiated that keeping closely interrelated activities together resulted in better throughput. Although, Config-2 used internal communication, it showed the worst throughput. It could also be a limitation of WADE/JADE performer agents to execute different types of fragments, simultaneously.

6.3.3 Experiment-3 (evaluating response-time, constant-request-rate, variable-message-size)

This experiment was conducted according to Experiment-3 in boundary conditions specified in Section 4.8.1. Figure 6.3 shows how the message size affected the response-time. Particularly, after increasing the number of iterations (*n*), when the request rate was equal to 500 requests per minute, the response-time steeply increased. Even in Config-1 in which there was no message passing between the fragments, when the message size exceeded 50KB, the response-time rocketed.

Another point was a better response-time for Config-4 compared to Config-2 and 3, when the message size was increased. It showed that increasing the message size resulted in lack of memory in Config-2 and 3 due to long messages queues. Yet another point was the climbing of response-time in Config-2 compared to Config-3. It was due to more parallelism of fragments in Config-3. The Config-3 executed the

fragments in two performer agents with separate queues and memory quota while Config-2 had more constraints to manage the fragments.

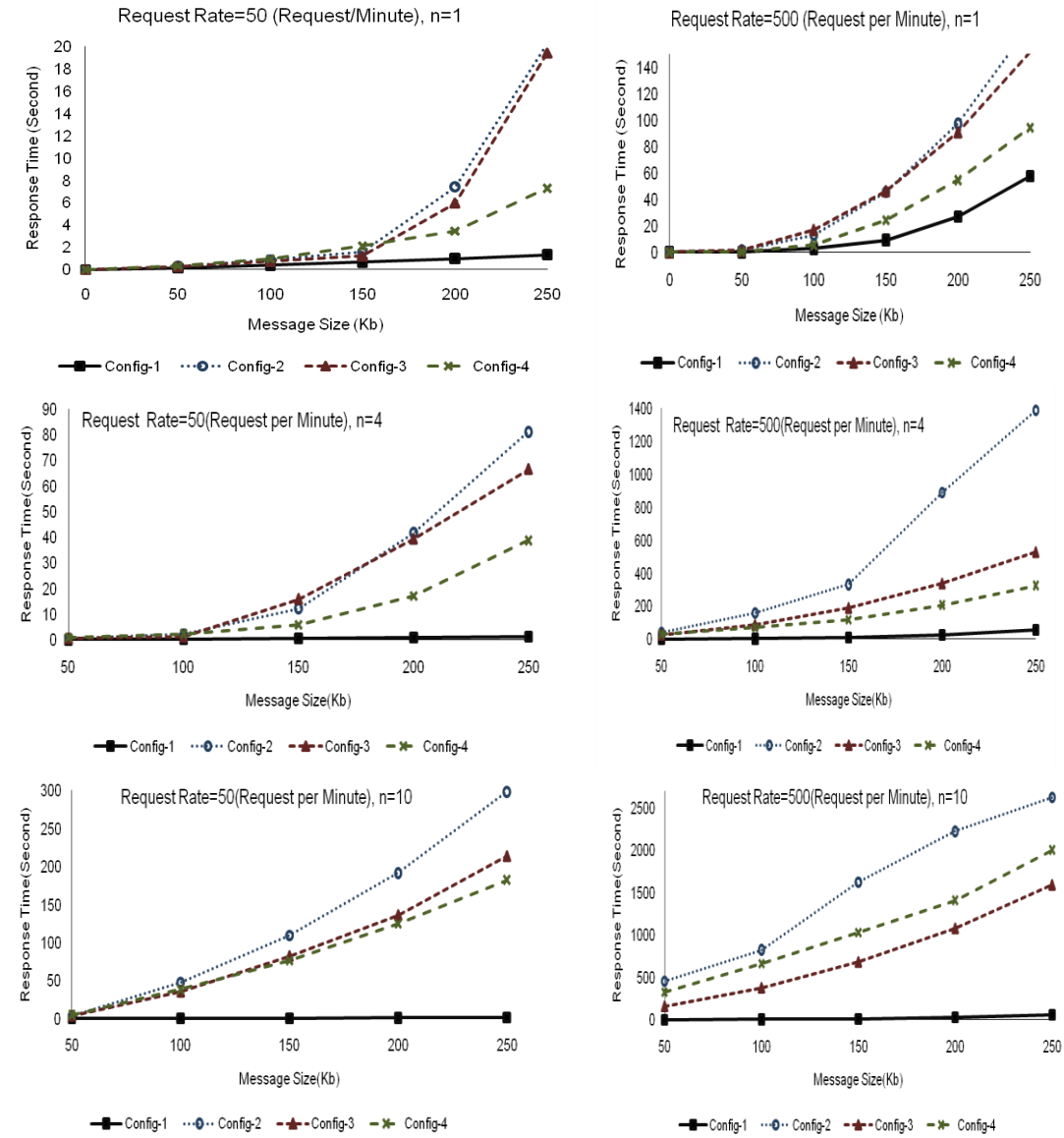


Figure 6.3: Boundary-condition Experiment, Response-time Comparison, Constant-Request-Rate, Variable-Message-Size

6.3.4 Experiment-4 (evaluating throughput, constant-request-rate, variable-message-size)

This experiment was conducted according to Experiment-4 in boundary conditions specified in Section 4.8.1. Figure 6.4 shows how the message size affected the throughput. Particularly, after increasing the number of iterations (*n*), when request rate was equal to 500 requests per minute, the throughput declined sharply. Even in

Config-1 in which there was no message passing between the fragments, when the message size exceeded 50KB, the throughput fell slightly.

Another point was better throughput in Config-4 compared to Config-2 and 3, when the message size was increased. It showed that increasing the message size resulted in lack of memory due to long message queues. Yet another point was the plummeting of throughput in Config-2 compared to Config-3. It was due to more parallelism of Config-3 as well. Config-3 executed the fragments in two performer agents with separate queues and memory quota while Config-2 had to manage the fragments with more constraints.

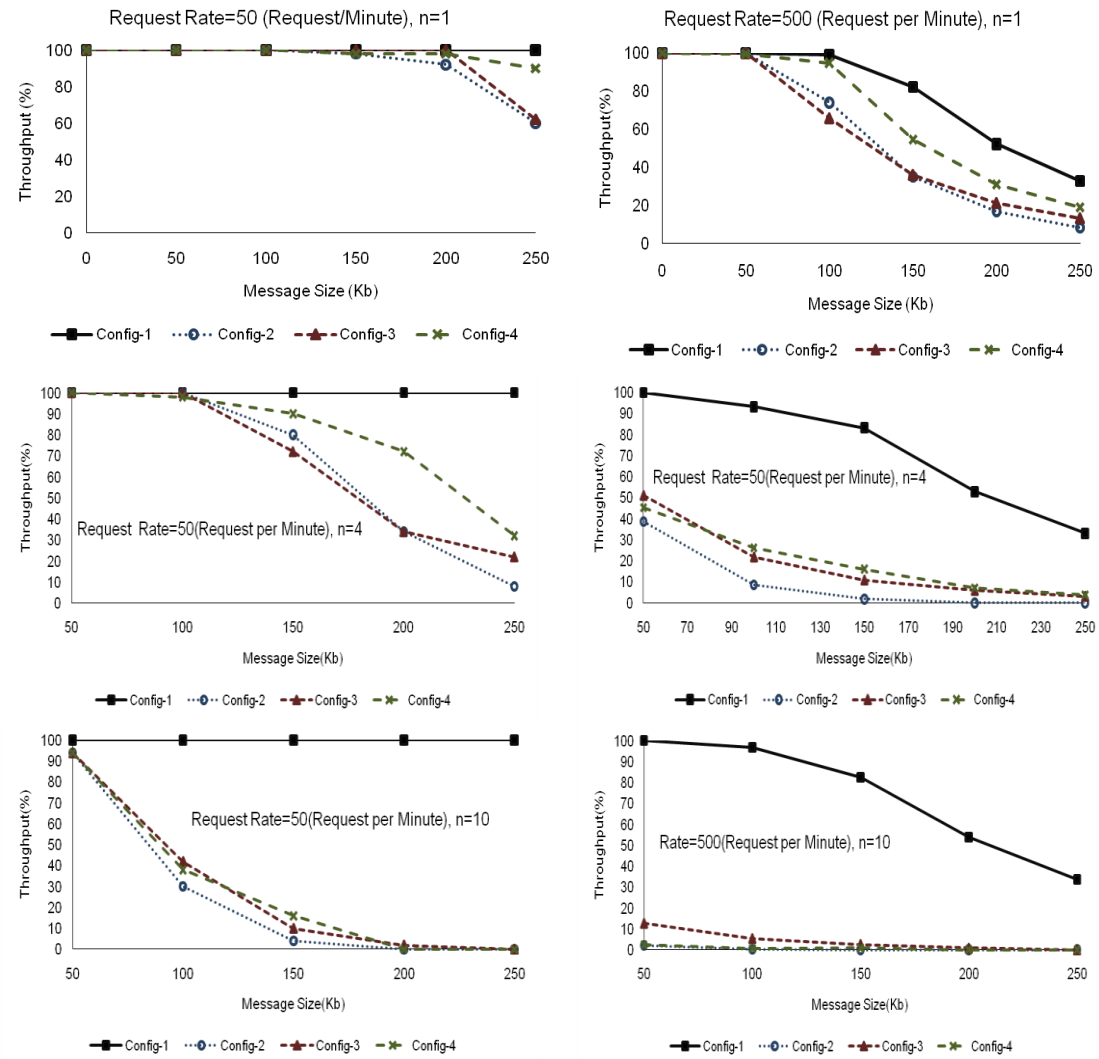


Figure 6.4: Boundary-condition Experiment, Throughput Comparison, Constant-Request-Rate, Variable-Message-Size

6.3.5 Experiment-5 (evaluating bandwidth-usage, variable-request-rate, constant-message-size)

This experiment was conducted according to Experiment-5 in boundary conditions specified in Section 4.8.1. Figure 6.5 shows the comparison of exchanged messages in the four configurations. There is totally no message passing in Config-1 while Config-2 and 3 use an internal message passing mechanism. Config-4 is the only case that uses external message passing and occupies the system bandwidth.

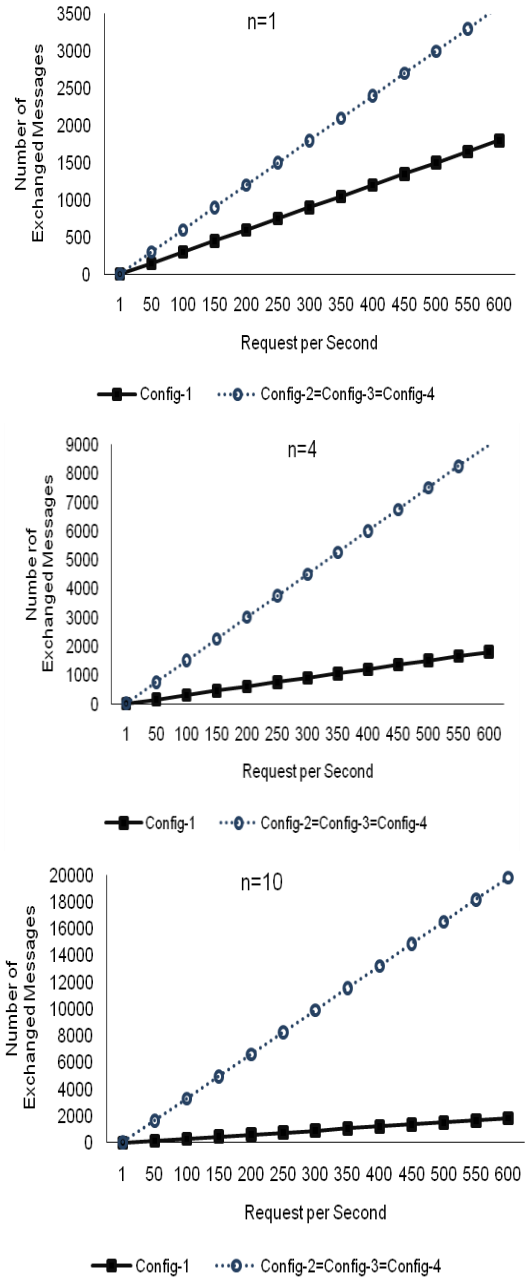


Figure 6.5: Boundary-condition Experiment, Comparison of Exchanged Messages Number

6.3.6 Boundary-condition Evaluation Discussion

The boundary-condition experiments showed response-time, throughput and message passing of the elementary elements implemented by WADE/JADE agents in different situations. The experiments 1 and 2 with variable-request-rate/constant-message-size proved that the Config-1 outweighed the other configurations in all cases. Moreover, there was a balance between Config-3 and Config-4; however, Config-4 was outweighed by Config-3 most of the time due to an external message passing. Config-2 was also outperformed by Config-3; although, they were located on the same machines. This case can be considered as an implementation constraint of WADE/JADE because there are limitations to access shared resources among fragments running on the same performer agent. The Experiment-3 and 4 with constant-request-rate/variable-message-size showed the results similar to the prior experiments except in Config-3 and Config-4. There was a balance between Config-3 and Config-4 so that by increasing the number of message passing among the fragments, Config-3 passed Config-4 and showed better response-time; although, in most of the cases, Config-4 showed better result due to better scalability. The equilibrium of request rate, message size, memory size, message queues and communication media resulted in such a behavior. Experiment-5 also showed the low number of exchanged messages in Config-2, 3 and 4 compared to Config-1.

6.4 Frequent-path and Proportional-fragment Evaluation Experiments

The main goals of this section are first, substantiating the frequent-path adaptability for the basic as well as the loan business processes through the experiments 6, 7, 8, 9 and 10; second, proving the proportional-fragment adaptability through the experiments 11, 12 and 13 only for the loan business process because it has adequate fragments to be executed on a number of machines. The basic If, While, and Flow

processes are not the subject of this experiment due to small number of fragments; third, comparing the decentralization methods with the Centralized fragmentation, which does not use any message passing in minimum distribution; fourth, comparing the FPD method with other decentralization methods in Min-Max distributions. The whole experiments are conducted based on the method described in Section 4.8.2.

After fragmenting business processes, the workflow fragments were encapsulated in workflow components and deployed in both architectures Type-1 and Type-2. The experiment results are shown in the sequel as well. The improvement percentage of different fragmentation models compared to the FPD is calculated based on Equations (6.1) and (6.2). The FPD response-time (RT) was mainly more than other methods; therefore, the FPD was improved by the other methods. Indeed, the improvement percentage of RT was calculated as shown in Equation (6.1). Throughput (TP) of each method has already been measured in terms of percentage; therefore, throughput improvement was considered by calculating the throughput difference of two methods in Equation (6.2).

$$improvement_{RT} = \frac{(RT_{fpd} - RT_{other\ method}) \times 100}{RT_{fpd}} \tag{6.1}$$

$$improvement_{TP} = TP_{other\ method} - TP_{fpd} \tag{6.2}$$

It is also stressed that in the following experiments the comparison of decentralization methods in maximum distribution was not fair due to different number of machines dedicated to them; however, this comparison showed how response-time and throughput behavior in the maximum distribution of fragments tended towards the Centralized method, which had the ideal response-time and throughput. Indeed, from now on, there will not be any discussion on the maximum

distribution until the fragment proportionality of the Loan-process fragments is studied.

6.4.1 If-process Experiment-6 to Experiment-10

This section focuses on decentralization of the basic If-process based on the Experiments 6, 7, 8, 9 and 10 introduced in Section 4.8.2. At first, the fragmentation of the process is discussed and then it will be used by the experiments in the following sections.

Fragmenting Basic If-process

If (or Switch) is a structured activity built on the well-known if-elseif-else format. According to (Active-Endpoints, 2011; OASIS, 2011), the If activity lets you choose exactly one execution path from among many such as If structure in common programming languages. Figure 6.6 (A) shows a basic If BPEL process in its tree representation including four levels. Figure 6.6(B-D) is also decentralizations of the basic If-process using the HPD method considering various granularity levels, g, where $0 \leq g < 3$. In addition, Figure 6.6(E-G) shows HIPD decentralization of the basic If BPEL process. The assumption for the HIPD is that path [(Sequence0 (receive, If (if-condition-branch (assign1)), reply))] is a frequent path. Indeed, the client was setup so that 70% and 30% of its requests resulted in executing the If and Else branches to make the mentioned path a frequent path. The HPD method is idempotent for this frequent path due to the independency of fragmentation from the executed path; however, the detection of different frequent paths by the HIPD resulted in new fragments. In this case, HPD0 is equal to the Centralized model and HPD2 is analogous to the FPD fragmentation.

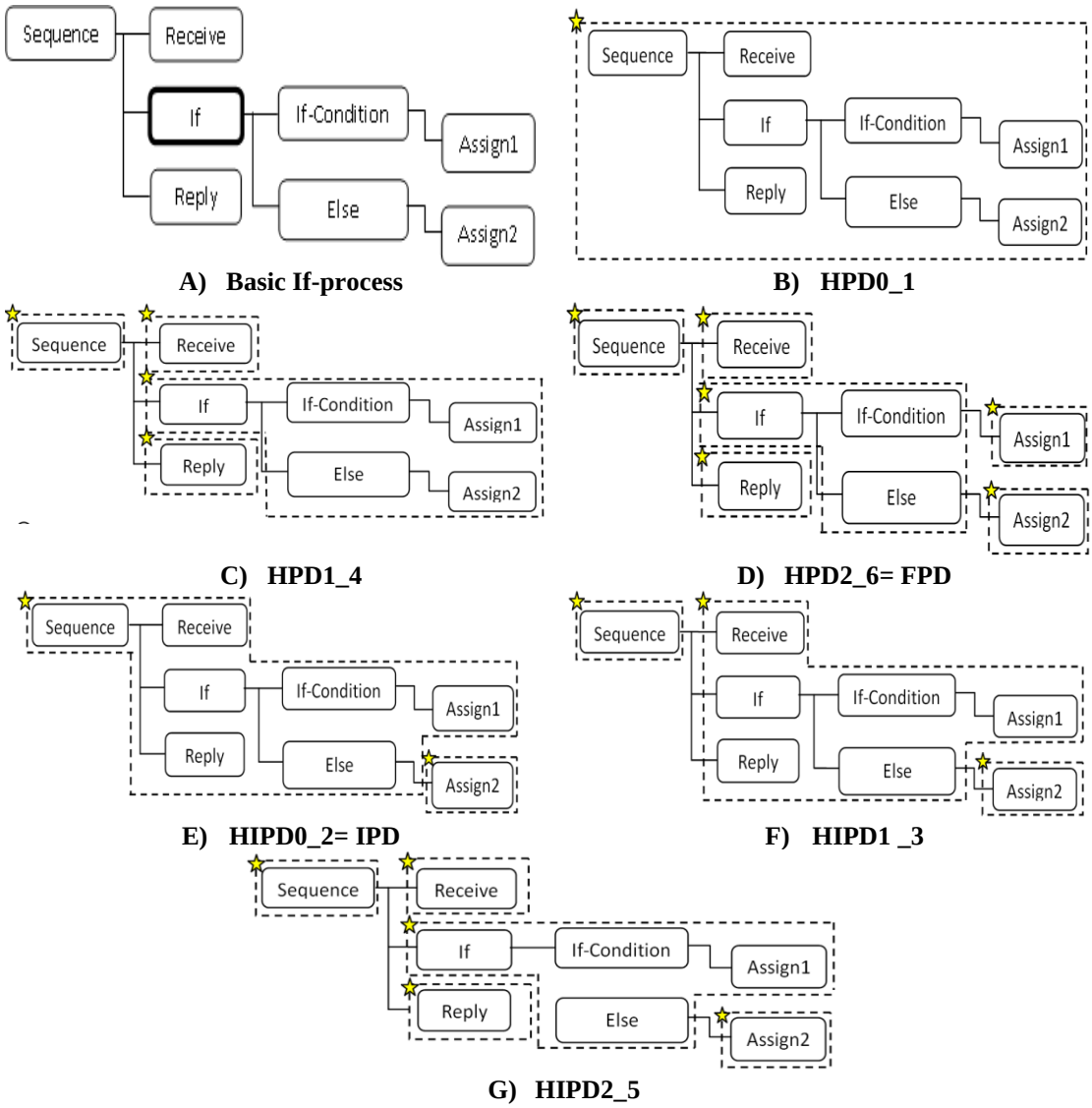


Figure 6.6: If-process Decentralization Patterns

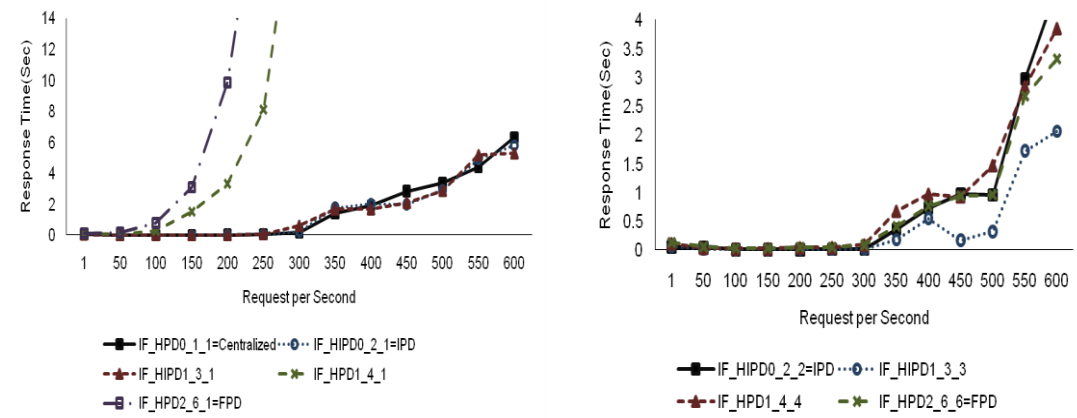
Consequently, the following experiments were conducted using the various versions of the If-process including the Centralized version with one fragment, If_HIPD0_2_1 with two fragments, If_HIPD1_3_1 with three fragments and the FPD version with six fragments.

Experiment-6 (frequent-path adaptability, response-time evaluation, variable-request-rate/constant-message-size)

This experiment was conducted according to Experiment-6 specified in Section 4.8.2. Figure 6.7 and Figure 6.8 show the comparison of the basic If-process response-time in Min-Max distributions, running on both architectures Type-1 and Type-2.

Type-1: Figure 6.9 shows the improvement trend of the FPD by other decentralization methods**.** It is clear that the Centralized and HIPD fragments improved the FPD. Table 6.1 also depicts that the average improvement percent of the FPD was around 98.92%, 98.85% and 98.81% by the Centralized, IPD and HIPD1_3_1 fragments. In comparison, HPD1_4_1 improved the average response-time about 57.01%.

Type-2: Figure 6.9 shows the improvement trend of the FPD by other decentralization methods**.** It is clear that the Centralized and HIPD fragments improved the FPD. Table 6.1 also depicts that the average improvement percent of the FPD was around 64.24%, 63.77% and 60.98% by the Centralized, IPD and HIPD1_3_1. In comparison, HPD1_4_1 improved the average response-time about 24.58%.



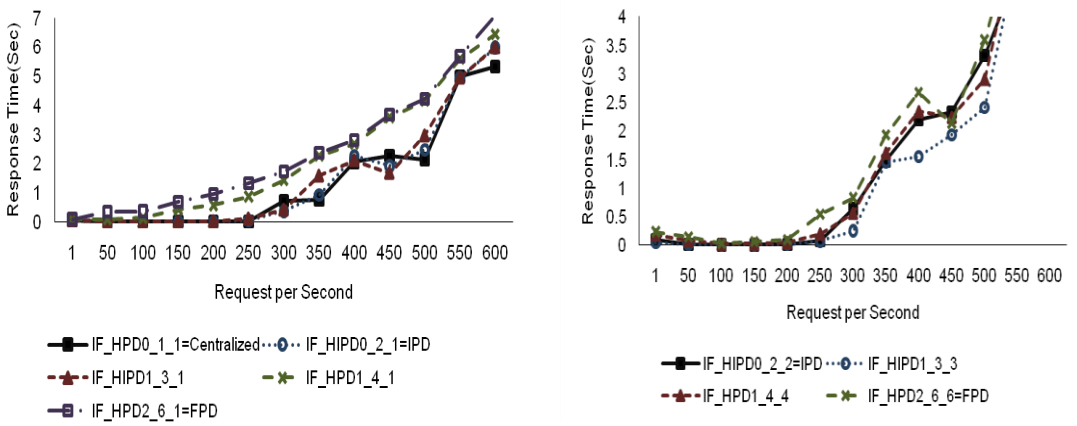
Minimum Distribution **Maximum Distribution**

Figure 6.7: If-process Response-time Comparison, Type-1, Variable-Request-Rate, Constant-Message-Size, Min-Max distributions

The experiments showed that the number of fragments and naturally exchanged messages had effective impact on response-time. The If-process HIPD results were considerably similar to the Centralized due to encapsulating the frequent paths; however, it should be noted that the HIPD fragments were decentralized. The HPD

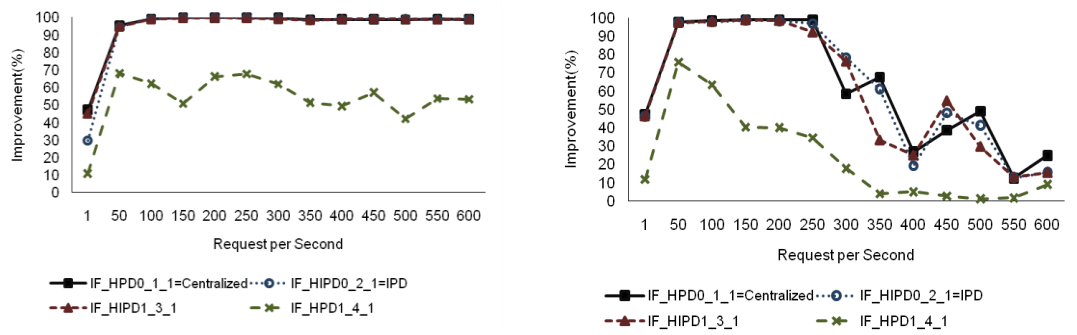
method did not consider the relevancy of activities; however, the lessened number of exchanged messages in HPD1_4_1 improved the FPD.

This experiment also showed calling various versions of the If-process in different rates, in absence of message size, resulted in improving the response-time of the If-process FPD version when the fragments were executed by both architectures Type-1 and Type-2. The response-time improvement in architecture Type-1 was more than Type-2, which substantiated the resiliency of Type-2 against Type-1.



Minimum Distribution **Maximum Distribution**

Figure 6.8: If Response-time Comparison, Type-2, Variable-Request-Rate, Constant-Message-Size, Min-Max distributions



Type-1 **Type-2**

Figure 6.9: Type-1 and Type-2, If-process Response-time Improvement Comparison

Table 6.1: Average Response-time Improvement of If-process Fragments Compared to FPD

	IF_HPD0_1_1 (Centralized)	IF_HIPD0_2_1 (IPD)	IF_HIPD1_3_1	IF_HPD1_4_1
Type-1	98.92 %	98.85 %	98.81 %	57.01 %
Type-2	64.24 %	63.77 %	60.98 %	24.58 %

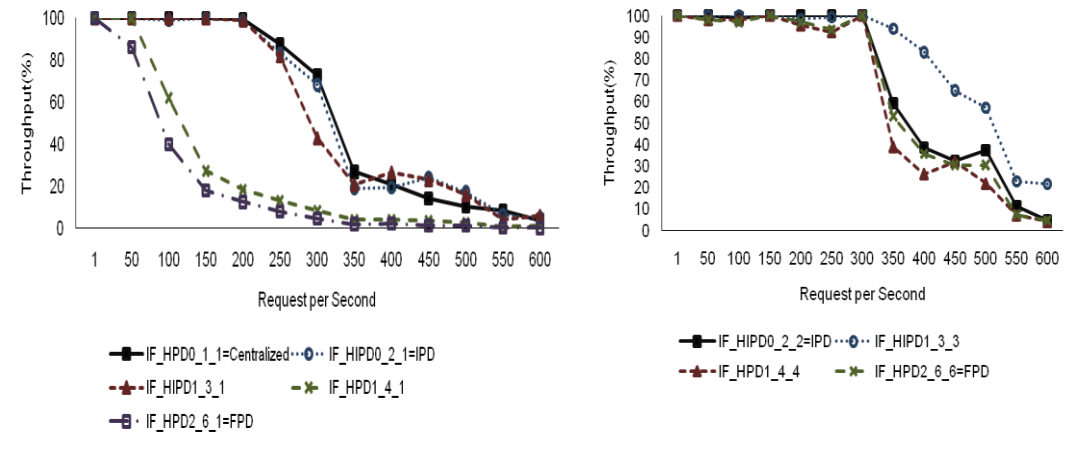
Experiment-7 (frequent-path adaptability, throughput evaluation, variable-request-rate/constant-message-size)

This experiment was conducted according to Experiment-7 specified in Section 4.8.2. Figure 6.10 and Figure 6.11 show the comparison of throughput in Min-Max distributions in both architectures Type-1 and Type-2.

Type-1: Figure 6.10 shows the improvement trend of the FPD by other decentralization methods. It is clear that the Centralized and HIPD fragments improved the FPD. Table 6.2 also depicts that the average throughput of the FPD was improved around 38.97%, 38.66% and 37.02%. In comparison, HPD1_4_1 improved the average throughput around 5.76%.

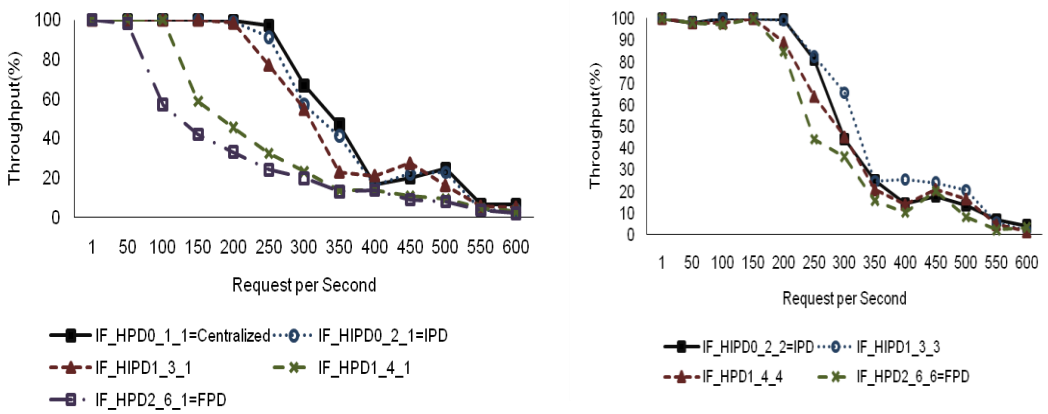
Type-2: Figure 6.10 shows the improvement trend of the FPD by other decentralization methods. It is clear that the Centralized and HIPD fragments improved the FPD. Table 6.2 also depicts that the average throughput of the FPD was improved around 27.85%, 30.15% and 25.50%. In comparison, HPD1_4_1 improved the average throughput around 7.65%.

This experiment also showed calling various versions of the If-process in different rates, in absence of message size, resulted in improving the throughput of the If-process FPD version when the fragments were executed by both architectures Type-1 and Type-2. The HIPD fragments of the If-process decreased the message passing by encapsulating closely interrelated activities. The If-process greatly improved the total process throughput. The HPD fragments in the If-process were not as good as the HIPD fragments; nonetheless, they were better than the FPD. Throughput improvement in architecture Type-1 was more than Type-2, which substantiated the resiliency of Type-2 against Type-1.



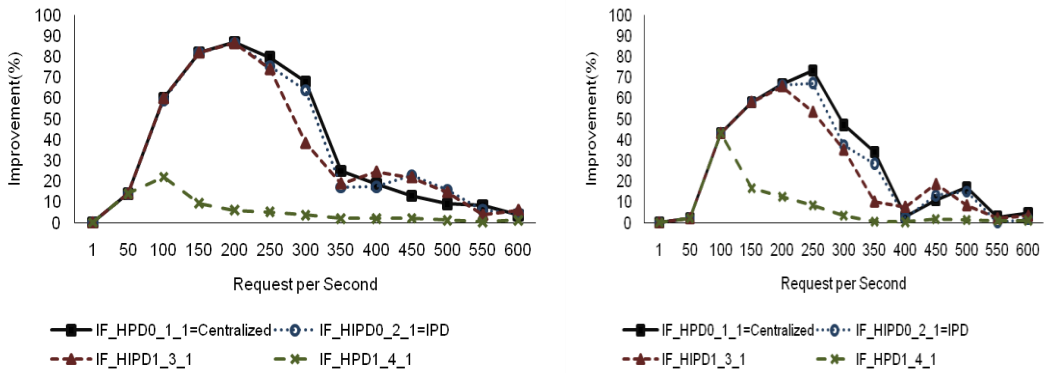
Minimum Distribution **Maximum Distribution**

Figure 6.10: Type-1 If-process Throughput Comparison in Min-Max distributions



Minimum Distribution **Maximum Distribution**

Figure 6.11: Type-2 If-process Throughput Comparison in Min-Max distributions



Type-1 **Type-2**

Figure 6.12: Type-1 and Type-2, Throughput Improvement Comparison

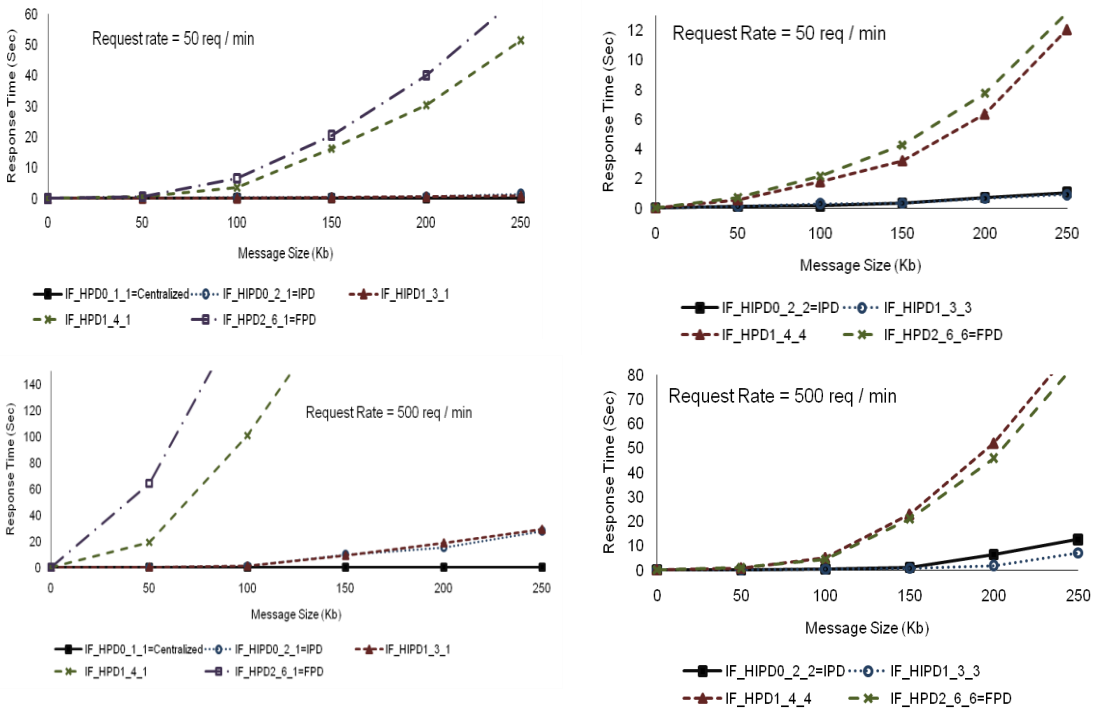
Table 6.2: Average Throughput Improvement of If-process Fragments Compared to FPD

	IF_HPD0_1_1 (Centralized)	IF_HIPD0_2_1 (IPD)	IF_HIPD1_3_1	IF_HPD1_4_1
Type-1	38.97 %	38.66 %	37.02 %	5.76 %
Type-2	27.85 %	30.15 %	25.50 %	7.65 %

Experiment-8 (frequent-path adaptability, response-time evaluation, constant-request-rate/variable-message-size)

This experiment was conducted according to Experiment-8 specified in Section 4.8.2. The experiment was performed for Min-Max distributions for both architectures Type-1 and Type-2.

Type-1: Figure 6.13 shows response-time behavior of the If-process decentralized versions in Min-Max distributions running on the architecture Type-1. In addition, Figure 6.15 shows the improvement trend of the FPD response-time by other methods. According to Table 6.3, the average improvement of response-time in request rate 500 by the Centralized, IPD and HIPD1_3_1 was 97.97%, 98.06% and 97.96%, respectively. In comparison, HPD1_4_1 improved the average response-time about 49.79%.

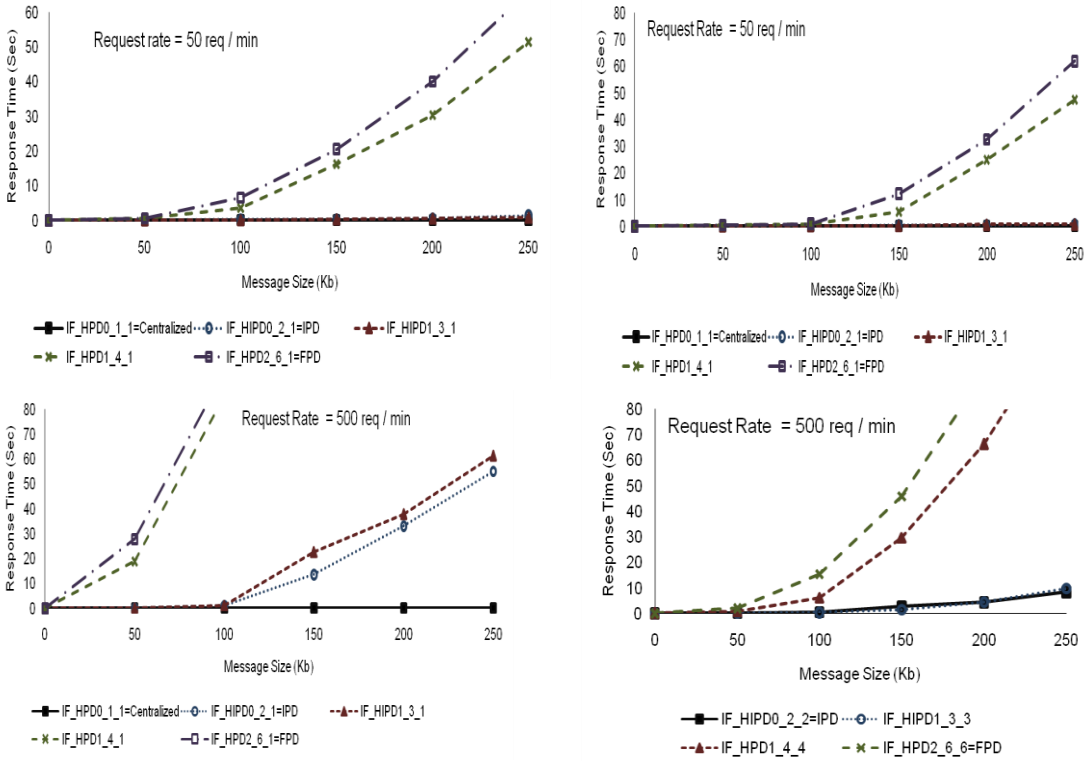


Minimum Distribution **Maximum Distribution**

Figure 6.13: If-process Response-time Comparison, Type-1, Constant-Request-Rate, Variable-Message-Size, Min-Max distributions

Type-2: Figure 6.14 shows response-time behavior of the If decentralized versions in Min-Max distributions running on the architecture Type-1. In addition, Figure 6.15

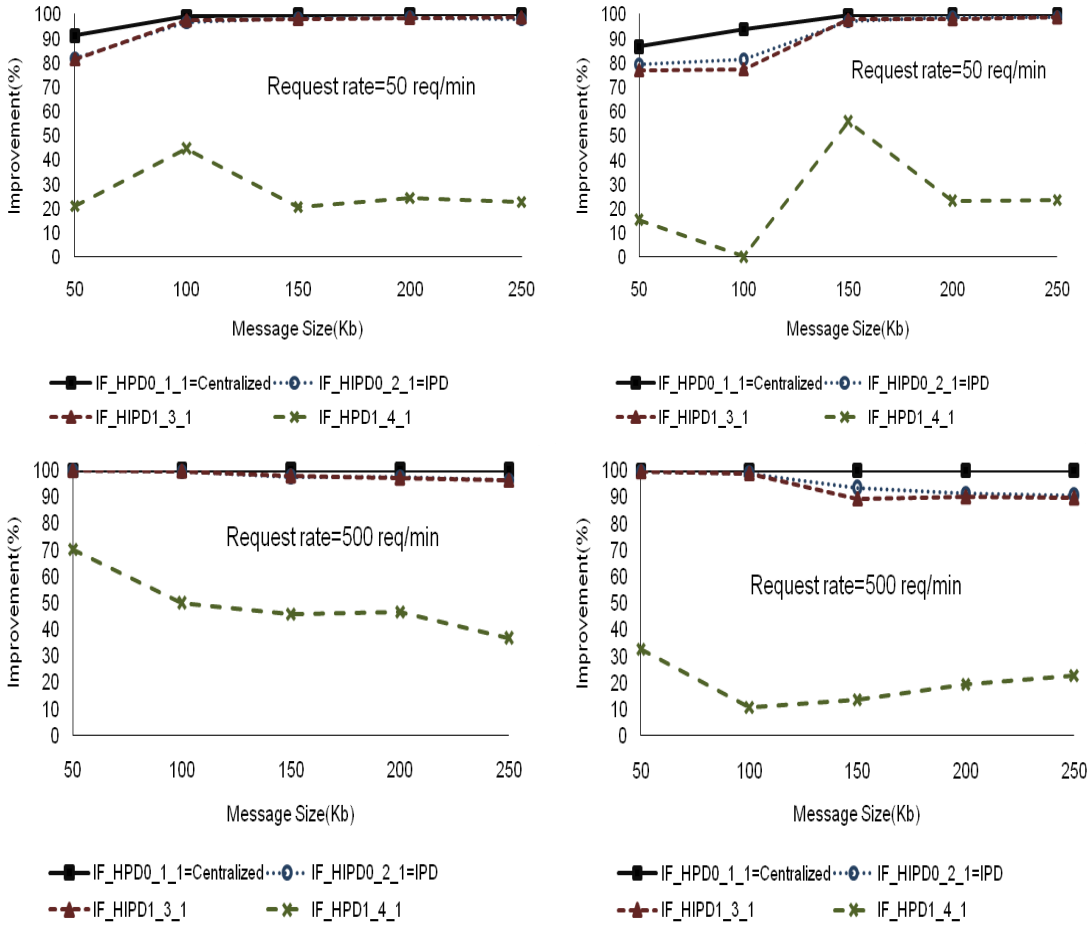
shows the improvement trend of the FPD response-time by other methods. According to Table 6.3, the average improvement of response-time in request rate 500 by the Centralized, IPD and HIPD1_3_1 was 99.93%, 94.73% and 93.42%, respectively. In comparison, HPD1_4_1 improved the average response-time about 19.71%.



Minimum Distribution **Maximum Distribution**

Figure 6.14: If-process Response-time Comparison, Type-2, Constant-Request-Rate, Variable-Message-Size, Min-Max distributions

In both architectures, the FPD response-time was improved; particularly, when the request rate and the size of messages were increased. The trend of improvement was directly proportional to the number of fragments and the size of messages exchanged among them. Consequently, the lower number of fragments and the size of exchanged messages among the If-process activities resulted in better response-time of the process. The frequent-path adaptability of the HIPD was shown in high request rate and large message sizes in that its response-time was very close to the Centralized.



Type-1 **Type-2**

Figure 6.15: Type-1 and Type-2, If-process Response-time Improvement with Request Rates 50 and 500

Table 6.3: Average Response-time Improvement of If Fragments Compared to FPD

	Req / Min	IF_HPD0_1_1 (Centralized)	IF_HIPD0_2_1 (IPD)	IF_HIPD1_3_1	IF_HPD1_4_1
Response-time	50	97.97%	94.59%	94.79%	26.68%
	500	99.97%	98.06%	97.96%	49.79%
Type-1					
	Req / Min	IF_HPD0_1_1 (Centralized)	IF_HIPD0_2_1 (IPD)	IF_HIPD1_3_1	IF_HPD1_4_1
Response-time	50	95.95%	91.05%	89.74%	23.66%
	500	99.93%	94.73%	93.42%	19.71%
Type-2					

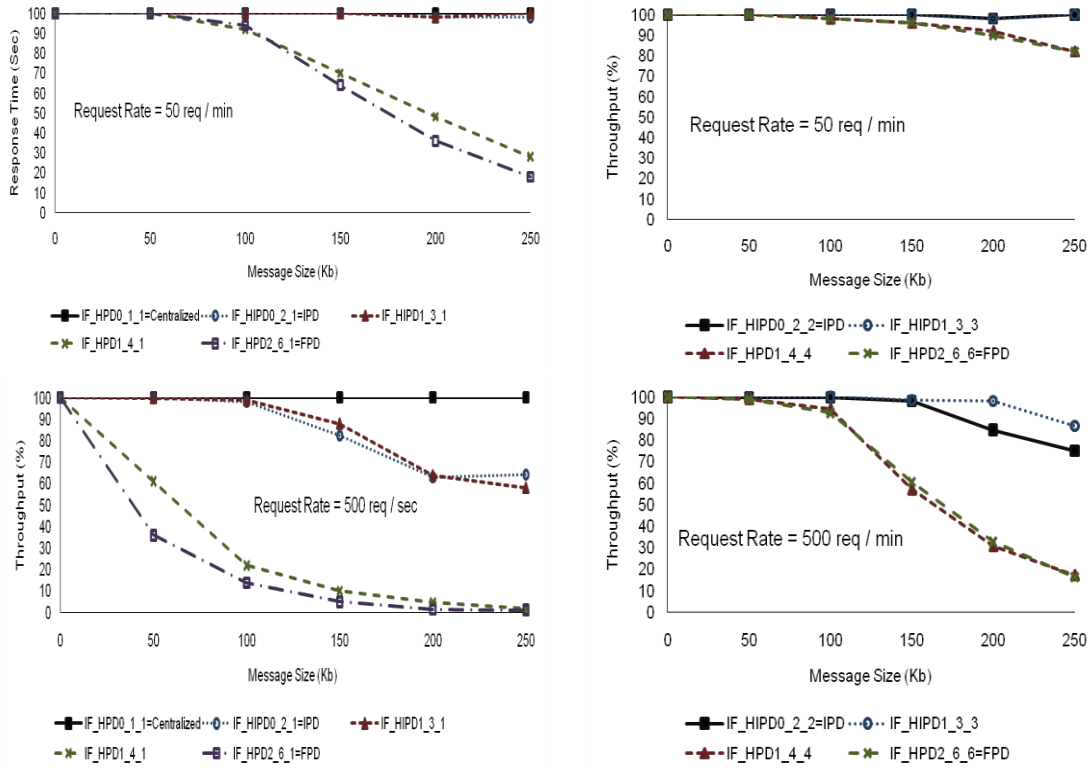
Experiment-9 (frequent-path adaptability, evaluating throughput, constant-request-rate, variable-message-size)

This experiment was conducted according to the Experiment-9 specified in Section 4.8.2. The experiment was performed for Min-Max distributions in both architectures Type-1 and Type-2.

Type-1: Figure 6.16 shows the behavior of decentralized versions of the If-process in Min-Max distributions running on the architecture Type-1. In addition, Figure 6.18 shows the percent of throughput improvement for the decentralization methods compared to the FPD fragments. As shown in Table 6.4, in request rate 500, the average improvement of throughput by the Centralized, IPD and HIPD1_3_1 was 88.72%, 70.20% and 70.28%. In comparison, HID1_4_1 improved the FPD around 8.56%.

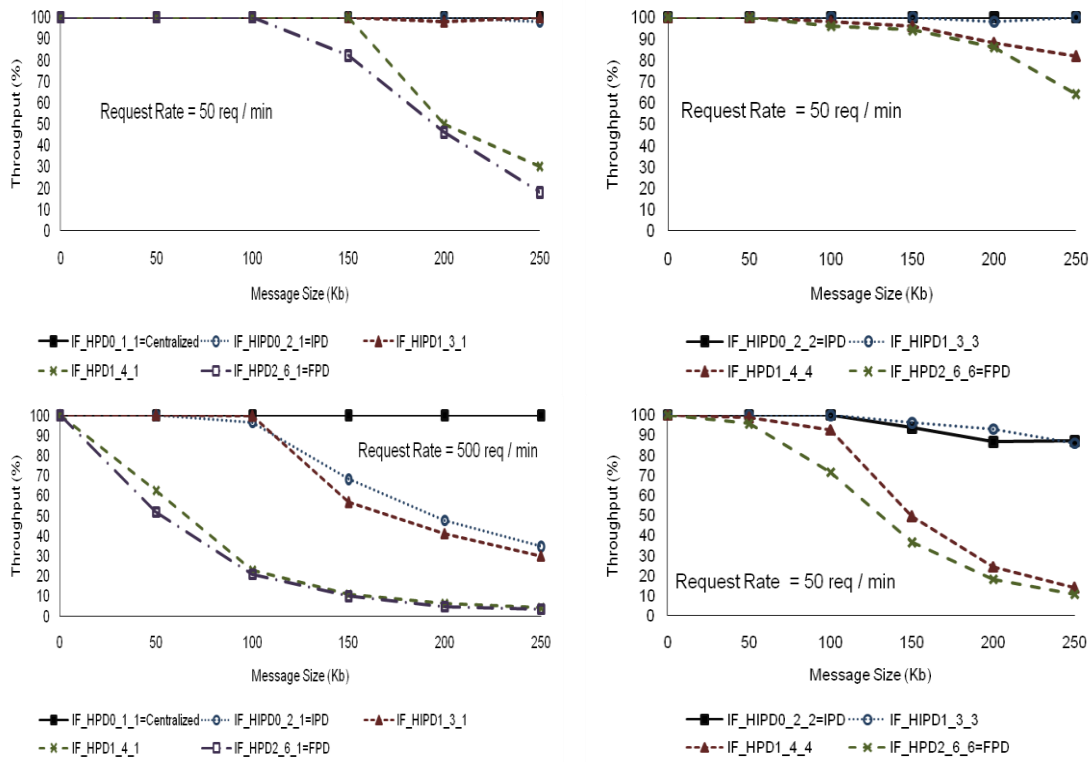
Type-2: Figure 6.17 show the behavior of decentralized versions of the If-process in Min-Max distributions running on the architecture Type-2. In addition, Figure 6.18 shows the percent of throughput improvement for the decentralization methods compared to the FPD fragments. As shown in Table 6.4, in request rate 500, the average improvement of throughput by the Centralized, IPD and HIPD1_3_1 was 81.88%, 51.28% and 47.28%. In comparison, HID1_4_1 improved the FPD around 3.16%.

In both architectures, the FPD throughput was improved; in particular, when the request rate and size of messages were increased. In the HPD fragments, the trend of improvement was directly proportional to the number of fragments and the size of messages exchanged among them. Consequently, the lower number of fragments and the size of exchanged messages among the Loan-process activities resulted in better response-time for the process. However, the If HIPD fragments were better from throughput view, due to encapsulating frequent paths and eliminating message passing.



Minimum Distribution **Maximum Distribution**

Figure 6.16: If-process Throughput Comparison, Type-1, Constant-Request-Rate, Variable-Message-Size, Min-Max distributions



Type-1 **Type-2**

Figure 6.17: If-process Throughput Comparison, Type-2, Constant-Request-Rate, Variable-Message-Size, Min-Max distributions

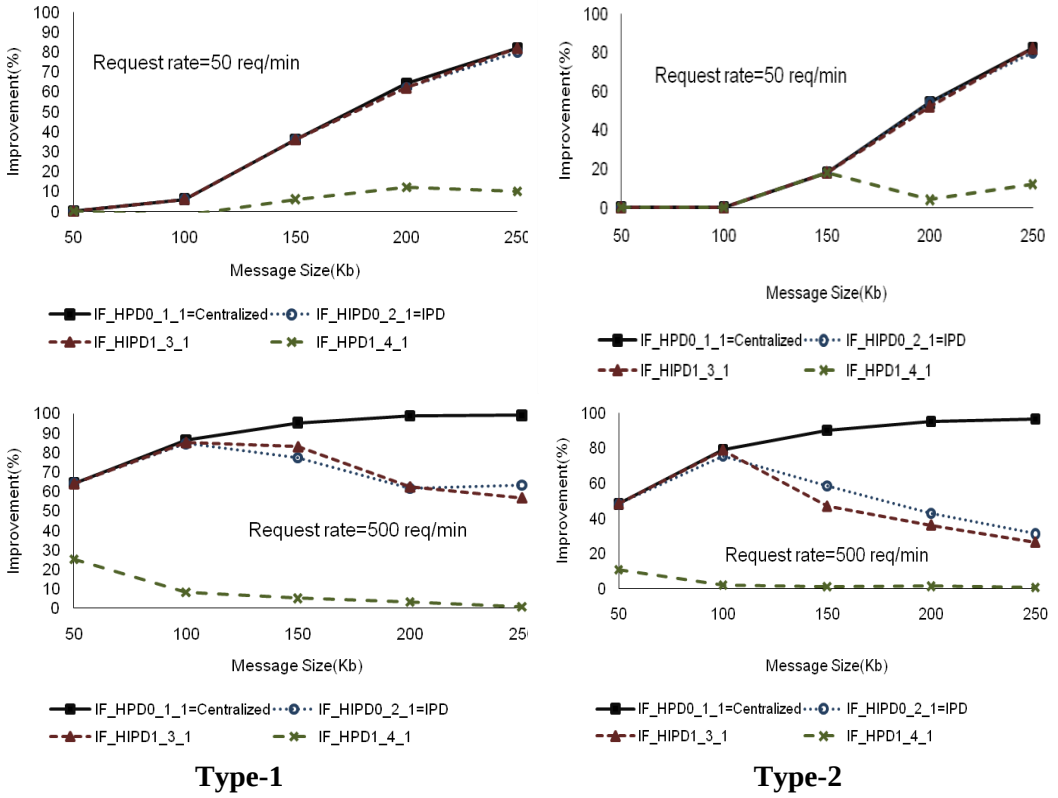


Figure 6.18: Type-1 and Type-2, If-process Throughput Improvement with Request Rates 50 and 500

Table 6.4: Average Throughput Improvement of If Fragments Compared to FPD

	Req / Min	IF_HPD0_1_1 (Centralized)	IF_HIPD0_2_1 (IPD)	IF_HIPD1_3_1	IF_HPD1_4_1
Throughput	**50**	37.60%	36.80%	37.20%	5.20%
	500	88.72%	70.20%	70.28%	8.56%
Type-1					
	Req / Min	**IF_HPD0_1_1 (Centralized)**	**IF_HIPD0_2_1 (IPD)**	**IF_HIPD1_3_1**	**IF_HPD1_4_1**
Throughput	**50**	30.80%	30.40%	30.40%	6.80%
	500	81.88%	51.28%	47.28%	3.16%
Type-2					

Experiment-10 (frequent-path adaptability, evaluating bandwidth-usage, variable-request-rate/constant-message-size)

This experiment was conducted according to Experiment-10 specified in Section 4.8.2. Figure 6.19 shows the number of exchanged messages by the fragments of different decentralization methods. The Centralized and HIPD methods were considerably similar, due to the less number of fragments and relevancy of activities.

The FPD and HPD versions of the If-process were the same due to the higher number of fragments.

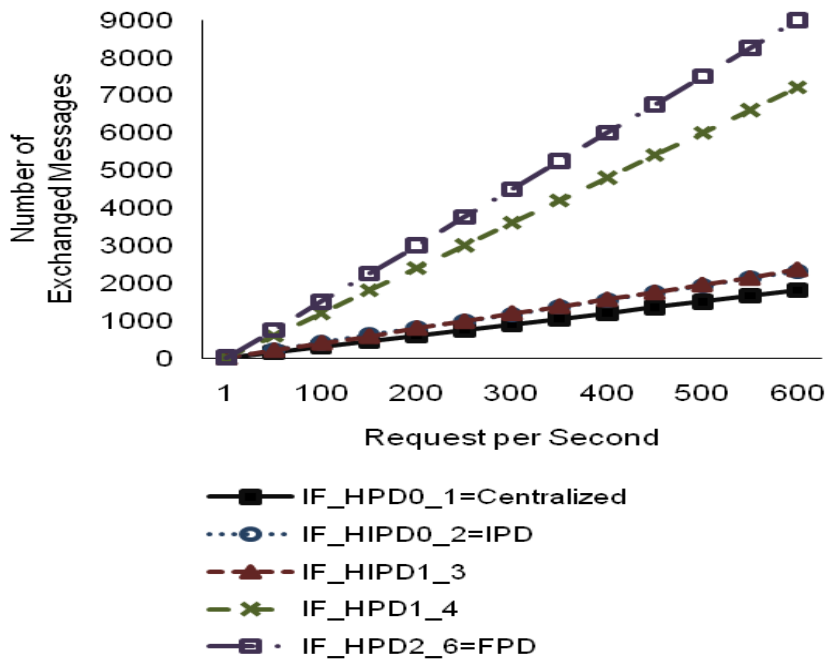


Figure 6.19: Exchanged Message Comparison of If-process Fragments

6.4.2 While-process Experiment-6 to Experiment-10

This section focuses on the decentralization of the basic While-process based on the Experiment 6, 7, 8, 9 and 10 introduced in Section 4.8.2. At first, the fragmentation of the process is discussed and then it will be used by the experiments in the following sections.

Fragmenting Basic While-process

In BPEL specification, there are three loop structures among which While loop is selected to study how the decentralization policies work for this activity. The While activity repeats the enclosed statement block until the conditional statement evaluates to False (Active-Endpoints, 2011; OASIS, 2011). Figure 6.20(A) shows a basic While BPEL process. Figure 6.20(B-D) is also the decentralizations of the basic While-process using the HPD method based on different levels of granularity.

Furthermore, Figure 6.20(E) shows HIPD1 decentralization of the While-process. The assumption for the HIPD was that path [(sequence (receive, while (assign1), reply))] was frequent. Indeed, the process was setup so that for each process invocation, the while loop was repeated ten times. In this case, HPD0 is equal to the Centralized and HIPD0. The fragments of HPD2 and HIPD2 are equal and fully decentralized as well.

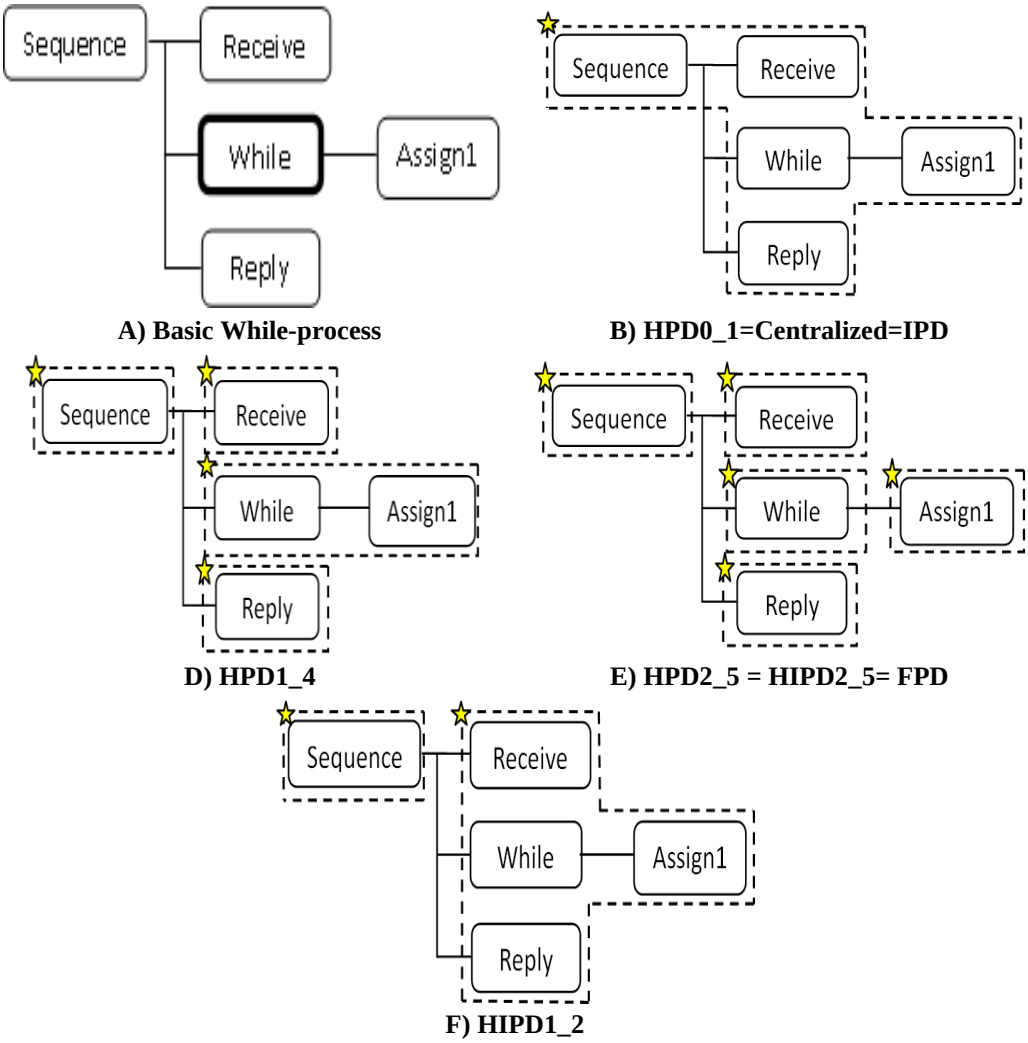


Figure 6.20: While-process Decentralization Patterns

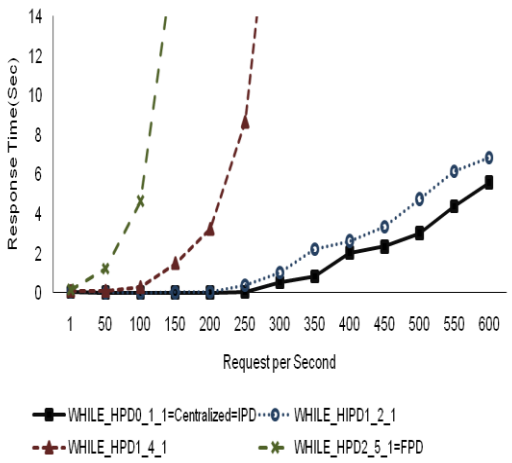
Consequently, the following experiments were conducted using the various versions of the While-process including the Centralized model, which in this case is analogous to the IPD with one fragment, WHILE_HIPD1_2 with two fragments, WHILE_HPD1_4 with four fragments and the FPD method with five fragments.

Experiment-6 (frequent-path adaptability, response-time evaluation, variable-request-rate/constant-message-size)

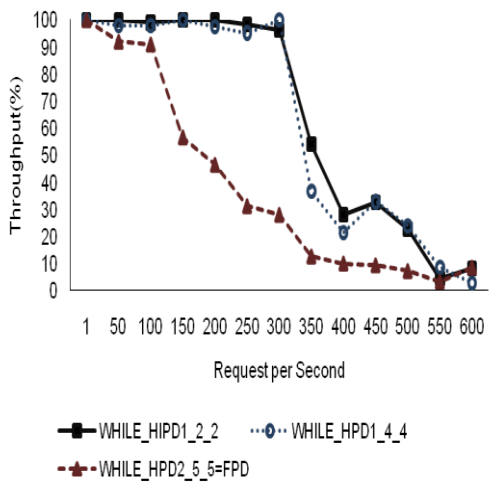
This experiment was conducted according to the Experiment-6 specified in Section 4.8.2. The experiment was performed for Min-Max distributions of the While fragments on both architectures Type-1 and Type-2.

Type-1: Figure 6.21 shows response-time behavior of the decentralized versions of the While-process in Min-Max distributions running on the architecture Type-1. In addition, Table 6.5 also shows that the FPD was improved by the Centralized=IPD and HIPD1_2_1 around 99.72%, 99.53%. In comparison, HPD1_4_1 improved the average response-time of the FPD about 86.78%.

Type-2: Figure 6.22 shows response-time behavior of decentralized versions of the While-process in Min-Max distributions running on the architecture Type-2. In addition, Table 6.5 also shows that the FPD was improved by the Centralized=IPD and HIPD1_2_1 around 87.20% and 83.25%. In comparison, HPD1_4_1 improved the average response-time of the FPD about 74.54%.



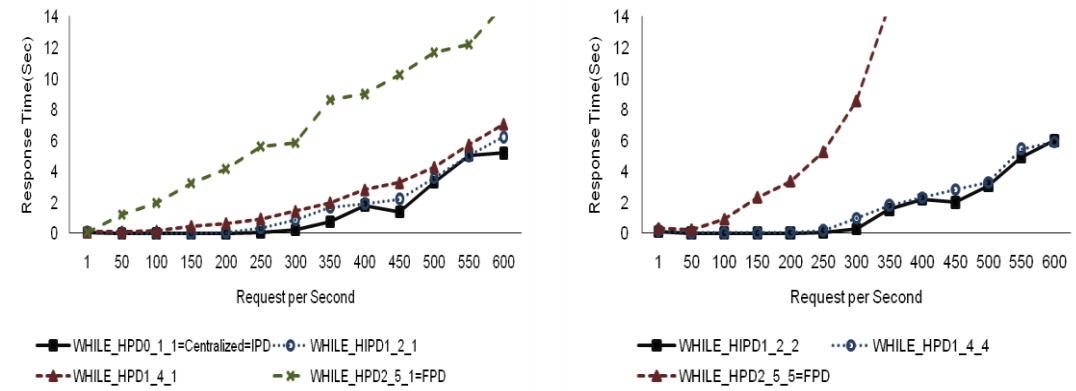
Minimum Distribution



Maximum Distribution

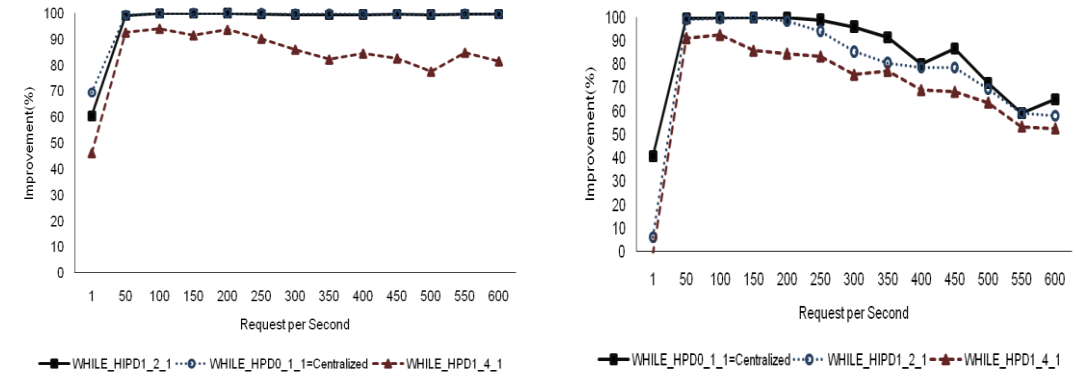
Figure 6.21: While-process Response-time Comparison, Type-1, Variable-Request-Rate, Constant-Message-Size, Min-Max distributions

This experiment showed that the number of fragments and naturally exchanged messages had effective impact on response-time. The result of the While HIPD fragments was very similar to the Centralized even in high request rate despite of high communication among the While and its enclosed activities. It was due to the relevancy of activities inside fragments. The HPD method did not consider the relevancy of activities; however, the lessened number of exchanged messages in HPD1_5_1 improved the FPD.



Minimum Distribution **Maximum Distribution**

Figure 6.22: While-process Response-time Comparison, Type-2, Variable-Request-Rate, Constant-Message-Size, Min-Max distributions



Type-1 **Type-2**

Figure 6.23: Type-1 and Type-2, While-process Response-time Improvement Comparison

The experiment also showed calling various versions of the While-process in different rates, in absence of message size, resulted in improving the While FPD version when the fragments were executed by both architectures Type-1 and Type-2.

Response-time improvement in architecture Type-1 was more than Type-2, which substantiated the resiliency of Type-2 compared to Type-1 in executing different fragment Types. The reason was due to allocating private memory quota for the performer agents in Type-2 instead of a shared space for all of them in Type-1.

Table 6.5: Average Response-time Improvement of While Fragments Compared to FPD

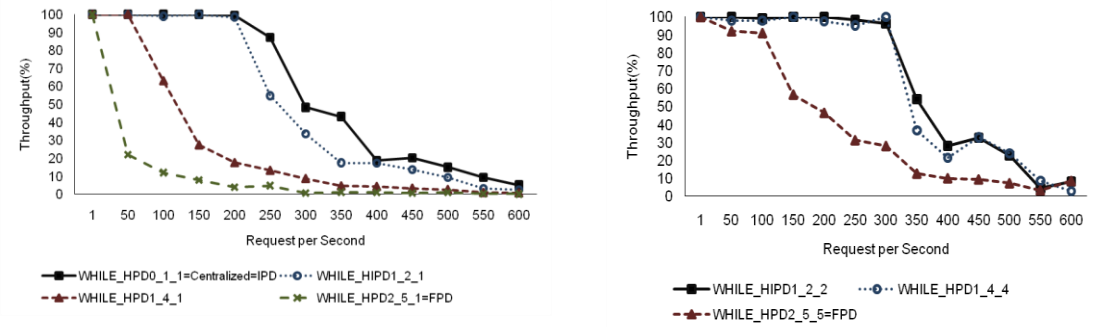
	WHILE_HPD0_1_1 (Centralized=IPD)	WHILE_HIPD1_2_1	WHILE_HPD1_4_1
Type-1	99.72%	99.53%	86.78%
Type-2	87.20%	83.25%	74.54%

Experiment-7 (frequent-path adaptability, throughput evaluation, variable-request-rate/constant-message-size)

This experiment was conducted according to the Experiment-7 specified in Section 4.8.2. The experiment was performed on both architectures Type-1 and Type-2 for Min-Max distributions.

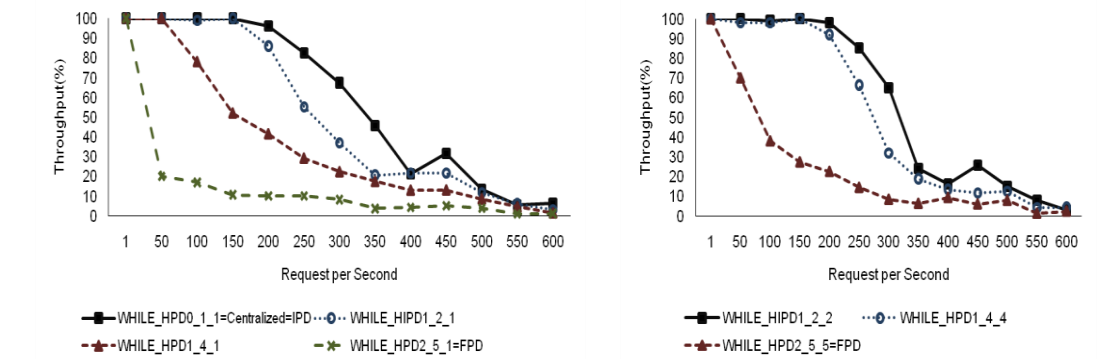
Type-1: Figure 6.24 shows the improvement trend of the FPD by the other fragments of the basic While-process**.** It is obvious that the Centralized, HIPD1_2_1 and HPD1_4_1 fragments improved the FPD throughput according to Figure 6.26. Table 6.6 also depicts that the average throughput of the FPD was improved around 45.46%, 37.98% and 14.63%, respectively.

Type-2: Figure 6.25 shows the improvement trend of the FPD by the other While fragmentation models**.** It is obvious that the Centralized, HIPD1_2_1 and HPD1_4_1 fragments improved the FPD throughput according to Figure 6.26. Table 6.6 also depicts that the average response-time of the FPD was improved around 47.89%, 38.94% and 23.85%, respectively.



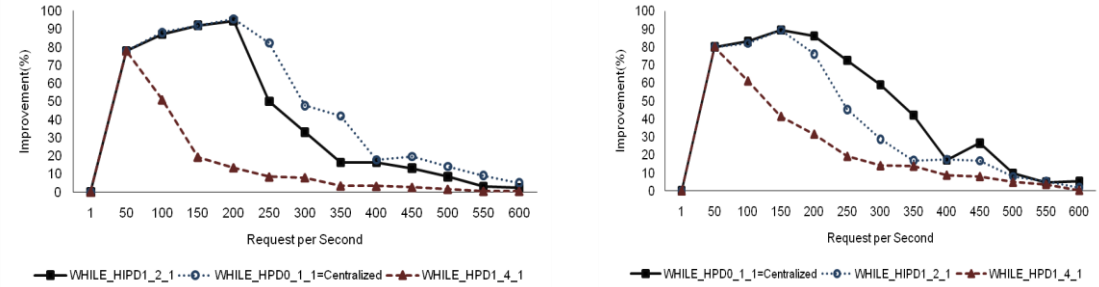
Minimum Distribution **Maximum Distribution**

Figure 6.24: While-process Throughput Comparison, Type-1 , Variable-Request-Rate, Constant-Message-Size, Min-Max distributions



Minimum Distribution **Maximum Distribution**

Figure 6.25: While-process Throughput Comparison, Type-2, Variable-Request-Rate, Constant-Message-Size, Min-Max distributions



Type-1 **Type-2**

Figure 6.26: Type-1 and Type-2, While-process Throughput Improvement Comparison

This experiment showed that the number of fragments and naturally exchanged messages impacted the throughput effectively. The high communication among the activities in the While-process had less impact on the throughput of the While HIPD fragments due to the relevancy of activities inside the fragments. The HPD method did not consider the relevancy of activities; however, the lessened number of fragments and exchanged messages in HPD1_5_1 improved the FPD.

The experiment also showed calling various versions of the While-process in different request rates in absence of message size, resulted in improving the While FPD version when the fragments were executed by both architectures Type-1 and Type-2. Throughput improvement in architecture Type-1 was more than Type-2, which substantiated the resiliency of Type-2 compared to Type-1 in executing different fragment Types.

Table 6.6: Average Throughput Improvement of While Fragments Compared to FPD

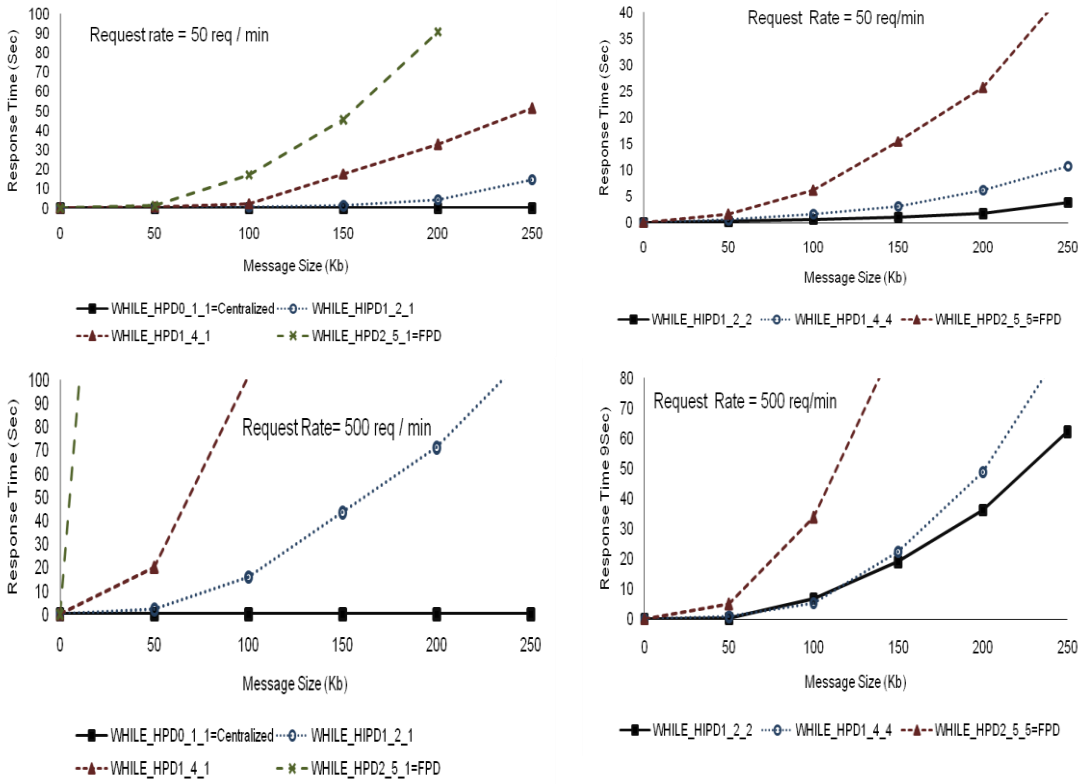
	WHILE_HPD0_1_1 (Centralized)	WHILE_HIPD1_2_1	WHILE_HPD1_4_1
Type-1	45.46%	37.98%	14.63%
Type-2	47.89%	38.94%	23.85%

Experiment-8 (frequent-path adaptability, response-time evaluation, constant-request-rate/variable-message-size)

This experiment was conducted according to the Experiment-8 specified in Section 4.8.2. It was performed for Min-Max distributions on both architectures Type-1 and Type-2.

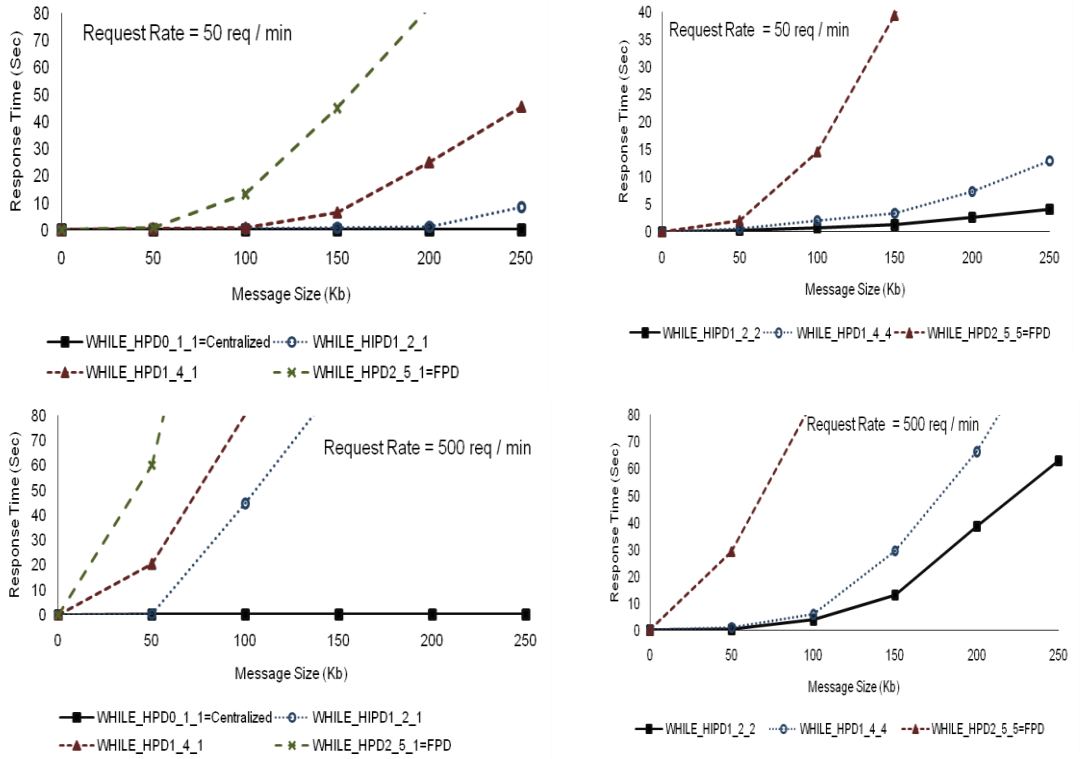
Type-1: Figure 6.27, shows response-time behavior of the While fragments in Min-Max distributions running on the architecture Type-1. Figure 6.29 shows the improvement trend of the FPD response-time by the other While fragments. According to Table 6.7, the average improvement of response-time in request rate 500 by the Centralized and HIPD1_2_1 was 99.99% and 96.52%. In comparison, HPD1_4_1 improved the average response-time about 83.61%.

Type-2: Figure 6.28, shows response-time behavior of the While fragments in Min-Max distributions running on the architecture Type-2. Figure 6.29 shows the FPD improvement trend of response-time by the other While fragments. According to Table 6.7, the average improvement of response-time in request rate 500 by the Centralized and HIPD1_2_1 was 99.97%, 85.83%. In comparison, HPD1_4_1 improved the average response-time about 65.45%.



Minimum distribution **Maximum Distribution**

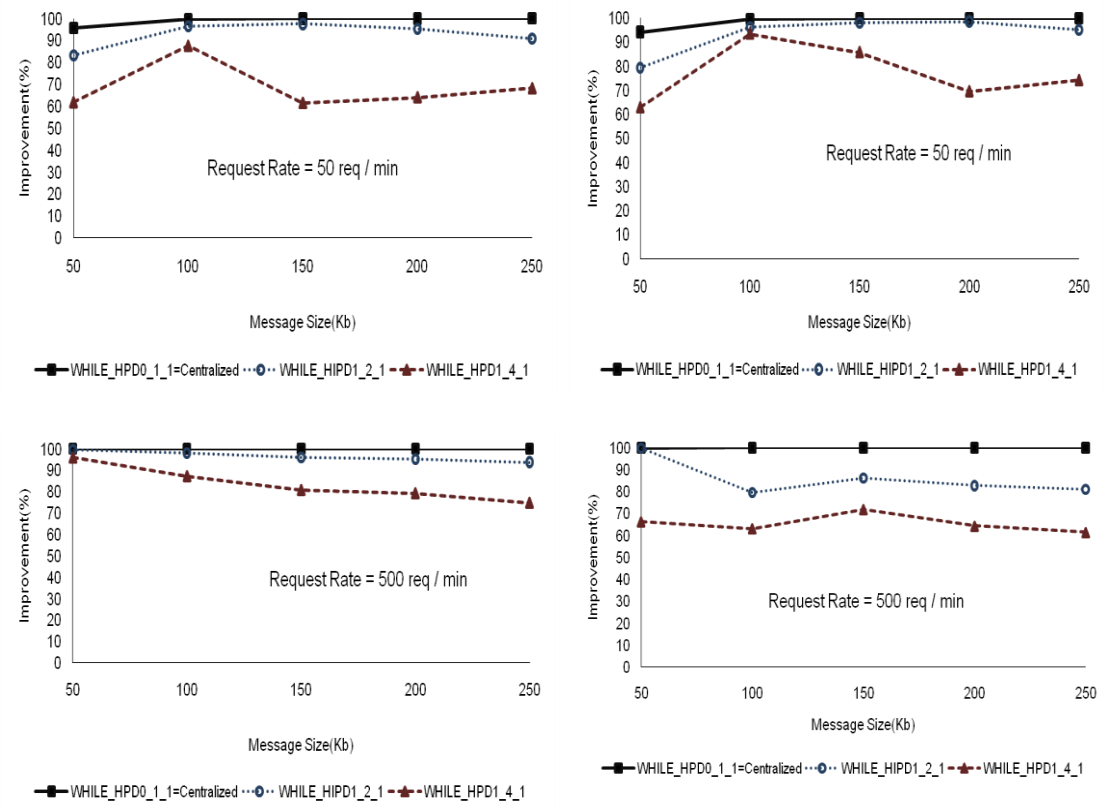
Figure 6.27: While-process Response-time Comparison, Type-1, Constant-Request-Rate, Variable-Message-Size, Min-Max distributions



Minimum Distribution **Maximum Distribution**

Figure 6.28: While-process Response-time Comparison, Type-2, Constant-Request-Rate, Variable-Message-Size, Min-Max distributions

In both architectures, the FPD response-time was improved; particularly, when the request rate and the size of messages were increased. The trend of improvement was directly proportional to the number of fragments and the size of messages exchanged among them. Consequently, lower number of fragments and smaller size of exchanged messages resulted in better response-time.



Type-1 **Type-2**

Figure 6.29: Type-1 and Type-2, While-process Response-time Improvement with Request Rates 50 and 500

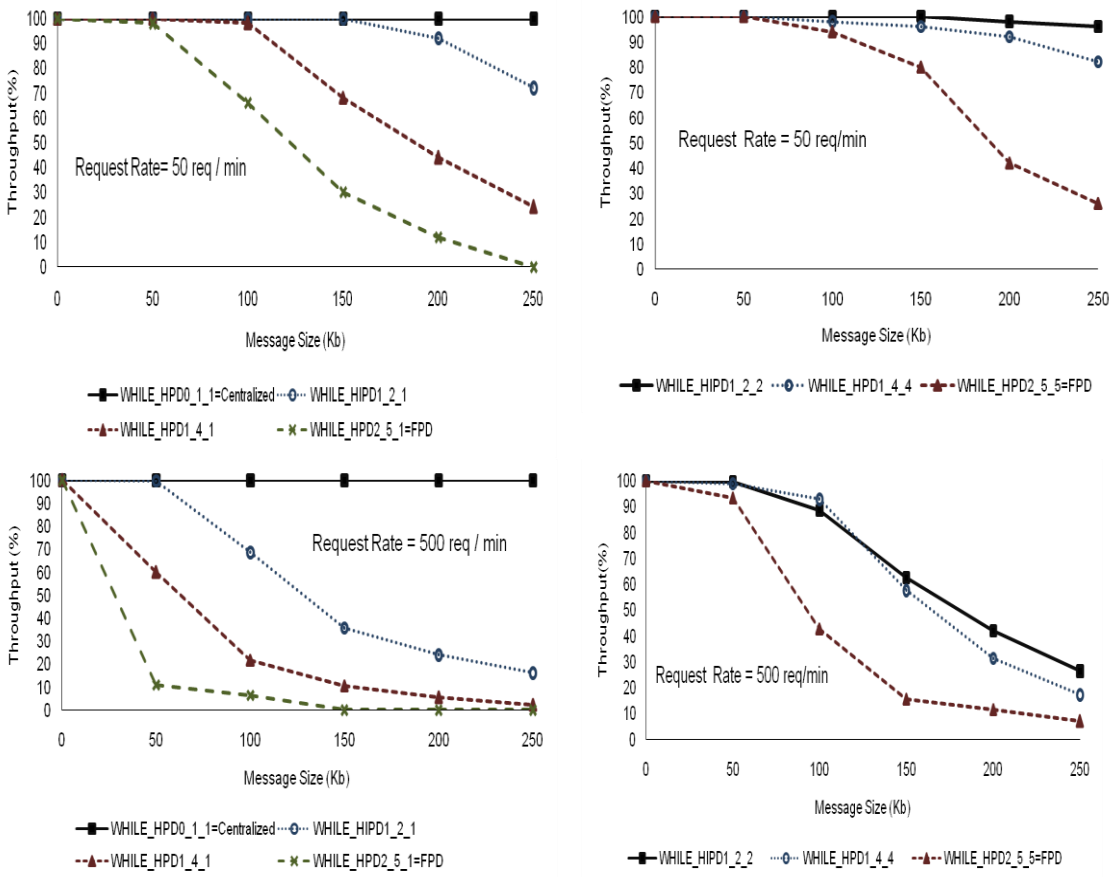
Table 6.7: Average Response-time Improvement of While Fragments Compared to FPD

	Req / Min	WHILE_HPD0_1_1 (Centralized)	WHILE_HIPD1_2_1	WHILE_HPD1_4_1
Response-time	**50**	99.03	92.67	68.58
	500	99.99	96.52	83.61
Type-1				
	Req / Min	**WHILE_HPD0_1_1 (Centralized)**	**WHILE_HIPD1_2_1**	**WHILE_HPD1_4_1**
Response-time	**50**	98.71%	93.50%	77.17%
	500	99.97%	85.83%	65.45%
Type-2				

Experiment-9 (frequent-path adaptability, evaluating throughput, constant-request-rate, variable-message-size)

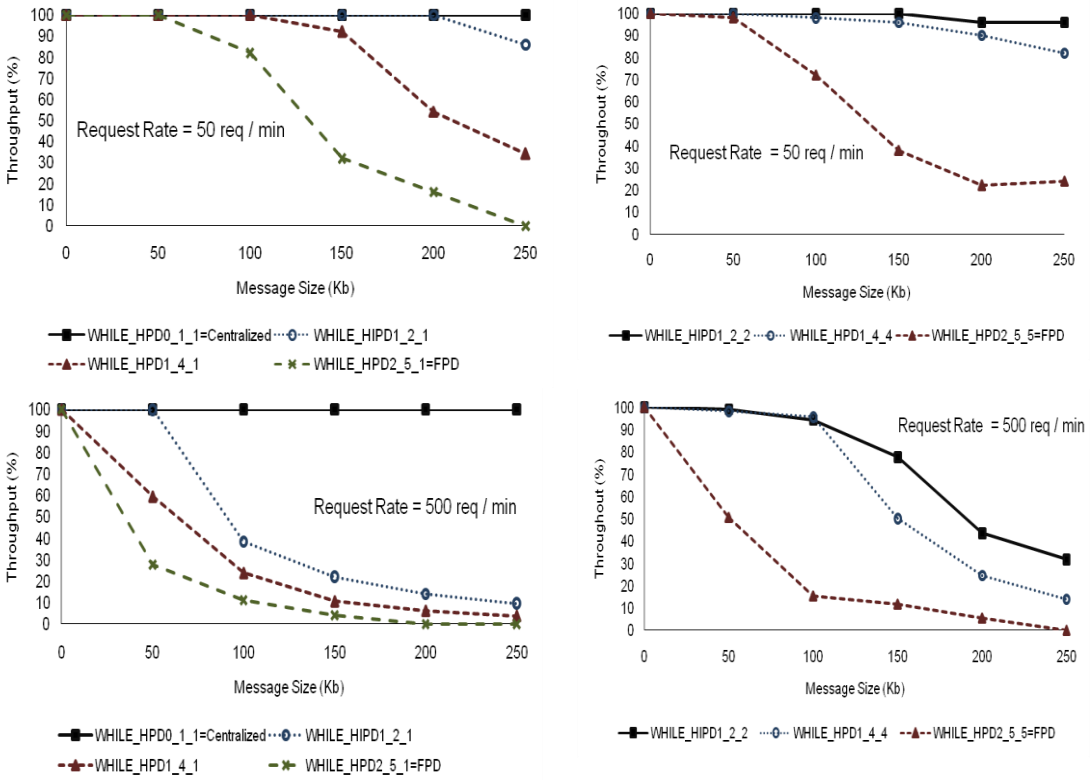
This experiment was conducted according to the Experiment-9 specified in Section 4.8.2. The experiment was performed for Min-Max distributions on both architectures Type-1 and Type-2.

Type-1: Figure 6.30 shows throughput behavior of the While fragments in Min-Max distributions running on the architecture Type-1. Figure 6.32 shows the improvement trend of the FPD throughput by the other fragments. According to Table 6.8, the average improvement of throughput in request rate 500 by the Centralized and HIPD_1_2_1 was 96.52% and 45.40%. In comparison, HPD1_4_1 improved the average response-time about 16.52%.



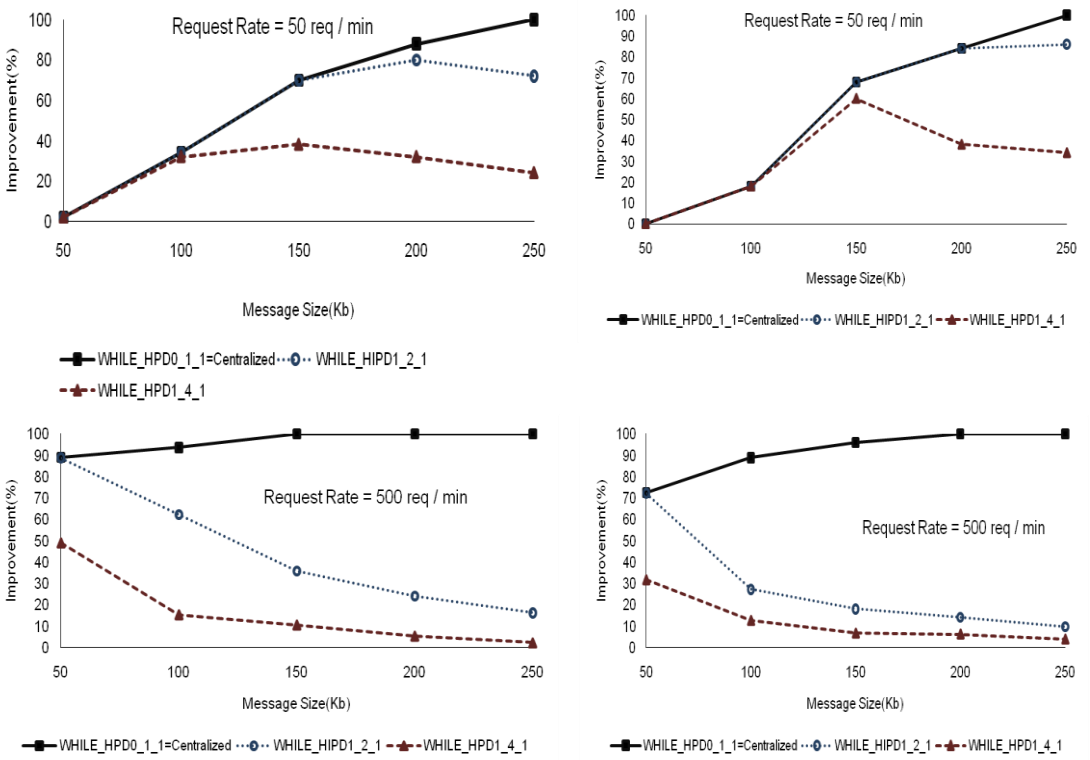
Minimum Distribution **Maximum Distribution**

Figure 6.30: While-process Throughput Comparison, Type-1, Constant-Request-Rate, Variable-Message-Size, Min-Max distributions



Minimum Distribution **Maximum Distribution**

Figure 6.31: While-process Throughput Comparison, Type-2, Constant-Request-Rate, Variable-Message-Size in Min-Max distributions



Type-1 **Type-2**

Figure 6.32: Type-1 and Type-2, While-process Throughput Improvement with Request Rates 50 and 500

Type-2: Figure 6.31 shows throughput behavior of the While fragments in Min-Max distributions running on the architecture Type-2. Figure 6.32 shows the improvement trend of the FPD throughput by the other fragments. According to Table 6.8, the average improvement of throughput in request rate 500 by the Centralized and HIPD_1_2_1 was 91.44% and 28.20%. In comparison, HPD1_4_1 improved the average throughput about 12.20%.

Table 6.8: Average Throughput Improvement of While Fragments Compared to FPD

	Req / Min	WHILE_HPD0_1_1 (Centralized)	WHILE_HIPD1_2_1	WHILE_HPD1_4_1
Throughput	**50**	58.80%	51.60%	25.60%
	500	96.52%	45.40%	16.52%
Type-1				
	Req / Min	**WHILE_HPD0_1_1 (Centralized)**	**WHILE_HIPD1_2_1 (IPD)**	**WHILE_HPD1_4_1**
Throughput	**50**	54.00%	51.20%	30.00%
	500	91.44%	28.20%	12.20%
Type-2				

In both experiments, the FPD throughput was improved; especially, when the request rate and the size of messages were increased. The trend of throughput improvement was directly proportional to the number of fragments and the size of messages exchanged among them. Consequently, the lower number of fragments and size of exchanged messages among the While-process activities resulted in better throughput. The size of the messages had a negative impact on the throughput due to filling system memory by long message queues.

Experiment-10 (frequent-path adaptability, evaluating bandwidth-usage, variable-request-rate/constant-message-size)

This experiment was conducted according to Experiment-10 specified in 4.8.2. Figure 6.33 compares the number of exchanged messages by fragments produced by different decentralization methods. The Centralized as well as HIPD were considerably similar due to the less number of fragments. As much HPD granularity

level increased, the number of exchanged messages was increased as well, which was due to the fact that higher granularity levels provided more fragments compared to lower granularity levels. The FPD method showed the worst number of message passing while the Centralized and HIPD were the best methods.

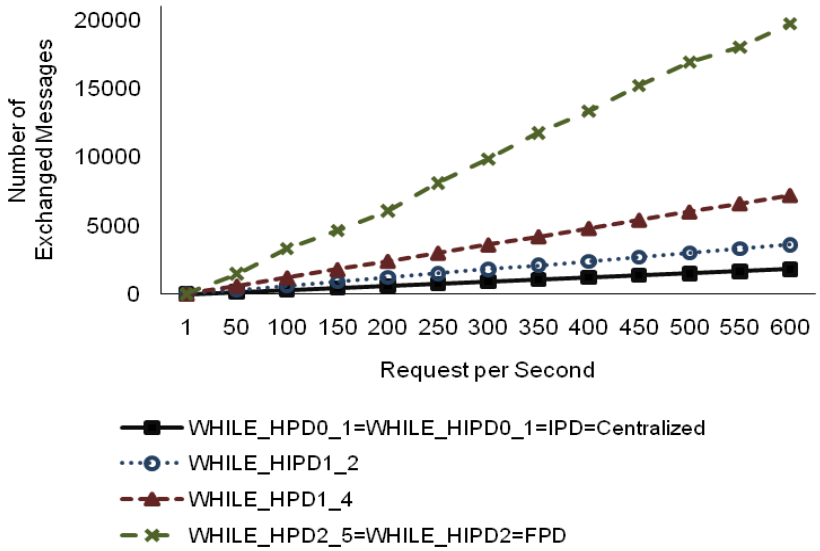


Figure 6.33: Exchanged Message Comparison of While-process Fragments

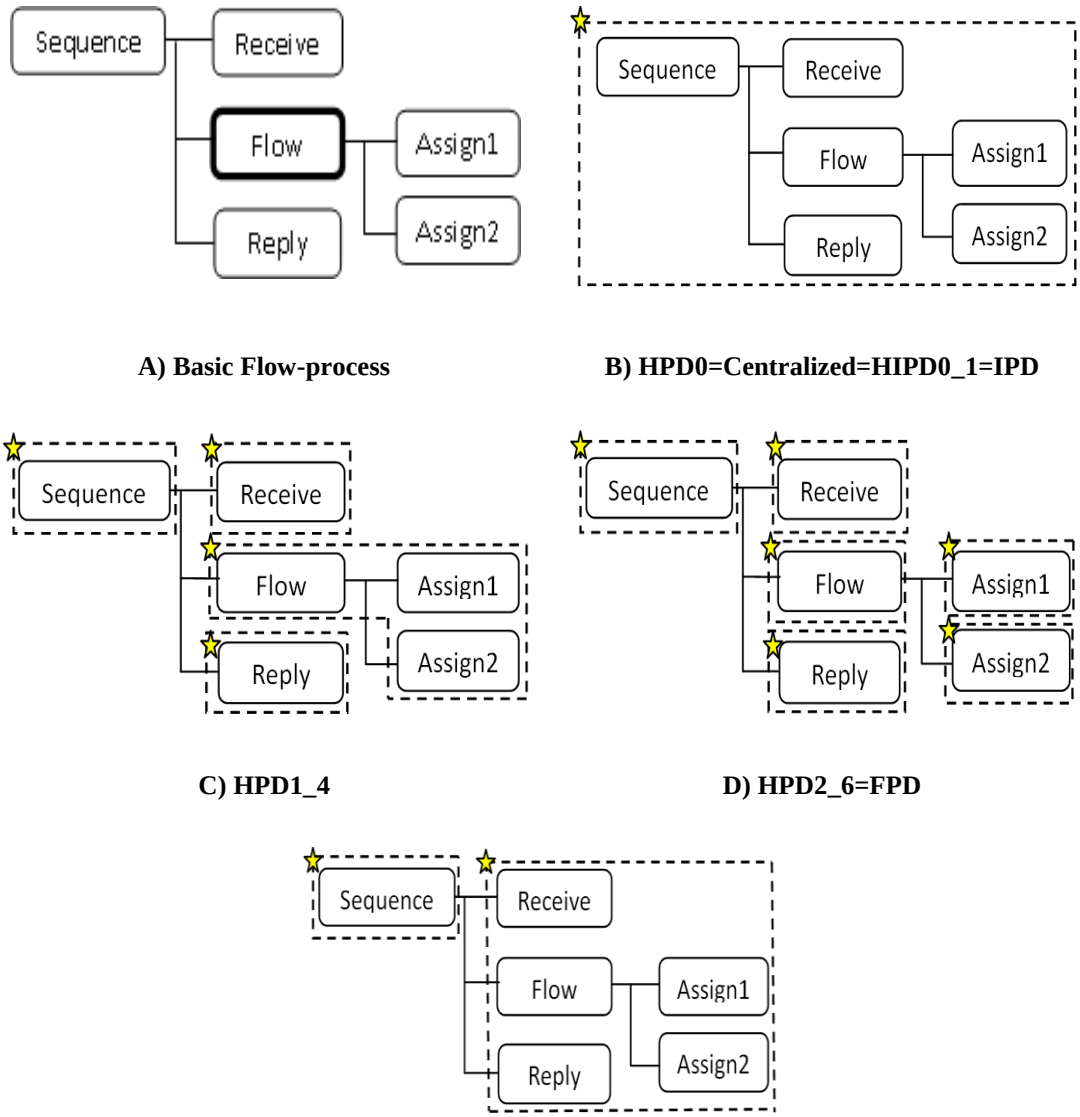
6.4.3 Flow-process Experiment-6 to Experiment-10

This section focuses on decentralization of the basic Flow business process based on the Experiment 6, 7, 8, 9 and 10 introduced in Section 4.8.2. At first, the fragmentation of the process is discussed and then the fragments will be used by the experiments in the following sections.

Basic Flow-process

A Flow activity (Active-Endpoints, 2011; OASIS, 2011) defines one or more child activities executed concurrently. The Flow activity also allows us to synchronize activities; one activity starts when another ends. The implementation of this synchronization is similar to the synchronization of parallel threads in programming languages. Figure 6.34(A) shows a basic Flow BPEL process and Figure 6.34(B-D)

is also the decentralization of the basic Flow-process using the HPD method in different levels of granularity. Furthermore, Figure 6.34(E) shows HIPD1 decentralization of the Flow and path [(sequence (receive, flow (assign1, assign2), reply))] is assumed to be a frequent path. In case of the Flow-process, the Centralized model is also analogous to HIPD0.



A) Basic Flow-process

B) HPD0=Centralized=HIPD0_1=IPD

C) HPD1_4

D) HPD2_6=FPD

E) HIPD1_2

Figure 6.34: Flow-process Decentralization Patterns

Consequently, the following experiments were conducted using the various versions of the Flow-process including the Centralized model with one fragment,

FLOW_HIPD1_2 with two fragments, FLOW_HPD1_4 with four fragments and the FPD method with six fragments.

Experiment-6 (frequent-path adaptability, response-time evaluation, variable-request-rate, constant-message-size)

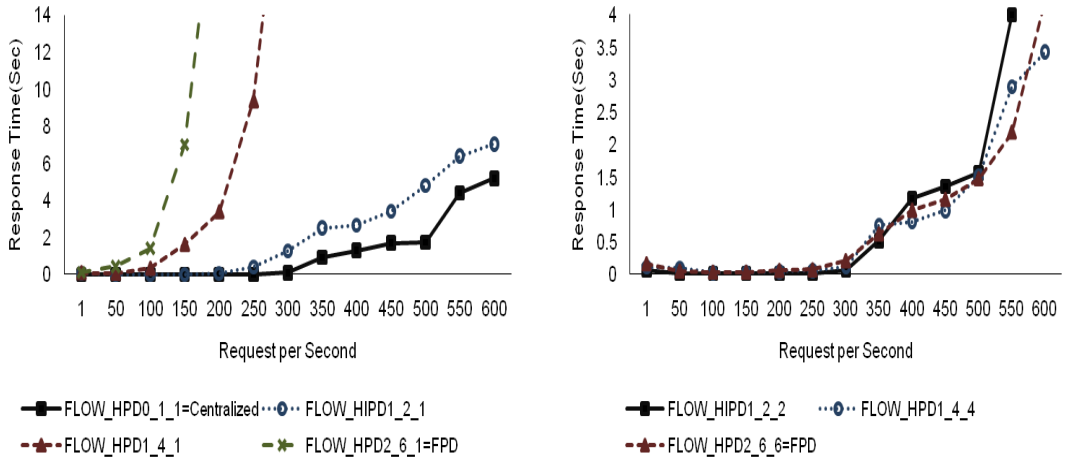
This experiment was conducted according to the Experiment-6 specified in Section 4.8.2. Figure 6.35 and Figure 6.36 compare the Flow-process response-time in Min-Max distributions, running on both architectures Type-1 and Type-2.

Type-1: Figure 6.37 shows the response-time improvement trend of the FPD by the other decentralization methods**.** It is clear that the Centralized and HIPD fragments improved the FPD. Table 6.9 also depicts that the average response-time improvement of the FPD was around 99.52% and 99.10% by the Centralized and IPD. In comparison, HPD1_4_1 improved the average response-time about 78.09%.

Type-2: Figure 6.37 also shows the response-time improvement trend of the FPD by the other decentralization methods**.** It is clear that the Centralized and HIPD fragments improved the FPD. Table 6.9 also depicts that the average response-time improvement of the FPD was around 65.10% and 56.54% by the Centralized and IPD. In comparison, HPD1_4_1 improved the average response-time about 26.35%.

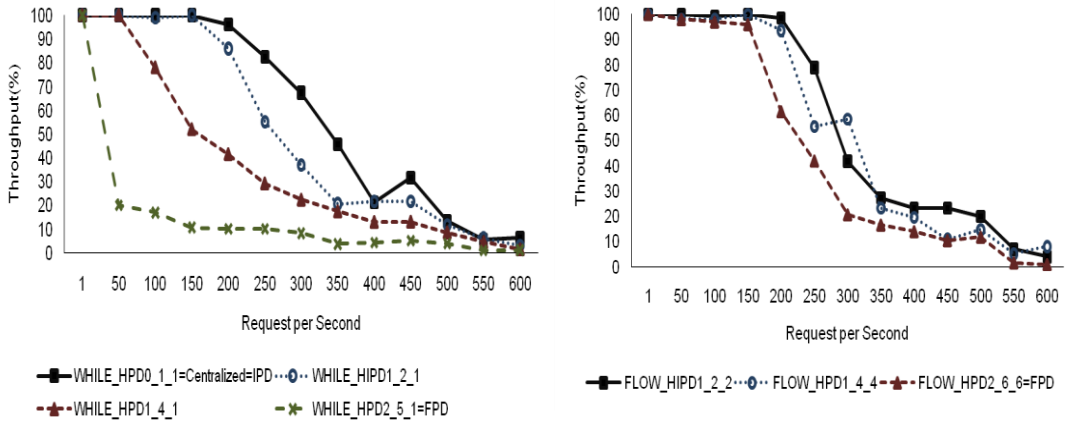
The experiment showed that the number of fragments and naturally exchanged messages had impacted the response-time effectively. In case of the Flow-process, it not only benefited from the parallel execution of activities, but also used the frequent path decentralization. The HPD method did not consider the relevancy of activities; however, the lessened number of fragments and exchanged messages in HPD_1_4_1 improved the FPD.

This experiment also showed calling various fragments of the Flow-process in different rates, in absence of message size, resulted in improving the Flow FPD version when the fragments executed by both architectures Type-1 and Type-2. Response-time improvement in architecture Type-1 was more than Type-2, which substantiated the resiliency of Type-2 compared to Type-1 in executing different fragment types.



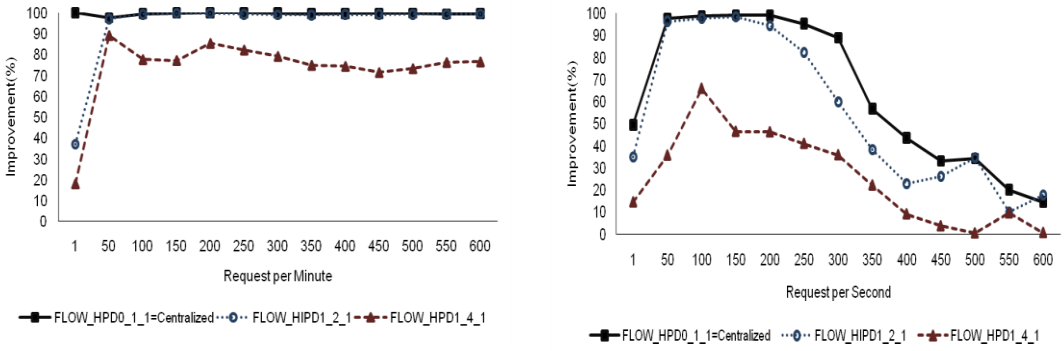
Minimum Distribution **Maximum Distribution**

Figure 6.35: Flow-process Response-time Comparison, Type-1, Variable-Request-Rate, Constant-Message-Size, Min-Max distributions



Minimum Distribution **Maximum Distribution**

Figure 6.36: Flow-process Response-time Comparison, Type-2, Variable-Request-Rate, Constant-Message-Size, Min-Max distributions



Type-1 **Type-2**

Figure 6.37: Type-1 and Type-2, Flow-process Response-time Improvement Comparison

Table 6.9: Average Response-time Improvement of Flow Fragments Compared to FPD

	FLOW_HPD0_1_1 (Centralized)	FLOW_HIPD1_2_1	FLOW_HPD1_4_1
Response-time	99.52%	99.10%	78.09%
	65.10%	56.54%	26.35%

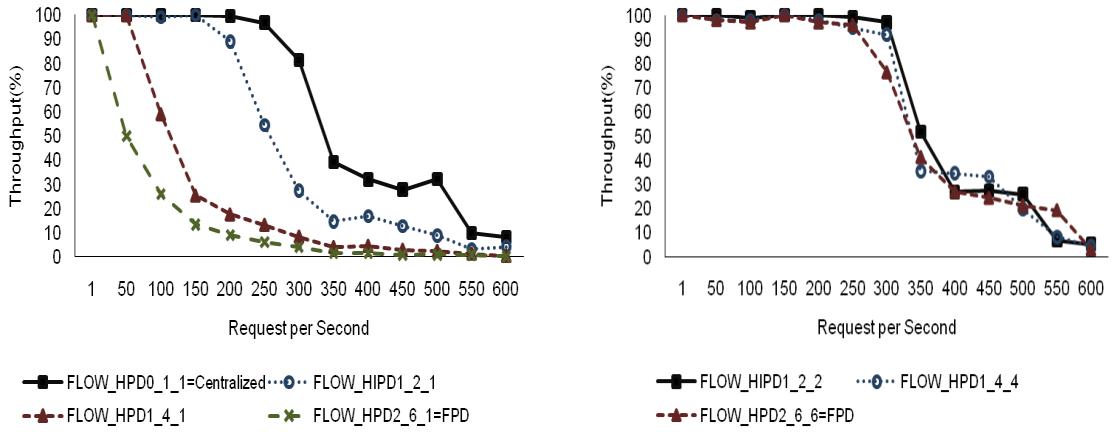
Experiment-7 (frequent-path adaptability, throughput evaluation, variable-request-rate, constant-message-size)

This experiment was conducted according to Experiment-7 specified in Section 4.8.2 and Figure 6.38 and Figure 6.39 show the comparison of the Flow fragments throughput in Min-Max distributions, run by both architectures Type-1 and Type-2.

Type-1: Figure 6.40 also shows the throughput improvement trend of the FPD by other decentralization methods**.** It is clear that the Centralized and HIPD fragments improved the FPD. Table 6.10 also depicts that throughput improvement of the FPD was around 65.10%, 34.64% and 10.38% by the Centralized, HIPD1_2_1 and HPD1_4_1.

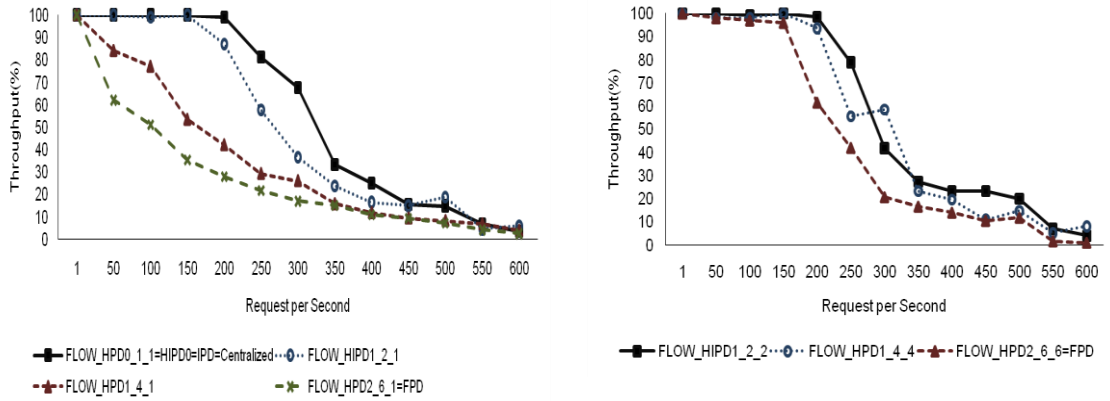
Type-2: Figure 6.40 also shows the improvement trend of the FPD by other decentralization methods**.** It is clear that the Centralized and HIPD fragments improved the FPD. Table 6.10 also depicts that throughput improvement of the FPD was around 31.86%, 25.02% and 8.57% by the Centralized, HIPD1_2_1 and HPD1_4_1.

This experiment also showed calling Flow fragments in different rates, in absence of message size, resulted in improving the Flow FPD version in both architectures Type-1 and Type-2. The throughput was improved because of the less amount of message passing in both HPD and HIPD fragments. In the HPD, the number of messages was decreased due to less number of fragments; however, in the HIPD because of applying frequent paths. The throughput improvement in architecture Type-1 was more than Type-2, which substantiated the resiliency of Type-2 compared to Type-1 in executing different fragment types.



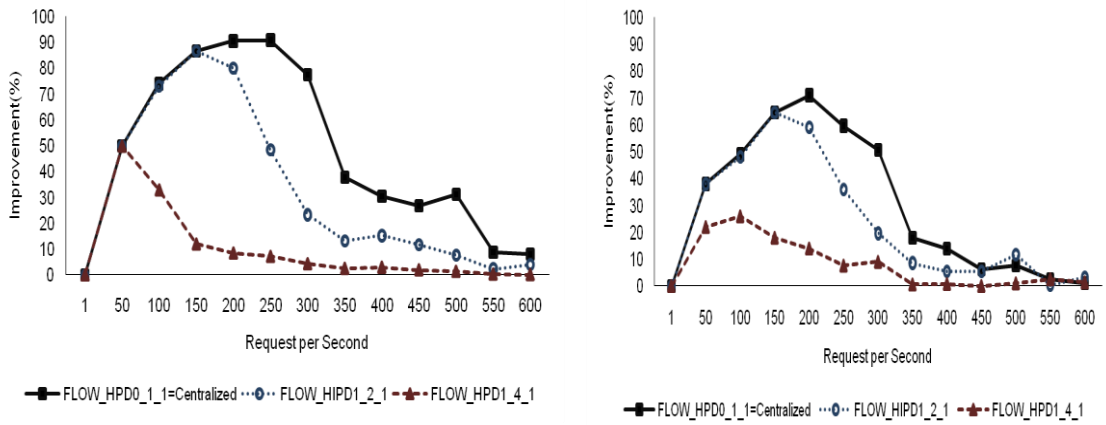
Minimum Distribution **Maximum Distribution**

Figure 6.38: Flow-process Throughput Comparison, Type-1, Variable-Request-Rate, Constant-Message-Size, Min-Max distributions



Minimum Distribution **Maximum Distribution**

Figure 6.39: Flow-process Throughput Comparison, Type-2, Variable-Request-Rate, Constant-Message-Size, Min-Max distributions



Minimum Distribution **Maximum Distribution**

Figure 6.40: Type-1 and Type-2, Flow-process Throughput Improvement Comparison

Table 6.10: Average Throughput Improvement of Flow Fragments Compared to FPD

	FLOW_HPD0_1_1 (Centralized)	FLOW_HIPD1_2_1	FLOW_HPD1_4_1
Type-1	65.10%	34.64%	10.38%
Type-2	31.86%	25.02%	8.57%

Experiment-8 (frequent-path adaptability, response-time evaluation, constant-request-rate, variable-message-size)

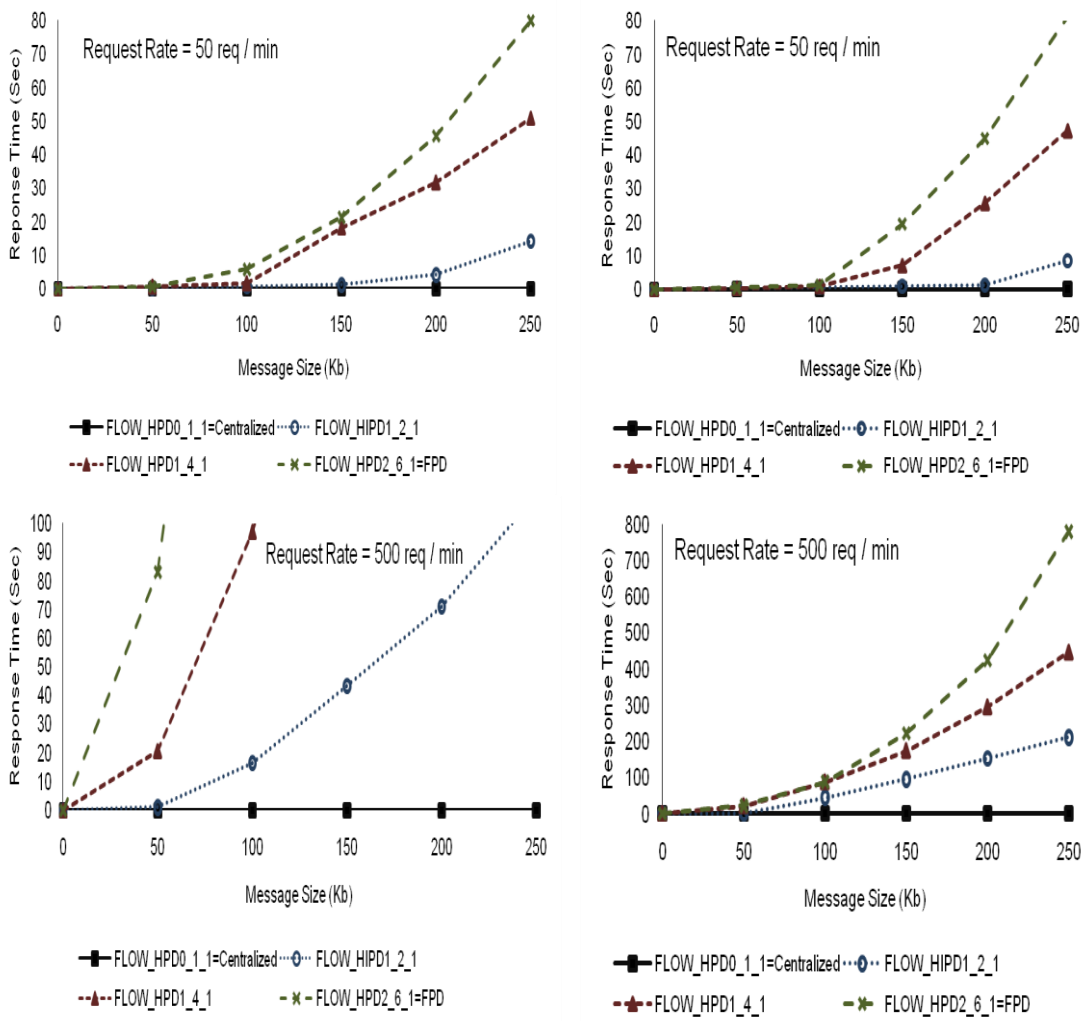
This experiment was conducted according to Experiment-8 specified in Section 4.8.2. It was performed for Min-Max distributions, running on both architectures Type-1 and Type-2.

Type-1: Figure 6.41 shows response-time behavior of Flow fragments in Min-Max distributions running on the architecture Type-1. In addition, Figure 6.43 shows the trend of the FPD response-time improvement by the other methods. The average improvement of response-time in request rate 500 by the Centralized and HIPD1_2_1 was 99.98% and 94.01%. In comparison, HPD1_4_1 improved the average response-time about 67.85%.

Type-2: Figure 6.42 also shows response-time behavior of the Flow in Min-Max distributions running on the architecture Type-2. In addition, Figure 6.43 shows the trend of the FPD response-time improvement by the other methods. The average

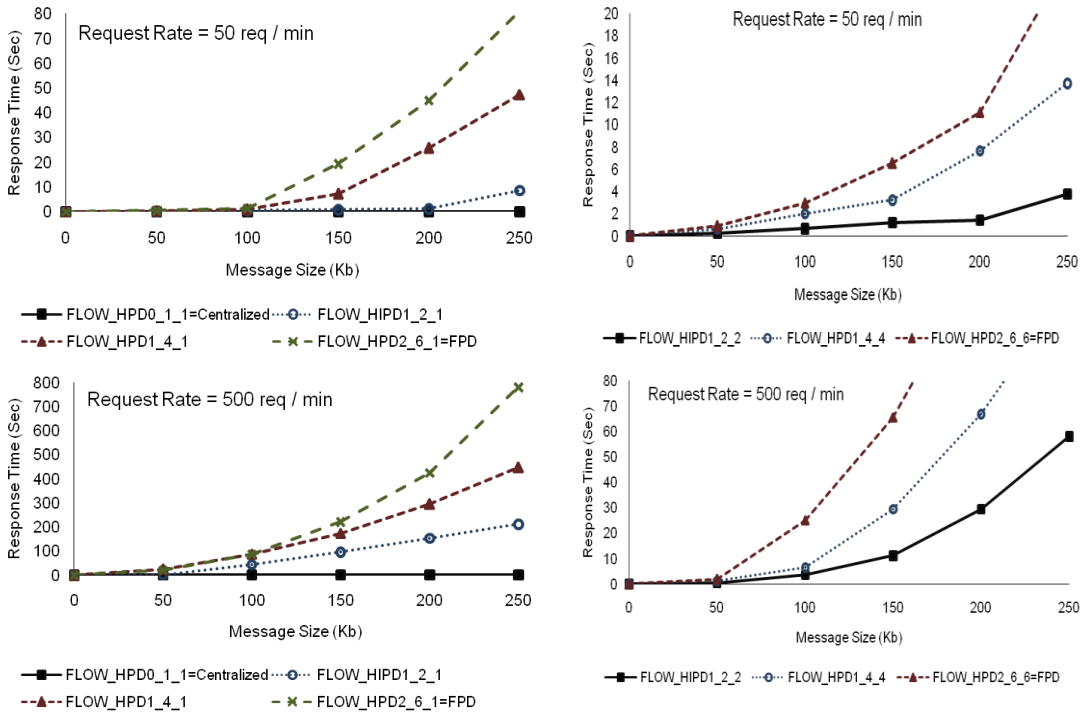
improvement of response-time in request rate 500 by the Centralized and HIPD1_2_1 was 99.92% and 67.75%. In comparison, HPD1_4_1 improved the average response-time about 20.11%.

The FPD was improved in both architectures; in particular, when the request rate and the size of messages were increased. The trend of improvement was directly proportional to the number of fragments and the size of messages exchanged among them. Consequently, the lower number of fragments and size of exchanged messages among the Flow-process activities resulted in better response-time for the process.



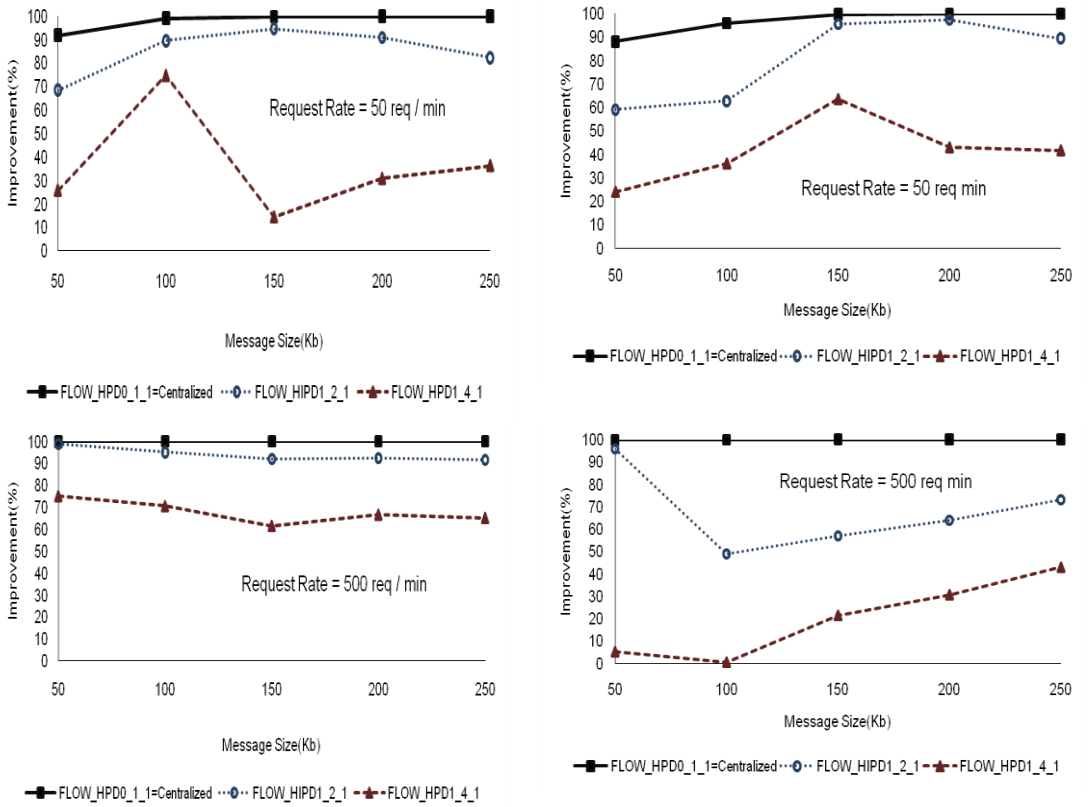
Minimum Distribution **Maximum Distribution**

Figure 6.41: Flow-process Response-Time Comparison, Type-1, Constant-Request-Rate, Variable-Message-Size, Min-Max distributions



Minimum Distribution **Maximum Distribution**

Figure 6.42: Flow-process Response-Time Comparison, Type-2, Constant-Request-Rate, Variable-Message-Size, Min-Max distributions



Type-1 **Type-2**

Figure 6.43: Type-1 and Type-2, Flow-process Response-time Improvement with Request Rates 50 and 500

Table 6.11: Average Response-time Improvement of Flow Fragments Compared to FPD

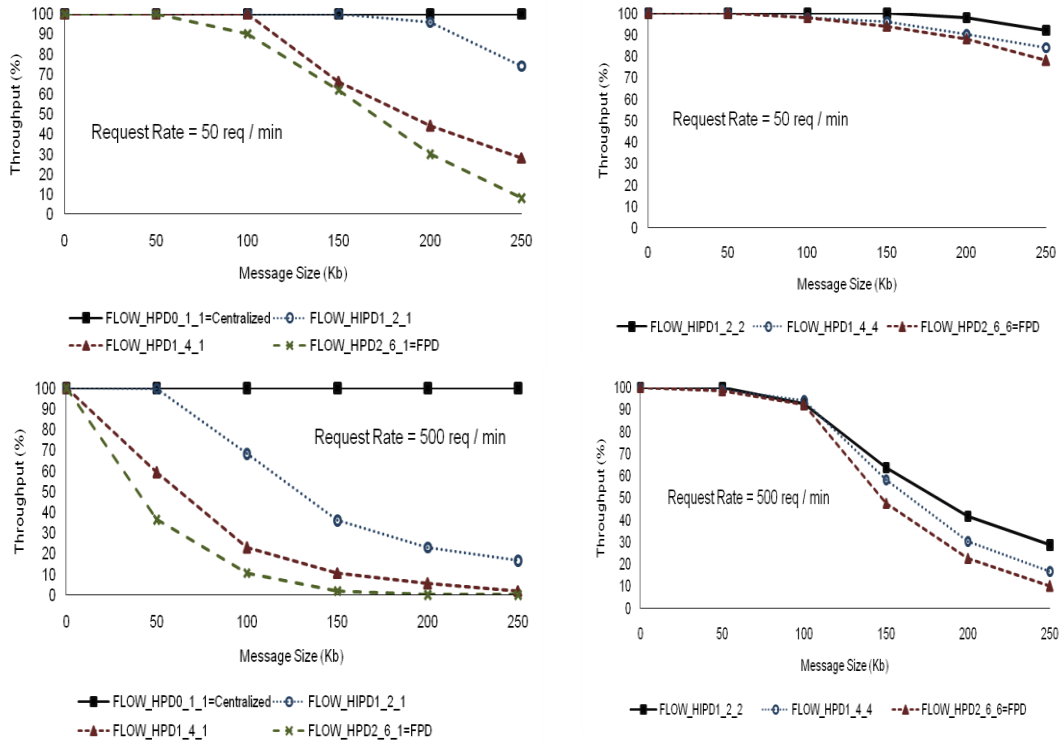
	Req / Min	FLOW_HPD0_1_1 (Centralized)	FLOW_HIPD1_2_1	FLOW_HPD1_4_1
Response-time	50	98.10%	85.20%	36.39%
	500	99.98%	94.01%	67.85%
Type-1				
	Req / Min	FLOW_HPD0_1_1 (Centralized)	FLOW_HIPD1_2_1	FLOW_HPD1_4_1
Response-time	50	96.70%	80.81%	41.67%
	500	99.92%	67.75%	20.11%
Type-2				

Experiment-9 (frequent-path adaptability, evaluating throughput, constant-request-rate, variable-message-size)

This experiment was conducted according to the Experiment-9 specified in Section 4.8.2. The experiment was performed for Max-Min distributions on both architectures Type-1 and Type-2.

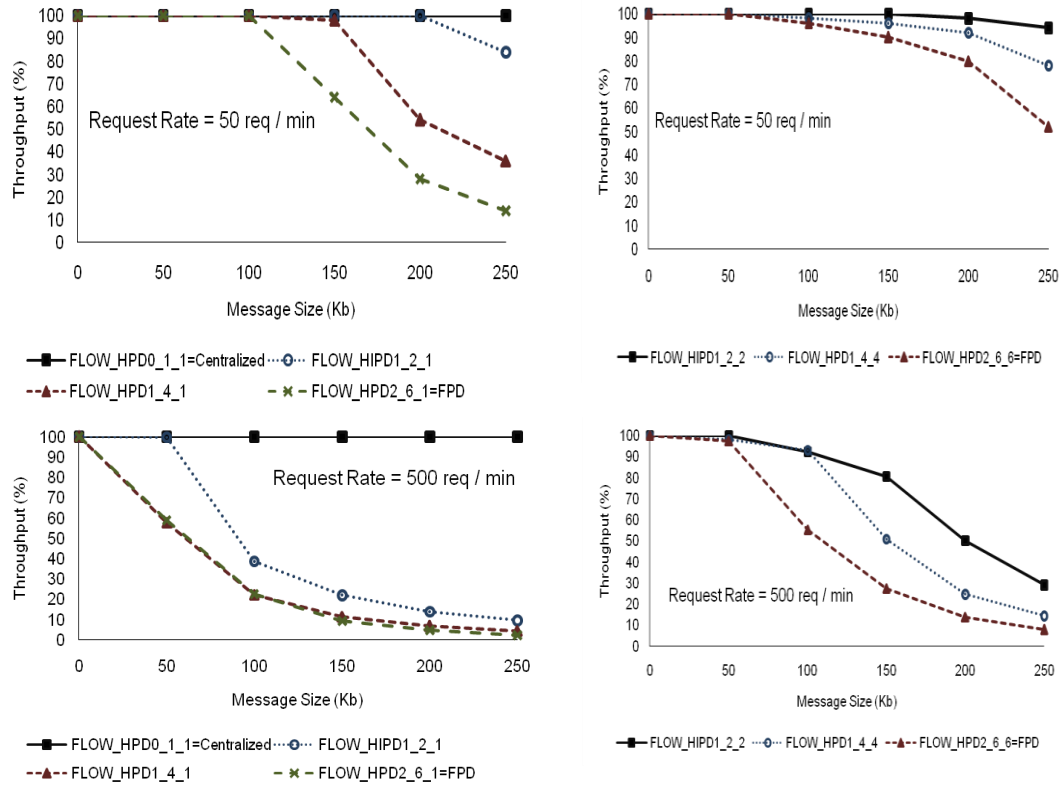
Type-1: Figure 6.44 shows the behavior of the decentralized versions of the Flow-process in Min-Max distributions running on the architecture Type-1. In addition, Figure 6.46 shows throughput improvement percent of the FPD by the other decentralization methods. According to Table 6.12, in request rate 500, the average improvement of throughput by the Centralized and IPD was 90.20% and 38.92%. In comparison, and HIPD1_4_1 improved the FPD around 10.16%.

Type-2: Figure 6.45 shows the behavior of the decentralized versions of the Flow-process in Min-Max distributions running on the architecture Type-2. In addition, Figure 6.46 shows throughput improvement percent of the FPD by other decentralization methods. According to Table 6.12, in request rate 500, the average improvement of throughput by the Centralized and IPD was 80.72% and 17.52%. In comparison, and HIPD1_4_1 improved the FPD around 1.52%.



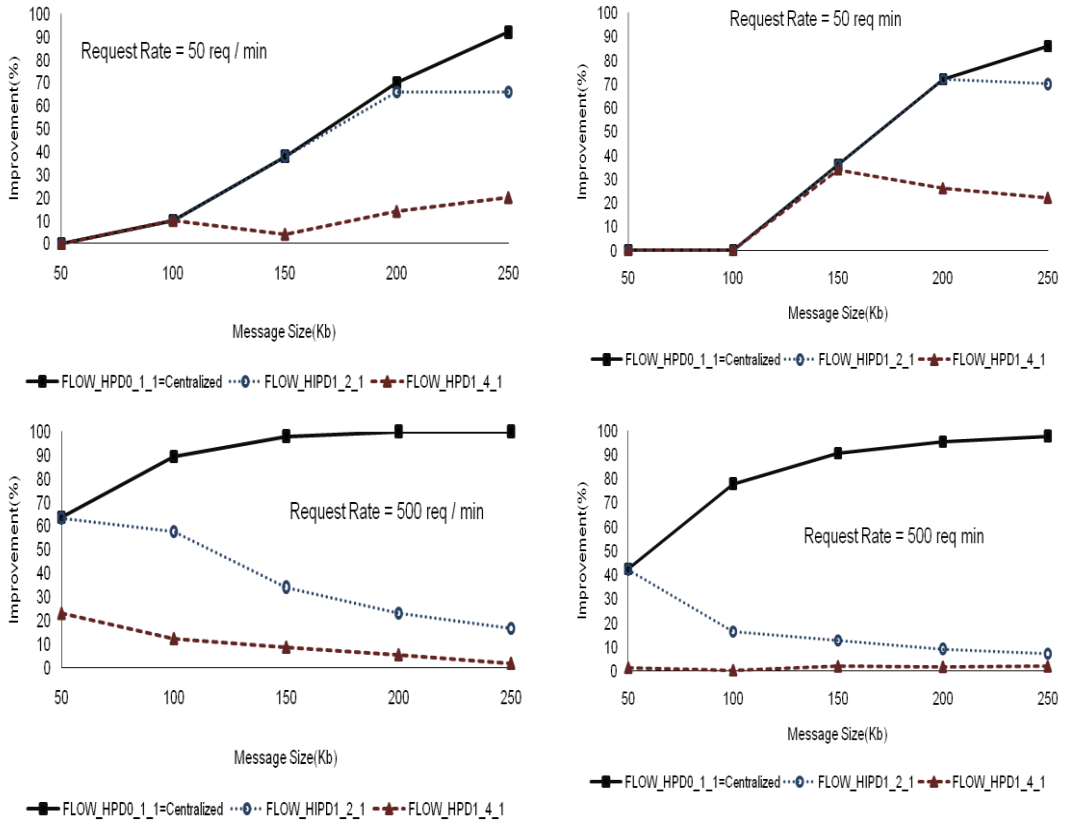
Minimum Distribution **Maximum Distribution**

Figure 6.44: Flow-process Throughput Comparison, Type-1, Constant-Request-Rate, Variable-Message-Size, Min-Max distributions



Minimum Distribution **Maximum Distribution**

Figure 6.45: Flow-process Throughput Comparison, Type-2, Constant-Request-Rate, Variable-Message-Size, Min-Max distributions



Type-1 **Type-2**

Figure 6.46: Type-1 and Type-2, Flow-process Throughput Improvement with Request Rates 50 and 500

Table 6.12: Average Throughput Improvement of Flow Fragments Compared to FPD

	Req / Min	FLOW_HPD0_1_1 (Centralized)	FLOW_HIPD1_2_1	FLOW_HIPD1_4_1
Throughput	**50**	42.00%	36.00%	9.60%
	500	90.20%	38.92%	10.16%
Type-1				
	Req / Min	**FLOW_HPD0_1_1 (Centralized)**	**FLOW_HIPD1_2_1**	**FLOW_HIPD1_4_1**
Throughput	**50**	38.80%	35.60%	16.40%
	500	80.72%	17.52%	1.52%
Type-2				

The FPD was improved in both architectures; in particular, when the request rate and the size of messages were increased. The trend of improvement was directly proportional to the number of fragments and the size of messages exchanged among them. Consequently, the lower number of fragments and size of exchanged messages among the Flow-process activities resulted in better throughput for the process.

Experiment-10 (frequent-path adaptability, evaluating bandwidth-usage, variable-request-rate/constant-message-size)

This experiment was conducted according to the Experiment-10 specified in Section 4.8.2. Different decentralization methods produce a variety number of fragments. The more number of fragments, the higher number of message passing. This experiment measures the number of exchanged messages while requests with different rates are sent for the business process.

Figure 6.47 compares the number of exchanged messages by the fragments produced by different decentralization methods. The Centralized as well as HIPD methods were considerably similar due to the less number of fragments. As much the HPD granularity level increased, the number of exchanged messages increased as well. Higher granularity levels provided more fragments compared to lower granularity levels. The FPD method showed the worst number of message passing while the Centralized and HIPD were the best methods.

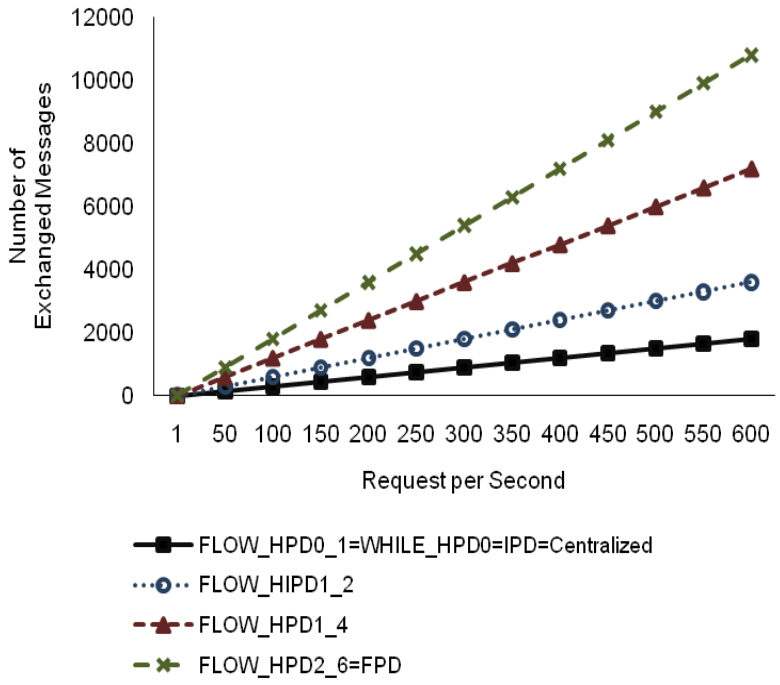


Figure 6.47: Exchanged Message Comparison of Flow-process Fragments

6.4.4 Loan-process Experiment-6 to Experiment-13

This section focuses on the evaluation of HPD and HIPD decentralization applying the loan application business process. The HPD and HIPD decentralization of the Loan-process are shown in Section 5.6 and Section 5.7, respectively. The produced fragments are then evaluated by the experiments 6, 7, 8, 9, 10 for proving the frequent-path adaptability and by experiments 11, 12 and 13 for substantiating the proportional-fragment adaptability based on the method specified in Section 4.8.2.

Experiment-6 (frequent-path adaptability, response-time evaluation, variable-request-rate/constant-message-size)

This experiment was conducted according to the Experiment-6 specified in the Section 4.8.2. The response-time behavior of the loan fragments in Min-Max distributions running on the architecture Type-1 and Type-2 are shown in Figure 6.48 and Figure 6.49.

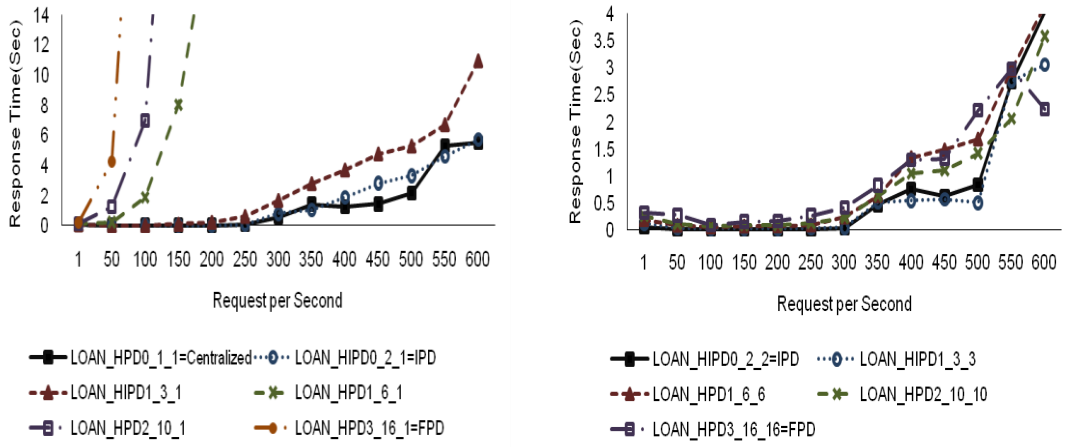
Type-1: Figure 6.50 shows the improvement trend of the FPD by the other decentralization methods**.** It is crystal clear that the Centralized and HIPD fragments improved the FPD. Table 6.13 also depicts that the FPD average response-time was improved around 99.96%, 99.95% and 99.90% by the Centralized, IPD and HIPD1_3_1. In comparison, HPD1_6_1 and HPD2_10_1 plummeted the average response-time about 92.13 % and 71.53%.

Type-2: Figure 6.50 also shows the improvement trend of the FPD by the other decentralization methods**.** Similar to the Type-1, it is crystal clear that the Centralized and HIPD fragments improved the FPD. Table 6.13 also depicts that the FPD average response-time was improved around 87.61%, 84.27% and 81.24% by the Centralized, IPD and HIPD1_3_1. In comparison, HPD1_6_1 and HPD2_10_1

decreased the average response-time about 63.98% and 44.27%, which can be justified as Type-1.

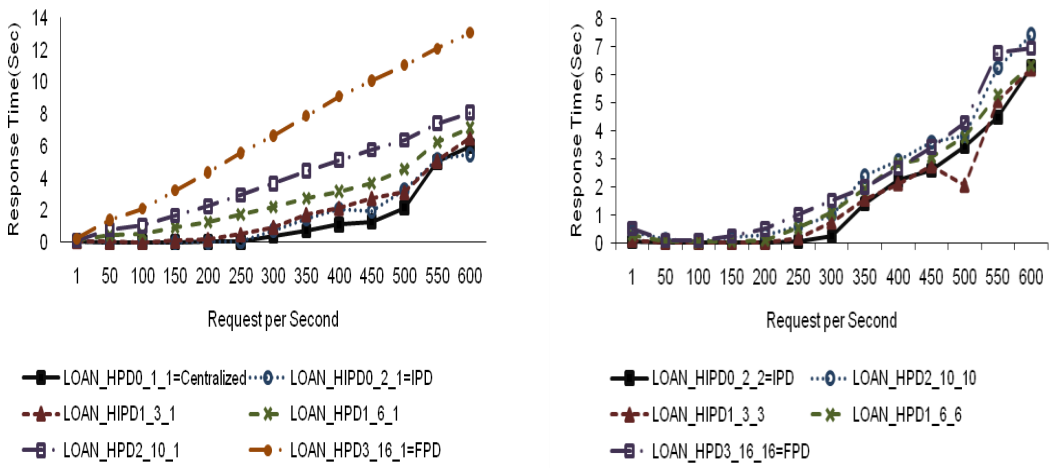
The experiments showed that the number of fragments and naturally exchanged messages had effective impact on the response-time. The HIPD fragments due to less number of fragments and the relevancy of activities inside fragments showed low response-time very similar to the Centralized. It completely showed the ability of the frequent-path adaptability. The HPD method did not consider the relevancy of activities; however, the lessened number of exchanged messages in HPD1_6_1 and HPD2_10_1 improved the FPD, which implicitly proved the proportional-fragment adaptability.

This experiment also showed calling various versions of the Loan-process in different rates, in absence of message size, resulted in improving the Loan-process FPD version when the fragments were executed by both architectures Type-1 and Type-2. Response-time improvement for a more complex business process in architecture Type-1 was more than Type-2. It substantiated the resiliency of Type-2 compared to Type-1 in executing different fragment types.



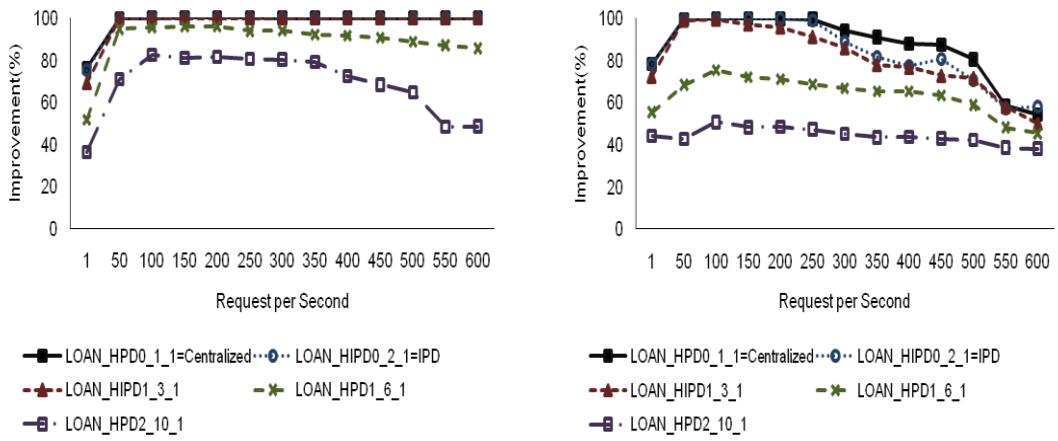
Minimum Distribution **Maximum Distribution**

Figure 6.48: Loan-process Response-time Comparison, Type-1, Variable-Request-Rate, Constant-Message-Size, Min-Max distributions



Minimum Distribution **Maximum Distribution**

Figure 6.49: Loan-process Response-time Comparison, Type-2, Variable-Request-Rate, Constant-Message-Size, Min-Max distributions



Type-1 **Type-2**

Figure 6.50: Type-1 and Type-2, Loan-process Response-time Improvement

Table 6.13: Average Response-time Improvement of Loan Fragments Compared to FPD

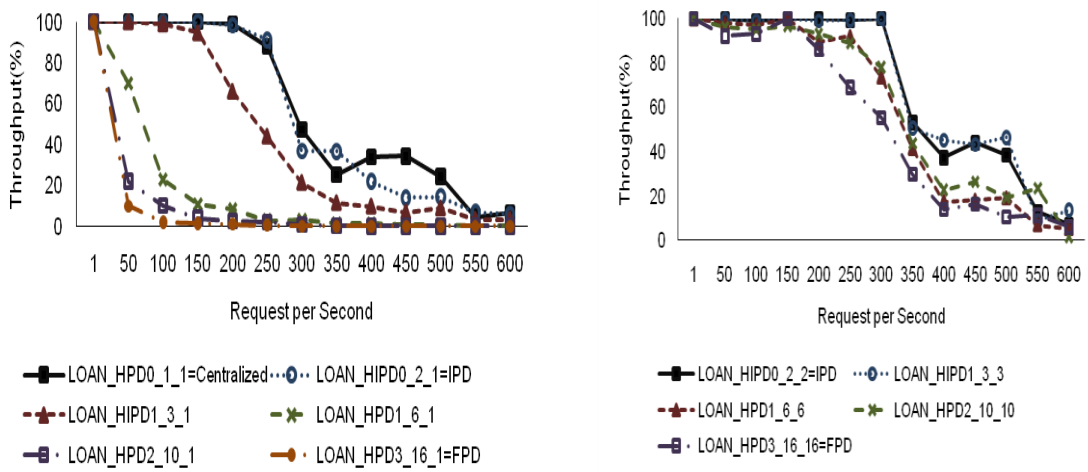
	LOAN_HPD0_1_1 (Centralized)	LOAN_HIPD0_2_1 (IPD)	LOAN_HIPD1_3_1	LOAN_HPD1_6_1	LOAN_HPD2_10_1
Type-1	99.96%	99.95%	99.90%	92.13%	71.53%
Type-2	87.61%	84.27%	81.24%	63.98%	44.27%

Experiment-7 (frequent-path adaptability, throughput evaluation, variable-request-rate/constant-message-size)

This experiment was conducted according to the Experiment-7 specified in the Section 4.8.2. Figure 6.51 and Figure 6.52 show throughput behavior of the decentralized versions of the Loan-process in Min-Max distributions running on the architectures Type-1 and Type-2.

Type-1: Figure 6.53 also shows the throughput improvement trend of the FPD by the other loan fragments**.** It is crystal clear that the Centralized and HIPD fragments improved the FPD. Table 6.14 also depicts that the average throughput of the FPD was improved around 54.03%, 50.99% and 54.03% by the Centralized, IPD and HIPD1_3_1. In comparison, HPD1_6_1 and HPD210_1 improved the average throughput about 9.01 % and 2.25%.

Type-2: Figure 6.53 also shows the improvement trend of the FPD by the other loan fragments**.** It is crystal clear that the Centralized and HIPD fragments improved the FPD. Table 6.14 also depicts that the average improvement percent of the FPD was improved around 50.06%, 43.38% and 35.61% by the Centralized, IPD and HIPD1_3_1. In comparison, HPD1_6_1 and HPD2_10_1 improved the average throughput about 12.56% and 4.98%.

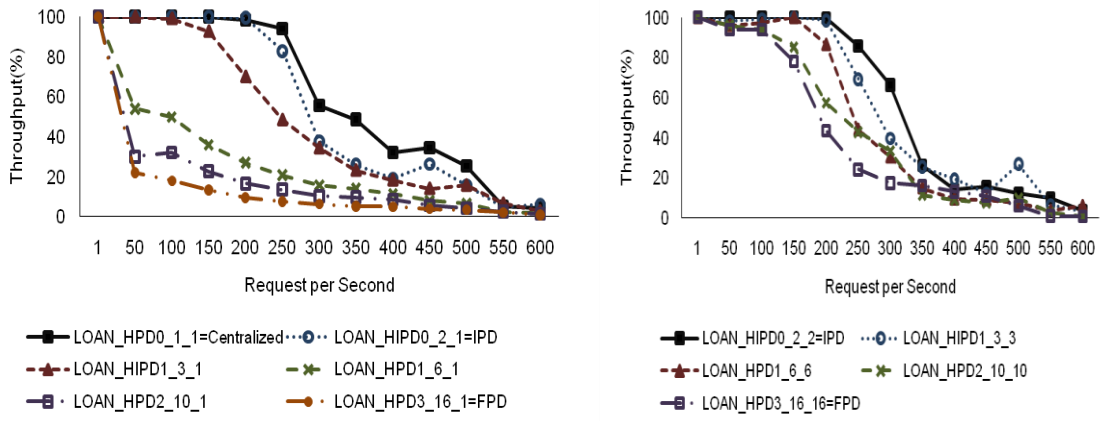


Minimum Distribution **Maximum Distribution**

Figure 6.51: Loan-process Throughput Comparison, Type-1, Variable-Request-Rate, Constant-Message-Size, Min-Max distributions

The execution of the Loan-process HIPD fragments showed better throughput result due to considering the frequent-path adaptability. The HPD method did not consider the relevancy of activities; however, considering fragment proportionality resulted in

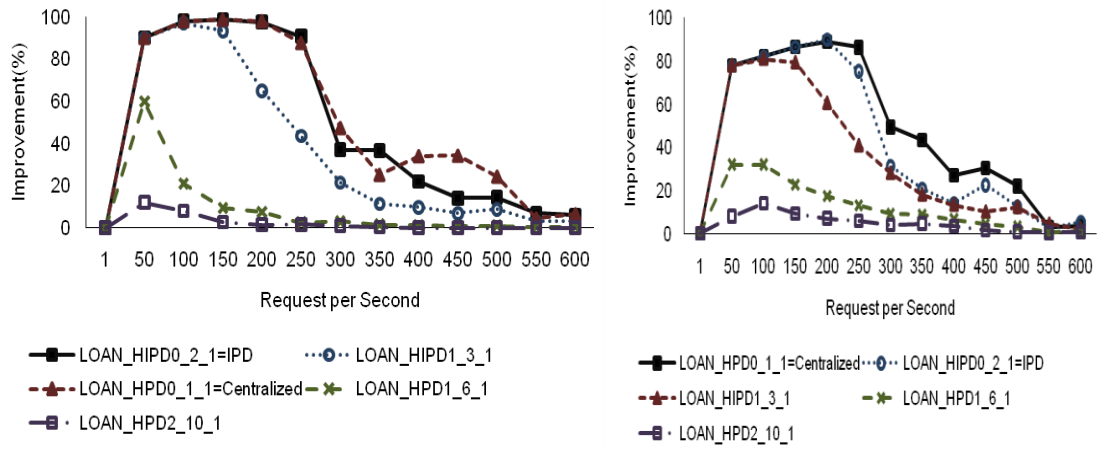
creating less number of fragments and exchanged messages in HPD1_6_1 and HPD2_10_1 fragment sets so that both of them improved the FPD.



Minimum Distribution **Maximum Distribution**

Figure 6.52: Loan-process Throughput Comparison, Type-2, Variable-Request-Rae, Constant-Message-Size, Min-Max distributions

This experiment also showed that calling various decentralized versions of the Loan-process in different rates, in absence of message size, resulted in improving the Loan-process FPD version when the fragments were executed by both architectures Type-1 and Type-2. Throughput improvement in architecture Type-1 was more than Type-2, which substantiated the resiliency of Type-2 comparing to Type-1 in executing different fragment types. Type-2 allocated separate memory quota for each performer agent while in Type-1 it was a shared space.



Type-1 **Type-2**

Figure 6.53: Type-1 and Type-2, Loan-process Throughput improvement

Table 6.14: Average Throughput Improvement of Loan Fragments Compared to FPD

	LOAN_HPD0_1_1 (Centralized)	LOAN_HIPD0_2_1 (IPD)	LOAN_HIPD1_3_1	LOAN_HPD1_6_1	LOAN_HPD2_10_1
Type-1	54.03%	50.99%	54.03%	9.01%	2.25%
Type-2	50.06%	43.38%	35.61%	12.56%	4.98%

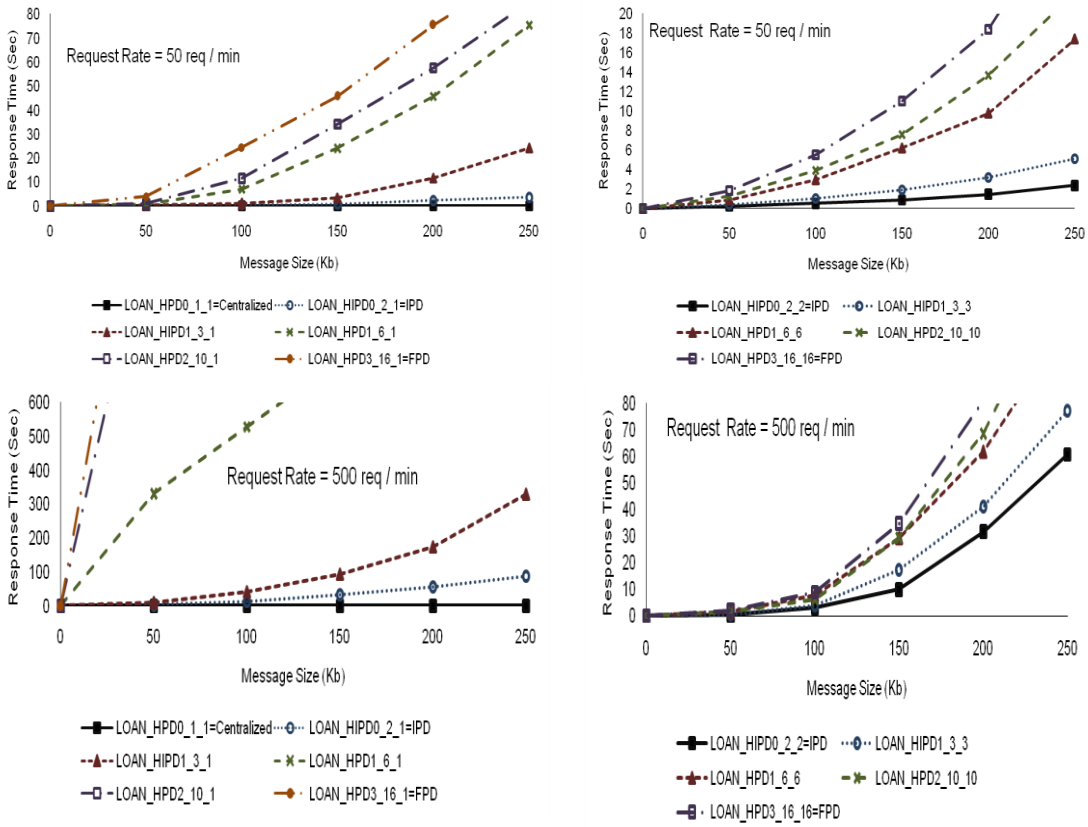
Experiment-8 (frequent-path adaptability, response-time evaluation, constant-request-rate/variable-message-size)

This experiment was conducted according to Experiment-8 specified in Section 4.8.2. Figure 6.54 and Figure 6.55 show response-time behavior of the Loan-process fragments in Min-Max distributions running on the architecture Type-1 and Type-2. After running the experiments, the execution results of different fragments were compared against the FPD.

Type-1: Figure 6.54 shows response-time behavior of the loan fragments in Min-Max distributions running on the architecture Type-1. In addition, Figure 6.56 shows the percent of response-time improvement of the loan fragments compared to the FPD fragments. According to Table 6.15, in request rate 500, the average improvement of response-time by the Centralized, IPD and HIPD1_3_1 was 99.23%, 9.99% and 97.39%, respectively. The improvement trend was directly proportional to the number of fragments and the size of messages exchanged among them. In comparison, HPD1_6_1 and HPD2_10_1 improved the average response-time about 81.86% and 16.85%. Consequently, the lower number of fragments and size of exchanged messages among the Loan-process activities, the better response-time for the process.

Type-2: Figure 6.55 shows response-time behavior of the loan fragments in Min-Max distributions running on the architecture Type-2. In addition, Figure 6.56 shows the percent of response-time improvement of the loan fragments compared to the

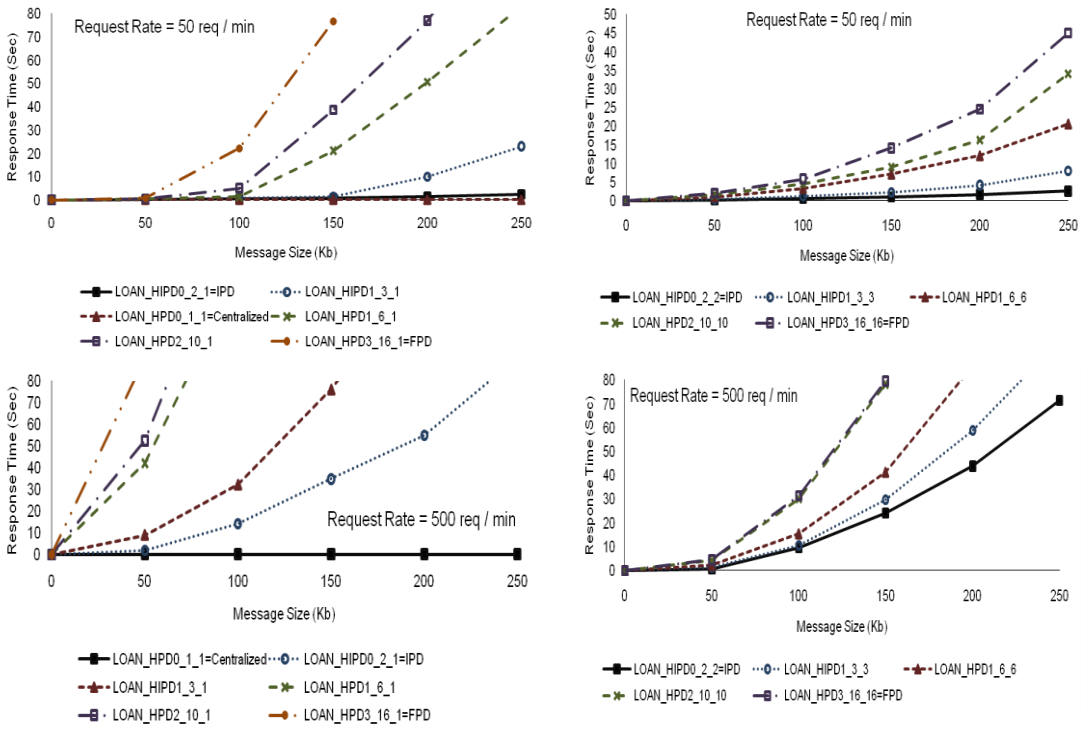
FPD fragments. According to Table 6.15, in request rate 500, the average improvement of response-time by the Centralized, IPD and HIPD1_3_1 was 99.95%, 95.55% and 88.81%, respectively. The trend of improvement was directly proportional to the number of fragments and the size of messages exchanged among them. In comparison, HPD1_6_1 and HPD2_10_1 improved the average response-time about 56.40% and 42.56%. Consequently, the lower number of fragments and size of exchanged messages among the Loan-process activities, the better response-time for the process.



Minimum Distribution **Maximum Distribution**

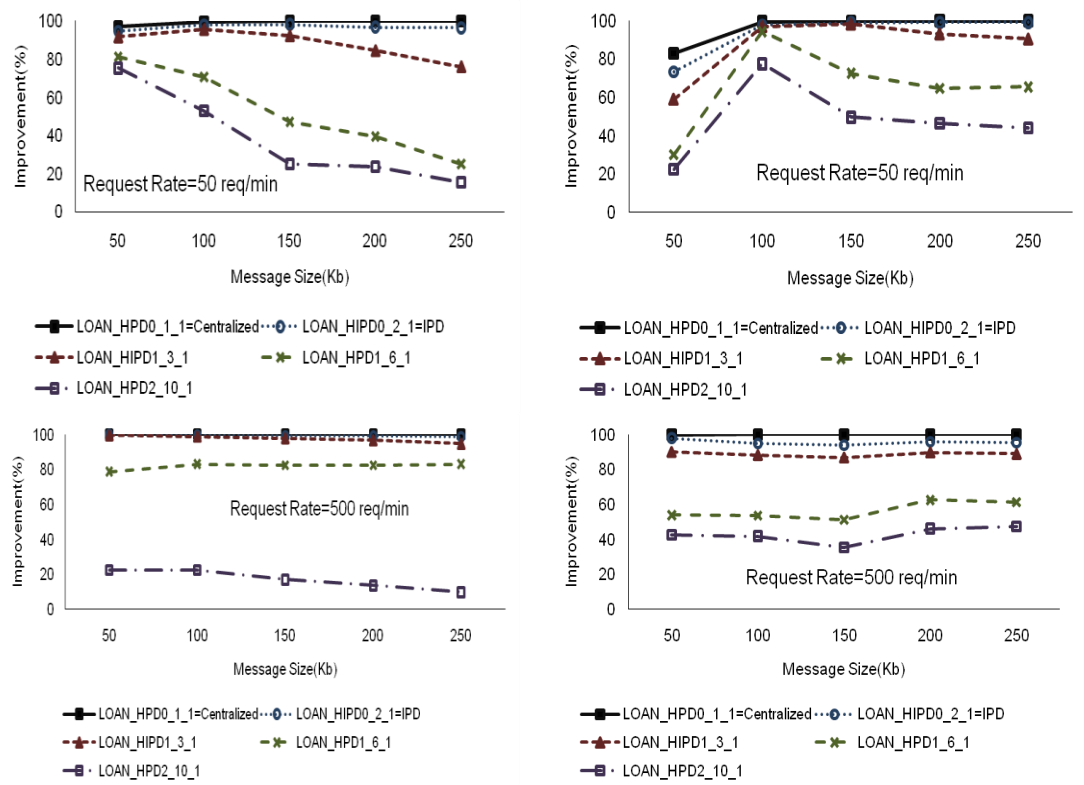
Figure 6.54: Loan-process Response-Time Comparison, Type-1, Constant-Request-Rate, Variable-Message-Size, Min-Max distributions

The number of fragments and the size of exchanged messages among process activities impacted the results, effectively. The FPD was improved in both architectures; particularly, when the request rate and the size of messages were increased. It showed that the frequent-path and proportional-fragment adaptability



Minimum Distribution **Maximum Distribution**

Figure 6.55: Loan-process Response-Time Comparison, Type-2, Constant-Request-Rate, Variable-Message-Size, Min-Max distributions



Type-1 Type-2

Figure 6.56: Type-1 and Type-2, Loan-process Response-time Improvement with Request Rates 50 and 500

worked well for a complex business process. Exceptionally, in HPD2_10_1, the improvement of the FPD in Type-1 was more than Type-2. This case showed that Type-1 is more resilient when huge-size messages with high request rates were sent to servers. The number of produced agents and long messages queues in Type-2 resulted in reducing the performance of the architecture Type-2.

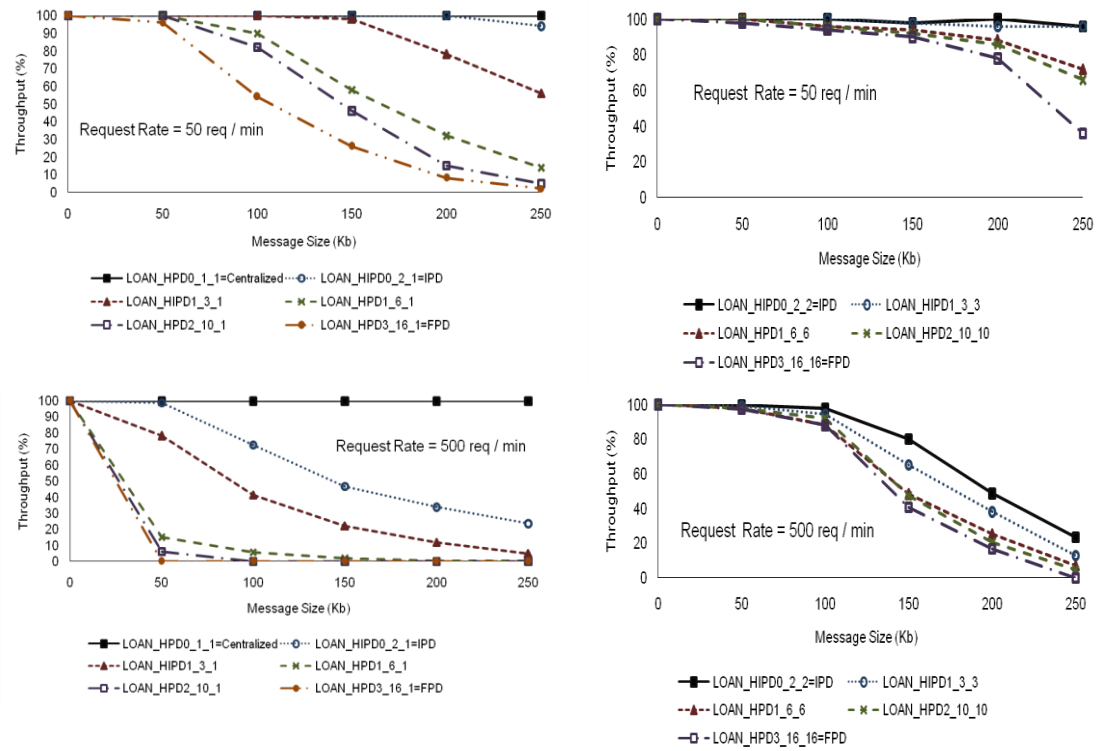
Table 6.15: Average Response-time Improvement of Loan Fragments Compared to FPD

	Req/ Min	LOAN_HPD0_1_1 (Centralized)	LOAN_HIPD0_2_1 (IPD)	LOAN_HIPD1_3_1	LOAN_HPD1_6_1	LOAN_HPD2_10_1
Response-time	**50**	99.20%	96.79%	87.97%	52.66%	38.44%
	500	99.23%	99.99%	97.39%	81.86%	16.85%
Type-1						
	Req/ Min	**LOAN_HPD0_1_1 (Centralized)**	**LOAN_HIPD0_2_1 (IPD)**	**LOAN_HIPD1_3_1**	**LOAN_HPD1_6_1**	**LOAN_HPD2_10_1**
Response-time	**50**	96.40%	93.59%	87.51%	65.37%	47.96%
	500	99.95%	95.55%	88.81%	56.40%	42.56%
Type-2						

Experiment-9 (frequent-path adaptability, evaluating throughput, constant-request-rate, variable-message-size)

This experiment was conducted according to the Experiment-9 specified in Section 4.8.2. Figure 6.57 and Figure 6.58 show response-time behavior of the loan fragments in Min-Max distributions running on the architecture Type-1 and Type-2. After running the experiments, the execution results of different fragments were compared against the FPD.

Type-1: Figure 6.57 shows throughput behavior of the Loan-process fragments in Min-Max distributions running on the architecture Type-1. In addition, Figure 6.59 shows the percent of throughput improvement for the decentralization methods compared to the FPD fragments. In request rate 500, the average improvement of throughput by the Centralized, IPD and HIPD1_3_1 was 62.80%, 61.60% and 49.20%, respectively. In comparison, HPD1_6_1 and HPD2_10_1 improved the average throughput around 21.60% and 12.40%.

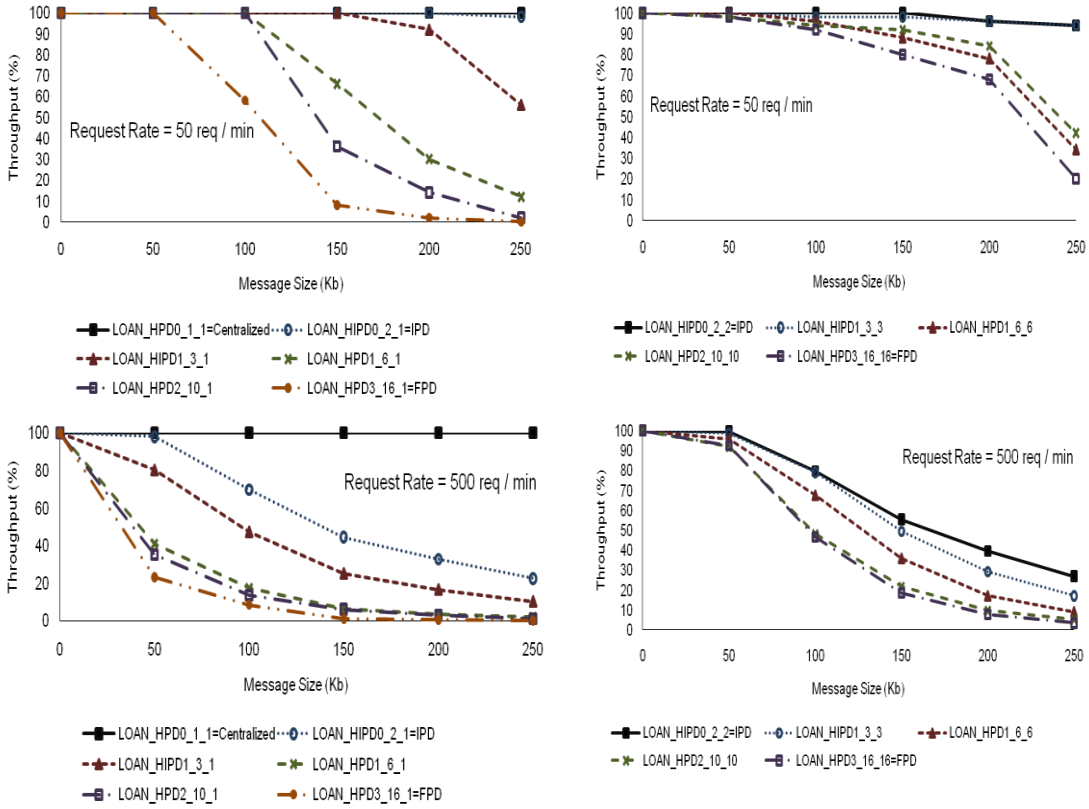


Minimum Distribution **Maximum Distribution**

Figure 6.57: Loan-process Throughput Comparison, Type-1, Constant-Request-Rate, Variable-Message-Size, in Min-Max distributions

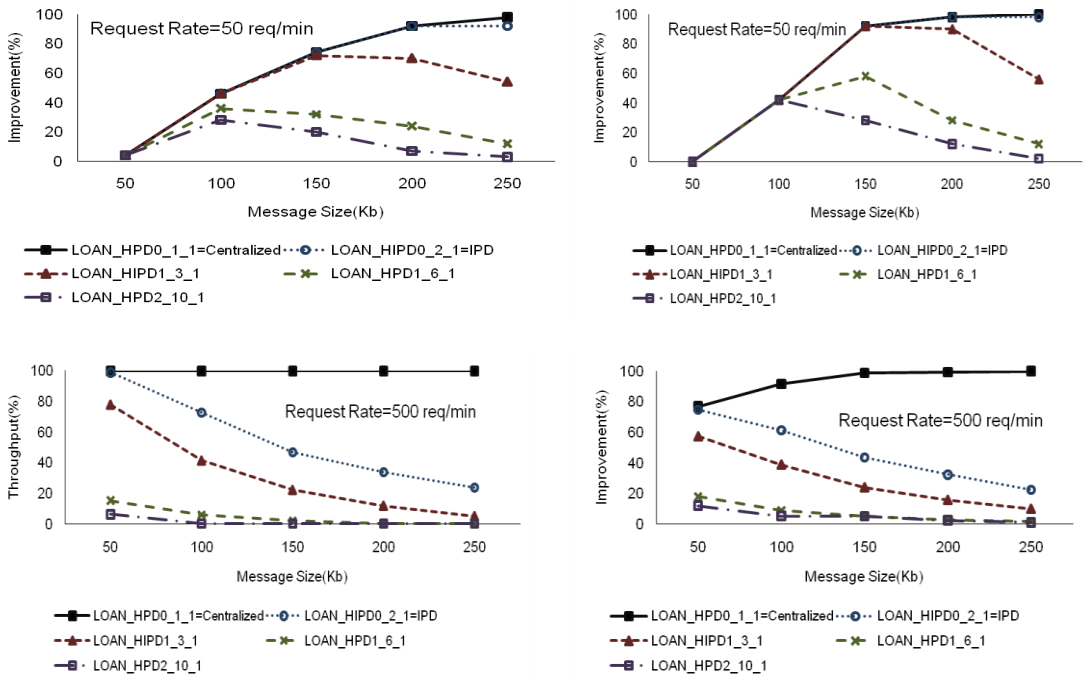
Type-2: Figure 6.58 shows throughput behavior of the Loan-process fragments in Min-Max distributions running on the architecture Type-2. In addition, Figure 6.59 shows the percent of throughput improvement of the decentralization methods compared to the FPD method. In request rate 500, the average improvement of throughput by the Centralized, IPD and HIPD1_3_1 was 93.20%, 46.80% and 29.16%, respectively. In comparison, HPD1_6_1 and HPD2_10_1 improved the average throughput about 7.28% and 5.12%.

The number of fragments, number of exchanged messages and size of messages exchanged among process activities were effective in this experiment. The FPD was improved in both architectures; particularly, when the request rate and the size of messages were increased. The throughput was also improved due to considering the frequent-path and proportional-fragment adaptability. Exceptionally, in HPD2_10_1,



Minimum Distribution **Maximum Distribution**

Figure 6.58: Loan-process Throughput Comparison, Type-2, Constant-Message-Size, Variable-Message-Sizes, Min-Max distributions



Type-1 **Type-2**

Figure 6.59:Type-1 and Type-2, Loan-process Throughput Improvement, Variable-Message-Size, Constant-Request-Rate 50 and 500

the improvement of the FPD in Type-1 was more than Type-2. This case showed that Type-1 is more resilient when huge-size messages with high request rates were sent to servers. The number of produced agents in Type-2 and long message queues resulted in weaker results.

Table 6.16: Average Throughput Improvement of Loan Fragments Compared to FPD

	Req / Min	LOAN_HPD0_1_1 (Centralized)	LOAN_HIPD0_2_1 (IPD)	LOAN_HIPD1_3_1	LOAN_HPD1_6_1	LOAN_HPD2_10_1
Throughput	**50**	62.80%	61.60%	49.20%	21.60%	12.40%
	500	99.8%	55.12%	31.68%	4.64%	1.24%
Type-1						
	Req / Min	**LOAN_HPD0_1_1 (Centralized)**	**LOAN_HIPD0_2_1 (IPD)**	**LOAN_HIPD1_3_1**	**LOAN_HPD1_6_1**	**LOAN_HPD2_10_1**
Throughput	**50**	66.4%	66.00%	56.00	28.00%	16.80%
	500	93.20%	46.80%	29.16%	7.28%	5.12%
Type-2						

Experiment-10 (frequent-path adaptability, evaluating bandwidth-usage, variable-request-rate/constant-message-size)

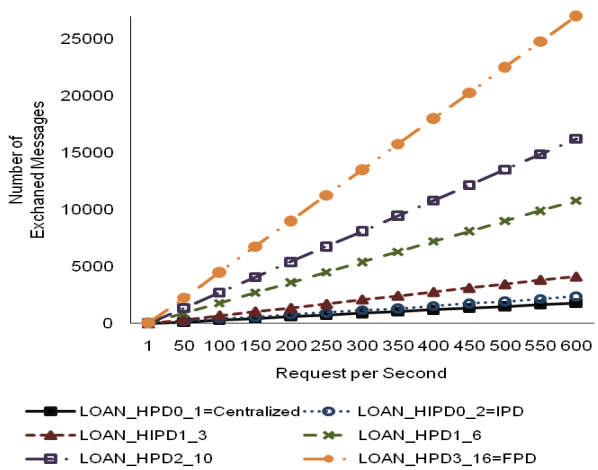


Figure 6.60: Exchanged Message Comparison of Loan-process Fragments

This experiment was conducted according to the Experiment-10 specified in Section 4.8.2. Figure 6.60 shows the comparison of number of exchanged messages by the loan fragments produced by different decentralization methods. The Centralized as well as HIPD methods were considerably similar due to the less number of fragments. As much the HPD granularity level increased, the number of exchanged messages increased as well. The FPD method showed the worst number of message

passing while the Centralized and HIPD were the best methods. The HIPD fragments applied the frequent-path adaptability, which is the reason why the Centralized and HIPD fragments had similar results.

Experiment-11 (proportional-fragment adaptability, response-time and throughput evaluation, variable-request-rate/constant-message-size)

This experiment was conducted according to Experiment-11 specified in Section 4.8.2. It elaborated the proportional-fragment adaptability, which was analogous to the relation between the number of produced fragments and the number of workflow engine machines, when request rate was variable and message size was around zero.

The main goal was to work out what number of fragments must be provided when a specific number of machines exist. In this experiment, the response-time and throughput of the mentioned decentralization methods were compared when a specific number of machines were available. Indeed, the loan application process was fragmented and executed on various numbers of machines in both architectures Type-1 and Type-2. For each experiment only those fragment sets were selected that their number of fragments were equal to the number of dedicated machines.

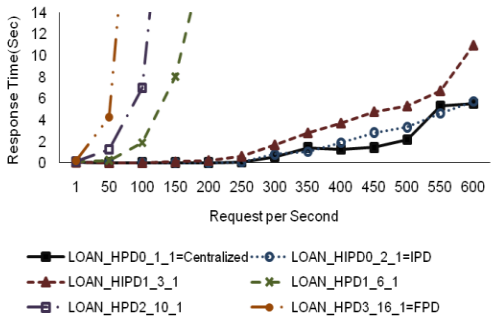
Type-1: Figure 6.61 shows the response-time comparison of the loan fragments. In addition, Table 6.17 and Figure 6.62 illustrate the response-time improvement of the FPD by the other loan fragments. The FPD response-time was improved [62.25%-99.96%] in Type-1 based on the available machines. Table 6.18 and Figure 6.63 show the throughput improvement of the FPD by the other loan fragments. The FPD throughput was improved [21.93%-61.47%] in terms of the number of machines.

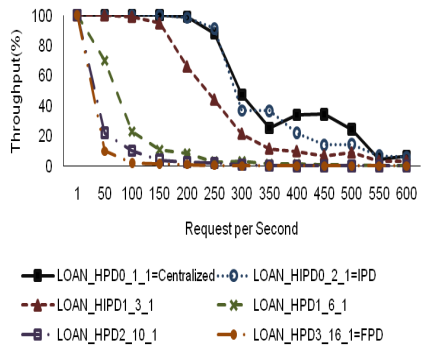
Type-2: Figure 6.64 shows the response-time comparison of the loan fragments. In addition, Table 6.19 and Figure 6.65 illustrate the response-time improvement of the FPD by the other loan fragments. The FPD response-time was improved [9.60%-

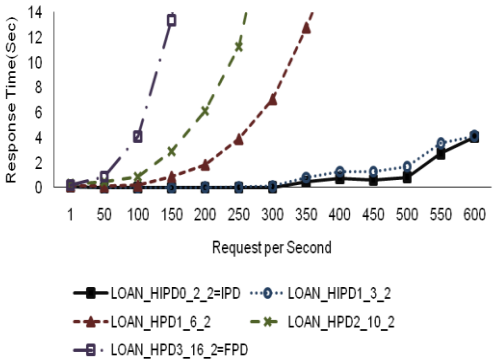
75.14%] based on the available machines. Table 6.20 and Figure 6.66 show the throughput improvement of the FPD by the other loan fragments. The FPD throughput was improved [9.60% - 75.14%] in terms of the number of machines.

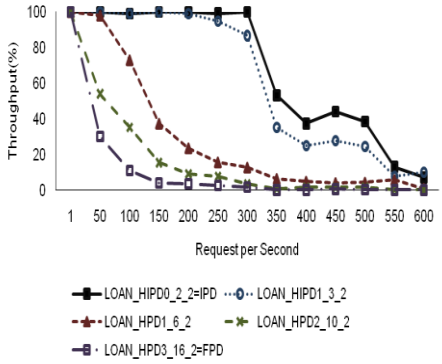
Increasing the number of machines affected the system response-time and throughput in two ways. First, it added more resources such as CPU, memory, etc. to the whole system; second, it substituted the internal message passing with the external one. The former empowered the total system while the later was a communication constraint and degraded the system; particularly, when the request rate was high. Anticipating the balance of the above aspects was not straightforward. In other words, in order to achieve the scalability, the external message passing was unavoidable, which was the reason why after increasing the number of machines, the improvement of the FPD response-time and throughput decreased.

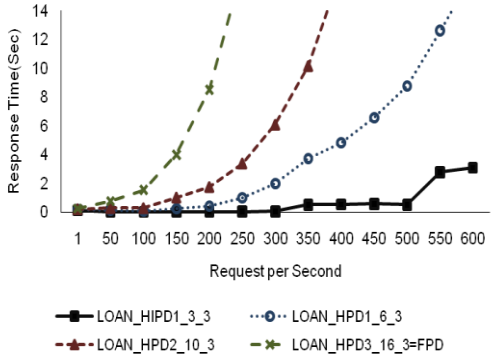
An immediate conclusion was omitting the internal message passing among the fragments. It came about when the number of fragments was proportional to the number of machines dedicated to the workflow engine or according to the ADWEF definition (5.1), for each business process the fragments satisfying the condition $|F| \approx |M|$ were more proportional fragment than others.

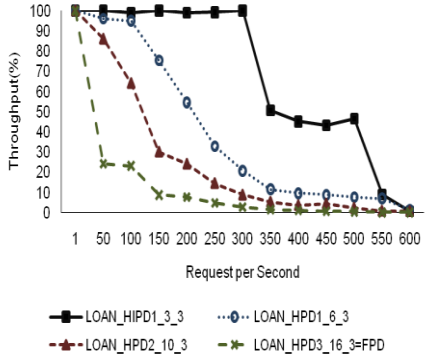

Response Time(Sec)
Request per Second
LOAN_HPD0_1_1=Centralized
LOAN_HIPD0_2_1=IPD
LOAN_HIPD1_3_1
LOAN_HPD1_6_1
LOAN_HPD2_10_1
LOAN_HPD3_16_1=FPD

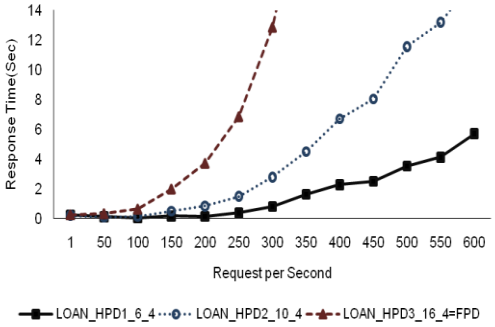

Throughput(%)
Request per Second
LOAN_HPD0_1_1=Centralized
LOAN_HIPD0_2_1=IPD
LOAN_HIPD1_3_1
LOAN_HPD1_6_1
LOAN_HPD2_10_1
LOAN_HPD3_16_1=FPD

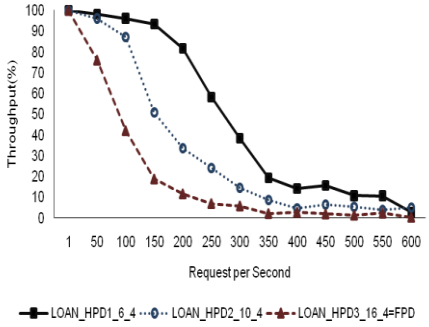

Response Time(Sec)
Request per Second
LOAN_HIPD0_2_2=IPD
LOAN_HIPD1_3_2
LOAN_HPD1_6_2
LOAN_HPD2_10_2
LOAN_HPD3_16_2=FPD

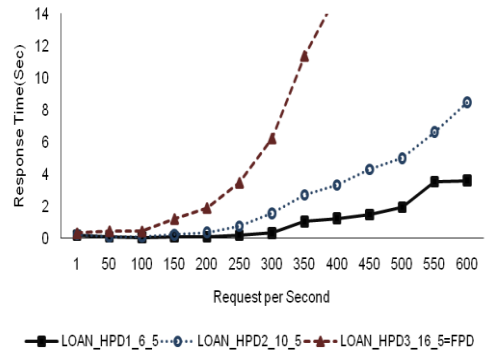

Throughput(%)
Request per Second
LOAN_HIPD0_2_2=IPD
LOAN_HIPD1_3_2
LOAN_HPD1_6_2
LOAN_HPD2_10_2
LOAN_HPD3_16_2=FPD

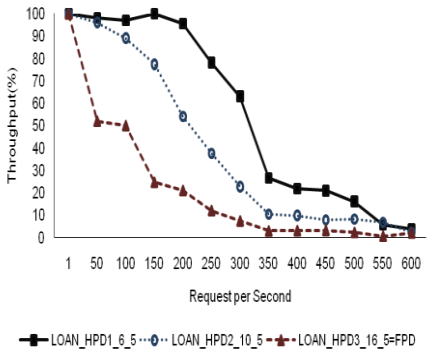

Response Time(Sec)
Request per Second
LOAN_HIPD1_3_3
LOAN_HPD1_6_3
LOAN_HPD2_10_3
LOAN_HPD3_16_3=FPD


Throughput(%)
Request per Second
LOAN_HIPD1_3_3
LOAN_HPD1_6_3
LOAN_HPD2_10_3
LOAN_HPD3_16_3=FPD


Response Time(Sec)
Request per Second
LOAN_HPD1_6_4
LOAN_HPD2_10_4
LOAN_HPD3_16_4=FPD


Throughput(%)
Request per Second
LOAN_HPD1_6_4
LOAN_HPD2_10_4
LOAN_HPD3_16_4=FPD


Response Time(Sec)
Request per Second
LOAN_HPD1_6_5
LOAN_HPD2_10_5
LOAN_HPD3_16_5=FPD


Throughput(%)
Request per Second
LOAN_HPD1_6_5
LOAN_HPD2_10_5
LOAN_HPD3_16_5=FPD

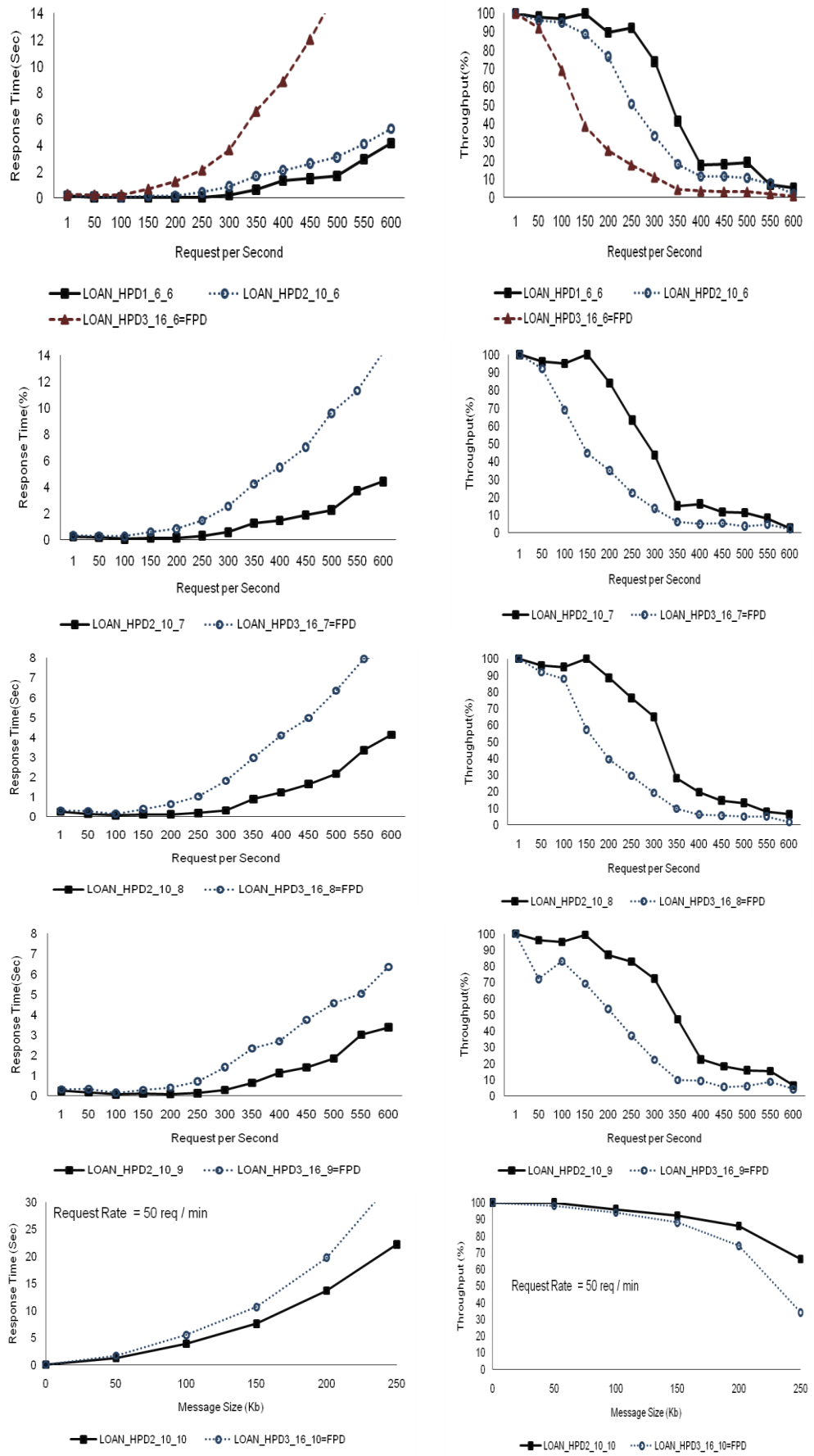


Figure 6.61: Fragment Proportionality of Loan fragments in terms of Response-time and Throughput, Type-1, Variable-Request-Rate, Constant-Message-Size

Table 6.17: Average Percentage of Response-time Improvement by Loan Fragments in Architecture Type-1

		Average Response-Time Improvement				
Number of Machines	1	LOAN_HPD0_1_1 (Centralized)	LOAN_HIPD0_2_1 (IPD)	LOAN_HIPD1_3_1	LOAN_HPD1_6_1	LOAN_HPD2_10_1
		99.96%	99.95%	99.90%	92.13%	71.53%
	2		LOAN_HIPD0_2_2 (IPD)	LOAN_HIPD1_3_2	LOAN_HPD1_6_2	LOAN_HPD2_10_2
			99.76%	99.63%	94.29%	79.73%
	3			LOAN_HIPD1_3_3	LOAN_HPD1_6_3	LOAN_HPD2_10_3
				99.24%	93.44%	79.40%
	4				LOAN_HPD1_6_4	LOAN_HPD2_10_4
					91.61%	79.36%
	5				LOAN_HPD1_6_5	LOAN_HPD2_10_5
					91.94%	80.33%
	6				LOAN_HPD1_6_6	LOAN_HPD2_10_6
					87.39%	75.28%
	7					LOAN_HPD2_10_7
						71.97%
	8					LOAN_HPD2_10_8
						66.99%
	9					LOAN_HPD2_10_9
						63.51%
	10					LOAN_HPD2_10_10
						62.25%

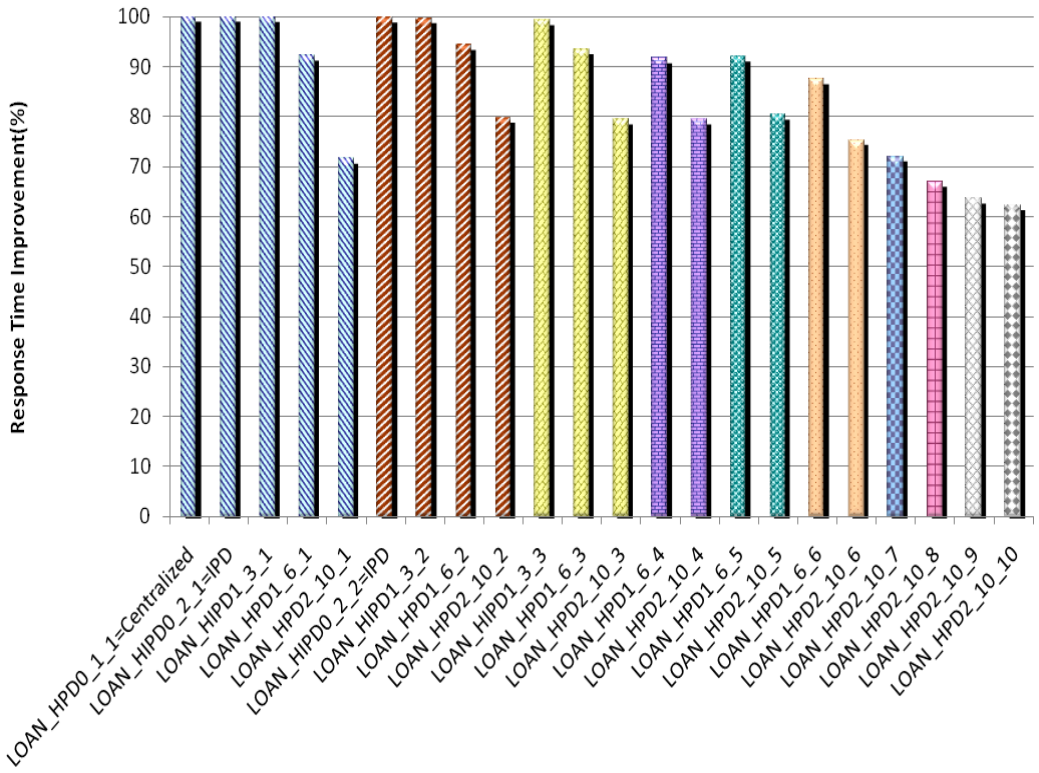


Figure 6.62: Average Response-time Improvement, Type-1, Variable-Request-Rate, Constant-Message-Size

Table 6.18: Average Percentage of Throughput Improvement by Loan Fragments in Architecture Type-1

		Average Throughput Improvement				
Number of Machines	1	LOAN_HPD0_1_1 (Centralized)	LOAN_HIPD0_2_1 (IPD)	LOAN_HIPD1_3_1	LOAN_HPD1_6_1	LOAN_HPD2_10_1
		54.03%	50.99%	37.80%	9.01%	2.25%
	2		LOAN_HIPD0_2_2 (IPD)	LOAN_HIPD1_3_2	LOAN_HPD1_6_2	LOAN_HPD2_10_2
			61.47%	54.66%	19.38%	6.35%
	3			LOAN_HIPD1_3_3	LOAN_HPD1_6_3	LOAN_HPD2_10_3
				60.91%	28.83%	14.12%
	4				LOAN_HPD1_6_4	LOAN_HPD2_10_4
					30.59%	14.02%
	5				LOAN_HPD1_6_5	LOAN_HPD2_10_5
					37.12%	20.08%
	6				LOAN_HPD1_6_6	LOAN_HPD2_10_6
					32.31%	19.22%
	7					LOAN_HPD2_10_7
						20.24%
	8					LOAN_HPD2_10_8
						20.97%
	9					LOAN_HPD2_10_9
						23.07%
	10					LOAN_HPD2_10_10
						21.93%

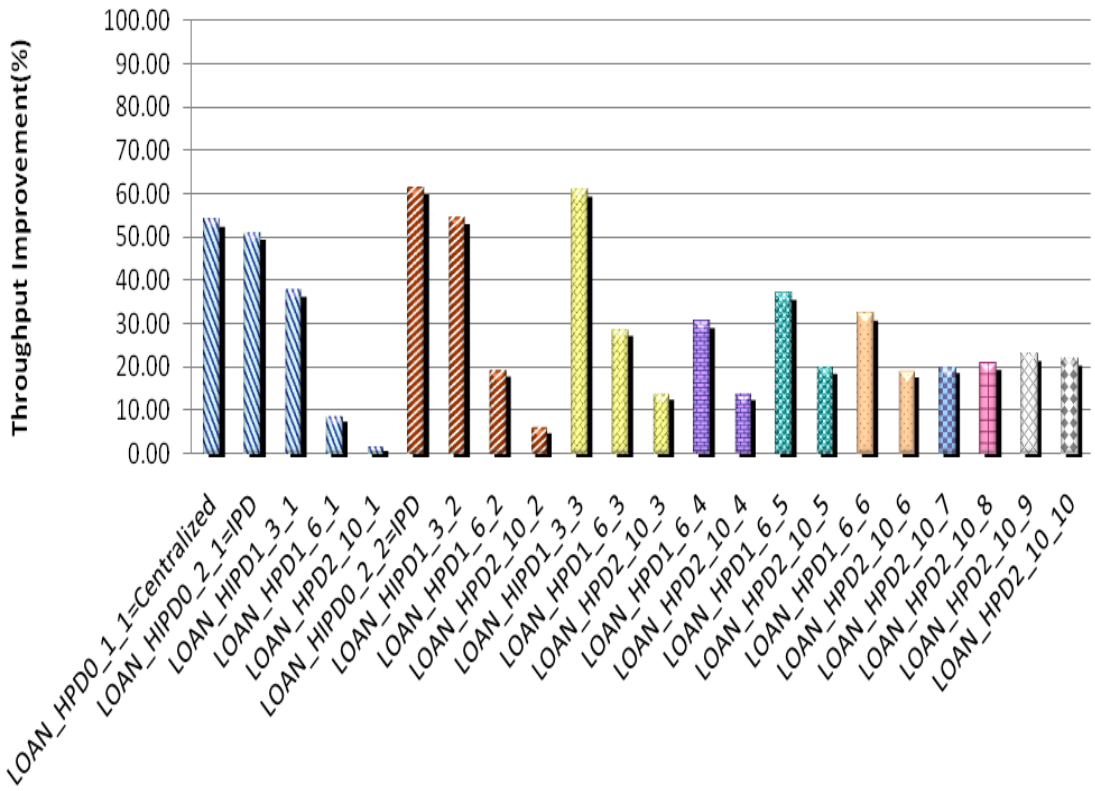
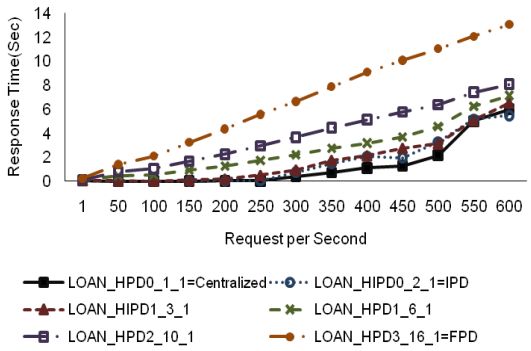
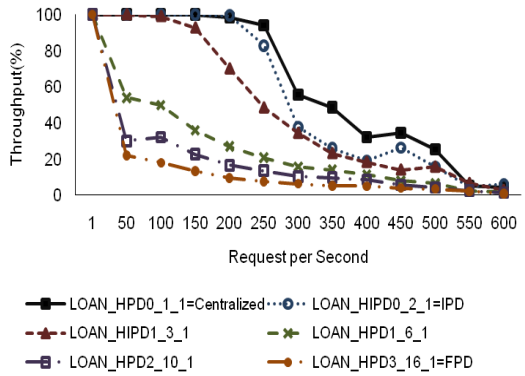
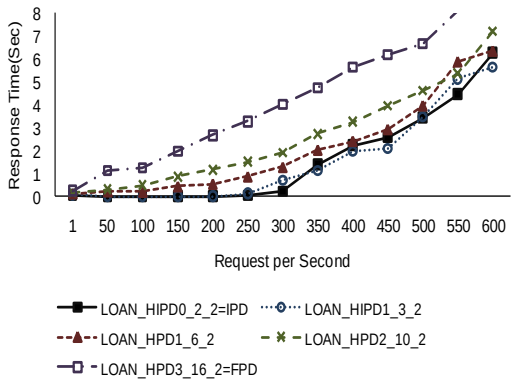
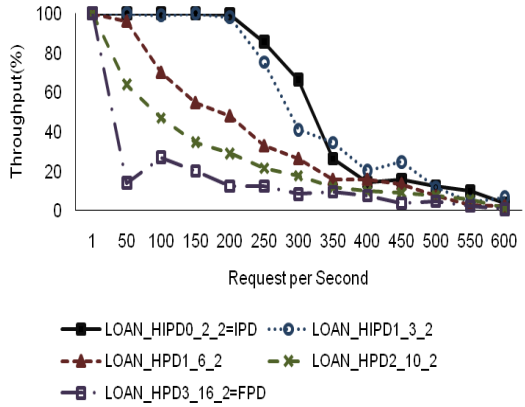


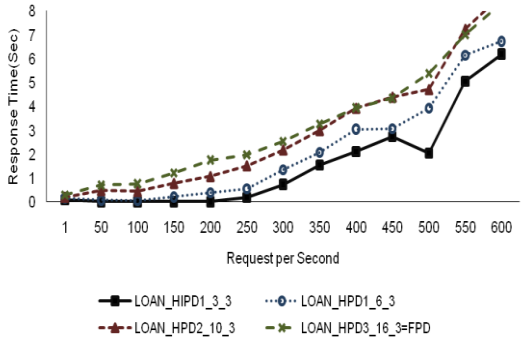
Figure 6.63: Average Throughput Improvement, Type-1, Variable-Request-Rate, Constant-Message-Size

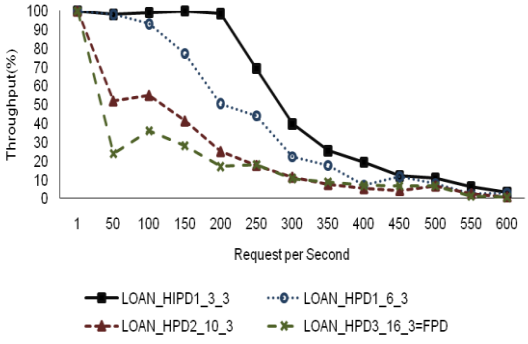

Response Time(Sec)
Request per Second
LOAN_HPD0_1_1=Centralized
LOAN_HIPD0_2_1=IPD
LOAN_HIPD1_3_1
LOAN_HPD1_6_1
LOAN_HPD2_10_1
LOAN_HPD3_16_1=FPD

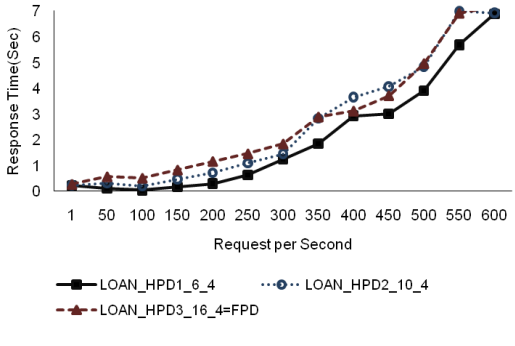

Throughput(%)
Request per Second
LOAN_HPD0_1_1=Centralized
LOAN_HIPD0_2_1=IPD
LOAN_HIPD1_3_1
LOAN_HPD1_6_1
LOAN_HPD2_10_1
LOAN_HPD3_16_1=FPD

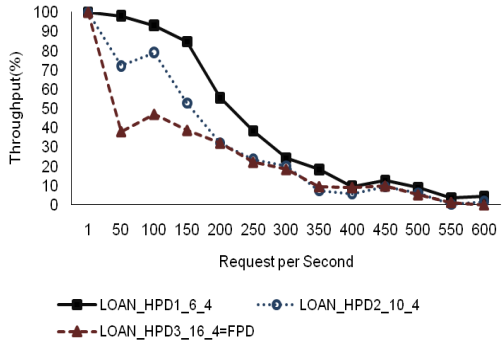

Response Time(Sec)
Request per Second
LOAN_HIPD0_2_2=IPD
LOAN_HIPD1_3_2
LOAN_HPD1_6_2
LOAN_HPD2_10_2
LOAN_HPD3_16_2=FPD

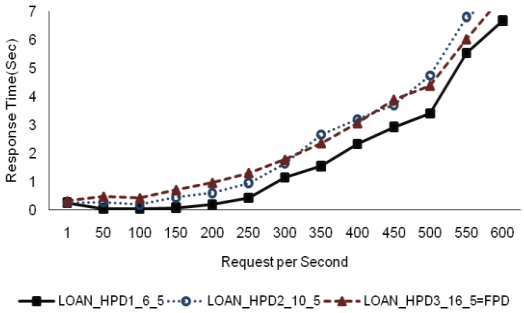

Throughput(%)
Request per Second
LOAN_HIPD0_2_2=IPD
LOAN_HIPD1_3_2
LOAN_HPD1_6_2
LOAN_HPD2_10_2
LOAN_HPD3_16_2=FPD

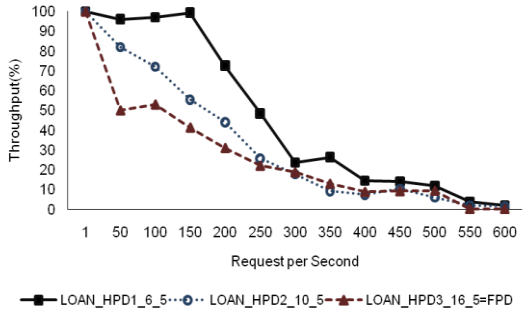

Response Time(Sec)
Request per Second
LOAN_HIPD1_3_3
LOAN_HPD1_6_3
LOAN_HPD2_10_3
LOAN_HPD3_16_3=FPD


Throughput(%)
Request per Second
LOAN_HIPD1_3_3
LOAN_HPD1_6_3
LOAN_HPD2_10_3
LOAN_HPD3_16_3=FPD


Response Time(Sec)
Request per Second
LOAN_HPD1_6_4
LOAN_HPD2_10_4
LOAN_HPD3_16_4=FPD


Throughput(%)
Request per Second
LOAN_HPD1_6_4
LOAN_HPD2_10_4
LOAN_HPD3_16_4=FPD


Response Time(Sec)
Request per Second
LOAN_HPD1_6_5
LOAN_HPD2_10_5
LOAN_HPD3_16_5=FPD


Throughput(%)
Request per Second
LOAN_HPD1_6_5
LOAN_HPD2_10_5
LOAN_HPD3_16_5=FPD

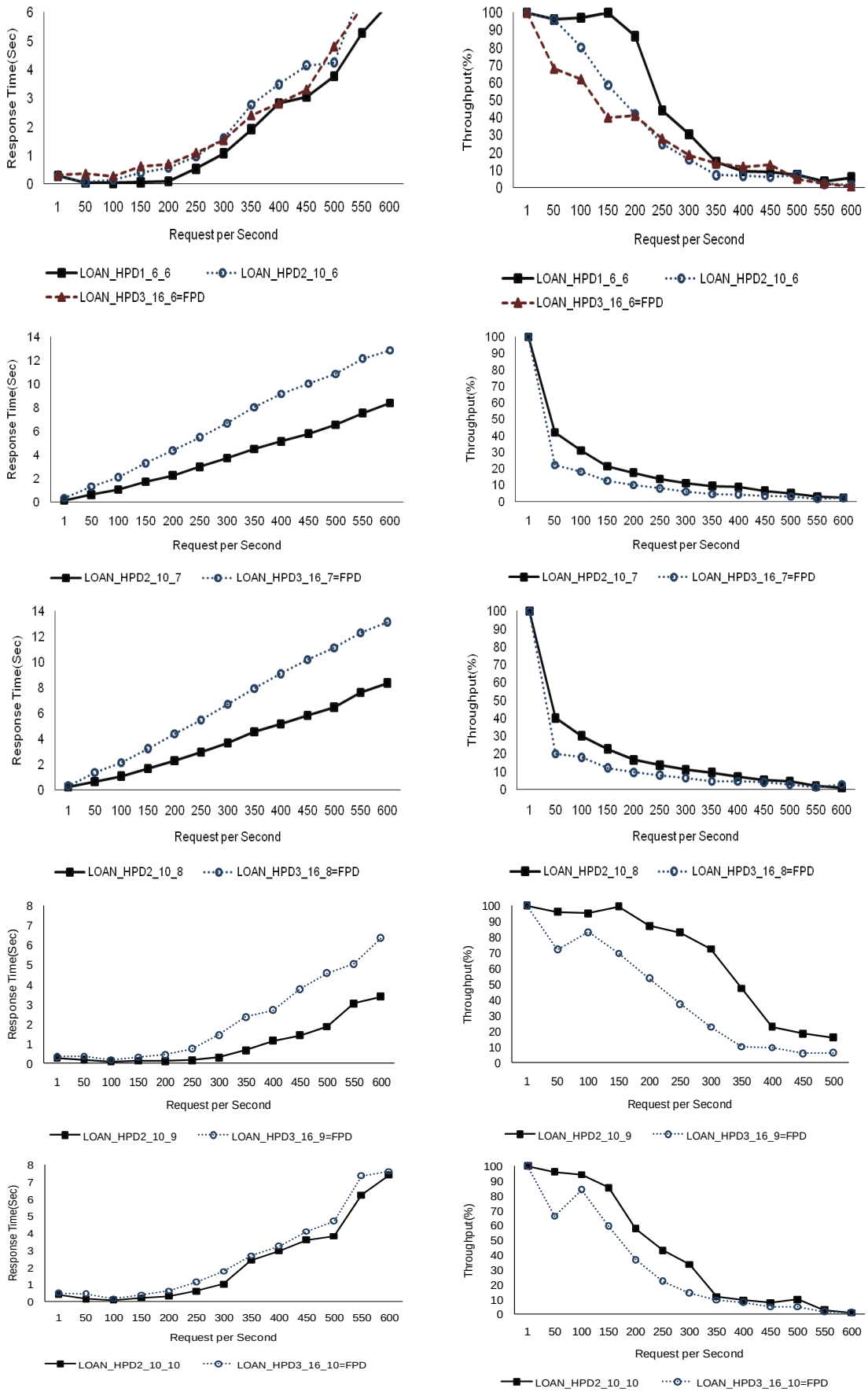


Figure 6.64: Loan-process Fragment Proportionality in terms of Response-time and Throughput, Type-2, Variable-Request-Rate, Constant-Message-Size

Table 6.19: Average Percentage of Response-time Improvement by Loan Fragments in Architecture Type-2

		Average Response-time Improvement				
Number of Machines	1	LOAN_HPD0_1_1 (Centralized)	LOAN_HIPD0_2_1 (IPD)	LOAN_HIPD1_3_1	LOAN_HPD1_6_1	LOAN_HPD2_10_1
		87.61%	84.27%	81.24%	63.98%	44.27%
	2		LOAN_HIPD0_2_2 (IPD)	LOAN_HIPD1_3_2	LOAN_HPD1_6_2	LOAN_HPD2_10_2
			75.13%	75.14%	60.22%	46.11%
	3			LOAN_HIPD1_3_3	LOAN_HPD1_6_3	LOAN_HPD2_10_3
				67.17%	50.43%	16.66%
	4				LOAN_HPD1_6_4	LOAN_HPD2_10_4
					42.94%	17.60%
	5				LOAN_HPD1_6_5	LOAN_HPD2_10_5
					48.30%	14.36%
	6				LOAN_HPD1_6_6	LOAN_HPD2_10_6
					42.13%	9.60%
	7					LOAN_HPD2_10_7
						44.26%
	8					LOAN_HPD2_10_8
						44.59%
	9					LOAN_HPD2_10_9
						44.88%
	10					LOAN_HPD2_10_10
						32.02%

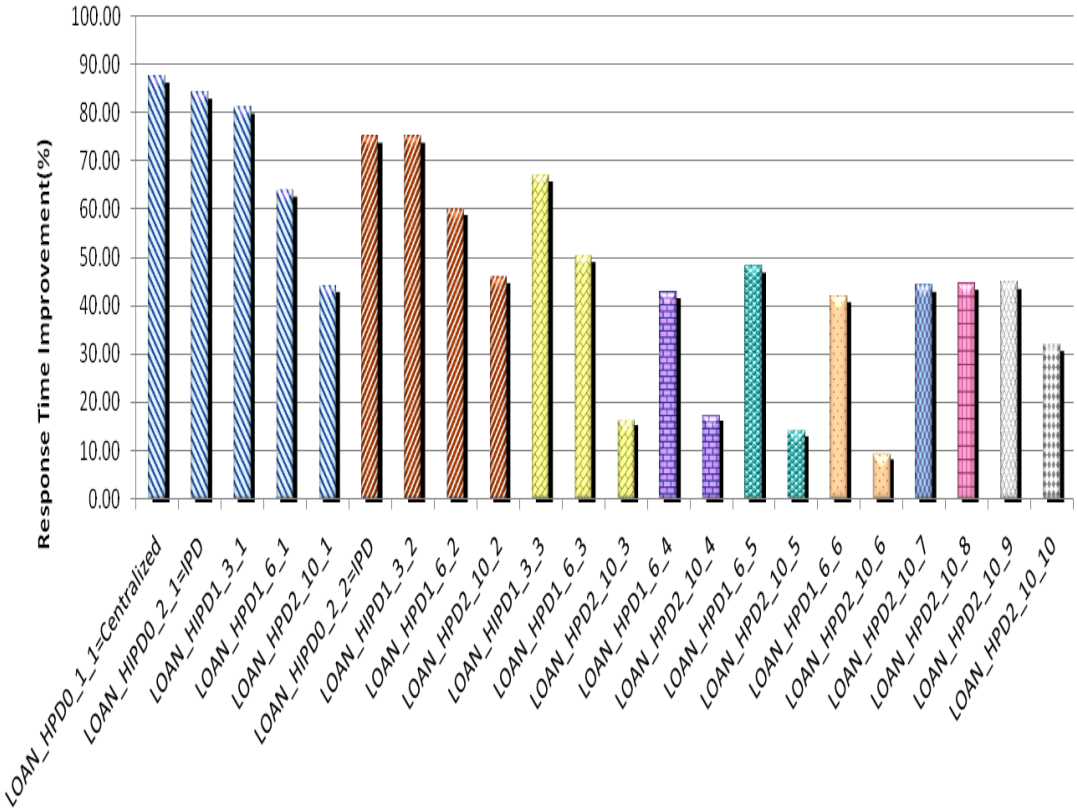


Figure 6.65: Average Response-time Improvement in Architecture Type-2, Variable-Request-Rate, Constant-Message-Size

Table 6.20: Average Percentage of Throughput Improvement by Loan Fragments in Architecture Type-2

		Average Throughput Improvement				
Number of Machines	1	LOAN_HPD0_1_1 (Centralized)	LOAN_HIPD0_2_1 (IPD)	LOAN_HIPD1_3_1	LOAN_HPD1_6_1	LOAN_HPD2_10_1
		50.06%	43.38%	35.61%	12.56%	4.98%
	2		LOAN_HIPD0_2_2 (IPD)	LOAN_HIPD1_3_2	LOAN_HPD1_6_2	LOAN_HPD2_10_2
			42.52%	41.09%	21.88%	11.34%
	3			LOAN_HIPD1_3_3	LOAN_HPD1_6_3	LOAN_HPD2_10_3
				36.06%	22.51%	5.37%
	4				LOAN_HPD1_6_4	LOAN_HPD2_10_4
					18.35%	6.57%
	5				LOAN_HPD1_6_5	LOAN_HPD2_10_5
					21.04%	6.34%
	6				LOAN_HPD1_6_6	LOAN_HPD2_10_6
					16.60%	3.69%
	7					LOAN_HPD2_10_7
						6.34%
	8					LOAN_HPD2_10_8
						5.78%
	9					LOAN_HPD2_10_9
						6.51%
	10					LOAN_HPD2_10_10
						11.62%

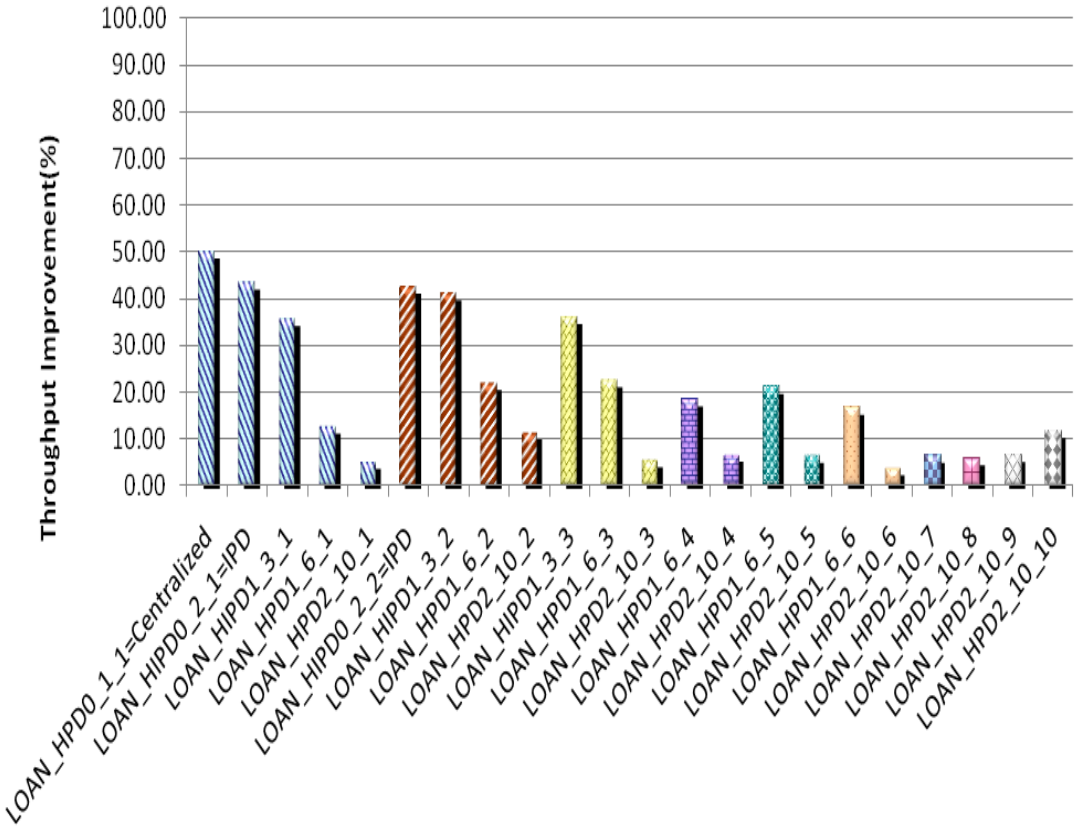


Figure 6.66: Average Throughput Improvement in Architecture Type-2, Variable-Request-Rate, Constant-Message-Size

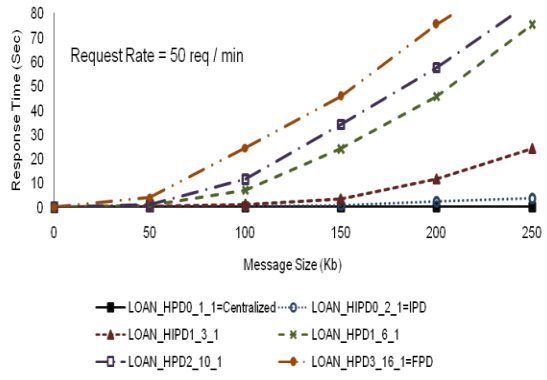
Experiment-12 (proportional-fragment adaptability, response-time evaluation, constant-request-rate/variable-message-size)

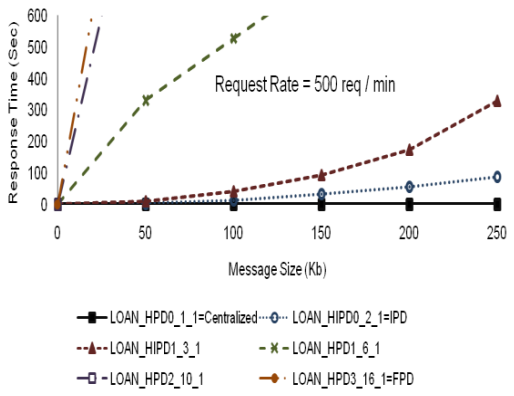
This experiment was conducted according to Experiment-12 specified in Section 4.8.2. Accordingly, the loan application process was fragmented and executed on various numbers of machines in both architectures Type-1 and Type-2. For each experiment only those fragment sets were selected that their number of fragments were equal to the number of workflow engine machines.

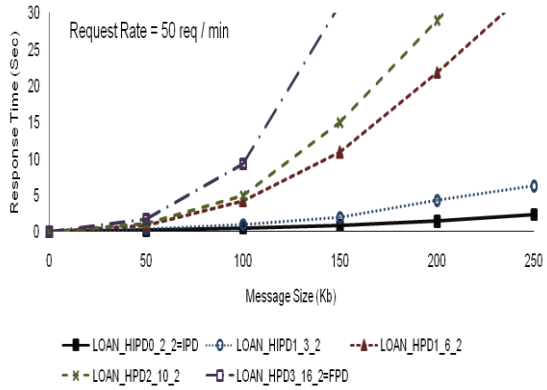
Figure 6.67 shows the response-time comparison of the loan fragments in Type-1. Table 6.21 and Figure 6.68 also show the response-time improvement of the FPD by the other loan fragments. The FPD response-time was improved in range [16.85%-100%]. Moreover, Figure 6.69 also show the response-time comparison of the loan fragments in Type-2. Table 6.22 and Figure 6.70 also show the response-time improvement of the FPD loan fragments by the other loan fragments. In this case, the FPD response-time was also improved in a range of [21.52%-99.96%].

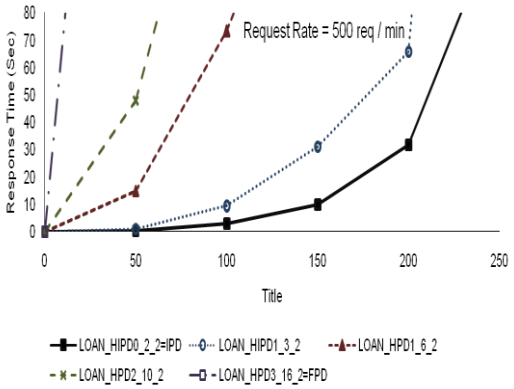
Increasing the number of machines in constant-request-rate and variable-message-size affected the system response-time in two ways. First, it was equal to applying more resources such as CPU, memory, etc.; second, it substituted the internal message passing with the external one. The former provided more system resources while the latter was a communication constraint and degraded the system. Anticipating, the balance of the above aspects was not straightforward. On the other hand, in order to obtain the scalability, the external message passing was unavoidable. In both architectures, after increasing the number of machines, due to increasing the external message passing and huge-size messages, the improvement of the FPD response-time was less.

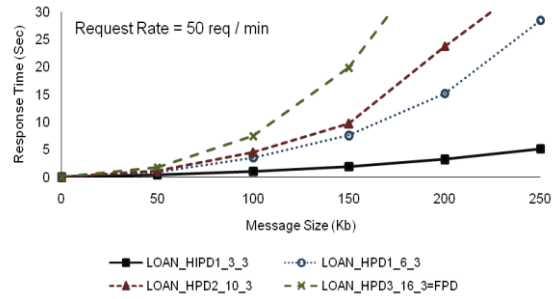
An immediate conclusion was omitting the internal messages among the fragments. It was realized when the number of fragments was proportional to the number of machines dedicated to the workflow engine or according to the ADWEF definition (5.1) for each business process the fragments satisfying the condition $|F| \approx |M|$ were more proportional fragment than others.

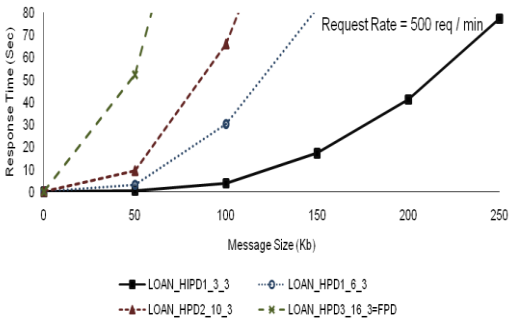

Request Rate = 50 req / min
Response Time (Sec)
Message Size (Kb)
LOAN_HPD0_1_1=Centralized
LOAN_HIPD0_2_1=IPD
LOAN_HIPD1_3_1
LOAN_HPD1_6_1
LOAN_HPD2_10_1
LOAN_HPD3_16_1=FPD

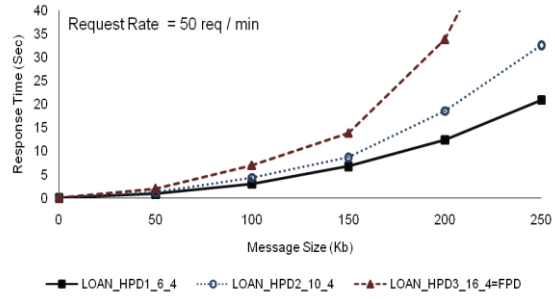

Request Rate = 500 req / min
Response Time (Sec)
Message Size (Kb)
LOAN_HPD0_1_1=Centralized
LOAN_HIPD0_2_1=IPD
LOAN_HIPD1_3_1
LOAN_HPD1_6_1
LOAN_HPD2_10_1
LOAN_HPD3_16_1=FPD

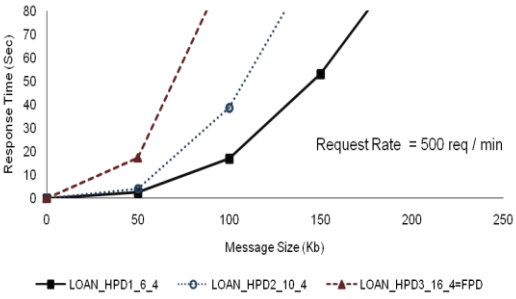

Request Rate = 50 req / min
Response Time (Sec)
Message Size (Kb)
LOAN_HIPD0_2_2=IPD
LOAN_HIPD1_3_2
LOAN_HPD1_6_2
LOAN_HPD2_10_2
LOAN_HPD3_16_2=FPD

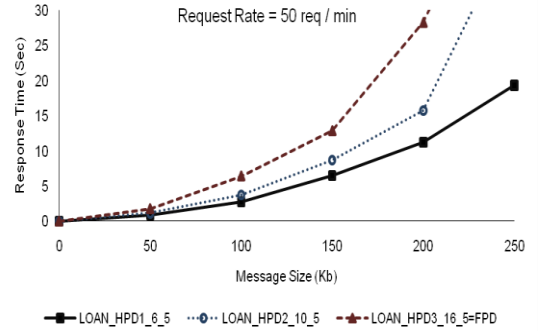

Request Rate = 500 req / min
Response Time (Sec)
Title
LOAN_HIPD0_2_2=IPD
LOAN_HIPD1_3_2
LOAN_HPD1_6_2
LOAN_HPD2_10_2
LOAN_HPD3_16_2=FPD

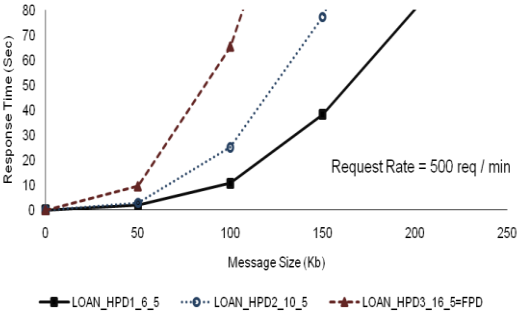

Request Rate = 50 req / min
Response Time (Sec)
Message Size (Kb)
LOAN_HIPD1_3_3
LOAN_HPD1_6_3
LOAN_HPD2_10_3
LOAN_HPD3_16_3=FPD


Request Rate = 500 req / min
Response Time (Sec)
Message Size (Kb)
LOAN_HIPD1_3_3
LOAN_HPD1_6_3
LOAN_HPD2_10_3
LOAN_HPD3_16_3=FPD


Request Rate = 50 req / min
Response Time (Sec)
Message Size (Kb)
LOAN_HPD1_6_4
LOAN_HPD2_10_4
LOAN_HPD3_16_4=FPD


Request Rate = 500 req / min
Response Time (Sec)
Message Size (Kb)
LOAN_HPD1_6_4
LOAN_HPD2_10_4
LOAN_HPD3_16_4=FPD


Request Rate = 50 req / min
Response Time (Sec)
Message Size (Kb)
LOAN_HPD1_6_5
LOAN_HPD2_10_5
LOAN_HPD3_16_5=FPD


Request Rate = 500 req / min
Response Time (Sec)
Message Size (Kb)
LOAN_HPD1_6_5
LOAN_HPD2_10_5
LOAN_HPD3_16_5=FPD

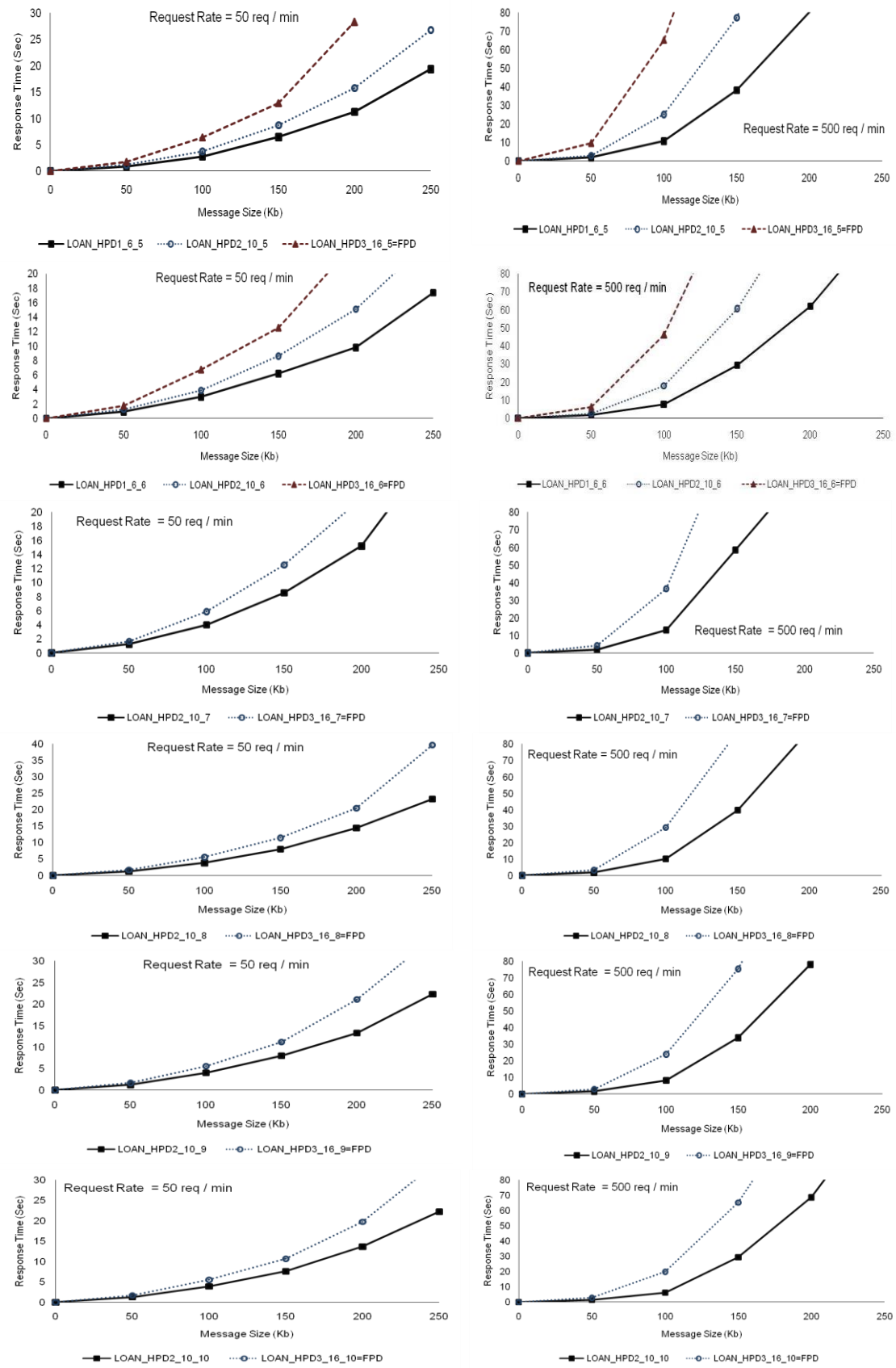


Figure 6.67: Loan-process Fragment Proportionality in terms of Response-time, Type-1, Constant-Request-Rate, Variable-Message-Size

Table 6.21: Average Percentage of FPD Response-time Improvement by HPD and HIPD methods, Type-1, Constant-Request-Rate, Variable-Message-Size

		Average Response-time Improvement									
Number of Machines	1	LOAN_HPD0_1_1 (Centralized)		LOAN_HIPD0_2_1 (IPD)		LOAN_HIPD1_3_1		LOAN_HPD1_6_1		LOAN_HPD2_10_1	
		50KB	500KB	50KB	500KB	50KB	500KB	50KB	500KB	50KB	500KB
		99.20%	100%	96.79%	99.23%	87.97%	97.39%	52.66%	81.86%	38.44%	16.85%
	2			LOAN_HIPD0_2_2 (IPD)		LOAN_HIPD1_3_2		LOAN_HPD1_6_2		LOAN_HPD2_10_2	
				50KB	500KB	50KB	500KB	50KB	500KB	50KB	500KB
				94.69	98.52	88.43	96.73	54.34	84.66	44.85	72.92
	3					LOAN_HIPD1_3_3		LOAN_HPD1_6_3		LOAN_HPD2_10_3	
						50KB	500KB	50KB	500KB	50KB	500KB
						88.22	94.85	58.44	78.48	42.70	54.88
	4							LOAN_HPD1_6_4		LOAN_HPD2_10_4	
								50KB	500KB	50KB	500KB
								59.23	73.96	40.26	39.58
	5							LOAN_HPD1_6_5		LOAN_HPD2_10_5	
								50KB	500KB	50KB	500KB
								56.33	81.14	35.23	56.26
	6							LOAN_HPD1_6_6		LOAN_HPD2_10_6	
								50KB	500KB	50KB	500KB
								55.83	75.62	31.31	56.36
	7									LOAN_HPD2_10_7	
										50KB	500KB
										31.51	55.41
	8									LOAN_HPD2_10_8	
										50KB	500KB
										31.84	51.87
	9									LOAN_HPD2_10_9	
										50KB	500KB
										31.11	53.15
	10									LOAN_HPD2_10_10	
										50KB	500KB
										30.16	52.82

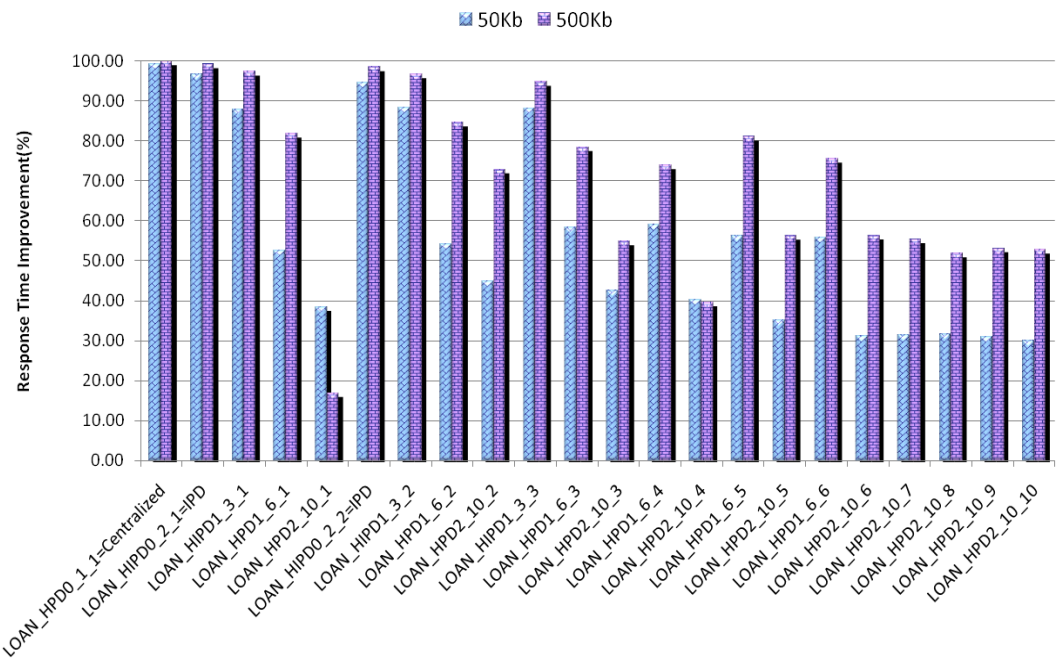
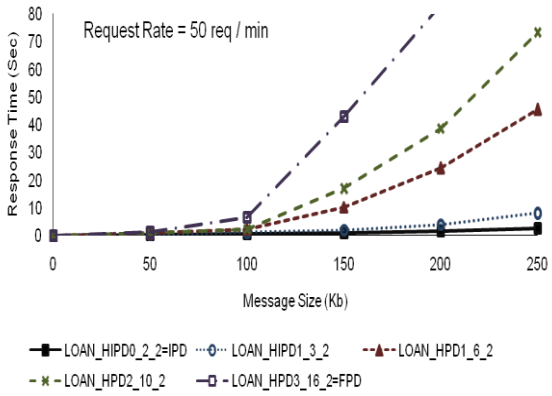
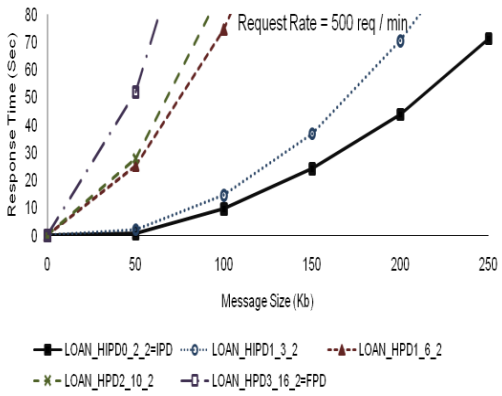
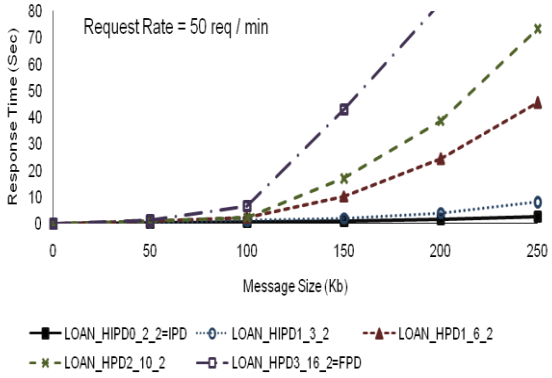
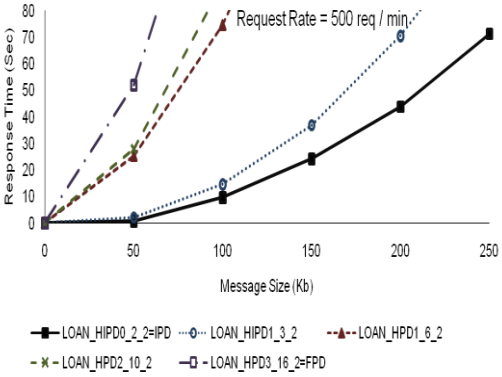


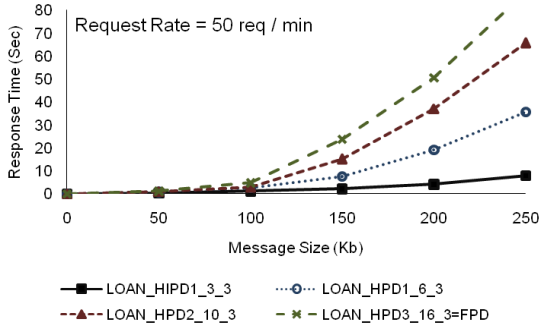
Figure 6.68: Average Percentage of FPD Response-time Improvement by HPD and HIPD, Type-1, with Constant-Request-Rate, Variable-Message-Size

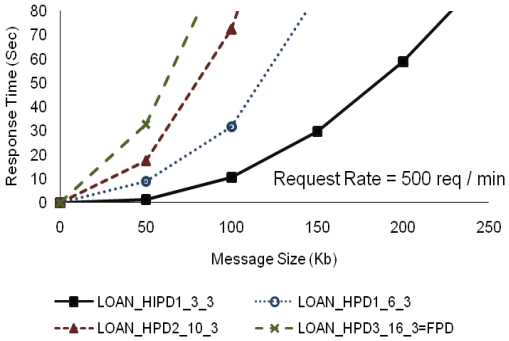

Request Rate = 50 req / min
Response Time (Sec)
Message Size (Kb)
LOAN_HIPD0_2_2=IPD
LOAN_HIPD1_3_2
LOAN_HPD1_6_2
LOAN_HPD2_10_2
LOAN_HPD3_16_2=FPD

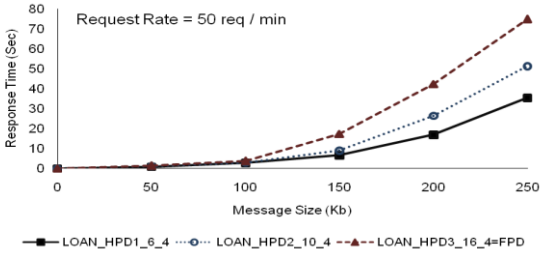

Request Rate = 500 req / min
Response Time (Sec)
Message Size (Kb)
LOAN_HIPD0_2_2=IPD
LOAN_HIPD1_3_2
LOAN_HPD1_6_2
LOAN_HPD2_10_2
LOAN_HPD3_16_2=FPD

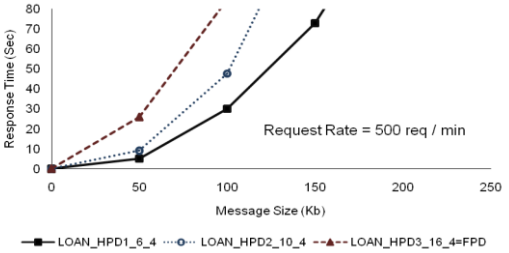

Request Rate = 50 req / min
Response Time (Sec)
Message Size (Kb)
LOAN_HIPD0_2_2=IPD
LOAN_HIPD1_3_2
LOAN_HPD1_6_2
LOAN_HPD2_10_2
LOAN_HPD3_16_2=FPD

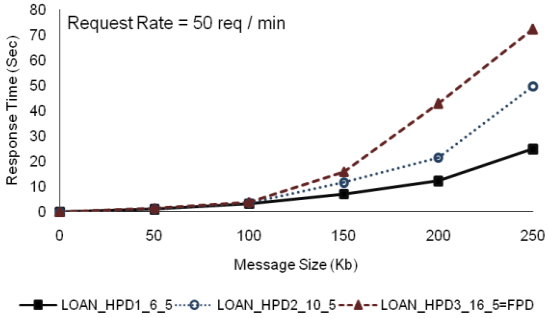

Request Rate = 500 req / min
Response Time (Sec)
Message Size (Kb)
LOAN_HIPD0_2_2=IPD
LOAN_HIPD1_3_2
LOAN_HPD1_6_2
LOAN_HPD2_10_2
LOAN_HPD3_16_2=FPD

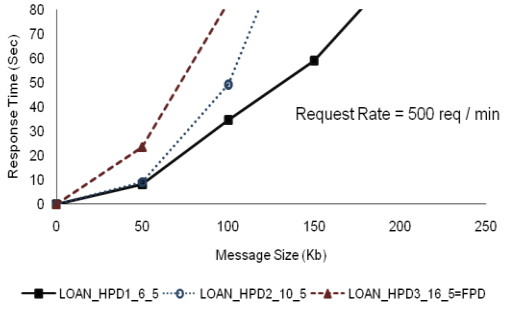

Request Rate = 50 req / min
Response Time (Sec)
Message Size (Kb)
LOAN_HIPD1_3_3
LOAN_HPD1_6_3
LOAN_HPD2_10_3
LOAN_HPD3_16_3=FPD


Request Rate = 500 req / min
Response Time (Sec)
Message Size (Kb)
LOAN_HIPD1_3_3
LOAN_HPD1_6_3
LOAN_HPD2_10_3
LOAN_HPD3_16_3=FPD


Request Rate = 50 req / min
Response Time (Sec)
Message Size (Kb)
LOAN_HPD1_6_4
LOAN_HPD2_10_4
LOAN_HPD3_16_4=FPD


Request Rate = 500 req / min
Response Time (Sec)
Message Size (Kb)
LOAN_HPD1_6_4
LOAN_HPD2_10_4
LOAN_HPD3_16_4=FPD


Request Rate = 50 req / min
Response Time (Sec)
Message Size (Kb)
LOAN_HPD1_6_5
LOAN_HPD2_10_5
LOAN_HPD3_16_5=FPD


Request Rate = 500 req / min
Response Time (Sec)
Message Size (Kb)
LOAN_HPD1_6_5
LOAN_HPD2_10_5
LOAN_HPD3_16_5=FPD

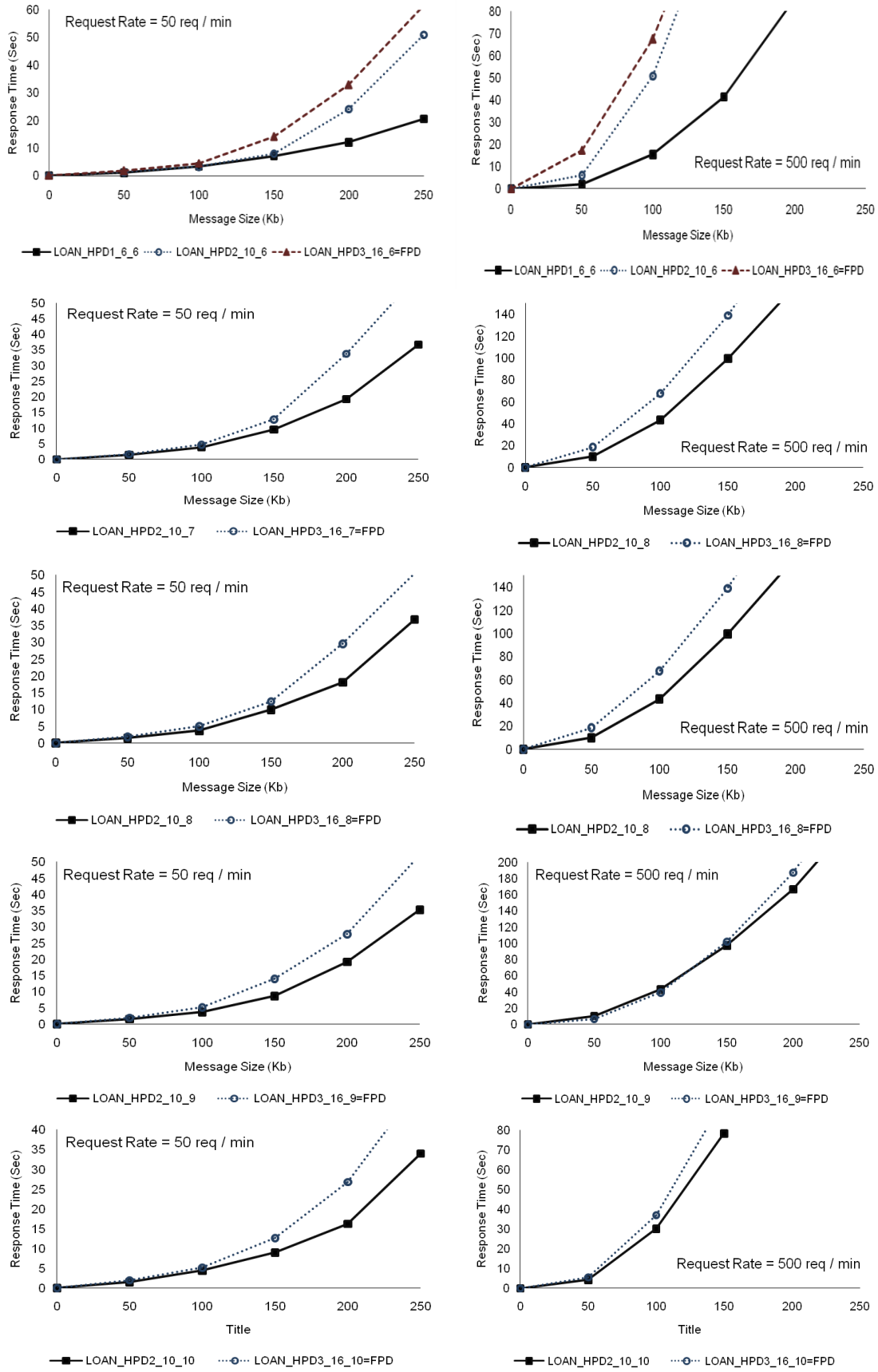


Figure 6.69: Fragment Proportionality of Loan Fragments in terms of Response-time, Type-2, Constant-Request-Rate, Variable-Message-Size

Table 6.22: Loan Average Percentage of FPD Response-time Improvement, by HPD and HIPD, Type-2, Constant-Request-Rate, Variable-Message-Size

		Average Response-time Improvement									
Number of Machines	1	LOAN_HPD0_1_1 (Centralized)		LOAN_HIPD0_2_1 (IPD)		LOAN_HIPD1_3_1		LOAN_HPD1_6_1		LOAN_HPD2_10_1	
		50KB	500KB	50KB	500KB	50KB	500KB	50KB	500KB	50KB	500KB
		96.41	99.96	93.59	95.55	87.52	88.81	65.38	56.40	47.96	42.56
	2			LOAN_HIPD0_2_2 (IPD)		LOAN_HIPD1_3_2		LOAN_HPD1_6_2		LOAN_HPD2_10_2	
				50KB	500KB	50KB	500KB	50KB	500KB	50KB	500KB
				93.21	94.28	86.66	90.64	61.17	56.17	47.72	41.35
	3					LOAN_HIPD1_3_3		LOAN_HPD1_6_3		LOAN_HPD2_10_3	
						50KB	500KB	50KB	500KB	50KB	500KB
						82.68	89.01	52.21	65.66	28.07	29.21
	4							LOAN_HPD1_6_4		LOAN_HPD2_10_4	
								50KB	500KB	50KB	500KB
								46.73	63.89	31.13	35.15
	5							LOAN_HPD1_6_5		LOAN_HPD2_10_5	
								50KB	500KB	50KB	500KB
								48.18	66.36	26.68	32.58
	6							LOAN_HPD1_6_6		LOAN_HPD2_10_6	
								50KB	500KB	50KB	500KB
								47.73	73.96	27.44	21.52
	7									LOAN_HPD2_10_7	
										50KB	500KB
										27.09	39.01
	% 8									LOAN_HPD2_10_8	
										50KB	500KB
										35.42	46.69
	9									LOAN_HPD2_10_9	
										50KB	500KB
										32.87	42.75
	10									LOAN_HPD2_10_10	
										50KB	500KB
										34.11	44.78

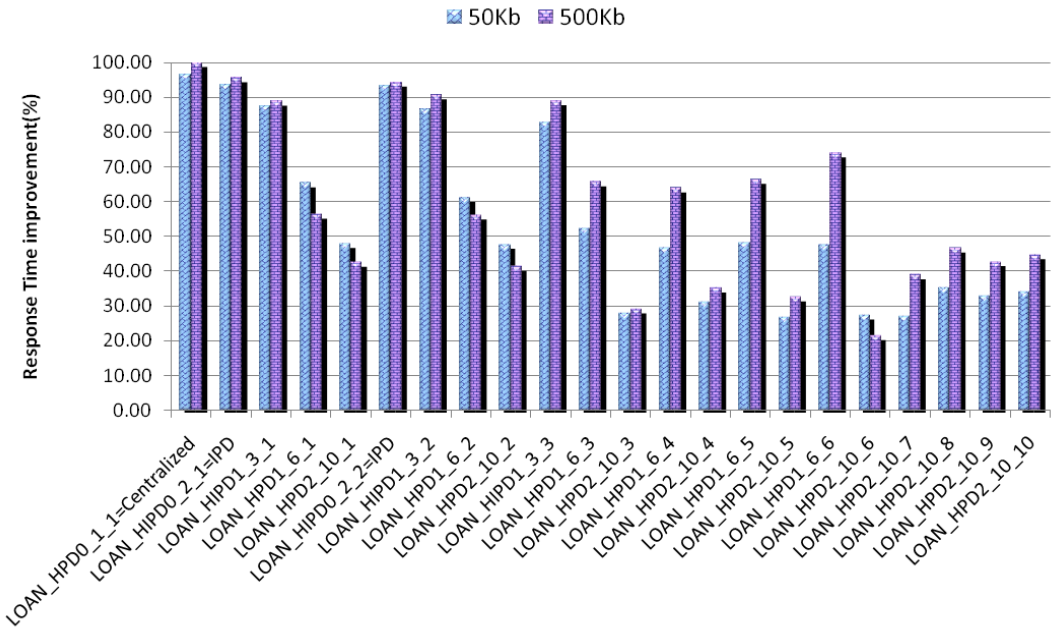


Figure 6.70: Average Percentage of FPD Response-time Improvement by HPD and HIPD, Type-2, Constant-Request-Rate, Variable-Message-Size

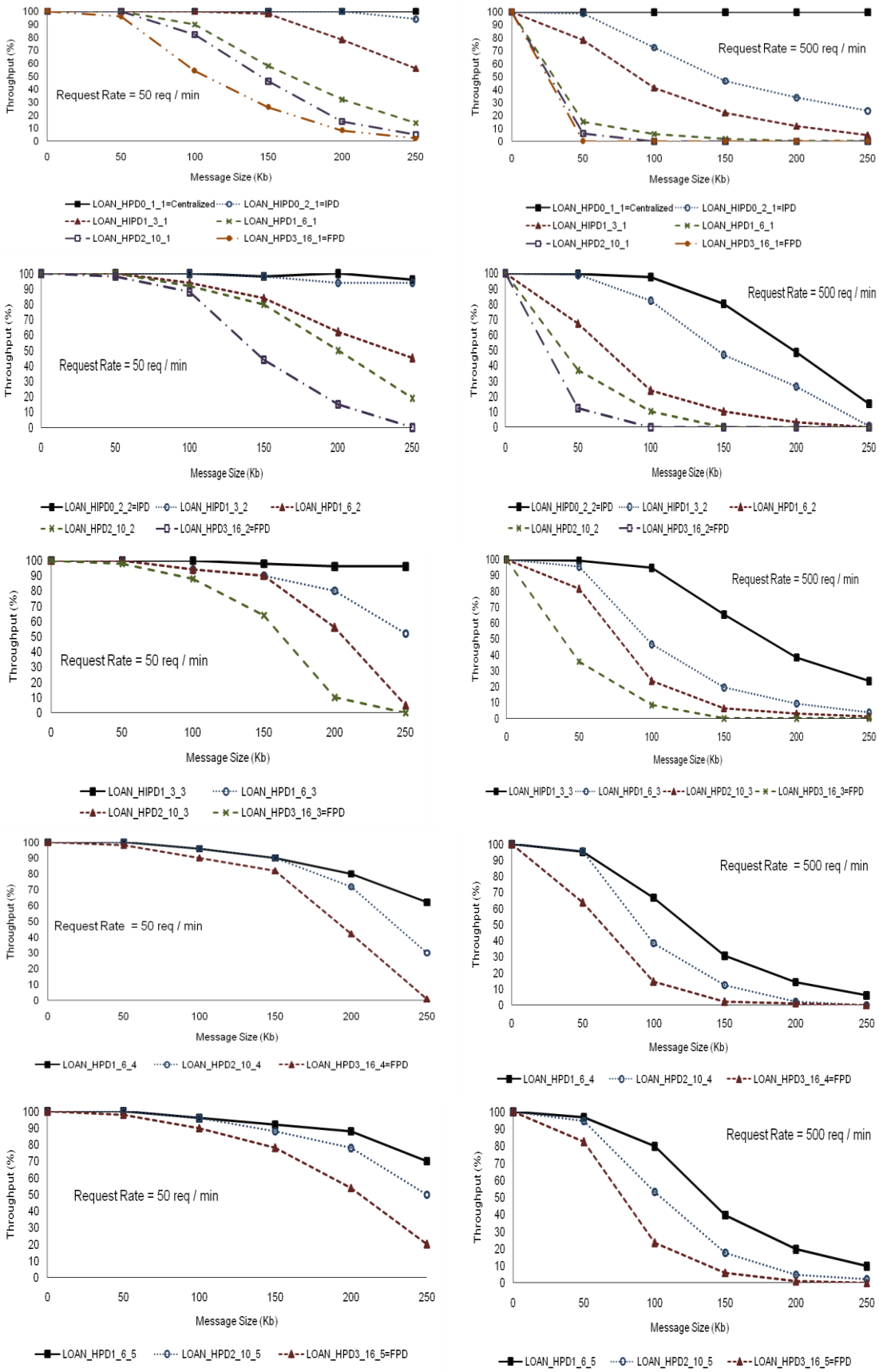
Experiment-13 (proportional-fragment adaptability, throughput evaluation, constant-request-rate/variable-message-size)

This experiment was conducted according to Experiment-13 specified in Section 4.8.2. Accordingly, the loan application process was fragmented and executed on various numbers of machines in both architectures Type-1 and Type-2. For each experiment only those fragment sets were selected that their number of fragments were equal to the number of workflow engine machines.

Figure 6.71 shows the response-time comparison of the loan fragments. Table 6.23 and Figure 6.72 also show the throughput improvement of the FPD by the other loan fragments in Type-1. The FPD throughput was improved in range [1.24% - 99.80%]. In addition, Figure 6.73 also shows the throughput comparison of the loan fragments in Type-2. Table 6.24 and Figure 6.74 also show the throughput improvement of the loan fragments in Type-2. The FPD throughput was improved in range [5.12% - 66.40%].

Increasing the number of machines in constant-request-rate and variable-message-size affected the system throughput in two ways. First, it was equal to applying more resources such as CPU, memory, etc.; second, it substituted the internal message passing with the external one. The former improved the system resources while the latter was a communication constraint and degraded the system. Anticipating, the balance of the above aspects was not straightforward. On the other hand, in order to obtain scalability, the external message passing was unavoidable. In both architectures, by increasing the number of machines due to increasing the number of external message passing and huge-size messages, the improvement of the FPD throughput was less.

An immediate conclusion was omitting internal messages among the fragments. It came about when the number of fragments was proportional to the number of machines dedicated to the workflow engine or according to the ADWEF definition (5.1) for each business process the fragments satisfying the condition $|F| \approx |M|$ were more proportional fragment than others.


Request Rate = 50 req / min
Request Rate = 500 req / min
Throughput (%)
Message Size (Kb)
LOAN_HPD0_1_1=Centralized
LOAN_HIPD0_2_1=IPD
LOAN_HIPD1_3_1
LOAN_HPD1_6_1
LOAN_HPD2_10_1
LOAN_HPD3_16_1=FPD
LOAN_HIPD0_2_2=IPD
LOAN_HIPD1_3_2
LOAN_HPD1_6_2
LOAN_HPD2_10_2
LOAN_HPD3_16_2=FPD
LOAN_HIPD1_3_3
LOAN_HPD1_6_3
LOAN_HPD2_10_3
LOAN_HPD3_16_3=FPD
LOAN_HPD1_6_4
LOAN_HPD2_10_4
LOAN_HPD3_16_4=FPD
LOAN_HPD1_6_5
LOAN_HPD2_10_5
LOAN_HPD3_16_5=FPD

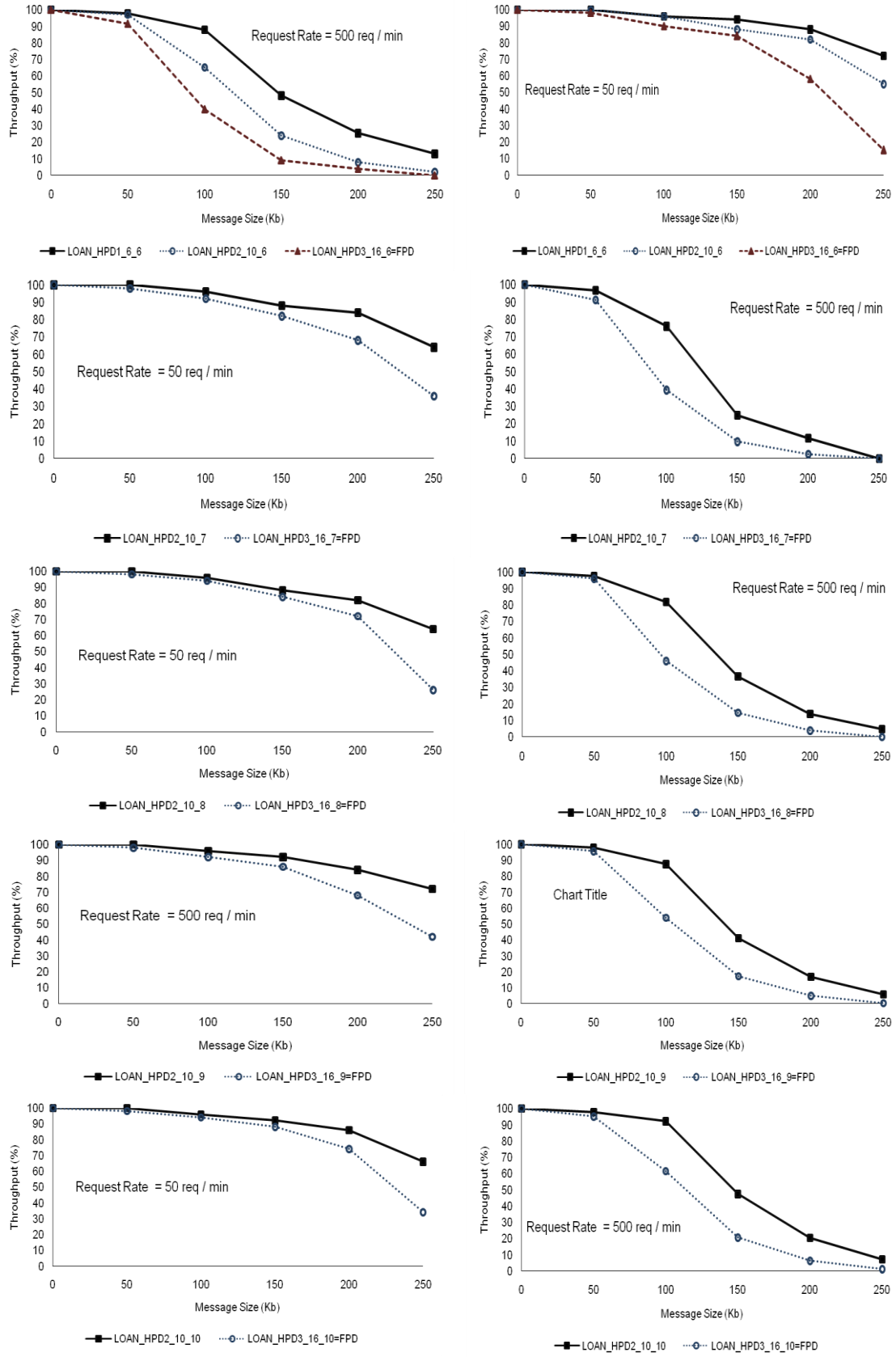


Figure 6.71: Fragment Proportionality of Loan Application Process in terms of Throughput, Type-1 with Constant-Request-Rate, Variable-Message-Size

Table 6.23: Average Percentage of FPD Throughput Improvement, by HPD and HIPD methods, Type-1, Constant-Request-Rate, Variable-Message-Size

		Average Response-time Improvement									
Number of Machines	1	LOAN_HPD0_1_1 (Centralized)		LOAN_HIPD0_2_1 (IPD)		LOAN_HIPD1_3_1		LOAN_HPD1_6_1		LOAN_HPD2_10_1	
		50KB	500KB	50KB	500KB	50KB	500KB	50KB	500KB	50KB	500KB
		62.80%	99.8%	61.6%	55.12%	49.2%	31.68%	4.64%	21.60%	12.4%	1.24%
	2			LOAN_HIPD0_2_2 (IPD)		LOAN_HIPD1_3_2		LOAN_HPD1_6_2		LOAN_HPD2_10_2	
				50KB	500KB	50KB	500KB	50KB	500KB	50KB	500KB
				49.8%	67.64%	47.4%	51.56%	25.4%	18.68%	19.2%	7%
	3					LOAN_HIPD1_3_3		LOAN_HPD1_6_3		LOAN_HPD2_10_3	
						50KB	500KB	50KB	500KB	50KB	500KB
						46%	55.28%	31.2%	26%	19.8%	14%
	4							LOAN_HPD1_6_4		LOAN_HPD2_10_4	
								50KB	500KB	50KB	500KB
								23%	26.36%	12.8%	13.56%
	5							LOAN_HPD1_6_5		LOAN_HPD2_10_5	
								50KB	500KB	50KB	500KB
								21.2%	26.6%	12.2%	11.48%
	6							LOAN_HPD1_6_6		LOAN_HPD2_10_6	
								50KB	500KB	50KB	500KB
								21%	25.56%	8%	10.32%
	7									LOAN_HPD2_10_7	
										50KB	500KB
										11.2%	13.32%
	% 8									LOAN_HPD2_10_8	
										50KB	500KB
										11.2%	14.84%
	9									LOAN_HPD2_10_9	
										50KB	500KB
										11.6%	15.48%
	10									LOAN_HPD2_10_10	
										50KB	500KB
										10.4%	16.00%

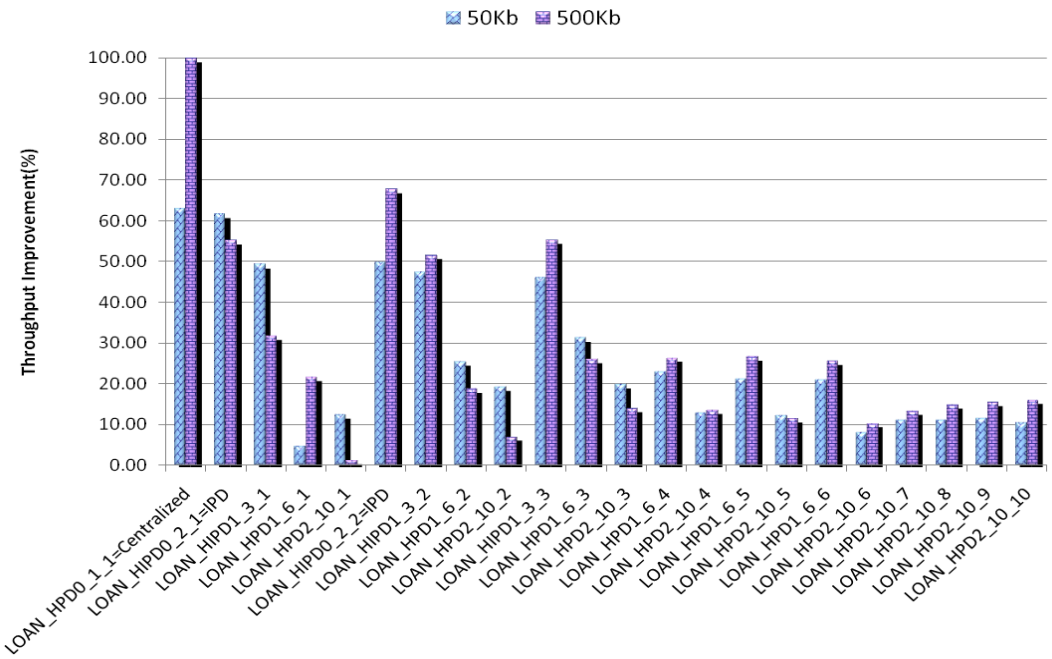
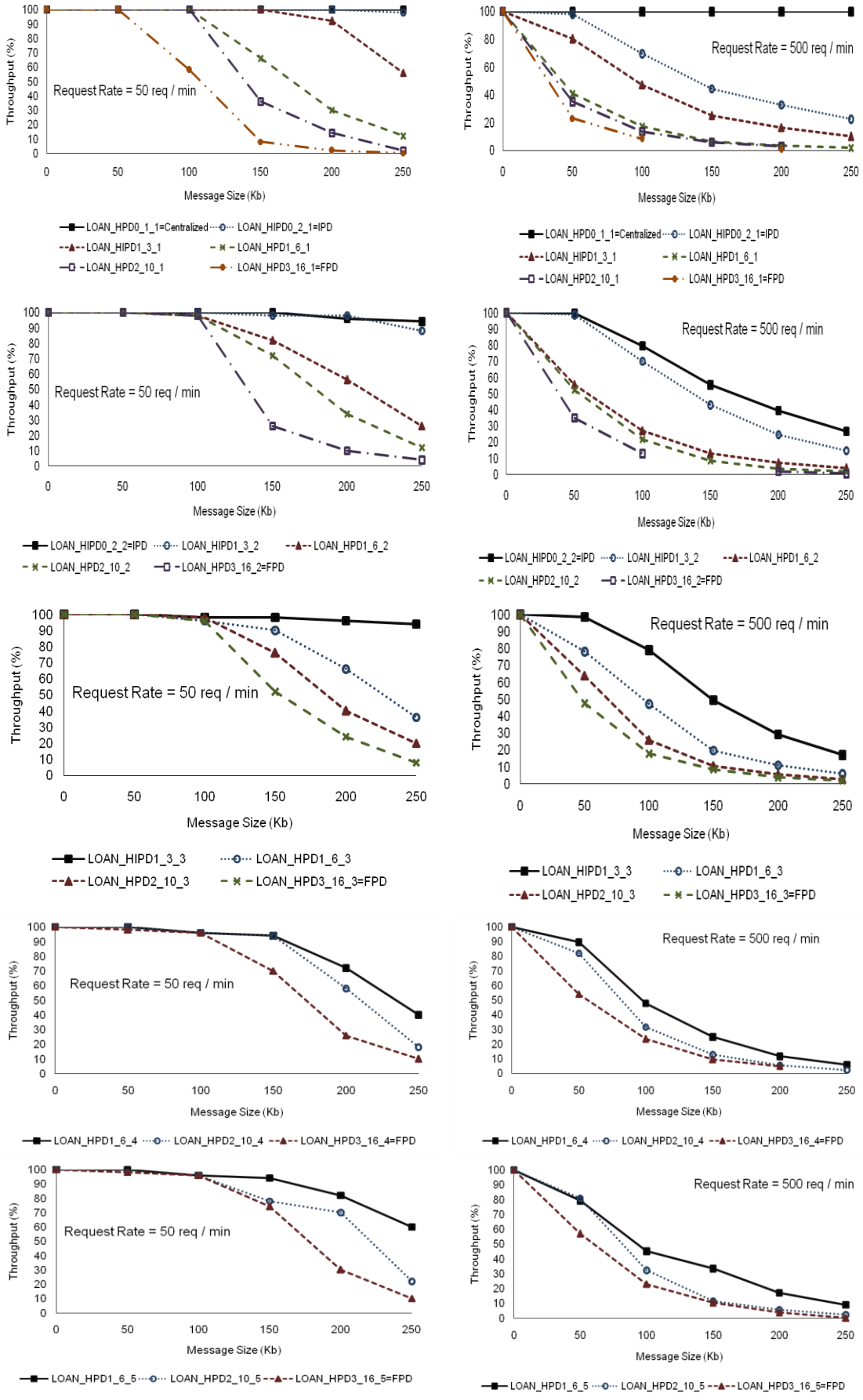


Figure 6.72: Average Percentage of FPD Throughput Improvement, by HPD and HIPD methods, Type-1, Constant-Request-Rate, Variable-Message-Size


Throughput (%)
Message Size (Kb)
Request Rate = 50 req / min
Request Rate = 500 req / min
LOAN_HPD0_1_1=Centralized
LOAN_HIPD0_2_1=IPD
LOAN_HIPD1_3_1
LOAN_HPD1_6_1
LOAN_HPD2_10_1
LOAN_HPD3_16_1=FPD
LOAN_HIPD0_2_2=IPD
LOAN_HIPD1_3_2
LOAN_HPD1_6_2
LOAN_HPD2_10_2
LOAN_HPD3_16_2=FPD
LOAN_HIPD1_3_3
LOAN_HPD1_6_3
LOAN_HPD2_10_3
LOAN_HPD3_16_3=FPD
LOAN_HPD1_6_4
LOAN_HPD2_10_4
LOAN_HPD3_16_4=FPD
LOAN_HPD1_6_5
LOAN_HPD2_10_5
LOAN_HPD3_16_5=FPD

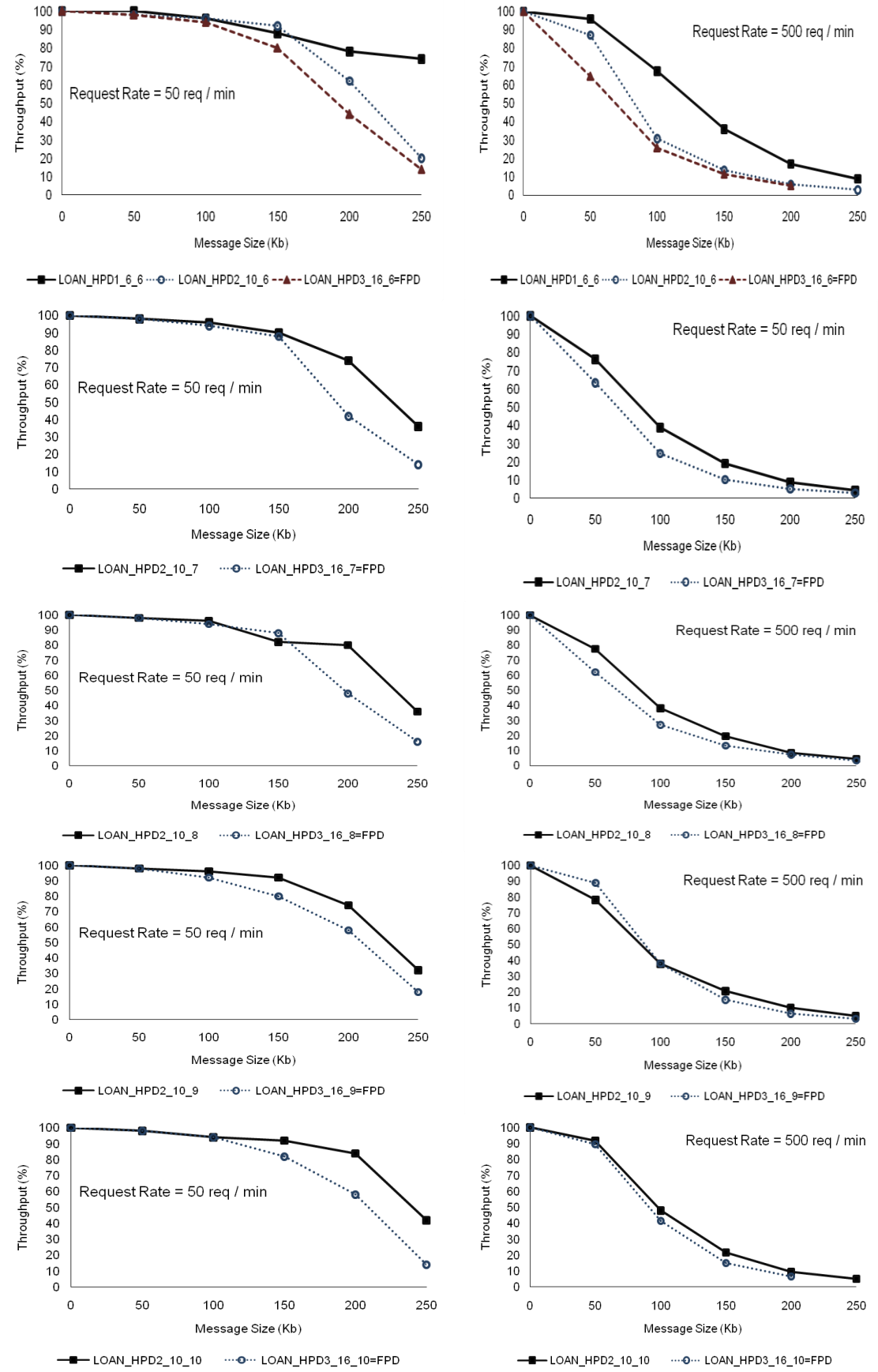


Figure 6.73: Fragment Proportionality of Loan Application Process in terms of Throughput, Type-2, Constant-Request-Rate, Variable-Message-Size

Table 6.24: Average Percentage of FPD Throughput Improvement, by HPD and HIPD methods, Type-2, Constant-Request-Rate, Variable-Message-Size

		Average Response-time Improvement									
Number of Machines	1	LOAN_HPD0_1_1 (Centralized)		LOAN_HIPD0_2_1 (IPD)		LOAN_HIPD1_3_1		LOAN_HPD1_6_1		LOAN_HPD2_10_1	
		50KB	500KB	50KB	500KB	50KB	500KB	50KB	500KB	50KB	500KB
		66.40%	93.20%	66.00%	46.80%	56.00%	29.16%	28.00%	7.28%	28.00%	5.12%
	2			LOAN_HIPD0_2_2 (IPD)		LOAN_HIPD1_3_2		LOAN_HPD1_6_2		LOAN_HPD2_10_2	
				50KB	500KB	50KB	500KB	50KB	500KB	50KB	500KB
				50.40%	48.96%	49.20%	39.04%	24.80%	10.12%	15.60%	6.44%
	3					LOAN_HIPD1_3_3		LOAN_HPD1_6_3		LOAN_HPD2_10_3	
						50KB	500KB	50KB	500KB	50KB	500KB
						41.20%	38.72%	21.60%	16.48%	10.80%	5.88%
	4							LOAN_HPD1_6_4		LOAN_HPD2_10_4	
								50KB	500KB	50KB	500KB
								20.40%	17.16%	13.20%	8.36%
	5							LOAN_HPD1_6_5		LOAN_HPD2_10_5	
								50KB	500KB	50KB	500KB
								24.80%	17.60%	11.20%	7.76%
	6							LOAN_HPD1_6_6		LOAN_HPD2_10_6	
								50KB	500KB	50KB	500KB
								13.20%	23.04%	7.60%	6.28%
	7									LOAN_HPD2_10_7	
										50KB	500KB
										11.60%	8.12%
	8									LOAN_HPD2_10_8	
										50KB	500KB
										9.60%	7.00%
	9									LOAN_HPD2_10_9	
										50KB	500KB
										9.20%	4.44%
	10									LOAN_HPD2_10_10	
										50KB	500KB
										12.80%	4.14%

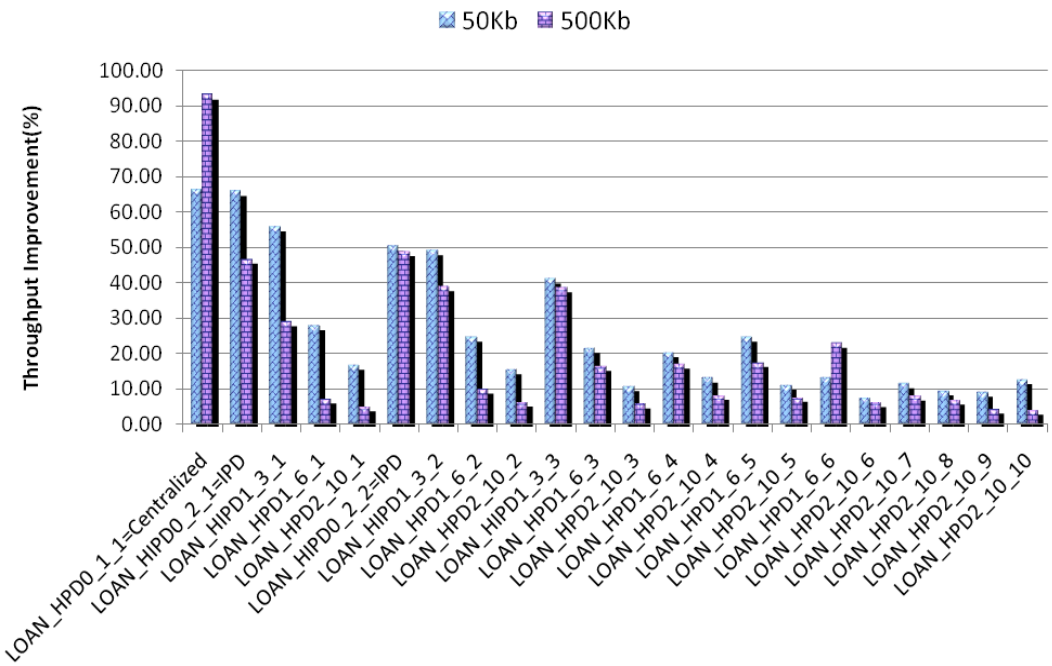


Figure 6.74: Average Percentage of FPD Throughput Improvement, by HPD and HIPD, Type-2, Constant-Request-Rate, Variable-Message-Size

6.4.5 Frequent-path and Proportional-fragment Evaluation Discussion

The main goals of the decentralization methods experiments were i) to work out the outcome of incorporating the frequent-path adaptability in decentralization; ii) to study the result of adopting the proportional-fragment adaptability in decentralization; iii) to compare the decentralization methods with the Centralized fragmentation, which did not use message passing in the minimum distribution and could be considered as an ideal case; and iv) to study the minimum and maximum (Min-Max) distributions of decentralization methods in comparison to the FPD.

The experiments were entirely on evaluating fragments created by the HPD and HIPD methods applying three basic business processes and a complex loan application process. Both traditional decentralization methods including the Centralized and Fully Process Decentralization (FPD) were also implemented as part of both HPD and HIPD decentralization methods. The experiments were conducted on a variety number of machines and the results can be classified as follows:

1. The Centralized method envisaged that the best response-time and throughput in minimum distribution; however, it was not scalable.
2. The fully process decentralization was severely constrained by the number of fragments, exchanging messages and communication media that were the reason why it showed the worst response-time and throughput in all cases.
3. The experiments proved that the frequent-path adaptability of decentralization. In all cases, HIPD fragments showed better response-time and throughput considerably close to the Centralized fragments due to the reduced number of message passing among the high-relevant process fragments.
4. The experiments substantiated that the proportional-fragment adaptability of decentralization. In all cases, the closeness of number of fragments and number

of machines resulted in better response-time and throughput, which was to due to omitting message passing among the fragments running on the same machine.

In a nutshell, the evaluations of the implementation of the ADWEF decentralization methods demonstrated a range of improvements compared to the FPD methods as follows: A) the frequent-path adaptability along with variable-request-rate/constant-message-size improved response-time and throughput about [71%-99%] and [9%-54%] in Type-1, [44%-87%] and [4%-50%] in Type-2; B) the frequent-path adaptability along with constant-request-rate/variable-message-size improved response-time and throughput about [16%-99%] and [1%-99%] in Type-1, [47%-96%] and [16%-66%] in Type-2; C) Both adaptability aspects along with variable-request-rate/constant-message-size, resulted in response-time and throughput improvement [62%-99%] and [2%-61%] in Type-1, [10%-88%] and [5%-50%] in Type-2. D) Both adaptability aspects along with constant-request-rate/variable-size-message resulted in response-time and throughput improvement [16%-99%] and [1%-99%] in Type-1, [26%-99%] and [5%-93%] in Type-2.

6.5 Available-bandwidth Experiments

The focus of this section is to encompass the behavior of the Fuzzy Decentralization Decision Making (FDDM) unit. The unit is expected to join and unify the frequent-path, proportional-fragment and available-bandwidth adaptability aspects. The following experiments were conducted to show the behavior of the unit according to Section 4.8.3. The improvement percentage of different fragmentation models was compared to the FPD. The number of exchanged messages (NoM) in the FPD was more than other methods in all cases; therefore, it was the FPD that was improved by

the other methods. Indeed, the improvement percentage of NoM was calculated as shown in Equation (6.3).

$$improvement_{NoM} = \frac{(NoM_{fpd} - NoM_{other\ method}) \times 100}{NoM_{fpd}} \tag{6.3}$$

6.5.1 Experiment-14 (evaluating available-bandwidth adaptability, HPD decentralization, random bandwidth)

This experiment was performed based on the Experiment-14 specified in Section 4.8.3. First, the behavior of the FDDM unit is shown; and second, as the main goal of the experiment, the number of exchanged messages with and without applying the FDDM unit is evaluated. The unit was only equipped with the HPD decentralization, a random bandwidth simulator, and was tested in presence of 2, 8, 13 and 16 machines during one minute with a constant-request-rate of 500 req/min.

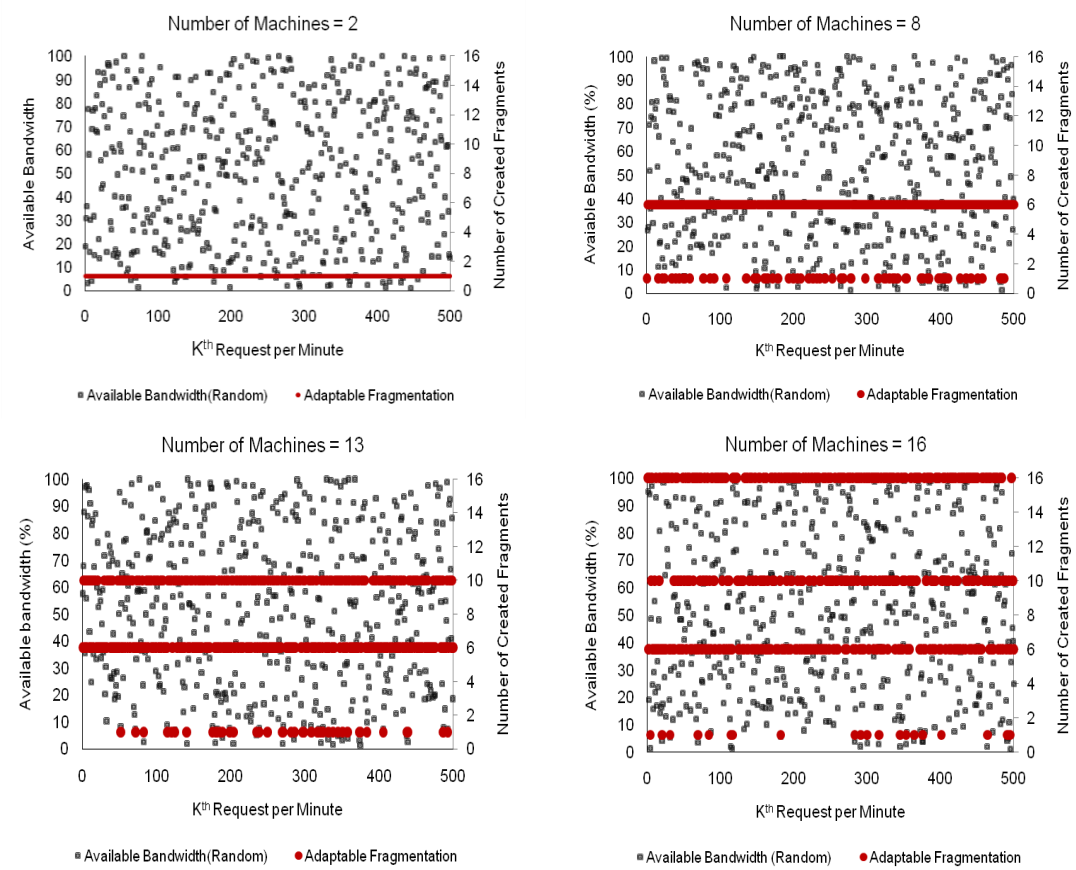


Figure 6.75: Reaction of FDDM unit to Fragment Proportionality, Random Bandwidth, HPD Decentralization

Figure 6.75 plots the trend of bandwidth changes on the left axis and the number of created fragments on the right axis of the graphs while the horizontal axis shows each request individually. The randomness of the bandwidth caused unpredictable fragmentation; nevertheless, the FDDM only considered fragment proportionality due to nondeterministic bandwidth behavior.

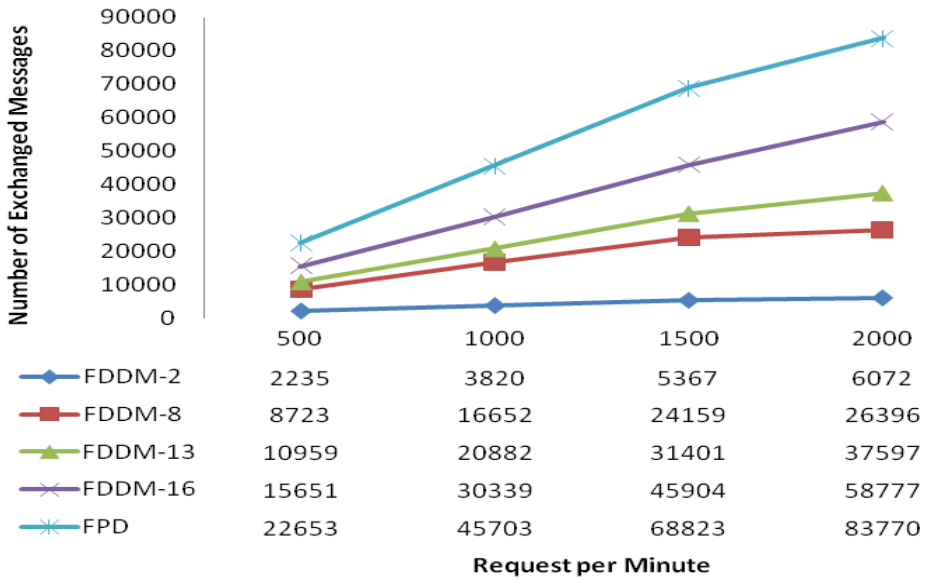


Figure 6.76: FPD and FDDM Number of Exchanged Message Comparison, Considering Fragment Proportionality, Random Bandwidth, HPD Decentralization

The experiment was again performed with constant-request-rate of 500, 1000, 1500 and 2000 per minute. During the experiment, number of exchanged messages was counted with/without applying the FDDM unit as shown in Figure 6.76. The FDDM unit with different number of machines improved the bandwidth usage due to considering fragment proportionality and creating more granular fragments.

Figure 6.77 plots the improvement of exchanged messages in the FPD by the other decentralization methods. According to Table 6.25, the FDDM regardless of various request rates improved the FPD around 91.68%, 64.61%, 53.86% and 31.92% using 2, 8, 13, and 16 machines, respectively. The fragment proportionality resulted in creating coarser fragments with less number of machines and eventually less

message passing. Alternatively, more number of machines resulted in softer fragments and subsequently more message passing.

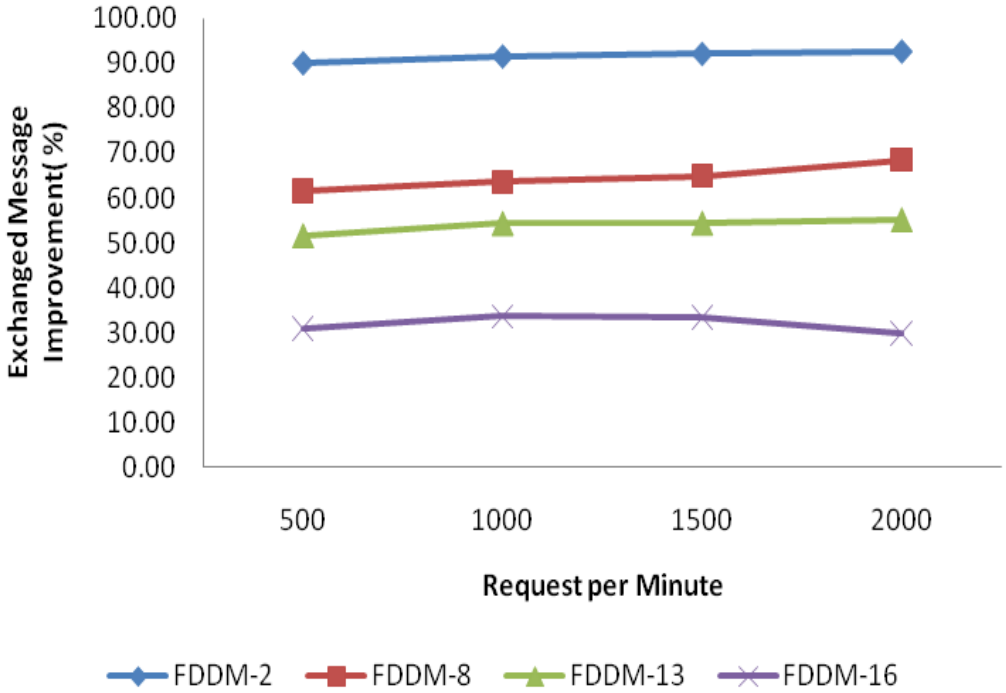


Figure 6.77: Exchanged Message Improvement of FPD, Applying HPD, Random Bandwidth

Table 6.25: Average Exchange Message Improvement, HPD, Random Bandwidth

	FDDM Execution with Variable Number of Machines			
Improvement (%)	FDDM-2	FDDM-8	FDDM-13	FDDM-16
	91.68%	64.61%	53.86%	31.92%

6.5.2 Experiment-15 (evaluating available-bandwidth adaptability, HPD decentralization, exponential bandwidth)

This experiment was performed based on the Experiment-15 specified in Section 4.8.3. First, the behavior of the FDDM unit is shown; and second, as the main goal of the experiment, the number of exchanged messages with and without applying the FDDM unit is evaluated. The unit was equipped with only the HPD decentralization method and an exponential bandwidth generator and it was tested in presence of 2, 8, 13 and 16 machines during one minute with a constant-request-rate of 500 req/min.

Figure 6.78 plots the trend of bandwidth changes on the left axis and the number of created fragments on the right axis of the graphs while the horizontal axis shows each request individually.

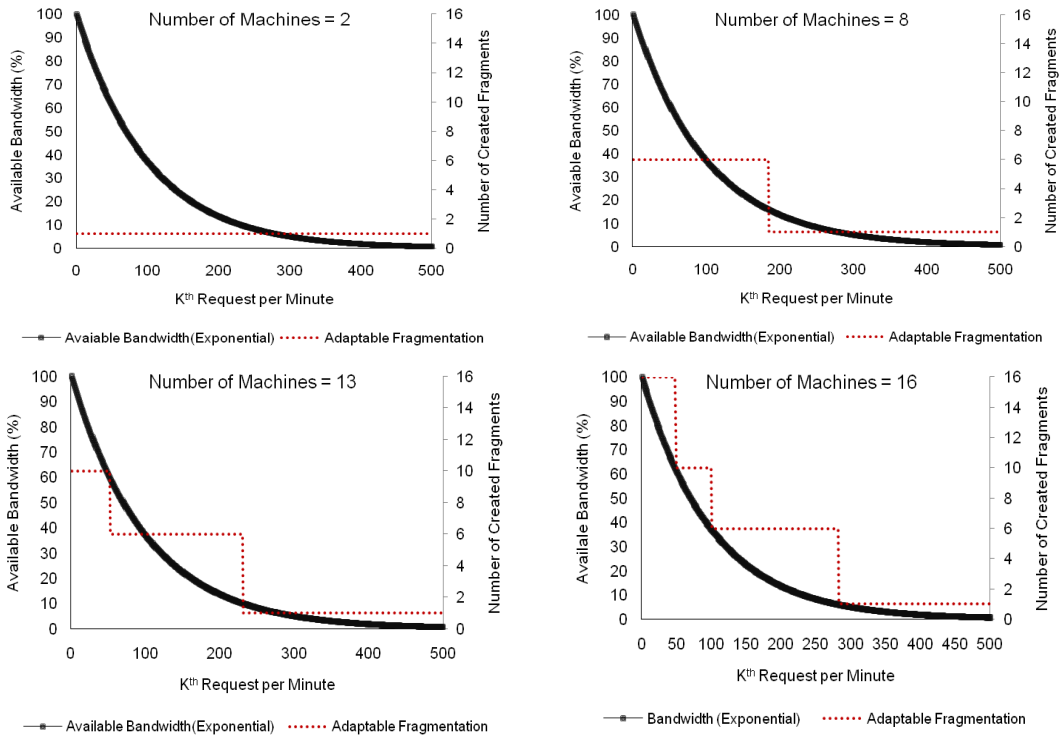


Figure 6.78: Reaction of FDDM unit to Exponential Bandwidth and HPD Decentralization

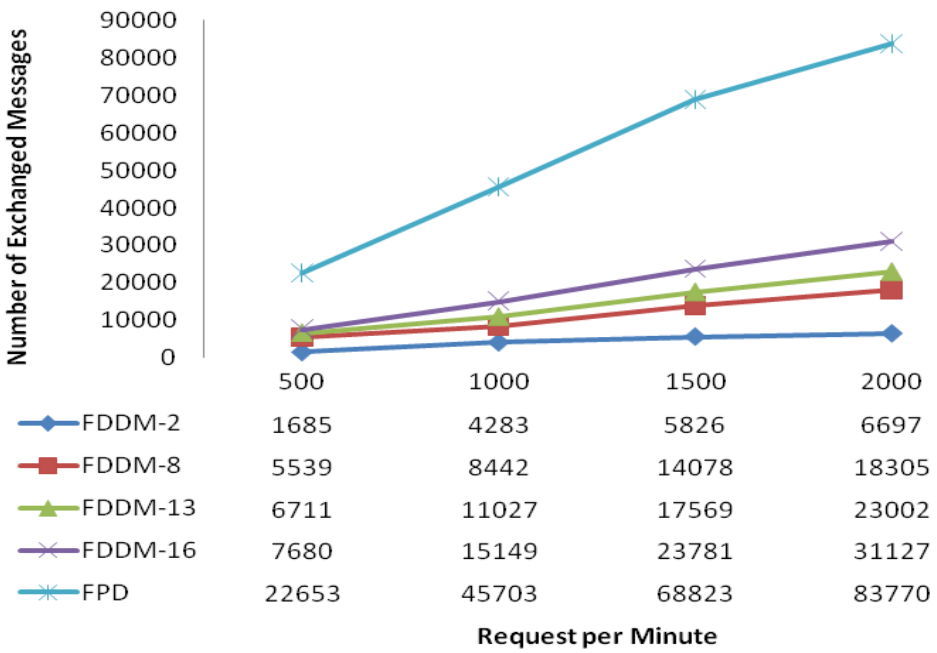


Figure 6.79: FPD and FDDM Number of Exchanged Message Comparison, Exponential Bandwidth and HPD Decentralization

The experiment was again performed with constant-request-rate of 500, 1000, 1500 and 2000 per minute. During the experiment with an exponential bandwidth generator, the number of exchanged messages was counted with/without applying the FDDM unit. As shown in Figure 6.79, using the FDDM unit with different number of machines caused decreasing of bandwidth usage. In this experiment, the FDDM not only considered fragment proportionality, but also consistently adapted the fragmentation of workflows to the available bandwidth. As the bandwidth followed the exponential distribution pattern, the FDDM directed the decentralization from more decentralized fragments to more centralized fragments.

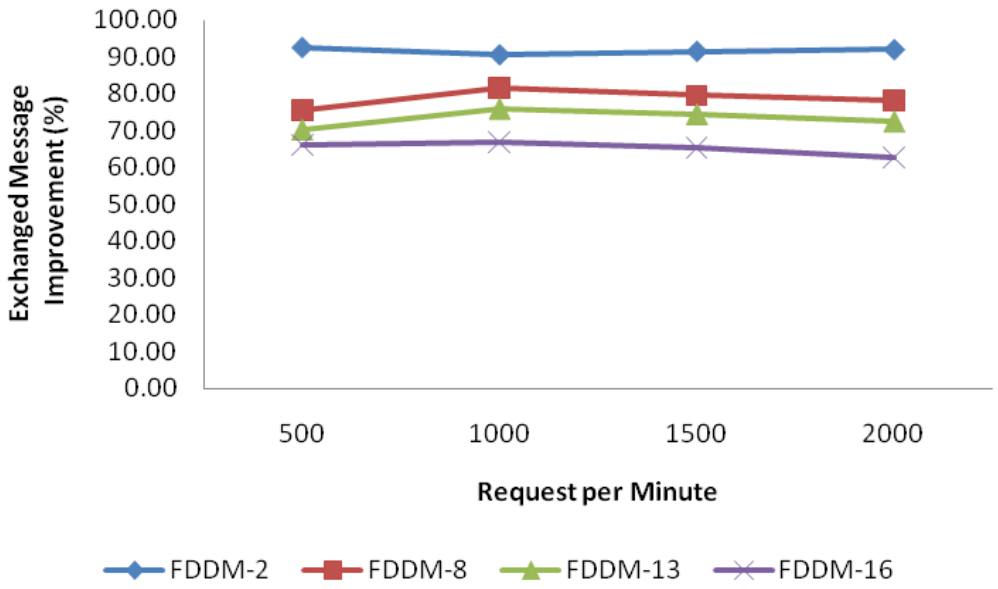


Figure 6.80: Exchanged Message Improvement of FPD Applying HPD and Exponential Bandwidth

Table 6.26: Average Exchange Message Improvement, HPD, Exponential Bandwidth

	FDDM Execution with Variable Number of Machines			
Improvement (%)	FDDM-2	FDDM-8	FDDM-13	FDDM-16
	91.68%	78.69%	73.32%	65.31%

The FPD improvement trend using various numbers of machines is illustrated in Figure 6.80 and the average improvement by each method is shown in Table 6.26. In spite of increasing the request rate, the number of exchanged messages improved around 91.68%, 78.69%, 73.32% and 65.31% applying 2, 8, 13 and 16 machines,

respectively. In this experiment, not only the fragment proportionality, but also the available-bandwidth adaptability contributed to the improvement of exchanged messages. The FDDM reduced the scalability in a high-traffic network and created coarser fragments, whereas it produced softer fragments in an empty communication medium.

6.5.3 Experiment-16 (evaluating available-bandwidth, HPD/HIPD decentralization, random bandwidth)

This experiment was performed based on the Experiment-16 specified in Section 4.8.3. It aimed at evaluating the number of exchanged messages with and without applying the FDDM unit. The unit was equipped with both HPD and HIPD decentralization, a random bandwidth simulator and was tested in presence of 2, 8, 13, and 16 machines during one minute with constant-request-rate of 500 req/min.

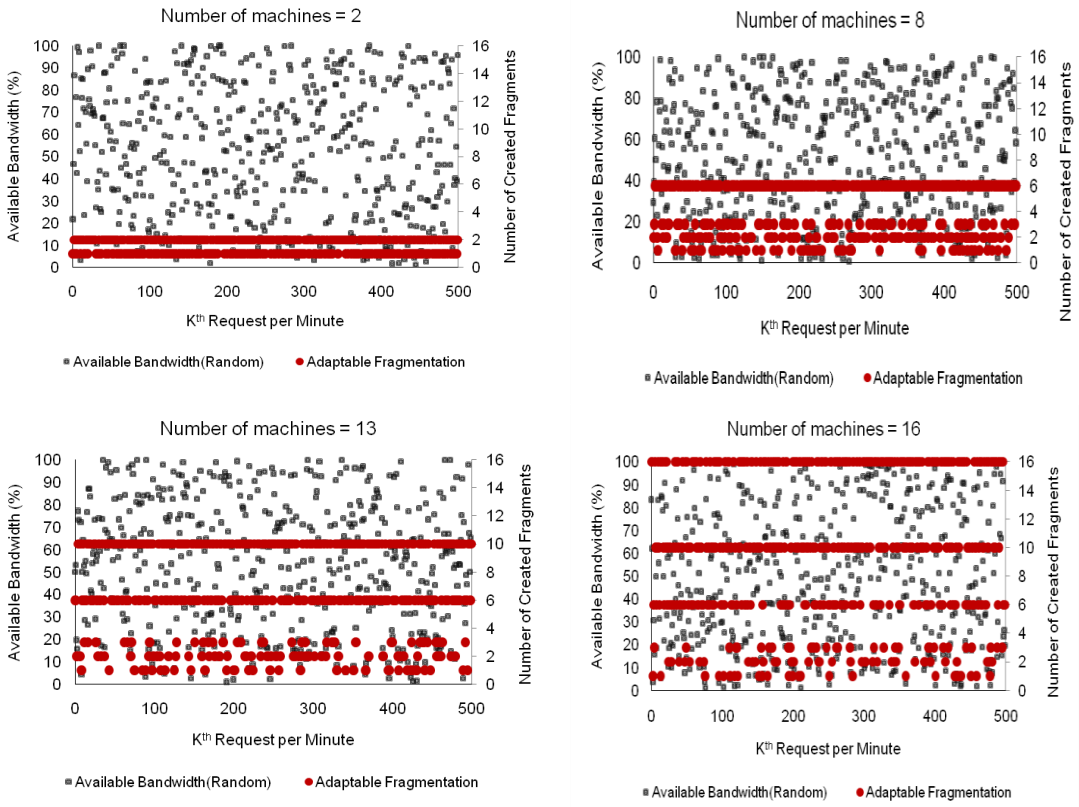


Figure 6.81: Reaction of FDDM unit to Random Bandwidth, HPD and HIPD Decentralization

Figure 6.81 plots the trend of bandwidth changes on the left axis and the number of created fragments on the right axis of the graphs while the horizontal axis shows each request individually. The randomness of the bandwidth caused unpredictable fragmentation; nevertheless, fragment proportionality regulated it. On the other hand, the FDDM only considered fragment proportionality due to the nondeterministic bandwidth behavior.

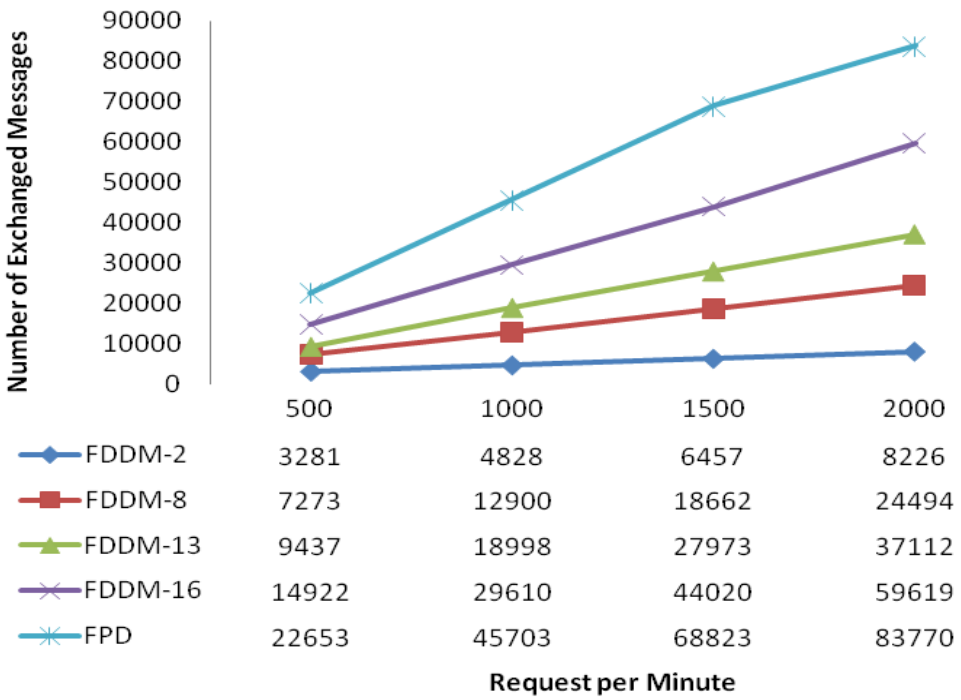


Figure 6.82: FPD and FDDM Number of Exchanged Message Comparison, Random Bandwidth, HPD and HIPD Decentralization

The experiment was again performed with constant-request-rate of 500, 1000, 1500 and 2000 per minute. During the experiment with a random bandwidth generator, the number of exchanged messages was counted with/without applying the FDDM unit. The FDDM created fragments commensurate with the number of machines and generated bandwidth. Due to using both HPD and HIPD fragments, the FDDM unit had more potential to provide more adaptable fragments. As shown in Figure 6.82, using the FDDM unit with different number of machines resulted in decreasing of

bandwidth usage. It was due to considering fragment proportionality, creating more granular fragments and consequently less number of exchanged messages.

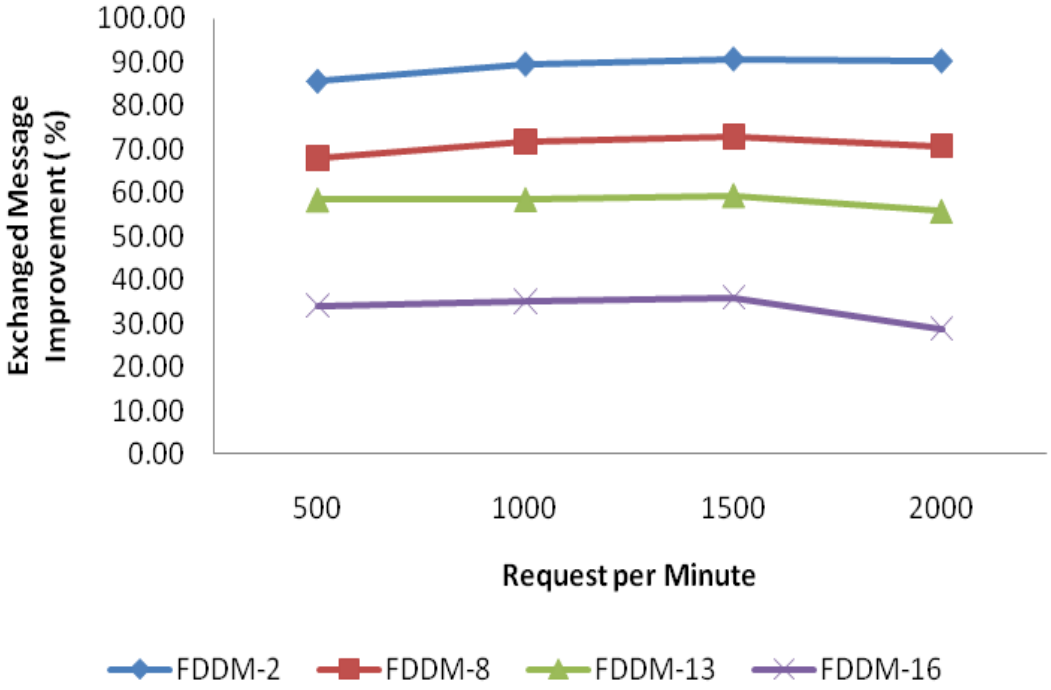


Figure 6.83: Exchanged Message Improvement of FPD Applying HPD/HIPD, Random Bandwidth

Table 6.27: Average Exchange Message Improvement, HPD/HIPD, Random Bandwidth

	FDDM Execution with Variable Number of Machines			
Improvement (%)	FDDM-2	FDDM-8	FDDM-13	FDDM-16
	88.94%	70.83%	57.96%	33.55%

Figure 6.83 plots the FPD improvement trend of exchanged messages by the other decentralization methods. According to Table 6.27, the FDDM regardless of various request rates improved the FDDM 88.94%, 70.83%, 57.96% and 33.55% using 2, 8, 13 and 16 machines, respectively. The fragment proportionality showed that applying less number of machines resulted in creating coarser fragments and eventually less message passing. Alternatively, more number of machines resulted in creating softer fragments sand subsequently more exchanged messages.

6.5.4 Experiment-17 (evaluating available-bandwidth adaptability, HPD/HIPD decentralization, exponential bandwidth)

This experiment was performed based on the Experiment-17 specified in Section 4.8.3. First, the behavior of the FDDM unit is shown; and second, as the main goal of the experiment, the number of exchanged messages with and without applying the FDDM unit is evaluated. The FDDM was equipped with both HPD and HIPD decentralization methods and was tested in presence of 2, 8, 13 and 16 machines during one minute with a constant-request-rate of 500 req/min.

Figure 6.84 shows the trend of bandwidth fluctuations on the left axis and the number of created fragments on the right axis of the graphs while the horizontal axis each request individually. As the number of machines and available bandwidth were increased, the FDDM unit created new proportional fragments. After being occupied the network, the FDDM created less number of fragments.

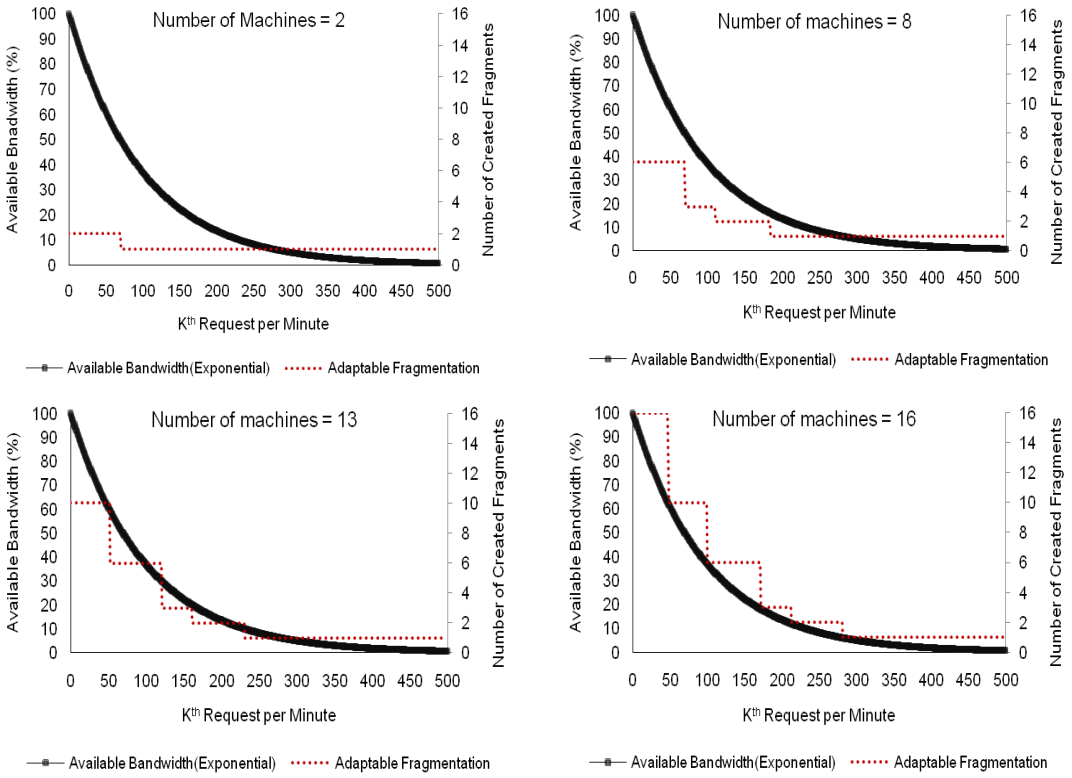


Figure 6.84: Reaction of FDDM unit to Exponential Bandwidth, HPD and HIPD Decentralization

The experiment was again performed with constant-request-rate 500, 1000, 1500 and 2000 per minute. During the experiment with an exponential bandwidth generator, the number of exchanged messages was counted with/without applying the FDDM unit. The FDDM created fragments commensurate with the number of machines and generated bandwidth.

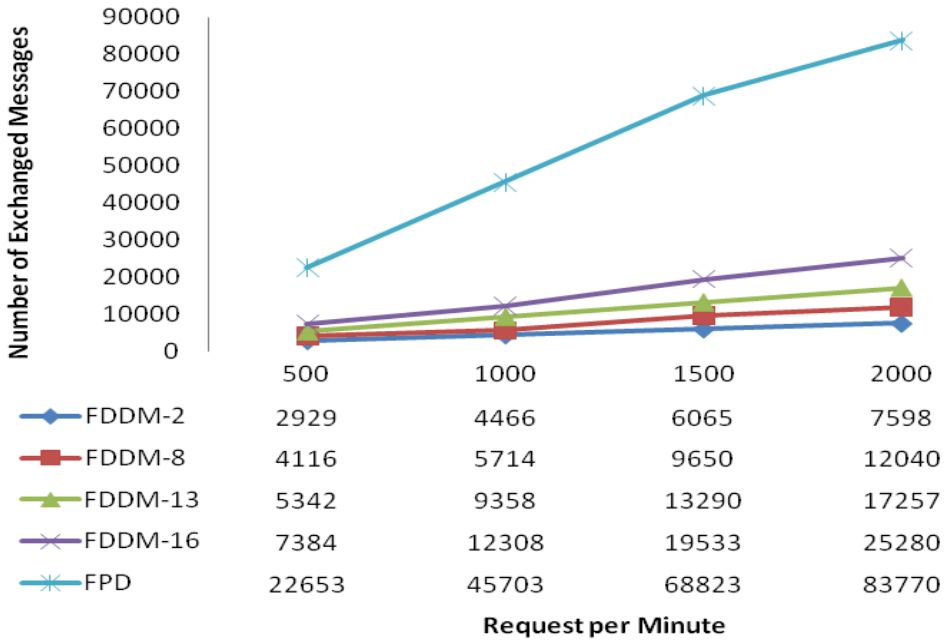


Figure 6.85: FPD and FDDM Number of Exchanged Message Comparison, Considering Exponential Bandwidth, HPD and HIPD Decentralization

As shown in Figure 6.85, using the FDDM unit with different number of machines caused decreasing of bandwidth usage. In this experiment, the FDDM not only considered fragment proportionality, but also consistently adapted the fragmentation of workflows to the available bandwidth. The only difference between this experiment and the Experiment-15 was that the FDDM in the current experiment had more flexibility in choosing a fragment set with better fragment proportionality due to collecting the fragments of both HPD and HIPD. As the bandwidth followed the exponential distribution pattern, the FDDM directed the decentralization from more decentralize to less decentralized fragments. This is the reason why the number of

exchanged messages is totally less than the random model by comparing data in Figure 6.76 and Figure 6.85.

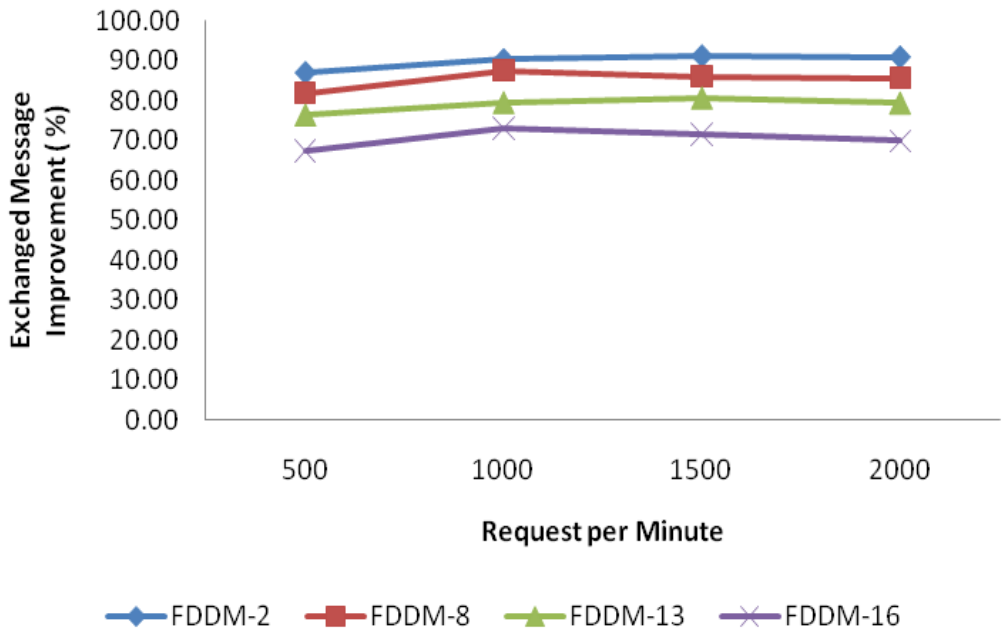


Figure 6.86: Exchanged Message Improvement of FPD Applying HPD/HIPD, Exponential Bandwidth

Table 6.28: Average Exchange Message Improvement, HPD/HIPD, Exponential Bandwidth

	FDDM Execution with Variable Number of Machines			
Improvement (%)	FDDM-2	FDDM-8	FDDM-13	FDDM-16
	89.85%	85.23%	79.01%	70.48%

6.5.5 Available-bandwidth Evaluation Discussion

The main goal of the Fuzzy Decentralization Decision Making unit (FDDM) was to figure out the result of incorporating the available-bandwidth adaptability in business process decentralization. The FDDM used the frequent-path and proportional-fragment adaptability to implement the available-bandwidth adaptability. The experiments applied the loan application business process fragments along with applying random/exponential bandwidth simulator, the HPD/HIPD Loan-process fragments and they were executed by a variety number of machines. The FDDM unit considered the number of machines and available bandwidth and chose a suitable decentralization method. The experiments outcomes can be classified as follows:

1. The FDDM along with the HPD decentralization was able to adapt decentralization to fragment proportionality; however, in exponential request rate it was able to manage decentralization more consistently.
2. The FDDM along with both HPD and HIPD decentralization was able to realize both frequent-path and fragment-proportionality aspects of decentralization; however, in an exponential request rate it was able to manage decentralization more consistently. The HIPD added more fragments with the least amount of messages to the HPD fragments, which contributed to the FDDM to decentralize a business process with more flexibility.
3. The FDDM balanced the scalability and adaptability in that it considered maximum scalability based on the fragment proportionality as well as the available-bandwidth adaptability based on the current bandwidth. In all experiments, the FDDM balanced the number of exchanged messages so that it was reduced compared to the FPD method and it simultaneously considered the scalability of decentralization.

Table 6.29: Average FPD Exchange Message Improvement

		FDDM Execution with Variable Number of Machines			
		FDDM-2	FDDM-8	FDDM-13	FDDM-16
HPD	**Random**	91.68%	64.61%	53.86%	31.92%
	Exponential	91.68%	78.69%	73.32%	65.31%
HPD/HIPD	**Random**	88.94%	70.83%	57.96%	33.55%
	Exponential	89.85%	85.23%	79.01%	70.48%

Concisely, the FDDM unit balanced the scalability and decentralization by applying three aspects of adaptability including frequent-path, proportional-fragment and available-bandwidth. The available-bandwidth adaptability was realized by the FDDM based on the frequent-path and proportional-fragment adaptability aspects. Evaluations of the FDDM demonstrated the improvement of the FPD according to Table 6.29.

6.6 Discussion

Three categories of the experiments were conducted including boundary-condition experiments to show the behavior of experiment environment in boundary conditions, the frequent-path and proportional-fragment experiments to prove Hypothesis-2, and the available-bandwidth experiments to substantiate Hypothesis-3

The frequent-path and proportional-fragment experiments shown in the prior sections were based on the three basic processes as well as one more complex loan application process. i) The Centralized method in comparison to the minimum distribution of the other methods showed the best response-time, throughput and bandwidth-usage, entirely. It was due to parallel execution of workflow fragments in runtime threads and the number of exchanged messages, which was obviously around zero. ii) In addition, the HIPD method in different levels of fragmentation and in Min-Max distributions was considerably similar to the Centralized method, which was the reason why the frequent-path adaptability improved the decentralization. iii) The last level of fragmentation, which was equal to the FPD, showed the worst response-time, throughput and message passing in minimum distribution due to the high amount of message passing among the fragments and number of created fragments; however, by scaling up the system, the FPD method gradually showed better results. Nonetheless, the results were weaker than the execution results of the Centralized, HPD and HIPD fragments. iv) In minimum distribution, when all fragments encapsulated in their respective performer agents on the same machine, response-time and throughput were in the worst case. In contrast, in maximum distribution, response-time and throughput of all processes became considerably closer to each other. v) The fragment proportionality showed that when the number of produced fragments was approximately equal to the number of dedicated

machines, the response-time and throughput were considerably improved compared to the FPD method that did not consider adaptability. vi) The architecture Type-1 had less strength against the number of fragments compared to the architecture Type-2.

The last category of experiments examined the available-bandwidth adaptability. In fact, the FDDM unit balanced the scalability and adaptability of decentralization. When communication media was empty and allowed scalability, the FDDM decentralized the processes based on the fragment proportionality. After being occupied the communication medium, the system considered adaptability and created less decentralized fragments to alleviate the crowdedness of communication media.

6.7 Summary

This chapter focused on evaluations of the three adaptability aspects in the Adaptable Decentralization Workflow Execution Framework (ADWEF). The whole experiments were based on the research methodology represented in Chapter 4 and targeted at substantiating the research three hypotheses. Accordingly, three categories of experiments including boundary-condition, frequent-path and proportional-fragment, and available-bandwidth were conducted and discussed.

CHAPTER 7

SUMMARY, CONCLUSION AND RECOMMENDATIONS FOR FUTURE RESEARCH

7.1 Research Summary

This research was motivated by (1) applications of business processes in huge organizations with lots of branches; (2) cloud computing in which software is considered as a service and in the same way an orchestration engine may also be considered as a service. In both cases, an orchestration engine may be targeted at receiving a huge number of requests at a moment. Due to the fact that centralized orchestration engine suffer from the lack of scalability, business processes are decentralized and fragmented at design time by designers. Several research activities tried to decentralize business processes to its activity-level ingredients which is fragmentation in the softest grains while some researchers choose coarser grain fragments. A few research works decentralize business processes at runtime as well; however, without considering runtime circumstances. On the other hand the open-world software paradigm expects software to be adaptable with ever-changing runtime environment. The adoption of this idea in business process decentralization was the novelty of this thesis, which is analogous to applying runtime feedbacks in business process decentralization to create runtime-adaptable fragments.

In this research, three aspects of runtime adaptability in decentralization were studied. The first aspect was *frequent-path* adaptability, which is equal to detecting closely-interrelated activities and encapsulating them in the same fragment. Another aspect was *proportional-fragment* adaptability, which is analogous to the proportionality of produced fragments with number of machines. The last aspect was

available-bandwidth adaptability, which is process fragmentation based on current circumstances of communication media.

The following research objectives were considered in this thesis as well: 1) to propose a framework including architectures to support execution of dynamically created fragments, decentralization methods to create adapted fragments, and a feedback loop from runtime environment to consider runtime circumstances in business process decentralization; 2) to improve response-time and throughput of decentralized business processes applying the frequent-path and proportional-fragment adaptability aspects; 3) to improve bandwidth-usage of decentralized business processes applying the available-bandwidth adaptability.

In addition, the contributions of this research were as follows: i) An Adaptable and Decentralized Workflow Execution Framework (ADWEF) was introduced that proposes an abstraction of a runtime adaptable decentralization in the SOA orchestration layer; ii) two architectures Type-1 and Type-2 were presented for the ADWEF that are able to support the execution of dynamically created fragments; iii) three aspects of runtime adaptability in decentralization namely frequent-path, proportional-fragment and available-bandwidth were introduced; iv) two decentralization methods called Hierarchical Process Decentralization (HPD) and Hierarchical and Intelligent Process Decentralization (HIPD) were presented, which were capable of providing various fragments. The latter considers the frequent-path adaptability and both of them together satisfy both frequent-path and proportional-fragment adaptability aspects; v) A Fuzzy Decentralization Decision Making unit (FDDM) was presented based on the fuzzy logic to choose a suitable method of decentralization that satisfies the three adaptability aspects frequent-path,

proportional-fragment and available-bandwidth; and, vi) an algorithm was introduced for wiring of dynamic fragments.

7.2 Conclusion

The research conducted by this thesis showed that the runtime adaptability of fragments is of high importance. The runtime-adaptable fragmentation of business processes not only increases workflow engines scalability but also keeps the system quality factors close to centralized engines. By changing runtime environment circumstances, a system will able to re-configure itself and avoid violating its thresholds. It means software is able to continuously react to its environment, which is analogous to evolving system behavior at runtime.

The three adaptability aspects in the ADWEF were evaluated, afterwards. The frequent-path adaptability substantiated that relations of workflow activities must be considered in fragmentation. The proportional-fragment adaptability proved that number of fragments must be proportional to number of workflow engines machines. The available-bandwidth adaptability reduced number of exchanged messages. The three adaptability aspects met each other and were unified by the FDDM unit. It created fragments commensurate with runtime environment circumstances in terms of relations among workflow activities, number of workflow engine machines, and avail bandwidth of communication media.

7.3 Recommendations for Future Research

This work can be extended in several ways based on the experiences achieved in this research.

- The current work focuses on block-structured processes due to their extensive usage; however, a similar attempt may consider graph-structured processes.

- This current research study opens discussions on applying intelligent methods for business process decentralization and the incorporation of the data mining and fuzzy logic in decentralization were detailed. More adaptable criteria and effective intelligent methods are also required to decentralize a business process based on runtime requirements.
- When a workflow engine approaches its system thresholds, effective reconfiguration algorithms should come to action. The algorithms must re-fragment the workflows based on the current situation.
- The adaptable decentralization of workflows may be applied by some new cloud computing ideas such as software or platform as service. In the cloud platform, a runtime environment may receive thousands of requests from around the world. Thus, tasks in a cloud may be described by workflow metaphors and be decentralized in an adaptable way.
- In addition, satisfying stakeholders' requirements is of high importance; particularly, when they are ensured by service level agreements (SLA) (Lamanna et al., 2003). The idea presented in this thesis may also be integrated with the SLA concept to better ensure the SLA commitments.
- Dynamicity of fragmentation raises new requirements for managing fragments state, exception handling and transactions. There should be design and runtime patterns for each of them.
- The governance of workflow fragments by various organizations requires suitable policies. For instance, there should be some policies to determine which part of a workflow have to be fragmented and must be run by which machine, in which department/organization, etc.

REFERENCES

Active-Endpoints. (2011). ActiveVOS Education Center from http://www.activevos.com/developers

Aiello, M., Papazoglou, M., Yang, J., Carman, M., Pistore, M., Serafini, L., & Traverso, P. (2002). A request language for web-services based on planning and constraint satisfaction. *Technologies for E-Services*, 9-38.

Alast, W. v. d. (2000). Loosely coupled interorganizational workflows: modeling and analyzing workflows crossing organizational boundaries. *Information and Management, 37*(2), 67-75.

Alonso, G., Casati, F., Harumi, k., & Vijay, M. (2004). *Web services: concepts, architectures and applications*: Springer Verlag

Apache-Camel. (2011). The Apache Software Foundation - Apache Camel. from http://camel.apache.org/index.html

Atluri, V., Chun, S. A., Mukkamala, R., & Mazzoleni, P. (2007). A decentralized execution model for inter-organizational workflows. *Distributed and Parallel Databases, 22*(1), 55-83.

Barbara, D., Mehrotra, S., & Rusinkiewicz, M. (1996). INCAs: Managing dynamic workflows in distributed environments. *Journal of Database Management, 7*(1), 5-5.

Baresi, L., & Guinea, S. (2010). Self-supervising BPEL Processes. *IEEE Transactions on Software Engineering, 37*(2), 247 - 263

Baresi, L., Guinea, S., Nano, O., & Spanoudakis, G. (2010). Comprehensive Monitoring of BPEL Processes. *IEEE Internet Computing, 14*(3), 50-57.

Baresi, L., Maurino, A., & Modafferi, S. (2004). *Workflow partitioning in mobile information systems.* Paper presented at the IFIP TC8 Working Conference on Mobile Information Systems (MOBIS 2004), 93-106.

Baresi, L., Maurino, A., & Modafferi, S. (2006). Towards distributed BPEL orchestrations. *Electronic Communications of the EASST, 3*, 1-14.

Baresi, L., Nitto, E. D., & Ghezzi, C. (2006). Toward open-world software: Issue and challenges. *IEEE Journal of Computer, 39*(10), 36-43.

Bausch, W., Pautasso, C., & Alonso, G. (2003). *Programming for Dependability in a Service-based Grid*, 164-171.

Bellifemine, F., Caire, G., & Greenwood, D. (2007). *Developing Multi-Agent Systems with JADE*: WILEY.303

Bellifemine, F., Caire, G., Poggi, A., & Rimassa, G. (2008). JADE: A software framework for developing multi-agent applications. Lessons learned. *Inf. Softw. Technol., 50*(1-2), 10-21.

Bellifemine, F., Poggi, A., & Rimassa, G. (1999). *JADE–A FIPA-compliant agent framework.* Paper presented at the Fourth International Conference on The Practical Application of Intelligent Agents and Multi-agent Technology (PAAM'99), London, UK, *99*, 97-108.

Bellifemine, F., Poggi, A., & Rimassa, G. (2001). Developing multi-agent systems with a FIPA-compliant agent framework. *Software Practice and Experience, 31*(2), 103-128.

Benatallah, B., Sheng, Q. Z., & Dumas, M. (2003). The Self-Serv Environment for Web Services Composition. *IEEE Internet Computing, 7*(1), 40-48.

Bhaduri, K., Das, K., Liu, K., Kargupta, H., & Ryan, J. (2011). Distributed Data Mining Bibliography. from http://www.cs.umbc.edu/~hillol/DDMBIB/

Cai, T., Gloor, P. A., & Nog, S. (1996). DartFlow: A workflow management system on the web using transportable agents. *Dartmouth College, Hanover, NH*.

Caire, G., Gotta, D., & Banzi, M. (2008). *WADE: a software platform to develop mission critical applications exploiting agents and workflows.* Paper presented at the 7th international joint conference on Autonomous agents and multi-agent systems: industrial track (AAMAS'08), Richland, SC, 29-36.

Cao, J., Li, M., Wei, W., & Zhang, S. (2006). A Distributed Re-configurable Grid Workflow Engine. *Computational Science–ICCS 2006*, 948-955.

Carriero, N., & Gelernter, D. (1989). Linda in context. *Journal of Communications of the ACM, 32*(4), 444-458.

Cass, A. G., Staudt, B., Eric, L., Mccall, K., Osterweil, L. J., & Wise, E. (2000). *Little-JIL/Juliette: a process definition language and interpreter*. Paper presented at the 22nd International Conference on Software Engineering (ICSE 2000), 754-757.

Chafle, G. B., Chandra, S., Mann, V., & Nanda, M. G. (2004). *Decentralized orchestration of composite web services*. Paper presented at the Proceedings of the 13th international World Wide Web conference on Alternate track papers and posters. 134-143.

Chmiel, K., Gawinecki, M., Kaczmarek, P., Szymczak, M., & Paprzycki, M. (2005). Efficiency of JADE agent platform. *Journal of Scientific Programming, 13*(2), 159-172.

Cicirelli, F., Furfaro, A., & Nigro, L. (2010). A service-based architecture for dynamically reconfigurable workflows. *Elsevier Journal of Systems and Software, 83*(7), 1148-1164.

Contenti, M., Mecella, M., Termini, A., & Baldoni, R. (2005). *A Distributed Architecture for Supporting e-Government Cooperative Processes*. Paper presented at the TED Conference on e-Government Electronic democracy: The challenge ahead (TCGOV'05). *3416*, 181-192.

Das, S., Kochut, K., Miller, J., Sheth, A., & Worah, D. (1996). ORBWork: A reliable distributed CORBA-based workflow enactment system for METEOR2. 2011, from http://www.cs.uga.edu/~jam/papers/zLSDISpapers/das96orbwork.pdf

Dimitris, N. C. (2011). *Cloud Computing Strategies*: CRC Press

Discenza., F. C. a. A. (2001). Modeling and managing interactions among business processes. *Journal of Systems Integration, 10(2):145-168*.

Dubouloz, B., & Toklu, C. (2005). *Discovering the Most Frequent Patterns of Executions in Business Processes Described in BPEL*. Paper presented at the Web Information Systems Engineering (WISE 2005), *3806*, 701-710.

Dustdar, S., & Gombotz, R. (2006). Discovering web service workflows using web services interaction mining. *International Journal of Business Process Integration and Management, 1*(4), 256-266.

Dustdar, S., & Papazoglou, M. P. (2008). Services and Service Composition-An Introduction (Services und Service Komposition-Eine Einführung). *it - Information Technology, 50*(2), 86-92.

Erl, T. (2005). *Service-oriented architecture: concepts, technology, and design*: Prentice Hall PTR Upper Saddle River, NJ, USA

Fdhila, W., Yildiz, U., & Godart, C. (2009). *A Flexible Approach for Automatic Process Decentralization Using Dependency Tables (ICWS'09).* Paper presented at the Proceedings of the 2009 IEEE International Conference on Web Services. 847-855.

Fidler, E., Jacobsen, H., Li, G., & Mankovski, S. (2005). The PADRES distributed publish/subscribe system. *Feature interactions in telecommunications and software systems VIII*, 12.

Fortino, G., Garro, A., & Russo, W. (2006a). *Distributed Workflow Enactment: an Agent-based Framework.* Paper presented at the Dagli Oggetti Agli Agenti (WOA2006), Catania, Italia, 110-117.

Fortino, G., Garro, A., & Russo, W. (2006b). *From modeling to enactment of distributed workflows: an agent-based approach.* Paper presented at the Proceedings of the 2006 ACM symposium on Applied computing, Dijon, France, 128-129.

Freeman, E., Hupfer, S., & Arnold, K. (1999). *JavaSpaces(TM) Principles, Patterns, and Practice*: Sun Microsystems Inc.

Garcia-Banuelos, L. (2008). *Pattern Identification and Classification in the Translation from BPMN to BPEL.* Paper presented at the OTM 2008 Confederated International Conferences on the Move to Meaningful Internet Systems (OTM'08), Monterrey, Mexico, *5331*, 436–444.

Gong, Y., Janssen, M., Overbeek, S., & Zuurmond, A. (2009). *Enabling flexible processes by ECA orchestration architecture*. Paper presented at the Proceedings of the 3rd International Conference on Theory and Practice of Electronic Governance. 19-26.

Guo, L., Robertson, D., & Chen-Burger, Y.-H. (2008). Using multi-agent platform for pure decentralised business workflows. *Web Intelligence and Agent Systems, 6*(3), 295-311.

Guo, L., Robertson, D., & Chen-Burger, Y. H. (2005). *A novel approach for enacting the distributed business workflows using bpel4ws on the multi-agent platform.* Paper presented at the 2005 IEEE International Conference on e-Business Engineering (ICEBE'05), Beijing 657-664.

Han, J., & Kamber, M. (2001). *Data Mining: Concepts and Techniques*: Academic Press.225-276

Hewlett-Packard Company, H. O. V. (2011). from http://www.openview.hp.com

Hsu, M., & Kleissner, C. (1996). ObjectFlow: towards a process management infrastructure. *Distrib. Parallel Databases, 4*(2), 169-194.

IBM. (2011a). Business process execution language for web services, version 1.1. *Standards proposal by BEA Systems, IBM, Microsoft Corporation, SAP AG, Siebel Systems*, from http://www.ibm.com/developerworks/library/specification/ws-bpel/

IBM. (2011b). Web Services Flow Language in Action. from http://www.ibm.com/developerworks/webservices/library/ws-wsfl1/

Jung, J., Lee, W., & Kang, S. (2006). Process Decomposition and Choreography for Distributed Scientific Workflow Enactment. *Computational Science and Its Applications-ICCSA 2006*, 942-951.

Jupitermedia-Corporation. (2011). Webopedia: Online Dictionary for Computer and Internet Terms. from http://www.webopedia.com/

Kay, M. (2007). *XPath 2.0 Programmer's Reference*: Wrox

Khalaf, R., Kopp, O., & Leymann., F. (2008). Maintaining Data Dependencies Across BPEL Process Fragments. *International Journal of Cooperative Information Systems (IJCIS), 17*(3), 259-282.

Khalaf, R., & Leymann, F. (2006, Sep 18-22). *E Role-based Decomposition of Businesses using BPEL*. Paper presented at the IEEE International Conference on Web Services (ICWS'06), Chicago, IL 770-780.

Kirsten, V. G., Lenz, K., & Oberweis, A. (1998). *Distributed workflow execution based on fragmentation of petri nets.* Paper presented at the 15th IFIP World Computer Congress Telecooperation – The Global Office, Teleworking and Communication Tools Vienna, Austria, 114-125.

Knuth, D. E. (1997). *The Art of Computer Programming (TAOCP)* (3rd ed. Vol. 1). Boston: Addison-Wesley

Kopp, O., Martin, D., Wutke, D., & Leymann, F. (2008, November 27 - 28). *On the Choice between Graph-Based and Block-Structured Business Process Modeling Languages.* Paper presented at the Modellierung betrieblicher Informations systeme (MobIS 2008), Bonn, Germany, *141*, 59-72.

Lamanna, D. D., Skene, J., & Emmerich, W. (2003, May 28-30). *SLAng: A Language for Defigning Service Level Agreements.* Paper presented at the Ninth IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS'03), 100-106

Li, G., Muthusamy, V., & Jacobsen, H.-A. (2010). A Distributed Service Oriented Architecture for Business Process Execution. *ACM Transactions on the Web (TWEB), 4*(1), 1-33.

Liu, D., & Deters, R. (2007). *BUST: enabling scalable service orchestration.* Paper presented at the 2nd international conference on Scalable information systems (InfoScale '07), Suzhou, China, 1-10.

Mamdani, E., & Assilian, S. (1975). An experiment in linguistic synthesis with a fuzzy logic controller. *International Journal of Man-Machine Studies, 7*(1), 1-13.

Maurino, A., & Modafferi, S. (2005). Partitioning rules for orchestrating mobile information systems. *Personal and Ubiquitous Computing, 9*(5), 291-300.

Mendling, J., Lassen, K. B., & Zdun, U. (2008). On the transformation of control flow between block-oriented and graph-oriented process modelling languages. *International Journal of Business Process Integration and Management, 3*(2), 96-108.

Microsoft. (2001). XLANG. from http://www.gotdotnet.com/team/xml_wsspecs/xlang-c

Miller, J., Sheth, A., Kochut, K., & Wang, X. (1996). CORBA-based run-time architectures for workflow management systems. *Journal of database Management, 7*(1), 16-16.

Mirkov Viroli, E. D., Alessandro Ricci. (2007). Engineering a BPEL orchestration engine as a multi-agent system. *Journal of Science of Computer Programming*.

Mittasch, C., Weise, T., & Hesselmann, M. (2000). Decentralized control structures for distributed workflow applications. *Integr. Comput.-Aided Eng.*, *7*(4), 327-341.

Mohan, A., Alonso, G., Mohan, C., Gunthor, R., Agrawal, D., Abbadi, A. E., & Kamath, M. (1995). *Exotica/FMQM: A Persistent Message-Based Architecture for Distributed Workflow Management.* Paper presented at the IFIP WG 8.1 Workgroup Conference on Information Systems Development for Decentralized Organizations (ISD095), Trondheim, Norway, 1-18.

Muthusamy, V., Jacobsen, H.-A., Chau, T., Chan, A., & Coulthard, P. (2009). *SLA-driven business process management in SOA*. Paper presented at the Proceedings of the 2009 Conference of the Center for Advanced Studies on Collaborative Research. 86-100.

Nanda, M. G., Chandra, S., & Sarkar, V. (2004). Decentralizing execution of composite web services. *ACM SIGPLAN Notices*, *39*(10), 170-187.

Nanda, M. G., & Karnik, N. (2003). *Synchronization analysis for decentralizing composite Web services*. Paper presented at the Proceedings of the 2003 ACM symposium on Applied computing. 407-414.

Nitto, E. D., Ghezzi, C., Metzger, A., Papazoglou, M., & Pohl, K. (2008). A journey to highly dynamic, self-adaptive service-based applications. *Automated Software Engg.*, *15*(3-4), 313-341.

Novell. (2011). Suse Linux Enterprise Real Time Extension. from http://www.novell.com/products/realtime/

OASIS. (2011). Advancing Open Standards for the information society. from http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

OMG. (2011a). BPMN Specification. from http://www.bpmn.org/

OMG. (2011b). Common Object Request Broker Architecture (CORBA). from http://www.corba.org/

ORACLE. (2011a). Oracle Service-Oriented Architecture. from http://www.oracle.com/us/technologies/soa/index.html

ORACLE. (2011b). Sun Java Real-Time System. from http://java.sun.com/javase/technologies/realtime/index.jsp

Ouyang, C., Dumas, M., Aalst, W. M. P. V. D., Hofstede, A. H. M. T., & Mendling, J. (2006). From business process models to process-oriented software systems: The BPMN to BPEL way. *ACM Transactions on Software Engineering and Methodology (TOSEM), 19*(1), 1-37.

P. Muth, D. Wodtke, J. Weisenfels, & A. K. Dittrich, a. G. W. (1998). From centralized workflow specification to distributed workflow execution. *Journal of Intelligent Information Systems*, 10(12):159-184.

Papazoglou, M. P., Traverso, P., Dustdar, S., & Leymann, F. (2007). Service-oriented computing: State of the art and research challenges. *Computer, 40*(11), 38-45.

Papazoglou, M. P., Traverso, P., Dustdar, S., & Leymann, F. (2008). Service-oriented computing: A Research Roadmap. *International Journal of Cooperative Information Systems, 17*(2), 223-255.

Perez-Palacin, D., Merseguer, J., & Bernardi, S. (2010). *Performance aware open-world software in a 3-layer architecture*. Paper presented at the Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering. 49-56.

Philip, B., & Nigel, W. (2003). *JavaSpaces in practice*: Addison-Wesley

Reese, C., Ortmann, J., Offermann, S., Moldt, D., Markwardt, K., & Carl, T. (2006). Fragmented workflows supported by an agent based architecture. *Agent-Oriented Information Systems III, Volume 3529/2006*, 200-215.

Richards, M., Monson-Haefel, R., & Chappell, D. A. (2009). *Java Message Service, Second Edition*: O'Reilly Media

Sadiq, W., Sadiq, S., & Schulz, K. (2006). *Model driven distribution of collaborative business processes*, 281-284.

Sheng, Q., Benatallah, B., Dumas, M., & Mak, E. (2002). *SELF-SERV: A platform for rapid composition of web services in a peer-to-peer environment*, 1054.

Sheth, A. P., Kochut, K., Miller, J. A., Worah, D., Das, S., Lin, C., Palaniswami, D., Lynch, J., & Shevchenko, I. (1996). *Supporting State-Wide Immunisation Tracking Using Multi-Paradigm Workflow Technology*. Paper presented at the Proceedings of the 22th International Conference on Very Large Data Bases. 263-273.

Singh, M., & Huhns, M. (2005). *Service-oriented computing: semantics, processes, agents*: John Wiley & Sons Inc

Sugeno, M., & Kang, G. (1988). Structure identification of fuzzy model. *Fuzzy sets and systems, 28*(1), 15-33.

Syed A. Ahson, & llyas, M. (2011). *Cloud Computing and Software Services: Theory and Techniques*: CRC Press

Tan, W., & Fan, Y. (2005). Model fragmentation for distributed workflow execution: A petri net approach. *Advanced Distributed Systems*, 207-214.

Tan, W., & Fan, Y. (2007). Dynamic workflow model fragmentation for distributed execution. *Comput. Ind., 58*(5), 381-391.

Tomcat, A. (2011). The Apache Software Foundation - Apache Tomcat. from http://tomcat.apache.org/

Tsai, C.-H., Huang, K.-C., Wang, F.-J., & Chen, C.-H. (2010). A distributed server architecture supporting dynamic resource provisioning for bpm-oriented workflow management systems. *Elsevier Journal of Systems and Software, 83*(8), 1538-1552.

UDDI-Consortium. (2001). UDDI_Executive_White - UDDI Executive White Paper. from http://www.uddi.org/pubs/UDDI_Executive_White_Paper.pdf

Van der Aalst, W. M. P., Van Dongen, B. F., Herbst, J., Maruster, L., Schimm, G., & Weijters, A. (2003). Workflow mining: A survey of issues and approaches. *Data & Knowledge Engineering, 47*(2), 237-267.

Vanhatalo, J., Völzer, H., & Koehler, J. (2009). The refined process structure tree. *Data & Knowledge Engineering, 68*(9), 793-818.

W3C. (2011a). Web Services Architecture Requirements. from http://www.w3.org/standards/

W3C. (2011b). Web services description language (WSDL) version 1.2 part 1: Core language. *World Wide Web Consortium, Working Draft WD-wsdl12-20030611*, from http://www.w3.org/TR/wsdl20/

Walmsley, P. (2009). *XQuery Search Across a Variety of XML Data*: O'Reilly Media

WFMC. (2011). Workflow Management Coalition. from http://www.wfmc.org/

Wodtke, D., Weißenfels, J., Weikum, G., & Dittrich, A. K. (1996). *The Mentor project: Steps toward enterprise-wide workflow management.* Paper presented at the International Conference on Data Engineering (ICDE'96), 556–565.

Wutke, D., Martin, D., & Leymann, F. (2008, March 16-20). *Model and Infrastructure for Decentralized Workflow.* Paper presented at the ACM Symposium on Applied Computing (SAC 2008), Fortaleza Cerara, Brazil, 90-94.

Yan, J., Yang, Y., & Raikundalia, G. (2003). Enacting business processes in a decentralised environment with p2p-based workflow support. *Advances in Web-Age Information Management*, 290-297.

Yildiz, U., & Godart, C. (2007, March 11-15). *Towards decentralized service orchestrations.* Paper presented at the Proceedings of the 2007 ACM symposium on Applied Computing (SAC '07), Seoul, Korea, 1662-1666.

Zadeh, L. (1965). Fuzzy sets*. *Information and control*, *8*(3), 338-353.

APPENDIX A

Motivator Examples of Using Mining-based Decentralization; Intelligent Process Decentralization (IPD)

In this section, some motivator examples are presented for several very simple BPEL activities to show how intelligent process decentralization (IPD) applying a frequent-path mining may improve the response-time of business processes compared to the FPD fragmentation method. The response-time of each example is analyzed and modeled by two formulas for before and after using process mining in fragmentation. The formulas are then extrapolated to show the behavior of response-time when the process mining increase the possibility of detecting more relevant activities in fragments.

Owing to the fact that a number of activities in a process are juxtaposed and closely related, the activities of a process call their spatial neighbors based on the process metaphor. Hereafter, the BPEL activities are examined one by one to show how process mining playing role is. For more simplicity, equal execution time (t_A) and communication time (t_C) for all activities are considered. In addition, the time cost of message passing is much more than the execution time of one activity $(t_A \gg t_C)$ except for those activities like invoke, receive or reply, which send or receive messages to external web services. Indeed, we presume $t_C \approx 1000t_A$ and $t_A = 1$ everywhere the numerical value of (t_A) and (t_C) are needed.

1) If (Switch) Activity

The If (Switch) activity is a structured activity and is built based on an if-elseif-else format. According to (Active-Endpoints, 2011; OASIS, 2011), the If activity lets

you choose exactly one execution path from among many such as If structure in common programming languages.

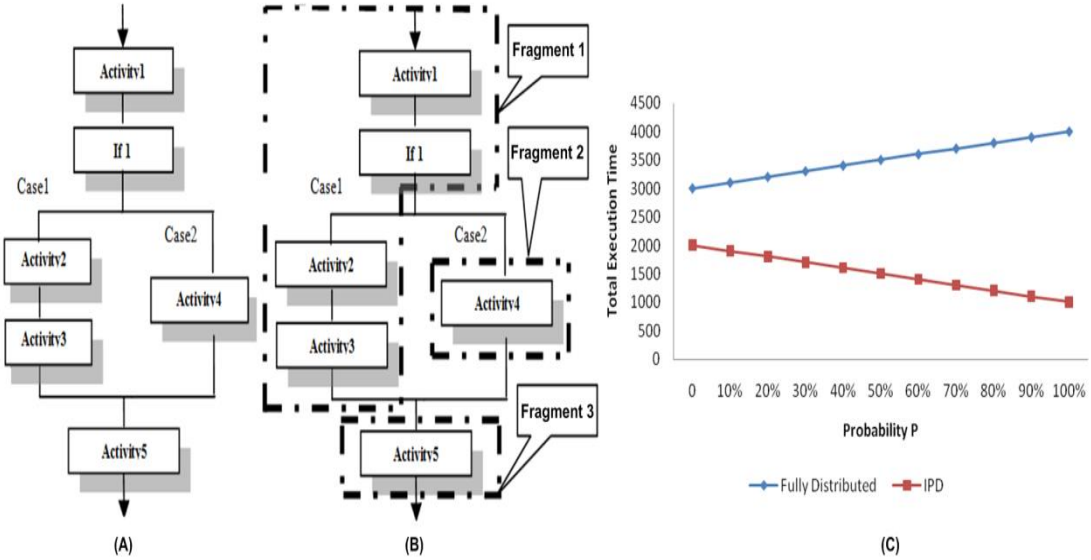


Figure A.1: Using If Activity in Two Methods

According to the FPD fragmentation method, the sample BPEL process in Figure A.1 (a) should be converted to six fragments. The execution time of the above If activity with probability of *p* for executing *Case1* and probability of *1-p* to execute *Case2* is shown in Equation (A.1).

$$t_{total} = t_A + t_C + t_{if} + p(t_C + t_{A2} + t_C + t_{A3}) + (1-p)(t_C + t_{A4}) + t_C + t_{A5} \approx 4t_A + 3t_C + p(t_A + t_C) \quad \text{(A.1)}$$

Suppose the recommended process mining method detects the execution path including Activity1, If1, Activity2 and Activity3 is more traversed than the alternative path including Activity3. Indeed, three fragments should be appeared according to Figure A.1 (b). The total time for processing the If activity is calculated based on Equation (A.2).

$$t_{total} = p\left(t_{A1} + t_{if1} + t_{A2} + t_{A3}\right) + (1-p)\left(t_{A1} + t_{if1} + t_C + t_{A4}\right) + t_C + t_{A5} \approx p \times 4t_A + (1-p)(3t_A + t_C) + t_A + t_C \quad \text{(A.2)}$$

Obviously, the higher number of activities increases the communication time in the fully process distributed (FPD) approach. It is also the process mining algorithm that

detects the probability of p is high and reduced t_{total}. Figure A.1(c) compares the behavior of two approaches.

2) While Activity

In BPEL specification, three loop structures are considered among which While loop is selected. The While repeats the enclosed statement block until the conditional statement evaluates to False (Active-Endpoints, 2011; OASIS, 2011).

A sample *While* activity is shown in Figure A.2 Figure(a). According to the FPD method, a *While* activity is converted to a *While* fragment. Evaluating the condition, the While fragment triggers the subsequent activity. We assume that the possibility of repeating the loop for *n* times is *p* and with possibility of *(1-p)* the loop condition will not be satisfied. The total execution time for While is shown in Equation (A.3).

$$\begin{aligned} t_{total} &= t_{A1} + t_C + p \times n \times (t_W + t_{C+}t_{A3} + t_C + t_{A3} + t_C) \\ &\quad + (1-p)(t_W + t_C) + t_{A4} \\ &\approx 2t_A + t_C + p \times n \times (3t_A + 3t_C) + (1-p)(t_A + t_C) \end{aligned} \tag{A.3}$$

Again using the mining approach, the activities in the While structure will be encapsulated in three fragments owing to the fact that they are closely related as shown in Figure A.2 (b). Indeed, the total execution time using the mining approach will be calculated according to Equation (A.4).

$$\begin{aligned} t_{total} &= p \times n \times (t_{A1} + t_W + t_{A2} + t_{A3}) + (1-p)(t_{A1} + t_W) + t_C + t_{A4} \\ &\approx p \times n \times 4t_A + (1-p)(2t_A) + t_A + t_C \end{aligned} \tag{A.4}$$

Based on the assumption that process mining has already detected that the probability of *p* is high; therefore, better result is expected for the FPD model. For this While activity the number of loop iterations is assumed *n=100*. Obviously, the

output result is highly dependent on the value of *n*. Figure A.3 (c) shows the comparison charts of two methods.

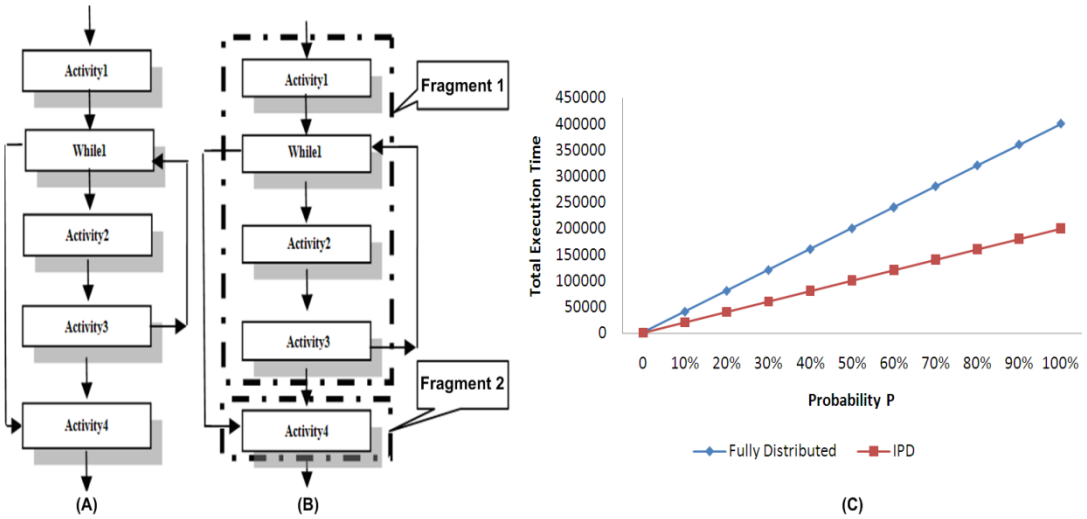


Figure A.2: Using While Activity in two Methods

3) Flow Activity

The basic functionality of Flow activity is that it may include several activities executed concurrently. The Flow activity also allows you to synchronize activities such that one activity starts when another ends. We've dealt with activities that implement *program-like* structures until now, but with Flow *graph-like* structures are examined (Active-Endpoints, 2011; OASIS, 2011).

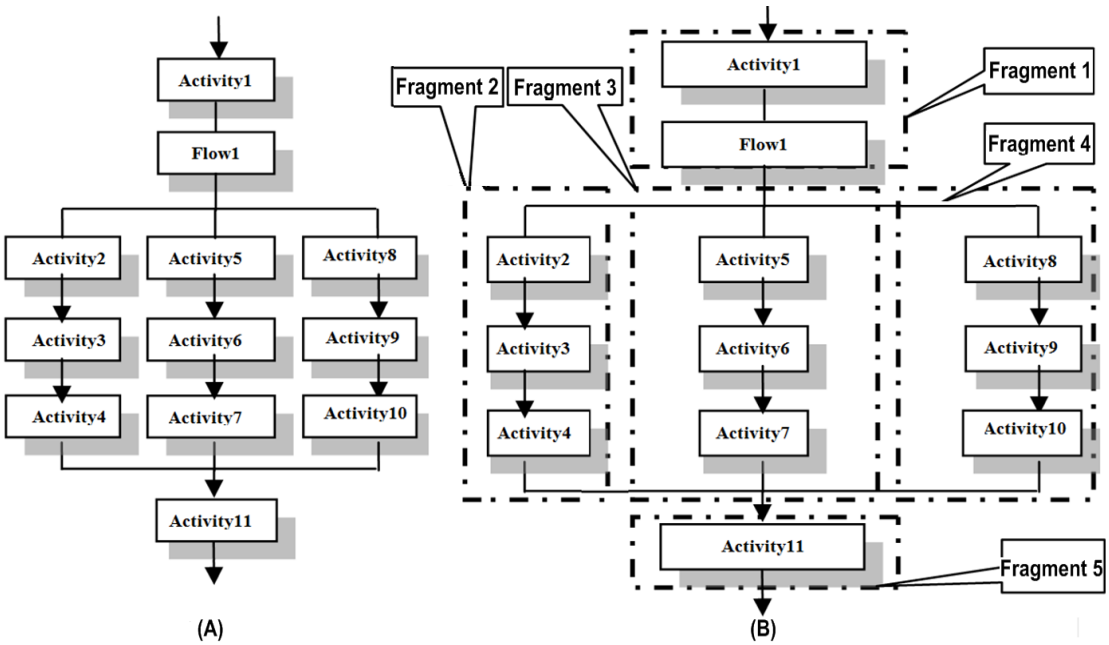


Figure A.3: Using Flow Activity in Two Methods

Figure A.4 (a) shows a sample Flow activity in its simplest use. Accordingly, there are three branches in the Flow structure, which are able to run concurrently. Based on what is recommended in the FPD, this sample structure will be compiled to twelve fragments. The total time for processing this Flow activity is illustrated in Equation (A.5):

$$\begin{aligned} t_{total} = t_{A1} + t_F + t_C \\ + \max[(t_{A2} + t_C + t_{A3} + t_{A4}) \vee (t_{A5} + t_C + t_{A6} + t_C \\ + t_{A7}) \vee (t_{A8} + t_C + t_{A9} + t_C + t_{A10})] + t_C + t_{A11} \\ \approx 6t_A + 5t_C \end{aligned} \tag{A.5}$$

In contrast, the mining approach results in three groups of closely related activities. Indeed, the normal way of handling this situation is creating one fragment for each group and the total number of produced fragments is shown in Figure A.5 (b). The total execution time of the Flow activity using the mining approach is calculated based on Equation (A.6).

$$\begin{aligned} t_{total} = t_{A1} + t_F + t_C \\ + \max[(t_{A2} + t_{A3} + t_{A4}) \vee (t_{A5} + t_{A6} + t_{A7}) \\ \vee (t_{A8} + t_{A9} + t_{A10})] + t_C + t_{A11} \approx 6t_A + 2t_C \end{aligned} \tag{A.6}$$

Consequently, mining approach improves the total execution time of Flow activity according to $6t_A + 2t_C \ll 6t_A + 5t_C$. Obviously, increasing the number of activities worsen the communication time in the FPD approach.

4) Pick Activity

The Pick activity forces the process to wait until one event is triggered. All of these events are either *onMessage* or *onAlarm* elements. There must be several onMessage and onAlarm activities; however, exactly one of them will be executed. Once one event is executed, the others are disabled (Active-Endpoints, 2011; OASIS, 2011).

Figure A.5 (a) shows a sample Pick activity including two onMessage and onAlarm elements. In its normal distribution according to the FPD, it should be converted to four fragments. The probability of running onMessage is *p* unless *OnAlarm* would be run. Consequently, the total time for running this process is obtained by Equation (A.7).

$$t_{total} = t_{A1} + t_C + t_p + p(t_C + t_{A2} + t_C + t_{A3}) + (1-p)(t_C + t_{A4}) \approx 3t_A + 2t_C + p(t_A + t_C) \quad \text{(A.7)}$$

Suppose that mining process approach detects Activity1, Pick1, Activity2 and Activity3 are closely interrelated through *OnMessage* element. The BPEL fragments are shown in Figure A.5 (b). The total execution time of the Pick activity is calculated as Equation (A.8).

$$t_{total} = p(t_{A1} + t_{P+}t_{A2} + t_{A3}) + (1-p)(t_{A1} + t_{P+}t_C + t_{A4}) \approx p \times 4t_A + (1-p)(3t_A + t_C) \quad \text{(A.8)}$$

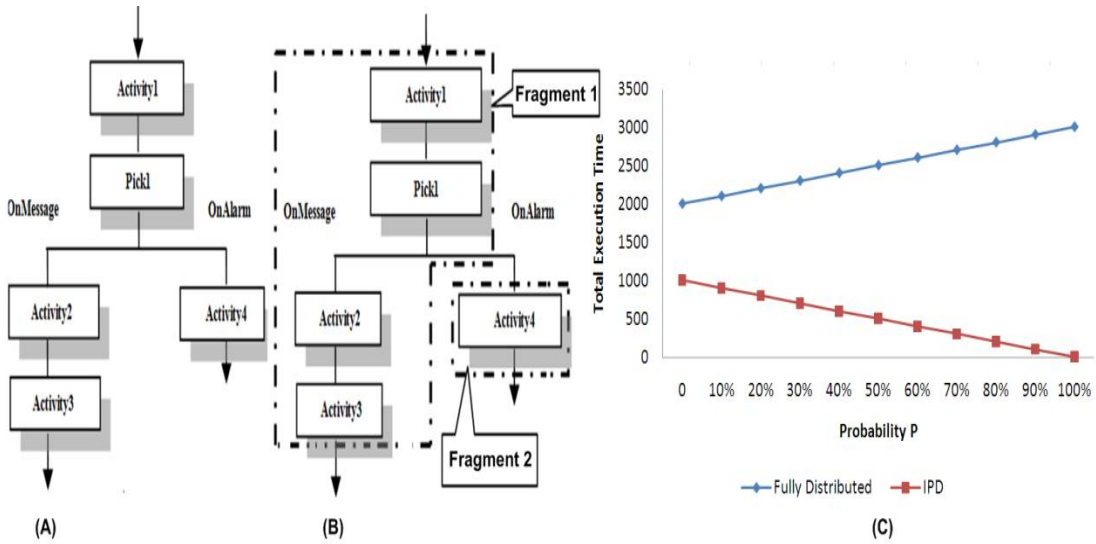


Figure A.4: Using Pick Activity in Two methods

Obviously, increasing the number of activities worsen the communication time in the FPD approach. Considering the hypothesis that process mining has already detected the high probability of *p*; therefore, total execution time is expected to be decreased. Besides, Figure A.5 (c) shows the comparison of two mentioned methods for Pick activity.

5) Scope, Compensate and Fault Activities

Scopes allow us to break up a business processes into logical units of work. In fact, Scopes provide a context for the execution and/or documentation of enclosed activities, and they can have variables that are visible and usable within the Scope level. Scopes may have both default and defined Fault and Event handling logic, and they can be undone, if necessary. Also, undoing the work of a Scope involves the concept of compensation. When designing your BPEL processes are designed, they should be organized into logical units of work that can be undone (Active-Endpoints, 2011; OASIS, 2011).

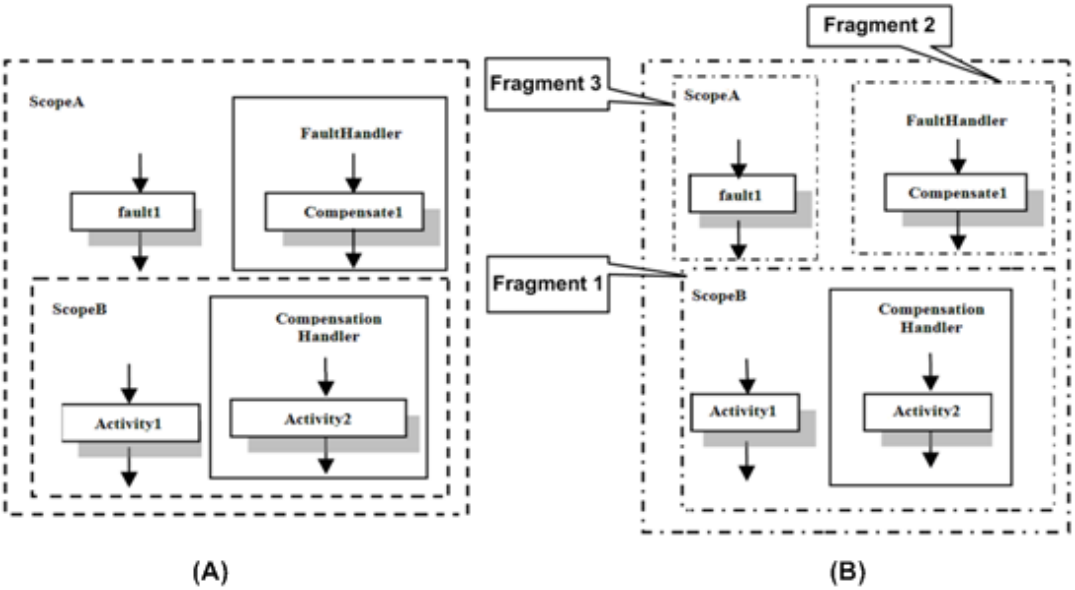


Figure A.5: Scope, Compensate and Fault

From Compensate point of view, previous activities, which had successfully completed, may need to be undone. In addition, business processes are intrinsically long-running and asynchronous. In BPEL, a Compensation Handler is a local declaration, done at the Scope level only, and is not normally available for the process itself. However, designers add extensions to the scope to facilitate process level compensation handling. In contrast, Fault Handlers enable the process to recover from abnormally terminated actions, whereas Compensation Handlers undo

successfully completed actions. The other difference is detecting the context from which they are called. A BPEL Fault Handler is invoked in response to a Fault in the same context (i.e., the same Scope), whereas Compensation Handlers are invoked in response to a Fault in a higher context (i.e., the parent Scope). Moreover, the order in which compensation handlers are performed usually makes a difference. By default, compensation is performed in the reverse order of the completion of involved Scopes (Active-Endpoints, 2011; OASIS, 2011).

According to the FPD a Scope activity will be converted to a fragment and the fragment will be in contact with its included activities, Fault and Compensation handlers through a middleware. From mining approach point of view, the Scope activity might be considered as a separate activity or might be encapsulated with its Compensate and Fault handlers.

According to the FPD, a simple Compensate shown in Figure(a) will be converted to two Scope fragments A and B. Scope A's Fault handler involves the compensation handler in the Scope B. Indeed, Scopes as well as Compensates have been encapsulated into fragments.

If the mining algorithm shows that the Compensation Handler in the Scope B is frequently used, then it will be possible to encapsulate Activity 1 along with Activity 2 and Scope B all together in a single fragment to reduce the number of fragments. The produced fragments using the mining approach is drawn in Figure A.5 (b) for more illustration.

In this case encapsulating Fault or Compensate handlers in one fragment omits communication costs of the fragments and it results in decreasing the total execution time of the Scope activities and its included Compensate and Fault Handlers.

BIO DATA OF STUDENT

Faramarz Safi Esfahani was born in Iran, September 1974. He received his B.S. degree in Software Engineering in 1998 and M.S. in Distributed Systems in 2001 from Islamic Azad University (IAU) of Najafabad (IAUN), Esfahan, Iran. He has been on faculty of IAUN since 2002 in department of computer science and information technology. Simultaneously, through collaboration with Jooya Informatics Company, he was an Oracle database administrator in Mobarakeh Steel Company (MSC) during 2002-2003, Senior System Analyst in providing an ICT Master Plan for Medical University of Shahrekord during 2003-2004, Project Manager in providing E-From services in Tabriz city during 2004-2005, Project Manager in providing the portal of Mobarakeh Steel Company during 2005, Project Manager in providing an SOA infrastructure and producing governmental E-Services (E-Government) in Rasht city during 2005-2006. Through collaboration with BASA Informatics Company he was the Technical Manager in developing an SOA infrastructure for Statistical Centre of Iran (SCI), in Tehran capital city during 2006-2007. In addition, he collaborated with the research center of Esfahan University in authoring a reference book entitled "RUP Methodology to Develop Software Systems" published in 2004 and collaborated with BASA Informatics Company as an author of the book entitled "Software Engineering and Service Oriented Architecture", published in 2008. In 2007, he joined to faculty of computer science and information technology, University of Putra Malaysia (UPM) to persuade his PhD. During his study, he investigated the adaptability of business process decentralization with runtime environment. His field of interests are E-Government, Self-* systems, Distributed systems (Service oriented and Cloud computing), Soft Computing (Semantic web, Software agents, and intelligent techniques).

LIST OF PUBLICATIONS

Journal Papers	
Under Review	"Self-adaptation of Business Process Decentralization with Runtime Environment", Faramarz Safi Esfahani, Masrah Azrifah Azmi Murad, Md.Nasir B. Sulaiman, Nur Izura Udzir, Submitted to the Elsevier Journal of Systems and Software (JSS), Especial Issue on State of the Art in Self-Adaptive Software Systems, March 31 2011.
March 2011	"Adaptable and Decentralized Service Oriented Architecture", Faramarz Safi Esfahani, Masrah Azrifah Azmi Murad, Md.Nasir B. Sulaiman, Nur Izura Udzir, Accepted by the Elsevier Journal of Systems and Software (JSS), http://dx.doi.org/10.1016/j.jss.2011.03.031, 2011.
December 2008	"An Intelligent Business Process Distribution Approach", Faramarz Safi Esfahani, Masrah Azrifah Azmi Murad, Md. Nasir B. Sulaiman, Nur Izura Udzir, Journal of Theoretical and Applied Information Technology, Vol 4. No12. 2008.
Conference Proceedings	
February 2011	"Run-time Adaptable Business Process Decentralization", Faramarz Safi Esfahani, Masrah Azrifah Azmi Murad, Md. Nasir B. Sulaiman, Nur Izura Udzir – IARIA/eKNOW'11, Information, Process, and Knowledge Management, 2011, France.
October 2009	"A Case Study of The Intelligent Process Decentralization Method", Faramarz Safi Esfahani, Masrah Azrifah Azmi Murad, Md. Nasir B. Sulaiman, Nur Izura Udzir, IAENG/WCECS, World Congress on Engineering and Computer Science, 2009, San Francisco, USA.
July 2009	"A Frequent Path Mining Algorithm to Intelligent Business Process Decomposition", Faramarz Safi Esfahani, Masrah Azrifah Azmi Murad, Md. Nasir B. Sulaiman, Nur Izura Udzir, WORLDCOMP/SWWS'09, The 2009 International Conference on Semantic Web and Web Services, 2009, Las Vegas, Nevada, USA.
March 2009	"Using Process Mining to Business Process Distribution", Faramarz Safi Esfahani, Md. Nasir B. Sulaiman, Nur Izura Udzir, Proceedings of the ACM Symposium on Applied Computing, 2009, Honolulu, Hawaii, USA.
February 2009	"SLA-Driven Business Process Distribution", Faramarz Safi Esfahani, Masrah Azrifah Azmi Murad, Md. Nasir B. Sulaiman, Nur Izura Udzir, IEEE/IARIA/eKNOW'09, Information, Process, and Knowledge Management, 2009, Mexico.
Grants and Awards	
2007-2010	Graduate Research Fellowship (GRF) – University of Putra Malaysia (UPM), 43400, Serdang, Selangor, Malaysia.
February 2010	Won a Bronze medal in Malaysia Technology Expo (MTE), for the Invention/Innovation of An Adaptable Business Process Execution Engine, February 2010, Kuala Lumpur, Malaysia.
July 2009	Won a Gold medal in PRPI (Exhibition in Design, Research and Innovation) for the Invention/Innovation of An Adaptable Business Process Execution Engine, July 2009, University of Putra Malaysia (UPM), 43400, Serdang, Selangor, Malaysia.