

Adaptation of Business Rules in Business Workflow Systems

This dissertation is submitted for the degree
of

Doctor of Philosophy

By
Kanana Ezekiel

Date: May 2021



School of Computing and Digital Media
London Metropolitan University

166-220 Holloway Road, London, N7 8DB

Acknowledgement

This PhD thesis is a result of four years and ten months of hard work carried out at London Metropolitan university. The results of this research would not have been conceivable without the help of others.

First and foremost, I would like thank God for everything, thank you Lord Jesus. Second, my gratitude goes to my supervisors at the university, Professor Karim Ouazzane and Dr Vassil Vassilev for their excellent supervision of this research. This work has benefitted from great ideas, useful discussions, guidance and valuable comments on how to advance my work further. I'm grateful for keeping sure this research stayed on the right course. Also, I would like to thank my fellow students at London Metropolitan University for various discussions and exchanges of ideas. Special thanks to Joe and Gill Hayden for their excellent editing and proofreading of the thesis.

I am also grateful for the funding received from Veritiv Corporation formerly Emerson Power and Network (Avocent International Ltd) and especially to Lori DeMatteis (formerly Vice-President at Emerson Power and Network), who approved my funding and has been supportive of my career.

Last but not least, I would like to thank my parents (Mr and Mrs S J Ezekiel), family and friends for believing in me, encouraging and supporting me in prayers.

Kanana Ezekiel, Sep 2020

Abstract

The term “workflow” is widely recognised in the business community. Workflows are commonly seen as the top priority for companies wanting to survive the current competitive markets. A business management system with configurable workflow provides many benefits including not only a central repository for the way companies do business but also boosts teams efficiency with structured processes, process automation which means errors are greatly reduced, less time is needed for training, fewer repetitive tasks and many more. Despite the many benefits that workflows bring, the complexities of configuring workflows cause major roadblocks for companies moving towards workflow solutions. The need for having configurable workflows in dynamic environments have been discussed and well documented by various authors in research communities. In the existing research, there is a lack of support on accounting for business rules dependencies within workflows. Without accounting the logical dependencies, we cannot have an efficient mechanism for business rules adaptation in real-time. To tackle the configuration problem, this research proposed a business rule component-based formal model for development of business workflows. The formal model accounts for logical dependencies between business rules in the form of AND-OR graphs. The graphs are created through Event, Condition and Action (ECA) components of business rules. The business rule change propagation is implemented as an algorithm of graph traversal through the AND-OR graph patterns. A two-levels inference mechanism is built as a vehicle for controlling the business process execution and adaptation of the business rules at real time based on propagating changes between business rules dependencies. The major advantage of our research is the universal, strictly logic-based event-driven framework for business process modelling and control which allows automatic adaptation of the business rules governing the business workflows based on accounting their structural dependencies. The framework is entirely domain-independent and can be used across industries.

Author Related Publications

- Ezekiel K., Vassilev, V., Ouazzane, K. and Patel, Y. (2019), "Adaptive business rules framework for workflow management", Business Process Management Journal, Vol. 25 No. 5, pp. 948-971 publisher by Emerald Publishing Limited, <https://doi.org/10.1108/BPMJ-08-2017-0219>.
- Kanana Ezekiel, Vassil Vassilev, Karim Ouazzane (2018) "Two-level Architecture for Rule-based Business Process Management". BUSTECH 2018, The Eighth International Conference on Business Intelligence and Technology February 18, 2018 to February 22, 2018 - Barcelona, Spain. https://www.researchgate.net/publication/325115311_Two-level_Architecture_for_Rule-based_Business_Process_Management

Table of Contents

Acknowledgement	1
Abstract	2
Author Related Publications	3
1. Introduction.....	14
1.1 Motivation.....	14
1.2 Research Hypotheses	17
1.3 Aim and Objectives.....	18
1.4 Research Methodology	21
1.4.1 Analytical Method	21
1.4.1.1 Business Rules Acquisition	21
1.4.1.2 Business Rules Preparation and Classification.....	22
1.4.2 Constructive Method.....	22
1.4.2.1 Business Rules Model Development.....	22
1.4.2.2 Business Rules Model Optimisation.....	23
1.4.2.3 Productionisation	23
1.4.3 Experimental Method.....	23
1.5 Thesis Structure	24
2. Literature and Applications Review	26
2.1 Existing Research Studies.....	26
2.1.1 Business Rules in Workflows	27
2.1.2 Business Rules Formal Models in Workflows.....	31
2.1.3 Change Propagation and Adaptation Algorithms	32
2.1.4 Indexing Mechanisms	34
2.2 Vendors Applications and Systems	35
2.2.1 IBM BRMS and BPM.....	36
2.2.2 CLIPS.....	36
2.2.3 JESS	37
2.2.4 ORACLE BRMS and BPM	37
2.2.5 JBOSS DROOLS BRMS and jBPM.....	37
2.2.6 OpenRules.....	38
2.2.7 PRRP & SBVR	38
2.3 Summary	38
3. Business Rules, Process and Workflows	41

3.1	Definition of Business Rule and BRMS	41
3.2	Definition of Business Process, Workflow and BPMS	42
3.3	Workflow Patterns	42
3.3.1	Sequence (Serial) Workflow Patterns	43
3.3.2	Parallel Workflow Patterns	43
3.4	Business Rules Structure.....	43
3.5	Business Rules Ontology	44
3.6	Summary	48
4.	Proposed Business Rules Model.....	50
4.1	Methodological foundation of the framework	50
4.2	Two-Levels Architecture	52
4.3	Business Rules Classification	55
4.4	Business Process Ontology and Formal Specification.....	58
4.5	Business Rules Relationships	68
4.5.1	Business Rules Formal Description	68
4.5.2	Relationships between Business Rules	69
4.5.3	Business Rules Dependency Graphs (AND-OR Graphs).....	71
4.5.3.1	Direct AND Dependency patterns	72
4.5.3.2	Direct OR Dependency patterns	75
4.5.3.3	Indirect AND Dependency patterns	78
4.5.3.4	Indirect OR Dependency patterns.....	82
4.5.4	Business Rules Dependency Patterns	82
4.5.5	Application of AND-OR Graphs	84
4.6	Summary	86
5.	Software Architecture, Metarules and Indexing	87
5.1	Software Architecture	87
5.2	Metarules.....	88
5.3	Indexing of Business Rules.....	91
5.3.1	Index Structure.....	92
5.3.2	Path Dependency Pattern Indexing.....	93
5.4	Summary	96
6.	Change Propagation and Adaptation Algorithms	97
6.1	Business Rules Change Propagation.....	97
6.2	Change Propagation via Dependency Patterns	99

6.2.1	Path Dependency Propagation	99
6.2.2	Direct-Node Dependency Propagation	105
6.2.3	Level-Based Dependency Propagation	105
6.2.4	Neighbour Dependency Propagation	105
6.2.5	Indirect Node Dependency Propagation	105
6.3	Algorithm for Business Rules Change Propagation	106
6.4	Business Rules Adaptation in Business Workflows	109
6.4.1	Initiating Business Rule	112
6.4.2	Terminating Business Rule	112
6.4.3	Sequential Flow Business Rules	113
6.4.4	Parallel AND-Split Flow Business Rules	113
6.4.5	Parallel OR-Split Flow Business Rules	114
6.4.6	Parallel AND-Merge Flow Business Rules	115
6.4.7	Parallel OR-Merge Flow Business Rules	116
6.5	Algorithm for Business Rules Adaptation in Workflows	117
6.6	Summary	122
7.	Implementation	124
7.1	Drools Overview	124
7.2	ECA Model Prototype.....	125
7.2.1	Business Rule Classes	127
7.2.2	Business Rule Template.....	131
7.2.3	Indexing Path Dependency Patterns	132
7.2.4	Change Propagation and Adaptation Algorithms	135
7.2.4.1	Business Rules Change Propagation Algorithm.....	135
7.2.4.2	Business Rules Adaptation Algorithm	136
7.2.5	Business Rules Editor (ECA Model Test Client)	140
7.3	Summary	144
8.	ECA Model Validation	145
8.1	Validation Criteria	146
8.2	Data Centre Use cases.....	148
8.2.1	Add Business Rules Components and Propagate Change	149
8.2.2	Update Business Rules Components	154
8.2.3	Delete Business Rules Components.....	155
8.2.4	Enable Business Rules to Initiate and Terminate Workflow	156

8.2.5	Sequential Flow Patterns.....	158
8.2.6	Parallel-OR Merge Flow Patterns.....	160
8.2.7	Parallel-AND Merge Flow Patterns.....	161
8.2.8	Parallel-OR Split Flow Patterns.....	163
8.2.9	Parallel-AND Split Flow Patterns.....	164
8.3	Experiments	165
8.3.1	Validation of Dynamic Business Rules and Change propagation	167
8.3.1.1	Adding Business Rules Components & Change Propagation	167
8.3.1.2	Modifying Business Rules Components.....	174
8.3.1.3	Deleting Business Rules & Components	177
8.3.2	Validation of Business Rules Adaptation in Workflows	178
8.3.2.1	Enabling Business Rules Components to Initiate & Terminate Workflow	178
8.3.2.2	Enabling Sequential Process Flow Patterns.....	181
8.3.2.3	Enabling Parallel-OR Merge Process Flow Patterns	188
8.3.2.4	Enabling Parallel-AND Merge Process Flow Patterns	195
8.3.2.5	Enabling Parallel-OR Split Process Flow Patterns	202
8.3.2.6	Enabling Parallel-AND Split Process Flow Patterns.....	209
8.4	Summary	216
9.	Conclusion and Future Research	222
9.1	Reflection on Research Questions	222
9.2	Reflection on Aim and Objectives.....	224
9.3	Contributions to the Knowledge	226
9.4	Limitations and Future Research	228
	References.....	230
	Appendices.....	241
	Appendix I – Possible Validation Scenarios.....	241
	Appendix II – Business Rules in XYZ Equipment Install Workflow	244
	Appendix III – XYZ Business Rules in Drools DRL format.....	245
	Appendix IV – XYZ Business Rules Insertion via Drools DRL	247
	Appendix V – Level-Based Dependency Pattern Index Algorithm.....	250
	Appendix VI – JBoss Drools Setup and Installation	252
	Appendix VII – JBoss Drools Components.....	255
	Appendix VIII – Editing Business Rules via Test Client	257

List of Tables

Table 3.5. 1 Ontology Concepts Description.....	47
Table 3.5. 2 OWL Concepts	47
Table 5.3. 2 Business rule – rack utilization exceeds by data centre location.....	94
Table 6.5. 1 Business Rules Dependency Mapping Table.....	118
Table 7.2. 1 Description of Core ECA Model Classes	129
Table 8.1. 1 Research Objectives, Challenges, Validation Criteria and Experiments.....	145
Table 8.1. 2 Validation Criteria Description.....	147
Table 8.2. 1 Use cases from DC Workflow	149
Table 8.2. 2 Use case #1 - Business Rule Components (ECA)	153
Table 8.2. 3 Use case #1 - Inserted Business Rule (R13)	154
Table 8.2. 4 Use case #2 - Business Rule Components (ECA)	155
Table 8.2. 5 Use case #4 - Business Rule Components (ECA)	158
Table 8.2. 6 Use case #5 - Business Rule Components (ECA)	159
Table 8.2. 7 Use case #6 - Business Rule Components (ECA)	161
Table 8.2. 8 Use case #7 - Business Rule Components (ECA)	162
Table 8.2. 9 Use case #8 - Business Rule Components (ECA)	164
Table 8.2. 10 Use case #9 - Business Rule Components (ECA)	165
Table 8.3. 1 Experiments and Use cases.....	167
Table 8.4. 1 Validation Results.....	221
Table 9.1. 1 Reflection of Research Questions	223
Table 9.2. 1 Status of Achieved Objectives	225

Code Snippets

Code Snippet 3.5. 1 Framework development stages	48
Code Snippet 5.2. 1 Business Rule using DRL Syntax	88
Code Snippet 5.2. 2 Metarule using DRL Syntax.....	89
Code Snippet 5.2. 3 Metarule updating an existing event	89
Code Snippet 5.2. 4 Index creation using Metarule.....	90
Code Snippet 5.3.2. 1 Procedural programming (Java Syntax).....	95
Code Snippet 6.3. 1 Business Rules Change Propagation Algorithm	108
Code Snippet 6.5. 1 Transform Business Rules into Processes.....	120
Code Snippet 6.5. 2 Define Business Rule Dependencies as per Table 6.5.1	121
Code Snippet 6.5. 3 Workflow Creation using Drools APIs	122
Code Snippet 7.2.2. 1 ObjectDataCompiler for Rule Template (Java syntax)	132
Code Snippet 7.2.3.1 1 Indexing Method for Path Dependency Patterns.....	134
Code Snippet 7.2.4. 1 Business Rule Change Propagation Algorithm.....	135
Code Snippet 7.2.4.2. 1 Convert Business Rules into Processes Algorithm	137
Code Snippet 7.2.4.2. 2 Build Dependency Graphs Algorithm.....	138
Code Snippet 7.2.4.2. 3 Generate Workflow Algorithm	140
Code snippet 7.2.5. 1 Demonstrate “Add Rule” Swing Button.....	143

List of Figures

Figure 1. 1 Actual process maturity of the organisations	15
Figure 1. 4 Research Methods	21
Figure 3.5. 1 Ontology graph using Protégée	50
Figure 4. 1 Framework development stages	50
Figure 4.2. 1 Business Process Level.....	52
Figure 4.2. 2 Business Rules Control Level	53
Figure 4.2. 3 Business Rules Classification.....	53
Figure 4.2. 4 Two-Levels Architecture.....	54
Figure 4.3. 1 Initiation Business Rule.....	55
Figure 4.3. 2 Event Business Rule	56
Figure 4.3. 3 Flow Business Rule	57
Figure 4.3. 4 Termination Business Rule.....	57
Figure 4. 4 Workflow and Associated Business Rules (Example)	58
Figure 4.5.2 1 Event to Event Relationships.....	69
Figure 4.5.2 2 Event to Condition Relationships.....	69
Figure 4.5.2 3 Event to Action Relationships	70
Figure 4.5.2 4 Condition to Condition Relationships	70
Figure 4.5.2 5 Condition to Action Relationships	70
Figure 4.5.2 6 Action to Action Relationships	70
Figure 4.5.3.1. 1 Strong Direct Event-AND Graph	72
Figure 4.5.3.1. 2 Strong Direct Condition-AND Graph	73
Figure 4.5.3.1. 3 Strong Direct Action-AND Graph.....	74
Figure 4.5.3.2. 1 Weak Direct Event-OR Graph.....	75
Figure 4.5.3.2. 2 Weak Direct Condition-OR Graph.....	76
Figure 4.5.3.2. 3 Weak Direct Action-OR Graph	77
Figure 4.5.3.3. 1 Strong Indirect Event-AND Graph.....	78
Figure 4.5.3.3. 2 Strong Indirect Condition-AND Graph	80
Figure 4.5.3.3. 3 Strong Indirect Action-AND Graph	81
Figure 4.5. 4 AND-OR Graph with Dependency Patterns.....	84
Figure 4.5. 5 Business Rule Dependency Graph Control Processes.....	86
Figure 5. 1 Software Architecture Diagram.....	87
Figure 5.3. 1 Graph Dependency Index Structure	93
Figure 5.3. 2 Path Dependency Indexing Graph.....	95
Figure 6.2. 1 Business Rules Path Dependencies	101
Figure 6.2. 2 Business Rule 6 Inserted	102
Figure 6.2. 3 Business Rule 3 Deleted.....	103
Figure 6.2. 4 Business Rule 3 Updated.....	104
Figure 6.4. 1 Request Cancellation Workflow.....	110
Figure 6.4. 2 Business Rules modelling Request Cancellation Workflow	110
Figure 6.4. 3 Initiate Workflow via Business Rules	112
Figure 6.4. 4 Terminate Workflow via Business Rules	113
Figure 6.4. 5 Business Rules modelling Sequential Process Flows.....	113
Figure 6.4. 6 Business Rules modelling Parallel AND-Split Process Flows.....	114
Figure 6.4. 7 Business Rules modelling Parallel OR-Split Process Flows.....	115

Figure 6.4. 8 Business Rules modelling AND-Merge Process Flows	115
Figure 6.4. 9 Business Rules modelling OR-Merge Process Flows	116
Figure 6.5. 1 Business Process Nodes Creation.....	117
Figure 6.5. 2 Business Rules Mapping Table to Workflow.....	119
Figure 7.1. 1 Business Logic Integration Platform	124
Figure 7.2. 1 Business Rules Modules Integration within Drools Platform	125
Figure 7.2. 2 Key Implementation Steps	126
Figure 7.2.1. 1 UML Class Diagram showing ECA Model major classes	130
Figure 7.2.2. 1 Business Rules Template Structure	131
Figure 7.2.3. 1 UML Class Diagram for ECAIndexPatternGraph Class.....	133
Figure 7.2.5. 1 ECA Model Test Client	141
Figure 7.2.5. 2 Business Rule and Components Insertion	143
Figure 8.2. 1 DC Floor Plan with Equipment Installed	152
Figure 8.2.2. 1 XYZ Equipment Install Workflow	152
Figure 8.2.2. 2 XYZ Equipment Install Workflow after Deleting R5	156
Figure 8.2.2. 3 Equipment Move Workflow Diagram.....	157
Figure 8.2.2. 4 Sequential Workflow	159
Figure 8.2.2. 5 Parallel-OR Merged Workflow	160
Figure 8.2.2. 7 Parallel-AND Merge Workflow	162
Figure 8.2.2. 8 OR Split Workflow.....	163
Figure 8.2.2. 9 AND Split Workflow	165
Figure 8.3.1. 1 Adding R1 and Components via ECA Model Test Client	169
Figure 8.3.1. 2 Adding R5 Event and Action Components via ECA Test Client.....	170
Figure 8.3.1. 3 Adding R6 Condition and Action Components via ECA Test Client.....	171
Figure 8.3.1. 4 Insert R13's Condition and Action causing Change Propagation.....	173
Figure 8.3.1. 5 Modify R5 and Components via ECA Model Test Client	175
Figure 8.3.1. 6 Modifying R5 Event and Action Components via ECA Test Client.....	176
Figure 8.3.1. 7 Deleting R5 Event and Action Components via ECA Test Client.....	177
Figure 8.3.2. 1 Initiating (R101) and Terminating (R102 & R04) Business Rules	179
Figure 8.3.2. 2 Initiating (R101) and Terminating (R102 & R04) via ECA Test Client.....	180
Figure 8.3.2. 3 R101 causing Start and (R102 & R04) causing End Workflow.....	181
Figure 8.3.2. 4 Business Rules presenting Sequential Relationships	182
Figure 8.3.2. 5 R201, R202 and R203 and Relationships in DRL Format	183
Figure 8.3.2. 6 Business Rules and Relationships enabling Sequential Process Flows ..	183
Figure 8.3.2. 7 Insert R204 and Relationships via ECA Model Test Client.....	184
Figure 8.3.2. 8 Insertion of R204 causing Sequential Process Flows.....	185
Figure 8.3.2. 9 Update R204 and Relationships via ECA Model Test Client	186
Figure 8.3.2. 10 Modification of R204 causing Sequential Process Flows	186
Figure 8.3.2. 11 Delete R204 and Relationships via ECA Model Test Client	187
Figure 8.3.2. 12 Deletion of R204 causing removal of Process P204 and Connections ..	188
Figure 8.3.2. 13 Business Rules presenting Parallel Merge-OR Relationships.....	189
Figure 8.3.2. 14 R301, R302, R303, R304 and Relationships in DRL Format	190
Figure 8.3.2. 15 Rules & Relationships enabling Parallel-OR Merge Process Flows.....	190
Figure 8.3.2. 16 Insert R305 and Relationships via ECA Model Test Client.....	191
Figure 8.3.2. 17 Insertion of R305 causing Parallel-OR Merge Process Flows	192
Figure 8.3.2. 18 Update of R303 and Relationships via ECA Model Test Client.....	193

Figure 8.3.2. 19 Modification of R303 enabling Parallel-OR Merge Process Flows	193
Figure 8.3.2. 20 Deletion of R302 and Relationships via ECA Model Test Client.....	194
Figure 8.3.2. 21 Deletion of R302 causing Sequential Process Flows	195
Figure 8.3.2.21. 1 Business Rules presenting Parallel-AND Merge Relationships.....	196
Figure 8.3.2. 22 R301, R302, R303, R304 and Relationships in DLR format	197
Figure 8.3.2. 23 Rules & Relationships enabling Parallel-AND Process Flows	197
Figure 8.3.2. 24 Insert R305 and Relationships via ECA Model Test Client.....	198
Figure 8.3.2. 25 Insertion of R305 causing Parallel-AND Process Flows	199
Figure 8.3.2. 26 Update R303 and Relationships via ECA Model Test Client	200
Figure 8.3.2. 27 Modification of R303 causing Parallel-AND Merge Process Flows	200
Figure 8.3.2. 28 Delete R302 and Relationships via ECA Model Test Client	201
Figure 8.3.2. 29 Deletion of R302 causing Sequential Process Flows	202
Figure 8.3.2. 30 Business Rules presenting Parallel-OR Split Relationships.....	203
Figure 8.3.2. 31 R404, R405, R406 and Relationships in DLR format.....	204
Figure 8.3.2. 32 Rules & Relationships enabling Parallel-OR Split Process Flows.....	204
Figure 8.3.2. 33 Insert R407 and Relationships via ECA Model Test Client.....	205
Figure 8.3.2. 34 Insertion of R407 causing Parallel-OR Process Flows.....	206
Figure 8.3.2. 35 Update R405 and Relationships via ECA Model Test Client	207
Figure 8.3.2. 36 Updating R404 causing Parallel-OR Split Process Flows.....	207
Figure 8.3.2. 37 Delete R404 and Relationships via ECA Model Test Client	208
Figure 8.3.2. 38 Deletion of R404 causing removal of P404 and Connections.....	209
Figure 8.3.2. 39 Business Rules presenting Parallel-OR Split Relationships.....	210
Figure 8.3.2. 40 R501, R502, R503, R504 and Relationships in DRL Format	211
Figure 8.3.2. 41 Rules & Relationships enabling Parallel-AND Split Process Flows ...	211
Figure 8.3.2. 42 Insert R305 and Relationships via ECA Model Test Client.....	212
Figure 8.3.2. 43 Update R503 and Relationships via ECA Model Test Client	213
Figure 8.3.2. 44 Modification of R503 causing Parallel-AND Split Process Flows	214
Figure 8.3.2. 45 Delete R502 and Relationships via ECA Model Test Client	215
Figure 8.3.2. 46 Deletion of R502 causing removal of P502 and connected paths	215

List of Acronyms

AAL-DL	Advanced Adaptation Logic Description Language
AI	Artificial Intelligence
BPEL	Business Process Execution Language, also known as WS-BPEL or BPEL4WS
BPEL4WS	Business Process Execution Language for Web Services
BPM	Business Process Modelling
BPMN	Business Process Modelling Notation
BRM	Business Rules Management
BRMS	Business Rules Management Systems
CLIPS	C Language Integrated Production System
DC	Data Centre
DFD	Data Flow Diagrams
DRL	Drools Rule Language
EBNF	Extended Backus–Naur Form
ECA	Event Condition Action - Business Rules components
FR	Flow Rule
IR	Initiation Rules
JESS	Java Expert System Shell
MOF	Meta Object Facility
OCL	Object Management Group's Object Constraint Language
OWL	Web Ontology Language
POJO	Plain Old Java Object
PRRP	Production Rules Representation
SBVR	Semantics of Business Rules
TR	Termination Rules
UML	Unified Modeling Language
URML	UML-Based Rule Modelling Language
VIDRE	Vienna Distributed Rules Engine
WS-BPEL	Web Services Business Process Execution Language

1. Introduction

It is widely recognised that a business management system with configurable workflow holds the potential to standardize the processes and activities associated with each transaction. This creates several benefits including not only a central repository for the way companies do business but also boosts teams efficiency with structured processes, provides process automation which means that errors are greatly reduced, less time is needed for training, there are fewer repetitive tasks, custom developments are unnecessary and many more. Indeed, workflow and business process management systems are widely seen as the top priority for companies wanting to survive the current competitive markets [35]. Despite the many benefits, the complexities of configuring workflows cause major roadblocks for companies moving towards workflow solutions. The need for having configurable workflows in mobile and dynamic environments have been discussed and well documented by various authors including Harrison [48], Raza [93] and Ben et al. [71]. This research is primarily focused on adaptation of business rules to manage and support workflows including the configuration task. The Chapter is divided into five sections including research motivation, research hypotheses, aim and objectives, approach taken to reach the goal of the research (research methodology) and the outline of the thesis.

1.1 Motivation

Businesses rely on processes to function. These are accomplished by using business rules (policies), which are implemented as components of workflows in computing applications. Business rules can be applied to software applications or systems; almost all workflows are based on some sort of rule-based systems. As described in [102], business rules control the behaviour of business processes and enforce best practices in the workflow domain. For example, in a data centre workflow application, a business rule may exist to ensure that before the equipment installation process is executed, rack space utilization is less than rack space capacity. Therefore, it makes sense to use business rules to manage workflow processes. The biggest strength behind the use of business rules comes from having multiple and changing business rules that interact with each other and with processes. In workflows, an essential element for success is the degree to which the business rules can

quickly change and propagate their changes in real time. However, when more business rules are added, modified, and inter-relations are established, business rules and workflow require extensive work to maintain their consistency. To date, it is still very difficult to configure and automate workflows [27]. The workflow modification process requires expert knowledge and it is time consuming. This is probably one of the reasons why business process management and workflow systems get a bad rap as an ineffective process management solution. Workflow systems, vendors and experts typically offer generic workflow solutions. Workflows are typically defined on an abstract level and customised by a specific system to fit processes in the local environments [119]. With custom fixed workflows, the process status and rules are defined in advance. If rules change in later days, the workflow becomes unusable and requires major work to change them. Dynamic conceptual models should be incorporated to support the changed workflow processes and rules. The recent state of sales report by Salesforce [106] shows that sales reps spend 66% of their working time weekly on non-selling duties. However, with automation and configurable workflows, certain business rules can be enforced to allow them to concentrate on their core responsibilities. The whole idea of having workflow systems is to help implementing repeated tasks consistently, easily, and effectively, problems being documented and prevented in future by improving the processes. Even though 54% of the processes are documented, they are unmanaged (Figure 1.1). Based on the observation below, we can envisage the cause being the complexity of configuring workflows.

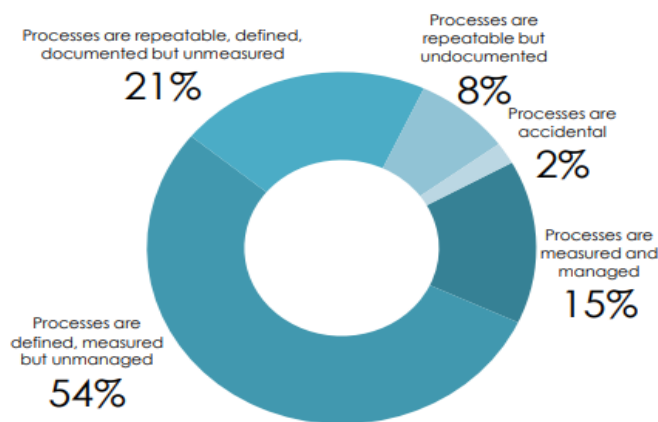


Figure 1. 1 Actual process maturity of the organisations

Source PROCESOWCY.PL's [89].

In the survey conducted by PROCESOWCY.PL [89], it turned out that workflows are implemented but ceased to work after some time. Companies are encountering difficulties in configuration of the workflows to adapt to changes.

According to report by Mulholland in [72], a well-known company “Triaster Limited” discovered that by changing just one process that was often run, their client could save over £300,000 per year. The process in question originally cost £396 for every run and was cut down to just £173 by making some changes. The process would run 247 times per year in a single business unit, saving that unit £43,000. Across the whole company, the process was run roughly 1,812 times per year, making the improvements save a massive £313,476. Enabling workflows with ability to support changes can result in faster response times, thus improving workflow change and configuration experience.

Despite many benefits that workflows bring, one of three common challenges that disrupt clinical workflows is the configuration problem which usually is caused by a disconnect between workflow design and business processes. This arise due to new processes being introduced or requirements changes. In articles [54,125], highlighted concerns regarding risks introduced by workflow disruptions. Beside productivity losses, there is a huge risk to patients in form of inadequate care. Workflows problems may lead to medical errors, the third leading cause of death in United States.

Problems mentioned above have inspired the development of a novel approach to automate workflows using a framework of business rules and business rules relationships. The model foundation is on the Event-Condition-Action components, which is based on a formal business rules ontology. The prototype is built on an object-oriented technology using Java by applying business rule change propagation, business rule adaptation and indexing algorithms. The outcome of this research is an enhanced and efficient mechanism to allow workflows to be easily modified.

1.2 Research Hypotheses

The key research questions, which present the scientific significance of this investigation is (1) how to develop an innovative framework based on dynamic business rules to support and control workflow processes (adaptation of business rules in workflows) and (2) to investigate and learn about the business rules structure and behaviour patterns for the creation of business change propagation mechanisms to support change propagation between related business rules. This study will therefore address the following research questions:

- Business rules can be considered to control business processes to improve and allow higher adaptability during design and execution of a workflow. What factors limit the adaptation of the business rules in workflows?
- How to develop an ontology of the business workflows which makes it possible to formalize the business rules using templates so that dependencies between the rules can be described.
- Changes often command other related changes, so the question here is how can we specify the dependencies between the rules on the base of the ontology model so that the rules can be adapted to the changing conditions in real-time, making it possible to propagate the necessary changes? To be more precise, is it possible to create an efficient algorithm for change propagation, which enables the run-time adaptation of the business workflows?
- How can we optimise business rules to improve execution performance and provide runtime modification? How efficiently can the underlying business rules be retrieved?
- How we can use the business rule dependencies to construct an efficient mechanism for adapting the rules in the case of changes? How can we enable adaptation of the business rules in real-time with reasonable complexity?

- Can the proposed model structure be able to generalise to new business rules in a workflow not seen during prototype validation?

1.3 Aim and Objectives

The aim of this research is to construct an efficient mechanism for adapting the business rules to the changing conditions of the business workflows. This is achieved by first investigating the problem of managing dynamic business rules in workflow systems. Second by providing a flexible and adaptable real-time solution to deal with the complexity of managing changes and propagation. Henceforth improve the efforts required to identify, modify and maintain business rules in workflows. The desired outcome is to provide dynamic business rules that can control workflows in real time.

The main objectives are:

1. Study existing research works through literature review in the area of business rules and workflows. This objective is further divided into the following sub objectives:
 - a. Information gathering by identifying relevant published research papers, journals, articles, posters, etc.
 - b. Reviewing existing approaches and methods for accessing and modifying business rules reported in the research papers
 - c. Studying possible approaches and methods of formalizing business rules
 - d. Providing critical analysis and evaluation of the researched papers to establish real gaps and limitations to the existing business rules problem.
2. Study business rules and workflow systems and products in the market today. This objective is also divided into the following sub objectives:
 - a. Identify and get familiar with relevant workflow business rules systems and products to understand the trends of what has been done in today's market.
 - b. Review existing approaches and methods for modifying business rules provided by these systems and products
 - c. Provide critical analysis of the systems and products to establish the real gaps and limitations to the existing business rules problem.

3. Using a suitable methodology to establish and design concepts necessary to support the management and administration of business rules in workflows
 - a. Define business rule structure
 - b. Define business rules concepts
 - c. Define business process concepts to be supported by business rule concepts
4. Develop a formal model to define business rules concepts and relationships.
 - a. Define a methodology of a proposed business rules model
 - b. Define the framework of proposed business rules model for formal business rules concepts / definitions
 - c. Define business rules classifications
 - d. Define business rules relationships formal definitions and dependency graphs
5. Validate the proposed model by using a prototype to demonstrate the following capabilities:
 - a. Provide runtime support for dynamic creation, modification and deletion of business rule and event, condition, action components, expressing a higher level of business rules abstraction, usability and adaptability in real-time. It is intended to address the lack of support for managing dynamic business rules and components (events, conditions, actions) in the existing frameworks. As mentioned in [94], a typical business rule is a script buried in a program code and it is never easy to modify parts or components (event, condition action) of the rule. However, the dynamic nature of business rule and the likelihood of competing change requirements means the rules and components will need to be modified but changes are not known in advance, which makes it difficult to specify adequate changing components or parts priori. Hence, business rule and components (event, condition and action) cannot be static and must constantly evolve to remain relevant.

- b. Provide support for managing business rules and components relationships in real time. It is intended to address the lack of support for managing related and conflicting business rules at components (event, condition, action) level. In the current business rule management frameworks, relationships are not based on business rules components (event, condition, action), instead relationships are formed at the rule level. There is no way of determining relationships at business rule components level. This means a change of one component would require the whole business rule to be modified.
- c. Provide support for managing change propagation between business rules and components.
- d. Provide support for managing business rules adaptation to control and govern a workflow, hence provide support for managing process flows within a workflow. It is intended to implement the adaptation of business rules to control business processes in a workflow. The objective is to provide a mechanism to help to solve the common workflow configuration problem. Business processes are rigid and difficult to maintain [95]. Rigidity is characterised by not accepting changes at runtime without programming and recompiling of workflow during reconfiguration process. The strategy of our solution is to describe processes using business rules that are afterwards translated into graphs to manage the objects and dependencies providing flexibility during runtime modification. Business rules' flow patterns in a graph provide a means of connecting process activities together. The business rules' flow patterns play a big role in determining how the process will be executed in a workflow, henceforth the key objectives include the ability to enable the start of a process, disabling of the end process and execution of intermediate processes.

1.4 Research Methodology

The methodology of this research combines several methods, which are needed to address the objectives specified in section 1.3 in an adequate way. Due to the nature of the proposed solution, the methodology combines analytical, constructive, and experimental methods. The analytical method will be applied during the early stage when there is a need to critically analyse business rules to build a rich set to support the validation step. The constructive method will be used for modelling, formulating building blocks of the framework, and constructing the software prototype. The experimental method will be used to test and validate the model based on the use of cases and scenarios identified. Figure 1.4 presents the methods performed.

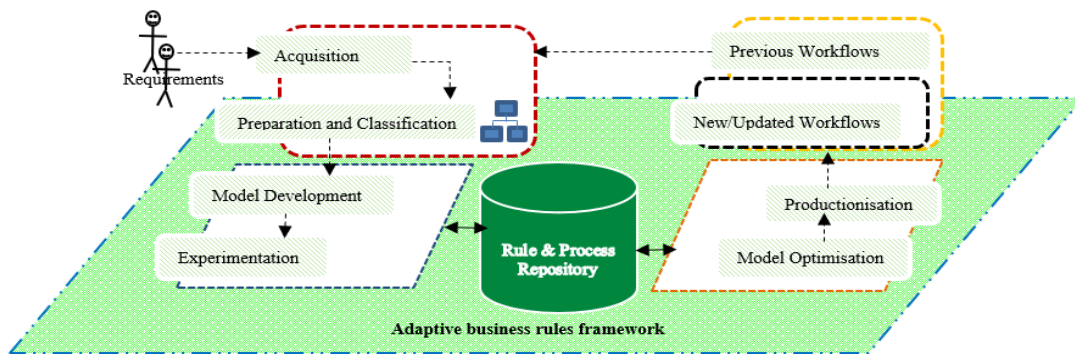


Figure 1. 4 Research Methods

1.4.1 Analytical Method

The analytical method details important analysis activities necessary to provide a rich set of business rules to be used for validation step. These activities include:

1.4.1.1 Business Rules Acquisition

The purpose of having a special business rule acquisition activity is to make the acquisition process more systematic as to ensure all the business rules are acquired. The business rules are captured from users to determine the user requirements. Also captured within previous business rules and process management systems are stored in the repository and are

available for further and detailed analysis. Note, the proposed ECA Model itself presents an important source for business rules.

1.4.1.2 Business Rules Preparation and Classification

This activity ensures business rules are atomic, each rule belongs to one category, and is formally described using a business rule language. The business rule classification simplifies the formalisation and ensures higher clarity and consistency of the business rules and components. For each business rule, an appropriate rule template is defined to capture rule components (event, condition and action). A business rule template represents a pattern that tells what part of business rule description belongs to event, condition and action. Three categories of business rules are collected: initiation rules, execution rules (process rules and flow rules) and termination rules. These will allow:

- A business rule to trigger a process or activity.
- A business rule to restrict the execution of a process or activity.
- A business rule to execute processes in workflows.

1.4.2 Constructive Method

This provides a constructive platform to achieve our mentioned objectives. To build a model to cater for dynamic business rules adaptation in workflows. The model will continually learn and adapt to new business rules as they emerge and change. This includes the support for advanced algorithms such as business rule indexing, change propagation and adaptation of rules in workflow. Followed by building a prototype and running a series of experiments on use cases to provide results. The activities of this method include:

1.4.2.1 Business Rules Model Development

This method is concerned with building an ontology of objects used to construct the workflows and business rules, which govern them. This means adopting an approach which relies on an object-oriented modelling paradigm to make it possible to define objects, classes and relationships between objects in a bottom-up manner. This is suitable for

representing business rules complexities in a more structured and controllable manner. Also advocating the use of AND-OR graphs as a solution for managing changing behaviour of the workflow. The purpose is to provide graphical representations that help to understand business rules relationships. The graphical representations of business rules will be captured in the rule repository. The objective in this activity is to identify business rules that are complex and to provide them with graphical representations, which will make them simpler for execution of business rules and their relations in workflows.

1.4.2.2 Business Rules Model Optimisation

This is a valuable activity providing a means for measuring performance. It is concerned with investigation of which algorithms best fit the needs to build a dynamic and adaptable business rules in workflow and making use of mixed algorithms including rule change propagation algorithm, adaptation of rule in workflow algorithm, and use of the graph patterns and indexing mechanism to provide quick retrieval of business rules and components. It will also provide a runtime modification capability through use of Metarules concept (Chapter 5).

1.4.2.3 Productionisation

This activity is responsible for creating the adaptive business rules framework for workflow management to meet real-world working conditions. The business rules to be entered through a graphical interface, which translated directly in DRL Drools format. An incremental algorithm associated with this interface builds the corresponding indexing graphs, which represent the dependencies between the rules internally for further use. Seamless, business rules and components are translated and mapped to control process in the workflow. And finally, the workflow is executed using the built-in engine of Drools.

1.4.3 Experimental Method

This method is concerned with validation of both the proposed model and the identified algorithms through the prototype based on use cases defined. The experimental approach is covered in more detail in Chapter 8.

1.5 Thesis Structure

To report the findings of the research in detail, the remainder of the thesis is organized as follows:

- Chapter 2 presents literature review; various research papers have been studied and analysed in the context of the business rules in the workflow domain. The nature of the existing researches, state of art tools, products in the market, methodologies and proposed solutions to problems are studied, and gaps and limitations found are summarised.
- Chapter 3 provides some definitions of terms used in this research including Business Rule, BRMS, Workflow, Flow Patterns, etc. It presents the basic structure and concepts of a business rule then provides insights into business categorisation based on an ontological approach.
- Chapter 4 forms the core of the thesis providing details about the conceptual model and framework, describing the ECA model formalisation and business rule components dependencies using the AND-OR Graphs.
- Based on the conceptual framework, Chapter 5 discusses the ECA Model systems architecture of the framework, followed by Metarule concepts to support runtime modification of business rules. Also, the business rule indexing approach to provide better performances on adding, updating, deleting and executing of business rules is discussed.
- Chapter 6 adopts comprehensive approaches to provide change propagation and rule adaptation in workflows based on the well-defined model described in Chapter 4.
- Chapter 7 presents the implementation of prototype, verifying and evaluating the developed transformation method. The prototype handles the creation, modification

and deletion of business rules events, conditions and actions components. Changes are supported with the implementation of a dynamic object-oriented technique that accounts for abstraction through definitions of business rules classes (events, conditions and actions) and rule fact classes as POJO. The classes are then mapped to Drools template to be translated into Drools DRL for execution by the Drool runtime rule engine.

- Using the prototype developed, Chapter 8 discusses the validation process of the ECA Model framework.
- Chapter 9 provides conclusion of the thesis, describing what has been achieved, recaps on the research contribution and recommendations for the future work.

2. Literature and Applications Review

This chapter presents a survey of research studies in areas of business rules and workflows. It introduces the background knowledge that is vital for understanding what has been achieved and carried out so far. Most research papers that we came across do not deal directly with business rule component structures and adaptation but rather deal with rules in general (at a higher-level) and the actual data associated with business rules. As there are fewer academic papers dealing directly with the issues being investigated, it is desirable to also explore the state of art tools (applications, systems and products) that are relevant to this research. The review was conducted by firstly collecting and studying the literature materials and existing applications based on research objectives and questions. Keywords such as Business Process, BPMN, Workflows, Business Rules, Dependencies, Rules Adaptation, Rules Propagation, Dependency Tree were used to search from conference papers, journals, articles and products pertinent to the topic of this thesis. The investigation was carried out to discover how these articles have addressed our research questions. This followed by evaluating their results to highlight how these have contributed to the work carried out in this research. The sections in this chapter are broken down into review of existing research studies, state of art applications and a summary of gaps and limitations that require further investigation and studies.

2.1 Existing Research Studies

The trend in research studies of business rule-governed workflows is focused primarily on theories and practices of custom-tailored workflows and much less on exploring business rules dependencies and the necessity of adapting the business rules to the changing conditions. This section reviews research studies that are directly focusing on business rules in workflows, change propagation and business rule adaptation mechanisms (algorithms) in workflows. In addition, various indexing approaches are also reviewed to support our indexing implementation. The indexing approach enhances the creation and execution process of business rules and components.

2.1.1 Business Rules in Workflows

A survey was conducted by the authors in [40, 42] to look at business rules methodologies to support workflow systems and applications automation. Graml et al. [42] stated that “A problem of today's standard business process automation systems is that they are too rigid to cope with changing business demands, especially for long running business processes. A solution to overcome this problem is to combine business process with business rules”. In that research, the assumption is made that using business rules, business processes can be made agile at run-time. The derivation rules are defined for decisions in the process models, constraints are applied by those decisions and process rules are created for logical dependencies of activities. Their proposed solution focused only on the modelling standpoint for integration of rules into business processes using a standard language such as BPEL. While there is a support for workflow activities dependencies, there is no support for business rules dependencies and change propagation. In Chapter 4, we will show how business rules dependencies can be defined using the AND-OR graphs. By representing business rules in the AND-OR graphs, we will show how business rules can be formalized and make them more expressive; hence makes it easy to incorporate in business processes. Moreover, Chapter 6 will show how various change propagation dependency patterns will be defined in this research to provide a systematic runtime modification of related rules and processes. Also using Metarules (Chapter 5), business rules and components structure can be easily modified at runtime.

An article written by Rowe et al., in [102] discusses how business rules are significant in the design of workflows. The paper explains how different classes of workflow systems can apply the business rules to support their execution. The authors identify two approaches of using business rules in workflows. The first approach is to embed business rules engine with process or workflow engine (process centric). Another approach is to include business rules in an application (data centric). Although there have been tremendous developments to both approaches there remain many unanswered questions. For example, no explanation is given to show how business rules dependency and change propagation are achieved. A great deal of effort is required to manage large set of dynamic business rules with multiple relations.

Flexible approaches towards workflow systems have also been discussed in other research papers such as [7,16,44,45,62,118]. For example, Casati et al. in [16] acknowledges that further techniques are needed to design workflows capable of adapting to changes. [16] presented an approach for a flexible workflow design using rules and patterns. It discusses a rule-based approach to handle exceptions based on a separate description of workflow activities. The approach provides a higher degree of flexibility during the design task since it makes it possible to model exceptional situations. The focus of their work is on specific rules that deal with exceptions during workflows execution. Still, it remains difficult to describe and account for the dependencies between the rules. It becomes even more complicated to deal with multiple changing rules as the rule management remains a tedious manual task. In fact, this is one of the main reasons why rule-based approaches have not been a popular choice for managing workflows. In this thesis, the algorithms for semantic indexing of business rules and change propagations, which account for business rules dependencies using the AND-OR dependency graph will be introduced. The structuring of the business rules into AND-OR graphs will provide a greater support for executing dynamic business rules and propagating changes.

Goh et al. in [41] supports the use of Event Condition Action rules (ECA) in workflows product development. In their approach, workflow activities are associated with ECA rules to govern how activities are executed. They recognised that a set of related business rules have a potential to be invoked and applied to the wide organisation applications. However, there is no mentioning of how the rule dependency is implemented for the set of related business rules. The emphasis of their work is rather on high-level integration platforms for building flexible workflows, rather than business rules, process structures and their dependencies. Furthermore, the authors discuss the adaptation to workflow change by changing and inserting new rules. However, they do not explain how the different workflow patterns are realized. This thesis (section 7.4) considers sequential and parallel flow patterns of workflows.

There have been attempts to model business rules as components in themselves, separate from the business objects and the application-logic [3, 11]. While the business user is free to define and modify the rules, there is no formal definitions of business rule components in the same ontology. Furthermore, there is no definitions for rule classifications and the business rule components dependencies. Hence, hinders the adaptability of business rules. In this thesis the conceptual model and framework topic to describe business rules and components dependencies formalisation is presented in Chapter 4.

Another interesting approach in [36] focuses on the implementation of an event-driven engine for distributed workflows. To control the distributed workflows, the authors in [36] maintain an explicit list of events. Their approach addresses the problem of distributed events in workflow execution by focusing on reactive event-based coordination and integration but because the inter-relations are not defined explicitly, they still do not offer much flexibility in controlling the business rule dependencies and change propagation. In their later publication [116], they focus on formal aspects of event-driven workflow execution using brokers to determine the proper semantics of all the involved components of workflows. This provides the description of formalised semantics of higher-level constructs with regards to event histories. However, the condition and action components of business rules have not been considered. It is insufficient to assume that only events change. Business rules changes may result from the condition and action parts as well. The discussion on business rules components (event, condition, action) formalisation is covered in Chapter 4 of this research.

The Vienna Distributed Rules Engine (VIDRE [101]) approach provides a definition of distributed business rules to enable business processes to be accessed via business rules through exposing them as web services. This approach brings together the rule-based techniques with the advantages of service-oriented computing to provide access to business rules as services. A workflow process or activity is implemented as a distributed business rule. This approach is particularly powerful if several business rules are involved. Nevertheless, the focus of this work is at a business rule level not business rule component (event, condition, action) level. Furthermore, the paper does not discuss how changes to

distributed business rules are managed and propagated. The implementation of business rule change propagation is vital to provide consistency and adaptable business rule model as discussed in Chapter 6 of this report.

Other articles such as [53, 60, 76, 79, 96, 112] present techniques for using object models to organise and structure business rules around objects. Object modelling techniques help to define and present the business rules, which can then be easily mapped into workflow processes. A business rule will typically be described with some properties, for example unique identifier. The unique identifier allows a quick access to business rules in a workflow. In [79], the Object Management Group's Object Constraint Language (OCL) and Meta Object Facility (MOF) standards have been incorporated to include constraints (rules) on objects and associations. OCL is a formal language used in well-known UML models [10]. According to [96], OCL allows adaptation of the process model by using constraints (rules) and prior post conditions. During the assessment of an OCL expression, two assumptions are made: (1) States of objects in the model cannot change during the execution. (2) OCL expression must remain true for all instances of that type (collection) for which the expression is created. The above points form a sort of restriction, as they do not allow dynamic creation and modification of objects based on the business rules. In [60, 112], a modelling tool that supports UML-Based Rule Modelling Language (URML) is presented. URML is an extension to UML standard [10], which supports rules in UML class diagrams. [112] proposed a UML graphical notation for managing rules based on an Object-Oriented methodology; [112] introduced the technique of governing object diagrams (UML classes) to describe constraints and dynamic business rules behaviour. The focus of the above studies is on modelling aspects of business rules. Main issues regarding business rule dependencies, change propagation and rule adaptation in workflow are not discussed.

Another notable effort with respect to the concept of business rules and business processes integration, can be seen in [55]. However, the implementation approach is different from ours. Using the custom or user defined business rules, the role patterns are implemented in Prolog programming language. The approach is flexible because the user-defined rules can

also be added to present additional requirements. While there is an association of role patterns and business processes to provide further support for managing process flows, still there is a need for tight amalgamation between business rules and processes.

2.1.2 Business Rules Formal Models in Workflows

The foundation of vast majority of existing business rules models come from the area of Artificial Intelligence (AI) and logical programming [80]. When referring to the business rule model, the primary concern is formalization of business rule components. Business rule model is a description of a rule at the type level; the actual rule becomes an instantiation of it. There are two ways business rules can be applied to a workflow [49]. One is to provide an embeddable model such as a separate rule engine where any application can use or link to the model. An alternative is to include rules into the workflow model, which means only specific workflow applications can use it. The later approach is a process-focused model used by Business Process Modelling (BPM) systems as discussed in [65, 75, 77].

Formalization of business rules is today's hot topic of many explorations because it makes possible to manage unpredictable business rules behaviour. Interesting research studies concerning business rules formalization and models can be seen in [20, 21, 57, 98]. However, to the best of our knowledge none of the existing formalizations provide a completely formal business rule structure model to simplify the execution of processes in workflows. Furthermore, a detailed acquirement of business rules at component levels such as at event property, condition property, action property, qualitative and quantitative measures of the rule will not be available as expressed in this research. The most prevalent business rules models in workflow are Business Process Execution Language (BPEL), also known as WS-BPEL or BPEL4WS. In [4, 51, 88, 126], the authors offer an approach to describe business processes, together with their business rules in both abstract and executable ways. Business Process Model and Notation (BPMN) provides a set of graphic elements for modelling generic business processes [81, 82]. However, in order to define business rules, this research proposes some patterns of representation to entirely map

business rules components (event, condition, action) with processes. The BPMN is an Object Management Group standard used by developers and business analysts to define and develop processes. According to the authors of [82], BPMN process has been defined to enable graphic editing of service-oriented business process models. BPMN depicts processes as interactions between agents or process roles represented graphically. A more complete BPEL and BPMN abstract syntax can be found in [84, 87]. Even though BPEL and BPMN definitions are more detailed, they only include elements relating to process and data manipulation. Details relating to business rule structures are absent or barely mentioned. It is important to communicate that BPMN especially is not to be a formal model for expressing business rules as it only deals with processes abstraction. The support of valuable features for both models (BPMN and BPEL) creates building blocks to various components of business rules. Moreover, one of the aims of this research work is to formulate business rules to control business processes so it is important to incorporate these well-established process and workflow models. In addition, our proposed model adopts an approach which relies on an object-oriented modelling paradigm [104]. The object orientation makes it possible to define objects, classes and relationships between objects in a bottom-up manner, suitable for representing business rules complexities in a more structured manner to model workflow components and behaviour. The business rules will be used not only to initiate and terminate business process but also to manage different flow patterns (Sequential, Parallel Merged, Parallel Split) as explained later in section 6.4.

Adaptive Object-Models have been created to address the need for change by mapping information as data rather than code [23, 129]. Object-Model defines the objects, their states, the events, and the conditions under which an object changes state [129]. Business rules could be specified in the adaptive object model to provide the support needed to handle the challenge of business rules modification.

2.1.3 Change Propagation and Adaptation Algorithms

A change of a business rule component can affect other related business rules (dependencies) and processes in a workflow. Therefore, providing a dynamic change

propagation mechanism is inevitable. This will ensure changes to business rules are made to all related business rule components in a consistent and correct manner. The purpose of this section is to survey some of the piecemeal business rules algorithms that have been proposed to address the specific challenges in business rule change propagation and adaptation.

Existing algorithms can conspicuously differ about propagation and adaptation mechanisms of business rules, but they all agree that rules design and structure are the key to rise to this challenge. However, as concluded by several authors such as [5, 22, 68], business rules are often hard-coded or designed in ad-hoc manner, making updating or reusing them a very difficult task. Moreover, business rules change propagation and adaptation become virtually impossible. Many studies, for example [5, 6, 56, 70, 111] have been more interested in modelling rules adaptation in order to ensure more flexibility and reusability. [56] defines rules as specifications using a metamodel, which is supported by visual notations. The algorithm significantly shortens the process of designing adaptation for user interface environments. Inspired by the AGG tool, the rule adaptation and transformation approach in [111] is specified using UsiXML language based on graph grammar. [5] presents a rule-based framework (Tukuchiy) that generates dynamic UIs while preserving usability criteria. According to [70], several techniques of HCI can be mapped to adaptation concepts to adjust them to different users and contexts. [6] provides a taxonomy of adaptation concepts describing adaptation nature and process according to the user's profile, its context of use, its tasks and the UI model. Advanced Adaptation Logic Description Language (AAL-DL) can be applied to UIs described in others MDE languages. The focus of many of these studies is primarily on the scope of applications and user interfaces, they fall short in facilitating or offering a run-time flexibility of handling change propagation and adaptation of business rules and components. Unlike these works, the business rules are mapped to user interface components, in our algorithm the business rules are mapped to workflow components. A lot more attention is required to enable both rule change propagation and adaptation in workflows, see discussion in Chapter 6.

In the latest paper published in 2020 by [110], the authors recognise the lack of simple and accurate rule adaptation algorithms to support different rule modifications. The authors in that paper present an approach to enhancement rules adaptation over a larger number of data. They employ a Bayesian multi-armed-bandit algorithm [18] to adapt rules based on the collected data over time. They suggest a summarization technique, which offers a set of high-level conceptual features for interpreting the data by finding the semantical relationships between them. In contrast, our approach proposes a formal model based on the components (event, condition, action) of business rules to govern workflows. The business rules dependencies are defined after structuring them into dependency trees, which are in the form of AND-OR graphs (Chapter 4) corresponding to the mutual coexistence of the rules. The dependency trees make it easier to understand the relationship between rules. Ideally, structuring of the business rules into dependency trees would allow implementing of an efficient indexing algorithm for searching the rules (Chapter 5). Different patterns of inclusion of the rules in the trees will provide additional information to control the flow of execution as the business processes progress. In addition, we can use the trees to analyse the process behaviour in real time.

2.1.4 Indexing Mechanisms

Another important contribution to this research is the indexing of business rules to improve search and run-time performance. Consequently, it is important to explore the existing indexing techniques. Indexing techniques for managing business rules have been explored by various researchers, include [8, 32, 107, 127, 128, 130] to name a few. The authors of [127] proposed G Index algorithm that uses frequent patterns as index features. Frequent patterns are known to reduce the index space as well as improve the filtering rate. Despite the benefits, G Index has some disadvantages. First, there is no support for graphs implementation where nodes represent rules and edges represent rules relationships. Second, construction of indexes requires an exhaustive listing of paths, which in turn causes high space and time overhead. Like in [127], our approach also considers ‘graph’ data structure for indexing business rules (Chapter 5). Unlike in [127], our graph nodes represent business rules and edges represent their relationships. Business rule components

with similar patterns are grouped together and indexes using the graph data structure. Moreover, the graph structure is created using two layers (logical and physical layers). To improve the search and execution performance, the logical layer consists of two important levels (root and dependency patterns). These provide multi-level indexing to allow the business rules to be quickly accessed. Another approach [8] described a metric-based indexing on attributed relational graphs for content image retrieval. Graphs are grouped in hierarchy according to their distances and indexed by M-trees. Queries are processed in a top-down manner by routing the query along the reference graphs of groups. Triangle inequality is used for pruning redundant nodes. To manage such a large set of business rules, they are often grouped along several dimensions as described in [122].

An overview of graph structures, graph indexing techniques and their associated querying techniques can be found in [107]. In an article by [128], the authors propose a structure-aware and attribute-aware index to process approximate graph matching in a property graph. Authors of [130] introduce a graph index (Lindex), which indexes subgraphs contained in database graphs to improve subgraph-querying. Unfortunately, no indexing mechanism is enabled to specifically support the business rule components structure and their dependencies described in this research. Henceforth, there is still a room for improvement as far as business rules change and adaption in workflow domain is concerned. Chapter 5 introduces our indexing approach.

2.2 Vendors Applications and Systems

There are several popular Business Rules Management Systems (BRMS) with business process management and workflow applications on the market today, but it is still very difficult to configure and automate real-life workflow applications as the study by [19] revealed. BRMS applications have been explored by various authors such as [11, 13, 17, 26, 29, 50, 61, 70, 123] and others. In a typical case, the BRMSs use a rule engine for business rules management and Business Process Management (BPM) for process management, providing APIs for modelling business rules and processes. For this study, we will take a closer look at the following leading applications or systems:

2.2.1 IBM BRMS and BPM

According to the articles in [17, 61, 109], the IBM BRMS has the most inclusive set of business rules capabilities in the market. IBM Business Process Manager (BPM) includes IBM's Operational Decision Manager (ODM) tool which incorporates tools such as Eclipse to give developers the ability to create and modify business rules. Explored by the authors in [11], the IBM BRMS WebSphere ILOG JRules, which is now part of ODM provides flexible tool for modelling business rules. Although IBM BRMS provides an integrated environment with rich and flexible tools for business rule modelling, there are notable limitations in relation to managing changes to business rules as explained below:

- There is no easy way of changing rules that affect more than one process or system.
- Multiple changes to business processes will need to be applied even for simplest business rule changes. This limits the business agility that business rules are designed to provide
- There is no separation of the different parts of the business rules components i.e. Event, Condition and Action. This means change made on the “condition” part of the rule will require invoking the whole rule. Externalizing different parts of the rule brings flexibility and increases performance as only the part that needs changing is exposed on the business rule application. Henceforth, different parts of the rule need to be stored in appropriate structures to facilitate their management, as it is with the existing structures for data in database systems.
- Rules are executed one by one in a procedural manner. This results in poorer performance when processed and creates additional work when rule sequences change or when the actual rule change (edited, modified or deleted).
- Inability to perform logical deduction, hence its inability to manage changes to multiple business rule hierarchies [47].

2.2.2 CLIPS

CLIPS is specifically designed to facilitate the development of software to model human knowledge or expertise [37]. CLIPS expert shell provides a platform where expert knowledge may be categorized as rules. To enhancement its rules management capability, CLIPS is enabled to perform the inference procedure whereby business rules are interpreted to produce various actions [74]. This mechanism takes advantage of the

embedded pre-existing business rules knowledge as “facts” to produce a recommended conclusion to a problem through its inference engine. Although CLIPS environment is interactive for editing business rules, there is no dedicated database. Hence, business rules are volatile and are removed from the memory as soon its execution is ended. An external database must be integrated with CLIPS to overcome this fundamental limitation. This adds to complexity and cost for managing rules. The problem becomes worse when changes to business rules are introduced.

2.2.3 JESS

As discussed by various authors such as [42,63], JESS is another rule engine originated from CLIPS and written entirely using Java. According to [64], there is an extension called Visual JESS, which enhance JESS. Furthermore, [114] proposed an approach to manage changes to business rule by using JESS language. Their approach is made up of two phases: first, the business rules are identified for the application is represented in terms of general syntax; then the rules are converted into Jess syntax, in order to provide flexibility when dynamic changes are made. Unfortunately, JESS also suffers similar limitations described above. Pitfalls of JESS for dynamic systems are well documented in [91].

2.2.4 ORACLE BRMS and BPM

According to a survey and analysis study of business process management done by [124], Oracle BRMS and BPM [85] is probably one of the best products in the market. Oracle BRMS and BPM [85] product offers many powerful features including rule and process management, author, web based graphical authoring environment that enables creation of business rules. With more of interest, Oracle product provides an embeddable business rules engine to its workflow [86] or process manager system [92, 102]. Oracle workflow application provides ability to add, remove and change the state of business objects (including rules) in the working memory. It permits the rules engine to reason and modify the original business rule information. Like IBM BRMS, Oracle BRMS solution is focused on the underlying data about the rules which is not the purpose of this research.

2.2.5 JBOSS DROOLS BRMS and jBPM

JBoss Drools BRMS [52] is a well-known and sophisticated open source BRMS and has a lot of functionalities, which allow users to write and validate business rules that can then

be pulled into Java Applications [13]. Drools also offer an open-source workflow engine (jBPM) written in Java to execute business processes described in BPMN. Unfortunately, Drools execute processes and rules using a programmatic approach, which makes it more complicated to understand for non-programmers. Certainly, this brings complexity in terms of usability and manageability. Furthermore, it suffers with the same problems of only handling the underlying data about rules, while our proposed approach is looking at the structure and components of business rules.

2.2.6 OpenRules

The authors in [29] refers to OpenRules as another powerful BRMS for rule-based application development. It provides a complex environment for editing business rules but supports the building of user interface to improve its usability. Furthermore, it allows the use of tools such as MS Excel, Google Docs, and Eclipse IDE to create a complex decision support system. It is easy to integrate with Java and the main advantage of OpenRules above the others is the way the rules are modified by using excel tables. Unfortunately, OpenRules also focus on the underlying data about rules not the structure and components of business rules to allow easy adaptation.

2.2.7 PRRP & SBVR

Proposals by authors in [28, 83] discussed the PRRP and SBVR on business rule management; they focus on defining business rules from the business perspective. However, these proposals do not address the aspect of providing logic implementation power on business rule structures and adaptability models.

2.3 *Summary*

Although BRMSs in most cases allow for business rules to be specified separately from the business processes, which support a two-step approach of business process modelling and business rules specification, it remains impossible to specify the dependencies between the rules based on the relationships between workflow objects. This causes multiple changes to be necessary to adjust already configured workflows and to update existing business rules even in the case of simple change. The main reason for this situation is the

lack of a consistent model of the components of the business rules themselves. Typically, rules are composed of events, conditions and actions, which are specified separately and are not related through the objects used to formulate them. This means that change made on the “condition” parts of a rule will require invoking the whole rule rather than only the condition component. Externalizing different parts of a rule (components) would bring flexibility and increase the performance as only that part which needs changing would be processed explicitly, while the adjustment can be automated.

Business rules without a knowledge base or vocabulary cannot convey information effectively since no clear definition is given to the business rules components. Most of the Business Rules Management (BRM) products offer some functionality to build a business vocabulary, but to the best of my knowledge there are no formal specifications to support adaptation of business rule structural components in workflow in an efficient manner. The requirement for having a formal business rules vocabulary has often been hinted at by various studies. [99] recognizes and explains the need to use a common facts and terms model. Although the Semantic of Business Vocabulary and Business Rules standard [83] is set up to specify business rules, its specification is very general, and its focus is towards a more static unified business rules standard. Additional approaches still need to be developed when considering the dynamic nature of business rules. A survey done by [120] also concluded that there are no clear definitions and scientific foundations to even well-known workflow management systems such as Business Process Management (BPM). Furthermore, when it comes to rule change propagation, there is no formal specification that will support rules that can span across multiple processes in workflow applications.

The existing SBRV vocabulary appears to ignore the possible relationships between different business rule components (Event, Condition, Action). The SBRV limitations justify the demands for better concepts formalization for business rules. Our research focuses on formalization of business rule components (objects) that are specifically found in workflow systems. By adopting a bottom-up approach, business rule objects and relationships can be determined. This will allow us to configure business processes in workflows adequately. At the same time, it will permit to represent various domain-specific

heuristics, which govern the progress of the workflow in real-time. The advantage of the bottom-up approach is that the final developed model is likely to have more appropriate language and terminologies because it would include concepts directly from business rules and workflow arena that are relevant [12]. This should increase the content legitimacy and improved responsiveness to change.

In this research we describe and provide the flexibility of defining business rules on objects, attributes and associations in the object model by enabling logic programming power (Prolog-kind) in terms of binding, unification, backtracking etc. over object models. Our work in this research enables the specification of business rules during modelling to qualify association with conditions and enables the creation of that association at run-time between the objects that satisfy the conditions at run-time. Previously, such business rules were not modelled and were buried deep inside the code. Model developers and model-maintainers would be oblivious to such rules and the object model may not actually reflect the true state of the run-time model. This research attempts to address some of these issues by introducing the model to formalised business rules component structures and dependencies. Furthermore, change propagation and business rule adaptation in workflows algorithms are implemented to tackle the propagation and adaptation problem discussed in this thesis.

3. Business Rules, Process and Workflows

This Chapter provides some definitions of various concepts used in this research including business rule, business process and workflow. These are important concepts and building blocks. They are defined to support the development of the proposed formal model. Using the notations from the Extended Backus–Naur Form (EBNF), the definitions of these concepts are further described in Chapter 4.

3.1 Definition of Business Rule and BRMS

“A business rule is a directive that is intended to influence or guide business behaviour. Such directives exist in support of business policy, which is formulated in response to risks, threats or opportunities” [14]. “A business rule is an atomic piece of reusable business logic, specified declaratively” [100]. A central principle of business rules officially advocated by [46] is that: “Business rules are made up of facts, and facts consist of concepts that can be conveyed and presented as terms. Terms are usually business concepts; facts present declarations about concepts; rules govern and the facts”.

For example, a business rule might state that only people between the ages of 16-70 may drive a car. Other examples of business rules include requiring a bank to prohibit a loan if a customer’s credit rating is low, requiring students to apply for a course if they met requirements, requiring correct username and password to be supplied when logging in to a work account, a shop requiring to give a discount when customers purchase over a certain amount. Business rules can be used to provide predictive analytics, i.e. if the past year sale is increased by 10% then next year sales will increase at the same rate. So, these definitions and examples describe a business rule as an instruction that constrains or expresses an activity on a fact (person, software, service, systems, etc), which will resolve to either true or false. It generally involves conditions and actions.

A BRMS (Business Rules Management System) is a software system used to define, deploy, execute, monitor and maintain business rules [39]. Examples of BRMS include Drools, Oracle BRMS, IBM Operational Decision Manager, SAS Business Rules Manager, etc.

3.2 Definition of Business Process, Workflow and BPMS

A process is an activity in implementation or execution, for example moving an equipment from one data centre to another. A Workflow is a sequence of activities (processes, tasks, steps) that implement a set of data. Workflows can be found across every kind of business and industry, for example a bank transaction to check user balance can be created as a workflow [67]. Workflows are concerned with the flow of activities and related data in business processes based on imposed business rules. Workflow management system is a software that allows users to setup and monitor a set sequence of activities in the form of a flow diagram. The flow diagrams (i.e. BPMN) usually help to capture the start to the end of activities. Business rules are used to define the structure and development of workflow management systems [132]. Business Process Management System (BPMS) focuses on defining and refining business processes to make an organisation operate more efficiently. Processes are documented to capture the current state of end-to-end of organisation processes. Like workflows, BPM systems are implemented across a variety of different sectors including healthcare, manufacturing, construction, finance, etc. Both defines the series of task to produce some outcome, however the workflow is more general term than business process. Some concepts used are very similar for example both provide support for process flow patterns (Sequence, Parallel Split, Parallel Merge, etc). However, BPM systems focus mainly on analysis business processes and not its interactions. A business process is mostly used to achieve business objectives. A business process system describes how and when process interact, but not what is exchanged or transformed. Normally, a workflow implements a single process in more details and flow patterns form a major building block. Hence, it is vital to examine various process flow patterns. This is to ensure that possible scenarios can be handled using the business rules model formalism. In the following section, the commonly workflow patterns are presented.

3.3 Workflow Patterns

In computing, a pattern is a reusable template or solution created to resolve repeated problems within a given context in a software and application design and development. A

workflow (process flow) pattern is a specific form of pattern defined to support dependencies between activities in workflows. In this research, the relevant process flow patterns presented in [121] are considered. These patterns capture the elementary execution facets of the workflow or process level (part of Two-level Architecture presented in Section 4.2). These execution patterns provide the way in which business rules are to be run to control processes in a workflow, mostly in serial or parallel fashion.

3.3.1 Sequence (Serial) Workflow Patterns

The Sequence pattern is defined as being an ordered series of processes, with one process starting after a previous process has completed. In the Sequence pattern typically, processes in the workflow flow from one to the other based on some business rules (Events, Conditions and Actions) that determine how the workflow flows from one process to the next, and process can wait for the preceding process to complete.

3.3.2 Parallel Workflow Patterns

The Parallel patterns are generally used when a workflow might have more than one path that is active at the same time. In the parallel pattern, the workflow splits at some point into separate paths (parallel split patterns), each of which may contain multiple processes. At some point, these paths may merge back together again (parallel merge patterns). Based on some business rules, the workflow can wait for all preceding paths to complete or continue as soon as the first path reaches the merge point. Business rules determine what should happen at the split and merge points in the workflow.

3.4 Business Rules Structure

The inclusive structure of a business rule in its simplest form is made up of the following logical statement “*When Event(s) If Condition(s) Then Action(s)*”. According to [78, 113], the constructs of the Event-Condition-Action (ECA) rules originated from the area of active database systems. The ECA rules state that when there is an event, condition is evaluated and if the condition is fulfilled then perform the action or actions [78].

In workflow applications, a business rule specifies that when an event occurs and if there is a condition set then specify a list of actions to be done. A common observation is that when there is an event which most likely is associated with a process to be implemented in the workflow, the "if" normally executes the flow part, where, at a specific point in time, a condition is to be checked. When a business rule is applied to a fact (role, process, data), it may cause the business rule to activate again (recursive) or activate other rules, hence causing a change of other rules (propagation). Setting rule recursive and propagation options will allow modification of the same rule once more or other rules for the current set of facts. Business rule change propagation is discussed further in Chapter 7. Business rule events provide statements that trigger or influence the behaviour of the rule; it may be to kick-start, update or close a business rule against a process in a workflow.

The Event-Condition-Action (ECA) business rule and its variations i.e. Event-Action (EA) and Condition-Action (CA) rules provide well-understood formal structures for modelling active business rules. In the ECA, the distinction between EA and CA business rules components provides a level of abstraction, thus increasing reusability [33]. The component structure of ECA rule conceives basic concepts covered in section 3.5. To specify business rules concepts, business rules are encoded using ECA, CA, EA, ECAA, etc., formats. The ECA and variations are modelled to provide a modularization of clear and well-defined business rule concepts within workflows. An important advantage of this approach is that the business rules components are extended to inform the workflows.

3.5 Business Rules Ontology

One of the aims of this research is to provide a formal structure for a business rule in workflows. Using Bottom-UP approach, business rule concepts are introduced as part of our Description Specification Language (DSL) to provide formal structure. Business rules can easily become very composite. For this reason, it is very advantageous for a business rule to be decomposed into smaller concepts to allow different parts of the rule to be discovered. DSL is adopted to describe and present different parts of business rules knowledge in the workflow domain. According to [59], DSL is known to be helpful for

declarative knowledge engineering. Specifically, it makes use of OWL to representing the meaning of business rules and components. [1] describes OWL as a description Logic based on ontology language for the Semantic Web. It is designed to present rich and complex knowledge about things/objects, groups of things/objects, and relations between things/objects and semantics. Hence, the OWL is well structured and suitable to define and represent meanings to business rules concepts in the workflow domain. Based on the foregoing discussion on OWL representation, one can describe the workflow domain in terms of business rules concepts, properties and relationships between business rules concepts. Table 3.5.1 briefly describes concepts depicted for derivation of the proposed model ratification. Table 3.5.2 lists the OWL entities that would be created and utilised in the implementation of the ECA model.

Concepts	Description
Process Flow	<p>Flow provides an important concept that allows us to manage and control the flow of both Information and Material, which links processes. The Flow is a superclass of Information Flow Class and Material class:</p> <p>➤ Information Flow depicts and expresses shared data between processes by which a Business Rule is applied or imposed. It is a primary construct for the proposed model. Information Flow may be connected directly or indirectly to Material Flow.</p> <p>➤ Material Flow represents physical resources or goods (input, observed and output) used and transformed by business processes. Material Flow Class will be used to express resource dependencies between processes. The consumed resources (input and observed)</p>
Fact: Data Material	<p>Processed information or value of a field on a record. A field can be a field on a database or form. The data or material has basic properties such as name as well as qualitative properties qualitative and quantities properties. The quantities refer to amount i.e. “80%” while qualitative refers measurement i.e. “higher” Usually material/information forms a link between processes in a workflow.</p>
Process	<p>As described in BPMN and BPEL models by authors such as [87, 126] a workflow consists of one or more processes. The processes represent well-defined business activities or functions designed to receive some input and produce some output. For example, a process to manage rack space availability is a well-defined business activity in a data centre. This process is designed to receive some input about the rack (rack name) and produce the</p>

	amount of space available or utilized for each rack in a typical data centre. The process will be represented as object and properties necessary to initiate creation and support for its execution. The information about processes may include various properties as well as associated objects such as Flow Objects.
Task Activity	Tasks are workflow steps, can be performed by an application program or a team of humans (role/user), or a combination of these.
Event	Events provide means for communication within and across Process and Rule levels of the workflows. The event class represents both synchronous and asynchronous events which may happen during process execution. Signals or notifications that an incident has occurred or is going to occur also cause events. The events always have temporal dimension that is absolutely or relatively to the beginning of the workflow execution. In workflow, the term “event” is usually generalized; used to express different kind of things. The start of a process, the end of a process, the change of the state of a process, information, or message arrival, etc., all could be considered events. However, we restrict the use of events to include only those types of events that will affect the flow of Process. We categorize Event concept into three main types, namely Start Event, Observed Event and End Event.
Condition	Condition is a logic statement that specifies what must be checked to enable evaluation of some facts. This evaluation is necessary to fire the rules. For instance, the condition is specified as “If Rack Space Utilization is greater than the 80% of Rack Capacity”. Condition may take a form of a check of a value, a database query, result of the execution of a function/procedure call. Conditional may contain multiple expressions joined by the logical connectives such as “AND” and “OR”.
Action	An Action describes what can be done to other objects with a possible outcome. For instance, in an action “send email”, “send” is an action and “email” is an object parameter. Each action may involve one or more parameters and in turn objects and object properties are created or transformed because of the actions. Furthermore, an execution of one action may cause one or more further actions to be executed in a kind of a chain reaction. The rules can prescribe many actions to be executed. Action Class specifies what needs to be implemented to complete the workflow process or to match the business rules which govern it.
Role System	Roles are responsible for implementing activities in a workflow. Relationship between the people and process are roles. Users can be a member of multiple roles.
Initiators User-agent Actor	These are the originators or creators of the workflow. They can also be the users that the ability to update workflow as well as add workflow users. They can be system/application users of the workflow

Time	Predefined period for action to be performed e.g. escalation times. Point based semantic (temporal logic), qualitative combined quantitative. Interval logic, relationship between interval duration of the processes. Event based (sequence of event)
------	--

Table 3.5 1 Ontology Concepts Description

OWL Concept	OWL Entities
Classes	ECA Rule (Parent class comprises of Event, Condition, Action) Event, Condition, Action, Process, Task, Flow, Fact/Information/Material, Role, etc.

Table 3.5 2 OWL Concepts

Ontologies help in defining possible data set of business rules entities or categories as well as representing the relationships between entities depicted in workflow and business rules domains. Henceforth, Figure 3.5.1 illustrate the ontology hierarchy graph of the business rules entities outlined in Tables 3.5. 2. The aim is to develop the content of terms depicted in workflow and business rules domains, ultimately illustrating how the ECA model can describe knowledge through a vocabulary of interwoven entities.

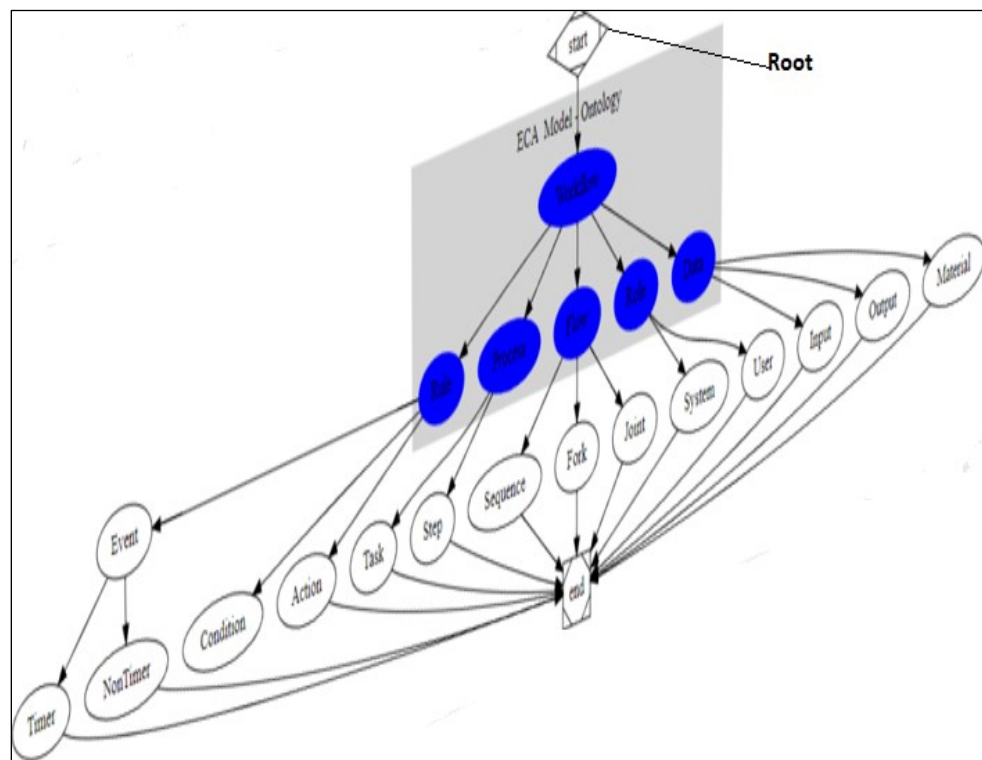


Figure 3.5. 1 Ontology graph using Protégée

Code Snippet 3.5.1 describes how the ontology hierarchy graph was created using Protégé Ontology Editor [73]. Protégé system provides both a repository as well as web services for defining both business rules and workflows ontologies. The basic concepts can be expanded by updating the script in the Code Snippet 3.5.1. The benefit is a standardized and configurable way to support the ontologies development maturation of the proposed model.

```

1 - digraph ECAGraph {}
2 - subgraph cluster_0 {
3 -   style=filled;
4 -   color=lightgrey;
5 -   node [style=filled,color=blue];
6 -   Workflow --> Rule;
7 -   Workflow --> Process;
8 -
9 -   Workflow --> Flow;
10 -   Workflow --> Role;
11 -   Workflow --> Data;
12 -   label = "ECA Model - Ontology";
13 - }
14 - start --> Workflow;
15 - Process --> Task;
16 - Process --> Step;
17 -
18 - Rule --> Event;
19 - Rule --> Condition;
20 - Rule --> Action;
21 -
22 - Role --> System;
23 - Role --> User;
24 -
25 - Flow --> Sequence;
26 - Flow --> Fork;
27 - Flow --> Joint;
28 -
29 - Data --> Input;
30 - Data --> Output;
31 - Data --> Material;
32 - Event --> Timer;
33 - Event --> NonTimer;
34 -
35 - Task --> end;
36 - Step --> end;
37 - Timer --> end;
38 - NonTimer --> end;
39 - Condition --> end;
40 - Action --> end;
41 - System --> end;
42 - User --> end;
43 - Sequence --> end;
44 - Fork --> end;
45 - Joint --> end;
46 - Input --> end;
47 - Output --> end;
48 - Material --> end;
49 -
50 - start [shape=Mdiamond];
51 - end [shape=Msquare];
52 - }

```

Code Snippet 3.5. 1 Framework development stages

3.6 Summary

This Chapter focused on establishing key concepts for the business rules and workflow adaptation approach. The Chapter starts by providing some definitions of terms and constructs used in this research including Business Rule, Process and Flow Patterns. The business rule concept provides the necessary information and structure to guide workflows, hence forms the underpinning concept of the proposed model. The structure of a business rule in its simplest form is defined as a logical statement “When Event(s) If Condition(s) Then Action(s)” (ECA). A significant advantage of this structure is that the ECA components are expressed to support business rules dependencies and adaptation in

workflows. Furthermore, the Chapter defines Business Rules Management Systems, Workflows and Process Management Systems. Along with ease of implementation, these systems also include comprehensive testing and deployment functionalities to allow execution of business rules and processes. The Chapter ends with a section on insights into how business rules entities are categorised based on a well-known ontological approach (OWL) and illustrated using the ontology hierarchy graph. The interconnectedness of entities is easier to perceive in the hierarchical graph.

4. Proposed Business Rules Model

A model generally represents how information is formalised. This Chapter presents a discussion on proposed model formalisation. The formalisation work presented in this section, initially appeared in our published journal paper [27]. The methodology is discussed first, to explain the steps needed for design and implementation of the proposed model.

4.1 Methodological foundation of the framework

A methodology is a comprehensive term used in software engineering to describe methods or procedures that are to be followed to resolve a problem or deliver a solution [66]. The methodology of the framework describes our approach for designing and implementing the proposed business rules model. Moreover, it provides stages necessary to support the development of the proposed model and its prototype. The prototype presents the realization of business rules change management and adaptation of rules in a workflow application. It is important to note that our methodology follows the design science research approach, which involves artefact analysis, design, development, testing and validation. The analysis phase surveys and determines existing problems in business rules management systems and provides design objectives for the proposed model. After the analysis phase, the proposed model design and prototype development phases will follow. The validation phase is included to test the proposed model through the prototype and observation of its implementation in Drools development environment. The key stages are identified in Figure 4.1 below.

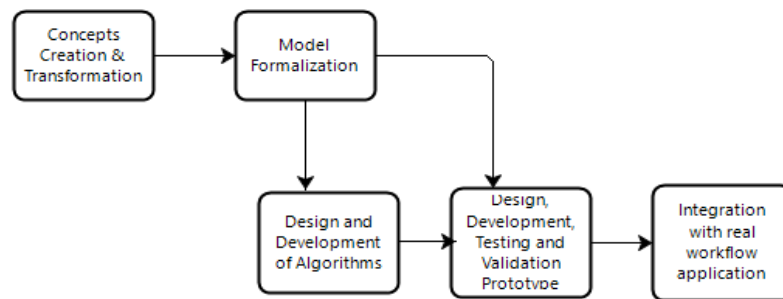


Figure 4. 1 Framework development stages

- Concepts Transformation

This task involves transformation of business rules and business processes identified in Chapter 3 into unified building blocks (concepts) that glue together and control the workflow. The building blocks (concepts) are expressed in terms of classes, objects, object properties and metarules.

- Model Formalization

The model formalization task consists of formal concepts definitions, formal concepts classification definitions, formal rules relationships definitions and meta-rules definitions. There after the ECA rule can be translated into the EBNF format for formal definitions. In addition, identify relationship between rule components then provide relationship formal definitions to a complete inscription of the model.

- Development of Algorithms

During this task, the algorithms will be developed for processing dependency trees (rule relationships) to handling of the business rule change propagation problem and rule adaptation in workflows.

- Development of Prototype

Like most methodologies, the step that involves development of a proof of concept is important. It permits converting ideas and theories into reality. Hence, this task is devoted for the implementation of a prototype using Drools, an open source development environment for business rules and workflows. The prototype development task extends the tasks performed to include model and prototype testing.

- Integration of the proposed model into workflow application (Drools)

The goal of the integration task is to provide the architecture for an integration of our business rules model with real time workflow systems. This is really a validation step of the proposed model on a workflow.

4.2 Two-Levels Architecture

The formal model presented in this research is based on the understanding of existing business workflows as event-driven and as a constantly evolving process of incremental development, execution and control. This model operates on two levels, namely Process Level (Figure 4.2.1) and Business Rule Control Level (Figure 4.2.2). The business rules are building blocks that control workflows and they are made up of event, condition and action components, or the famous “When <event> If <condition> Then <action>” structure, whereas the workflows are made up of business processes (directed structures), process steps (primitive procedures), process flows (material and information links between processes), roles, etc. For instance, if some events are observed during execution of a working process then the corresponding business rules which depend on these events are invoked and lead to actions which in turn perform the transition to a new step which may execute other processes or amend the parameters of the current process. The model uses structuring rules to glue together the processes from start to finish in a workflow (Figure 4.2.1).

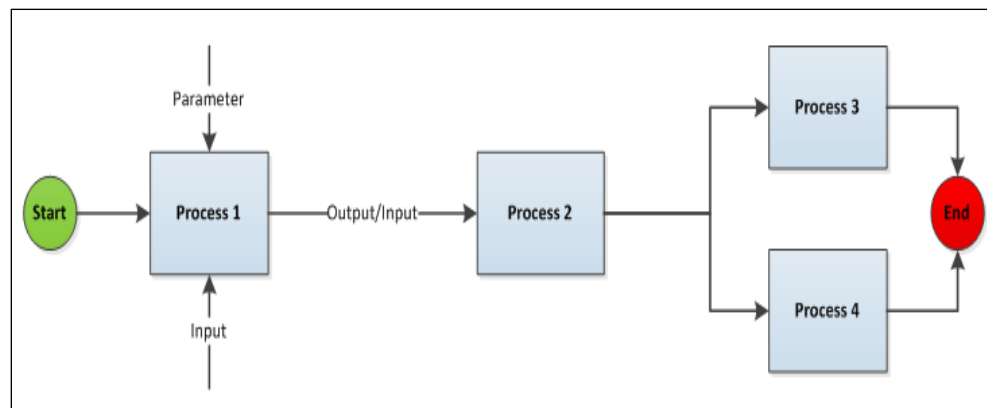


Figure 4.2. 1 Business Process Level

The business rules control level provides a level of abstract “independence” from the process level (Figure 4.2.2), suggesting that the rules can be changed without affecting the part of the current workflow which has already been completed. The business rule control

level automates complex business processes to perform business logic without writing a new code.

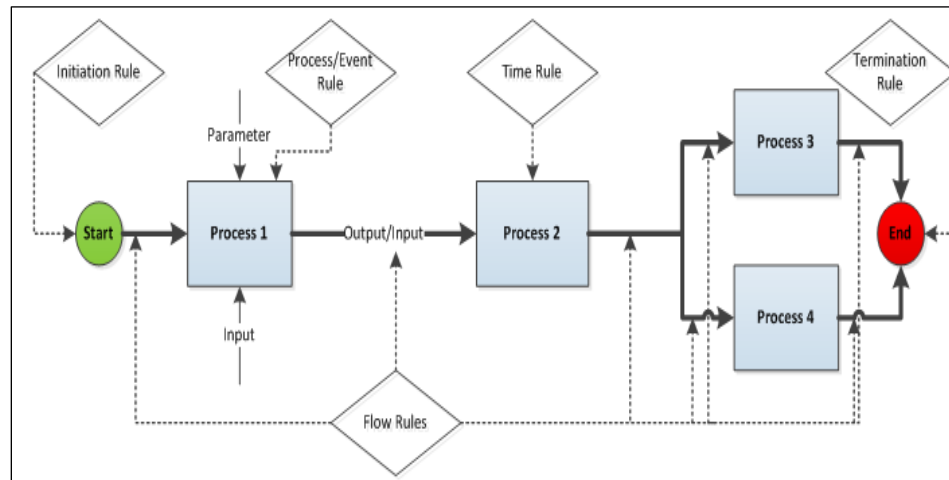


Figure 4.2. 2 Business Rules Control Level

The business rule control level supports various stages of execution of the workflows: Initiation, Execution and Termination. Based on the different role they play along the workflow progression; business rules can be organised in a kind of taxonomic hierarchy (Figure 4.2.3). In this taxonomy Execution rules are divided into Flow and Process rules, Flow rules are divided into Sequence, Fork and Join rules and Process rules are divided into Time-based and Non-Time-based rules. Additional rules known as Data rules (not covered in this paper) may be considered when some conditions are applicable directly to the input and output data to maintain the integrity of the flow.

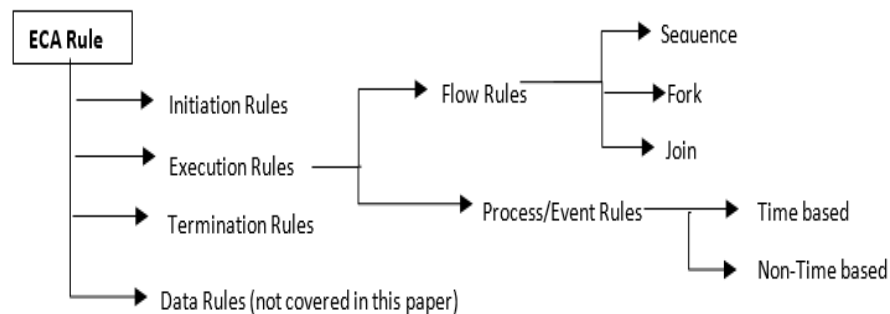


Figure 4.2. 3 Business Rules Classification

The Two-Levels Architecture (Figure 4.2.4) is essentially a representation of holistic, multi-dimensional views of the proposed model components and integration between process control and business rule control level components in a workflow.

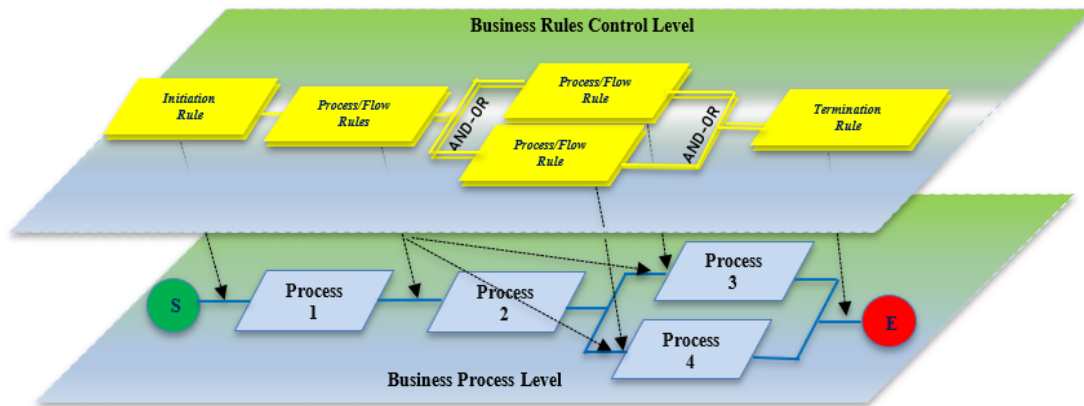


Figure 4.2. 4 Two-Levels Architecture

(Business Rules and Process Levels; S = Start and E = End)

Process Level considers:

- ✓ Business Objects (Processes, Flows, Events, Conditions, Actions)
- ✓ Object Properties (Identification properties, Qualitative description, Quantitative description)

Rule Level considers:

- ✓ Business Rules (Initiation, Event or Process, Flow and Termination Rules)
- ✓ Meta-Rules,
- ✓ Rules relationships and dependencies

The key difference of this architecture compared with other existing systems is the use of the business rule control level, which contains business rules, Metarules (Chapter 5) and business rules relationships to manage the execution of the processes in the process (workflow) level. In architecturally real environments, we will maintain many processes or flow rules based on workflow processes and business rules dependencies. Also, it is important to point out that due to our approach of account for business rule dependencies, one flow rule could execute multiple processes hence reducing number of business rules to be triggered by a business rule management system.

4.3 Business Rules Classification

Business rule classification (Figure 4.2.3) identifies types of business rules that are defined in workflows. This research will consider the following fundamental classifications of business rules in workflows.

Initiation Rules (IR)

The Initiation Rule (IR) formally depicts rules that specifically initiate a process. Depending on the conditions of the rule, the process can be launched and thus continue the workflow execution. Some Initiation Rules are driven by events only, hence known as the Start Event. As an example, Figure 4.3.1 presents the Equipment Installation workflow of an organization with three processes “Create Request”, “Send Message” and “Order New Rack”. In the background, the initiation rule “When receive a request, start message and then start” looks up and assigns the “Create Request” process whenever the rule is invoked. The rule is invoked when the request message is received.

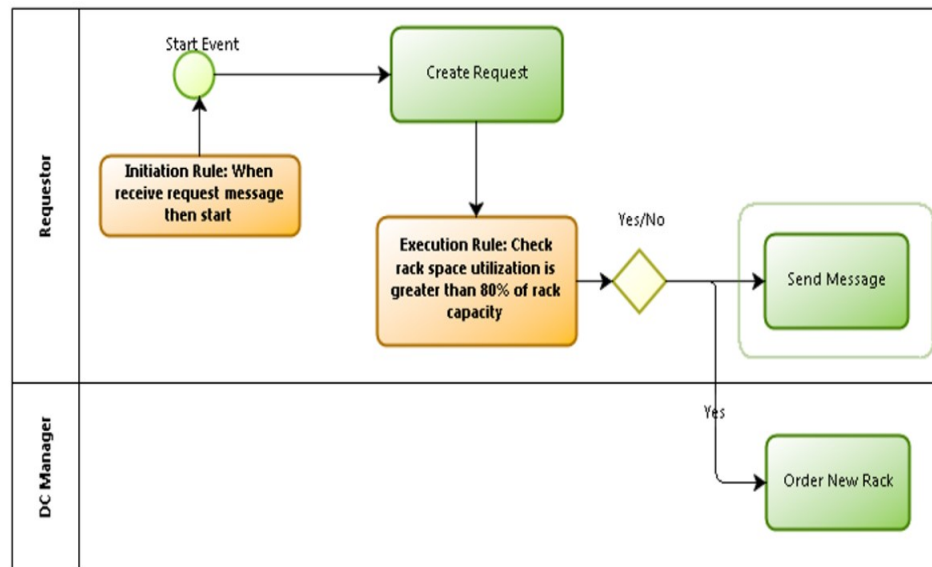


Figure 4.3. 1 Initiation Business Rule

“When receive request message then start process”

Event Rules:

The Event Rule class group rules are specifically defined on Processes during the execution of a workflow. An example of such an event rule is one which requires the drivers to stop when the road traffic light colour changes to red (Figure 4.3.2).

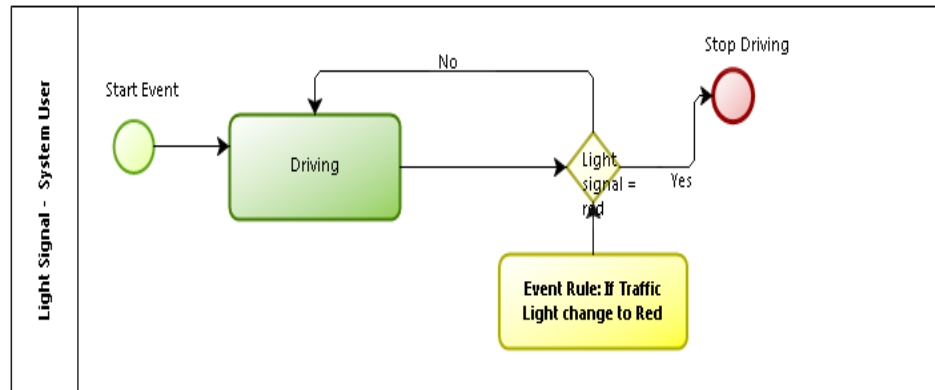


Figure 4.3. 2 Event Business Rule

“When light colour changes to red, stop driving”

Flow Rules:

The Flow Rule (FR) class formally depicts rules that specifically define the flow of workflow processes. All workflows depend on flow rules to progress from one process to another. In other words, flow rules determine the start process and the transition through a chain of processes until the workflow ends. Flow rules can move the workflow along a single chain of processes or split it into multiple pathways, thus forming an acyclic graph. For instance, a path can be established between “Create Request” and “Approve Request” processes to connect the two related processes in a workflow (Figure 4.3.3). Important flow patterns that will be covered in this research include sequential, parallel split and merge. From this perspective Flow Rules define the transition pattern and allow the ordering of the business processes in the workflow dynamically at runtime.

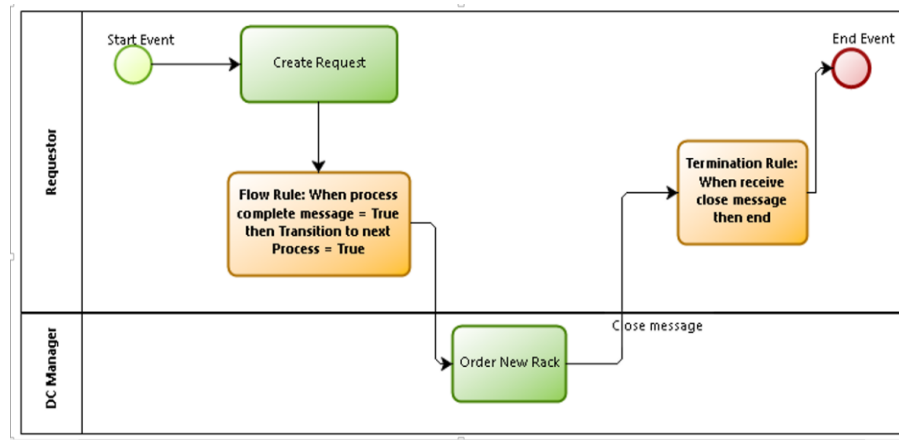


Figure 4.3. 3 Flow Business Rule

“When process completes, then move to the next process.”

Termination Rules (TR)

The Termination Rule (TR) class formally depicts rules that specifically trigger the end of a workflow. Some Termination Rules are driven by events only, hence known as the End Event. Figure 4.3.4 presents the Equipment Installation workflow of an organization with three processes “Create Request”, “Send Message” and “Order New Rack”. In the background, the TR “When receive closing message then end” looks up and ends processes whenever the rule is invoked. The rule is invoked when the request message is received.

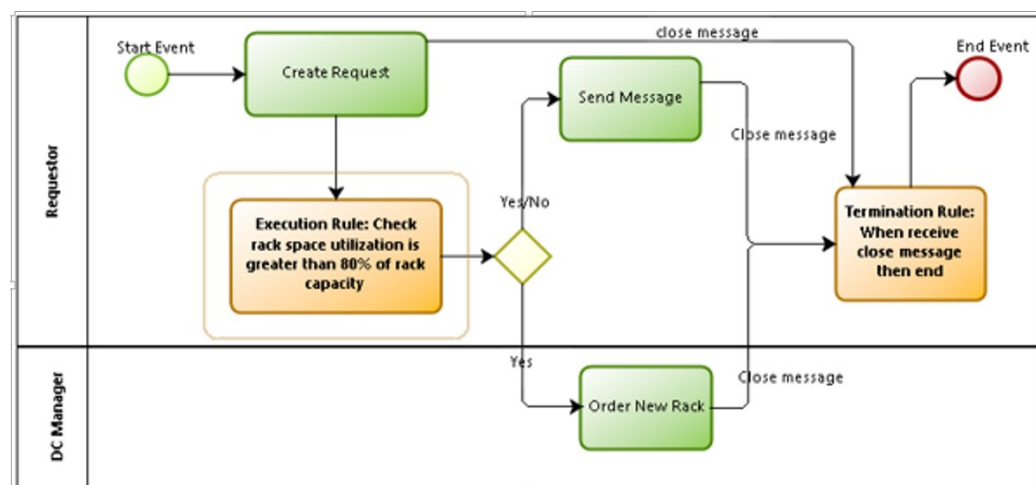


Figure 4.3. 4 Termination Business Rule

“When receive closing message then end.”

4.4 Business Process Ontology and Formal Specification

This section presents the basic ontology of objects used to construct the workflows and the rules which govern them. Using an example (Figure 4.4), the ontology is developed in a bottom-up manner. All examples have been illustrated using DFD diagrams.

Objects

The objects are the building blocks for describing business processes, rules and workflows.

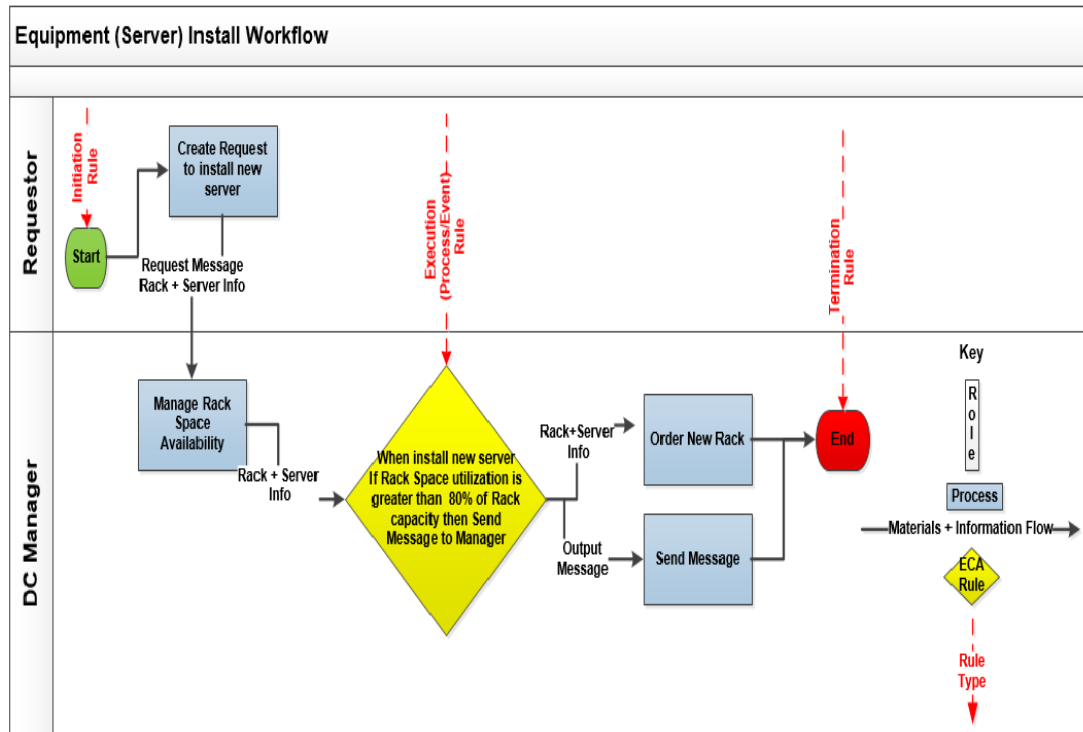


Figure 4. 4 Workflow and Associated Business Rules (Example)

If we consider the workflow in Figure 4.4, we can see that it defines a business rule: “When you install new equipment (Server), if Rack Space Utilization is greater than the 80% of Rack Capacity, then send message”.

Analysing the above example, the following concepts are identified:

4 Processes: (Create Request to install new Server, Manage Rack Space Availability, Send Message and Order New Rack).

2 Roles: Requestor, DC Manager which has not been covered in this paper.

Flow: Capturing data/material and information in and out the processes. Rack Capacity, Rack Utilization, New Equipment and even the Request are examples of Information and Material flows.

Initiation Rule:

Start Event - Notify new install requests and new equipment has been ordered (Note: Workflow can be manually or automatically started by initiation or triggering events).

Execution Rule:

Event - triggers or kick-starts the rule: “When Install new equipment”

Condition - criteria for the rule to execute: “If rack utilization is greater than 80% of rack capacity”

Action - can be performed within the workflow or externally by the users of the workflow.

The execution rule is used to check rack space availability. The decision to install a new server onto a rack depends on the rule. Through the event “When Install new equipment”, the rule links two processes “Manage Rack Space Availability” and “Order New Rack”. The event “When Install new equipment” is observed in relation to process “Create Request to install new server”, then the rule which depends on this event is invoked and leads to an action which performs the transition to “Order New Rack” process.

Termination Rule:

End event - Workflow can be manually or automatically ended by termination event trigger. The workflow termination is always based on the termination rule, invoked by a suitable termination event AFTER the process is finished, or on a process execution rule DURING execution in the case of emergency.

Following the terminology of the object models of [9, 21] we refer to Process, Flow (Material, Information) and Rule (Event, Condition and Action) as first-class objects.

Representing them as first-class objects is conceptually and computationally easier because they may have several function characteristics, which may be added or deleted in the design stage [32].

Object Properties

Informally speaking, the business rules and workflows can be constructed in terms of object characteristics. The object properties provide information about the characteristics of the objects. For example, the object “Process” may have as properties process id, name, status, and creation date. From the viewpoint of the conceptualization of our ontology, object properties can be classified onto one of the following types:

Identification properties - examples are process id, name, type, context and scope;

Qualitative description properties - these are categorical or nominal properties, which can be described only qualitatively – for example, status, deviation, and trend.

Quantitative description properties – these properties can be described using a fixed value, which can be estimated qualitatively or specified quantitatively- for example, the number of closed processes in a chemical plant.

[131] describes object properties as a common approach to specify characteristics or attributes of a real-world object instance, which in turn helps to understand how to interact with the object. An object property value may be of different primitive type, including numeric, non-numeric (strings/text/etc.), Boolean, etc. Properties may have single or multiple values. By introducing property characterisation for each object, our model can fulfil the requirements for flexibility and maintainability of the formulation of Business Rules and the versatility of the Process Workflow. Since the objects are building blocks of both the process workflows and the business rules which govern them, the object properties are the main vehicle for analysing the dependencies between the business rules themselves. They will be the bridge between the process ontology and the algorithm for propagating the changes in the business rules. The primary role of qualitative and quantitative property measures is to accurately describe object properties rather than the usual identification and classification. The more sophisticated are the properties, the more elaborated are the

dependencies we can formulate. Some object properties may be used to identify, name and categorize the objects. Others may be used to quantify and qualify the objects. There are circumstances where qualitative and quantitative properties are also used for identification of an object. We can even introduce properties for “potentially active” characterisation of the objects, like reflexive regularities, directed constraints and associative interdependencies between the properties of several objects. For instance, Business Rules may involve an array of object properties with objective estimation based on value measurement along with highly subjective value judgments based on qualitative estimations. Finally, using the object properties we can organise them into groups and hierarchies which enables the use of object-oriented technology. Using Object model concepts as described by David in [25] and EBNF notation as described in [34], each concept established in the previous section is presented in a separate class in the following sections:

Flow Class

Flow Class provides an important concept that allows us to manage and control the flow of both Information and Material Flows between processes. Hence, Flow Class consists of Information Flow Class and Material Flow Class.

Information Flow Class

Information Flow depicts and expresses shared data between processes by which a Business Rule is applied or imposed. It is a primary construct for the proposed model. Information Flow may be connected directly or indirectly to Material Flow. Information Flow is made up of one or more objects. Objects are made up of properties including object identification, qualitative and quantitative property measures. The following EBNF Information Flow definition depicts objects and properties:

```

<Information Flow> ::=
    <Object> |
    <Information Flow> <Binary Operators> <Information Flow> |
    <Information Flow> <Binary Operators> <Value> |
    <Object Property> |
    <Quantitative Property Measure> |
    <Qualitative Property Measure>
    <Object Property> ::= <Object> <Property> |
    <Object Property> <Binary Operators> <Object Property> |
    <Object Property> <Binary Operators> <Value> |
    <Unary Operators> <Object Property>
    <Quantitative Property Measure> ::= <Object> <Property> <Value>
    <Qualitative Property Measure> ::= <Quality Measure> <Object> <Property> <Value>
    <Property> ::= attribute <Domain Type> <Property Name>
    <Quality Measure> ::= String
    <Value> ::= Decimal | String | Date | Boolean
    <Binary Operators> ::= <Identity Operators> | <Logical Operators>
    <Identity Operator> ::= == | != | > | >= | < | <=
    <Logical Operators> ::= && Conditional-AND | || Conditional-OR
    <Unary Operators> ::= More | Less | Not

```

Material Flow Class

Material Flow Class represents physical resources or goods (input, observed and output) used and transformed by business processes. Material Flow Class will be used to express resource dependencies between processes. The consumed resources (input and observed) may produce one or more output resources. In a nutshell, Material Flow Class is made up of one or more objects consisting of input, observed and output resources. Therefore, we propose Material Flow Class be identified by three flow types namely “input”, “observed” and “output”. Like Information Flow in the section above, Material Flow Class will be made up of three kinds of properties. These are identification property, qualitative and quantitative property measures. The following EBNF Material Flow definition is a part of the Workflow level model depicting objects and properties:

```

<Material Flow> ::=
    <Flow Type> <Object> |
    <Material Flow> <Binary Operators> <Material Flow> |
    <Material Flow> <Binary Operators> <Value> |
    <Object Property> |
    <Quantitative Property Measure> |
    <Qualitative Property Measure>
<Object Property> ::=
    <Flow Type> <Object> <Property> |
    <Object Property> <Binary Operators> <Object Property> |
    <Object Property> <Binary Operators> <Value> |
    <Unary Operators> <Object Property>
<Quantitative Property Measure> ::= <Object> <Property> <Value>
<Qualitative Property Measure> ::= <Quality Measure> <Object> <Property> <Value>
<Property> ::= attribute <Domain Type> <Property Name>
<Flow Type> ::= "input" | "observed" | "output"
<Value> ::= Decimal | String | Date | Boolean
<Binary Operators> ::= <Identity Operators> | <Logical Operators>
<Identity Operator> ::= == | != | > | >= | < | <=
<Logical Operators> ::= && Conditional-AND | || Conditional-OR
<Unary Operators> ::= More | Less | Not

```

We have now identified and established Information Flow Class and Material Flow Class. The Flow Class is a superclass of Information Flow Class and Material Flow Class. The following EBNF Flow definition depicts Information and Material objects:

```

<Flow> ::=
    <Information Flow> <Material Flow> <Flow> |
    <Information Flow> <Flow> |
    <Material Flow> <Flow>

```

Process Class

As described in BPMN and BPEL models, a workflow consists of one or more processes. A process represents a well-defined business activity or function designed to receive some input and produce some output. For example, "Manage Rack Space Availability" is a well-defined business activity in a data centre (Figure 4.4). This process is designed to receive some input about the rack detail (rack name) and produce amount of space available or utilized for each rack in a data centre. Generally, Process is designed to emphasise how a unit of work is done and what is needed to accomplish the work. Hence the following statements are true:

- Each process is associated with a system or workflow user or role responsible for its implementation. Note, Process role or user is out of scope for this research.

- A process uses Information and Material Flows to implement activities. Information, unlike material, is not transformed by the process, but rather it is used as informative to the process. On the other hand, material can be used by the process to create or produce new materials. In this research we refer to processed input as observed or parameters. The final converted or transformed input we referred to as output.

The preliminary Process definition aims at providing contextual information which applies to Process in a workflow. The information includes properties of the process as well as associated objects such Flow Objects. The process will be represented as object and properties necessary to initiate creation and support its execution. The following EBNF Process definition depicting objects and properties:

```

<Process Object> ::= <Object> <Process Info> |
                    <Process Object> AND <Process Object> |
                    <Process Object> OR <Process Object> |
                    <Process Object> XOR <Process Object> |
                    NOT <Process Object> |
                    <Process Property>

<Process Info> ::= <Flow>

<Process Property> ::= <Object> <Property> |
                     <Process Property> AND <Process Property> |
                     <Process Property> OR <Process Property> |
                     <Process Property> XOR <Process Property> |
                     NOT <Process Property>

<Flow> - Refer to Flow Class section - Flow Object Definition

```

Event Class

Event Class provides a concept of communication within and across Process and Rule levels. The event class represents both synchronous and asynchronous events which may happen during workflow execution. Additionally, the events always have a temporal dimension – at what time (absolutely or relatively to the beginning of the workflow execution). An event signals or notifies that an incident has occurred or is going to occur. In brief, an event is an occurrence of some sort during the time of a process. An event has a great control over the behaviour of business processes and actions in workflow; for

instance, consider a Business Rule, **“When request to install new server, if Rack Utilization is greater than Rack Capacity then send email to DC Manager”**. In this Business Rule, the event is **“When a request to install a new server”**. So, the **“Check Available Space”** process will not happen until the event **“When a request to install a new server or equipment”** becomes true.

The definition of Event Class needs to include not only operations or actual events but also source and target of the Signal object. Event affects the flow of the Process, usually handled by a catch and throw mechanism. In workflow, the term “event” is very general, used to express many things. The start of process, the end of process, the change of state of process, information or message that arrives, etc., all could be considered events. However, we restrict the use of events to include only those types of events that will affect the flow of Process Class. We categorize Event Class into three main types namely Start Event, Observed Event and End Event. The categories can be triggered by:

- Timer can be set to start, monitor, or end the Process
- Information (Message) and Material flow received from workflow participant
- Conditions become true or false
- Escalations
- Signal warnings, faults or errors interrupting the process
- Cancellations

The following EBNF Event definition depicting objects and properties:

```

<Event Object> ::= <Event Time> <Event Operation> <Signal Object> |
                  <Event Object> AND <Event Object> |
                  <Event Object> OR <Event Object> |
                  <Event Object> XOR <Event Object> |
                  NOT <Event Object> |
                  <Event Property>

<Event Time> ::= "Before" | "When" | "After" -- (Synchronous or Asynchronous)
<Event Operation> ::= "Start" | "Stop" | "Begin" | "End" | "Submit" | Change | "On Error" |
"Signal" | "Alert" | "Open" | "Close" | "Click" | "User Defined Event"
<Signal Object> ::= <Source Object> | <Target Object>
<Source Object> ::= <Object>
<Target Object> ::= <Object>
<Event Property> ::= <Object> <Property> |
                    <Event Property> AND <Event Property> |
                    <Event Property> OR <Event Property> |
                    <Event Property> XOR <Event Property> |
                    NOT <Event Property>

```

User Defined Event refers to other events not defined in this definition.

Condition Class

Condition is a logic statement that specifies what must be checked to enable a true or false evaluation of some records. This evaluation is necessary to fire the rules. For instance, in Figure 4.4, the condition is specified as **"If Rack Space Utilization is greater than the 80% of Rack Capacity"**. Condition has the following functions:

- Use to define, filter or constrain some aspect of Information and Material
- Manage and control events.
- Determine and guide transition of processes that come after rule execution.

Condition may take a form of an expression, a database query, function or procedure calls. This research considers only expression conditions. Conditional may contain multiple expression join by logical connectives such as "AND" and "OR". The following EBNF Condition definition depicting objects and properties:

```

<Condition Object> ::= <Object> <Condition Object> |
                       <Condition Object> AND <Condition Object> |
                       <Condition Object> OR <Condition Object> |
                       <Condition Object> XOR <Condition Object> |
                       NOT <Condition Object> |

<Condition Property> ::= <Condition Property>
                        <Object> <Property> |
                        <Condition Property> AND <Condition Property> |
                        <Condition Property> OR <Condition Property> |
                        <Condition Property> XOR <Condition Property> |
                        NOT <Condition Property>

```

Action Class

An Action is defined as what is done to other objects with a possible outcome. For instance, an action to “send email”, “send” is an action and “email” is an object. Each action may involve one or more objects; in turn objects and object properties are created or transformed. Furthermore, an execution of one action may cause in one or more further actions to occur. A workflow can contain many actions as part of business rule execution. Action Class specifies what needs to be implemented to complete the workflow process or rule. The following EBNF Action definition depicts objects and properties:

```

<Action Object> ::= <Action Implementation> <Object> |
                   <Action Object> AND <Action Object> |
                   <Action Object> OR <Action Object> |
                   <Action Object> XOR <Action Object> |
                   NOT <Action Object> |
                   <Action Property>
<Action Implementation> ::= <Database Operation> | <User Defined Operation>
<Database Operation> ::= “Select” | “Insert” | “Delete” | “Update”
<User Defined Operation> ::= <Other Operations>
<Action Property> ::= <Object> <Property> |
                     <Action Property> AND <Action Property> |
                     <Action Property> OR <Action Property> |
                     <Action Property> XOR <Action Property> |
                     NOT <Action Property>

<Other Operations>

```


4.5 Business Rules Relationships

According to [15], “No other topic in the BPM arena has suffered from more misinformation, disinformation and wilful ignorance as the relationship between business process and business rules. These two disciplines are most often put forward as alternative approaches rather than complementary aspects of managing the business. Business process management (BPM) and business decision management (BDM) need to be used together. Unfortunately, each discipline has historically spoken only to its own concerns with little interest in how it integrates with the other in fact with little understanding of what the other is trying to do”. Thus, the principle of functional dependency is adopted to express business rule components relationships in workflows to align with business processes. The concepts of business rules are semantically related to the business processes and applicable within workflow domain. The relationship between business rules and processes can be described as follows:

- Processes produce and respond to events, which can be fired by one or more rules
- Every rule produces two or more events where it needs to fire
- Processes transform/produce outputs from inputs.
- Rules evaluate whether the output is desired/acceptable or not.

4.5.1 Business Rules Formal Description

Consider a Business Rule set R containing a collection of rule samples controlling a workflow. A Rule set R has one or more related rules that have been put together to guide the movement of processes in the workflow. For instance, R may be made up of Initiation Rule, Flow Rule, Event or Process Rules and Termination Rule. Let every Rule in R be indexed $R = \{R_i, | i = 1, \dots, n\}$. Each Rule definition R_i consists of a collection of Event (E_i), Condition (C_i) and Action (A_i). We refer to E_i , C_i and A_i as sets of events, conditions and actions and call them components of R_i . Now, let E be expressed in terms of $\{E_i, | i = 1, \dots, n\}$. And C be expressed in terms of $\{C_i, | i = 1, \dots, n\}$. Also A be expressed in terms of $\{A_i, | i = 1, \dots, n\}$. In this research, we will use notation $E_{1i}(R_1)$, $C_{1i}(R_1)$ and $A_{1i}(R_1)$ where $E_{1i} \in E_1$, $C_{1i} \in C_1$ and $A_{1i} \in A_1$ to represent Business Rule basic definition. Note that for simplicity reasons, if a part of the Business Rule has no importance in a discussion, then it

will be omitted. For example, $C_{1i}(R_1)$ and $A_{1i}(R_1)$ will represent a Business Rule that contains Conditions and Actions only.

4.5.2 Relationships between Business Rules

The existence of a dependency between two rules expresses that communication occurs between components (Event, Condition, and Action) of the Business Rule. For example, one Business Rule action may invoke or trigger conditions of other Business Rules or the condition of one Business rule may depend on an event of another Business Rule. Therefore, Business Rules relationships can be described by analysing Business Rule components relationships. We consider the relationship between two rules to be represented by the symbol *Relate to* \rightarrow . For example, R_1 *Relate to* $\rightarrow R_2$ means Rule 1 relates to Rule 2. If one of R_1 action activates event for R_2 , we declare as $A_{1i}(R_1)$ *Relate to* $\rightarrow E_{2j}(R_2)$.

The structure of business rules relationships can be analysed and declared in one of the following six possible ways:

$E_{1i}(R_1)$ *Relate to* $\rightarrow E_{2j}(R_2)$

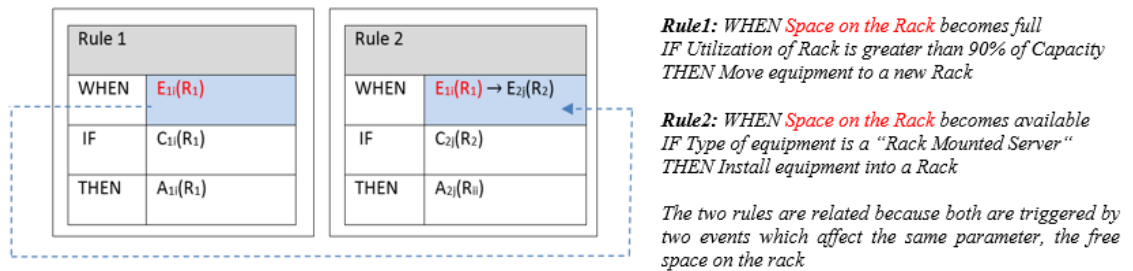


Figure 4.5.2 1 Event to Event Relationships

$E_{1i}(R_1)$ *Relate to* $\rightarrow C_{2j}(R_2)$

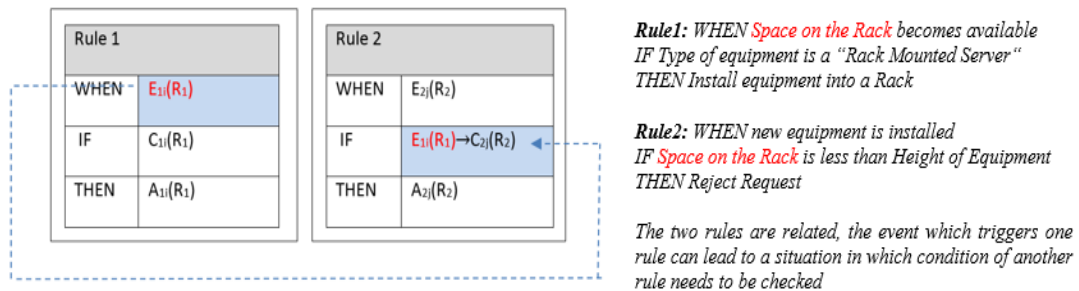
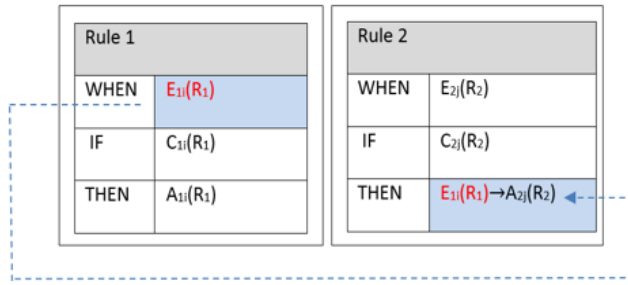


Figure 4.5.2 2 Event to Condition Relationships

$E_{1i}(R_1)$ Relate to $\rightarrow A_{2j}(R_2)$



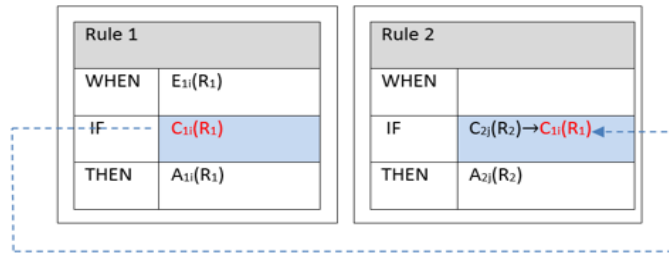
Rule1: WHEN *Space on the Rack* becomes available
IF Type of equipment is a "Rack Mounted Server"
THEN Install equipment into a Rack

Rule2: WHEN new equipment is installed
IF Utilization of Rack is less than 90% of Capacity
THEN New *Space on the Rack* = Old Space on Rack + Height of installed equipment

This relation between the rules will require the two rules to be compiled in a chain so that they are fired one after another in a strict order.

Figure 4.5.2 3 Event to Action Relationships

$C_{1i}(R_1)$ Relate to $\rightarrow C_{2j}(R_2)$



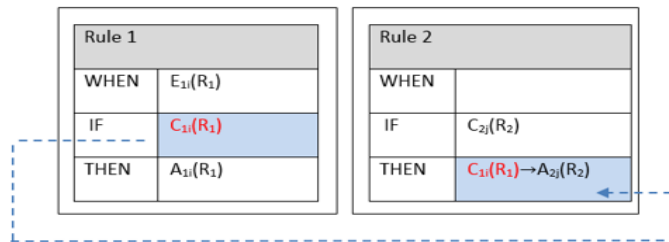
Rule1: WHEN *Space on the Rack* becomes available
IF Type of equipment is a "Rack Mounted Server"
THEN add equipment into a Rack

Rule2:
IF Rack named "Rack 01" and Type of equipment installed is a Rack Mounted
THEN Set Rack Type = Server Cabinet

A dependency exists between two rules via related Condition object (Equipment) and Property(Type). The result is that Rule 2 condition will be evaluated after Rule 1 condition. Is evaluated

Figure 4.5.2 4 Condition to Condition Relationships

$C_{1i}(R_1)$ Relate to $\rightarrow A_{2j}(R_2)$



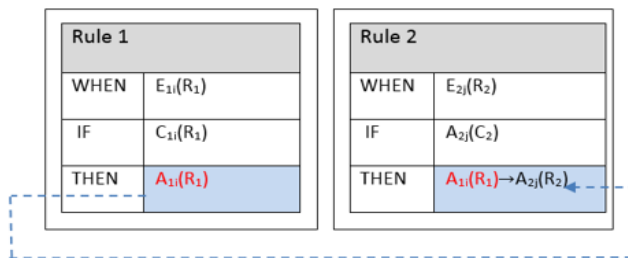
Rule1: WHEN Cancellation Alert submitted
IF Cancellation reasons of Request is no space availability and Status of Request is "Cancel"
THEN set Step of Request to "Cancel Approval"

Rule2:
IF Space on Rack is full
THEN change Status of Request to close
IF Space on Rack is not full
THEN change Status of Request to open

A dependency exists between two rules via related object property Status(Request). The result is that Rule 1 condition will cause Rule 2 action to execute

Figure 4.5.2 5 Condition to Action Relationships

$A_{1i}(R_1)$ Relate to $\rightarrow A_{2j}(R_2)$



Rule1: IF Decommission Date of equipment >30 Days and Reviewed Requirement-Status is approved
THEN Send message to manager and set category of Equipment to "risky"

Rule2: IF Category of Equipment is "risky"
THEN Discard "risky" Equipment

An action instance of Rule 1 influences the occurrence of action instance of Rule 2. A dependency exists between two rules via related Action object (Equipment). The result is Rule 2 action evaluates after Rule 1 executed its action.

Figure 4.5.2 6 Action to Action Relationships

These relationships are defined based on Objects and Objects properties involved in Condition, Event and Action components of the Rules. Moreover, relationship can be defined in terms of qualitative and quantitative characteristics of the object parameters. We examined six ways (Figures 4.5.2.1 - 4.5.2.6) of representing rule relationships based on the partial order relationships. However, it is far simpler and more natural to apply the tree structure to model and picture relationships between rules. Therefore, the next section introduces AND-OR dependency graphs and tree.

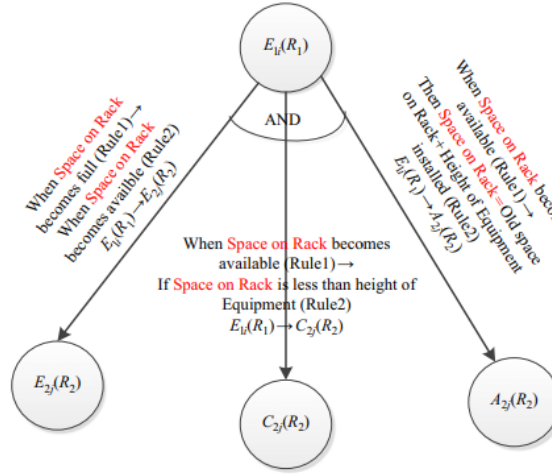
4.5.3 Business Rules Dependency Graphs (AND-OR Graphs)

The dependency graph is constructed using nodes starting with the root and going down to its leaves. The nodes will represent Rule components (ECA) and the edges will represent relationships between components of rules. Navigation through the graph forms the patterns; each pattern is illustrated in Figures 4.5.3.1.1 to 4.5.3.3.3. Dependency graphs or tree structures are widely used to describe rules order and priorities; a graph can be made up of many rules presented in an analytical and visual manner [24]. As the name AND-OR graph suggests, the relationships will be of two kinds: AND relationships, which group several rules that can be invoked consecutively, and OR relationships, which group several rules that can be invoked alternatively. Variations of AND-OR relationships exist, including Direct AND Dependency, Direct OR Dependency, Indirect AND dependency and Indirect OR Dependency.

The structuring of the rules into AND-OR graphs would allow the implementation of more efficient rules' propagation algorithms. Furthermore, the different patterns of inclusion of the rules in the trees will be used inside the algorithms to control the flow of execution of the rules as the business processes progress in real-time. In addition, we can describe behaviour and flow dependency patterns of rules. For each dependency pattern, we can provide a visual representation of the rule dependency. It is important to understand that although trees make it easier to understand the relationship between rules, they will need to be translated into rule language for workflow interpretation. Hence algorithms will be defined in addition to rule relationships definitions.

4.5.3.1 Direct AND Dependency patterns

Rule's Event-AND Graph



The Event-AND graph is so named because the Event component of one rule ($E_{1i}(R_1)$) forces another rule (R_2) to be invoked. R_2 is invoked when R_1 's event components relate to R_2 's event and condition and action components in such a way that the event of R_2 is causally connected to R_1 's event. If R_2 's condition is met, then R_2 's action will execute regardless of R_2 's event

Such dependence can be established using pattern matching of the rule components during rule acquisition. The rules can be indexed appropriately, which would facilitate the real-time control as well as the offline adaptation of the rule at a later stage

Figure 4.5.3.1. 1 Strong Direct Event-AND Graph

The above tree represents a direct AND dependency where each node corresponds to the root node/rule $E_{1i}(R_1)$. The following patterns are depicted:

- Direct edge ($E_{1i}(R_1), E_{2j}(R_2)$), with $E_{1i} \rightarrow E_{2j}$, means that the event of rule R_1 must influence the result of rule R_2 's event. This is $E_{1i}(R_1) \text{ Relate to } \rightarrow E_{2j}(R_2)$ relationship. An event instance of Rule₁ $E_{1i}(R_1)$ influences the occurrence of event instance of Rule₂ $E_{2j}(R_2)$. To analyse this scenario, suppose $E_{1i}(R_1) = \text{Request (Rule1)}$ and $E_{2j}(R_2) = \text{Request (Rule2)}$. Then we can say a dependency exists between two rules via a related Event object (Request). The result is that the workflow will evaluate a Rule₂ event after Rule₁ has executed its event. Hence, a change, introduced in Rule₁'event, may propagate through the dependencies to Rule₂'components.
- Direct edge ($E_{1i}(R_1), C_{2j}(R_2)$), with $E_{1i} \rightarrow C_{2j}$, means that the event of rule R_1 must influence the result of rule R_2 's condition. This is $E_{1i}(R_1) \text{ Relate to } \rightarrow C_{2j}(R_2)$ relationship. An object property of an event instance of Rule₁ $E_{1i}(R_1)$ influences the occurrence of an object property of condition instance of Rule₂ $C_{2j}(R_2)$. To analyse this scenario, suppose $E_{1i}(R_1) = \text{Threshold (Rack (Rule1))}$ and $C_{2j}(R_2) = \text{Threshold (Rack (Rule2))}$. Then we can say a dependency exists between two rules

via a related object property Threshold (Rack). The result is that the workflow will evaluate the Rule₂ condition after Rule₁ has executed its event. Hence, a change, introduced in Rule₁'event, may propagate through the dependencies to Rule₂'components.

- Direct edge ($E_{1i}(R_1), A_{2j}(R_2)$), with $E_{1i} \rightarrow A_{2j}$, means that the event of rule R_1 must cause change to rule R_2 's action. This is $E_{1i}(R_1) \text{ Relate to } \rightarrow A_{2j}(R_2)$ relationship. An event instance of Rule₁ $E_{1i}(R_1)$ influences the occurrence of action instance of Rule₂ $A_{2j}(R_2)$. To analyse this scenario, suppose $E_{1i}(R_1) = \text{Request (Rule1)}$ and $A_{2j}(R_2) = \text{Request (Rule2)}$. Then we can say a dependency exists between two rules via a related Event object (Request) and Action object (Request). The result is that the workflow will evaluate the Rule₂ event after Rule₁ has executed its event. Therefore, a change, introduced in the Rule₁'event may propagate through the dependencies to Rule₂'components.

We can also depict the following possible combination of AND patterns:

- $E_{1i}(R_1) \xrightarrow{\text{Relate to}} E_{2j}(R_2) \wedge E_{1i}(R_1) \xrightarrow{\text{Relate to}} C_{2j}(R_2)$
- $E_{1i}(R_1) \xrightarrow{\text{Relate to}} C_{2j}(R_2) \wedge E_{1i}(R_1) \xrightarrow{\text{Relate to}} A_{2j}(R_2)$
- $E_{1i}(R_1) \xrightarrow{\text{Relate to}} E_{2j}(R_2) \wedge E_{1i}(R_1) \xrightarrow{\text{Relate to}} A_{2j}(R_2)$
- $E_{1i}(R_1) \xrightarrow{\text{Relate to}} E_{2j}(R_2) \wedge E_{1i}(R_1) \xrightarrow{\text{Relate to}} C_{2j}(R_2) \wedge E_{1i}(R_1) \xrightarrow{\text{Relate to}} A_{2j}(R_2)$

Rule's Condition-AND Graph

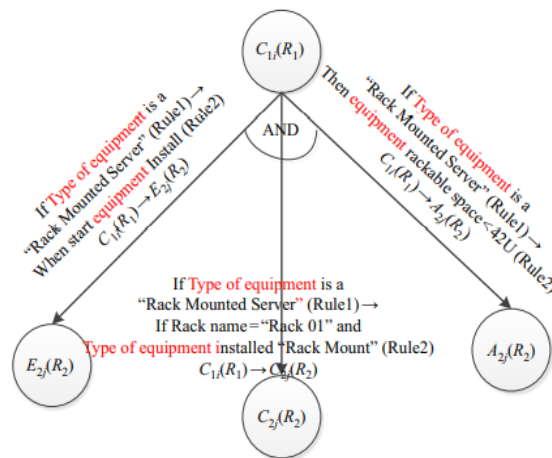


Figure 4.5.3.1. 2 Strong Direct Condition-AND Graph

The Condition-AND graph is so named because the condition component of one rule ($C_{1i}(R_1)$) influences another rule (R_2) to be invoked. If condition of R_1 is satisfied and its components relate to R_2 's event, condition and action then R_2 is also fired

This type of dependence is similar to the Event-AND dependence described earlier, but in this case the subsumption is between the conditions rather than between the event components of the rules. It can be the basis for another indexing scheme, similarly to Event-AND dependence.

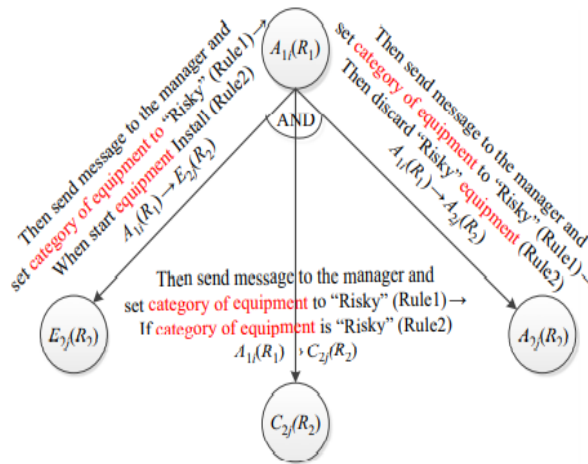
This tree represents a direct AND dependency where each node corresponds to the root node/rule $C_{1i}(R_1)$. The following patterns are depicted:

- Direct edge $(C_{1i}(R_1), E_{2j}(R_2))$, with $C_{1i} \rightarrow E_{2j}$, means that the condition of rule R_1 must influence or trigger rule R_2 's event. This is $C_{1i}(R_1) \text{ Relate to } \rightarrow E_{2j}(R_2)$ relationship.
- Direct edge $(C_{1i}(R_1), C_{2j}(R_2))$, with $C_{1i} \rightarrow C_{2j}$, means that the condition of rule R_1 must influence the result of rule R_2 's condition. This is $C_{1i}(R_1) \text{ Relate to } \rightarrow C_{2j}(R_2)$ relationship.
- Direct edge $(C_{1i}(R_1), A_{2j}(R_2))$, with $C_{1i} \rightarrow A_{2j}$, means that the condition of rule R_1 must cause change to rule R_2 's action. This is $C_{1i}(R_1) \text{ Relate to } \rightarrow A_{2j}(R_2)$ relationship.

We can also depict the following possible combination of AND patterns:

- $C_{1i}(R_1) \xrightarrow{\text{Relate to}} E_{2j}(R_2) \wedge C_{1i}(R_1) \xrightarrow{\text{Relate to}} C_{2j}(R_2)$
- $C_{1i}(R_1) \xrightarrow{\text{Relate to}} C_{2j}(R_2) \wedge C_{1i}(R_1) \xrightarrow{\text{Relate to}} A_{2j}(R_2)$
- $C_{1i}(R_1) \xrightarrow{\text{Relate to}} E_{2j}(R_2) \wedge C_{1i}(R_1) \xrightarrow{\text{Relate to}} A_{2j}(R_2)$
- $C_{1i}(R_1) \xrightarrow{\text{Relate to}} E_{2j}(R_2) \wedge C_{1i}(R_1) \xrightarrow{\text{Relate to}} C_{2j}(R_2) \wedge C_{1i}(R_1) \xrightarrow{\text{Relate to}} A_{2j}(R_2)$

Action-AND Graph



The Action-AND graph is so named because the Action component of one rule ($A_{1i}(R_1)$) causes another rule (R_2) to be invoked consecutively. R_2 is invoked upon execution of R_1 's action and R_1 's action components relate to R_2 's (event and condition and action) components.

There are two possible interpretations of this type of dependence between the rules – unconditional chaining of the action components or conditional chaining of the action components. In both cases the indexing algorithm will be identical but depending on the intended interpretation the runtime behaviour may be different.

Figure 4.5.3.1. 3 Strong Direct Action-AND Graph

This tree represents a direct AND dependency where each node corresponds to a root node/rule $A_{1i}(R_1)$. The following patterns are depicted:

- Direct edge $(A_{1i}(R_1), E_{2j}(R_2))$, with $A_{1i} \rightarrow E_{2j}$, means that the action of rule R_1 must influence the result of rule R_2 's event. This is $A_{1i}(R_1) \text{ Relate to } \rightarrow E_{2j}(R_2)$ relationship.
- Direct edge $(A_{1i}(R_1), C_{2j}(R_2))$, with $A_{1i} \rightarrow C_{2j}$, means that the action of rule R_1 must influence the result of rule R_2 's condition. This is $A_{1i}(R_1) \text{ Relate to } \rightarrow C_{2j}(R_2)$ relationship.

We can also depict the following possible combination of the AND pattern:

$$\square \quad A_{1i}(R_1) \xrightarrow{\text{Relate to}} E_{2j}(R_2) \wedge A_{1i}(R_1) \xrightarrow{\text{Relate to}} C_{2j}(R_2)$$

Consider patterns identified from Figures 4.5.3.1.1- 4.5.3.1.3. Such dependency patterns only appear when there is a strong relationship between one or more rules. The patterns are based on an AND join, one node (rule) is directly joined to another node (rule) through related components (event, condition, action). The relationship may include relation between objects, quantitative estimation of a property, and qualitative estimation of a property as well as relation between properties of object components (event, condition, and action). A combination of nodes (rules) can also be linked through an AND join.

4.5.3.2 Direct OR Dependency patterns

Rule's Event-OR Graph

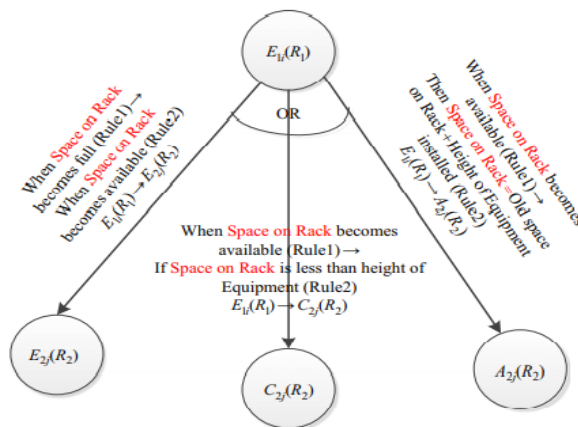


Figure 4.5.3.2. 1 Weak Direct Event-OR Graph

The Event-OR graph is so named because the Event component of one rule ($E_{1i}(R_1)$) may or may not trigger another rule. It may cause an event of another rule (R_2 's event) to be invoked or cause the condition of another rule (R_2 's condition) to be checked, regardless of (R_2 's event), or may cause the action of another rule (R_2 's action) to be executed.

There are different possible intended interpretations of the Event-OR dependencies. Our preference is that rules of this type introduce an alternative flow of control, thus forming a dynamically algorithmic structure. Another possible interpretation could be that such rules govern the processes asynchronously.

This tree represents a direct OR dependency where the following possible combination patterns are depicted when $E_{1i}(R_1)$ is a root node/rule:

- The execution of $E_{1i}(R_1)$ may or may not trigger the execution of $E_{2j}(R_2)$ depending on additional events, conditions or actions from the class, or guided by external events so each of these cases introduces a different degree of “weakness”
- The execution of $E_{1i}(R_1)$ may or may not trigger the execution of $C_{2j}(R_2)$ depending on additional events, conditions or actions from the class, or guided by external events.
- The execution of $E_{1i}(R_1)$ may or may not trigger the execution of $A_{2j}(R_2)$ depending on additional events, conditions or actions from the class, or guided by external events

We can also depict the following possible combination of OR patterns:

$$\begin{aligned}
 &\circ E_{1i}(R_1) \xrightarrow{\text{Relate to}} E_{2j}(R_2) \vee E_{1i}(R_1) \xrightarrow{\text{Relate to}} C_{2j}(R_2) \\
 &\circ E_{1i}(R_1) \xrightarrow{\text{Relate to}} C_{2j}(R_2) \vee E_{1i}(R_1) \xrightarrow{\text{Relate to}} A_{2j}(R_2) \\
 &\circ E_{1i}(R_1) \xrightarrow{\text{Relate to}} E_{2j}(R_2) \vee E_{1i}(R_1) \xrightarrow{\text{Relate to}} A_{2j}(R_2) \\
 &\circ E_{1i}(R_1) \xrightarrow{\text{Relate to}} E_{2j}(R_2) \vee E_{1i}(R_1) \xrightarrow{\text{Relate to}} C_{2j}(R_2) \vee E_{1i}(R_1) \xrightarrow{\text{Relate to}} A_{2j}(R_2)
 \end{aligned}$$

Rule's Condition-OR Graph

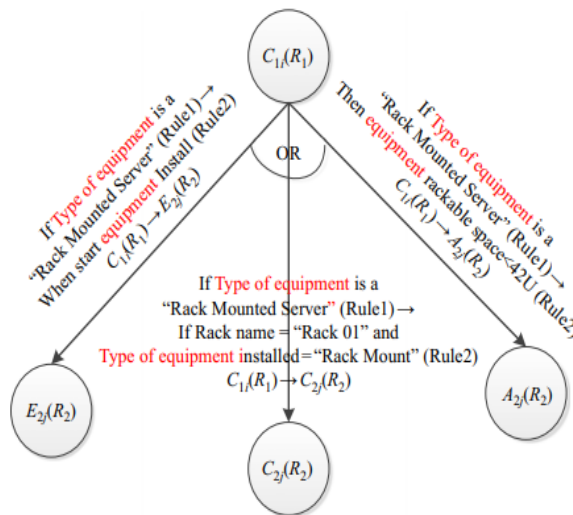


Figure 4.5.3.2. 2 Weak Direct Condition-OR Graph

The Condition-OR graph is so named because the condition component of one rule ($C_{1i}(R_1)$) may cause another rule to be checked. The event of R_2 is invoked when R_1 's condition relates to either R_2 's event or condition or action components.

As in the previous dependence structure, there are different possible intended interpretations. Our choice is that rules with such a dependence may split the control flow into concurrent subflows to control the concurrently executed business processes, subject to additional conditions according to the condition components of the rules.

This tree represents a direct OR dependency where the following possible combination patterns are depicted when $C_{1i}(R_1)$ is a root node/rule:

- The execution of $C_{1i}(R_1)$ may or may not trigger the execution of $E_{2j}(R_2)$ depending on additional events, conditions or actions from the class, or guided by external events so each of these cases introduces a different degree of “weakness”.
- The execution of $C_{1i}(R_1)$ may or may not trigger the execution of $C_{2j}(R_2)$ depending on add on events, conditions or actions from the class, or guided by external events
- The execution of $C_{1i}(R_1)$ may or may not trigger the execution of $A_{2j}(R_2)$ depending on add on events, conditions or actions from the class, or guided by external events

We can also devise the following possible combination of OR patterns:

- $C_{1i}(R_1) \xrightarrow{\text{Relate to}} E_{2j}(R_2) \vee C_{1i}(R_1) \xrightarrow{\text{Relate to}} C_{2j}(R_2)$
- $C_{1i}(R_1) \xrightarrow{\text{Relate to}} C_{2j}(R_2) \vee C_{1i}(R_1) \xrightarrow{\text{Relate to}} A_{2j}(R_2)$
- $C_{1i}(R_1) \xrightarrow{\text{Relate to}} E_{2j}(R_2) \vee C_{1i}(R_1) \xrightarrow{\text{Relate to}} A_{2j}(R_2)$
- $C_{1i}(R_1) \xrightarrow{\text{Relate to}} E_{2j}(R_2) \vee C_{1i}(R_1) \xrightarrow{\text{Relate to}} C_{2j}(R_2) \vee C_{1i}(R_1) \xrightarrow{\text{Relate to}} A_{2j}(R_2)$

Rule's Action-OR Graph

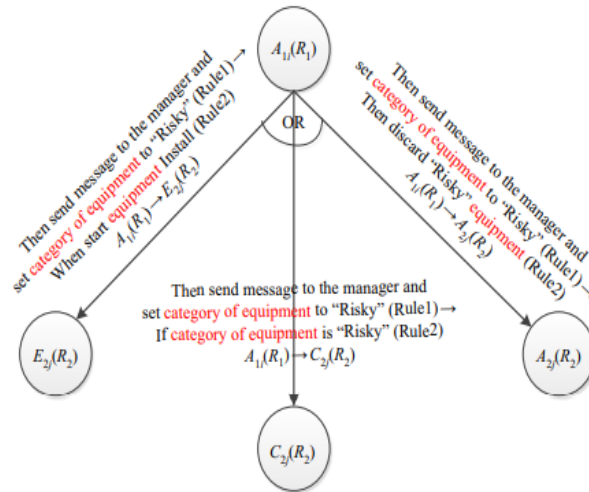


Figure 4.5.3.2. 3 Weak Direct Action-OR Graph

The Action-OR graph is so named because the Action component of one rule ($A_{1i}(R_1)$) may result in triggering an event of another rule (R_2 's event) or cause the Condition of another rule (R_2 's condition) to be checked or may cause the action of another rule (R_2 's action) to be executed.

This type of dependency between the rules can be interpreted as an indication for unconditional splitting of the control flow into concurrent flows at runtime. Another possible intended interpretation could be that such dependence exists between rules which control process execution within workflows synchronously.

The above tree represents a direct OR dependency where the following possible combination patterns are depicted when $A_{1i}(R_1)$ is a root node/rule:

- The execution of $A_{1i}(R_1)$ may or may not trigger the execution of $E_{2j}(R_2)$ depending on additional events, conditions or actions from the class, or guided by external events so each of these cases introduces a different degree of weakness.
- The execution of $A_{1i}(R_1)$ may or may not trigger the execution of $C_{2j}(R_2)$ depending on add on events, conditions or actions from the class, or guided by external events.
- We can also devise the following possible combination of OR patterns:

$$A_{1i}(R_1) \text{ Relate to } \rightarrow E_{2j}(R_2) \vee A_{1i}(R_1) \text{ Relate to } \rightarrow C_{2j}(R_2)$$

Consider patterns identified from Figures 4.5.3.2.1 - 4.5.3.2.3. Such dependency patterns only appear when there is a weak relationship between one or more rules. These dependency patterns are based on an OR join, one node (rule) is directly joined to another node (rule) through related components (event, condition, action). The relationship may include the relation between objects, quantitative estimation of a property, and qualitative estimation of a property as well as relation between properties of objects/ components (event, condition, and action). A combination of nodes (rules) can also be linked through an OR join.

4.5.3.3 Indirect AND Dependency patterns

Rule's Indirect Event-AND Graph

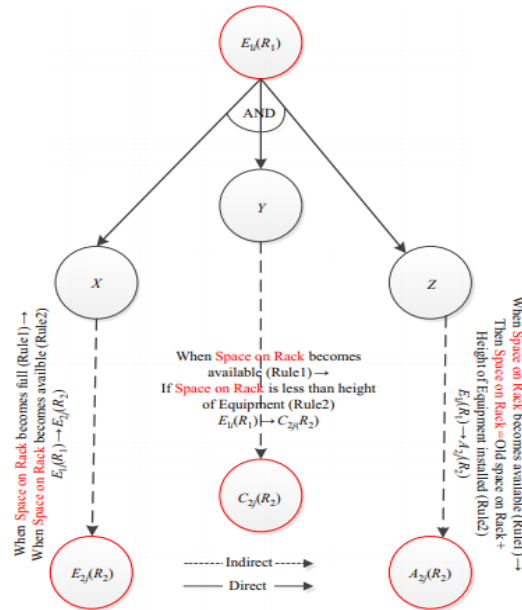


Figure 4.5.3.3. 1 Strong Indirect Event-AND Graph

The Indirect Event-AND graph is so named because the Event component of one rule ($E_{1i}(R_1)$) indirectly causes another rule (R_2) to be invoked. The source rule is linked to the target rule or rules via other rules. The event of R_2 is subsumed by R_1 's event. If the condition of R_2 is met, R_2 's action will execute regardless of R_2 's event.

This type of dependency requires preliminary analysis of the events which trigger the rules. It may be particularly useful if there is a taxonomic classification of the events, conditions and actions, since it may introduce useful patterns of control, specific to the problem domain. For example, children nodes might be interpreted as specialization of the parent nodes, which can be the basis for automatic indexing of the rules on the basis of the taxonomic classification of events, conditions and actions.

The following relationship patterns are depicted:

- Edge $(E_{1i}(R_1), X); (X, E_{2j}(R_2))$, with $E_{1i} \rightarrow X; X \text{ AND } \rightarrow E_{2j}$, means that the event of rule R_1 is indirectly influencing the result of rule R_2 's event through rule X . The relationship consists of pairs. $E_{1i}(R_1) \text{ Relate to } \rightarrow X$ and $X \text{ Relate to } \rightarrow E_{2j}(R_2)$. By the transitivity relation property, Di Nola A (1991) $E_{1i}(R_1) \text{ Relate to } \rightarrow E_{2j}(R_2)$
- Edge $(E_{1i}(R_1), Y); (Y, C_{2j}(R_2))$, with $E_{1i} \rightarrow Y; Y \text{ AND } \rightarrow C_{2j}$, means that the event of rule R_1 is indirectly influencing the result of rule R_2 's condition through rule Y . The relationship consists of pairs. $E_{1i}(R_1) \text{ Relate to } \rightarrow Y$ and $Y \text{ Relate to } \rightarrow C_{2j}(R_2)$. By the transitivity relation property, Di Nola A (1991) $E_{1i}(R_1) \text{ Relate to } \rightarrow C_{2j}(R_2)$.
- Edge $(E_{1i}(R_1), Z); (Z, A_{2j}(R_2))$, with $E_{1i} \rightarrow Z; Z \text{ AND } \rightarrow A_{2j}$, means that the event of rule R_1 is indirectly causing change to rule R_2 's action through rule Z . The relationship consists of pairs $E_{1i}(R_1) \text{ Relate to } \rightarrow Z$ and $Z \text{ Relate to } \rightarrow A_{2j}(R_2)$. By transitivity relation property, Di Nola A (1991) $E_{1i}(R_1) \text{ Relate to } \rightarrow A_{2j}(R_2)$.

We can also depict the following possible combination of AND-relationship patterns:

- $E_{1i}(R_1) \xrightarrow{\text{Relate to}} E_{2j}(R_2) \wedge E_{1i}(R_1) \xrightarrow{\text{Relate to}} C_{2j}(R_2)$
- $E_{1i}(R_1) \xrightarrow{\text{Relate to}} C_{2j}(R_2) \wedge E_{1i}(R_1) \xrightarrow{\text{Relate to}} A_{2j}(R_2)$
- $E_{1i}(R_1) \xrightarrow{\text{Relate to}} E_{2j}(R_2) \wedge E_{1i}(R_1) \xrightarrow{\text{Relate to}} A_{2j}(R_2)$
- $E_{1i}(R_1) \xrightarrow{\text{Relate to}} E_{2j}(R_2) \wedge E_{1i}(R_1) \xrightarrow{\text{Relate to}} C_{2j}(R_2) \wedge E_{1i}(R_1) \xrightarrow{\text{Relate to}} A_{2j}(R_2)$

Rule's Indirect Condition-AND Graph

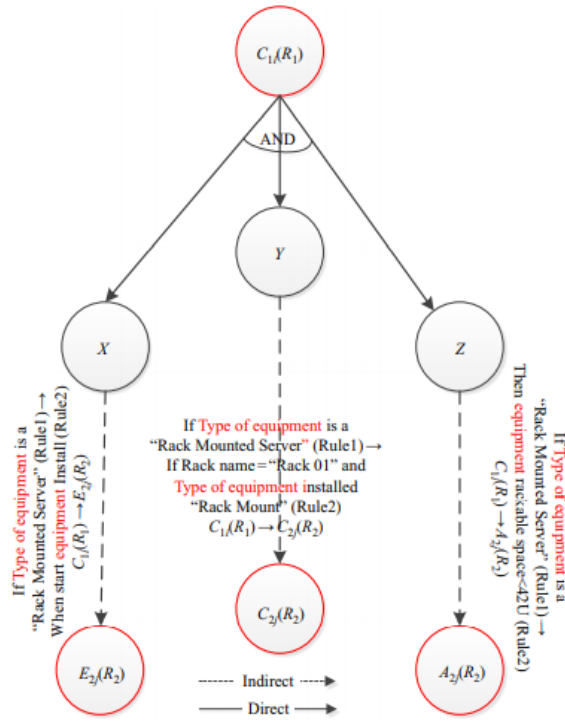


Figure 4.5.3.3. 2 Strong Indirect Condition-AND Graph

The Indirect Condition-AND graph is so named because the Condition component of one rule ($C_{1i}(R_1)$) indirectly causes another rule (R_2) to be checked when the condition of the related rule is checked. The source rule is linked to the target rule indirectly, via other nodes in the graph.

This type of dependency may be interpreted as a conditional variant of the indirect Action-AND dependency below. In both cases the rules actions can be executed upon a suitable event trigger but the Condition-AND related rules need an additional check of the condition which may not be necessary in the case of Action-AND dependency.

This interpretation allows bypassing some of the unnecessary checks to speed up the control-flow execution. Due to the non-strictly logical interpretation of such dependencies, however, the behavior of the business workflow management system will be implementation specific.

The following patterns are depicted:

- Edge ($C_{1i}(R_1)$ X); (X, $E_{2j}(R_2)$), with $C_{1i} \rightarrow X$; X AND $\rightarrow E_{2j}$ means that the condition of rule R_1 is indirectly influencing or triggering rule R_2 's event through rule X. The relationship consists of pairs $C_{1i}(R_1)$ *Relate to* $\rightarrow X$ and X *Relate to* $\rightarrow E_{2j}(R_2)$. By the transitivity relation property, Di Nola A (1991) $C_{1i}(R_1)$ *Relate to* $\rightarrow E_{2j}(R_2)$.
- Edge ($C_{1i}(R_1)$ Y); (Y, $C_{2j}(R_2)$), with $C_{1i} \rightarrow Y$; Y AND $\rightarrow C_{2j}$ means that the condition of rule R_1 is indirectly influencing the result of rule R_2 's condition through rule Y. The relationship consists of pairs $C_{1i}(R_1)$ *Relate to* $\rightarrow Y$ and Y *Relate to* $\rightarrow C_{2j}(R_2)$. By the transitivity relation property, Di Nola A (1991) $C_{1i}(R_1)$ *Relate to* $\rightarrow C_{2j}(R_2)$.
- Edge ($C_{1i}(R_1)$, Z); (Z, $A_{2j}(R_2)$), with $C_{1i} \rightarrow Z$; Z AND $\rightarrow A_{2j}$, means that the condition of rule R_1 is indirectly affecting rule R_2 's action through rule Z. The

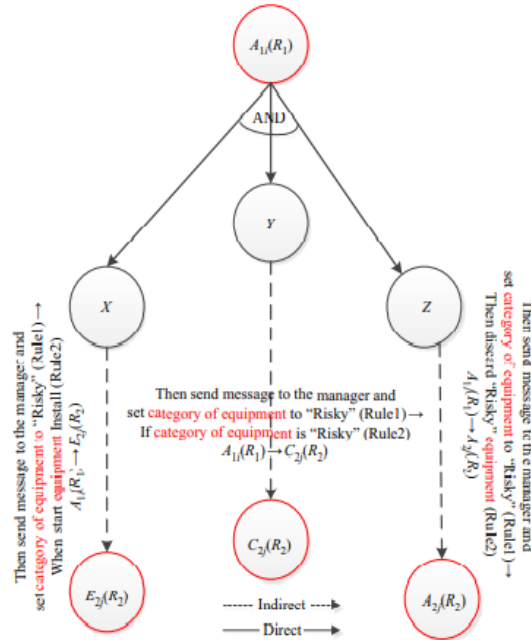
relationship consists of pairs $C_{1i}(R_1)$ *Relate to* $\rightarrow Z$ and Z *Relate to* $\rightarrow A_{2j}(R_2)$.

By transitivity relation property, Di Nola A (1991) $C_{1i}(R_1)$ *Relate to* $\rightarrow A_{2j}(R_2)$.

- We can also depict the following possible combination of AND-relationship patterns:

- $C_{1i}(R_1) \xrightarrow{\text{Relate to}} E_{2j}(R_2) \wedge C_{1i}(R_1) \xrightarrow{\text{Relate to}} C_{2j}(R_2)$
- $C_{1i}(R_1) \xrightarrow{\text{Relate to}} C_{2j}(R_2) \wedge C_{1i}(R_1) \xrightarrow{\text{Relate to}} A_{2j}(R_2)$
- $C_{1i}(R_1) \xrightarrow{\text{Relate to}} E_{2j}(R_2) \wedge C_{1i}(R_1) \xrightarrow{\text{Relate to}} A_{2j}(R_2)$
- $C_{1i}(R_1) \xrightarrow{\text{Relate to}} E_{2j}(R_2) \wedge C_{1i}(R_1) \xrightarrow{\text{Relate to}} C_{2j}(R_2) \wedge C_{1i}(R_1) \xrightarrow{\text{Relate to}} A_{2j}(R_2)$

Rule's Indirect Action-AND Graph



The Indirect Action-AND graph is so named because the Action component of one rule ($A_{1i}(R_1)$) indirectly causes another rule (R_2) to be checked. The source rule is linked to the target rule or rules via other rules. The event of R_2 is invoked when R_1 's action relates R_2 's event, condition and action. This causes R_1 and R_2 to execute consecutively.

As explained earlier, this type of dependency is open to interpretation. An alternative to the intended interpretation introduced earlier (unconditional action execution) could be given in terms of actions only. For example, rules linked using such a dependency may need to completely skip their actions in the case of previous execution of the actions of related rules. Since both alternatives are implementation specific, they will be tested at the implementation phase before fixing the intended interpretation.

Figure 4.5.3.3. 3 Strong Indirect Action-AND Graph

The above tree represents indirect AND dependency where nodes are indirectly connected to the root node/rule $A_{1i}(R_1)$ through rules (X, Y). The following patterns are depicted:

- The execution of $A_{1i}(R_1)$ indirectly triggers the execution of $E_{2j}(R_2)$ through additional events, conditions or actions of the X rule. The relationship consists of pairs $A_{1i}(R_1)$ *Relate to* $\rightarrow X$ and X *Relate to* $\rightarrow E_{2j}(R_2)$. By the transitivity relation property, Di Nola A (1991) $A_{1i}(R_1)$ *Relate to* $\rightarrow E_{2j}(R_2)$.

- The execution of $A_{1i}(R_1)$ indirectly triggers the execution of $C_{2j}(R_2)$ through additional events, conditions or actions from Y rule. The relationship consists of the pairs $A_{1i}(R_1) \text{ Relate to } \rightarrow Y$ and $Y \text{ Relate to } \rightarrow C_{2j}(R_2)$. By the transitivity relation property, Di Nola A (1991) $A_{1i}(R_1) \text{ Relate to } \rightarrow C_{2j}(R_2)$.
- We can also devise the following possible combination of AND-relationship patterns:

$$\circ \quad A_{1i}(R_1) \xrightarrow{\text{Relate to}} E_{2j}(R_2) \wedge A_{1i}(R_1) \xrightarrow{\text{Relate to}} C_{2j}(R_2)$$

The “Indirect AND Dependency” pattern (Figure 4.5.3.3.1 - 4.5.3.3.3) is such that rule nodes flow into two or more edges; the edges proceed and merge into a rule node where a connection or relationship is to be established; hence they are indirectly connected through intermediate nodes. This dependency pattern is based on indirect AND connections between nodes or rules on the same path. There must be at least one indirect rule from the nodes with an AND connection.

4.5.3.4 Indirect OR Dependency patterns

The Indirect OR dependencies between rules can be introduced similarly to the indirect AND dependencies. They are also open to interpretation and since the intended meaning largely depends on the implementation, we will leave this for that stage.

4.5.4 Business Rules Dependency Patterns

As seen in the preceding sections, relationships between the rules are defined by directly linking objects, objects properties and indirectly relating the quantitative and qualitative measures of their characteristics. Although the relationship patterns are different in terms of their semantics, they also bear some similarities in terms of the appearance of different components of the rules in the structures representing their use in real time. For example, in Figure 4.5.4 we can identify the following patterns of dependency between rules: rules on the same path (also known as chained rules), rules on the same level (alternative rules), rules with the same parents (alternative chains), directly related rules and indirectly related

rules. The AND-OR Tree (Figure 4.5.4) combines different relationship patterns presented earlier using dependency graphs. Relationship patterns in the AND-OR Tree can be classified as follows:

- Neighbour/Precedence dependencies: The relationship between the rules within the tree link successor and predecessor nodes. Such relationships can be defined within the same root; parent and child rule nodes are related.
- Level dependencies: Rules at the same level of precedence are related. The relationship between the rules can be defined within the same level on which they appear within the tree. But such relationships can form multilevel dependencies as well. Furthermore, this pattern can form an AND-OR dependency subtree.
- Path dependencies: Rules on the same paths within the tree are related from the top node to the leaf nodes, forming a transitive pathway.
- Direct node dependencies: Rules without a common root can be related. The relationship can be defined solely based on individual rule properties in relation with other rules. Such relationships may result in a non-tree structure of dependencies and can be inefficient for a large set of rules, since every node's relationships is to be checked. However, we can argue that this is still a tractable relationship since the dependence can be formulated by means of class dependencies.
- Indirect node dependencies: The dependency is established through intermediate nodes on the same root node. Such dependencies may exist although their handling can be complicated.

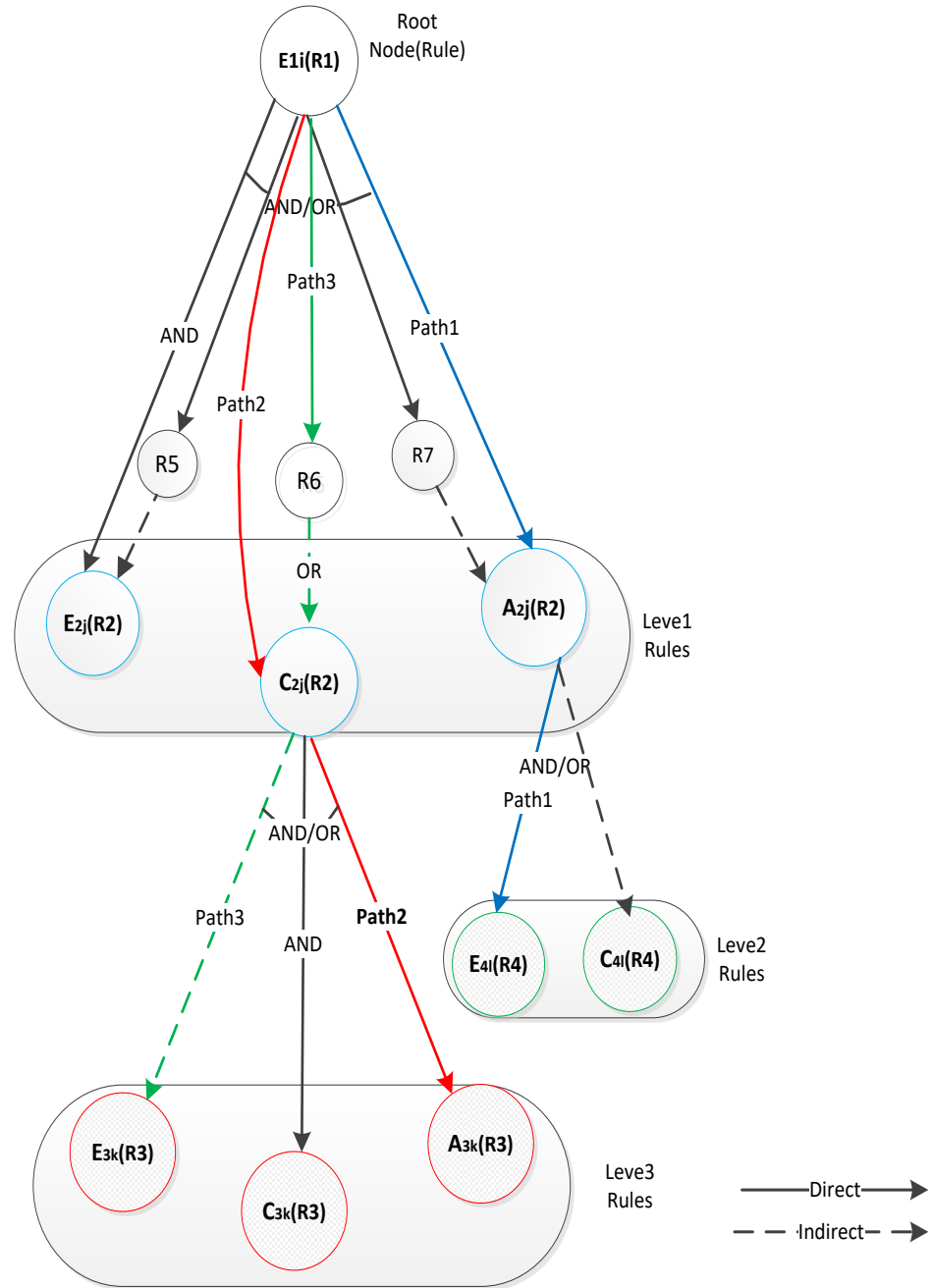


Figure 4.5. 4 AND-OR Graph with Dependency Patterns

4.5.5 Application of AND-OR Graphs

In workflows, each process object captures the function to be carried out; however, the behaviour of the workflow is controlled by the ECA rules. The proposed model advocates the use of AND-OR graphs as a solution for managing the changing behaviour of the

workflow. To demonstrate how AND-OR graphs (Figures 4.5.3.1.1 to 4.5.3.3.3) can be used to control processes, an example presented in Figure 4.4 is used. Only an Event-AND graph is illustrated here.

Now, consider Initiation and Execution Rules presented in Figure 4.4, also summarised below:

Rule1 (Initiation Rule)

Event: When Notify New Install Request and New Equipment has been ordered

Rule2 (Execution Rule)

Event: When Install New Equipment

Condition: If rack utilization is greater than 80% and Install Request status = Cancelled

Action: Send Message to Manager; Close Install Request

In this example event Notify Equipment(Rule1) relates to Event: Install Equipment (Rule2) - $E1_i(R1) \text{ Relate } \rightarrow E2_j(R2)$, event Notify Install Request(Rule1) relates to Condition: Install Request (Rule2) - $E1_i(R1) \text{ Relate } \rightarrow C2_j(R2)$ and event Notify Install Request(Rule1) relates to Action: Install Request (Rule2) - $E1_i(R1) \text{ Relate } \rightarrow A2_j(R2)$. The Event-AND graph is constructed using patterns: $E1_i(R1) \text{ Relate } \rightarrow E2_j(R2)$ AND $E1_i(R1) \text{ Relate } \rightarrow C2_j(R2)$ AND $E1_i(R1) \text{ Relate } \rightarrow A2_j(R2)$. Rule1's Event component relates to Rule2's Event and Condition and Action (ECA) components via common properties. Now when Rule1 is invoked the following happens:

=> Rule 2's Rule2's Event is ignored

=> Rule 2's Condition is checked

=> Rule's Action is executed

=> Process "Manage Rack Space Availability" will be skipped, so the workflow will flow from "Create Request to install new server" to "Order New Rack" or "Send Messages" instead of "Create Request to install new server" to "Manage Rack Space Availability" to "Order New Rack" or "Send Messages".

Changing the properties of ECA components can influence or affect the flow of processes. For example, changing components of Rule2 or removing Rule2 may cause the workflow

to flow from “Create Request to install new server” process to “Manage Rack Space Availability” process. By constructing AND-OR graphs, we can identify rule relationships and control the flow of processes in workflows. Figure 4.5.5 illustrates an AND-OR graph with various rules linked to processes. Rules on the same graph patterns are connected.

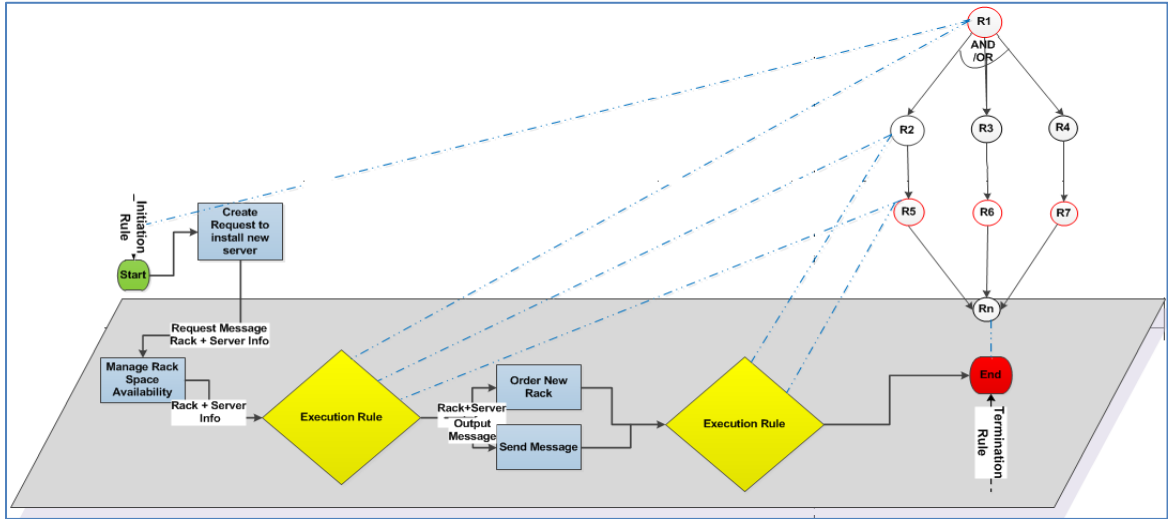


Figure 4.5. 5 Business Rule Dependency Graph Control Processes

4.6 Summary

This Chapter presents the formal theories of the proposed model. One important aspect being the specification of the Two-Levels Architecture, which is a representation of business rule control and business process levels. Essentially, the architecture provides an integration mechanism to allow business rules and Metarules to configure workflows. The other equally important aspect is the formalization of business rules and components using the AND-OR graphs. The AND-OR graph is considered as a set of business rule dependency patterns that have similar behaviour and shapes. The dependencies between business rules are formulated using the objects and their properties which are parameters of the business rule structure (Event, Condition, and Action). Since the coupling of rule components is loose in AND-OR graphs, changes to business rule components can be carried out separately. This becomes very important when there are many processes and the business rule changes are frequent. The Chapter concludes with an example to allow BPM professionals and academics to interpret business rules and apply the proposed model theories to control and configure workflow.

5. Software Architecture, Metarules and Indexing

The sketch of the architecture is presented to outline the main model components and their principal interactions. Section 5.1 presents the software architecture of a system which can implement the framework. The Metarule construct is explained in section 5.2 to further enrich the software architecture. Section 5.3 discusses our business rules indexing approach using various dependency patterns.

5.1 Software Architecture

In Figure 5.1, the ECA Model Adaptor component is responsible for implementing business rule formal definitions as discussed in Chapter 4. The business rule designer (editor) provides an environment to allow the user to enter, delete and update business rule components. Currently, users are responsible for authoring business rules through the business rule designer one at a time. Obviously in future, it would be ideal to provide tools such as decision tables in Drools for mass importing of business rules for production deployment. The business rule designer and ECA Model Adaptor form the main components of the system architecture. The adaptation layer through APIs provides an interface to communicate with external business rules and workflow management systems. At the time of writing this thesis, the prototype is only linked to JBoss Drools to store (rule repository) and execute business rules in real time. Furthermore, through APIs, workflow is executed based on the business rules stored in the rule repository. In JBoss Drools [52], business rules are stored in the production memory.

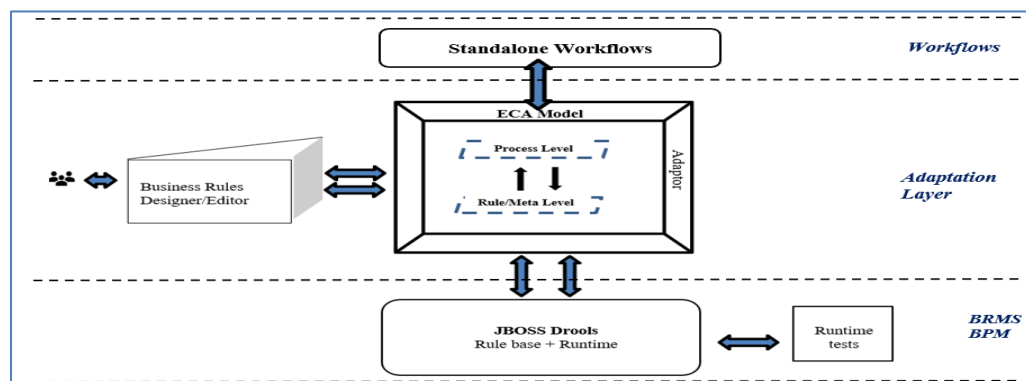


Figure 5.1 Software Architecture Diagram

The ECA Model Adaptor module consists of Java files required during business rules and workflow compilation and execution. In most cases, users will simply include all dependency files at runtime then connect to Drools rule management system. To allow flexibility for integration with other business rules and process systems, specific system APIs will need to be created. The core files are quite compact and only require few kilobytes for JAR files. The runtime performance may arise when there is a huge amount of related business rules that require changes. However, the dependency patterns indexing algorithm (Section 5.3) is written to provide faster business rule access and execution times. Using the ECA components and dependencies with correct data structures and indexes, we should be able to determine which bits need optimization.

5.2 Metarules

The metarule concept provides the ability to monitor and change business rules and indexes at runtime. Creating Metarules is very similar to creating normal business rules in JBoss Drools [52]. Let us consider Code Snippet 5.2.1 below representing a business rule using Drools Rule Language (DRL).

```
1 package org.kanana;
2 import org.kanana.Rack;
3 import org.kanana.Request;
4 import org.kanana.RuleName;
5 import function org.kanana.Utility.help;
6
7 rule "R3"
8 when
9     Request(Type == 'Install')
10    Rack(Utilization >= '2000')
11 then
12     help(drools,"Drools runtime Info... ");
13     System.out.println("Action after rule triggered: " + "Process(Name == 'P5-Managed DC Space')");
14 end
```

Code Snippet 5.2. 1 Business Rule using DRL Syntax

A Metarule is applied on the above business rule in such a way that a message is created to inform the DC Manager about installation of equipment whenever business rule “R3” is invoked and request type equals to install, as shown in Code Snippet 5.2.2 below.

```

1 package org.kanana;
2 import org.kanana.Rack;
3 import org.kanana.Request;
4 import org.kanana.RuleName;
5 import org.kanana.MessageFact;
6 import function org.kanana.Utility.help;
7
8
9 meta-rule "Meta_R3"
10 when
11     Rule(name=="R3")
12     Request(Type == 'Install')
13 then
14     help(drools,"Drools runtime Info... ");
15     System.out.println("Message(Description =='Info DC Manager')");
16 end

```

Code Snippet 5.2. 2 Metarule using DRL Syntax

Using Metarules, it is possible to add, remove or update existing business rule Events, Conditions and Actions by invoking methods to add, remove and update while passing the name of the business rule. For example, in Code Snippet 5.2.1 above to update the Event “Request (Type = ‘install’) to Request (Status = ‘open’)” is achieved through invoking method update Event on r while passing the new Event as an argument (Code Snippet 5.2.3)

```

1 package org.kanana;
2 import org.kanana.Request;
3 import org.kanana.RuleName;
4
5 import function org.kanana.Utility.help;
6
7 meta-rule "Meta_R4"
8 when
9     rN: Rule(name=="R3")
10
11 then
12     rN.updateEvent(Request(Status == 'open'))
13 end

```

Code Snippet 5.2. 3 Metarule updating an existing event

In similar fashion, the business rule indexes can be created and modified. Using Metarules, Code Snippet 5.2.4 presents index “Index_R1” is created to monitor any business rules in Node-based dependency pattern called ‘Rack’, and index “Index_R2” is created to monitor any business rules in a path-based dependency pattern. Notice that two classes, ‘NodeBasedPattern’ and ‘PathBasedPattern’ are called for implementation of relevant indexes.

```

1 package org.kanana;
2 import org.kanana.Rack;
3 import org.kanana.RuleName;
4 import org.kanana.Event;
5 import org.kanana.Condition;
6 import org.kanana.NodeBasedPattern;
7 import org.kanana.PathBasedPattern;
8
9
10 meta-rule "Index_R1"
11 when
12     Condition(name=="Rack")
13
14 then
15     NodeBasedPattern(IndexID = 0001);
16 end
17
18 meta-rule "Index_R2"
19 when
20     Rule(name in ('Event(nR1)', 'Condition(R2)'))
21
22 then
23     PathBasedPattern(IndexID = 0002);
24 end
25

```

Code Snippet 5.2. 4 Index creation using Metarule

As you can see like normal business rules, Metarules are also made up of a name, event, condition and action components. These can be directly added, deleted and modified using our business rule designer editor. It is important to understand that both ‘NodeBasedPattern’ and ‘PathBasedPattern’ classes implement an interface that has constructor classes and other methods i.e. add, remove and modify indexes. For example, NodeBasedPattern indexes are added by invoking the method (NodeBasedPattern.add) or use a constructor as shown in Code Snippet 5.2.4. Note, business rules R1 and R2 are referenced in Index_R2. This is because both are on the same path dependency.

5.3 Indexing of Business Rules

This Section provides an answer to research questions regarding the issue of how efficiently the underlying business rules can be retrieved. In computer programming, an index is a key that is used to point or order unsorted records for easy access [30, 117]. The key is typically used to reference records, for example a content page of a book provides indexes (table of content) to individual Chapters.

Comparable to the data in the database management systems, business rules in rule management systems are faced with the similar problems of storing and maintaining large volumes of information. Fields like financing, banking and insurance have a large set of business rules and processing them is a major task [122]. The execution performance on business rule applications is influenced by the number of business rules to be searched and processed. Agreeably, the storage and time complexities of business rule creation and execution are commonly connected to the number of business rules in rule management systems. As the number of business rules increase so does the execution time. The worse situation is when there are multiple relationships between business rules and children dependencies. What could eventually happen is that a business rule may need to be changed and propagate its changes to other business rules. The affluence of analytics is important because as the number of stored business rules increase, it becomes difficult to find and update business rules.

Indexing business rules is a way to optimize performance of a rule management system by minimizing the number of accesses required as business rules are searched, inserted, deleted and updated [58]. Such performance optimization is done by providing quick pointers (locators) of where the queried business rule components are. In our proposed model, an index is defined to representing a dependency pattern (Path dependency, Direct-Node dependency, Level dependency, Neighbour dependency, Indirect node dependency). All dependency patterns must first be found and then indexes are created for each pattern. Business rule dependency queries including change propagations are executed using such indexes. It is worth mentioning here, that for dependency patterns such as Direct-Node and Neighbour dependency patterns where information to be fetched or updated is situated

conveniently, it would be faster or simpler to query using the actual AND-OR graph nodes containing the business rule components. In this case, the well-known algorithms such as Depth-First Search and Breadth-First Search [108] are incorporated to traverse through the graphs. To be more precise, there is a level of the AND-OR graph, where querying at logical layers (Figure 5.3.1) stops being more effective.

Due to time constraints, the following discussion is limited to a method of indexing business rules using Path dependency patterns. The discussion is divided into two areas. First, the index data structure is introduced. The index structure is attuned so that using indexes speeds up the querying process. A structural index aids in evaluation of complex patterns by avoiding unnecessary retrieval operations. Second, a method for indexing Path dependency patterns is discussed in more detail.

5.3.1 Index Structure

The underlying index structure is built upon the well known 'graph' data structure. Business rules with similar characteristics or patterns are grouped together and indexes are constructed from the groups using the graph data structure. The graph structure, which forms the logical layer consists of two important levels, the root and dependency patterns levels. Figure 5.3.1 shows how the logical (containing indexes) and physical (containing actual business rule components) layers are linked. The black coloured nodes represent indexes, whereas the grey coloured nodes represent the actual business rules. Different from actual business rule components, the indexes are created using Metarules discussed in section 5.2.

For illustration, consider a simple set of business rules based on the workflow presented in Figure 4 in Chapter 4. Business rules are applied to ensure that there is no overload of equipment in racks and correct types of equipment are installed in racks. Business rules R_1 , R_2 , R_3 , R_4 and R_5 are managed and various path dependency patterns are drawn as shown in Figure 6.1. The root index node provides a link to dependency pattern indexes (**Pattern**

Index 1 and Pattern Index 2). The dependency pattern indexes point to actual matches for the graph dependency patterns ($E(R1) \rightarrow C(R2)$; $A(R2) \rightarrow C(R4)$ and $E(R1) \rightarrow C(R3)$).

Note that there is a difference between dependency pattern indexes and root indexes. Whereas dependency pattern indexes are formed by actual business rule components relationships, the root index points to dependency pattern indexes. The root index provides a high-level view of the dependency pattern indexes (Pattern Index 1 and Pattern Index 2 in Figure 5.3.1).

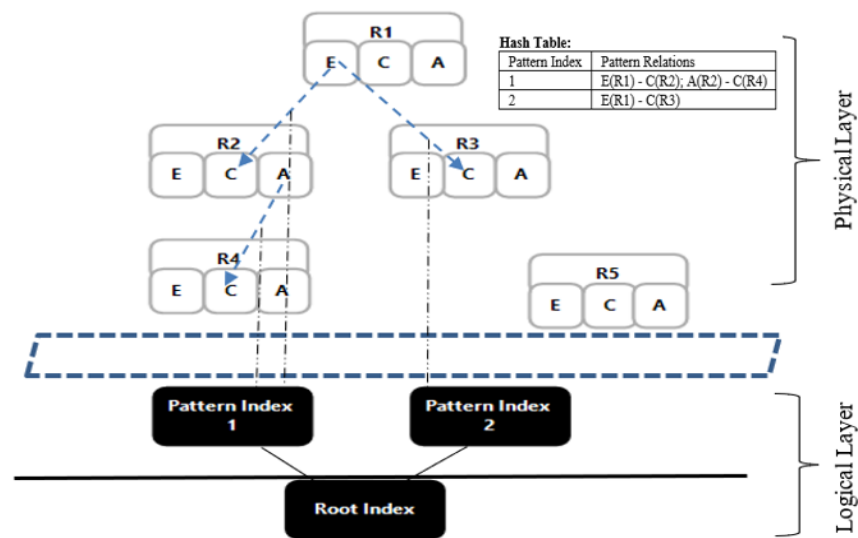


Figure 5.3. 1 Graph Dependency Index Structure

5.3.2 Path Dependency Pattern Indexing

A business rule repository is scanned to find graph dependency patterns that match a given path pattern. Then for each path that share the same business rule component nodes, a single node (Pattern Index) representing the path dependency pattern is created. Business rule component nodes and relationships within a path are linked to the appropriate Pattern Index. For each path dependency, an identifier is created. The identifier is created by combining unique keys of relationships that form a path dependency pattern. The root index is also created to complete the graph structure. An index name and a path dependency pattern that the index is built for are stored as properties within the root index. When a

business rule is added or updated in a dependency path, it is propagated to all its related business rule nodes. This means iterating over all the graph's business rule component nodes and propagating the change. This is a time-consuming exercise, especially if there are many business rule dependencies. However, by indexing the path, the change is propagated only to related business rule nodes (components) connected through indexes. Imagine the following business rule is fired when rack utilization is greater than rack capacity. The rack capacity threshold limit varies depending on a data centre location. Let us say, there is one business rule for different locations (Dar es Salaam, London and New York). There is also an additional business rule (R04) that checks if the data centre location is not London as shown in Table 5.3.2.

<i>If Datacentre Location equal to Dar es salaam, then Rack Capacity threshold equals to 45 units</i>	R01		<i>If Location (Data Centre) == 'DAR'</i>	<i>Set Capacity (Rack) = 45</i>
<i>If Datacentre Location equal to London, then Rack Capacity threshold equals to 42 units</i>	R02		<i>If Location (Data Centre) == 'LON'</i>	<i>Set Capacity (Rack) = 42</i>
<i>If Datacentre Location equal to New York, then Rack Capacity threshold equals to 41 units</i>	R03		<i>If Location (Data Centre) == 'NYC'</i>	<i>Set Capacity (Rack) = 41</i>
<i>If Datacentre Location not equal to London, then Rack Capacity threshold equals to 41 units</i>	R04		<i>If Location (Data Centre) <> 'LON'</i>	<i>Set Capacity (Rack) = 47</i>
<i>When equipment install request triggered, if rack utilization is greater than the rack space capacity, then set the Rack is full</i>	R05	<i>When Request Type (Equipment) == 'Install'</i>	<i>If Utilization (Rack) >= Capacity (Rack)</i>	<i>Set Space (Rack) == 'is full'</i>

Table 5.3.2: Business rule – rack utilization exceeds by data centre location

The above business rules will generate the following AND-OR Graph. The path patterns are highlighted in different colours.

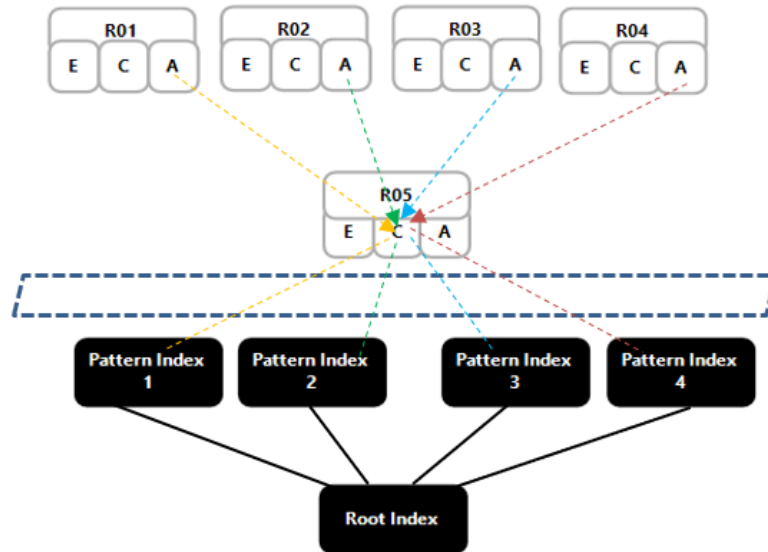


Figure 5.3. 2 Path Dependency Indexing Graph

The figure 5.3.2 above shows business rules with four Path dependency indexes when a change is propagated through the AND-OR graph from above. Let's say a data centre location is set to "NYC", this fact will propagate to business rule R03 and R04 completely avoiding business rules R01 and R02. With regards to computation complexity, the procedural programming (Code Snippet 5.3.2.1), would have to evaluate all conditions until it finds the match branches. The complexity is $O(n)$, where n is the number of if branches, the program has to check. However, by indexing the graph path, this effectively translates to a complexity of $O(1)$.

```
public class ECA_NoIndexing {
    public static void main (String args []) {
        if (datacentre.getLocationName().equals("DAR")) {
            //execute this if location is Dar es salaam
            ...
        }
        else if (datacentre. getLocationName().equals("LON")) {
            //execute this if location is London
            ...
        }
        else if (datacentre. getLocationName().equals("NYC")) {
            //execute this if location is New York
            ...
        }
        ...
    }
}
```

Code Snippet 5.3.2. 1 Procedural programming (Java Syntax)

5.4 Summary

This Chapter presented the ECA Model's system architecture of the framework, followed by discussion on implementation of the Metarule concept to support runtime modification of business rules. Moreover, the Chapter discussed the proposed model's indexing mechanism. The mechanism contributes to the implementation of an efficient indexing mechanism for path dependency patterns in an AND-OR graph. The graph data structure is introduced to hold a complete list of path dependency patterns within the graph. The index nodes within the graph data structure have a direct access to nodes that form path dependency patterns. Using a graph data structure to index business rules brings both advantages and disadvantages. For example, the business rule index can be queried in the same way actual business rule can be. So, querying indexes are done by using DRL or APIs that are provided by a specific graph engine. Pattern Indexes point directly to graph dependency pattern units via relationships created through unique identifiers of the dependency path. Thus, it is easy to find which business rule component node is part of a specific dependency pattern. This is very useful when updating indexes and propagating changes. The obvious disadvantage of the graph data structure is that it requires more storage space for larger graphs: two levels of the graph need to be considered for each index.

6. Change Propagation and Adaptation Algorithms

This Chapter presents important algorithms to provide a systematic approach for creation and modification of business rules and dependencies. It introduces two relevant algorithmic categories (algorithm for Business Rules Change Propagation and algorithm for Business Rules Adaptation in Workflows) to compute business rules change propagation and adaptation problems identified in Chapter 2. The algorithms are formulated using business rules components (ECA - three basic building blocks), dependency patterns of the AND-OR graphs, Metarules, Pattern Indexes as described in Chapter 4 and 5.

6.1 *Business Rules Change Propagation*

According to [115], the basic understanding of the term “propagation” in computer science means an action in which an interactive system adjusts its behaviour based on the change of information. The concept of propagation is also used in business rule management systems where the changes can have different granularity - a rule component, a separate rule and a whole set of related rules; whatever the change is, it can be translated in terms of the index patterns we created earlier: a change of a node, path or a whole subtree. This becomes the starting point of the algorithms for propagation and adaptation. One or more changes can be used to the business rules to update other business rules to new requirements. However, we cannot simply apply a change to one business rule component, because a change can overlap to more than one business rules. This may lead to dependency change impacts between multiple business rules. Some changes are predictable, but others may occur due to unexpected propagations on other parts of the business rules. Business rules may dynamically update multiple business rules every time they are processed by the business rules management systems. When there is a business rule change and that change needs to be propagated across a volume of related business rules, the constant scanning through rules can be costly and inefficient in terms of performance in execution of the business rules in an application. A vital challenge is to find ways of propagating the changes in an efficient manner. We appreciate that there are number of rule systems that deal with change propagation, but our method differs from the existing approaches. Our approach deals with the change propagation at business rule components level using

dependency graphs and patterns. Through dependency graphs, the relationships between rules components are unfolded based on information available at the time of change.

The propagation of the changes must reach the limits of the scope which is determined by the index tree. A changed business rule component in the graph must be propagated to all business rule component nodes that directly or indirectly depend on it. This means the ordering of all nodes relative to the changed node matters. Hence, the initial step in change propagation is to index the graph to provide some orderings to help with sorting business rules. The business rule nodes are indexed as described in Chapter 5. Business rule nodes on the same index pattern to the changed node are potentially related. Each gets examined to see whether their value really depends on the changed business rule node. The change gets propagated to the final list of business rule nodes that depend upon the changed node. Typically, business rules update would take efficient estimation of θn^2 , where n is the number of business rules. For example, in a real industrial workflow scenario with about 200K business rules, the change can take around 30 minutes, which is simply unacceptable if the process needs to flow as quickly as possible. A change of the business rule components may not only affect the business rules systems but also cause the workflow to behave differently to what was initially intended. Henceforth, an approach to speedy business rules change propagation at design time and runtime is required.

In a workflow, it is important to also consider that the outcome of a change propagation may result in new paths (flows) being created or extended, paths splits, new business rules being added, or existing rules deleted (no longer valid), etc. The modification, deletion or addition of a business rule component to the AND-OR graphs will typically lead to the creation of new business rule component nodes, deletion of the existing business rule component nodes, creation of new business rule component nodes relationships and deletion of existing business rule component nodes relationships. Section 6.2 presents business rule change propagation based on dependency patterns.

6.2 *Change Propagation via Dependency Patterns*

The change operations in an AND-OR graph can be collapsed into three: an insertion, deletion and update of business rule components and relationships, whereas the effect of these changes can be propagated by mapping the operations to operations over the index tree. The change operations provide the ability to create, destroy or convert business rule components and relationships in business rule systems or applications. This research considers the following operation types:

- Insertion of a rule component to an AND-OR graph
- Removal of a rule component from an AND-OR graph
- Modification (Update) of a rule component in an AND-OR graph

When there is a large computational of business rules component dependencies to deal with in an AND-OR graph, one complexity to consider is how to manage the mentioned change operations. One of the many reasons for defining dependency patterns in an AND-OR graph is to help to identify the relationships between business rule components. This allows us to determine which business rule components must be revised in case of change. All affected business rule components in a dependency pattern must be easily revised to ensure correct activation. Once dependency patterns are identified and defined, a change can be propagated through representation dependency patterns. The following five change propagation patterns form the basis of our change propagation algorithm. Only the Path Dependency Propagation pattern (section 6.2.1) is discussed in detail. Other dependency propagation patterns are briefly explained (section 6.2.2 – 6.2.5), leave the implementation aspects for future work.

6.2.1 Path Dependency Propagation

This refers to a chain of business rules affected by a change. The possibility to determine all change propagation paths between business rule components can provide valuable information, for example, removal of duplicate and inconsistent business rule components and relations as well as prediction of future business rule components change propagations to prevent unwanted change in the future. This approach is systematic, rather than based

on limited human knowledge of examining few paths. The reason behind business rule component change propagation in a path dependency pattern is to easily track and propagate the change in the case where business rule components are connected sequentially. The advantage of this is the ability to focus only on paths that need changing as well as handle batch path changes at once.

As we have seen in Chapter 4, construction of AND-OR graph starts with a root business rule component. All other business rules components are drawn as children of the root business rule component. An AND-OR graph is made up of one or more paths from the root business rule component. In an AND-OR graph, the path dependency pattern shows relationships between business rule components (source business rule component) and other business rule components (target business rule component) via properties and functionalities. The target business rule component of the first business rule component becomes the source business rule component of a second business rule component and so forth. These business rule components have a relationship between them. If the completion of the source business rule component (i.e. action) requires the completion of the target business rule component (i.e. event). The business rules components on the same path (source/upstream or target/downstream) to component 'n' are potentially dependent on component 'n'. Each business rule component in the path gets examined to check whether their value really depends on component n's properties/functionality or not. Having the final list of paths that depend upon component 'n', the changes get propagated to them, leaving the component at each path with updated values.

For illustration, consider a simple set of business rules based on the workflow presented in Figure 4 in Chapter 4. Business rules are applied to ensure that there is no overload of equipment in racks and correct types of equipment are installed in racks. Business rules R₁, R₂, R₃, R₄ and R₅ are managed and various path dependency patterns are built as shown in Figure 6.2.1 Business rules R₂ and R₃ are directly dependant on R₁. Rule R₄ is directly dependant on R₂. Rule R₅ is directly dependant on Rule R₃. Whereas, Rule R₄ is indirectly dependent on Rules R₁. The relationships exist under the conditions that the occurrence of event property "Rack Space" in rule R₁ forces rules R₂ and R₃ (with equivalent properties)

to be invoked. Furthermore, the occurrence of event property “Equipment Category” in R3 forces rule R5 to be invoked. There is also an indirect relationship from rule R1 to rule R4 via rule R2.

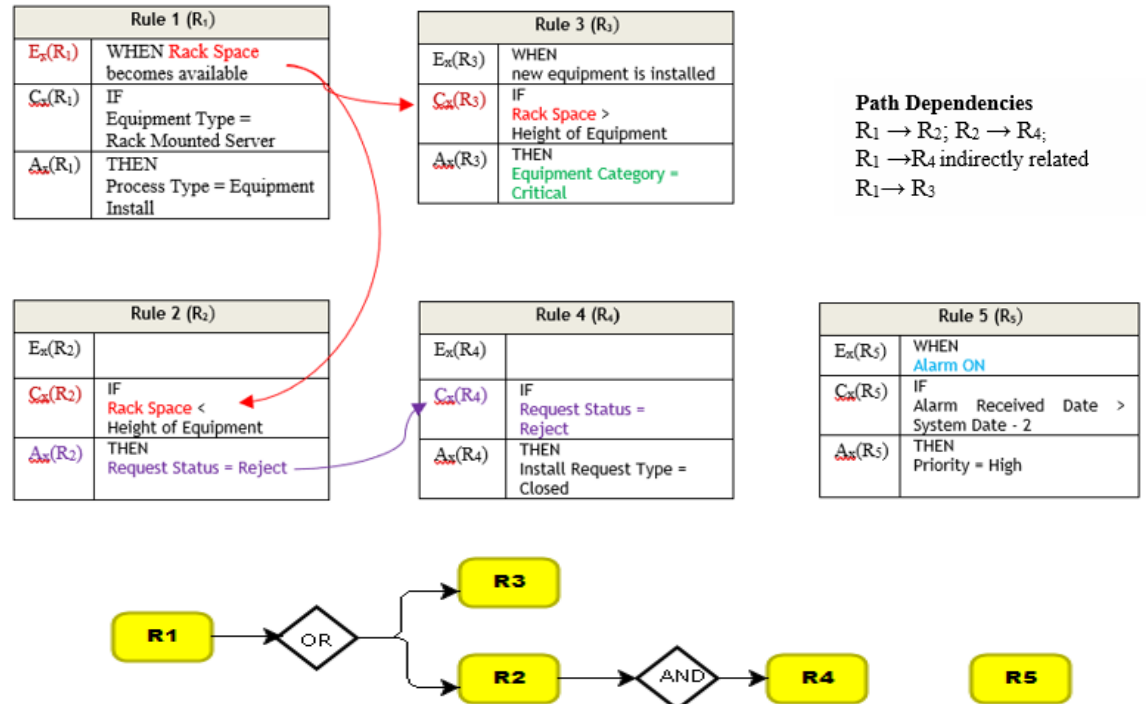


Figure 6.2. 1 Business Rules Path Dependencies

Whenever the underlying business rule component nodes change, the approach would be dynamically accessing the path using indexes and invalidating the next related applicable business rule component then propagate the changes before firing the business rules again. The outgoing path containing the chained business rules is checked and updated. The path always keeps its structure and content being updated, meaning that if a change is applied to the content or the structure of a path in an AND-OR graph, it automatically gets propagated to all dependent business rule component nodes that may get affected.

If a new business rule is inserted to the target business rules system, then certain parts of the currently legal business rule component relationships may become outdated. Figure 6.2.2 shows the changes in the path dependency graph when the business rule R₆ is inserted. All the changes happen in specific paths of the graph, some paths are not affected by the

new inserted business rule. Before R_6 is inserted, the business rule R_5 had no dependencies; As soon as R_6 is inserted, R_6 forms a direct dependency on R_5 and R_3 . R_6 has an indirect dependency on rule R_1 . The rule insertion proceeds in three stages:

- (i) By using path indexes, find the affected dependency paths because of the impact of a new inserted business rule,
- (ii) Insert the new business rule (example R_6) to the path and
- (iii) Define and modify dependencies to the affected paths (example R_1 , R_3 , R_5 , R_6).

While finding the affected paths, the algorithm recursively traverses through the dependency graph. It checks if the new business rule component ($C_x(R_6)$) intersects with any next business rule components in the dependency path. It inserts the new business rule or node then updates the intersecting path by collecting and adding it to the set of affected paths. The recursion proceeds by exploring the next path. This way, it ends up only exploring the relevant paths in the dependency graph. The business rule insertion results in adding a new business rule, adding new paths for the new business rule, and modifying existing dependencies.

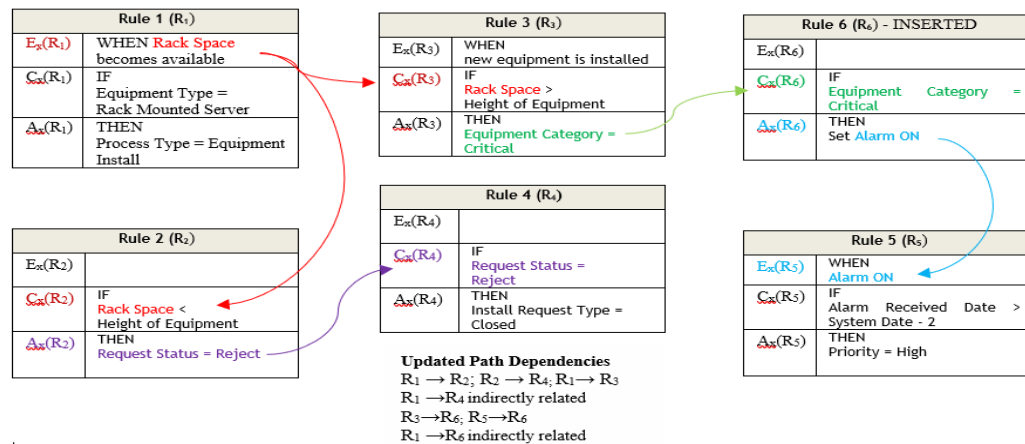


Figure 6.2. 2 Business Rule 6 Inserted

If a business rule component from the dependency graph is removed, all dependencies on the underlying deleted rule component become out-dated in which case the dependant edges will no longer be required. Figure 6.2.3 shows the changes in the path dependency graph when the rule R_3 is deleted. All the changes happen in a specific path of the dependency graph, some of the paths are not affected by the removed business rule. After R_3 is deleted, rule R_6 is no longer connected to R_1 . The deletion of a rule also consists of three stages: (i) Find the affected dependency paths because of the impact of a deleted business rule, (example $R_1, R_3, R_3, R_6, R_5, R_6$). (ii) Delete the business rule component (example R_3) from the path and (iii) Define and modify dependencies to the affected paths, for example $R_1 \rightarrow R_3; R_3 \rightarrow R_6$ paths will be removed.

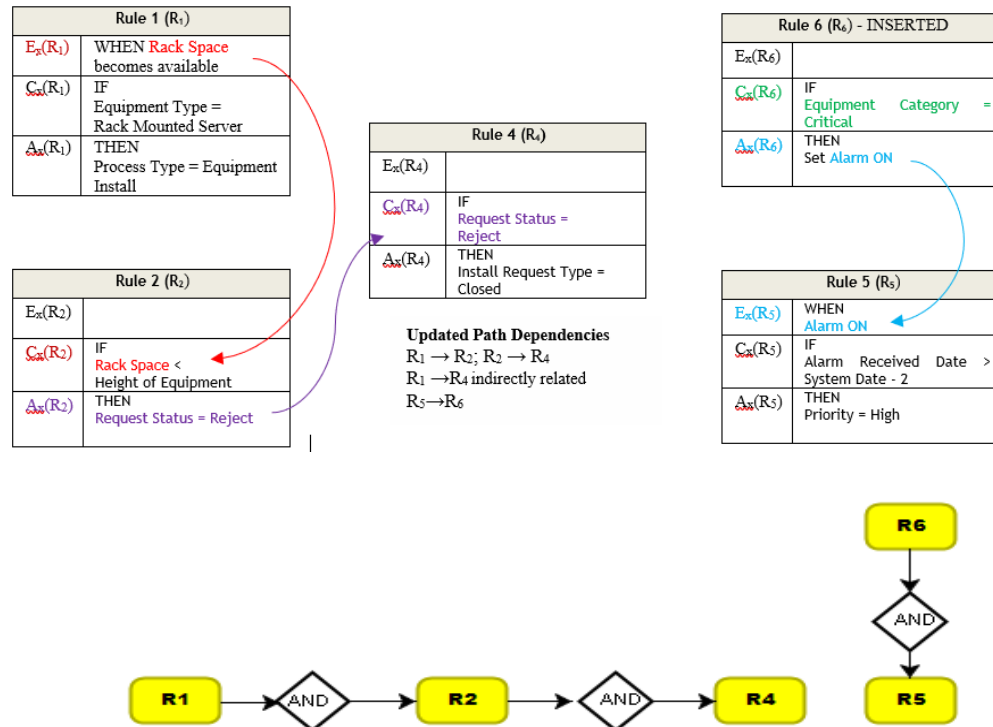


Figure 6.2. 3 Business Rule 3 Deleted

If a business rule component from the dependency graph is updated all dependencies on the underlying updated business rule component would need to be updated accordingly. All the changes happen in a specific path of the dependency graph, some of the paths are not affected by the updated business rule. Figure 6.2.4 shows the changes in the path dependency graph when the business rule R_3 is updated. The update of a business rule

component consists of three stages: (i) Find the affected dependency paths because of the impact of an updated rule component (ii) Update the rule component to the path and (iii) Define and modify dependencies to the affected paths. When an update happens on a node n, after the value of the node is updated, the change gets propagated through the graph to the nodes that depend on n, so that they can reevaluate their value based on updated inputs.

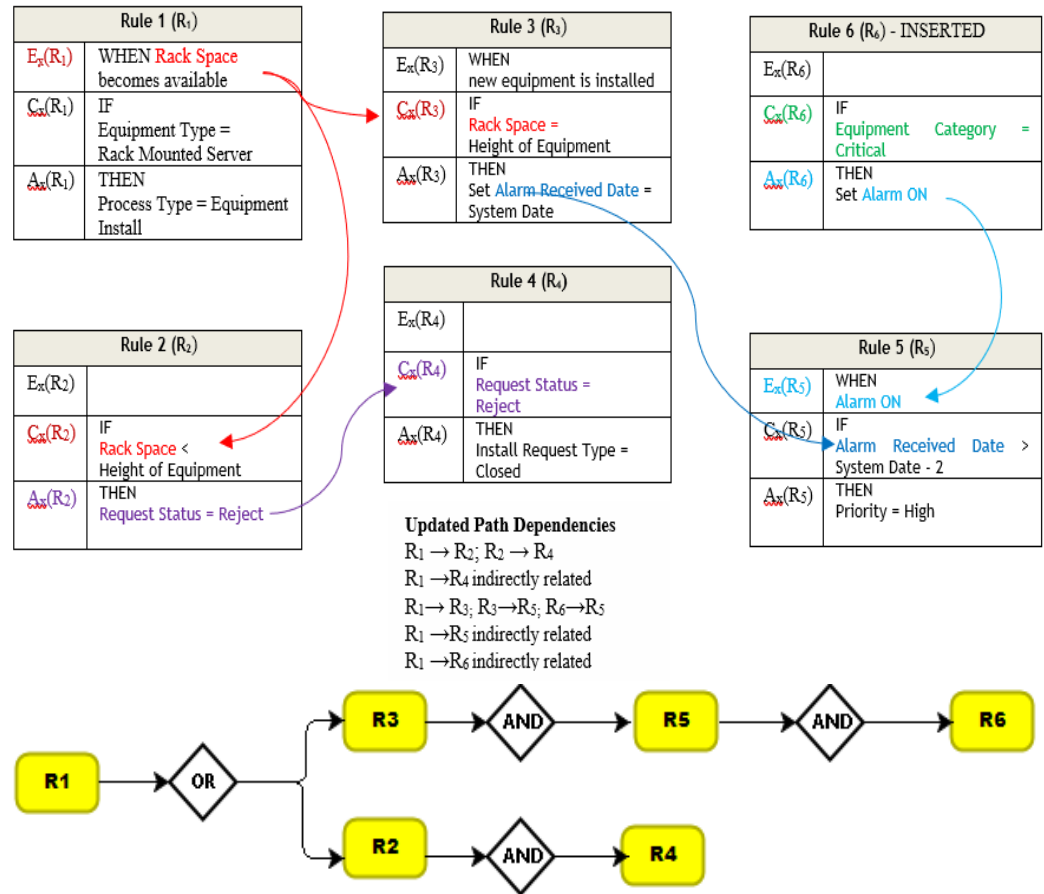


Figure 6.2. 4 Business Rule 3 Updated

By simply checking for path dependency patterns, not all the business rules dependencies are captured, for example, going by path dependency definition, only business rules on the same paths will be checked. However, if dependent business rules are stored on different paths, they would be missed by the path dependency algorithm. In this case, the scenario should observe the non-path dependency patterns, which are described next.

6.2.2 Direct-Node Dependency Propagation

Direct-Node dependency propagation or business rule component dependency propagation means business rules components are directly connected to the source (initiate) business rule components for change propagation. A business rule component can have multiple direct connections or dependencies, which means that changes to the initiating business rule component have a high effect on multiple business rules components linked to it. Knowing such information can greatly improve business rule change management as it can provide hints to which business rule components are highly connected and can cause great impact to multiple business rules components.

6.2.3 Level-Based Dependency Propagation

Level dependency propagation shows a change propagation on business rules components within the same levels of AND-OR dependency graph linked to the source (initiate) business rule components. As changes propagate between business rules components on the same levels, it is vital to know the links in order to manage change propagation and assess risk associated with component change. As different levels are assessed, it is important to know the extent of the likelihood and impact as well as the type of link. Exact representations for connectivity should show all aspects of change propagation in order to correctly support and manage change on related business rule components.

6.2.4 Neighbour Dependency Propagation

Neighbour dependency propagation shows change propagation between previous business rule components and succeeding business rules components. A business rule component has neighbour relationships. In our previous example, business rule R_3 's previous business rule component is R_1 's event and the succeeding business rule component is R_5 's condition. The R_1 's event activates business rule R_3 and R_3 's action cause R_5 's condition. There is an event-condition and action-condition dependencies between (R_1 and R_3) and (R_3 and R_5) respectively.

6.2.5 Indirect Node Dependency Propagation

Indirect node dependency propagation occurs when the source (initiate) business rule component has indirect connectivity with other business rule components causing a change to be propagated. Displaying indirect connectivity between components and related risk

values is a challenging task but crucial for the understanding of change propagation especially in a highly linked or connected business rules systems. However, due to time constraints, this research will only focus on indirect business rule components' relationships along the line of path dependencies. By utilizing dependency propagation patterns (described above) to confine business rules components, we provide a structured and formal environment for planning, organising, and managing changes in an orderly manner. Every change propagated, could be easily seen and analysed. At the time of writing this thesis, a simple algorithm for change propagation based on dependency patterns is implemented (Section 6.3). In future, an algorithm could be implemented to check the change propagation cost of each pattern based on impacted business rule components.

6.3 Algorithm for Business Rules Change Propagation

The five important business rules dependency patterns for change propagation were discussed in the previous section. This section introduces another important contribution to this research: the algorithm necessary for the implementation of business rule change propagation. The algorithm is created to provide a systematic runtime modification approach for the business rules change propagation challenge. Additionally, it offers the capability to minimize the performance issues during runtime execution of business rules change, using indexed propagation patterns described above. The unique feature of our work is expounding the change propagation challenge at business rule components level. The goal is to detect and map changes across business rules components in dependency patterns. The graph dependency patterns (mentioned in previous section), helps to determine which business rules are impacted by a change in a business rule component. If any business rule component changes, all connected direct and indirect business rule components must be revised. Formally the algorithm is defined in Definition 6.3 as follows:

Definition 6.3

Let each pattern be part of an AND-OR graph G which, consists of 'R' nodes representing business rules and arcs 'D' representing dependencies between business rule (nodes). Therefore $G(R, D)$ is a business rule graph and $c(ri)$ is a business rule node such that $c(ri)$

$\in R$; c is a component of a rule, i.e. event, condition and action. The set of $c(ri)$ business rule changed components are noted as $P(c(ri))$ such that $\forall c(rj) \in P(c(ri))$, $c(ri)$ is either path, level-based, direct-node, or neighbours business rule component for the source business rule component $c(rj)$. The letter ‘P’ stands for Propagation.

- We denote $P^p(c(ri))$ as a set of all $c(ri)$ business rules path components such that $\forall rj \in P(c(ri))$, $c(ri)$ is a path dependant business rule component for the source business rule component $c(rj)$.
- We denote $P^l(c(ri))$ as a set of all $c(ri)$ business rules level components such that $\forall rj \in P(c(ri))$, $c(ri)$ is a level dependant business rule component for the source business rule component $c(rj)$.
- We denote $P^{dn}(c(ri))$ as a set of all $c(ri)$ business rules direct-node components such that $\forall rj \in P(c(ri))$, $c(ri)$ is a direct-node dependant business rule component for the source business rule component $c(rj)$.
- We denote $P^n(c(ri))$ as a set of all $c(ri)$ business rules neighbour components such that $\forall rj \in P(c(ri))$, $c(ri)$ is a neighbour dependant business rule component for the source business rule component $c(rj)$.
- We denote $P^{rules}(c(ri))$ as a set of connected $c(ri)$ business rule components such that $P^{rules}(c(ri)) = P^p(c(ri)) \cup P^l(c(ri)) \cup P^{dn}(c(ri)) \cup P^n(c(ri))$. If $c(ri) \in R$ business rule component changes, then $P^{rules}(c(ri))$ are all business rule components, which will have to be revised. The revision may cause a propagation of business rule component change. Indeed, if one business rule component changes, the set of path, level, direct-nodes and neighbours business rules will be revised and the change properly propagated. This will raise the need to revise another set of path, level, direct-nodes and neighbours’ business rules of the business rule component that was revised and so on, until there are no more business rules to change. The following (Code Snippet 6.3.1) summarises the change propagation algorithm.


```

//Business Rule Change Propagation Algorithms: component is changed and propagated
//across dependency patterns listed in DependencyPatterns
//Input: business rule component

public void ECACHangePropagation(ECAModel changedECAcomponent, ECAGraph ecaRuleG){

//Declare variables
//Variable to store sorted/indexed dependency pattern so
//it's easy to propagate the change.
//Note calling IndexingGraphPatterns(ecaRuleG) provide different dependency patterns
//(Path, Level, Direct-Node and Neighbours dependencies)

    List<ECAModel> DependencyPatternsIndexes = IndexingGraphPatterns(ecaRuleG);
    ECAModel changedRulecomponent = new ECAModel();
    ECAModel changedLinkedRulecomponents = new ECAModel();

    //Check if the business rule component existing in the using
    //DependencyPatternsIndexes
    //loop through graph to identify dependency patterns

    for (int index=0; index < DependencyPatternsIndexes.size();index++){
        changedRulecomponent = changedRulecomponent.children().get(index);
        //Using rule component index list (DependencyPatternsIndexes)
        //to check if component exist
        if (changedRulecomponent == null) return;
        if (changedRulecomponent == changedECAcomponent) {

            //check if changedRulecomponentIndex has children
            if (changedRulecomponent.hasChildren()) {

                //get the index of the business rule component to be updated
                int changedRulecomponentIndex = changedRulecomponent.getruleIndex();

                //Using the changeRulecomponentIndex to propagate the change
                //to all dependency business rule components (children)
                //Perform delete and add to propagate the change to children components
                changedLinkedRulecomponents.children().remove(changedRulecomponentIndex)
                changedLinkedRulecomponents.children().add(changedRulecomponentIndex,
                    changedECAcomponent);
            }
        }
    }
}

```

Code Snippet 6.3. 1 Business Rules Change Propagation Algorithm

6.4 Business Rules Adaptation in Business Workflows

Rules adaptation in business workflows still deserves further investigation. Existing solutions do not completely address all problems. Particularly, the configuration of a business process is not always easy (business processes are rigid and difficult to maintain). According to [38], it is helpful to use rules to enforce how business process should work. [38] argues that rules give a flexible way to specify a process's control flow in a workflow. However, most business workflows usually offer capabilities for evaluating rules as either built-in the business process languages or implemented for specific applications. Therefore, it is entirely achievable to implement simple business rules in the business process engine, but this means any changes in workflows will require recompilation, full testing and redeployment. In case of complex ECA business rules, a separate service needs to be implemented, away from process. It is necessary to consider the complexity and the frequency of change in situation when business rules support the execution of business processes. The ability of ECA business rules to support dynamic changes in business rules allows us, in this case, to modify business process implementation without changing and redeploying it. To that end, the objective of this section is to describe business processes with a focus of using a set of connected business rules with the aim of providing flexibility in workflow configuration. We introduced Figure 6.4.2 to present the business rules relationships (dependencies) for the request cancellation workflow. Note, Figure 6.4.1 shows the original workflow with embedded business rules (highlighted in yellow). In the workflow, upon receipt of a cancellation notification from a requestor, the cancellation reason is checked. The request can be cancelled if the following cancellation reasons are true: no space for equipment, equipment causing power overload and equipment already existing in the data centre then the equipment type is defined as a server. When notified equipment type is a server, Power connections and Network connections are checked; note that these two processes are done simultaneously. A cancellation request can also be rejected (R4). When both power and network connections are removed, a request is set to be closed. Once the request is closed, request cancellation is complete. The typical workflow and business processes will be documented as follow:

- P1 - Request Notification Cancellation
- P2 - Network Connection Preparation

-
- ```

graph LR
 Start((Start)) -- R1 --> RNC[Request Notification Cancellation]
 RNC -- R4 --> RAR[Request Approve/Reject]
 RNC -- R3 --> PCP[Power Connection Preparation]
 RNC -- R2 --> NCP[Network Connection Preparation]
 RAR -- R5 --> RC[Request Complete]
 PCP -- R6 --> PCR[Power Connections Removal]
 NCP -- R7 --> NCR[Network Connections Removal]
 PCR -- R5 --> RC
 NCR -- R5 --> RC
 RC --> End((End))

```
- The flowchart illustrates the Request Approval Process. It begins with a 'Start' node (green circle) leading to 'Request Notification Cancellation' (blue rectangle) via edge R1. From 'Request Notification Cancellation', three paths emerge: one to 'Request Approve/Reject' (blue rectangle) via edge R4, one to 'Power Connection Preparation' (blue rectangle) via edge R3, and one to 'Network Connection Preparation' (blue rectangle) via edge R2. 'Request Approve/Reject' leads to 'Request Complete' (blue rectangle) via edge R5. 'Power Connection Preparation' leads to 'Power Connections Removal' (blue rectangle) via edge R6, which then leads to 'Request Complete' via edge R5. 'Network Connection Preparation' leads to 'Network Connections Removal' (blue rectangle) via edge R7, which then leads to 'Request Complete' via edge R5. Finally, 'Request Complete' leads to an 'End' node (red circle).

The diagram illustrates seven fuzzy inference rules (R<sub>1</sub> to R<sub>7</sub>) for a fuzzy expert system. Each rule is represented by a table with conditions (E, C) and actions (A). Arrows indicate the flow of inference between rules.

**Rule 1 (R<sub>1</sub>)**

| Rule 1 (R <sub>1</sub> )         |                                                                                                |
|----------------------------------|------------------------------------------------------------------------------------------------|
| E <sub>x</sub> (R <sub>1</sub> ) | WHEN Request Notification = Cancel                                                             |
| C <sub>x</sub> (R <sub>1</sub> ) | IF Request Cancellation Reason in (no space availability, cause power overload, already exist) |
| A <sub>x</sub> (R <sub>1</sub> ) | THEN Type (Equipment) = 'Server'                                                               |

**Rule 2 (R<sub>2</sub>)**

| Rule 2 (R <sub>2</sub> )         |                                  |
|----------------------------------|----------------------------------|
| E <sub>x</sub> (R <sub>2</sub> ) | WHEN Type (Equipment) = 'Server' |
| C <sub>x</sub> (R <sub>2</sub> ) | IF Network Connections > 0       |
| A <sub>x</sub> (R <sub>2</sub> ) | THEN NetworkRemoval = True       |

**Rule 3 (R<sub>3</sub>)**

| Rule 3 (R <sub>3</sub> )         |                                  |
|----------------------------------|----------------------------------|
| E <sub>x</sub> (R <sub>3</sub> ) | WHEN Type (Equipment) = 'Server' |
| C <sub>x</sub> (R <sub>3</sub> ) | IF Power Connections > 0         |
| A <sub>x</sub> (R <sub>3</sub> ) | THEN PowerRemoval = True         |

**Rule 4 (R<sub>4</sub>)**

| Rule 4 (R <sub>4</sub> )         |                                  |
|----------------------------------|----------------------------------|
| E <sub>x</sub> (R <sub>4</sub> ) | WHEN Type (Equipment) = 'Server' |
| C <sub>x</sub> (R <sub>4</sub> ) | IF Request Status = Reject       |
| A <sub>x</sub> (R <sub>4</sub> ) | THEN Request Type = Closed       |

**Rule 5 (R<sub>5</sub>)**

| Rule 5 (R <sub>5</sub> )         |                                  |
|----------------------------------|----------------------------------|
| E <sub>x</sub> (R <sub>5</sub> ) | WHEN Request Type = Closed       |
| C <sub>x</sub> (R <sub>5</sub> ) |                                  |
| A <sub>x</sub> (R <sub>5</sub> ) | THEN Request Status = 'Complete' |

**Rule 6 (R<sub>6</sub>)**

| Rule 6 (R <sub>6</sub> )         |                            |
|----------------------------------|----------------------------|
| E <sub>x</sub> (R <sub>6</sub> ) | WHEN PowerRemoval = True   |
| C <sub>x</sub> (R <sub>6</sub> ) |                            |
| A <sub>x</sub> (R <sub>6</sub> ) | THEN Request Type = Closed |

**Rule 7 (R<sub>7</sub>)**

| Rule 7 (R <sub>7</sub> )         |                            |
|----------------------------------|----------------------------|
| E <sub>x</sub> (R <sub>7</sub> ) | WHEN NetworkRemoval = True |
| C <sub>x</sub> (R <sub>7</sub> ) |                            |
| A <sub>x</sub> (R <sub>7</sub> ) | THEN Request Type = Closed |

110

The design of the workflow is based on business rules components. Figure 6.4.2 presents business rules' relationships, which are created to provide functionalities of the request cancellation workflow. With regards to adaptation of business rules in workflows, several business rules' constructs or concepts are formed. For example, the initiating business rule is responsible for generating the starting process of the workflow. The intermediate (sequential and parallel flow rules) are responsible for providing links between processes. These constructs or concepts are important in presenting the logic and semantics of business processes in a workflow. It is important the business rules support the following key workflow concepts:

- Ability to enable a business rule to initiate a process in a workflow
- Ability to enable a business rule to terminate a process in a workflow
- Ability to enable a business rule to generate sequential process flow patterns
- Ability to enable a business rule to generate AND-Parallel Split process flow patterns
- Ability to enable a business rule to generate OR-Parallel Split process flow patterns
- Ability to enable a business rule to generate AND-Merge process flow patterns
- Ability to enable a business rule to generate OR-Merge process flow patterns
- Due to time limitation, the exclusive alternatives flow patterns (XOR-split, XOR-Merge) are not considered in this research.

To build a complete algorithm that controls and manages an instance of a workflow, the following seven constructs or concepts are important. Each is defined below to show how it is used in a workflow generation and configuration:

1. Initiating Business Rules
2. Terminating Business Rules
3. Sequential Flow Rules
4. Parallel AND-Split Flow Rules
5. Parallel OR-Split Flow Rules
6. Parallel AND-Merge Flow Rules
7. Parallel OR-Merge Flow Rules

### 6.4.1 Initiating Business Rule

To model how a process is initiated or activated by using business rule components within a workflow, we simply create an initiating business rule to start a workflow process. This can be achieved by raising a business rule event component to cause condition and action components of another business rule component to execute or a business condition causes another business rule's action to be executed. Furthermore, a business rule action component can invoke an event of another business rule or cause another business rule's condition to occur. A business rule event component can be raised either explicitly or implicitly with an event from a workflow environment or other integrated applications e.g., an update from a database. Note, an initiation business rule is a starting business rule with no defined predecessor connection or linked business rule component. In Figure 6.4.2 above business rule R1 is an initiating business rule causing for R2, R3 and R4 to execute via action-event relationship as illustrated in Figure 6.4.3 below. The action component of R1 is associated with business rule event component of R2, R3 and R4.

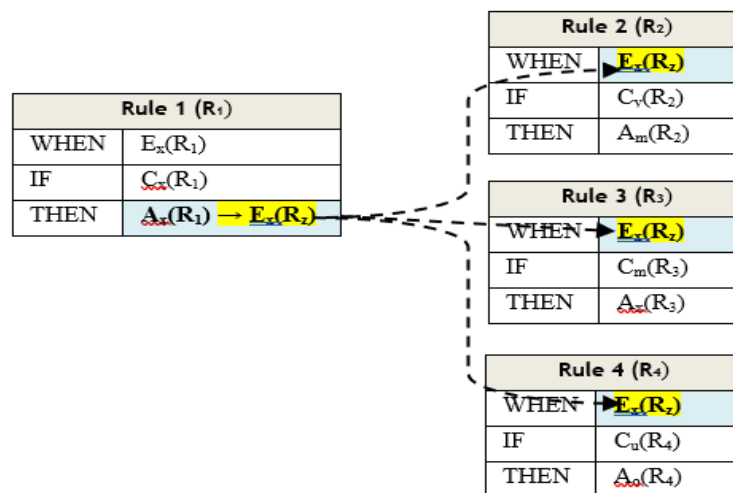


Figure 6.4. 3 Initiate Workflow via Business Rules

### 6.4.2 Terminating Business Rule

To model how a process is terminated or stopped by using business rule components within a workflow, we simply create a terminating business rule that ends a process in a workflow. This can be achieved in similar ways as above, however a terminating business rule has no defined successor connection or linked business rule component. As an example, for the

scenario in Figure 6.4.2, business rule action component from R4 causes R5 event component to be activated to perform action “request complete” to end the process. Figure 6.4.4 below illustrated how a process termination is linked using business rule components.

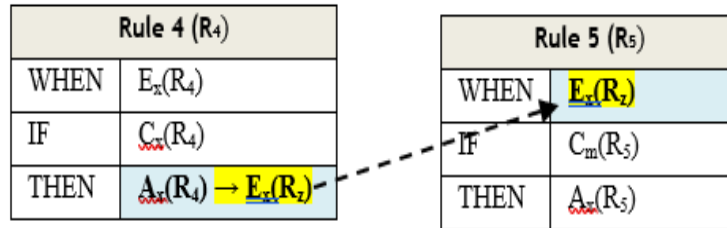


Figure 6.4. 4 Terminate Workflow via Business Rules

### 6.4.3 Sequential Flow Business Rules

A sequential process flow is simply achieved by linking associated business rules. An event of one business rule is raised when the preceding action of another business rule terminates. Also, an action of one business rule may cause the condition of another business rule to perform an action that causes sequential flow. The sequential connection between the business rule action components is founded on the fact that the business rule event component resulting from the preceding business rule action component appears as a triggering of a business rule event component. The workflow forms a chain of business rules linked via the business rules components. In Figure 6.4.2 above, business rule action component from R1 causes R3 event to be activated; thereafter R3 action causes R6 event component to be activated, which describes cancel notification request and power connections removal processes. Business rules are illustrated in Figure 6.4.5 below.

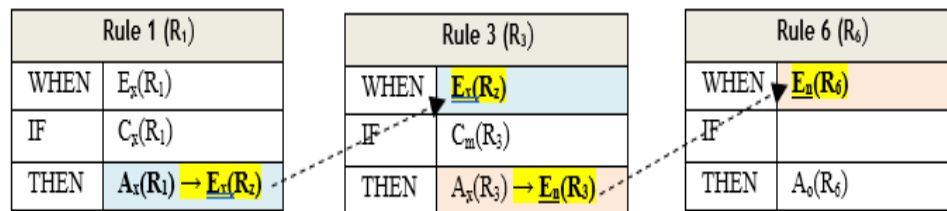


Figure 6.4. 5 Business Rules modelling Sequential Process Flows

### 6.4.4 Parallel AND-Split Flow Business Rules

To model parallel AND-Split process flows by using business rules within a workflow, we simply link the related business rules components. This can be achieved in different ways

by splitting the control flow into parallel paths; one is to trigger one or more business rules event components by an action from another business rule. The key here is that there will be parallel triggered business rules event components by an action. Another alternative is to define one or more business rules with the same business rule event component. An AND ( $\wedge$ ) conjunction will be specified to flow processes in parallel. The use of an AND conjunction operator is explained in [69]. In Figure 6.4.2 above business rule action component from R1 causes R2, R3 and R4 event to be activated by using parallel AND-Split flow as illustrated in Figure 6.4.6.

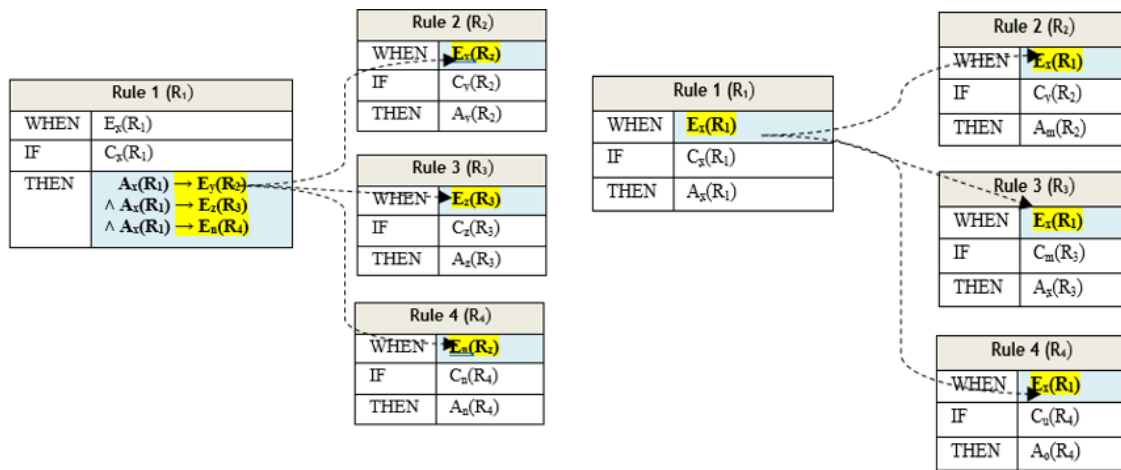


Figure 6.4. 6 Business Rules modelling Parallel AND-Split Process Flows

### 6.4.5 Parallel OR-Split Flow Business Rules

To demonstrate Parallel OR-Split process flows within a workflow, we simply link the related business rules components. The technique is comparable to a parallel AND-Split process flow. As seen above, one way is to trigger one or more business rules event components by an action from another business rule action component. The key here is that there will be parallel triggered business rules event components by an action. An alternative way is to define one or more business rules with the same business rule event component. Furthermore, an action of one business rule may cause multiple conditions of another business rule to perform alternative actions that cause a parallel OR-Split flow. The resulting flow paths are combined by a disjunction operator ( $\vee$ ) based on components of the business rules. The use of an “Or” disjunction operator is explained in [69]. In Figure

6.4.2 above business rule action component from R1 causes R2, R3 and R4 event to be activated using parallel OR-Split as illustrated in Figure 6.4.7 below.

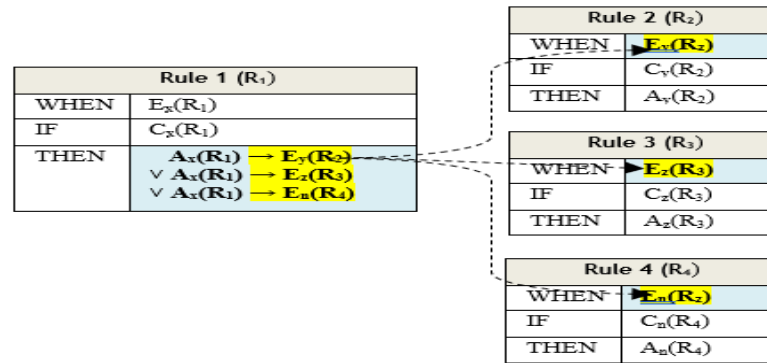


Figure 6.4. 7 Business Rules modelling Parallel OR-Split Process Flows

### 6.4.6 Parallel AND-Merge Flow Business Rules

To model AND-Merge process flow by using business rules within a workflow, we simply link the related business rules components. This can be achieved in different ways by merging the control flow of parallel paths into one; one way is to activate a business rules event component by multiple business rules action components. The key here is that there will be parallel business rules action components causing a single event component. An alternative way is to define several business rules event components to implement with option condition components to execute a single action. An AND ( $\wedge$ ) conjunction will be specified to merge process flows. In Figure 6.4.2 above business rule action component from R6 and R7 in parallel causes R5 event to be activated. Business rules for AND-Split parallel process flows are illustrated in Figure 6.4.8 below.

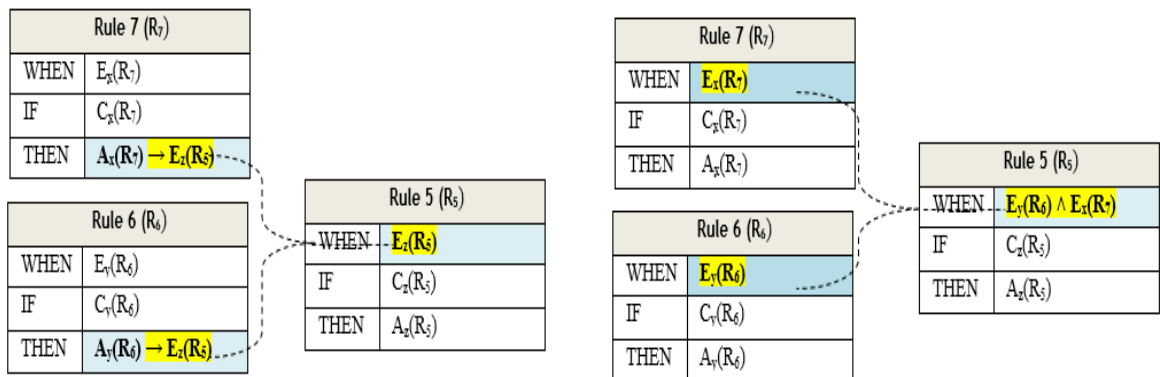


Figure 6.4. 8 Business Rules modelling AND-Merge Process Flows



### 6.4.7 Parallel OR-Merge Flow Business Rules

To model OR-Merge the process flow by using business rules within a workflow, we simply link the related business rules components. The technique is comparable to a parallel AND-Merge process flow. As seen above, this can be achieved in different ways by merging the control flow of parallel paths into one; one way is to activate a business rules event component by multiple business rules action components. The key here is that there will be parallel business rules action components causing a single event component. An alternative way is to define several business rules event components with option condition components to execute a single action. The flow paths are combined by a disjunction operator ( $\vee$ ) based on components of the business rules. In Figure 6.4.2 above business rule action component from R6 and R7 in parallel causes R5 event to be activated. Business rules for OR-Merge process flows are illustrated in Figure 6.4.9 below.

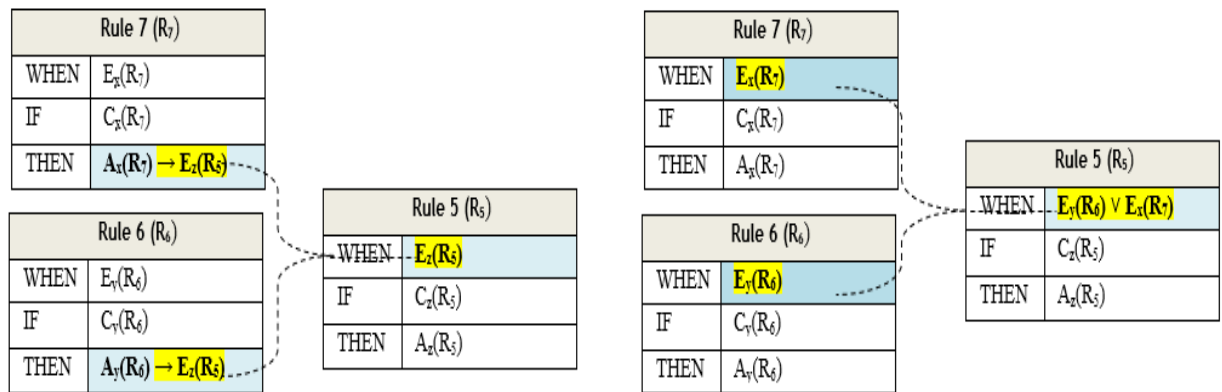
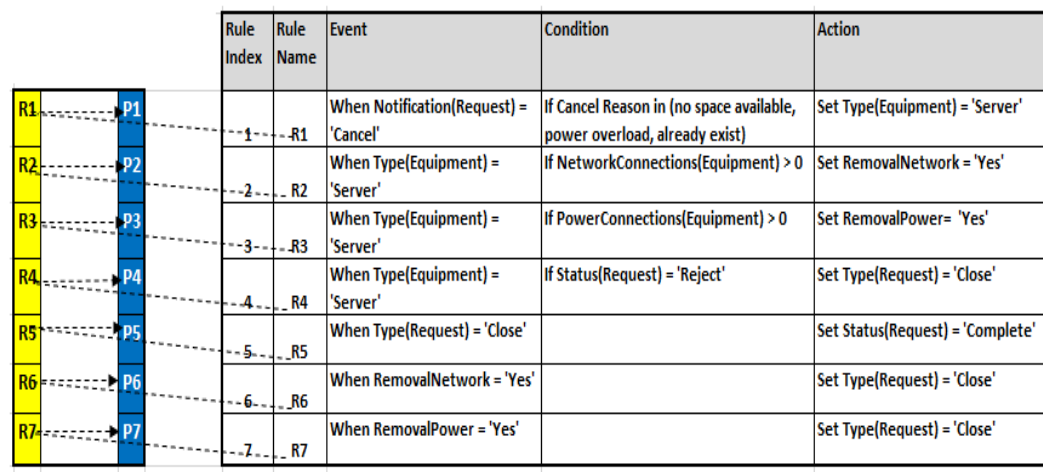


Figure 6.4. 9 Business Rules modelling OR-Merge Process Flows

## 6.5 Algorithm for Business Rules Adaptation in Workflows

This section introduces an important algorithm necessary for the implementation of rule adaptation in business workflows. The algorithm is created to provide a systematic approach of adaptation of rules to govern a workflow. If any business rule component changes, all underlying connected business processes are revised. It offers the capability of allowing users to generate and modify business processes automatically. Three steps are given to explain the adaptation algorithm:

1. Generating business process from business rules (Figure 6.5.1), note rules are indexed through dependency patterns. We argue that for each business rule that is generated, there is a process initiated by an event or activated by an action. So, user-friendly process names can be defined from such as event/action labels. However, for simplicity our processes will be labelled with an initial 'P' followed by the business rule number. For instance, process name  $P_1$  belongs to business rule  $R_1$ .



| Rule Index | Rule Name | Event                                 | Condition                                                               | Action                           |
|------------|-----------|---------------------------------------|-------------------------------------------------------------------------|----------------------------------|
| 1          | R1        | When Notification(Request) = 'Cancel' | If Cancel Reason in (no space available, power overload, already exist) | Set Type(Equipment) = 'Server'   |
| 2          | R2        | When Type(Equipment) = 'Server'       | If NetworkConnections(Equipment) > 0                                    | Set RemovalNetwork = 'Yes'       |
| 3          | R3        | When Type(Equipment) = 'Server'       | If PowerConnections(Equipment) > 0                                      | Set RemovalPower = 'Yes'         |
| 4          | R4        | When Type(Equipment) = 'Server'       | If Status(Request) = 'Reject'                                           | Set Type(Request) = 'Close'      |
| 5          | R5        | When Type(Request) = 'Close'          |                                                                         | Set Status(Request) = 'Complete' |
| 6          | R6        | When RemovalNetwork = 'Yes'           |                                                                         | Set Type(Request) = 'Close'      |
| 7          | R7        | When RemovalPower = 'Yes'             |                                                                         | Set Type(Request) = 'Close'      |

Figure 6.5. 1 Business Process Nodes Creation

2. We use an AND-OR graph to generate dependencies/edges between business rules components to identify source connected rule components, destination connected rule components, rule flow constructs and relation operators (Table 6.5.1). Rule flow constructs and operations (Initiating Rule, Terminating Rule, Sequential Flow Rule, Parallel AND-Split Flow Rules, Parallel OR-Split Flow Rules, Parallel AND-

Merge Flow Rules and Parallel OR-Merge Flow Rules) are important as they let us understand or determine the process flow transitions or directions. Essentially, the goal is to detect business rule relationships and type of relationships by using a graph. Note, the relationships are also indexed. The flow rule constructs are discussed in more detail in Section 6.4.

| Index | Rule Name      | Rule Index | Source Business Rule | Destination Business Rule | Rule-Flow Constructs | Relation Operator |
|-------|----------------|------------|----------------------|---------------------------|----------------------|-------------------|
| 1     | R <sub>1</sub> | 1          | Null                 | R <sub>3</sub>            | Parallel Split       | AND               |
| 2     | R <sub>1</sub> | 1          | Null                 | R <sub>2</sub>            | Parallel Split       | AND               |
| 3     | R <sub>1</sub> | 1          | Null                 | R <sub>4</sub>            | Parallel Split       | OR                |
| 4     | R <sub>2</sub> | 2          | R <sub>1</sub>       | R <sub>7</sub>            | Sequential           |                   |
| 5     | R <sub>3</sub> | 3          | R <sub>1</sub>       | R <sub>6</sub>            | Sequential           |                   |
| 6     | R <sub>4</sub> | 4          | R <sub>1</sub>       | R <sub>5</sub>            | Parallel Merge       | OR                |
| 7     | R <sub>5</sub> | 5          | R <sub>4</sub>       | Null                      | Terminating Rule     |                   |
| 8     | R <sub>5</sub> | 5          | R <sub>6</sub>       | Null                      | Terminating Rule     |                   |
| 9     | R <sub>5</sub> | 5          | R <sub>7</sub>       | Null                      | Terminating Rule     |                   |
| 10    | R <sub>6</sub> | 6          | R <sub>3</sub>       | R <sub>5</sub>            | Parallel Merge       | AND               |
| 11    | R <sub>7</sub> | 7          | R <sub>2</sub>       | R <sub>5</sub>            | Parallel Merge       | AND               |

Table 6.5. 1 Business Rules Dependency Mapping Table

- Steps 1 and 2 are merged to generate business processes and their connectivity. Process transitions or flows are determined through the connected business rule indexes, predecessors, successors, rule flow constructs and relation operators. Figure 6.5.2 shows pictorial stages of transforming business rules to a workflow.

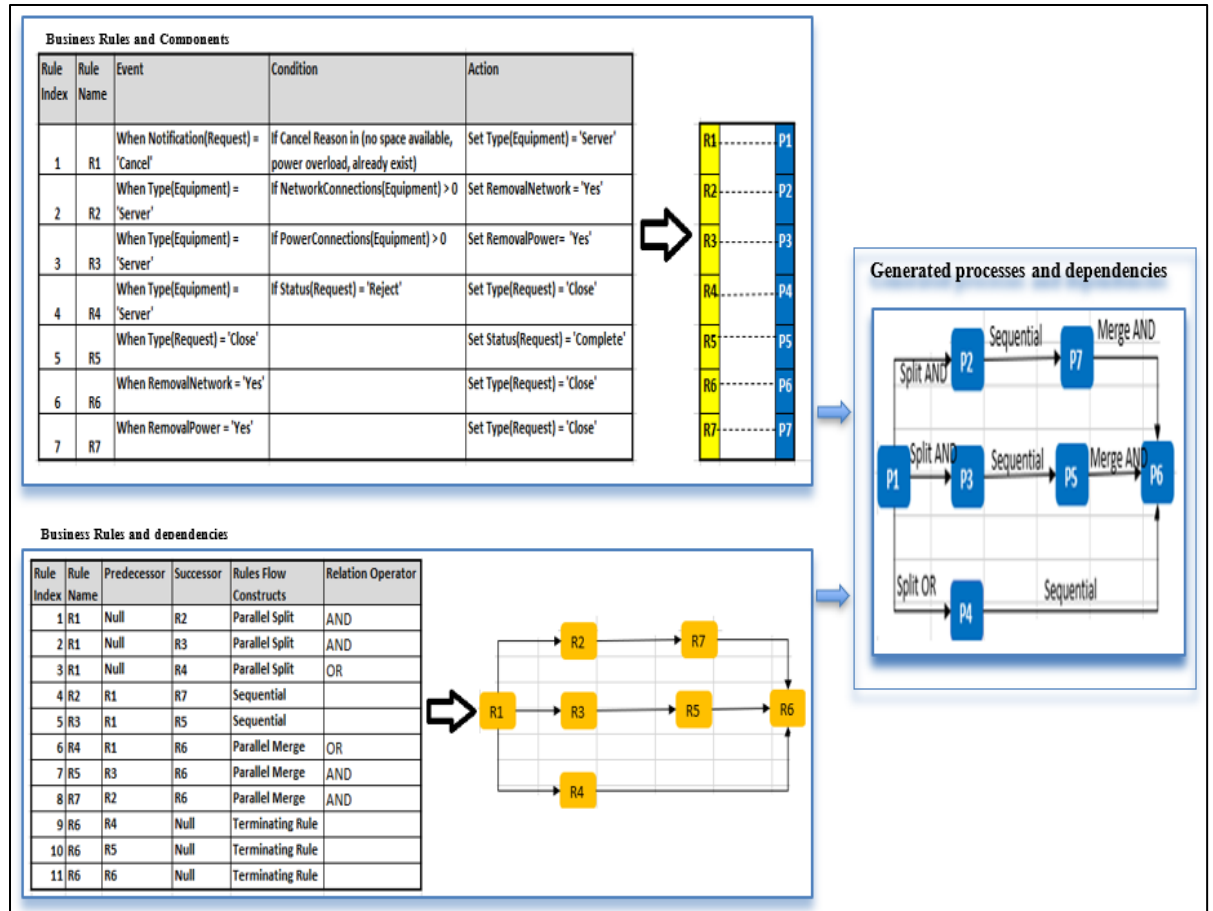


Figure 6.5. 2 Business Rules Mapping Table to Workflow

Before defining the adaptation algorithm, it is worth mentioning that the process of indexing business rule components and dependencies are introduced to optimize performance and minimize the number of accesses required when business rules are searched, inserted and updated. Refer to section 5.3 for more information on storage and performance conscious indexes for the AND-OR graphs. Code Snippets 6.5.1 - 6.5.3 show the pseudocode of the three phases of business rule workflow adaptation algorithm.

**//Phase 1: Convert/Transform business rules into processes**

**Process CreateProcessFromBusinessRules(List<ECAModel> RuleList)** {*//Input indexed ECA rule list then return Workflow*

*// Use business rules to define processes for the workflow: start, intermediate and end processes*

ArrayList<Process> ProcessList=new ArrayList<Process>();

for (int index = 0; index < RuleList.size(); index++) {

*//if a rule's predecessor is null then create start node*

List<ECAModel> ecaRuleList = RuleList;

if (ecaRuleList.get(index).isRoot()) {

StartNode pStart = new StartNode();

pStart.setIdx(index); *//Note index of a rule assigned to be used in the merged phase*

pStart.setName("P" + index);

*//Define process definition based on Action Nodes*

Process WF\_pStart = new Process(pStart);

WF\_pStart.setStartprocess(pStart);

ProcessList.add(WF\_pStart);

}

*//if a rule's predecessor or successor is not null then create intermediate node*

if (ecaRuleList.get(index).hasChildren() && ecaRuleList.get(index).isRoot()== false) {

ActionNode p1 = new ActionNode();

p1.setIdx(index); *//Note index of a rule assigned to be used in the merged phase*

p1.setName("P" + index);

DroolsAction action = new DroolsAction();

action.setMetaData("Action", new Action() {

public void execute(ProcessContext context) throws Exception {

System.out.println("Error define process node");

});

p1.setAction(action);

*//Define process definition based on Action Nodes*

Process WF\_p = new Process(p1);

WF\_p.setProcess(p1);

ProcessList.add(WF\_p);

}

*//if a rule's successor is null the create end node*

if (ecaRuleList.get(index).hasChildren() == false) {

EndNode pEnd = new EndNode();

pEnd.setIdx(index); *//Note index of a rule assigned to be used in the merged phase*

pEnd.setName("P" + index);

*//Define process definition based on Action Nodes*

Process WF\_pEnd = new Process(pEnd);

WF\_pEnd.setEndprocess(pEnd);

ProcessList.add(WF\_pEnd);

}

...

return ProcessList;

}

Code Snippet 6.5. 1 Transform Business Rules into Processes

**//Phase 2: Define Business Rule relationships (dependencies) as per Table 6.5**

**//Dependency is based on Object and Properties of one rule are matched using objects and properties of one or more rules**

**Void BusinessRulesDependency**(ECAGraph ecaRuleG) *{//It takes in the ECA AND-OR Graph parameter as a skeleton*

*//Create Rule relationships using the indexed RuleList then build the graph*

```
for (int i = 0; i < RuleList.size();i++) {
 String eventList1 = (RuleList.get(i).getevent()).toString();
 System.out.println("Check object 1" + eventList1);
 for (int j = i + 1; j < RuleList.size(); j++) {
 RuleList.get(j).getcondition(), RuleList.get(j).getaction();
 String eventList2 = (RuleList.get(j).getevent()).toString();
 String condList2 = (RuleList.get(j).getcondition()).toString();
 String actionList2 = (RuleList.get(j).getaction()).toString();
 System.out.println("Check object 2" + eventList2);
```

*/\*Event-AND Relationship scenario, note Condition and Action components are not shown here but they implemented in similar manner. Obtain the list of rules - RuleList. Object and properties of one rule's Event relates to event and condition and action of another rule's object and properties. if event component of one rule matches all: event, condition, action components of the second rule then create relationships between rules\*/*

```
if (eventList1.contains(eventList2) && eventList1.contains(condList2) && eventList1.contains(actionList2)) {
 rulelink = CreateRuleRelationships(ecaRuleG, i, j, RuleList);
}
```

*//Event-OR Relationship. Object and properties of rule 1 Event relates to either event or condition or action of //another rule's object and properties. if equal then create relationships between rules*

```
if (eventList1.contains(eventList2)) {
 rulelink = CreateRuleRelationships(ecaRuleG, i, j, RuleList, "E");
}
if (eventList1.contains(condList2)) {
 rulelink = CreateRuleRelationships(ecaRuleG, i, j, RuleList, "C");
}
if (eventList1.contains(actionList2)) {
 rulelink = CreateRuleRelationships(ecaRuleG, i, j, RuleList, "A");
}
}
```

*//The following method is used by " **BusinessRulesDependency** " for creation of business rules dependencies source/destination*  
**ECAGraph CreateRuleRelationships**(ECAGraph ecaRuleG, int Rulesrc, int Ruledest, List<ECAModel>RuleList, String ComponentType){

*//Define a variable to hold the created business rule relationship graph*  
 ECAGraph ecaRuleRelationshipsGraph;

*//Create relationship between source and destination*

```
ecaRuleG.ruleRelations.get(Rulesrc).add(RuleList.get(Ruledest).getruleName());
ecaRuleRelationshipsGraph = ecaRuleG;
```

*//Note: Once the business rules graph is defined, we can easily identify predecessors, successors, rule flow constructs and //relation operators as discussed in Phase 2 of adaptation ready for Phase 3*  
 return ecaRuleRelationshipsGraph;

}

Code Snippet 6.5. 2 Define Business Rule Dependencies as per Table 6.5.1

```

//Phase 3: Creation of business process connections and use of Drool's APIs to execute the workflow
//Input indexed ECA rule list then return Workflow

RuleFlowProcess CreateWorkflowFromBusinessRules(List<ECAModel> RuleList) {
//Create a template for the ECAWorkflow instance

RuleFlowProcess ruleworkprocess = new RuleFlowProcess();
ruleworkprocess.setId("ECAWorkflow");

//Define RuleProcessList to create process list
List<ECAModel> RuleProcessList = new ArrayList<ECAModel>();

//Using CreateProcessFromBusinessRules method to loop through ProcessList
for (int j = 0; j < ProcessList.size(); j++) {

//Using CreateRuleRelationships method to loop through connected business rule (source and destination)
for (int index=0; index< ecaRuleRelationshipsGraph.ruleRelations.size();index++){

//build workflow based on process and rules
if (j == index) {
//Use the ProcessRuleList to build relationships between processes from rules
RuleProcessList.add(new ECAModel(RuleList.get(index)), Rulesrc, Ruledest));
}
}
//Using Drools APIs for workflow
for (int i = 0; i < RuleProcessList.size();i++) {

if (RuleProcessList.get(i).getfromProcess().equals(Rulesrc) && i == 0) {
new ConnectionImpl(RuleProcessList.get(i).getfromProcess().getStartprocess(),
"DROOLS_DEFAULT", RuleProcessList.get(i).gettoProcess().getProcess(), "DROOLS_DEFAULT");

//Adding nodes to the workflow
ruleworkprocess.addNode(Rulesrc.getStartprocess());
}
if (RuleProcessList.get(i).getfromProcess().equals(Rulesrc) && i <> 0 && i <> RuleProcessList.size()) {
new ConnectionImpl(RuleProcessList.get(i).getfromProcess().getProcess(),
"DROOLS_DEFAULT", RuleProcessList.get(i).gettoProcess().getProcess(), "DROOLS_DEFAULT");

//Adding nodes to the workflow
ruleworkprocess.addNode(Rulesrc.getProcess());
}
if (RuleProcessList.get(i).gettoProcess().equals(Rulesrc) && i == RuleProcessList.size()) {
new ConnectionImpl(RuleProcessList.get(i).getfromProcess().getProcess(),
"DROOLS_DEFAULT", RuleProcessList.get(i).gettoProcess().getEndprocess(), "DROOLS_DEFAULT");
//Adding nodes to the workflow
ruleworkprocess.addNode(Rulesrc.getEndprocess());
}
}
return ruleworkprocess;
}

```

Code Snippet 6.5. 3 Workflow Creation using Drools APIs

## 6.6 Summary

This Chapter familiarized a reader with two important algorithms to this research, the implementation of business rule change propagation and business rules adaptation in

workflow. The business rules change propagation algorithm is created to provide a systematic runtime modification of related business rule components (events, conditions, actions). If any business rule component changes, all underlying connected business rules components are revised and change is applied. The business rules adaptation in workflows algorithm provides the implementation of business rules to control processes in workflows. It offers the capability of allowing users to generate and modify business processes automatically. The algorithm's application produces formal procedures for those who are concerned with building a framework of business rules to manage workflows. This presents an important contribution, which offers an innovative analytical and methodological approach in using business rule components and their relationships to propagate change, automation and configuration of workflows.



## 7. Implementation

So far, the proposed framework has been discussed theoretically. The previous two chapters introduced algorithms that have been created to provide and improve important functionalities of the proposed model. This Chapter covers the implementation aspect of the proposed model based on a prototype, which was developed using Java on JBoss Drools. JBoss Drools [52] is an open source business rule management system that provides a suitable environment for implementation, installation and execution of business rules and processes [90]. To get a better understanding on how business rules are implemented using Drools, section 8.1 presents an overview of Drools. Appendix VI outlines necessary steps for setting up and using Drools. Section 7.2 discusses the prototype implementation to demonstrate various model components and functionalities introduced in earlier Chapters. Section 7.3 presents a summary on implementation matters.

### 7.1 Drools Overview

The Drools project started back in 2001 and became an operational rule engine with its 2.0 first release. In 2005, it was acquired by JBoss and it became known as JBoss Rules. In 2006, JBoss was acquired by Red Hat. With monetary support from Red Hat, the JBoss Rules were rewritten and enhanced the Rete implementation with a GUI tool. In 2007, the “Drools” name was reclaimed and referred to it as "Drools" instead of JBoss Rules. Drools business rules management system is written in Java [105]. The current stable version is 7.38.0 [24]. The Drools is made up of several components that form a Business Logic Integration Platform (BLIP) - Figure 7.1.1.

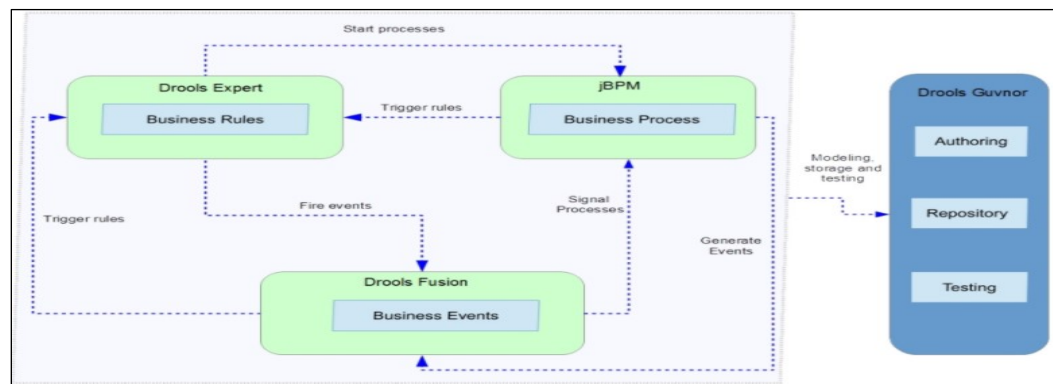


Figure 7.1. 1 Business Logic Integration Platform

(Source: [105])

Drools platform provides a complete solution for knowledge-based application development and management of business rules and processes. The components of Drools are briefly explained in Appendix VII.

## 7.2 ECA Model Prototype

The ECA Model prototype is written using Java in Eclipse IDE. At the time of writing this thesis, our prototype supports business rules that feature in a data centre workflow to allow installation, decommissioning and moving of equipment. For the implementation of our data centre use cases, the ideal approach is to provide implementation support for the business rules of ECA Model presented in Chapter 4. Automatically, templates are generated based on class definitions, which are used to map data obtained through Rule Designer. Figure 7.2.1 shows the flow of various business rules and components of ECA Model, to and from Drools Platform.

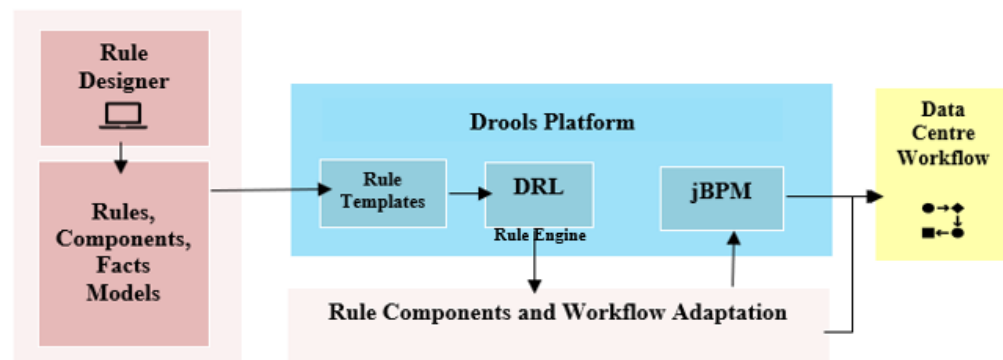


Figure 7.2. 1 Business Rules Modules Integration within Drools Platform

The following (Figure 7.2.2) are the key implementation steps of the prototype:

1. Implementation of user interface to allow business rules and components to be processed (inserted, deleted and updated). Converting business rule statements into business rules components (events, event operators, conditions, condition operators, actions and action operators). Note, in all major business rules

management systems, business rules still are expressed in forms of simple statements. This step is executed using the GUI.

2. Creation and implementation of business rules components, facts, process and other related classes. This involves creation of event class, condition class, action classes, ECA Model class and others. The ECA Model class is a parent class that is made up of event, condition and action classes. The classes are comprised of objects, properties, values, operators and methods/functions
3. Implementation of ECA Model class concepts to map rule components to Rule Template
4. Method of translating Rule Template into Drools DRL
5. Implementation of dependency patterns indexing algorithm to manage business rules components dependency and change propagation
6. Implementation of Metarules to manage runtime business rules
7. Implementation of business rule change propagation and rule adaptation algorithms
8. Business Rules and Workflow execution using Drools

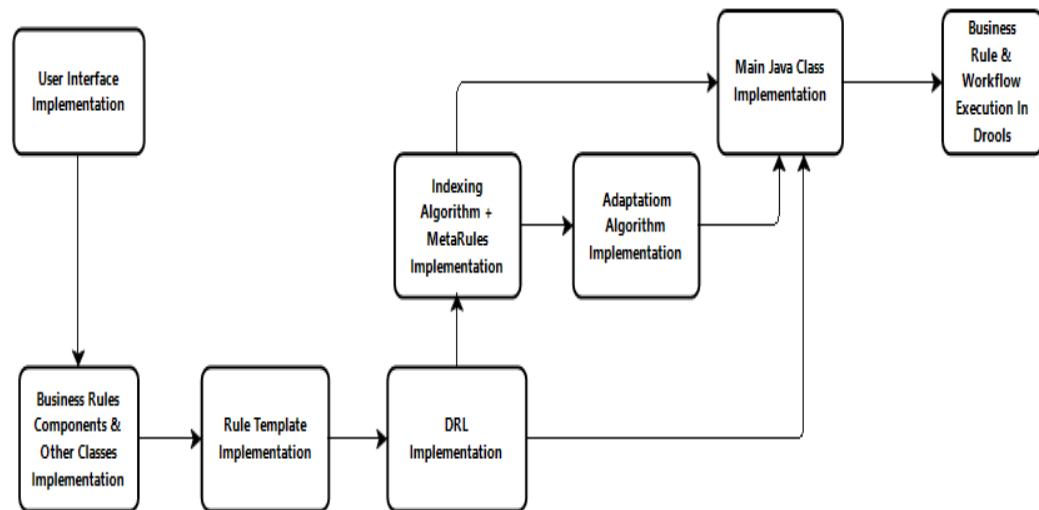


Figure 7.2. 2 Key Implementation Steps

### 7.2.1 Business Rule Classes

Implementation of the proposed model involves creation of several Java classes. Business rule components relations are implemented using the rule-inheritance concept. Mauricio in [105] discussed the idea of having a rule-hierarchy where rules allow inheritance between them. If a business rule R1 inherits business rule R2, R1's components inherit R2's components. Therefore, in this research the inheritance is implemented using dependencies between business rules that are formulated using objects and their properties. The properties are parameters of events, conditions and actions of a business rule. Each business rule component is implemented as an atomic class and created as a constructor of the Java Parent Class (ECAModel). In other words, the ECAModel class is made up of Event, Condition and Action classes. Every ECAModel is a node in the ECAGraph class. The ECAGraph class consists of lists of ECAModel (nodes), their properties and operations such as adding, removing and updating business rule components (nodes). In addition, the ECAGraph class provides the ability to define relationships between business rule components. The ECAGraph is executed directly in the main Java test class (ECAWorkflowTest). Table 7.2.1 displays the major classes and Figure 7.2.1.1 displays the UML class diagram of the major classes to show class information and relationships.

| Class Name             | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Event Class</b>     | An Event class is a blueprint from which business rule event objects are created. The Event class contains the declarations of the data that will be stored in each event object instance. Also contains declarations of methods that can be invoked using event objects. For example, in the Event class, we have variables (eventObject, eventObjectProperty, eventObjectProperyvalue, etc..) representing the data. Constructors and other regular methods have been defined to provide necessary operations for manipulation of event objects created as instances. Figures 8.3.1 shows the Event class and other related classes. Note, a single instance of Event class is created in ECAModel Class via the E_Component Class. The design makes it possible for implementation of an event which is part of the business rule (ECAModel object). The event can be used in multiple ECAModel class. |
| <b>Condition Class</b> | Condition class is a blueprint from which business rule condition objects are created. The Condition class contains the declarations of the data that will be stored in each                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |

|                                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                    | <p>condition object instance. Also, contains declarations of methods that can be invoked using condition objects. For example, in the Condition class, we have variables (conditionObject, conditionObjectProperty, conditionObjectPropertyvalue, etc) representing the data. Constructors and other regular methods have been defined to provide necessary operations for manipulation of condition objects created as instances. Figure 8.3.1 shows the Condition class and other related classes. Note, a single instance of Condition class is created in ECAModel Class via the C_Component Class. The design makes it possible for implementation of a condition, which is part of the business rule (ECAModel object). The condition class can be used in multiple ECAModel class.</p>                                                                                                                                                                                                                                                                                                                                      |
| <b>Act<br/>(Action) Class</b>      | <p>An Act class is a blueprint from which business rule action objects are created. The Act class contains the declarations of the data that will be stored in each action object instance. Also contains declarations of methods that can be invoked using action objects. For example, in the Act class, we have variables (actionObject, actionObjectProperty, actionObjectPropertyvalue, etc,) representing the data. Constructors and other regular methods have been defined to provide necessary operations for manipulation of action objects created as instances. Figure 8.2.1 shows the Act class and other related classes. Note, a single instance of Act class is created in ECAModel Class via the A_Component Class. The design makes it possible for implementation of an action which is part of the business rule (ECAModel object). The action can be used in multiple ECAModel class.</p>                                                                                                                                                                                                                     |
| <b>Fact class<br/>(Rack class)</b> | <p>Every business rule component has one or more associated facts against which they are fired. Facts are the data stored in working memory. An example might be a Rack fact object with utilization and capacity properties. Using facts, Drools identifies the matching business rules and performs the associated actions. The fact is instantiated dynamically using the getFactType method of the Knowledgebase. The getFactType method uses two parameters; the first one is the package name of the business rule where the fact was defined and the second one is the fact name. Note, the fact names are associated with components of a business rule. For example, consider business rule (R01) in Table 6.1, the value "Location" is a fact which is linked to a condition class. From Java standpoint, Facts are the POJO classes. To generate facts for business rule components, we defined POJO class using user defined data i.e. Rack, Equipment, Request, etc. Like any POJO class, we provide methods to set, get and others to manipulate fact values during rule execution. It is important to note that</p> |

|                              |                                                                                                                                                                                                                                                                                                                                                                                                              |
|------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                              | when a fact is changed or deleted, it does not change the value available in the business rule components. However, the change is made directly to a fact object in memory.                                                                                                                                                                                                                                  |
| <b>ECAModel class</b>        | The ECAModel class is a parent class made up of Event, Condition and Action classes. Every ECAModel object is a node in the ECAGraph class. Figure 8.3.1 shows the ECAModel's properties, methods and relationships                                                                                                                                                                                          |
| <b>ECAGraph class</b>        | The ECAGraph class as the name suggests is a graph class consists of lists of ECAModel (nodes), their properties and operations such as adding, removing and updating business rule components (nodes). In addition, the ECAGraph class provides the ability to define relationships between business rule components. The ECAGraph is executed directly in the main Java test class (ECAWorkflowTest.java). |
| <b>ECAWorkflowTest class</b> | The ECAWorkflowTest is the main java class containing the main method, which provides an entry point to the model prototype. The ECAGraph class is instantiated and executed directly in the main Java test class (ECAWorkflowTest.java).                                                                                                                                                                    |

Table 7.2. 1 Description of Core ECA Model Classes

To enhance the implementation of Fact classes, Java Spring framework implementation could be used to handle creation and deletion of various facts. The implementation of beans [103] in Spring is important to the use, allowing us to have Java fact classes that live within the application context without constantly creating new fact instances every time we need. Furthermore, the Spring framework can maintain the objects in the main memory effectively reducing the risk of running out of memory [97]. Spring works in a way that it finds most inactive or passive objects in the main memory then copies these to the secondary storage to create space for new objects.



## 7.2.2 Business Rule Template

Using Drools drools-templates API, rule template is implemented as a way of creating business rules and components in real time. Business rule component classes (Event, Condition, Action, etc) are parsed into a rule template which creates a DRL file. Typically, the structure and actual business rules are de-coupled. This means the same rule template can be used by different sets of business rules. Figure 7.2.2.1 displays the structure of a rule template. It contains special keywords to define a business rule name and mark different parts of the business rule component (@event, @condition and @action).

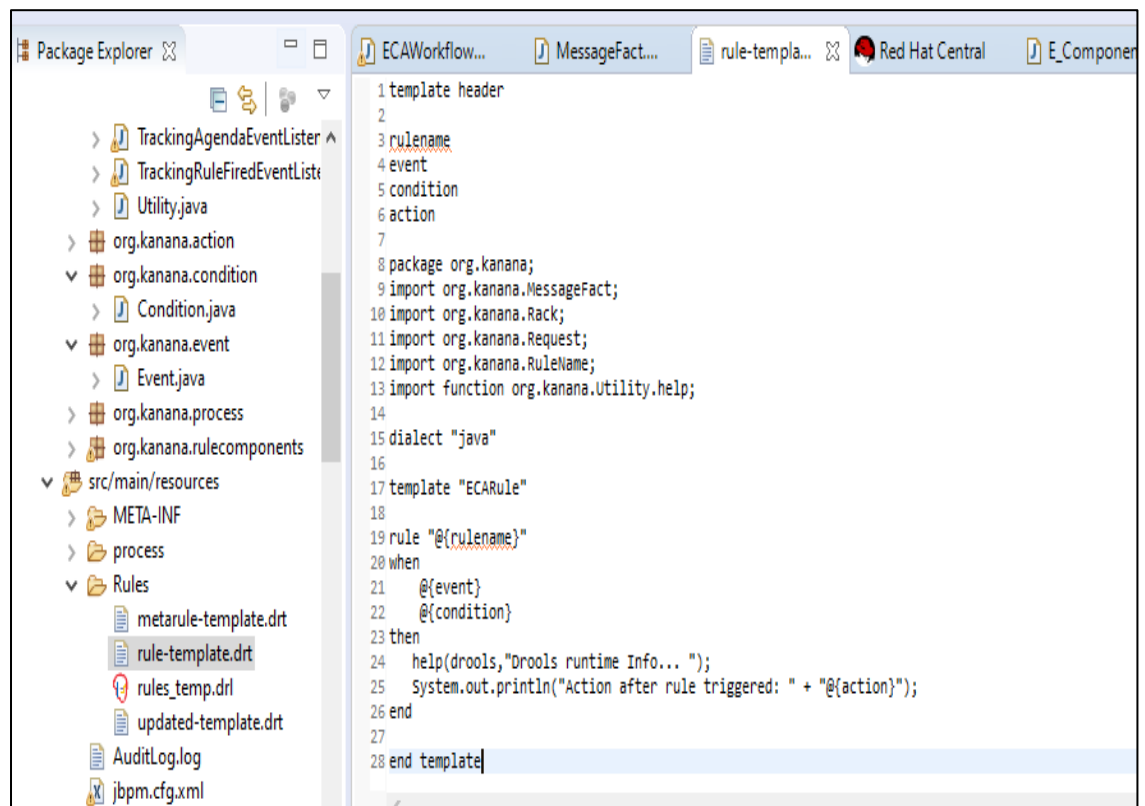


Figure 7.2.2. 1 Business Rules Template Structure

The Rule template provides necessary mappings from user input (Rule Designer) to a business rules format acceptable by the DRL. The implementation of the rule template is straight forward, the variable defined by using the syntax “@...” will be set as placeholders and substitution is done when value is passed from the main program. The rule template can contain multiple rule components to generate multiple business rules with different



structures. It is also possible to support other data sources, i.e. database SQL result sets and spreadsheets for multiple business rule import. It is worth to note the following:

- Line 1: Sets the DRL file to Drools “rule template”.
- Line 3 to 6: Sets rulename, event, condition and action to parameters.
- Line 8 to 13: Imports dependencies.
- Line 15: Provides Java syntax i.e. “System.out.println”
- Line 17: Defines name of the template
- Line 19, 21-22, 25: Variables `@{rulename}`, `@{event}`, `@{condition}` and `@{action}` are substituted with parameters at runtime.

To apply a business rule from the template is a matter of instantiating Drools `ObjectDataCompiler` and passing parameters as shown in Code Snippet 7.2.2.1

```
static private String applyECARuleTemplate(String ruleName, E_Component event,
C_Component condition, A_Component action) {

 Map<Object, Object> Ruledata = new HashMap<Object, Object>();
 ObjectDataCompiler objectDataCompiler = new ObjectDataCompiler();

 Ruledata.put("rulename", ruleName);
 Ruledata.put("event", event);
 Ruledata.put("condition", condition);
 Ruledata.put("action", action);

 return objectDataCompiler.compile(Arrays.asList(Ruledata),
Thread.currentThread().getContextClassLoader().getResourceAsStream("Rules/rule
-template.drt"));
}
```

Code Snippet 7.2.2. 1 ObjectDataCompiler for Rule Template (Java syntax)

### 7.2.3 Indexing Path Dependency Patterns

As described in Chapter 5, the graph data structure is introduced to hold a complete list of path dependency pattern indexes within a graph. Instances of `PathBasedPattern` class (Figure 7.2.3.1) are created to represent pattern indexes. A pattern index is identified by a group of nodes that form a graph dependency pattern (path dependency) mapped by such

a Pattern Index. Each Pattern Index can map one or more graph dependency patterns. Thus, the class PathBasedPattern holds information about IndexName, IndexID, PatternName, IndexPatternID and other methods (add, remove and modify) to support the functionality and manipulation of indexes. It also holds a root for the graph Root Index graph structure as discussed in Chapter 5.

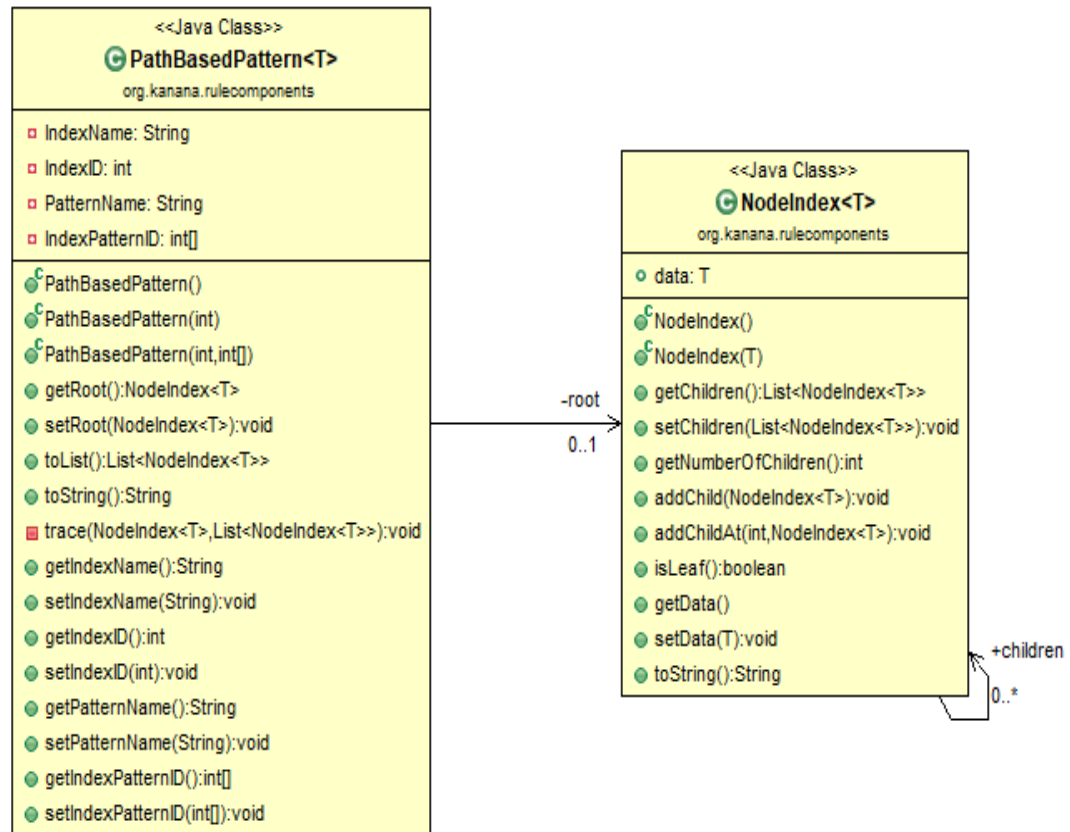


Figure 7.2.3. 1 UML Class Diagram for ECAIndexPatternGraph Class

Code Snippet 7.2.3.1 presents a method (IndexingGraphPatterns) that is used to create indexes for the Path dependency pattern. The method creates an instance of the PathBasedPattern class described above. To traverse through Path dependency pattern nodes (indexes), the “printPathAlgorithm” method part of the ECAGraph class uses Depth First Search Graph Algorithm to display root to leaf path nodes. In addition to other functionalities such as add, remove and modify supported by the PathBasedPattern class,

a Metarule construct (described in Chapter 5) is used to provide runtime modification indexes. Please refer to Appendix V for the LevelBasedPattern method.

```
//Creation of indexes for dependency patterns
public List<ECAModel> IndexingGraphPatterns(ECAGraph ecaRuleG){
//variables declaration
 int UniqueIndex = 0;
 List<ECAModel> DependencyPatterns = new ArrayList<ECAModel>();
 ECAModel rootcomponent = new ECAModel();
 Queue<ECAModel> queue;
 //create pattern index instance
 PathBasedPattern<ECAModel> pattenIndeces = new
 PathBasedPattern<ECAModel>(UniqueIndex, rootcomponent);
 //loop through graph to identify dependency patterns
 for (int index=0; index < ecaRuleG.ruleRelations.size();index++){
 rootcomponent = ecaRuleG.getNode(index);
 //Check the root rule
 if (rootcomponent == null) return null;
 //Create an empty stack and push the root rule to it
 Stack<ECAModel> nodeStack=new Stack<ECAModel>();
 nodeStack.push(rootcomponent);
 rootcomponent.visited=true;
 //Create a map to store parent pointers of graph nodes
 HashMap<ECAModel,ECAModel> parent =
 new HashMap<ECAModel, ECAModel>();
 //Parent of root is NULL
 parent.put(rootcomponent,null);
 //Traverse through Path Dependency Pattern then generate indexes
 while (!nodeStack.isEmpty()) {
 //Pop the top item from stack
 ECAModel current = nodeStack.pop();
 if(current.hasChildren()) {
 //Convert to object array
 ECAModel[] temppatterns = new
 ECAModel[current.children().size()];
 //ArrayList to Array Conversion to allow generation
 //of indexes for each path
 for (int pindex=0; pindex <
 current.children().size();pindex++){
 temppatterns[pindex] = current.children().get(pindex);
 for (ECAModel linkedIndex : temppatterns) {
 //pattenIndeces contains index for the ECA component
 node //and pattern indexes (combining linked ids)
 pattenIndeces = new
 PathBasedPattern<ECAModel>(pindex,linkedIndex);
 }
 //Create indexes for path dependency
 DependencyPatterns.add(new
 ECAModel(RuleList.get(pindex).getruleName(),
 current, pattenIndeces));
 }
 }
 }
 }
 return DependencyPatterns;
}
```

Code Snippet 7.2.3.1 1 Indexing Method for Path Dependency Patterns

## 7.2.4 Change Propagation and Adaptation Algorithms

This Section is divided into two. The first part presents the implementation aspect of the business rule component change propagation and the second part presents the implementation of business rules adaptation in a workflow.

### 7.2.4.1 Business Rules Change Propagation Algorithm

The “ECACHangePropagation” method (Code Snippet 7.2.4.1) has been implemented to support the business rule component change propagation algorithm exemplified in section 6.3.

```
//Business Rule Change Propagation Algorithms: component is changed and
propagated //across dependency patterns listed in DependencyPatterns
//Input: business rule component
public void ECACHangePropagation(ECAModel changedECAcomponent, ECAGraph
ecaRuleG){
 //Declare variables
 //Variable to store sorted/indexed dependency pattern so
 //it's easy to propagate the change.
 //Note calling IndexingGraphPatterns(ecaRuleG) provide different dependency
 patterns //(Path, Level, Direct-Node and Neighbours dependencies
 List<ECAModel> DependencyPatternsIndexes =
IndexingGraphPatterns(ecaRuleG);
 ECAModel changedRulecomponent = new ECAModel();
 ECAModel changedLinkedRulecomponents = new ECAModel();

 //Check if the business rule component existing in the using
 //DependencyPatternsIndexes
 //loop through graph to identify dependency patterns
 for (int index=0; index < DependencyPatternsIndexes.size();index++){
 changedRulecomponent = changedRulecomponent.children().get(index);

 //Using rule component index list (DependencyPatternsIndexes)
 //to check if component exist
 if (changedRulecomponent == null) return;
 if (changedRulecomponent == changedECAcomponent) {

 //check if changedRulecomponentIndex has children
 if (changedRulecomponent.hasChildren()) {

 //get the index of the business rule component to be updated
 int changedRulecomponentIndex = changedRulecomponent.getruleIndex();

 //Using the changeRulecomponentIndex to propagate the change
 //to all dependency business rule components (children)
 //Perform delete and add to propagate the change to children components
 changedLinkedRulecomponents.children().remove(changedRulecomponentIndex)
 changedLinkedRulecomponents.children().add(changedRulecomponentIndex,
 changedECAcomponent);
 }
 }
 }
 }
}
```

Code Snippet 7.2.4. 1 Business Rule Change Propagation Algorithm

### 7.2.4.2 Business Rules Adaptation Algorithm

Three methods have been implemented to support the adaptation of business rules in workflows. The first method, `CreateProcessFromBusinessRules` (Code Snippet 7.2.4.2.1) implements the transformation of business rules into processes. The second method, `BusinessRulesDependency` (Code Snippet 7.2.4.2.2) implements business rule relationships (dependencies) as per Table 6.5. Dependency is based on Object and Properties of one rule being matched using objects and properties of one or more rules. The third method, `CreateWorkflowFromBusinessRules` (Code Snippet 7.2.4.2.3) implements the execution of business process connections using Drool's APIs.

```
//Phase 1: Converting and Transforming business rules into processes
//Input indexed ECA rule list then return processes
ArrayList<Process> CreateProcessFromBusinessRules(List<ECAModel> RuleList) {

 //Use business rules to define business processes: start,
 //intermediate and end processes
 ProcessList=new ArrayList<Process>();
 for (int index = 0; index < RuleList.size(); index++) {
 //if a rule's predecessor is null then create start node
 List<ECAModel> ecaRuleList = RuleList;
 if (ecaRuleList.get(index).isRoot()) {
 StartNode pStart = new StartNode();
 pStart.setId(index); //Note index of rule assigned used in merged phase
 pStart.setName("P" +index);

 //Define process definition based on Action Nodes
 Process WF_pStart = new Process(pStart);
 WF_pStart.setStartprocess(pStart);
 ProcessList.add(WF_pStart);
 }

 //if a business rule's predecessor or successor
 //is not null then create intermediate node
 if (ecaRuleList.get(index).hasChildren()
 && ecaRuleList.get(index).isRoot()== false) {
 ActionNode p1 = new ActionNode();
 //Note index of a rule assigned to be used in the merged phase
 p1.setId(index);
 p1.setName("P" + index);
 DroolsAction action = new DroolsAction();
 action.setMetaData("Action", new Action() {

 public void execute(ProcessContext context) throws Exception {
 System.out.println("Error define process node");
 }
 });
 p1.setAction(action);
 }
 //Define process definition based on Action Nodes
 }
}
```

```

Process WF_p = new Process(p1);
WF_p.setProcess(p1);
ProcessList.add(WF_p);

}
//if a rule's successor is null the create end node
if (ecaRuleList.get(index).hasChildren() == false) {
 EndNode pEnd = new EndNode();
 //Note index of a rule assigned to be used in the merged phase

 pEnd.setId(index);
 pEnd.setName("P" + index);

 //Define process definition based on Action Nodes
 Process WF_pEnd = new Process(pEnd);
 WF_pEnd.setEndprocess(pEnd);
 ProcessList.add(WF_pEnd);
}
}
return ProcessList;
}

```

Code Snippet 7.2.4.2. 1 Convert Business Rules into Processes Algorithm

```

//Phase 2: Define Business Rule dependency graph as per Table 6.5
//Dependency is based on Object and Properties of one rule are matched using
objects //and properties of one or more rules
//It takes in the ECA AND-OR Graph parameter as a template
ECAGraph BusinessRulesDependency(ECAGraph ecaRuleG) {

//Create relationships using the indexed RuleList then build the graph
for (int i = 0; i < RuleList.size(); i++) {
 String eventList1 = (RuleList.get(i).getevent()).toString();
 System.out.println("Check object 1" + eventList1);
 for (int j = i + 1; j < RuleList.size(); j++) {
 RuleList.get(j).getcondition();
 RuleList.get(j).getaction();
 String eventList2 =
 (RuleList.get(j).getevent()).toString();
 String condList2 =
 (RuleList.get(j).getcondition()).toString();
 String actionList2 =
 (RuleList.get(j).getaction()).toString();
 System.out.println("Check object 2" + eventList2);

 /*Event-AND Relationship scenario, note Condition and
 Action components are not shown here but they implemented
 in similar manner. Obtain the list of rules - RuleList.
 Object and properties of one rule's Event relates to event
 and condition and action of another rule's object and
 properties. if event component of one rule matches all:

```

```

 event, condition, action components of the second rule
 then create relationships between rules*/

 if (eventList1.contains(eventList2) &&
 eventList1.contains(condList2) &&
 eventList1.contains(actionList2)) {
 rulecomponentgrap
 = CreateRuleDependency(ecaRuleG, i, j, RuleList,"E");
 }
 /*Event-OR Relationship. Object and properties of rule 1
 Event relates to either event or condition or action of
 another rule's object and properties. if equal then create
 relationships between rules*/

 if (eventList1.contains(eventList2)) {

 rulecomponentgrap
 = CreateRuleDependency(ecaRuleG, i, j, RuleList,"E");
 }
 if (eventList1.contains(condList2)) {

 rulecomponentgrap
 = CreateRuleDependency(ecaRuleG, i, j, RuleList,"C");
 }
 if (eventList1.contains(actionList2)) {

 rulecomponentgrap
 = CreateRuleDependency(ecaRuleG, i, j, RuleList, "A");
 }

 }

 return rulecomponentgrap;
}

//The following method is used by" BusinessRulesDependency " for creation of
//business rules dependencies source/destination
ECAGraph CreateRuleDependency(ECAGraph ecaRuleG, int Rulesrc, int Ruledest,
List<ECAModel>RuleList, String ComponentType){
 //Define a variable to hold the created business rule relationship graph
 ECAGraph ecaRuleRelationshipsGraph;

 //Create relationship between source and destination
 ecaRuleG.ruleRelations.get(Rulesrc).add(
 RuleList.get(Ruledest).getruleName());
 ecaRuleRelationshipsGraph = ecaRuleG;

 //Note: Once the business rules graph is defined, we can easily identify
 //predecessors, successors, rule flow constructs and relation operators
 as //discussed in Phase 2 of adaptation ready for Phase 3
 return ecaRuleRelationshipsGraph;
}

```

Code Snippet 7.2.4.2. 2 Build Dependency Graphs Algorithm

```

//Phase 3: Creation of business process connections (process dependency graph)
and //use of Drool's APIs to execute the workflow. Using process list and
business rules //graph generated from Phase 1 and 2
RuleFlowProcess CreateWorkflowFromBusinessRules(List<ECAModel> RuleList) {

 //Create a template for the ECAWorkflow instance
 RuleFlowProcess ruleworkprocess = new RuleFlowProcess();
 ruleworkprocess.setId("ECAWorkflow");

 //Define RuleProcessList to create process list
 List<Process> ProcessList = new ArrayList<Process>();
 List<ECAModel> RuleProcessList = new ArrayList<ECAModel>();
 //variables to hold rule component nodes
 ProcessList = CreateProcessFromBusinessRules(RuleList);

 //loop through ProcessList
 for (int j = 0; j < ProcessList.size(); j++) {
 rootProcess = new ECAModel();
 //loop through connected business rule graph (source and destination)
 for (int index=0; index
 rulecomponentgrap.ruleRelations.size();index++){
 //build workflow based on process and rules graph
 if (j == index) {
 rootProcess = rulecomponentgrap.getNode(index);
 //Check the root rule
 if (rootProcess == null) return null;
 //Create an empty stack and push the root rule to it
 Stack<ECAModel> nodeStack=new Stack<ECAModel>();
 nodeStack.push(rootProcess);
 rootProcess.visited=true;
 //Create a map to store parent pointers of tree nodes
 HashMap<ECAModel,ECAModel> parent
 =new HashMap<ECAModel, ECAModel>();
 //Parent of root is NULL
 parent.put(rootProcess,null);
 while (!nodeStack.isEmpty()) {
 //Pop the top item from stack
 ECAModel current = nodeStack.pop();
 //Top to Bottom path
 if (current.hasChildren()) {
 //Use the ProcessRuleList to build relationships between
 //processes from rules
 RuleProcessList.add(new
 ECAModel(RuleList.get(index).getruleName(), current,
 parent));
 }
 }
 }
 }
 //Using Drools APIs to build workflow form RuleProcessList which
 contains //source and destination processes
 for (int i = 0; i < RuleProcessList.size();i++) {
 if (RuleProcessList.get(i).getfromProcess().equals(rootProcess) && i==0)
 {

```



```

 newConnectionImpl(
RuleProcessList.get(i).getfromProcess().getStartprocess(),
"DROOLS_DEFAULT",
RuleProcessList.get(i).gettoProcess().getProcess(),
"DROOLS_DEFAULT");
//Adding nodes to the workflow
ruleworkprocess.addNode((Node) ((ProcessInstance)
rootProcess).getProcess());
 }
}
return ruleworkprocess;
}

```

Code Snippet 7.2.4.2. 3 Generate Workflow Algorithm

### 7.2.5 Business Rules Editor (ECA Model Test Client)

So far, we have covered the implementation of core ECA Model concepts (Section 7.2.1) and important algorithms (Section 7.2.3 and 7.2.4). The next section introduces our implementation of ECA Model Test Client (Figure 7.2.2.1) developed as part of this research to allow non-technical users to create, delete and update business rule components, as well as displaying the results of business rules in DRL file, business rule dependencies and execution of rules to a working workflow.

The ECA Model Test Client is a graphical visual interface or editor for managing business rules' input and output. The interface allows users to create, modify and delete business rules and components and, also provides the ability to test for change propagation and workflow adaption in the “Display Statistics” section of the editor. Furthermore, it provides a direct interface to Drools engine to allow users to execute business rules and processes on the fly.

The ECA Model Test Client incorporates a form for adding, modifying and deleting components (event, condition and action). The form fields are clearly labelled to allow users to enter information that pertains to the rule components in their requirements. The section at the top left below “*Enter Rule Name*” section is for adding new rule components (highlighted in yellow) and the section on the right is for modifying and deleting rule components (highlighted in pink). The checkboxes “*Create Rule*”, “*Modify Rule*” and “*Delete Rule*” must be checked depending on the on the operation to be performed. The

number of button components exit at the bottom of the editor so that a user clicks to trigger a specific function. For example, the “*Add Rule*” button adds new business rule component to Drools rule repository; the “*Change Rule*” button deletes or modifies business rule components depending on the selected checkbox option. The “*Execute Rules*” will fire the business rules and map business rules to processes for adaptation. The “*Display All*” button displays the result in the display areas.

The ECA Model Test Client is developed using the graphics classes through the Java Swing package. The Java Swing package provides Java Graphics APIs for constructing Graphical User Interface (GUI) applications. The APIs allow the creation of components such as window, buttons, checkboxes, text areas, text fields, panels, etc. These components are used to get input and output

Figure 7.2.5. 1 ECA Model Test Client

Code snippet 7.2.5.1 presents the implementation of “*Add Rule*” button when an action is performed. Business rule components (event, condition and action) are added to the rule template.

//The following method implements different actions performed by user using buttons from the ECA Model Test Client

```
public void actionPerformed(ActionEvent e) {

 //Define different variables
 ObjectDataCompiler converter = new ObjectDataCompiler();
 Map<Object, Object> Ruledata = new HashMap<Object, Object>();
 final Map<Object, Object> Ruledata2 = new HashMap<Object, Object>();

 InputStream template =
 Thread.currentThread().getContextClassLoader().getResourceAsStream("Rules/rule-template.drt");

 //rule field
 String ruleNameProperty_b = ruleNameTxt.getText();

 //Pass Boolean to check if rule is empty to continue or not
 boolean pass = true;

 //event, condition, action fields setup
 //Adding rule and components to the list entered by user from UI
 RuleList.add(new ECAModel(r1.getRuleName(), event, condition1, act1));
 RuleListUpd.add(new ECAModel(r1.getRuleName(), event, condition1, act1));

 int RuleNodeCount = RuleList.size();

 //Initialize the graph structure
 ECAGraph structuredGraph = new ECAGraph(RuleNodeCount);
 ECAGraph eca_graph = new ECAGraph(RuleListToComponetRuleList(RuleList));

 //When the create button is pressed, we take data from text fields
 //and output to an array.
 if(e.getSource() == ECAButton) {

 if (ruleNameProperty_b.equals("")) {
 System.out.println("Error: Enter Rule data.");
 pass = false;
 }
 //If passed, the program continues
 if (pass == true) {
 //Checking if rule already exists
 if (RuleList.contains(r1.getRuleName())) {

 System.out.println("Error: Rule exists, use another name.");
 }
 else {

 //loop and add data to the HashMap Ruledata
 for(int index = 0; index < RuleList.size();index++) {
 System.out.println("");
 }
 }
 }
 }
}
```

```

 Ruledata.put("rulename", RuleList.get(index).getruleName());
 Ruledata.put("event", RuleList.get(index).getevent());
 Ruledata.put("condition", RuleList.get(index).getcondition());
 Ruledata.put("action", RuleList.get(index).getaction());
 RuleList.add(Ruledata);
 }
 //Remove duplicate record from Rulelist arraylist
 for(int i = 0; i < RuleList.size(); i++) {
 for(int j = i + 1; j < RuleList.size(); j++) {
 if(RuleList.get(i).equals(RuleList.get(j))) {
 RuleList.remove(j);
 j--;
 }
 }
 }

 //Node based (Object and Property) Dependency Algorithm
}
}
drl = converter.compile(RuleList, template);

System.out.println("Displaying Original DRL (Rule Template)... ");
System.out.println(drl);
}
}

```

Code snippet 7.2.5. 1 Demonstrate “Add Rule” Swing Button

Figure 7.2.5.2 presents a print screen after the user clicked the “Add Rule” button. In “Display Drools DRL - Rule Template” text area; it shows business rule ‘R1’ and components added in DRL.

The screenshot displays the 'ECA Model Test Client' window. At the top, 'Enter Rule Name:' is set to 'R1'. Below this are three columns for defining rule components:

- New-EVENT:** Event Object: Request, Event Property: Type, Event Value: INSTALL, Operator: ==.
- New-CONDITION:** Condition Object: Rack, Condition Property: RackUtilization, Condition Value: 2000, Operator: >.
- New-ACTION:** Action Object: Process, Action Property: Name, Action Value: Order Rack, Operator: ==.

Below these is a 'Define Rule Relationships' section with 'Enter Source Rule:' and 'Enter Destination Rule:' fields, both set to 'Event'. At the bottom are buttons for 'Add Rule', 'Add Relation', and 'Execute'.

The main area contains two text editors:

- Display Drools DRL - Rule Template:** Contains the following DRL code:
 

```

package org.kanana;
import org.kanana.Request;
import org.kanana.RuleName;
import function org.kanana.Utility.help;
dialect "java"
rule "R1"
when
 Request(Type) == 'INSTALL'
 Rack(RackUtilization) > Rack(RackCapacity)
then
 help(drools,"Drools runtime Info... ");
 System.out.println("Action after rule triggered: " +
 "Process(Name == 'Order Rack')");

```
- Display Drools DRL - Updated-Rule Template:** Currently empty.

Figure 7.2.5. 2 Business Rule and Components Insertion

To avoid repetition, all functionalities of the ECA Model prototype are covered in the experimentation section (Section 8.3). However, below are steps to add, delete and update business rules and components are explained in Appendix VII.

### **7.3 *Summary***

This Chapter focused on the development aspect of the model's prototype. It described the implementation of various Java classes and algorithms for defining model concepts, rule indexing, change propagation and rule adaptation of business rules in a workflow environment. The prototype model with the ECA Model Test Client is developed on Drools environment providing an added value of integration with a rule engine and software platform for intelligent process automation. With the ECA Model Test Client, users can create, modify and delete business rules at runtime.

## 8. ECA Model Validation

This chapter discusses the validation process of the ECA Model, reviewing, analysing and validating issues that this research is attempting to resolve and using use cases to perform experiments that are based on research objectives as specified in Chapter 1. Experiments are carried out using the ECA Model prototype introduced in Chapter 7. A summary matrix table (Table 8.1.1) displays a breakdown of activities, problems, validation criteria and experiments performed. Table 8.1.1 helps to determine what objectives and experiments are being undertaken. Unless stated otherwise, business rules components always mean event, condition and action.

| Problems/Challenges                                                                                                                                                     | Validation Criteria                              | Activities                                                                                  |                                                                                                       |                                                                                                                     |                                                                                                                                                  |                                                                                                                                                  |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------|---------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                                                                                                                                         |                                                  | Creation, modification and deletion of business rules & components<br><b>Objective 5(a)</b> | Automatic generation of business rules components relationships/dependencies<br><b>Objective 5(b)</b> | Change propagation during insertion, modification and deletion of business rule components<br><b>Objective 5(c)</b> | Adaptation of business rules in a workflow environment to control the creation and termination of a process in workflow<br><b>Objective 5(d)</b> | Adaptation of business rules to control the processes (Sequential, AND/OR Parallel Split, AND/OR Merged) in a workflow.<br><b>Objective 5(d)</b> |
| Complexity of creation/modification/deletion of business rules structure at component level due to lack of high-level abstraction                                       | Adaptation and Flexibility                       | Experiments (1-3)                                                                           |                                                                                                       |                                                                                                                     |                                                                                                                                                  |                                                                                                                                                  |
| Complexity of creation/modification/deletion of business rules and components by non-technical users                                                                    | Usability                                        | Experiments (1-3)                                                                           |                                                                                                       |                                                                                                                     |                                                                                                                                                  |                                                                                                                                                  |
| Complexity of generating and determining business rules components relationships                                                                                        | Performance and Storage                          |                                                                                             | Experiment (1-3)                                                                                      |                                                                                                                     |                                                                                                                                                  |                                                                                                                                                  |
| Difficulty in propagating changes on related business rules at component level (slow performance as applied business rule change may take longer, decentralised manner) | Simplicity<br>Usability<br>Accuracy<br>Efficient |                                                                                             |                                                                                                       | Experiments (1-3)                                                                                                   |                                                                                                                                                  |                                                                                                                                                  |
| Providing support to automatic control the flow of processes in a workflow using business rules and components                                                          | Flexibility and Adaptation                       |                                                                                             |                                                                                                       |                                                                                                                     | Experiments (4)                                                                                                                                  | Experiments (5-9)                                                                                                                                |

Table 8.1. 1 Research Objectives, Challenges, Validation Criteria and Experiments

The chapter is divided into four sections. Section 8.1 introduces the validation criteria. Section 8.2 presents the nine examples/use cases or scenarios for the demonstration of business rules change management, propagation and adaptation requirements. Section 8.3 describes the actual experiments conducted using the ECA Model prototype. Section 8.4 concludes with a summary.

## **8.1 Validation Criteria**

In terms of testing, validation criteria are defined to introduce metrics for quantifying of our results. The validation criteria are based on aims and objectives of the research. We recall that the key research aims, and objectives of the proposed model were introduced in Chapter 1. The proposed business rule model aims at reconciling the three main concerns of the business rules change management, which are real time change of business rules and components, the change propagation, which is the effect of change on related business rules and the adaptation of business rules management in workflows. The primary validation objective is to measure both the integrated solution for managing dynamic business rule components (event, condition, action) and managing the adaptation of business rules to handle and control specific business process instances of a workflow. As such, it is beneficial to revisit the objectives (5a-5d) stated in section 1.3, concerning the development of the proposed model prototype. The following validation checks (Table 8.1.2) are considered to help with the assessment of these objectives:

| <b>Validation Checks/Criteria</b>                                                     | <b>Description</b>                                                                                                                                                                                                                                 |
|---------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Ability to add business rules and components (event, condition, action) on the fly    | This check is intended to assess the impact of creation of business rules and components at runtime; how the proposed model improves the quality of abstraction and adaptation of business rules and components when business requirements change. |
| Ability to modify business rules and components (event, condition, action) on the fly | This check is intended to assess the impact of modification of business rules and components at runtime, how it improves the quality of abstraction and adaptation of business rules and components when business requirements change.             |
| Ability to delete business rules and components (event, condition, action) on the fly | This check is intended to assess the impact of deletion of business rules and components at runtime, how it improves the quality of abstraction and                                                                                                |

|                                                                                                                                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                                                                                                                 | adaptation of business rules and components when business requirements change.                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| Check Agility/Flexibility of business rules components                                                                                          | <ul style="list-style-type: none"> <li>- Are business rules and components (event, condition, action) configurable to support real-time modification? Or every time a change occurs, a developer will be involved to change and recompile the code, less risks for unnecessary downtime.</li> <li>- Is the proposed model allowing users to gain insight into which business rules and components (event, condition, action) are changed and executed?</li> <li>- Is it configurable (otherwise you'd just code it instead)?</li> </ul>                  |
| Check change propagation (Accuracy, Usability and Simplicity): The interaction in particularly with chained business rules                      | <ul style="list-style-type: none"> <li>- Is the change being propagated across related rules and components?</li> <li>- Change propagation is required as part of change management. For typical rule applications, technical users are not primary requesters of the changes and Usability and Simplicity is important validation criteria to reduce the time spent by technical users for change propagation as non-technical users are able to make the change and, in some cases, minimizes the cost of change management or maintenance.</li> </ul> |
| Check adaptability of ECA rules within a workflow: Demonstrating the feasibility of automating business processes through use of business rules | <ul style="list-style-type: none"> <li>- Can a business rule control the execution of workflow processes?</li> <li>- Are business rules able to control the initiation of processes?</li> <li>- Are business rules able to control the termination processes?</li> <li>- Are business rules able to control the running of sequential processes?</li> <li>- Are business rules able to control the running of parallel/split and merge of processes?</li> </ul>                                                                                          |
| Check usability: Usefulness and ease of use                                                                                                     | <ul style="list-style-type: none"> <li>- How easy for users to change rules and their components</li> <li>- How easy is it for non-technical people to change rules?</li> </ul>                                                                                                                                                                                                                                                                                                                                                                          |
| Check design efficiency                                                                                                                         | - Does the ECA Model improve the design process of business rules and their components? (Business rules abstraction).                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| Check time efficiency                                                                                                                           | - Is it performant? Does the model shorten business rules execution time as a result of using AND OR Graphs and indexing structures (organizing rules and reducing the number of rules that need to be matched for execution at a given point and time)?                                                                                                                                                                                                                                                                                                 |
| Check data quality                                                                                                                              | - Correctness of the data after business rule change                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |

Table 8.1. 2 Validation Criteria Description

The validation checks are applied when testing the ECA Model. These checks are essential to demonstrate that the developed prototype is fit for the intended purpose. Use cases are



derived (section 8.2) in order to experimentally test the proposed model against the checks in Table 8.1.2.

## 8.2 Data Centre Use cases

The previous section has already explored validation checks, which clearly formulate criteria to be used to test the proposed model. In addition, this section presents useful use cases from data centre environment to illustrate how the proposed ECA Model is validated in a practical way. Due to dynamic nature of the business rules and lack of similar case studies that focus on business rules components change and propagation, it was decided to use a predictive validation technique. This technique allows for complex business rules and process management application scenarios to be used to simulate data needed for the validation process.

Data centre (Figure 8.2.1) operations are usually complex and constantly changing. Daily equipment is installed, decommissioned, moved and modified. Workflow applications are used to manage and track changes in an orderly manner as well as help data centre teams to optimize operations to get the highest efficiency and productivity. Naturally, the operations on data are implemented using business rules. Changes applied to the data and operations of a workflow are controlled by changes made on business rules.

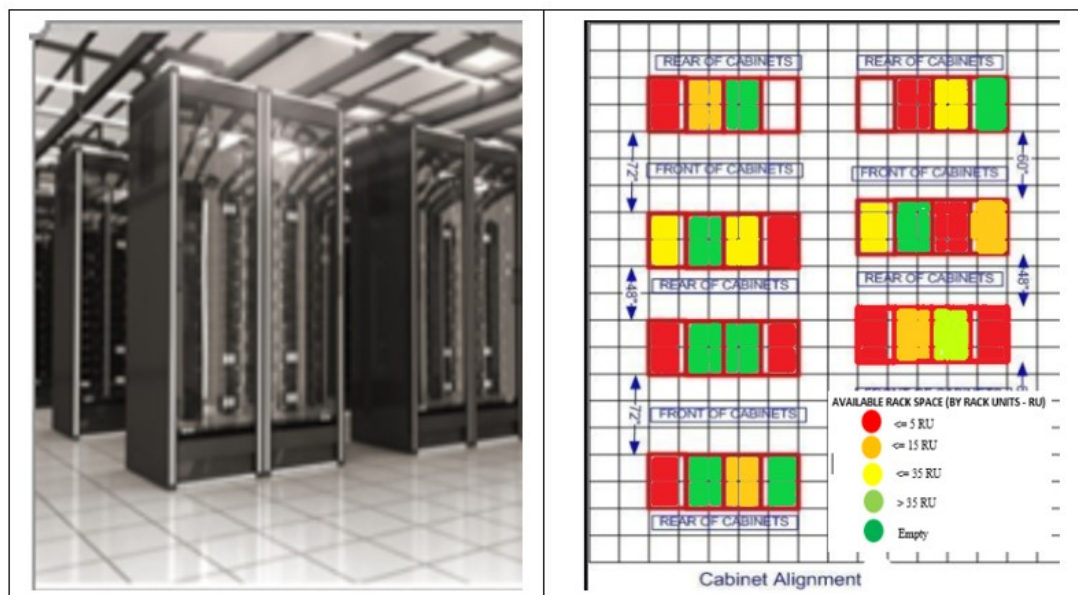


Figure 8.2. 1 DC Floor Plan with Equipment Installed

The next part of this section presents the simulated use cases to demonstrate our model's three key areas, business rules and components change, business rule change propagation as well as business rules adaptation in a workflow. The reader is advised that only a subset of business processes, and rules are selected from various data centre workflow scenarios for demonstration purposes. There are several scenarios that can be considered, however due to time constraints, only nine classifications are covered below (Table 8.2.1). Appendix I includes a list of possible test scenarios that this research could consider for future experimentation.

| Use case No | Use case Name                                                      |
|-------------|--------------------------------------------------------------------|
| Use case #1 | Adding business rules components and propagating the change        |
| Use case #2 | Updating business rules components                                 |
| Use case #3 | Deleting business rules and components                             |
| Use case #4 | Enabling business rules to initiate and terminate business process |
| Use case #5 | To show Sequential flow patterns for workflow adaptation           |
| Use case #6 | To show Parallel-OR Merged flow patterns for workflow adaptation   |
| Use case #7 | To show Parallel-AND Merged flow patterns for workflow adaptation  |
| Use case #8 | To show Parallel-OR Split flow patterns for workflow adaptation    |
| Use case #9 | To show Parallel-AND Split flow patterns for workflow adaptation   |

Table 8.2. 1 Use cases from DC Workflow

### 8.2.1 Add Business Rules Components and Propagate Change

#### Use case #1:

Consider a growing company (XYZ Ltd) that realized the need to add new business rules in its workflows to meet new and changing business requirements to accurately perform its data centre operations such as installation, decommission and move of equipment. As XYZ acquires new data centres, new business rules are added to its workflows. This means, XYZ needs to be able to not only integrate new business rules but also the ability to propagate the change to existing rules in the workflow. A typical implementation process, the technical experts/developers are employed to reconfigure the workflow to add new business rules, costing the company money and time. Generally, when a new data centre is added, equipment such as cabinets/racks, servers, power distribution units, power panels,

generators, circuit breakers, switches, network cards, etc., are installed. A requestor fills out a request form, which defines relevant information including the preferred location and other equipment requirements such as type, manufacturer details, power and network configurations. The form (request) can contain several pieces of equipment to be installed. The form captures data, processes and business rules to generate a company equipment installation workflow. When new business rules are added, the reconfiguration process is often slow and complicated. It takes days to get the workflow code updated to include new business rules and to ensure that the changes are propagated across related business rules. Below business processes and rules are recorded for XYZ installation workflow. Appendix II presents the description for each business rule in more detail. Figure 8.2.2.1 presents the XYZ data centre equipment installation workflow.

**Workflow Name:** Equipment Installation

**Roles:** Requestor, Reviewer/Approver, Data Centre (DC) Manager,  
Power and Network Technicians (Tech)

**Business Processes:**

- P1 - Create Request
- P2 - Review Request
- P3 - Approve Request
- P4 - Manage Rack Space
- P5 - Manage Data Centre Space
- P6 - Order Rack
- P7 - Install Equipment
- P8 - Provision Power
- P9 - Provision Network
- P10 - Provision Network Cables
- P11 - Completing Power and Network Provisioning
- P12 - Close Request

**Business Rules (R0 - R12):**

- When workflow start activity then create new process 'Create Request - P1' (R0)

- When submit request, if requestor is a member of the Platform capacity team then go to ‘Review’ step (R1) else go to ‘Approve’ step for data centre area manager to approve and set install request (R2)
- When install request, if rack utilization is greater than the rack space capacity, then count installed equipment and set the Rack is full and set process to manage data centre space (R3) else install the equipment in the available rack (R4)
- When Rack is full, set total no of racks to ‘installed Racks’ in the data centre (R5)
- If total number of racks is less data centre rack capacity, then order new rack (create new Order Rack process) (R6)
- If number of equipment power supplies is greater than zero, then set process name to provision power (R7)
- If number of equipment network ports is greater than zero, then set process name to Provision Network (R8)
- If number of equipment power connections is equal to equipment power supplies, then set process name to Completing Power and Network Provisioning (R9)
- If equipment network cable is required then set Process to Provision Network Cables (R10)
- If equipment network connections and cables are configured, then set process name to Completing Power & Network Provisioning and request status is set to close (R11)
- If request status is set to close, then set process name to close request (R12)

The equipment installation workflow (Figure 8.2.2.1) is broken down to a series of processes (tasks), some of which may or may not be enforced by business rules. Both the processes and business rules can be identified reasonably well, i.e. P1 represents the process “Create Request” and R3 represents the rule “When submit request, if requestor is a member of the Platform capacity team then go to ‘Review’ step”. Note that dependencies exist between business rules components, for example existing business rule (R2) action and new business rules (R3) and (R4) events. The later business rule depends on the former business rule’s actions to evaluate to true.

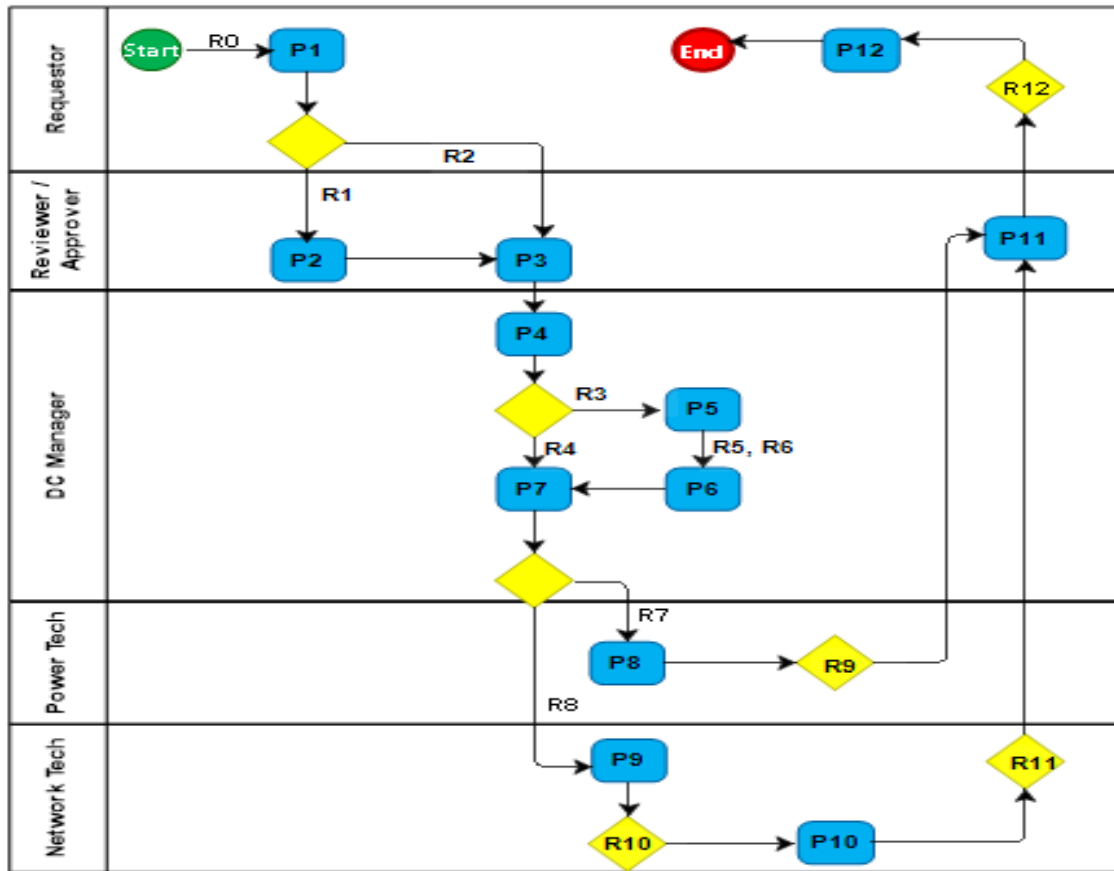


Figure 8.2.2. 1 XYZ Equipment Install Workflow

Using the proposed model, business rules statements from Use case#1 are formalized into business rules components. They are formatted in a way that makes it easy to be used as the bases for implementing them in business rules management systems. They are expressed in a simple way, so that it is easy to identify what part is an event, a condition or an action. This is valuable, especially to avoid inconsistent syntax when using ambiguity English like statements. In Table 8.2.2 we present the business rules components of R0 to R12. Appendix III presents the DRL file containing business rule R0 to R12 business rules components generated by the ECA Model. Using the AND-OR graph presented in Chapter 4, various dependency patterns are defined. For example, Business rule R5 is directly dependent on R3. This relationship exists because an action property “Rack Space” is full (R3) causes an event in R5 to be invoked. Also, the business rule R6 is directly dependent on R5. The action of R5 causes the condition of R6 to be checked.

| Business rules statements                                                                                                                                                                                                                                   | No  | Event                                      | Condition                                              | Action                                                                                                                  |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|--------------------------------------------|--------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------|
| When workflow start activity then create new process 'Create Request - P1'                                                                                                                                                                                  | R0  | When Activity (Workflow) == 'Start'        |                                                        | Set Name (Process) == 'P1-Create Request'                                                                               |
| When submit request, if requestor is a member of the Platform capacity team then go to 'Review' step else go to 'Approve' step for data centre area manager to approve and set equipment request type to install                                            | R1  | When Status (Request) == 'Submit'          | If Rolename (Requestor) == 'Platform Capacity'         | Set Name (Process) == 'P2-Review'                                                                                       |
|                                                                                                                                                                                                                                                             | R2  | When Status (Request) == 'Submit'          | If Rolename (Requestor) != 'Platform Capacity'         | Set Name (Process) == 'P3-Approve' && Request Type (Equipment) == 'Install'                                             |
| When equipment install request triggered, if rack utilization is greater than the rack space capacity, then count installed equipment and set the Rack is full and set process to manage data centre space else install the equipment in the available rack | R3  | When Request Type (Equipment) == 'Install' | If Utilization (Rack) >= Capacity (Rack)               | Set Count (Equipment) == 'installed equipment' And Space (Rack) == 'isfull' And Name (Process) == 'P5- Manage DC Space' |
|                                                                                                                                                                                                                                                             | R4  | When Request Type (Equipment) == 'Install' | If Utilization (Rack) < Capacity (Rack)                | Set Name (Process) == 'P7-Install Equipment'                                                                            |
| When Rack space is full then set installed racks equal to rack capacity                                                                                                                                                                                     | R5  | When Space (Rack) == 'isfull'              |                                                        | Set installedRacks (Rack) = Capacity (Rack)                                                                             |
| If available racks volume is less than 10, then order new rack (create new Order Rack process)                                                                                                                                                              | R6  |                                            | If installedRacks (Rack) < 10                          | Set Name (Process) == 'P6-Order Rack'                                                                                   |
| If number of equipment power supplies is greater than zero, then set process name to provision power                                                                                                                                                        | R7  |                                            | If PowerSupplies (Equipment) > 0                       | Set Name (Process) == 'P8-Power Provision'                                                                              |
| If number of equipment network ports is greater than zero, then set process name to Provision Network                                                                                                                                                       | R8  |                                            | If network ports (Equipment) > 0                       | Set Name (Process) == 'P9-Network Provision'                                                                            |
| If number of equipment power connections is equal to equipment power supplies, then set process name to Completing Power and Network Provisioning                                                                                                           | R9  |                                            | If power connections (Equipment) == 'Power Supplies'   | Set Name (Process) == 'P10-Complete Power and Network Provision'                                                        |
| If equipment network cable is required, then set process name to Provision Network Cables                                                                                                                                                                   | R10 |                                            | If network cablling (Equipment) == 'Yes'               | Set Name (Process) == 'P11-Network Cables Provision'                                                                    |
| If equipment network connections and cables are configured, then set process name to Completing Power and Network Provisioning and request status is set to close (R11)                                                                                     | R11 |                                            | If network and cablling configured (Equipment) = 'Yes' | Set Name (Process) == 'P10-Complete Power and Network Provision'                                                        |
| If request status is set to close, then set process name to close request                                                                                                                                                                                   | R12 |                                            | If Status (Request) == 'Close'                         | Set Name (Process) == 'P12-Close Request'                                                                               |

Table 8.2. 2 Use case #1 - Business Rule Components (ECA)

To ensure all their installation process are executed on time, the company XYZ decided to insert another business rule (R13) to check for install status of the request and update it to

submit state. Table 8.2.3 presents new business rule R13 to be inserted. Additional dependency patterns are defined after insertion of R13. Whenever new business rules are added or inserted, our approach would be dynamically accessing the related business rules through dependency graphs and updating by propagating the change to the applicable rules and components before firing the rules again using the business rule management system. Experiment 1 will demonstrate how the change in propagation is affected when R13 is inserted.

| Business rules statements                                                | No         | Event | Condition                               | Action                                  |
|--------------------------------------------------------------------------|------------|-------|-----------------------------------------|-----------------------------------------|
| <i>If request status is installed, then set request status to Submit</i> | <i>R13</i> |       | <i>If Status (Request) == 'Install'</i> | <i>Set Status (Request) == 'Submit'</i> |

Table 8.2. 3 Use case #1 - Inserted Business Rule (R13)

## 8.2.2 Update Business Rules Components

### Use case #2:

This use case is useful in a scenario where business rules components (event, condition, action) are to be modified separately without changing the entire business rules. One of the biggest compromises in data centres is power and capacity, the two costs are the biggest expenses. The common belief is that the two costs increase together. The more racks (space capacity), the more power is needed to run them. This means the more capacity, the more power is needed, which could result in cooling issues. If the data centre has enough cooling and power, it could easily run out of rack space capacity. Like most data centres, XYZ (the Company described in Use case #1) faces similar problems. Hence, a decision was made to modify its existing business rule (R5) in its equipment installation workflow to ensure that there is enough space and power to run XYZ data centres. However, the business rule was contained in codes requiring programming experts to make the change. Typically, the work to identify and change business rules proves to be hard and time-consuming. Only certain components of the business rules would need to be changed. However, because of the way the rules were written, the entire rules would need to be changed. Furthermore, the modification process of change propagation was complicated. The effort and time spent for such a change was not economical, sometimes causing loss of money due to the downtime during workflow configuration.

The business processes and rules presented in Use case#1 are employed and to demonstrate how business rules are changed, business rule (R5) is modified (see below). Note, the changes to R5 impact other business rules so change propagation will need to be implemented. Here is a summary of the business rules:

**Workflow Name:** Equipment Installation

**Roles:** Requestor, Reviewer/Approver, Data Centre (DC) Manager,  
Power and Network Technicians (Tech)

**Business Rules** (Modified and related business rules):

- Existing R5: When notify Rack is full then set number of racks installed in the data centre equals to capacity
- Modified R5: When notify Rack is full then set equipment power capacity equals to zero
- Related Business rule (R6): If the total number of installed racks is less data centre rack capacity then order new rack
- Related Business rule (R9): If the number of equipment power connections is equal to equipment power supplies, then set process name to Completing Power and Network Provisioning

Table 8.2.4 presents the modified business rule (R5), broken down into components.

| Update Business Rule                                                      | No | Event                                | Condition | Action                                  |
|---------------------------------------------------------------------------|----|--------------------------------------|-----------|-----------------------------------------|
| <i>When notify Rack is full then set equipment power capacity to zero</i> | R5 | <i>When Space (Rack) == 'isfull'</i> |           | <i>Set powerCapacity (Equipment)= 0</i> |

Table 8.2. 4 Use case #2 - Business Rule Components (ECA)

### 8.2.3 Delete Business Rules Components

Use case#3:

Use case#3 is useful in a scenario where business rules are to be removed and changes are propagated to the business rules that are related to the business rule being deleted. Consider the XYZ company presented in Use case #1 and Use case #2, where business rule R5 is to



be removed. As shown in Use case #2, business rules R6 and R9 depend on business rule R5. If R5 is removed, then business rules R6 and R9 will never get implemented. The deletion also removes process P6 from the workflow and the process P8 no longer flows to process P11. Figure 8.2.2.2 presents the updated XYZ data centre equipment installation workflow.

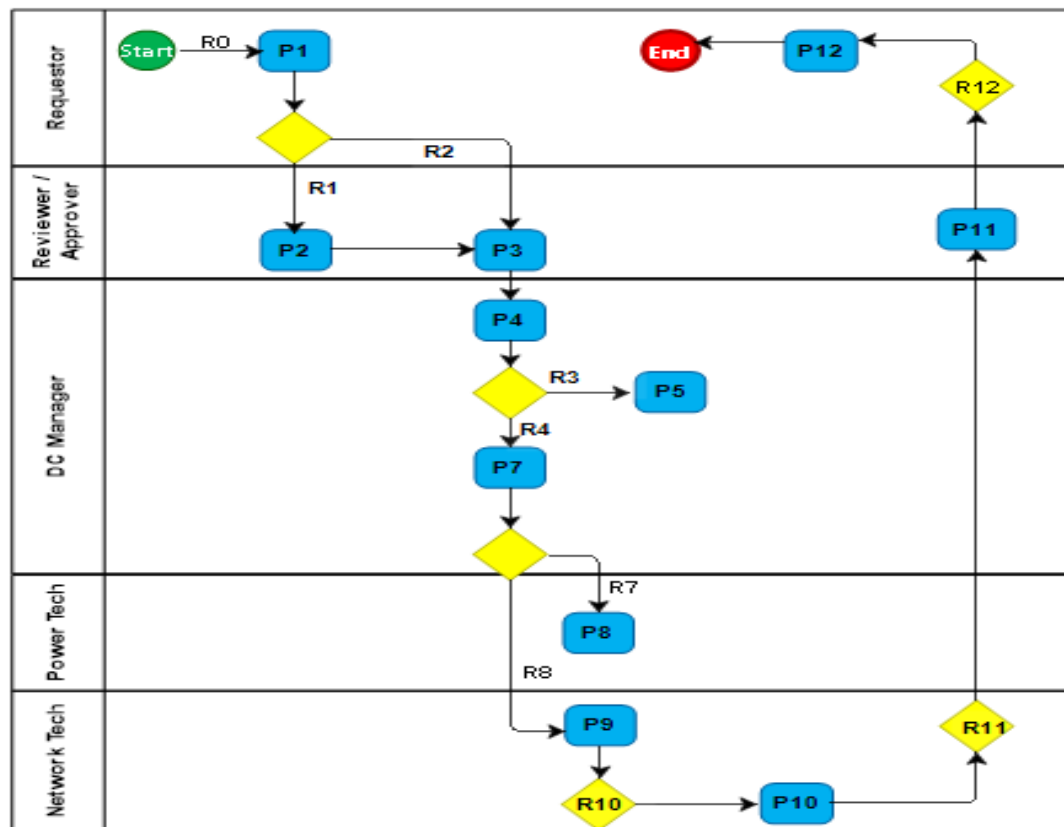


Figure 8.2.2. 2 XYZ Equipment Install Workflow after Deleting R5

## 8.2.4 Enable Business Rules to Initiate and Terminate Workflow

### Use case #4:

Consider the following example from data centre move workflow (Figure.8.2.2.3). When moving equipment from one data centre location to another, a requestor fills out a move form (request) to include equipment to be moved, current and new location, new power requirements, etc. Business rules exist to ensure power connected equipment is not moved around. The first business rule (R101) states that when the request type is “move”, then set

equipment power connections greater than zero. The second business rule (R102) states that if equipment power connections are greater than zero then request status is set to “close”. The third business rule (R103) states that if the equipment power connection is less than zero then the request status is set to power-provision and finally the fourth business rule (R104) states that if request status is set to power-provision then request status is set to close. Table 8.2.5 depicts the breakdown of business rules and their event, condition and action components.

**Workflow Name:** Move Equipment

**Roles:** Requestor, Data Centre (DC) Manager

**Business Processes:**

- P101 - Notify move request
- P102 - Provision power for equipment move
- P103 - Close Request

**Business Rules:**

- When notified move request then set equipment power connections greater than zero (R101)
- If equipment power connection is greater than zero, then request status is set to close (R102)
- If equipment power connection is less than zero, then request status is set to power-provision (R103)
- If request status is set to power-provision, then request status is set to close (R104)

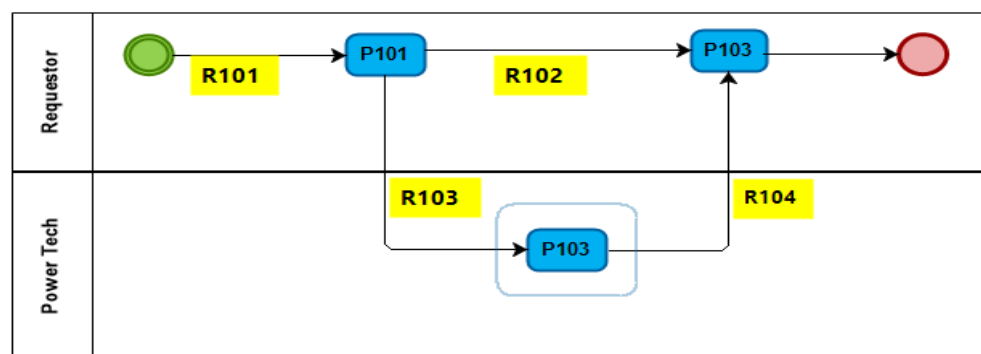


Figure 8.2.2. 3 Equipment Move Workflow Diagram

| Existing Business Rules                                                                               | No   | Event                         | Condition                                | Action                                    |
|-------------------------------------------------------------------------------------------------------|------|-------------------------------|------------------------------------------|-------------------------------------------|
| <i>When notified request type is move then set equipment power connections greater than zero</i>      | R101 | When Type (Request) == "move" |                                          | Set Power Connections (Equipment) > 0     |
| <i>If equipment power connections are greater than zero then request status is set to close</i>       | R102 |                               | If Power Connections (Equipment) > 0     | Set Status (Request) == 'close'           |
| <i>If equipment power connection is less than zero, then request status is set to power-provision</i> | R103 |                               | If Power Connections (Equipment) < 0     | Set Status (Request) == 'power-provision' |
| <i>If request status is set to power-provision, then request status is set to close</i>               | R104 |                               | If Status (Request) == 'power-provision' | Set Status (Request) == 'close'           |

Table 8.2. 5 Use case #4 - Business Rule Components (ECA)

## 8.2.5 Sequential Flow Patterns

### Use case #5:

Consider the following scenario from data centre decommission workflow (Figure.8.2.2.4). When equipment is decommissioning, the requestor fills out a decommission form. This form contains all the decommission information including the location and equipment to be removed. Business rules exist to ensure the validity of the equipment i.e. equipment end date, location, etc, and a request is checked before equipment can be scheduled for decommission. The business rules can be summarised as follows: first, if the equipment end period is reached, then request status is set to approval decommission. Second, when the notified request status is set to "approve decommission", then equipment status is set to "out of date". Third, if the equipment status is "out of date", then schedule a day for physical decommission, which is the current date plus a week. The workflow business processes and rules are summarised as follows and Figure.8.2.2.4 presents the actual workflow.

**Workflow Name:** Decommission Equipment

**Roles:** Requestor, Approver

**Business Processes:**

- P201 - Initiate equipment decommission request

- P202 – Approve
- P203 - Schedule Decommission

#### Business Rules:

- If equipment end period is less than today's date, then request status is set to decommission approved. (R201)
- When notified request status is decommission approved then equipment status is set to out of date (R202)
- If equipment status is out of date, then schedule for physical decommission - current date plus a week (R203)

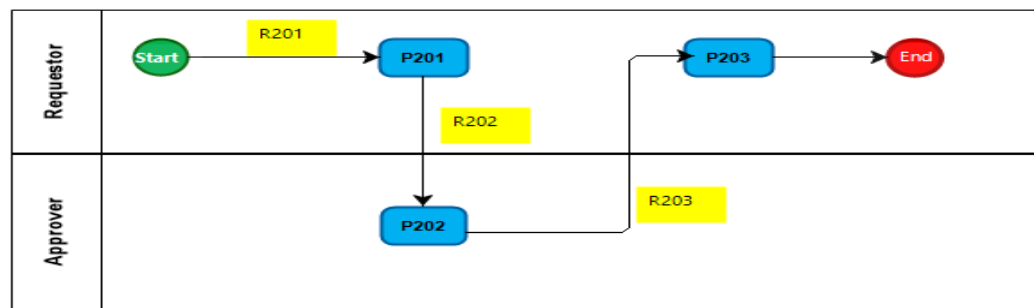


Figure 8.2.2. 4 Sequential Workflow

Mapping business rules statements into business rules components. Table 8.2.6 presents business rules (Use case #5) map into components and operators

| Existing Business Rules                                                                                       | No   | Event                                            | Condition                                          | Action                                                  |
|---------------------------------------------------------------------------------------------------------------|------|--------------------------------------------------|----------------------------------------------------|---------------------------------------------------------|
| <i>If equipment end period is less than today's date, then request status is set to decommission approved</i> | R201 |                                                  | <i>If End Period (Equipment) &lt; today's date</i> | <i>Set Status (Request) == 'decom-approved'</i>         |
| <i>When notified request status is decommission approved then equipment status is set to out of date</i>      | R202 | <i>When Status (Request) == 'decom-approved'</i> |                                                    | <i>Set Status (Equipment) == 'out-of-date'</i>          |
| <i>If equipment status is out of date, then schedule for physical decommission - current date plus a week</i> | R203 |                                                  | <i>If Status (Equipment) == 'out-of-date'</i>      | <i>Set Decom Schedule Date (Equipment) == Today + 7</i> |

Table 8.2. 6 Use case #5 - Business Rule Components (ECA)

## 8.2.6 Parallel-OR Merge Flow Patterns

### Use case #6:

Consider the following decommission workflow (Figure.8.2.2.5) where business rules have been added to ensure equipment is first disconnected by power or network provisioner before final decommission process is executed.

Workflow Name: Equipment Decommission

Roles: Requestor, Power Tech, Network Tech

#### Business Processes:

- P301 - Initiate equipment decommission request
- P302 - Power Decommission
- P303 - Network Decommission
- P304 - Close Request

#### Business Rules:

- If equipment end period is yes, then send notification for equipment decommission request (R301)
- When notified decommission request if power provisioner is yes then set request status to decom-approved (R302)
- When notified decommission request if network provisioner is yes then set request status to decom-approved (R303)
- If request status is decom-approved, then set request status to close (R304)

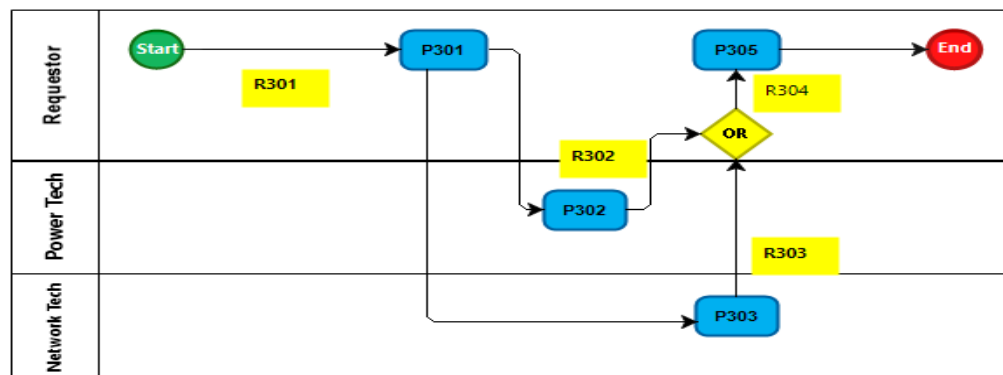


Figure 8.2.2. 5 Parallel-OR Merged Workflow

### Mapping business rules statements into business rules components

Table 8.2.7 presents the business rules presented in Use case #6, mapped into components (event, condition and action) and operators (==, !=, >=, <, etc.).

| Business Rules                                                                                                    | No   | Event                                             | Condition                                      | Action                                             |
|-------------------------------------------------------------------------------------------------------------------|------|---------------------------------------------------|------------------------------------------------|----------------------------------------------------|
| <i>If equipment end period is yes, then send notification for equipment decommission request</i>                  | R301 |                                                   | <i>If Endperiod (Equipment) == yes</i>         | <i>Set Notification (Request) = 'decommission'</i> |
| <i>When notified decommission request if power provisioner is yes then set request status to decom-approved</i>   | R302 | <i>When Notification (Request) = decommission</i> | <i>If PowerProvisioner (Equipment) = yes</i>   | <i>Set Status (Request) == 'decom-approve'</i>     |
| <i>When notified decommission request if network provisioner is yes then set request status to decom-approved</i> | R303 | <i>When Notification (Request) = decommission</i> | <i>If NetworkProvisioner (Equipment) = yes</i> | <i>Set Status (Request) == 'decom-approve'</i>     |
| <i>If request status is decom-approved, then set request status to close</i>                                      | R304 |                                                   | <i>If Status (Request) == 'decom-approve'</i>  | <i>Set Status (Request) == 'close'</i>             |

Table 8.2. 7 Use case #6 - Business Rule Components (ECA)

### **8.2.7 Parallel-AND Merge Flow Patterns**

#### Use case #7:

Consider the following decommission workflow (Figure 8.2.2.7), which is an extension to the workflow presented in Figure 8.2.2.5. However, in Figure 8.2.2.7 both power and network provisioner must approve for decommissioning of the equipment.

**Workflow Name:** Equipment Decommission

**Roles:** Requestor, Approver, Power Tech, Network Tech

**Business Processes:**

- P301 - Initiate equipment decommission request
- P302 - Power Decommission
- P303 - Network Decommission
- P304 - Close Request

**Business Rules:**

- If equipment end period is yes, then send notification for equipment decommission request (R301)
- When notified decommission request if power provisioner is yes then set request status to decom-approved (R302)
- When notified decommission request if network provisioner is yes then set request status to decom-approved (R303)
- If request status is decom-approved, then set request status to close (R304)

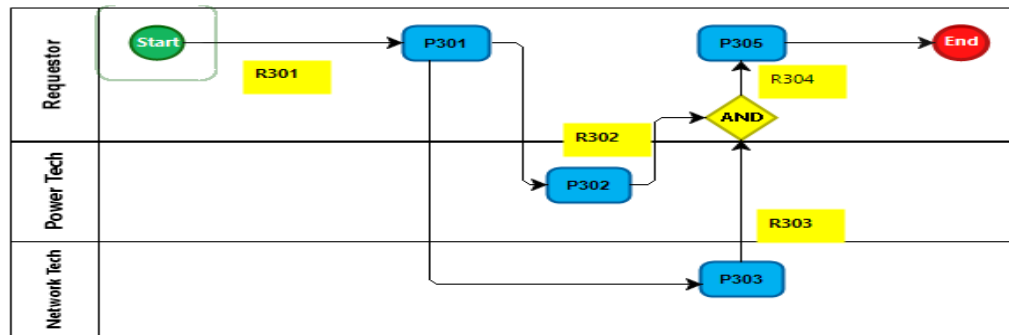


Figure 8.2.2. 7 Parallel-AND Merge Workflow

#### Mapping business rules statements into business rules components

Table 8.2.8 presents the business rules presented in Use case #7, mapped into components (event, condition and action) and operators (==, !=, >=, <, etc.).

| Business Rules                                                                                                    | No   | Event                                             | Condition                                      | Action                                             |
|-------------------------------------------------------------------------------------------------------------------|------|---------------------------------------------------|------------------------------------------------|----------------------------------------------------|
| <i>If equipment end period is yes, then send notification for equipment decommission request</i>                  | R301 |                                                   | <i>If Endperiod (Equipment) == yes</i>         | <i>Set Notification (Request) = 'decommission'</i> |
| <i>When notified decommission request if power provisioner is yes then set request status to decom-approved</i>   | R302 | <i>When Notification (Request) = decommission</i> | <i>If PowerProvisioner (Equipment) = yes</i>   | <i>Set Status (Request) == 'decom-approve'</i>     |
| <i>When notified decommission request if network provisioner is yes then set request status to decom-approved</i> | R303 | <i>When Notification (Request) = decommission</i> | <i>If NetworkProvisioner (Equipment) = yes</i> | <i>Set Status (Request) == 'decom-approve'</i>     |
| <i>If request status is decom-approved, then set request status to close</i>                                      | R304 |                                                   | <i>If Status (Request) == 'decom-approve'</i>  | <i>Set Status (Request) == 'close'</i>             |

Table 8.2. 8 Use case #7 - Business Rule Components (ECA)

## 8.2.8 Parallel-OR Split Flow Patterns

### Use case #8:

Consider the following scenario from data centre equipment SLA workflow (Figure 8.2.2.8 and Table 8.2.9). In a data centre, the Service Level Agreements (SLAs) are designed to ensure different data centre activities are completed within a specified period to improve performance by avoiding unnecessary delays in completion of activities. In this use case, for all critical equipment the SLA demands that all scheduling and installation related tasks are completed within 2 days of the start date of the activities. In the case of a breach of SLA, an escalation process is completed. It involves emailing the person who is supposed to complete the request as well as their manager for further action. An additional business rule exists to notify the requestor when the equipment is not installed within the agreed timescale.

**Workflow Name:** Equipment SLA

**Roles:** Data Centre Operator, Data Centre Manager

### **Business Processes:**

- P404 - Scheduling
- P405 - Installation
- P406 - Manage SLA

### **Business Rules:**

- If Equipment type is critical, then set SLA Request equal to yes (R404)
- When notify SLA Request; If request completion date is equal to start date – 2 days then set completion status to ‘On time’ (R405)
- When notify SLA Request; If request completion date is taking more than 2 days against the start date, then set completion status to ‘Delayed’ (R406)

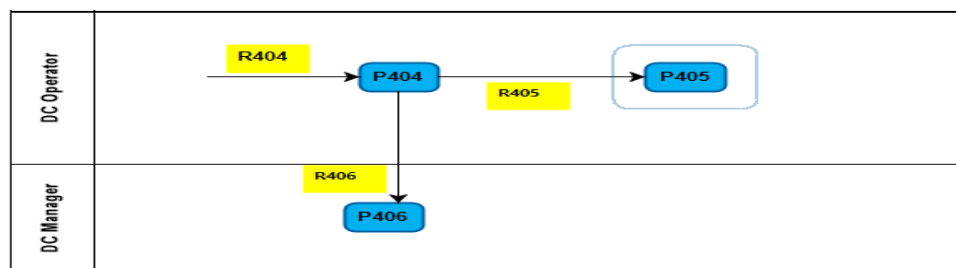


Figure 8.2.2. 8 OR Split Workflow



| Business Rules                                                                                                                                        | No   | Event                               | Condition                                                              | Action                                            |
|-------------------------------------------------------------------------------------------------------------------------------------------------------|------|-------------------------------------|------------------------------------------------------------------------|---------------------------------------------------|
| <i>If Equipment type is critical, then set Request type equal to SLA</i>                                                                              | R404 |                                     | <i>If Type (Equipment) == 'Critical'</i>                               | <i>Set Type (Request) = 'SLA'</i>                 |
| <i>When notify SLA Request; If request completion date is equal to startdate - 2, then set completion status to 'On time'</i>                         | R405 | <i>When Type (Request) == 'SLA'</i> | <i>If Completion Date (Request) &lt;= StartDate (Request) - 2 Days</i> | <i>Set completionStatus (Request) = 'On time'</i> |
| <i>When notify SLA Request; If request completion date is taking more than 2 days against the start date, then set completion status to 'Delayed'</i> | R406 | <i>When Type (Request) == 'SLA'</i> | <i>If Completion Date (Request) &gt; StartDate (Request) - 2 Days</i>  | <i>Set completionStatus (Request) = 'Delayed'</i> |

Table 8.2. 9 Use case #8 - Business Rule Components (ECA)

## 8.2.9 Parallel-AND Split Flow Patterns

### Use case #9:

Consider the following scenario from a data centre move workflow (Figure.8.2.2.9 and Table.8.2.10). When moving equipment from one location to the other, both power and network connections must be disconnected from the equipment. Below is a summary of the business processes and rules that are managed.

**Workflow Name:** Equipment Move

**Roles:** Data Centre Operator, Power Provisioner and Network Provisioner

### **Business Processes:**

- P501- Create Move Request
- P502 - Power Connections Decommission
- P503 - Network Connections Decommission
- P504 - Run Network Cable
- P505 - Close Request

### **Business Rules:**

- If Request type equals to 'move' then set equipment connectionflag to yes (R501)
- When equipment connectionflag is yes, if connection type equals to 'Power' then set equipment connection to 0 (R502)
- When equipment connectionflag is yes, if connection type equals to 'Network' then set equipment cableflag to yes to 0 (R503)
- When equipment cableflag is yes then set request status to 'close' (R504)

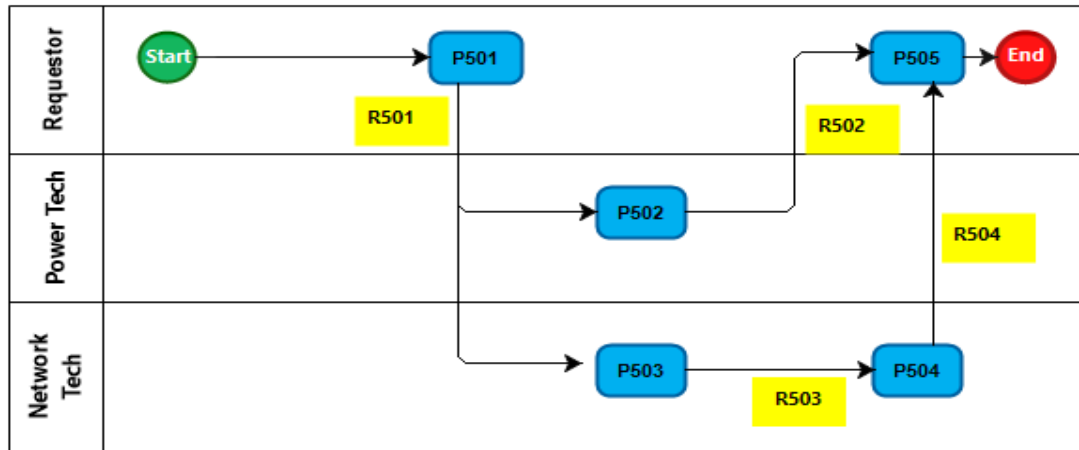


Figure 8.2.2. 9 AND Split Workflow

| Business Rules                                                                                                               | No   | Event                                            | Condition                                          | Action                                        |
|------------------------------------------------------------------------------------------------------------------------------|------|--------------------------------------------------|----------------------------------------------------|-----------------------------------------------|
| <i>If Request type equals to 'move' then set equipment connectionflag to yes</i>                                             | R501 |                                                  | <i>If Type (Request) == 'move'</i>                 | <i>set Connectionflag (Equipment) = 'yes'</i> |
| <i>When equipment connectionflag is yes, if connection type equals to 'Power' then set equipment connection to 0</i>         | R502 | <i>When Connection-flag (Equipment) == 'yes'</i> | <i>If connectionType (Equipment) == 'Power'</i>    | <i>Set connection (Equipment) = 0</i>         |
| <i>When equipment connectionflag is yes, if connection type equals to 'Network' then set equipment cableflag to yes to 0</i> | R503 | <i>When Connectionflag (Equipment) == 'yes'</i>  | <i>If Connection-Type (Equipment) == 'Network'</i> | <i>Set cableflag (Equipment) = 'yes'</i>      |
| <i>When equipment cableflag is yes then set then set equipment connection to 0</i>                                           | R504 | <i>When Set cableflag (Equipment) = 'yes'</i>    |                                                    | <i>Set connection (Equipment) == 0</i>        |

Table 8.2. 10 Use case #9 - Business Rule Components (ECA)

### 8.3 Experiments

The experiments were carried out using a series of tests derived from use cases (section 8.2), objectives (5a-5d) and the ECA Test Client prototype. The ECA Test Client was developed on top of the JBoss Drools rule engine to allow creation, deletion, modification and execution of business rules in real time. Furthermore, Drools provided a suitable

environment for the execution of workflow formulated by the executed business rule. Currently the prototype is a standalone application, which can be deployed on the user's desktop. It is worthy to note that these experiments were executed on a 64-bit windows operating system, equipped with 4GB of RAM and Intel R Core™ i5-4210U CPU @ 1.70GHz 2.40GHz. For easy of referencing, Table 8.3.1 lists down experiments along with a use case used.

| Experiment No  | Experiment Description                                                                                                                             | Use case #  |
|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------|-------------|
| Experiment 1   | Adding business rules & components structure at run time and change propagation                                                                    | Use case #1 |
| Experiment 2   | Modifying business rules and components structures and change propagation                                                                          | Use case #2 |
| Experiment 3   | Deleting business rules and components structures and change propagation                                                                           | Use case #3 |
| Experiment 4   | Ability to enable a business rule to initiate a process in a workflow                                                                              | Use case #4 |
| Experiment 4   | Ability to enable a business rule to terminate a process in a workflow                                                                             | Use case #4 |
| Experiment 5   | Ability to enable sequential process flow patterns                                                                                                 | Use case #5 |
| Experiment 5 A | Insertion of business rule components to generate sequential process flow patterns                                                                 | Use case #5 |
| Experiment 5 B | Modification of an existing business rule (changing source or target process flows) in the sequential workflow pattern disconnects process flows   | Use case #5 |
| Experiment 5 C | Deletion of an existing business rule in the sequential workflow pattern disconnects existing process flows                                        | Use case #5 |
| Experiment 6   | Ability to enable Parallel-OR Merge flow patterns                                                                                                  | Use case #6 |
| Experiment 6 A | Insertion of a new business rule in the OR Merged rules flow pattern create a new process flow connection                                          | Use case #6 |
| Experiment 6 B | Modification of an existing business rule (changing source or target process flows) in the OR Merged rules flow pattern disconnects process flows  | Use case #6 |
| Experiment 6 C | Deletion of an existing business rule in the OR Merged rules flow pattern disconnects existing process flows                                       | Use case #6 |
| Experiment 7   | Ability to enable Parallel-AND Merged flow patterns                                                                                                | Use case #7 |
| Experiment 7 A | Insertion of a new business rule in the AND Merged rules flow pattern create a new process flow connection                                         | Use case #7 |
| Experiment 7 B | Modification of an existing business rule (changing source or target process flows) in the AND Merged rules flow pattern disconnects process flows | Use case #7 |
| Experiment 7 C | Deletion of an existing business rule in the AND Merged rules flow pattern disconnects existing process flows                                      | Use case #7 |
| Experiment 8   | Ability to enable Parallel-OR Split flow patterns                                                                                                  | Use case #8 |
| Experiment 8 A | Insertion of a new business rule in the OR Parallel Split workflow pattern create a new process flow connection                                    | Use case #8 |

|                |                                                                                                                                                          |             |
|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|
| Experiment 8 B | Modification of an existing business rule (changing source or target process flows) in the OR Parallel Split workflow pattern disconnects process flows  | Use case #8 |
| Experiment 8 C | Deletion of an existing business rule in the OR Parallel Split workflow pattern disconnects existing process flows                                       | Use case #8 |
| Experiment 9   | Ability to enable Parallel-AND Split flow patterns                                                                                                       | Use case #9 |
| Experiment 9 A | Insertion of a new business rule in the AND Parallel Split workflow pattern create a new process flow connection                                         | Use case #9 |
| Experiment 9 B | Modification of an existing business rule (changing source or target process flows) in the AND Parallel Split workflow pattern disconnects process flows | Use case #9 |
| Experiment 9 C | Deletion of an existing business rule in the AND Parallel Split workflow pattern disconnects existing process flows                                      | Use case #9 |

Table 8.3. 1 Experiments and Use cases

### 8.3.1 Validation of Dynamic Business Rules and Change propagation

The first set of experiments (1-3) focus on validation of the proposed ECA Model's ability to deal with business rules' flexibility and change propagation problems discussed in previous chapters.

#### 8.3.1.1 Adding Business Rules Components & Change Propagation

##### Experiment 1:

This experiment is designed to show the proposed ECA Model's ability to handle the complexity of adding business rules and components at run time. Furthermore, the experiment demonstrates the model's ability to deal with the difficulty of propagating changes when new business rules and components are inserted. The process of adding business rules and components is designed to be flexible and adaptable. The change propagation process is automatic and seamless to the users, consequently reducing the efforts required for adding business rules and components into rule repositories, thus speeding up the response times at rule creation, runtime and improving usability. Allowing visibility of related business rules and components, removing duplication and promoting consistency are just some of the advantages of a better business rule management framework. For the sake of simplicity, the experiment is divided in two areas:

- Model's ability to add business rules and components (event, condition and action) at run time
- Model's ability to propagate changes

Ability to add business rule and components (event, condition, action) structure at runtime

The ECA Model as defined in this research makes business rules and components (event, condition and action) put upon the use of classes explicitly. Business rules are specified and added depending primarily on the chosen component class specification as described in chapter 4. The event class, condition class and action class are free parts of the business rule class. For this reason, the ECA Model classification provides a better background for creating business rules at components level, therefore helps with the following problems:

- The complexity of dynamic creation of business rules at component level due to lack of high level of abstraction. The ECA Model creates business rule abstraction, which makes it easier to design a component class and its properties. Keeping the components classes separate and being able to easily specify its properties reduces the complexity of the creation task. Also, it facilitates a consistent creation of business rule components before deployment.
- Complexity of creation of business rules and components by non-technical users (Usability). Figure 8.3.1.1 shows one of the business rules (R1) entered via ECA Model Test Client. The ECA Model Test Client allows for flexibility in business rule components creation and makes it easy for the end-users to capture business rules and components separately.

The business rules in Use case #1 were mapped into event, condition, action components as presented in Table 8.2.1. Using the ECA Model Test Client, they were added into the ECA Model and the Drools DRL file (Appendix III) was generated. The ECA Model Test Client allows business rule components to be inserted separately. For ease of use, the ECA Model Test Client sections are clearly identified and marked for entering components of business rules including objects, properties, values and other operator i.e. comparison

operators. With the very minimal training, anybody can add business rules and components.

First, we look at the model's ability to add entire business rules, which consists of three components (event, condition and action). As an example, R1 from Use case #1 is entered via the ECA Model Test Client. The data or information for each component of R1 is filled in appropriately. Drools DRL file is automatically generated, and the contents are displayed on the DRL Rule Template section of the ECA Model Test Client. The contents of the executed business rules are displayed under the statistics section, showing the number of business rules in the rule repository, number of business rules that are being fired, etc. Figure 8.3.1.1 captures the entire process of adding R1 to ECA Model and mapped the data into a correct format ready for Drool's runtime execution.

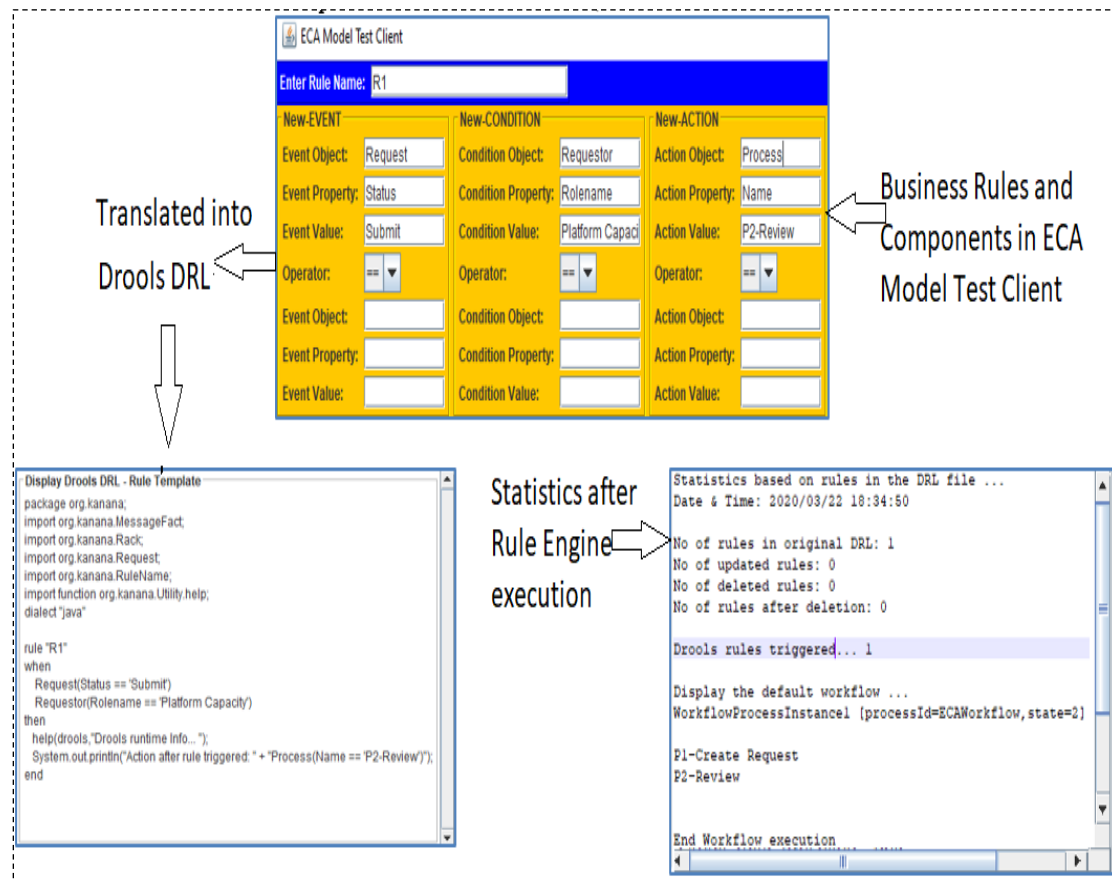


Figure 8.3.1. 1 Adding R1 and Components via ECA Model Test Client

Second, we look at the model's ability to add separate business rule components i.e. add event and action, condition and action, event only, action only. As an example, R5 is entered via ECA Model Test Client but this time only event and action components of the business rule are added. The data or information for event and action components of R5 are filled in appropriately. Drools DRL file is automatically generated and content of DRL is also displayed on DRL Rule Template section of the ECA Model Test Client as previously discussed. Figure 8.3.1.2 captures the process of adding R5's event and action components to ECA Model and mapped the data into a correct format ready for Drool's runtime execution.

Enter Rule Name: R5

| New-EVENT             | New-CONDITION       | New-ACTION                      |
|-----------------------|---------------------|---------------------------------|
| Event Object: Rack    | Condition Object:   | Action Object: Rack             |
| Event Property: Space | Condition Property: | Action Property: installedRacks |
| Event Value: isfull   | Condition Value:    | Action Value:                   |
| Operator: ==          | Operator:           | Operator: ==                    |
| Event Object:         | Condition Object:   | Action Object: Rack             |
| Event Property:       | Condition Property: | Action Property: Capacity       |
| Event Value:          | Condition Value:    | Action Value:                   |

Translated into Drools DRL

```

package org.kanana;
import org.kanana.Process;
import org.kanana.Rack;
import org.kanana.Requestor;
import org.kanana.RuleName;
import function org.kanana.utility.help;
dialect "java"

rule "R5"
when
 Rack(Space == 'isfull')
then
 help(drools,"Drools runtime Info... ");
 System.out.println("Action after rule triggered: " + "Rack(installedRacks == Capacity)");
end

rule "R1"
when
 Requestor(Status == 'Submit')
 Requestor(RoleName == 'Platform Capacity')
then
 help(drools,"Drools runtime Info... ");
 System.out.println("Action after rule triggered: " + "Process(Name == 'P1-Review')");
end

```

Statistics after Rule Engine execution

Statistics based on rules in the DRL file ...  
 Date & Time: 2020/03/22 18:34:50  
 No of rules in original DRL: 2  
 No of updated rules: 0  
 No of deleted rules: 0  
 No of rules after deletion: 0  
 Drools rules triggered... 1  
 Display the default workflow ...  
 WorkflowProcessInstance [processId=ECAWorkflow,state=2]  
 P1-Create Request  
 P2-Review  
 End Workflow execution

Figure 8.3.1. 2 Adding R5 Event and Action Components via ECA Model Test Client

Third, we look at the model's ability to add separate business rule components condition and action. As an example, R6 is entered via the ECA Model Test Client but this time only condition and action components of the business rule are added. The data or information for condition and action components of R6 are filled in appropriately. The Drools DRL file is automatically generated and the content of DRL is also displayed on DRL Rule Template section of the ECA Model Test Client as previously discussed. Figure 8.3.1.3 captures the process of adding R6's condition and action components to the ECA Model and maps the data into a correct format ready for Drool's runtime execution.

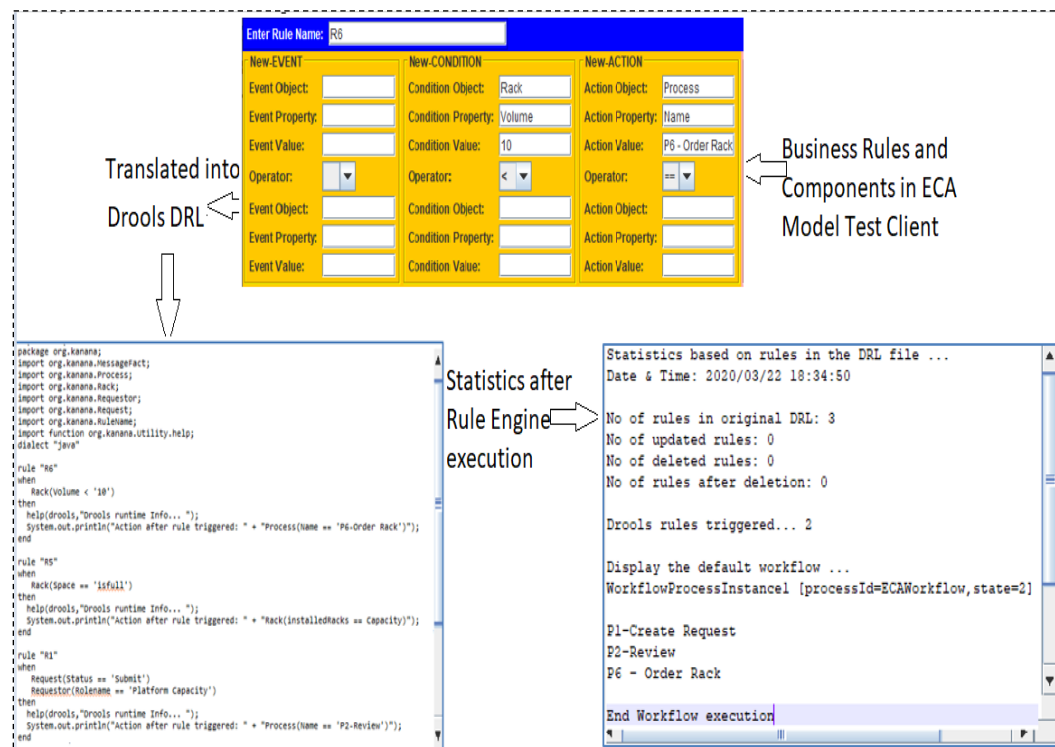


Figure 8.3.1. 3 Adding R6 Condition and Action Components via ECA Model Test Client

Using the proposed model, we can model business rules at components level. Users are free to enter any part/component of a business rule and any combination can be specified. The ECA Model satisfies the adaptability and flexibility of adding business rules and components. Furthermore, with a guided user interface (ECA Model Test Client), a non-technical user can add any number of business rules which will then be converted into Drool rule language.



Ability to propagate change when new business rules and components are inserted/added

Chapter 6 has shown the change propagation algorithm with associated combination of dependency graph patterns to allow the ECA Model to manage business rules components change. The ECA Model expresses the change propagation over business rules relationships. Utilizing the dependency graphs with propagation algorithms eases the change propagation complexity where several related business rules and components are maintained in a business rule repository. Although the proposed model provides a high level of abstraction by separating the business rules components, the structure of the graph dependency patterns (Path, Level, Direct-Node and Neighbour dependencies) provides links between business rules and components with similar behaviour and shapes. At each stage of the change propagation, the ECA Model concerns itself with the related business rules components patterns or layers. This means the ECA Model can express change propagation quickly with ease, while it would have increased complexity, as related business rules components were not well structured. For this reason, the ECA Model's change propagation method provides not only a method of tracking all related business rules but also updating the affected business rules at component level, thus resolving the following problems identified in Objective (5c) of this research:

- Difficulty in propagating changes on related business rules component level
- Performance or efforts needed to apply the business rule change; It may take longer to propagate changes due decentralised business rules and components

In this experiment, we apply the sample data from Use case #1 to demonstrate how our change propagation method works, in particularly looking at propagation patterns when new business rule (R13) and components are added. R13 connects to five business rules components. Business rules R2 has a direct dependency on R13's action component. This leads to indirect relation to R3 and R4 event components, R3's action component connects to R5's event component and R6's condition component. R13's change propagation to R2, R3, R4, R5 and R6 needs to be revised to guarantee the activation of all the rules. Figure 8.3.1.4 displays the process of adding business rules (R13) and components as well as

various dependency graphs to show which business rules will be affected by changing R13. At component level, business rules are linked or connected to each other, i.e. R13's action connects to R2's event. Drools DRL file is automatically updated to include new inserted business rule and components (Appendix IV). The dependency/change propagation graph is displayed in the "Display ECA Graph - Rule relationships" section of the ECA Model Test Client.

Change Propagation Algorithm Results (Showing different dependency patterns and which business rules components need to be revised after inserting R13)...

Paths dependency:

A(R13) → E(R2);

A(R13) → E(R2); A(R2) → E(R4)

A(R13) → E(R2); A(R2) → E(R3)

A(R13) → E(R2); A(R2) → E(R3); A(R3) → E(R5)

A(R13) → E(R2); A(R2) → E(R3); A(R3) → E(R5); A(R5) → C(R6)

Level dependency:

A(R13)

E(R2) A(R2)

E(R3) A(R3) E(R4)

E(R5) A(R5)

C(R6)

Direct-Modes (Parent/Child relations):

A(R13) → E(R2)

A(R13) → E(R2) V A(R2) → E(R4)

A(R13) → E(R5)

A(R5) → C(R6)

Neighbours dependency:

Predecessors:

Successor: A(R13) → E(R2)

ECA Model Test Client

Enter Rule Name: R13

| New-EVENT                            | New-CONDITION                            | New-ACTION                            |
|--------------------------------------|------------------------------------------|---------------------------------------|
| Event Object: <input type="text"/>   | Condition Object: Request                | Action Object: Request                |
| Event Property: <input type="text"/> | Condition Property: Status               | Action Property: Status               |
| Event Value: <input type="text"/>    | Condition Value: Install                 | Action Value: Submit                  |
| Operator: <input type="text"/>       | Operator: == <input type="text"/>        | Operator: == <input type="text"/>     |
| Event Object: <input type="text"/>   | Condition Object: <input type="text"/>   | Action Object: <input type="text"/>   |
| Event Property: <input type="text"/> | Condition Property: <input type="text"/> | Action Property: <input type="text"/> |
| Event Value: <input type="text"/>    | Condition Value: <input type="text"/>    | Action Value: <input type="text"/>    |

Figure 8.3.1. 4 Insert R13's Condition and Action causing Change Propagation

By using the dependency graphs to define new dependencies and regenerating existing relations of A(R13), the algorithm provides the ability to insert new business rules at component level A(R13) and propagate changes by revising all related business rule components as seen in Figure 8.3.1.11. We also look at the change cost to measure performance or efforts needed to apply or modify the business rule change. For example, if business rule R13 is added and business rule R2 is changed in the previous example, so the effective change effort applicable to business rule R13 concerns the efforts to change business rules R3, R4, R5 and R6 plus the efforts to change business rule R2. It is important to estimate the maximum change cost before making any changes. This will help to

determine and plan the change in advance hence giving a tangible estimation of the efforts needed to implement business rule changes. In our model, the cost of changes is based upon the business rule change dependency patterns in a graph. The arcs in a graph patterns are used as inputs to access the change. For example, the neighbour dependency's pattern will help to determine the effort required to change successors or predecessors of a given business rule component. The Level dependency pattern allows us to determine the distance between business rule components.

### **8.3.1.2 Modifying Business Rules Components**

#### **Experiment 2**

This experiment is designed to show the usefulness and competence of our ECA Model's ability to support the modification of business rules and components at runtime, enabling business rules and components to be modified by non-technical users (usability). The changes are propagated to the business rules that are related to the component being modified. However, the change propagation aspect for business rules modification is not considered in this section as it was covered in experiment 1. Ideally this experiment intends to validate the modification process of business rules and components on client workflow applications resulting in flexibility and visibility of business rules, consequently, reducing the efforts required to modify business rules and components in rule repositories, thus speeding up the response times on rule modification and improving usability.

First, we look at the model's ability to modify entire business rules, which consists of three components (event, condition and action). As an example, R5 from Use case #2 is entered via ECA Model Test Client. The data or information for each component of R5 is filled in appropriately. Drools DRL file is automatically generated and the contents are displayed on DRL Rule Template section of the ECA Model Test Client. The contents of executed business rules are displayed under the statistics section, showing the number of business rules in the rule repository, number of business rules that are being modified and fired, etc. Figure 8.3.1.5 captures the entire process of modifying R5 to ECA Model and mapped the data into a correct format, ready for Drool's runtime execution

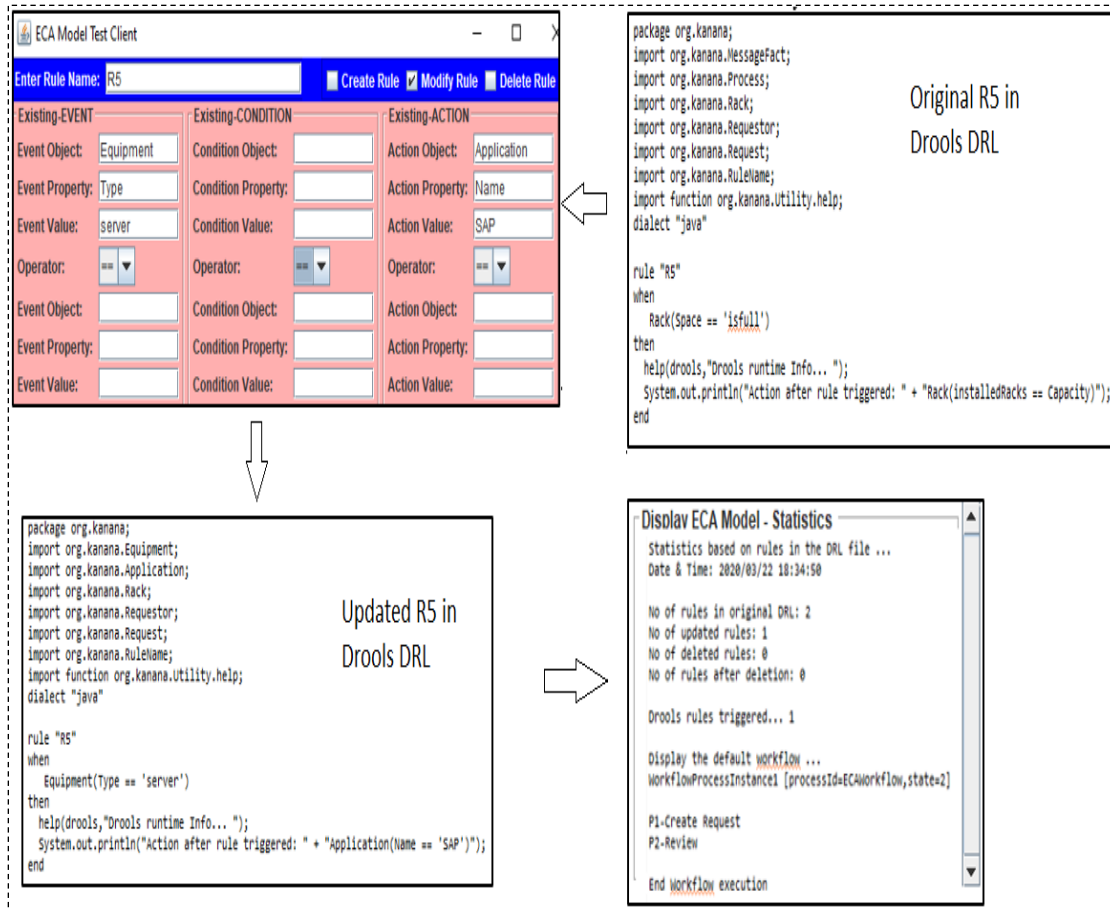


Figure 8.3.1. 5 Modify R5 and Components via ECA Model Test Client

Second, we look at the model's ability to modify separate business rule components i.e. modify event and action, condition and action, event only, action only etc. As an example, R5 is entered via the ECA Model Test Client but this time only the action component of the business rule is modified and so the event remains the same. The data or information for event component of R5 is filled in appropriately. Drools DRL file is automatically generated and content of DRL is also displayed on DRL Rule Template section of the ECA Model Test Client as previously discussed. Figure 8.3.1.6 captures the process of updating R5's action components to ECA Model and mapped the data into a correct format ready for runtime execution.

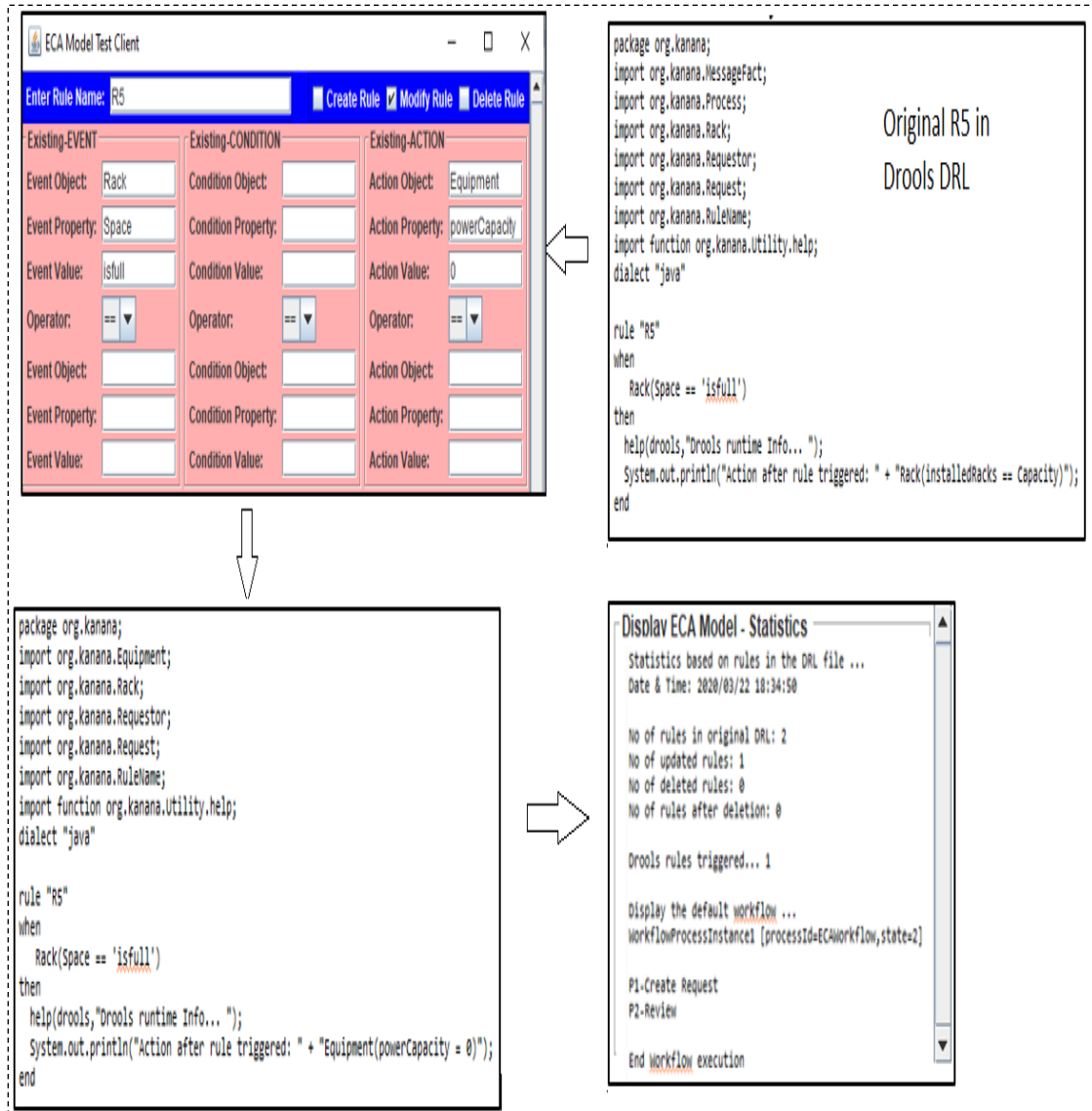


Figure 8.3.1. 6 Modifying R5 Event and Action Components via ECA Model Test Client

Using the proposed model, we can modify business rules at components level. Users are free to modify any whole or part/component of a business rule, any combination can be specified. The ECA Model satisfies the adaptability and flexibility of modifying business rules and components. As mentioned before, with a guided user interface (ECA Model Test Client), a non-technical user can modify any number of business rules.

### 8.3.1.3 Deleting Business Rules & Components

#### Experiment 3

In this experiment, we use sample data in Use case #3 to show the proposed ECA Model's ability to support the deletion of business rules and components at runtime. Obviously when business rules are deleted, all connected business rule components are impacted and need to be revised. However, the change propagation aspect for business rules deletion is not considered in this section as it was covered in experiment 1. Ideally this experiment intends to validate the deletion process of business rules and components on client workflow applications resulting in flexibility and visibility of business rules. As an example, R5 from Use case #3 is entered via ECA Model Test Client. The data or information for each component of R5 is filled in appropriately. Drools DRL file is automatically generated and the contents are displayed on DRL Rule Template section of the ECA Model Test Client. The contents of the executed business rules are displayed under the statistics section, showing the number of business rules in the rule repository, number of business rules that are being modified and fired, etc. Figure 8.3.1.7 captures the entire process of removing R5 to ECA Model and mapped the data into a correct format, ready for Drool's runtime execution.

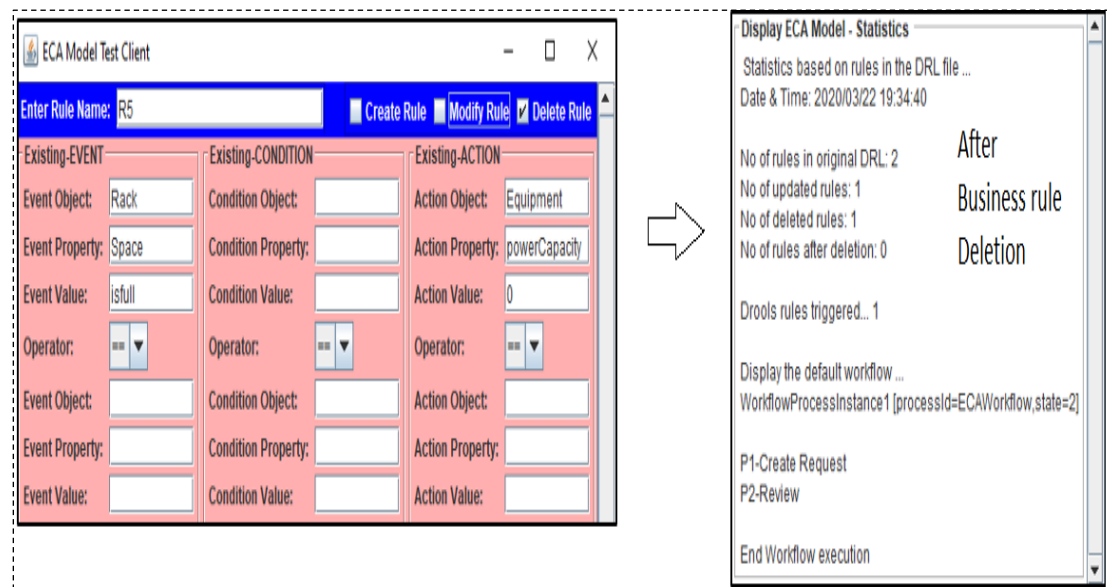


Figure 8.3.1. 7 Deleting R5 Event and Action Components via ECA Model Test Client

Using the proposed model, we can delete business rules on the fly. The ECA Model satisfies the adaptability and flexibility of deleting business rules and components. As mentioned before, with a guided user interface (ECA Model Test Client), a non-technical user should be able to delete business rules. As soon as the rules are deleted the DRL will be updated to reflect the change.

Business rules still are expressed in forms of simple statements. This is valuable, especially to avoid inconsistent syntax. As discussed above in ECA Model, business rules statements are formalized into business rules components (event, condition and action). They formatted and expressed in a simple way, easy to identify what part is event, condition or action for implementing them in the business rules management systems.

### **8.3.2 Validation of Business Rules Adaptation in Workflows**

The next set of experiments focus on validating the adaptation of business rules to control business processes in a workflow. Businesses must have dependable and flexible workflows to execute business processes. Reliability and flexibility are crucial issues because they help the business to become more efficient and effective. Hence, a validation process is required to not only assess the proposed model's ability to use business rules to control business processes but also to ensure that the adaptation process is accurate and reliable. Experiments 5-9 document validation of the key workflow constructs, which include initiating and terminating business processes as well as various business process flow patterns (sequential, parallel, merge, etc) as discussed in previous chapters. We believe these constructs are key elements for the functioning of a workflow and so it is important to validate these in our research.

#### **8.3.2.1 Enabling Business Rules Components to Initiate & Terminate Workflow**

##### Experiment 4

As demonstrated in section 6.4, the ECA Model algorithm provides initiating and terminating business rules constructs to enable a workflow to start and end. For modelling

the dynamic aspect of the workflow, the model provides the ability to modify initiating and terminating business rule components (event, condition and action) at runtime via the ECA Model Test Client.

Tests were performed using business rules in Use case #4. The business rule R101 is linked to succeeding business rules R102 and R103. R101's action component is connected to R102 and R103 via their condition components. Notice, R101 is not linked to any preceding business rules. We also see, business rules R102 and R104 have no succeeding business rules attached to them; both have preceding business rules. R102 is connected to R101 via condition-action relationships and R104 is connected to R103 via condition-action components. So, we can conclude that R101 is an initiating rule and (R102 & R104) are terminating rules. Figure 9.3.2.1 presents the relationships between business rules.

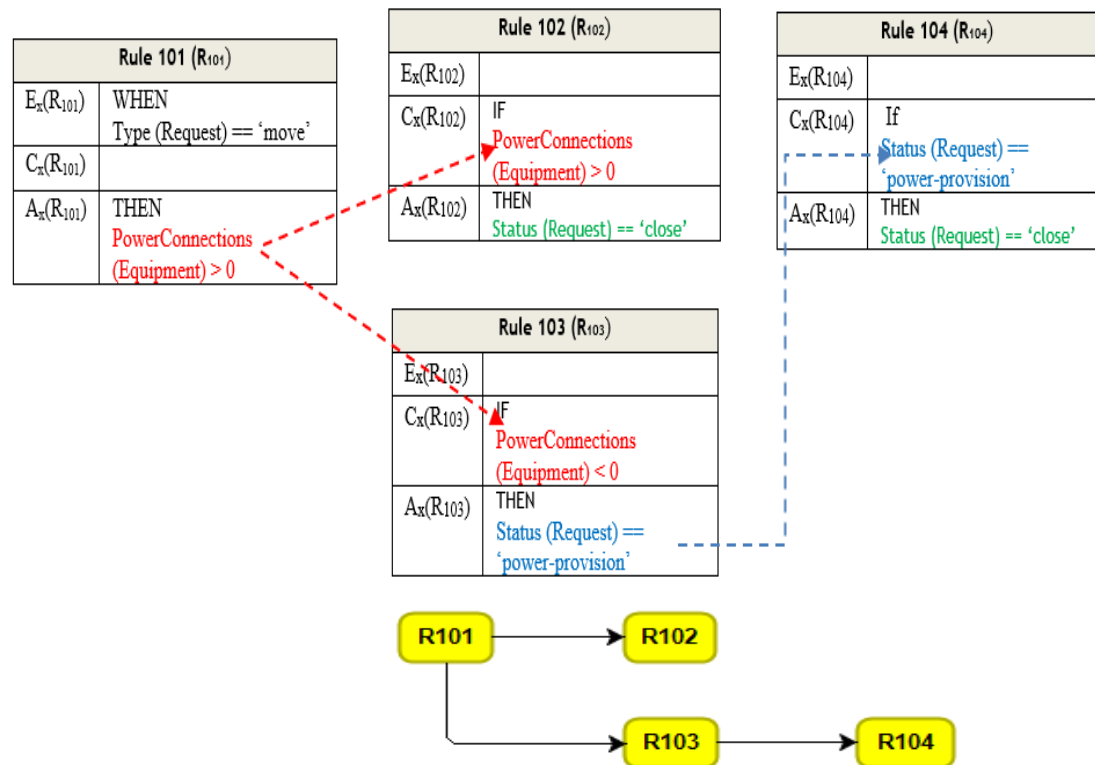


Figure 8.3.2. 1 Initiating (R101) and Terminating (R102 & R04) Business Rules



When business rule R101 and its relationships are generated using the dependency graph with a mapping table shown earlier in section 6.5, the R101 node is evaluated in which a corresponding starting business process node will be defined. Likewise, when business rules R102 and R104 and their relationships are generated, the R102 and R104 nodes are evaluated in which corresponding business processes and their links are formed. Figure 8.3.2.2 presents the business rules in DRL format. These rules are transformed into the dependency graph shown in Figure 8.3.2.2. For the users, the adaptation of business rules to transform “start” and “end” business processes is literally a matter of entering all business rules via the ECA Model Test Client. In the background, the ECA Model will proceed to generate business rules’ dependency graphs as soon as the rules are successfully executed in the rule engine, the business rule-process mapping table is generated via the adaptation algorithm described in section 6.5. The mapping table is generated to construct valid start and end processes based on rule relationships.

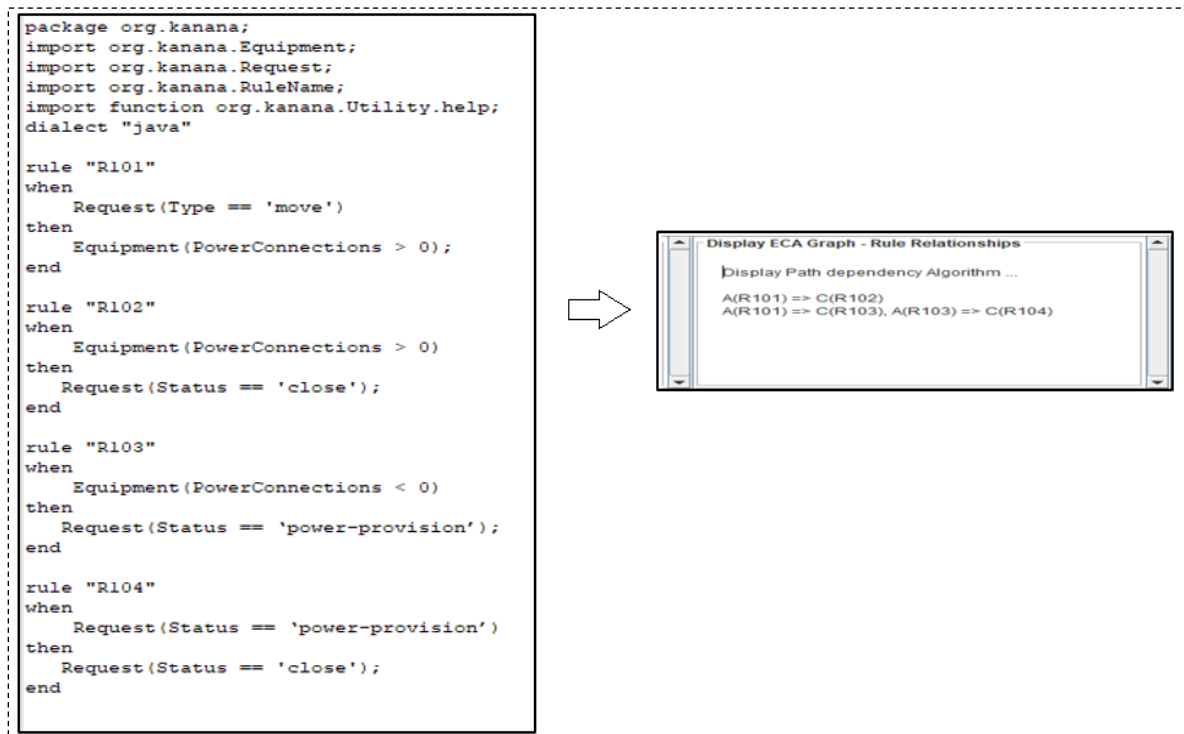


Figure 8.3.2. 2 Initiating (R101) and Terminating (R102 & R04) via ECA Model Test Client

By applying the business rules above, we can enable the workflow's start and end processes as shown Figure 8.3.2.3. As you can see, the same start and end processes as in the original workflow (Figure.8.2.2.3) are presented. However, it is worth noting the names of the process are currently based on the names of the business rules' properties. An extension to this work would be to allow the user to change process names at run time or generate a template of process names that can be used in the adaptation algorithm.

```

Start the ECA Model...

Use the ECA Model Test Client to load your rules...
Display the Workflow configuration ...

[main] INFO org.drools.compiler.kie.builder.impl.ClasspathKieProject - Found kmodule: file:/C:/Users/Kanana/workspace/AdaptiveECARuleModel/
[main] INFO org.drools.compiler.kie.builder.impl.KieRepositoryImpl - KieModule was added: FileKieModule[releaseId=ECARulesByGrace.group:Ada]

Display process in the Workflow...
P101
P102
P103
P104

Display relationships between generated business processes (Workflow)...
P101 --> P102
P101 --> P103; P103 --> P104

```

Figure 8.3.2. 3 R101 causing Start and (R102 & R04) causing End Workflow

If we analyse the dependency graph of our business rules (Figure 8.3.2.2), we notice something interesting: the root node (R01) enables the “start process” (P101) and the leaf business rule nodes (R102 and R104) enable the terminating processes (P102 and P04). By identifying the root and leaf business rules, we can determine and enable the initiating and terminating processes. The ECA Model implementation offers the ability to auto generate the initiating and terminating business processes by using defined business rules.

### 8.3.2.2 Enabling Sequential Process Flow Patterns

#### Experiment 5

As demonstrated in section 6.4, the ECA Model algorithm provides the Sequential Flow Rule construct to enable workflow processes to flow sequentially, one at the time. In this experiment, tests were carried out using business rules in Use case #5. The business rule

R201 is linked to a succeeding business rule R202. R201's action component is connected to R202 via event component. In this case, R201 preceding business rules are not important as flow is from P201 and P202. We also see that R202's succeeding business rule is R203 connected via an action-event component; the preceding business rules of R202 are R201- connected via event-action relationships. From this scenario, we can conclude that R201, R202 and R203 form a chain of business rules linked via connected business rules components. Figure 8.3.2.4 presents the relationships between business rules.

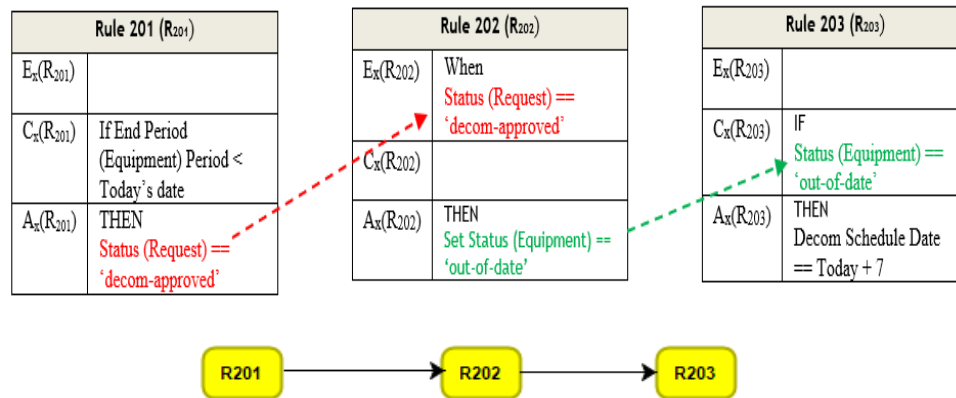


Figure 8.3.2. 4 Business Rules presenting Sequential Relationships

When business rule R201 and its relationships are generated using the dependency graph with a mapping table shown earlier in section 6.5, the R201 node is evaluated in which a corresponding business process node will be defined. Likewise, when business rules R202 and R203 and their relationships are generated, the R202 and R203 nodes are evaluated in which corresponding business processes and their links are formed. Figure 8.3.2.5 presents the business rules in DRL format. These rules are transformed into the dependency graph shown in Figure 8.3.2.5. For the users, the adaptation of business rules to enable sequential flow of business processes is literally a matter of entering all business rules via the ECA Model Test Client. In the background, the ECA Model will proceed to generate business rules' dependency graphs as soon as the rules are successfully executed in the rule engine, the business rule-process mapping table is generated via the adaptation algorithm described in section 6.5.

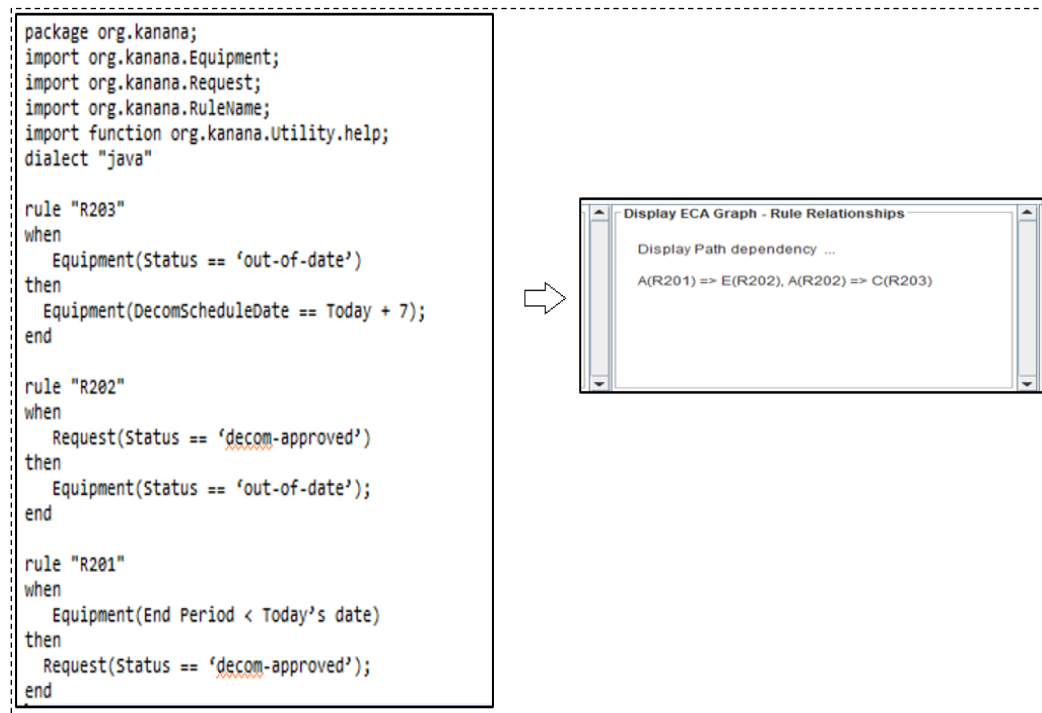


Figure 8.3.2. 5 R201, R202 and R203 and Relationships in DRL Format

By applying the business rules above, we can enable sequential flow of business process, from P201 to P202 and from P202 to P203 as shown Figure 8.3.2.6. As you can see, the business processes are chained together.

```
Start the ECA Model...

Use the ECA Model Test Client to load your rules...
Display the workflow configuration ...

[main] INFO org.drools.compiler.kie.builder.impl.ClasspathKieProject - Found kmodule: file:/C:/Users/Kanana/workspace/AdaptiveECARuleModel/
[main] INFO org.drools.compiler.kie.builder.impl.KieRepositoryImpl - KieModule was added: FileKieModule[releaseId=ECARulesByGrace.group:Ada]

Display process in the workflow
P201
P202
P203

Display relationships between generated business processes (workflow)...
P201 --> P202; P202 --> P203
```

Figure 8.3.2. 6 Business Rules and Relationships enabling Sequential Process Flows

Generated Paths: P201 → P202 → P203

If we analyse the dependency graph of our business rules (Figure 8.3.2.5), we notice something interesting: the intermediate node (R202) is connecting to both R201 and R203 to enable a chain of processes to flow sequentially. So, if we can observe this type of connectivity, we can determine whether sequential paths are to be generated for the workflow.

#### 5A) Insertion of business rule components

The proposed adaptation algorithm facilitates the insertion of a new business rules to support the insertion of business processes in a sequential flow situation. Consider a new business rule (R204) that is to be added to scenario in Use case#5. R204 is to ensure that when a request is set to decom-approve, then power connections are disconnected from the equipment to be removed. This forms a power decommission process (P204) that needs to be executed before the Approve process (P202). The new business rule R204 is inserted via the ECA Model Test Client then R204 and its relationships are generated using the dependency graph and a mapping table shown earlier in section 6.5, the R204 node is evaluated to which all connected business rule nodes are accessed and updated accordingly. Then a new process is created: the corresponding business process node and its relationship are updated accordingly. Figure 8.3.2.7 presents the business rules in DRL format, R204 is the new inserted business rule. Business rules are transformed into the dependency graph.

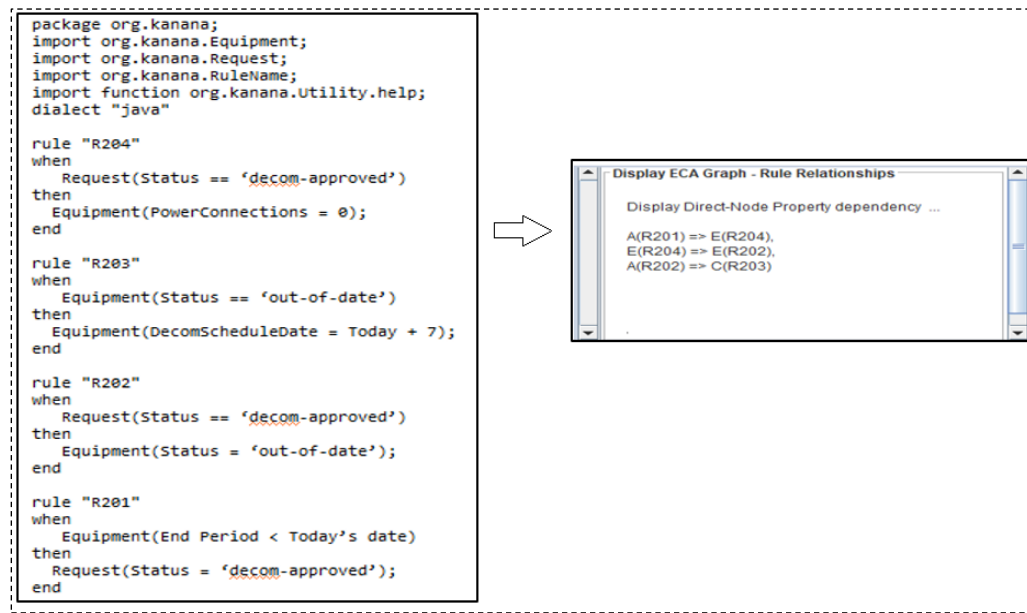


Figure 8.3.2. 7 Insert R204 and Relationships via ECA Model Test Client

By applying the business rules above, we can enable sequential flow to include new business process, from P201 to P204, from P04 to P202 and from P202 to P203 as shown Figure 8.3.2.8.

```

Start the ECA Model...

Use the ECA Model Test Client to load your rules...
Display the workflow configuration ...

[main] INFO org.drools.compiler.kie.builder.impl.ClasspathKieProject - Found kmodule: file:/C:/Users/Kanana/workspace/AdaptiveECARuleModel/
[main] INFO org.drools.compiler.kie.builder.impl.KieRepositoryImpl - KieModule was added: FileKieModule[releaseId=ECARulesByGrace.group:Ada]

Display process in the workflow
P201
P202
P203
P204

Display relationships between generated business processes (workflow)...
P201 --> P204; P204 --> P202; P202 --> P203

```

Figure 8.3.2. 8 Insertion of R204 causing Sequential Process Flows

Generated Paths: P201 → P204 → P202 → P203

#### 5B) Modify business rule (changing source or target process flows)

The proposed adaptation algorithm facilitates the modification of existing business rules to support the modification of business process in a sequential flow situation. Consider the Use case#5 scenario, whereby the user discovered that the business rule (R204) that has just been inserted to create a P204 process was wrongly positioned. The workflow was is supposed to flow from P201 → P202 → P204 → P203 and not P201 → P204 → P202 → P203. So, they would like to be able to update the business rule R204 so that when equipment is out of date, the power connections should be disconnected. The new business rule R204 is modified through the ECA Model Test Client and then its relationships are updated using the dependency graph and a mapping table shown earlier in section 6.5. The R204 node is evaluated to which all connected business rule nodes are accessed and updated accordingly and then the corresponding business process node and its relationship are updated accordingly. Figure 8.3.2.9 presents the business rules in DRL format, R204 is updated. The transformed business rules and their relationships are shown on the dependency graph.

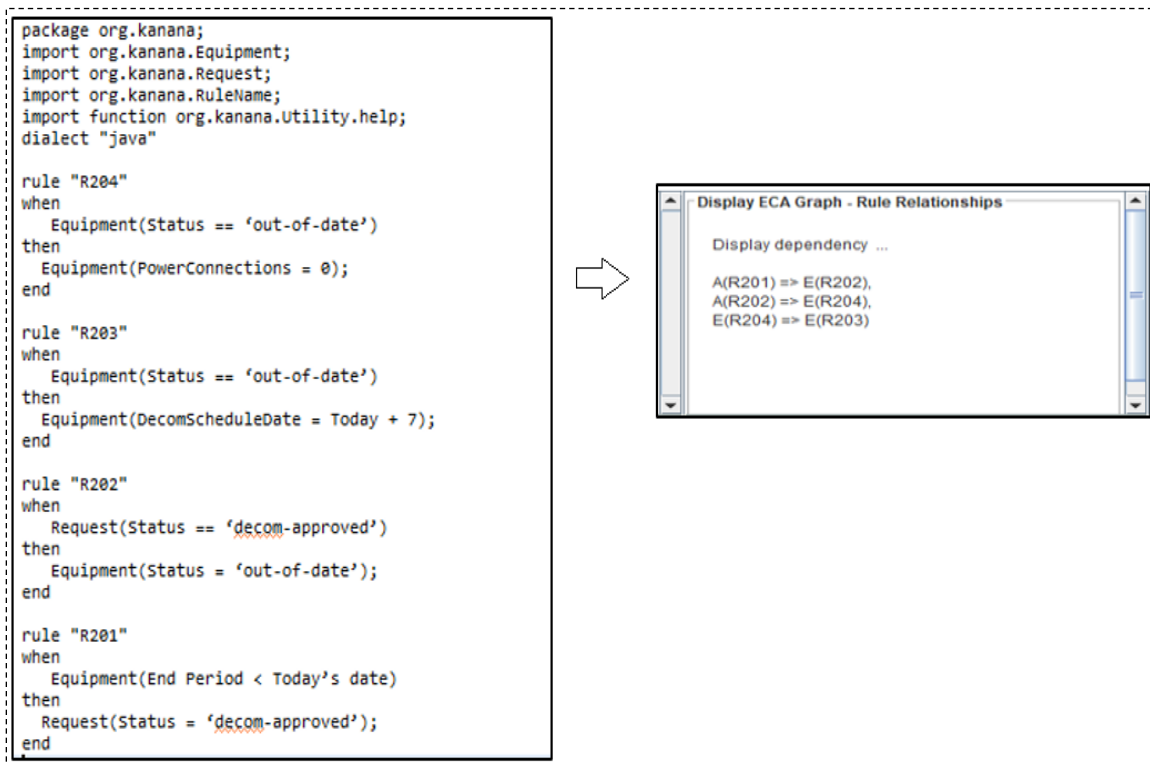


Figure 8.3.2. 9 Update R204 and Relationships via ECA Model Test Client

After applying the changes to R204, business processes are connected sequentially from P201 to P202, from P202 to P204 and from P204 to P203 as shown in Figure 8.3.2.10.

```
Start the ECA Model...

Use the ECA Model Test Client to load your rules...
Display the workflow configuration ...

[main] INFO org.drools.compiler.kie.builder.impl.ClasspathKieProject - Found kmodule: file:/C:/Users/Kanana/workspace/AdaptiveECARuleModel/
[main] INFO org.drools.compiler.kie.builder.impl.KieRepositoryImpl - KieModule was added: FileKieModule[releaseId=ECARulesByGrace.group:Ada

Display process in the workflow
P201
P202
P203
P204

Display relationships between generated business processes (workflow)...
P201 --> P202; P202 --> P204; P204 --> P203
```

Figure 8.3.2. 10 Modification of R204 causing Sequential Process Flows

Generated Path P201 → P202; P202 → P204; P204 → P203

### 5C) Deletion of existing business rules - disconnecting existing process flows

The proposed adaptation algorithm facilitates the deletion of existing business rules to support the deletion of business processes in a sequential flow situation. Consider the Use case#5 scenario, whereby the user would like to remove P204 from the workflow, which will involve deletion of R204. Business rule deletion is straight forward. Removing R204 will remove P204 as well as all connections from source to destination.

The business rule R204 is deleted through the ECA Model Test Client and then its relationships are updated using the dependency graph and a mapping table shown earlier in section 6.5. The R204 node is evaluated to which all connected business rule nodes are removed accordingly and then, the corresponding business process node and its relationships are updated accordingly. Figure 8.3.2.11 presents the business rules in DRL format, R204 is deleted. The transformed business rules and their relationships are shown on the dependency graph.

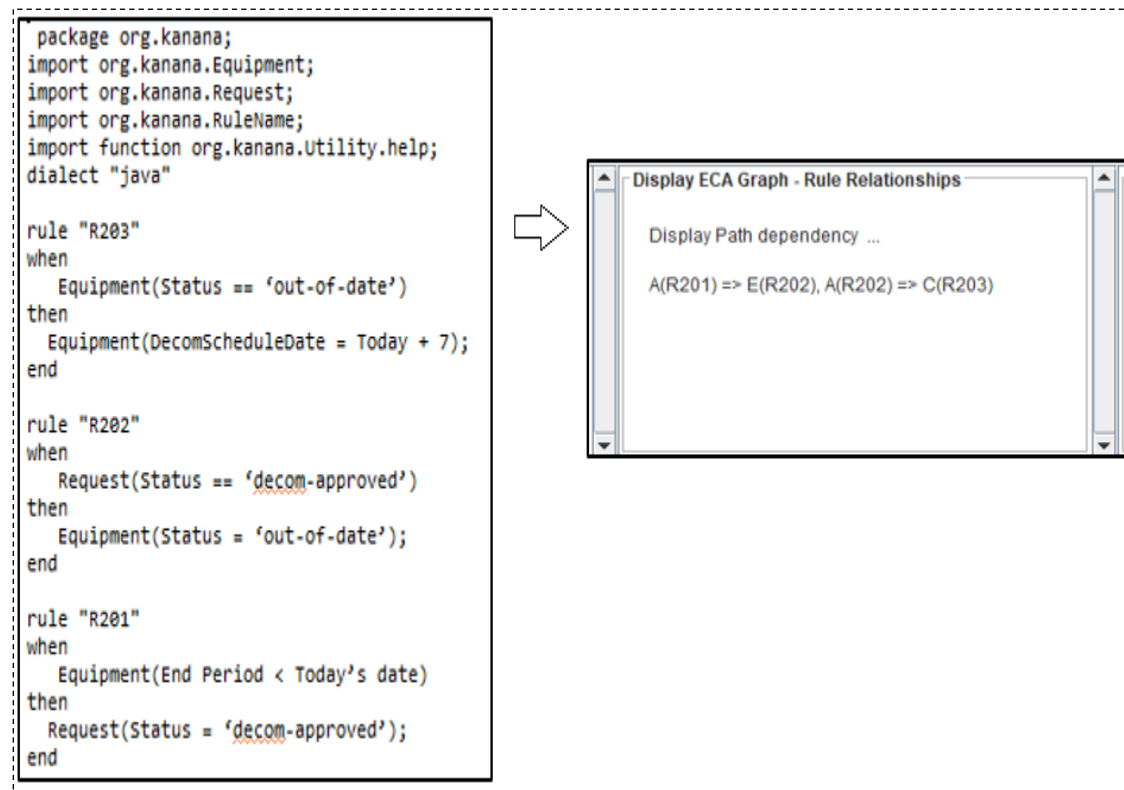


Figure 8.3.2. 11 Delete R204 and Relationships via ECA Model Test Client



After deleting business rule R204, business processes are connected sequentially from P201 to P202, from P202 to P204 and from P204 to P203 as shown in Figure 8.3.2.12.

```
Start the ECA Model...
Use the ECA Model Test Client to load your rules...
Display the workflow configuration ...

[main] INFO org.drools.compiler.kie.builder.impl.ClasspathKieProject - Found kmodule: file:/C:/Users/Kanana/workspace/AdaptiveECARuleModel/
[main] INFO org.drools.compiler.kie.builder.impl.KieRepositoryImpl - KieModule was added: FileKieModule[releaseId=ECARulesByGrace.group:Ada]

Display process in the workflow
P201
P202
P203

Display relationships between generated business processes (workflow)...
P201 --> P202; P202 --> P203
```

Figure 8.3.2. 12 Deletion of R204 causing removal of Process P204 and Connections

Generated Paths: P201 → P202; P202 → P203 after Business Rules (R204) deleted

### 8.3.2.3 Enabling Parallel-OR Merge Process Flow Patterns

#### Experiment 6

As demonstrated in Chapter 6.4, the ECA Model algorithm provides the Parallel Merge Flow Rule construct with an OR disjunction operator to enable workflow processes to form Parallel-OR Merge flow patterns. In this experiment, tests were carried out using business rules in Use case #6. The business rule R301 is linked to succeeding business rules R302 and R303. R301's action component invokes both business rules R302 and R303 but only one gets to activate R304 using the disjunction "OR". Business rules R302 and R303 are connected to R304 via action-condition components. Figure 8.3.2.13 presents the relationships between business rules.

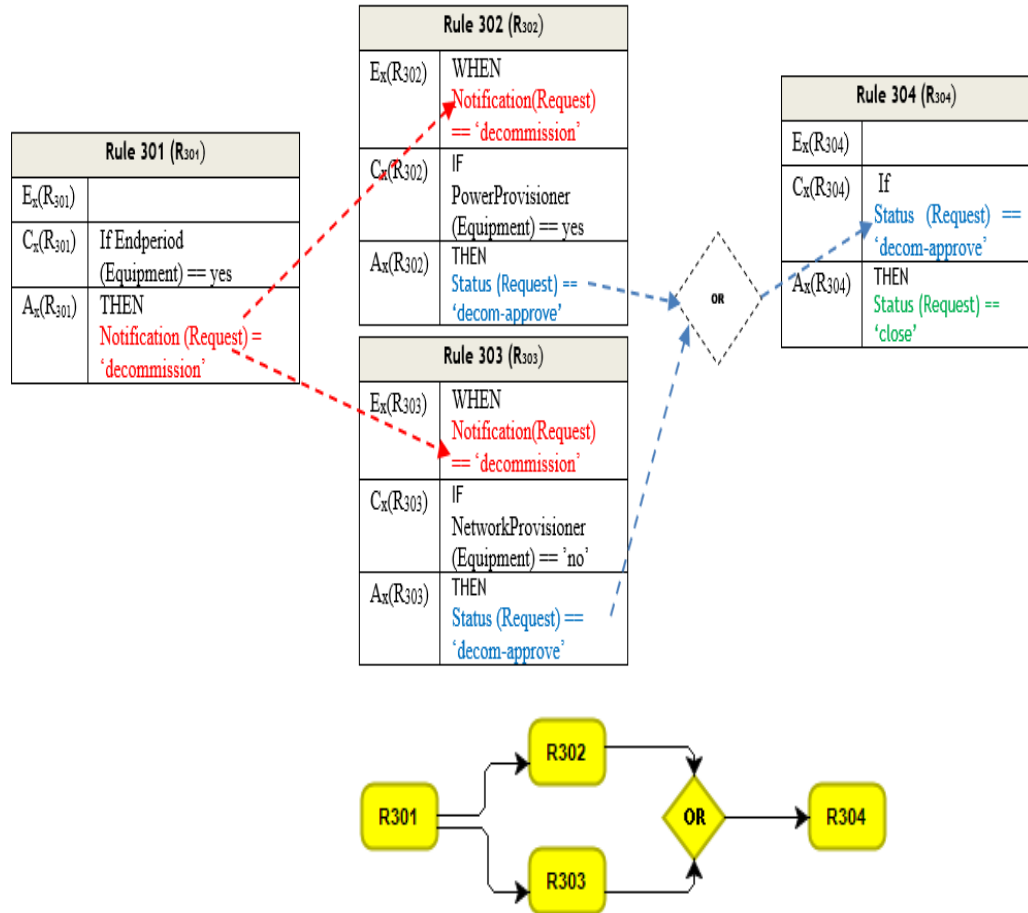


Figure 8.3.2. 13 Business Rules presenting Parallel Merge-OR Relationships

When business rules R301, R302, R303, R304 and their relationships are generated using the dependency graph with a mapping table shown earlier in section 6.5, the corresponding business processes and their links are formed. Figure 8.3.2.14 presents the business rules in DRL format. These business rules are transformed into the dependency graph.

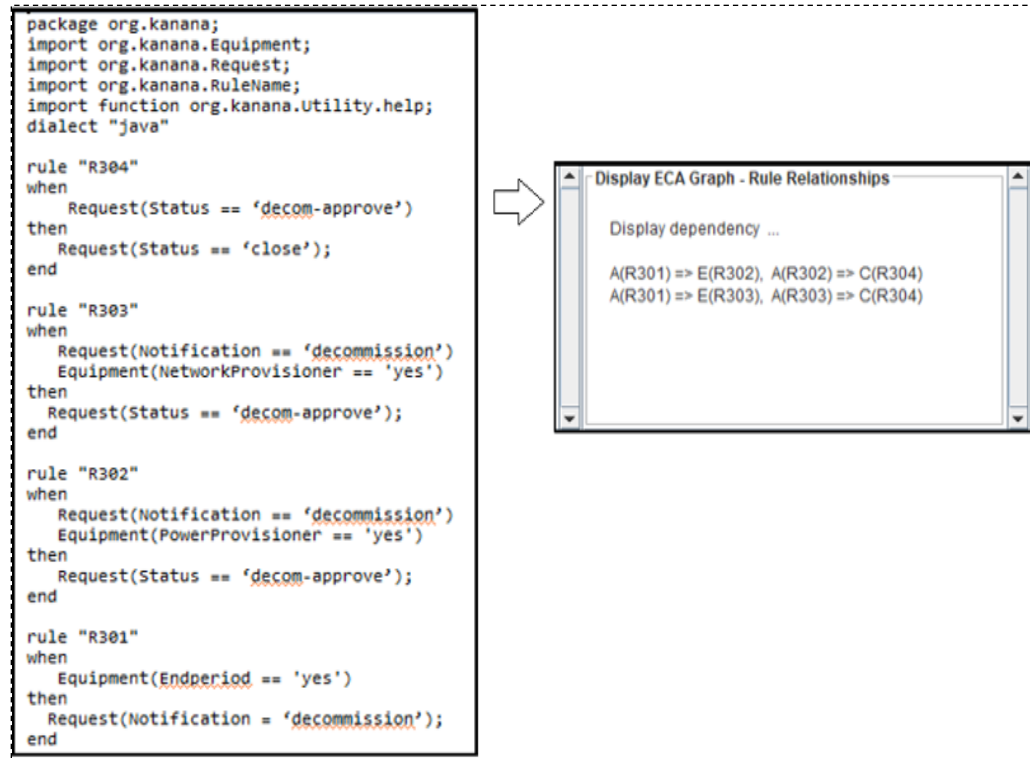


Figure 8.3.2. 14 R301, R302, R303, R304 and Relationships in DRL Format

By applying the business rules above, we can enable merged-OR flow patterns of a workflow. Two Parallel-OR Merge paths are generated, (from P301 to P302 and from P302 to P304 OR from P301 to P303 and from P303 to P304) as shown Figure 8.3.2.15.

```

Start the ECA Model...

Use the ECA Model Test Client to load your rules...
Display the workflow configuration ...

[main] INFO org.drools.compiler.kie.builder.impl.ClasspathKieProject - Found kmodule: file:/C:/Users/Kanana/workspace/AdaptiveECARuleModel/
[main] INFO org.drools.compiler.kie.builder.impl.KieRepositoryImpl - KieModule was added: FileKieModule[releaseId=ECARulesByGrace.group:Ada]

Display process in the workflow
P301
P302
P303
P304

Display relationships between generated business processes (workflow)...
P301 --> P302; P302 --> P304
P301 --> P303; P303 --> P304

```

Figure 8.3.2. 15 Business Rules & Relationships enabling Parallel-OR Merge Process Flows

Parallel-OR Merge paths 1) P301 → P302 → P304 OR 2) P301 → P303 → P304

## 6A) Insertion of business rule components

The proposed adaptation algorithm facilitates the insertion of new business rules to support the insertion of business processes in a parallel-or merged flow situation. Let us consider, a new business rule (R305) that is to be added to the scenario in Use case#6. R305 is to schedule the equipment for decommission when a request is set to decom-approve and then the equipment status is removed. This forms a scheduling decommission process (P305) that needs to be executed before the Request Close process (P304). The new business rule R305 is inserted via the ECA Model Test Client and then R305 and its relationships are generated using the dependency graph and a mapping table shown earlier in section 6.5. The R305 node is evaluated to which all connected business rule nodes are accessed and updated accordingly. Then the new process is created, corresponding business process nodes and relationships are updated accordingly. Figure 8.3.2.16 presents the business rules in DRL format, R305 is the new inserted business rule. The rules are transformed into the dependency graph.

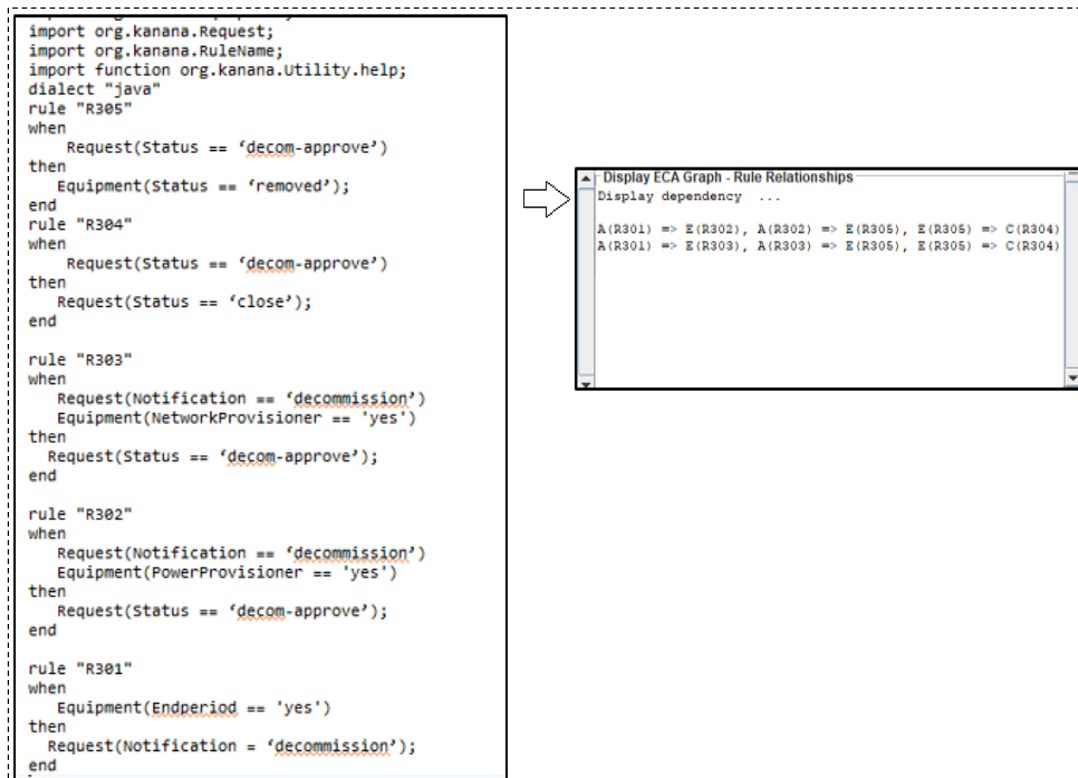


Figure 8.3.2. 16 Insert R305 and Relationships via ECA Model Test Client

By applying the business rules above, we can enable merged-OR flow patterns and insert a new process in a workflow. Two Parallel-OR Merged paths are generated, (from P301 to P302, from P302 to P305 and from P305 to P304 OR from P301 to P303, from P303 to P305 and from P305 to P304) as shown Figure 8.3.2.17.

```

Start the ECA Model...

Use the ECA Model Test Client to load your rules...
Display the workflow configuration ...

[main] INFO org.drools.compiler.kie.builder.impl.ClasspathKieProject - Found kmodule: file:/C:/Users/Kanana/workspace/AdaptiveECARuleModel/
[main] INFO org.drools.compiler.kie.builder.impl.KieRepositoryImpl - KieModule was added: FileKieModule[releaseId=ECARulesByGrace.group:Ada

Display process in the workflow
P301
P302
P303
P304
P305

Display relationships between generated business processes (workflow)...
P301 --> P302; P302 --> P305; P305 --> P304
P301 --> P303; P303 --> P305; P305 --> P304

```

Figure 8.3.2. 17 Insertion of R305 causing Parallel-OR Merge Process Flows

Parallel-OR Merged paths 1) P301 → P302 → P305 → P304 OR 2) P301 → P303 → P305 → P304

6B) Modify business rule (changing properties of the of the business rule hence process)  
The proposed adaptation algorithm facilitates the modification of existing business rules to support the modification of business process in a parallel-or merged flow situation. Consider the Use case#6 scenario, whereby business rule (R303) is modified because P303 process is to consider not just network provisioner but also storage provisioner. The business rule R303 is modified through the ECA Model Test Client and then its relationships are updated using the dependency graph and a mapping table shown earlier in section 6.5. The R303 node is evaluated, to which all connected business rule nodes are accessed and updated accordingly. Then, the corresponding business process node and its relationships are updated accordingly. Figure 8.3.2.18 presents the business rules in DRL format. R303 is updated. The transformed business rules and their relationships are shown on the dependency graph Figure 8.3.2.18

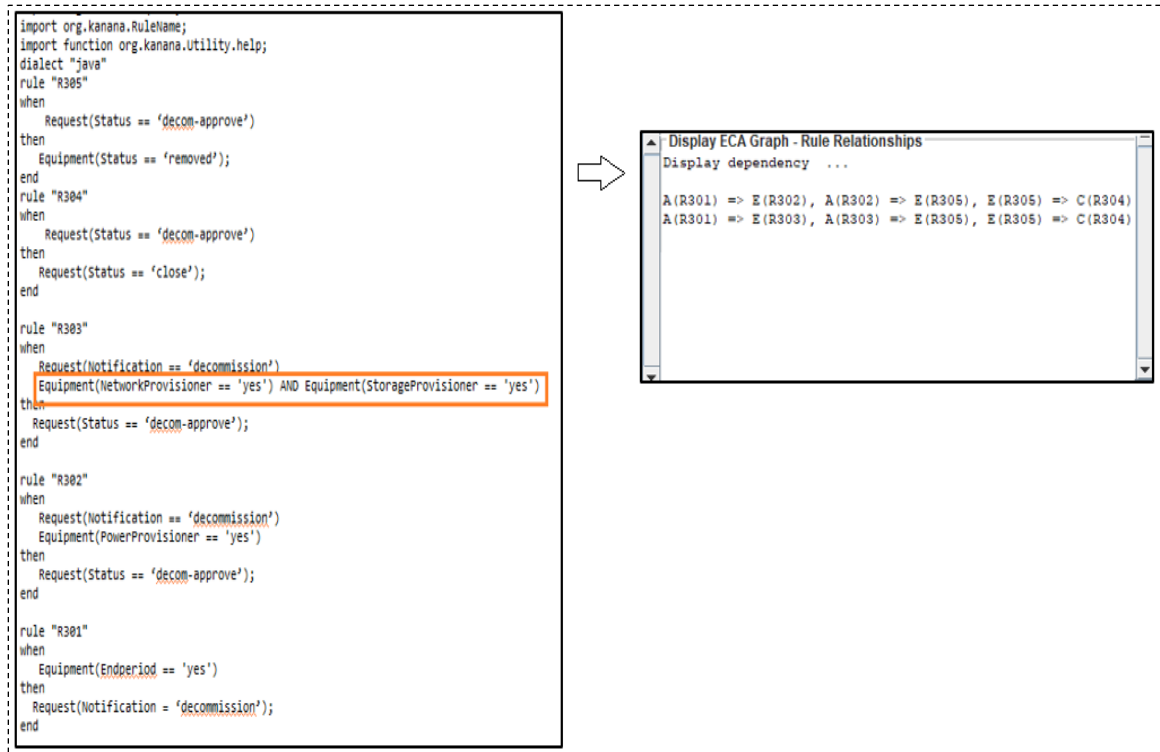


Figure 8.3.2. 18 Update of R303 and Relationships via ECA Model Test Client

By applying the business rules above, we can enable Parallel-OR Merged flow patterns and insert a new process in a workflow. Two parallel merged-or paths are generated, (from P301 to P302, from P302 to P305 and from P305 to P304 OR from P301 to P303, from P303 to P305 and from P305 to P304) as shown Figure 8.3.2.19.

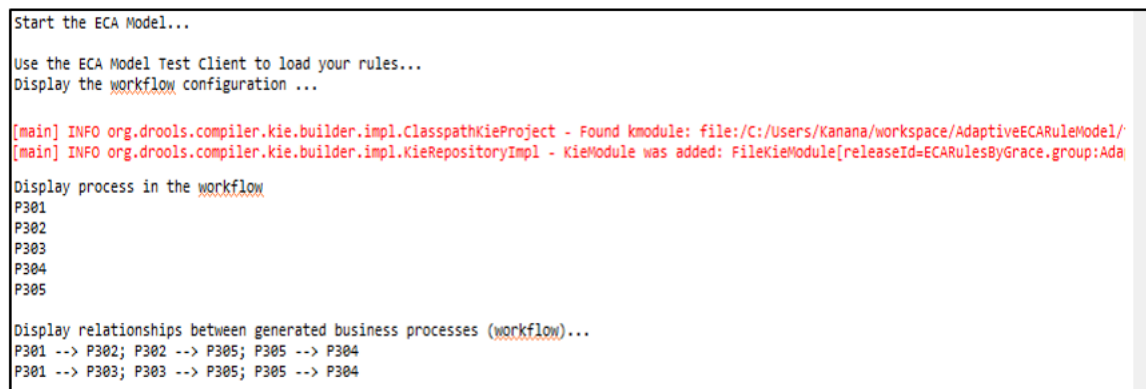


Figure 8.3.2. 19 Modification of R303 enabling Parallel-OR Merge Process Flows

Parallel-OR merged paths 1) P301 → P302 → P305 → P304 OR 2) P301 → P303 → P305 → P304

### 6C) Deletion of existing business rules - disconnecting existing process and flows

The proposed adaptation algorithm facilitates the deletion of existing business rules to support the deletion of business process in a parallel-or merged flow situation. Consider the Use case#5 scenario, whereby the user would like to remove P302 from the workflow, which will involve deletion of R302. Business rule deletion is straight forward. Removing R302 will remove P302 as well as all connections from source to destination. The business rule R302 is deleted through the ECA Model Test Client and then its relationships are updated using the dependency graph and a mapping table shown earlier in section 6.5. The R302 node is evaluated to which all connected business rule nodes are removed accordingly and then the corresponding business process node and its relationship are updated accordingly. Figure 8.3.2.20 presents the business rules in DRL format. R302 is deleted. The transformed business rules and their relationships are shown on the dependency graph.

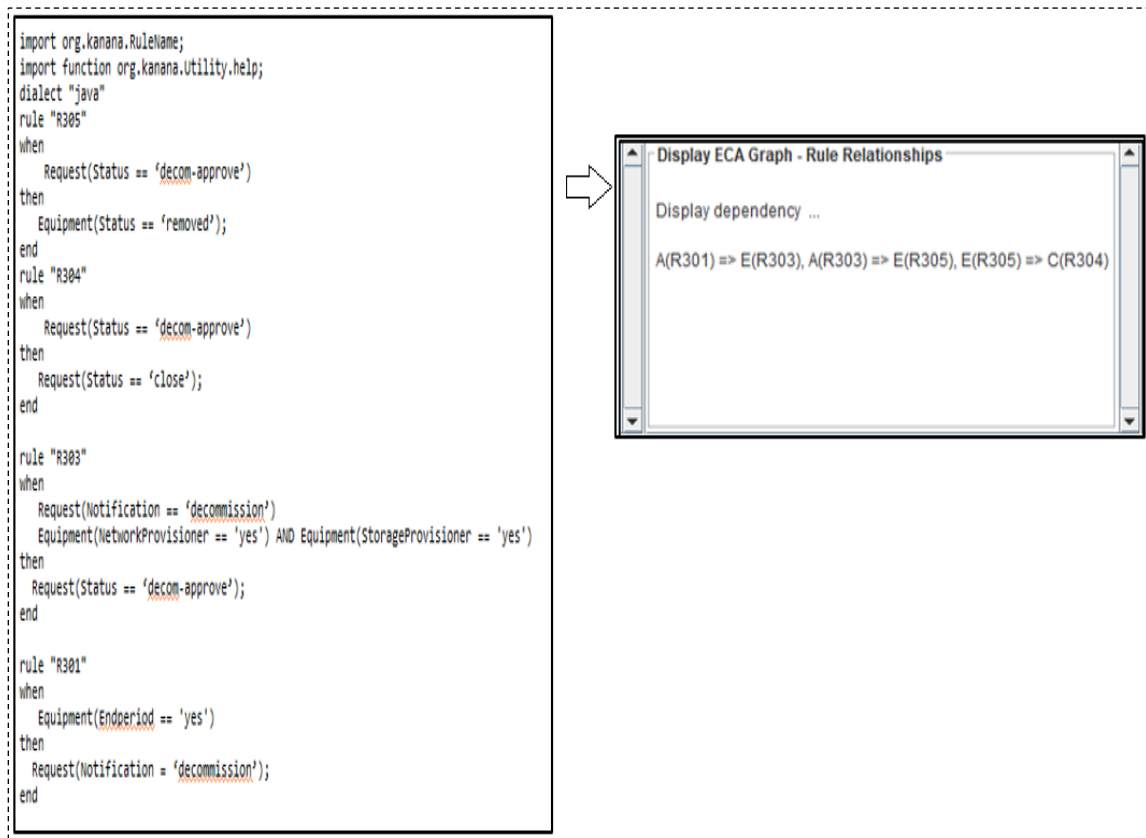


Figure 8.3.2. 20 Deletion of R302 and Relationships via ECA Model Test Client

After deleting R302, business processes are connected sequentially from P301 to P303, from P303 to P305 and from P305 to P304 as shown in Figure 8.3.2.21. Notice P302 and its dependencies have been removed.

```
Start the ECA Model...

Use the ECA Model Test Client to load your rules...
Display the workflow configuration ...

[main] INFO org.drools.compiler.kie.builder.impl.ClasspathKieProject - Found kmodule: file:/C:/Users/Kanana/workspace/AdaptiveECARuleModel/target/c
[main] INFO org.drools.compiler.kie.builder.impl.KieRepositoryImpl - KieModule was added: FileKieModule[releaseId=ECARulesByGrace.group:AdaptiveECA

Display process in the workflow

P301
P303
P304
P305

Display relationships between generated business processes (workflow)...

P301 --> P303; P303 --> P305; P305 --> P304
```

Figure 8.3.2. 21 Deletion of R302 causing Sequential Process Flows

Sequential Paths 1) P301 → P303 → P305; P305 → P304 generated after deletion

### 8.3.2.4 Enabling Parallel-AND Merge Process Flow Patterns

#### Experiment 7

As demonstrated in section 6.4, the ECA Model algorithm provides the Parallel-OR Merge Flow Rule construct with an AND conjunction operator to enable workflow processes to form Parallel-OR Merge flow patterns. In this experiment, tests were carried out using business rules in Use case #7. The business rule R301 is linked to succeeding business rules R302 and R303. R301's action component invokes both business rules R302 and R303 but only one gets to activate R304 using the conjunction "AND". Business rules R302 and R303 are connected to R304 via action-condition components. Figure 8.3.2.21.1 presents the relationships between business rules.



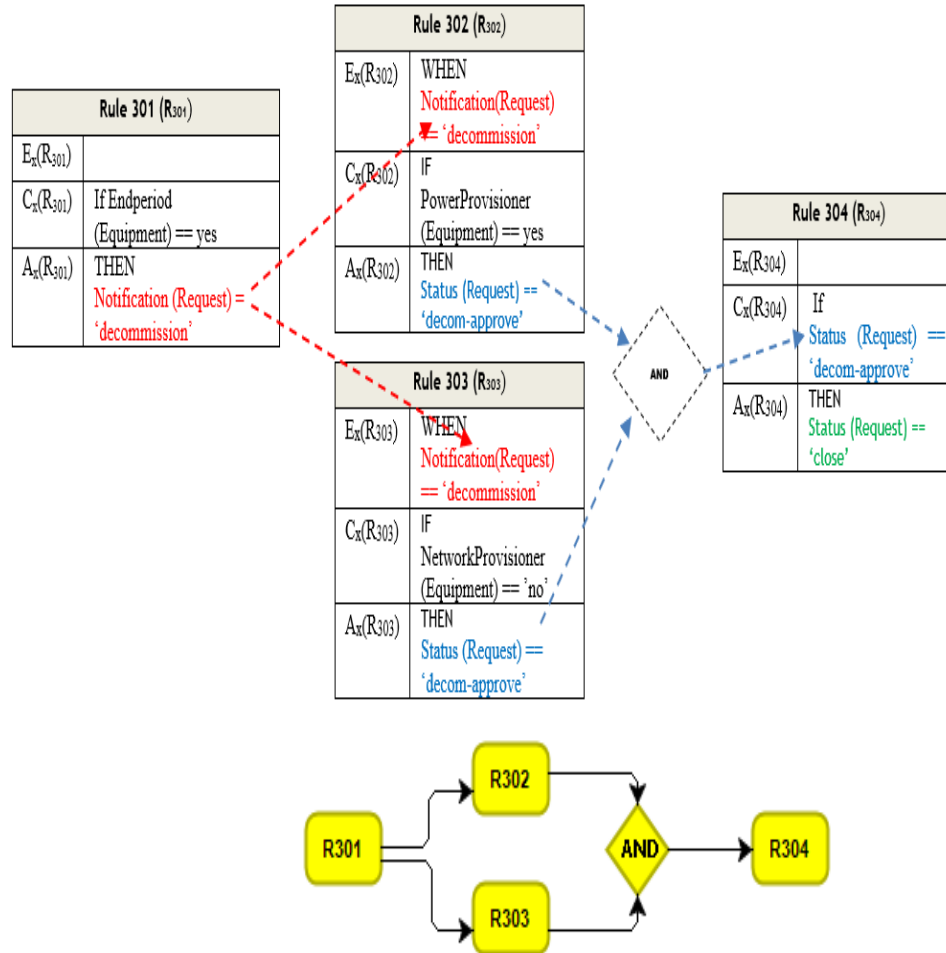


Figure 8.3.2.21. 1 Business Rules presenting Parallel-AND Merge Relationships

When business rules R301, R302, R303, R304 and their relationships are generated using the dependency graph with a mapping table shown earlier in section 6.5, the corresponding business processes and their links are formed. Figure 8.3.2.22 presents the business rules in DRL format. These rules are transformed into the dependency graph.

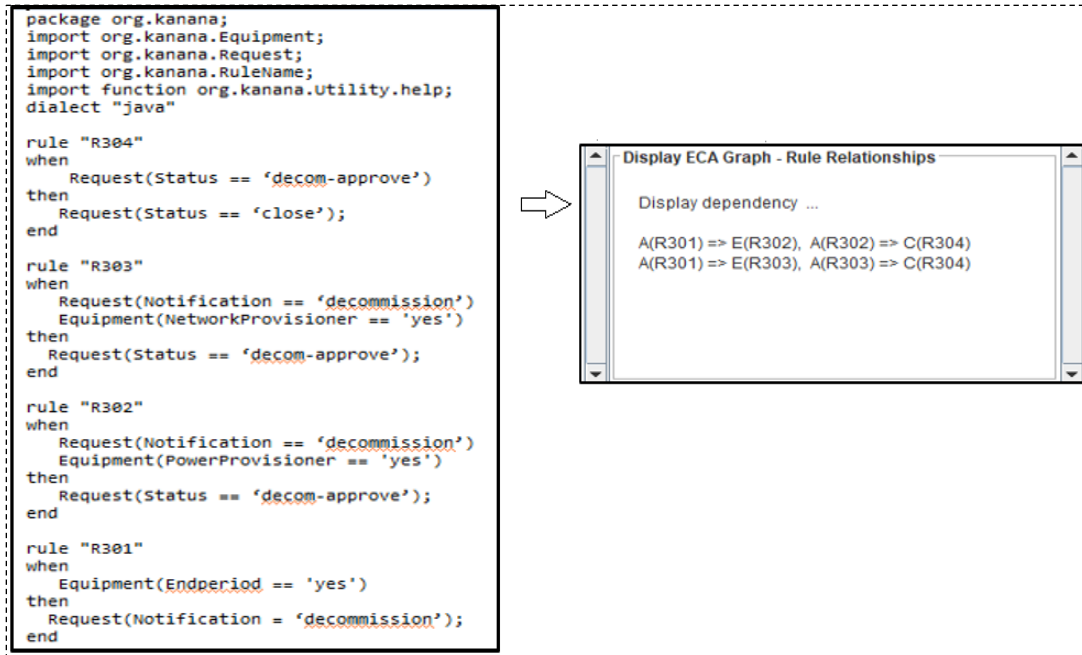


Figure 8.3.2. 22 R301, R302, R303, R304 and Relationships in DLR format

By applying the business rules above, we can enable Parallel-AND Merge flow patterns of a workflow. Two parallel merged-and paths are generated, (from P301 to P302 and from P302 to P304 AND from P301 to P303 and from P303 to P304) as shown Figure 8.3.2.23.

```

Start the ECA Model...

Use the ECA Model Test Client to load your rules...
Display the workflow configuration ...

[main] INFO org.drools.compiler.kie.builder.impl.ClasspathKieProject - Found kmodule: file:/C:/Users/Kanana/workspace/AdaptiveECARuleModel/
[main] INFO org.drools.compiler.kie.builder.impl.KieRepositoryImpl - KieModule was added: FileKieModule[releaseId=ECARulesByGrace.group:Ada]

Display process in the workflow
P301
P302
P303
P304

Display relationships between generated business processes (workflow)...
P301 --> P302; P302 --> P304
P301 --> P303; P303 --> P304

```

Figure 8.3.2. 23 Business Rules & Relationships enabling Parallel-AND Process Flows

Parallel-AND paths 1) P301 → P302 → P304 or 2) P301 → P303 → P304

#### 7A) Insertion of business rule components

The proposed adaptation algorithm facilitates the insertion of a new business rules to support the insertion of business processes in a parallel-or merged flow situation. Let us consider a new business rule (R305) that is to be added to scenario in Use case #7. R305 is

to schedule the equipment for decommission when a request is set to decom-approve then the equipment status is set to removed. This forms a scheduling decommission process (P305) that needs to be executed before the Request Close process (P304). The new business rule R305 is inserted via the ECA Model Test Client then R305 and its relationships are generated using the dependency graph and a mapping table shown earlier in section 6.5. The R305 node is evaluated to which all connected business rule nodes are accessed and updated accordingly. Then the new process is created, corresponding business process nodes and relationships are updated accordingly. Figure 8.3.2.24 presents the business rules in DRL format, R305 is the new inserted business rule. The rules are transformed into the dependency graph.

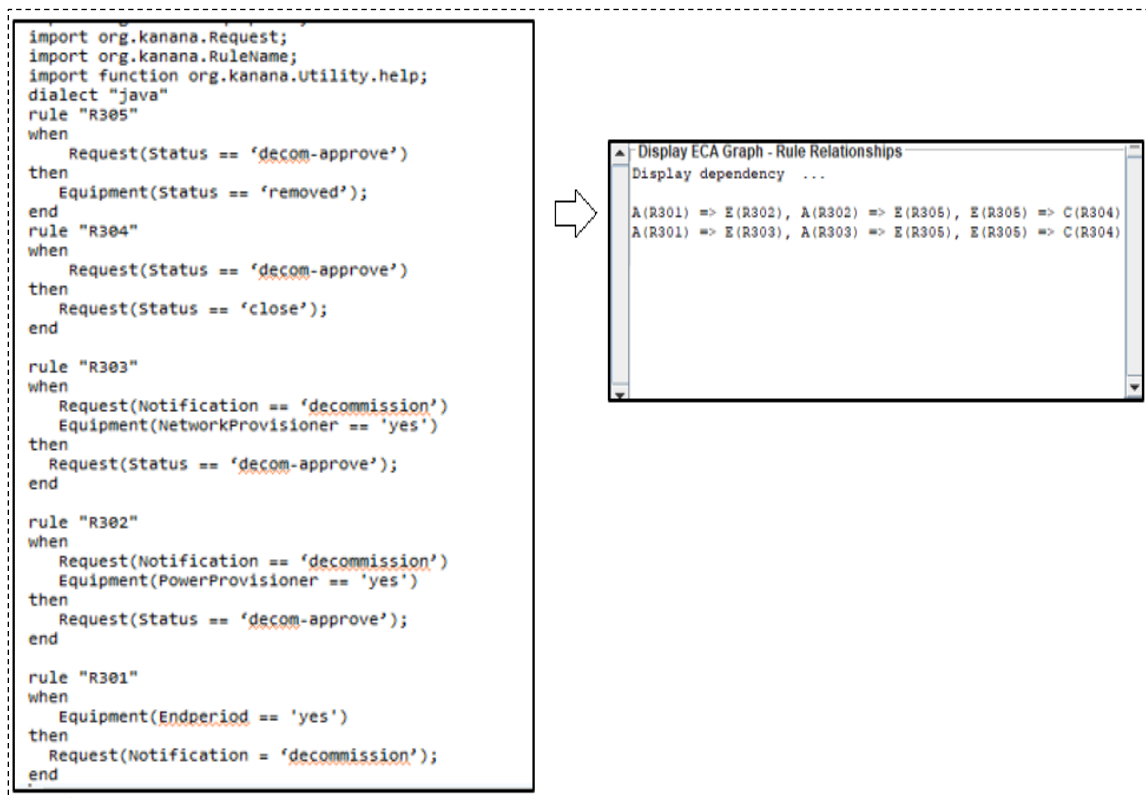


Figure 8.3.2. 24 Insert R305 and Relationships via ECA Model Test Client

By applying the business rules above, we can enable Parallel-AND Merge flow patterns and insert a new process in a workflow. Two parallel merged-or paths are generated, (from P301 to P302, from P302 to P305 and from P305 to P304 AND from P301 to P303, from P303 to P305 and from P305 to P304) as shown Figure 8.3.2.25.

```

Start the ECA Model...

Use the ECA Model Test Client to load your rules...
Display the workflow configuration ...

[main] INFO org.drools.compiler.kie.builder.impl.ClasspathKieProject - Found kmodule: file:/C:/Users/Kanana/workspace/AdaptiveECARuleModel/
[main] INFO org.drools.compiler.kie.builder.impl.KieRepositoryImpl - KieModule was added: FileKieModule[releaseId=ECARulesByGrace.group:Ada

Display process in the workflow
P301
P302
P303
P304
P305

Display relationships between generated business processes (workflow)...
P301 --> P302; P302 --> P305; P305 --> P304
P301 --> P303; P303 --> P305; P305 --> P304

```

Figure 8.3.2. 25 Insertion of R305 causing Parallel-AND Process Flows

Parallel-AND Merge paths 1) P301 → P302 → P305 → P304 AND 2) P301 → P303 → P305 → P304

#### 7B) Modify business rule (changing properties of the of the business rule hence process)

The proposed adaptation algorithm facilitates the modification of existing business rules to support the modification of business process in a parallel-or merged flow situation. Consider the Use case #7 scenario, whereby business rule (R303) is modified because P303 process is to consider not just network provisioner but also storage provisioner. The business rule R303 is modified through the ECA Model Test Client and then its relationships are updated using the dependency graph and a mapping table shown earlier in section 6.5. The R303 node is evaluated to which all connected business rule nodes are accessed and updated accordingly. Then the corresponding business process node and its relationship are updated accordingly. Figure 8.3.2.26 presents the business rules in DRL format. R303 is updated. The transformed business rules and their relationships are shown on the dependency graph.

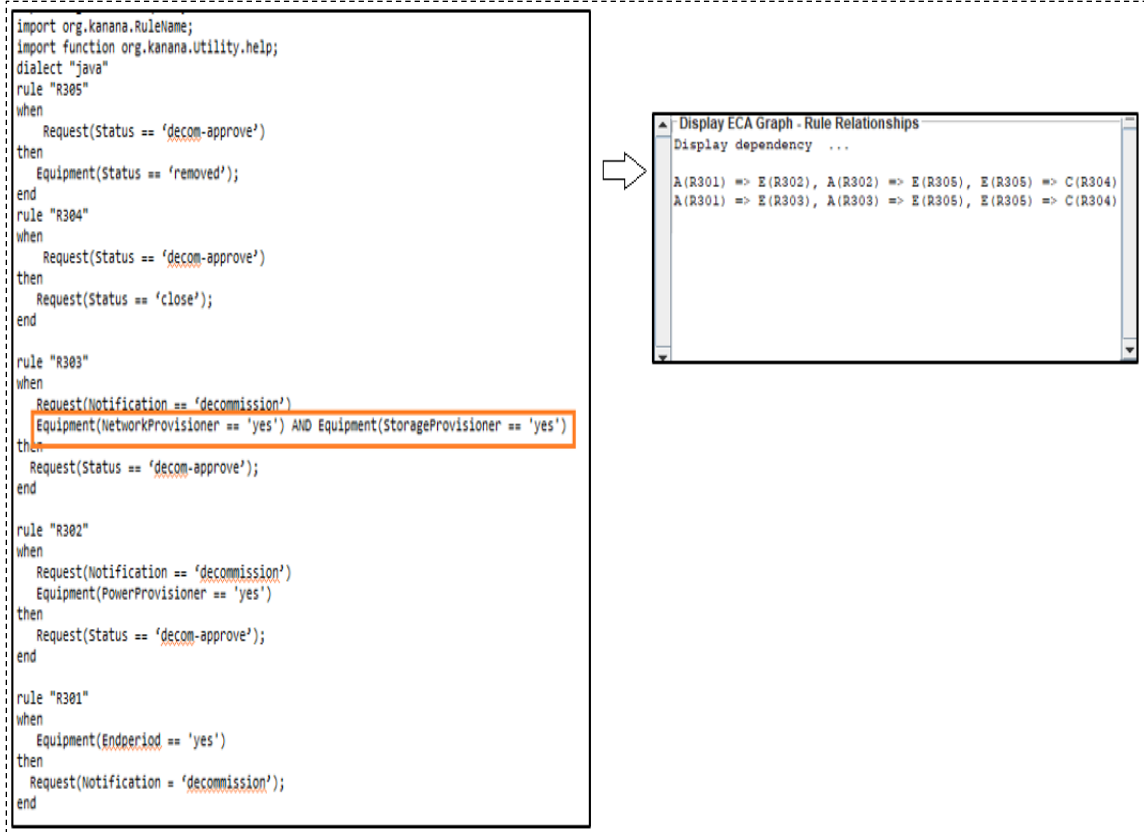


Figure 8.3.2. 26 Update R303 and Relationships via ECA Model Test Client

By applying the above business rules, we can enable Parallel-AND Merge flow patterns and modify a process in a workflow. Two parallel merged-or paths are generated, (from P301 to P302, from P302 to P305 and from P305 to P304 AND from P301 to P303, from P303 to P305 and from P305 to P304) as shown Figure 8.3.2.27.

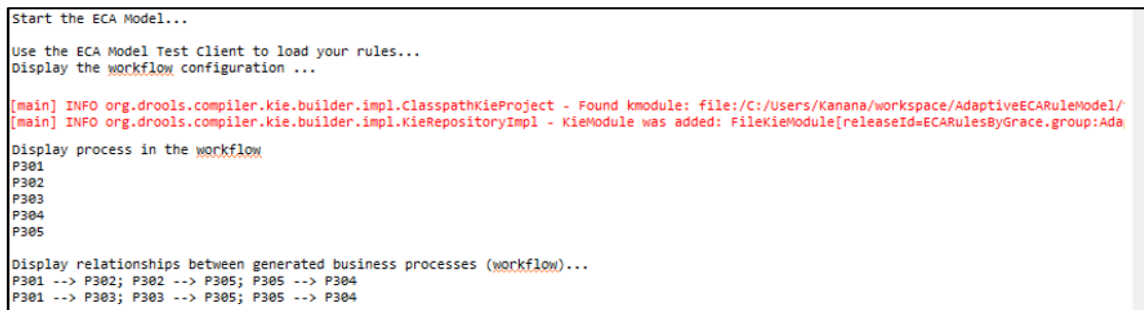


Figure 8.3.2. 27 Modification of R303 causing Parallel-AND Merge Process Flows

Parallel-AND Merge paths 1) P301→P302→P305→P304 AND 2) P301→P303→P305 → P304

### 7C) Deletion of existing business rules - disconnecting existing process and flows

The proposed adaptation algorithm facilitates the deletion of existing business rules to support the deletion of business process in a parallel-and merged flow situation. Consider the Use case #7 scenario, whereby the user would like to remove P302 from the workflow, which will involve deletion of R302. Business rule deletion is straight forward. Removing R302 will remove P302 as well as all connections from source to destination. The business rule R302 is deleted through the ECA Model Test Client and then its relationships are updated using the dependency graph and a mapping table shown earlier in section 6.5. The R302 node is evaluated to which all connected business rule nodes are removed accordingly. Then the corresponding business process node and its relationship are updated accordingly. Figure 8.3.2.28 presents the business rules in DRL format. R302 is deleted. The transformed business rules and dependency graph.

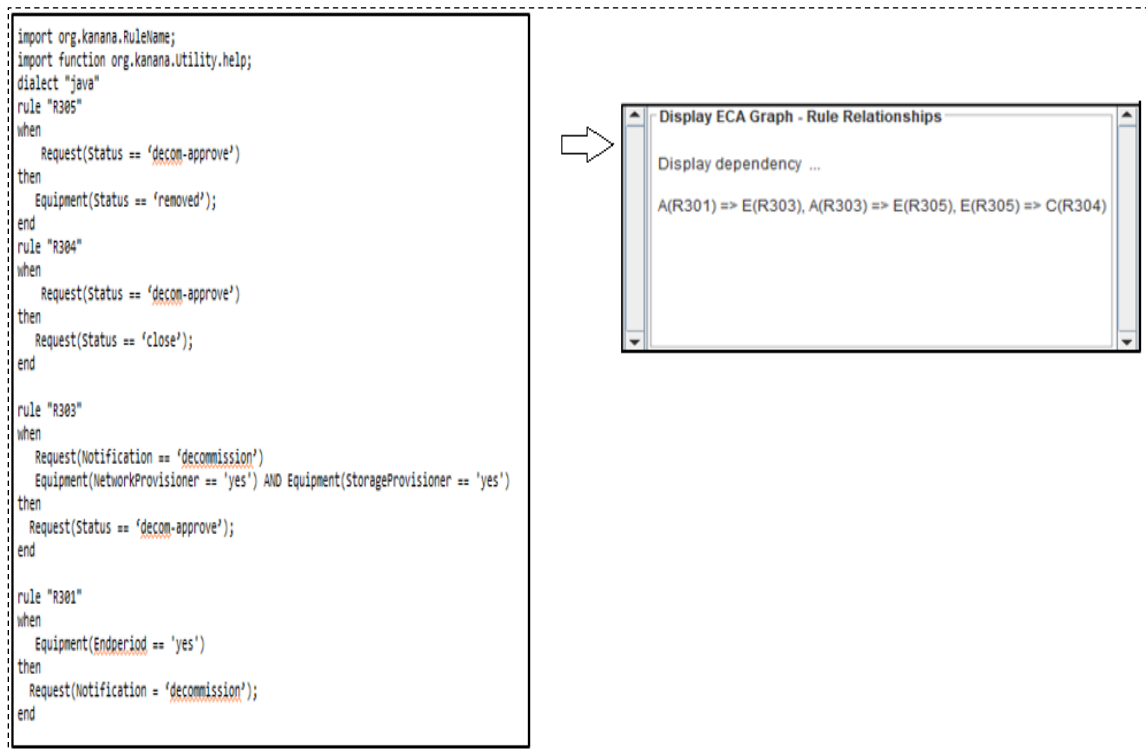


Figure 8.3.2. 28 Delete R302 and Relationships via ECA Model Test Client

After deleting R302, business processes are connected sequentially from P301 to P303, from P303 to P305 and from P305 to P304 as shown in Figure 8.3.2.29. Notice P302 and its dependencies have been removed.

```
Start the ECA Model...

Use the ECA Model Test Client to load your rules...
Display the workflow configuration ...

[main] INFO org.drools.compiler.kie.builder.impl.ClasspathKieProject - Found kmodule: file:/C:/Users/Kanana/workspace/AdaptiveECARuleModel/target/c
[main] INFO org.drools.compiler.kie.builder.impl.KieRepositoryImpl - KieModule was added: FileKieModule[releaseId=ECARulesByGrace.group:AdaptiveECA

Display process in the workflow

P301
P303
P304
P305

Display relationships between generated business processes (workflow)...

P301 --> P303; P303 --> P305; P305 --> P304
```

Figure 8.3.2. 29 Deletion of R302 causing Sequential Process Flows

Sequential Paths P301→P303; P303→P305; P305 → P304

### 8.3.2.5 Enabling Parallel-OR Split Process Flow Patterns

#### Experiment 8

As demonstrated in section 6.4, the ECA Model algorithm provides the Parallel Split Flow Rule construct with an OR disjunction operator to enable business processes to form parallel split flow patterns. In this experiment, tests were carried out using business rules in Use case #8. The business rule R404 is linked to succeeding business rules R405 and R406. R404's action component invokes both business rules R405 and R406 via action-event components using the disjunction "OR". Parallel-OR Split is like Parallel-OR Split. However, it activates all outgoing business rule components simultaneously. Business rules R404 activates R405 and R406 simultaneously. Figure 8.3.2.13 presents the relationships between business rules.

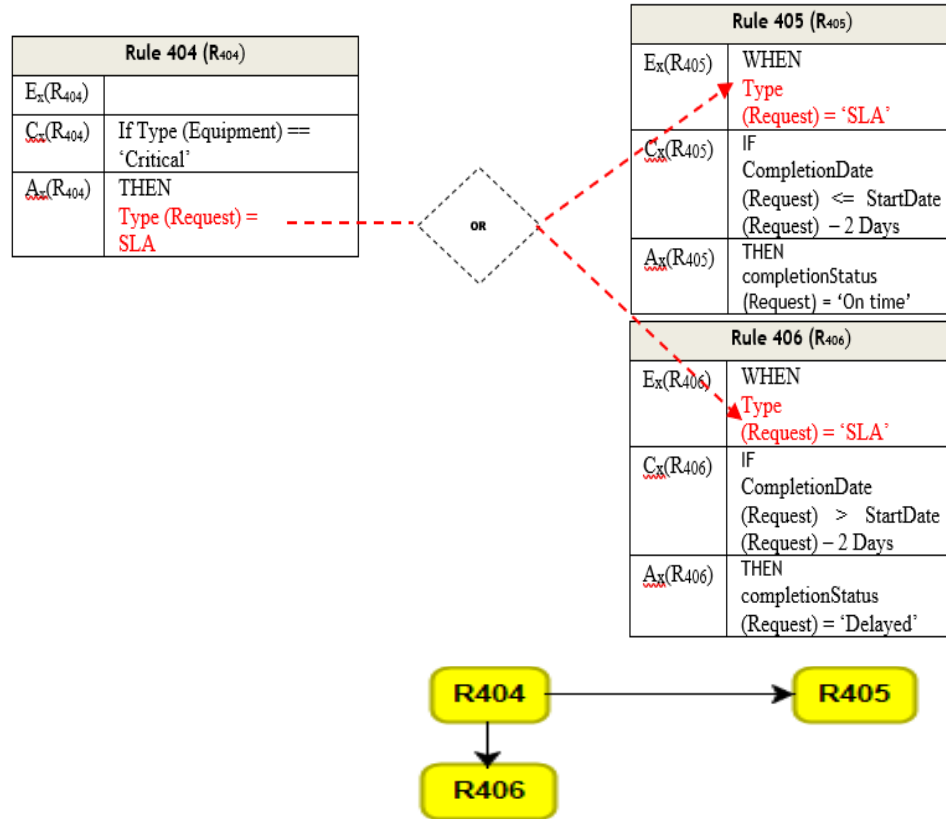


Figure 8.3.2. 30 Business Rules presenting Parallel-OR Split Relationships

When business rules R404, R405, R406 and their relationships are generated using the dependency graph with a mapping table shown earlier in section 6.5, the corresponding business processes and their links are formed. Figure 8.3.2.31 presents the business rules in DRL format and dependency graph.



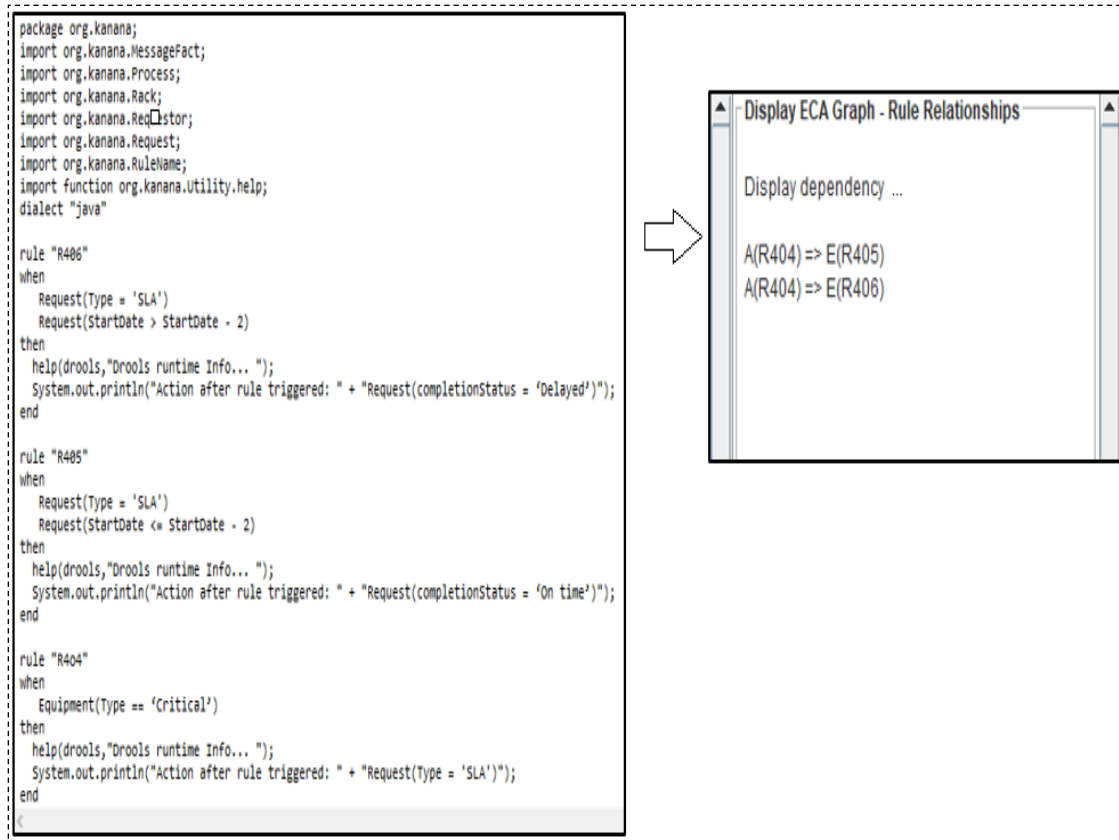


Figure 8.3.2. 31 R404, R405, R406 and Relationships in DLR format

By applying the business rules above, we can enable Parallel-OR Split flow patterns of a workflow. Two parallel split paths are generated, (from P404 to P405 OR from P404 to P406) as shown Figure 8.3.2.32.

```

Start the ECA Model...

Use the ECA Model Test Client to load your rules...
Display the workflow configuration ...

[main] INFO org.drools.compiler.kie.builder.impl.ClasspathKieProject - Found kmodule: file:/C:/Users/Kanana/workspace/AdaptiveECARuleModel/t
[main] INFO org.drools.compiler.kie.builder.impl.KieRepositoryImpl - KieModule was added: FileKieModule[releaseId=ECARulesByGrace.group:Adap

Display process in the workflow
P404
P405
P406

Display relationships between generated business processes (workflow)...
P404 --> P405
P404 --> P406

```

Figure 8.3.2. 32 Business Rules & Relationships enabling Parallel-OR Split Process Flows

Parallel-OR Split paths 1) P404 → P405 or 2) P404 → P406

## 8A) Insertion of business rule components

The adaptation algorithm facilitates the insertion of a new business rule to support the insertion of business processes in a parallel-or split flow situation. Consider a new business rule (R407), to be added to scenario in Use case#8. Business rule R407 is to be inserted between R404 and R406 to process the request completion date to be greater than the request start date based on the number of days the user specifies. So, R407 states that when the notified request is set to SLA and if the number of days within equipment SLA threshold  $> 2$ , then the set completion date is to be greater than the request start date. This forms a 'determine overdue SLA' process (P407). The new business rule R407 is inserted via the ECA Model Test Client then R407 and its relationships are generated using the dependency graph and a mapping table shown earlier in section 6.5. The new R407 node is evaluated to which all connected business rule nodes are accessed and updated accordingly. Then a new process node is created, and corresponding business process nodes and relationships are updated accordingly. Figure 8.3.2.33 presents the business rules in DRL format, R407 is the new inserted business rule. Business rules are transformed into the dependency graph.

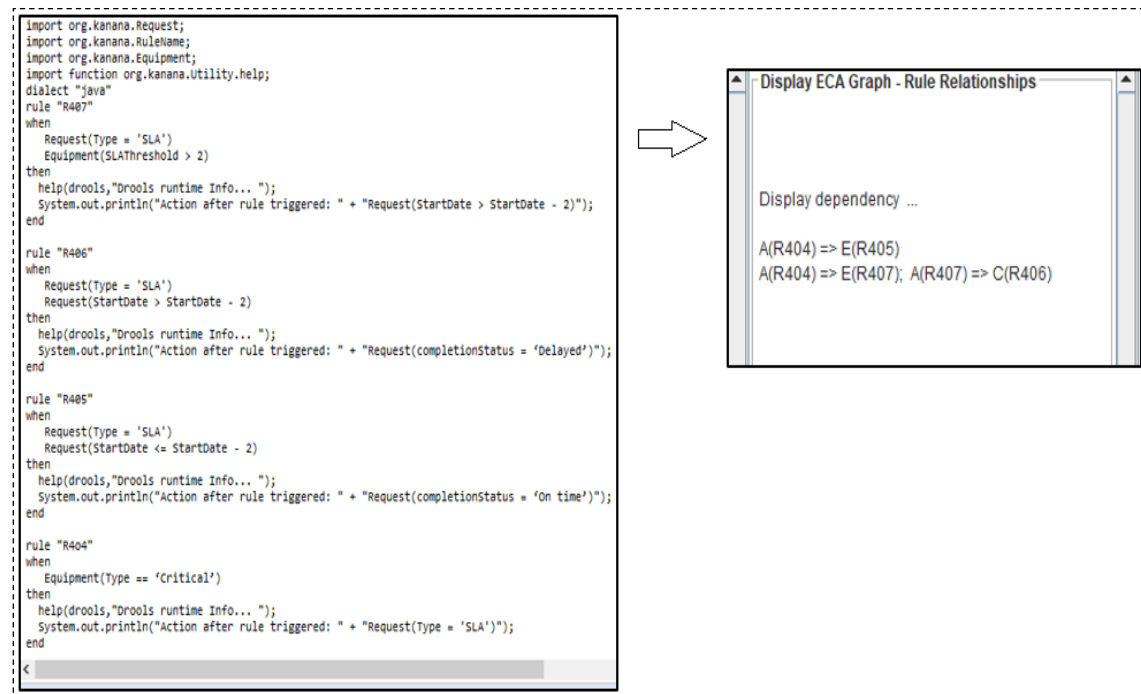


Figure 8.3.2. 33 Insert R407 and Relationships via ECA Model Test Client

By applying the business rules above, we can enable Parallel-OR flow patterns and insert a new process in a workflow. Two Parallel-OR Split paths are generated, (from P404 to P405 OR from P404 to P407 and from P407 to P406) as shown Figure 8.3.2.34.

```

Start the ECA Model...

Use the ECA Model Test Client to load your rules...
Display the workflow configuration ...

[main] INFO org.drools.compiler.kie.builder.impl.ClasspathKieProject - Found kmodule: file:/C:/Users/Kanana/workspace/AdaptiveECARuleModel/t
[main] INFO org.drools.compiler.kie.builder.impl.KieRepositoryImpl - KieModule was added: FileKieModule[releaseId=ECARulesByGrace.group:Adap

Display process in the workflow
P404
P405
P406
P407

Display relationships between generated business processes (workflow)...
P404 --> P405
P404 --> P407 --> P406

```

Figure 8.3.2. 34 Insertion of R407 causing Parallel-OR Process Flows

Parallel-OR Split paths 1) P404 → P405 OR 2) P404 → P407 → P406

8B) Modify business rule (changing properties of the of the business rule hence process)

The proposed adaptation algorithm facilitates the modification of existing business rules to support the modification of business process in a Parallel-OR Split flow situation. Consider the Use case #8 scenario, whereby business rule (R405) action component is modified to set equipment SLA threshold to less than 2 days. The business rule R405 is modified through the ECA Model Test Client and then its relationships are updated using the dependency graph and a mapping table shown earlier in section 6.5. The R405 node is evaluated, to which all connected business rule nodes are accessed and updated accordingly. Also, the corresponding business process node and its relationship are updated accordingly. Figure 8.3.2.35 presents the business rules in DRL format, R405 is updated. The transformed business rules and their relationships are shown on the dependency graph.

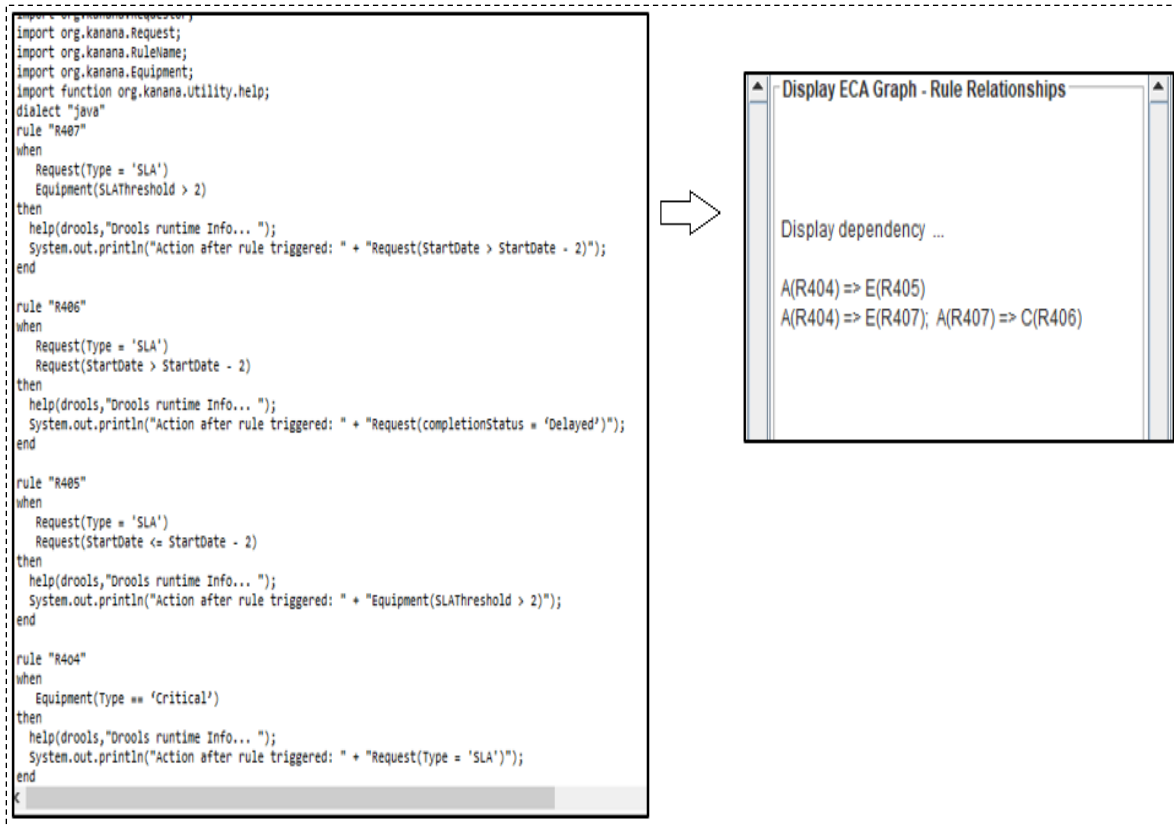


Figure 8.3.2. 35 Update R405 and Relationships via ECA Model Test Client

By applying the business rules above, we can enable Parallel-OR flow patterns and modify process in a workflow. Two Parallel-OR Split paths are generated (from P404 to P405 OR from P404 to P407 and from P407 to P406) as shown Figure 8.3.2.36.

```

Start the ECA Model...

Use the ECA Model Test Client to load your rules...
Display the workflow configuration ...

[main] INFO org.drools.compiler.kie.builder.impl.ClasspathKieProject - Found kmodule: file:/C:/Users/Kanana/workspace/AdaptiveECARuleModel/t
[main] INFO org.drools.compiler.kie.builder.impl.KieRepositoryImpl - KieModule was added: FileKieModule[releaseId=ECARulesByGrace.group:Adap

Display process in the workflow
P404
P405
P406
P407

Display relationships between generated business processes (workflow)...
P404 --> P405
P404 --> P407 --> P406

```

Figure 8.3.2. 36 Updating R404 causing Parallel-OR Split Process Flows

Parallel-OR Split paths 1) P404 → P405 OR 2) P404 → P407 → P406

### 8C) Deletion of existing business rules - disconnecting existing process and flows

The proposed adaptation algorithm facilitates the deletion of existing business rules to support the deletion of business process in a Parallel-OR Split flow situation. Consider the Use case #8 scenario, whereby the user would like to remove P404 from the workflow, which will involve deletion of R404. Business rule deletion is straight forward. Removing R404 will remove P404 as well as all destination connections. The business rule R404 is deleted through the ECA Model Test Client then its relationships are updated using the dependency graph and a mapping table shown earlier in section 6.5. The R404 node is evaluated to which all connected business rule nodes are removed accordingly. Also, the corresponding business process node and its relationship are updated accordingly. Figure 8.3.2.20 presents the business rules in DRL format, R404 is deleted. The transformed business rules and relationships are shown on the dependency graph Figure 8.3.2.37

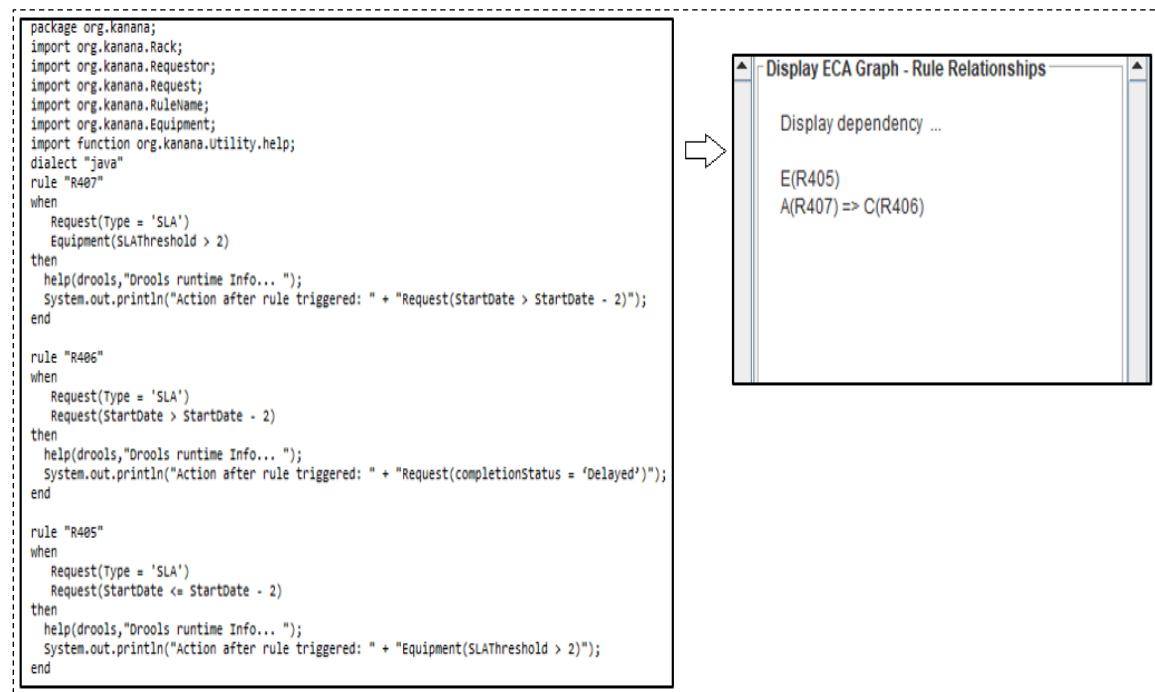


Figure 8.3.2. 37 Delete R404 and Relationships via ECA Model Test Client

After deleting R404, business process P404 is removed and P405 is no longer connected. P407 is connected P406 as shown in Figure 8.3.2.38. Notice P404 and its dependencies have been removed.

```
Start the ECA Model...

Use the ECA Model Test Client to load your rules...
Display the workflow configuration ...

[main] INFO org.drools.compiler.kie.builder.impl.ClasspathKieProject - Found kmodule: file:/C:/Users/Kanana/workspace/AdaptiveECA/
[main] INFO org.drools.compiler.kie.builder.impl.KieRepositoryImpl - KieModule was added: FileKieModule[releaseId=ECARulesByGrace]

Display process in the workflow
P405
P406
P407

Display relationships between generated business processes (workflow)...
P405
P407 --> P406
< >
```

Figure 8.3.2. 38 Deletion of R404 causing removal of P404 and Connections  
Result Paths P405; P407 → P406

### 8.3.2.6 Enabling Parallel-AND Split Process Flow Patterns

#### Experiment 9

As demonstrated in section 6.4, the ECA Model algorithm provides the Parallel Split Flow Rule construct with an AND conjunction operator to enable business processes to form parallel split flow patterns. In this experiment, tests were carried out using business rules in Use case #9. The business rule R501 is linked to succeeding business rules R502 and R503. R501's action component invokes both business rules R502 and R502 via action-event components using the conjunction "AND". Parallel-AND Split is like Parallel-AND Split. However, it activates all outgoing business rule components simultaneously. Business rule R501 activates R502 and R503 simultaneously. Figure 8.3.2.39 presents the relationships between business rule R501 and (R502 and R503, R504).

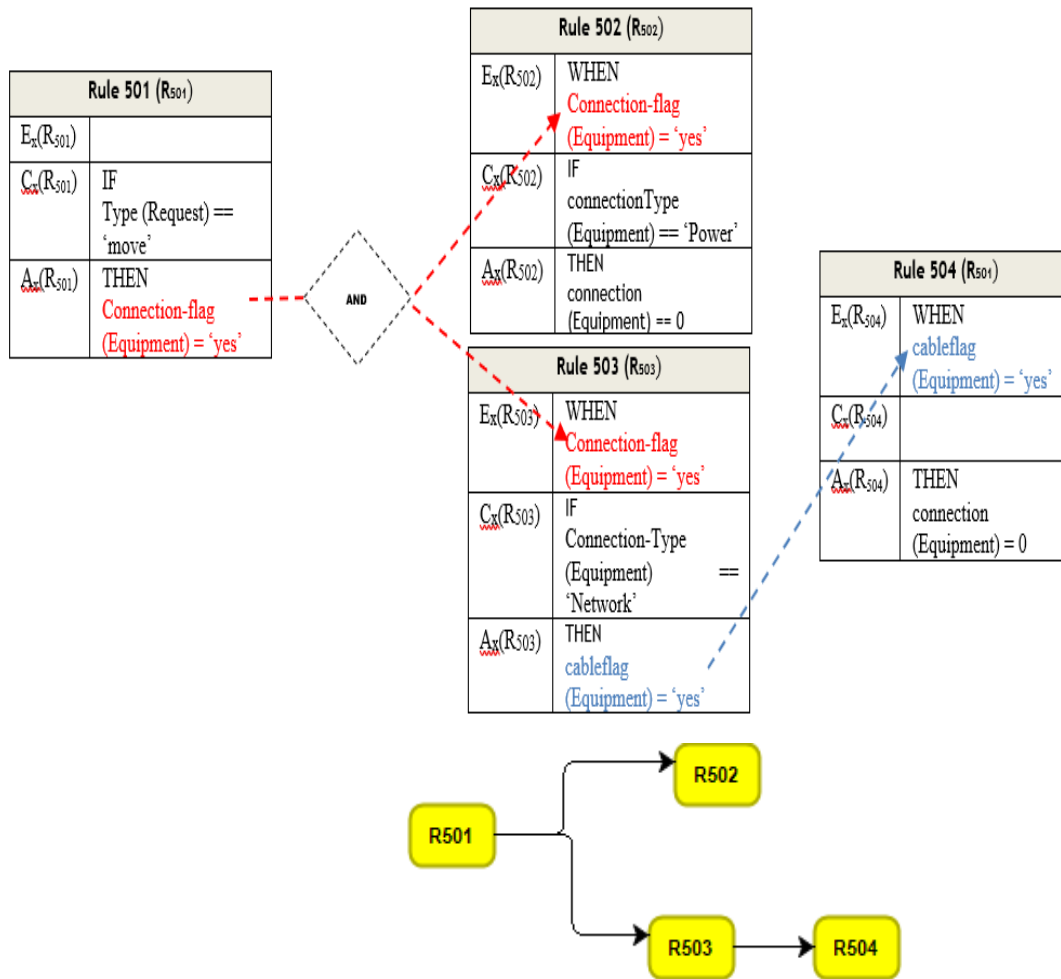


Figure 8.3.2. 39 Business Rules presenting Parallel-OR Split Relationships

When business rules R501, R502, R503, R504 and their relationships are generated using the dependency graph with a mapping table shown earlier in section 6.5, the corresponding business processes and their links are formed. Figure 8.3.2.40 presents the business rules in DRL format and dependency graph.

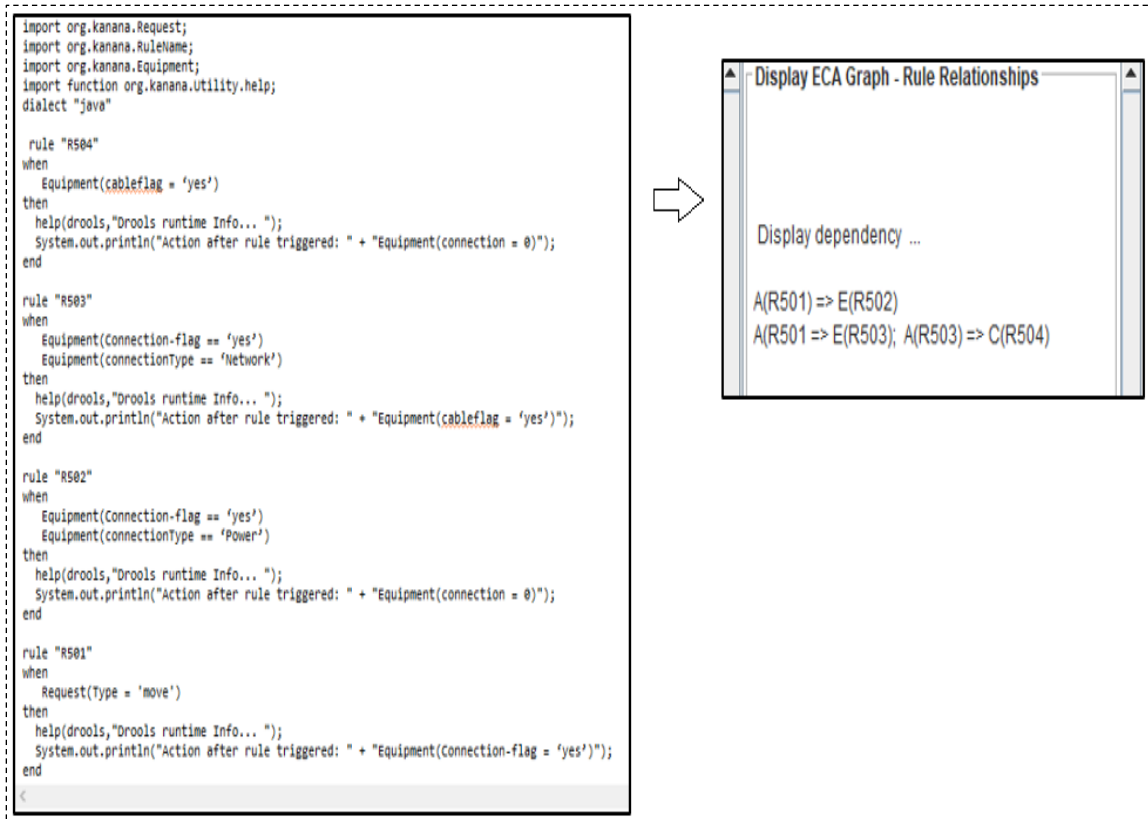


Figure 8.3.2. 40 R501, R502, R503, R504 and Relationships in DRL Format

By applying the business rules above, we can enable parallel-AND Split flow patterns of a workflow. Two parallel split paths are generated, (from P501 to P502 AND from P501 to P503 and from P503 to P504) as shown Figure 8.3.2.41.

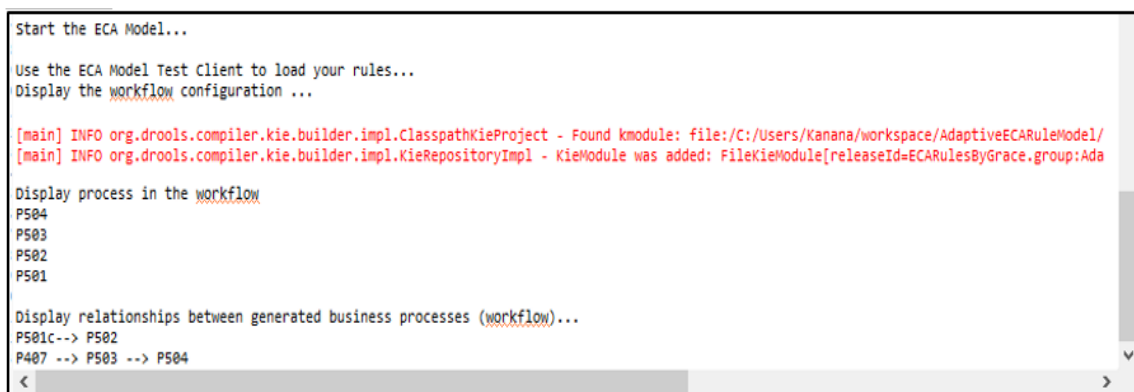


Figure 8.3.2. 41 Business Rules & Relationships enabling Parallel-AND Split Process Flows

Parallel-AND Split paths 1) P501 → P502 or 2) P501 → P503→ P504



### 9A) Insertion of business rule components

The adaptation algorithm facilitates the insertion of a new business rules to support the insertion of business processes in a parallel-or split flow situation. Consider a new business rule (R505) is added to the scenario in Use case #9. Business rule R505 is to be inserted between R501 and R502 to set connection type on equipment to power for equipment with power ports (no of power supplies). So, business rule R505 states that, when notified, connection flag is set to yes and if power ports are greater than zero, then set connection type on equipment to power. This forms a ‘check power supplies’ process (P505). The new business rule R505 is inserted via the ECA Model Test Client then R505 and its relationships are generated using the dependency graph and a mapping table shown earlier in section 6.5. The new R505 node is evaluated to which all connected business rule nodes are accessed and updated accordingly. Then a new process node is created, and corresponding business process nodes and relationships are updated accordingly. Figure 8.3.2.42 presents the business rules in DRL format, R505 is the new inserted business rule. Business rules are transformed into the dependency graph.

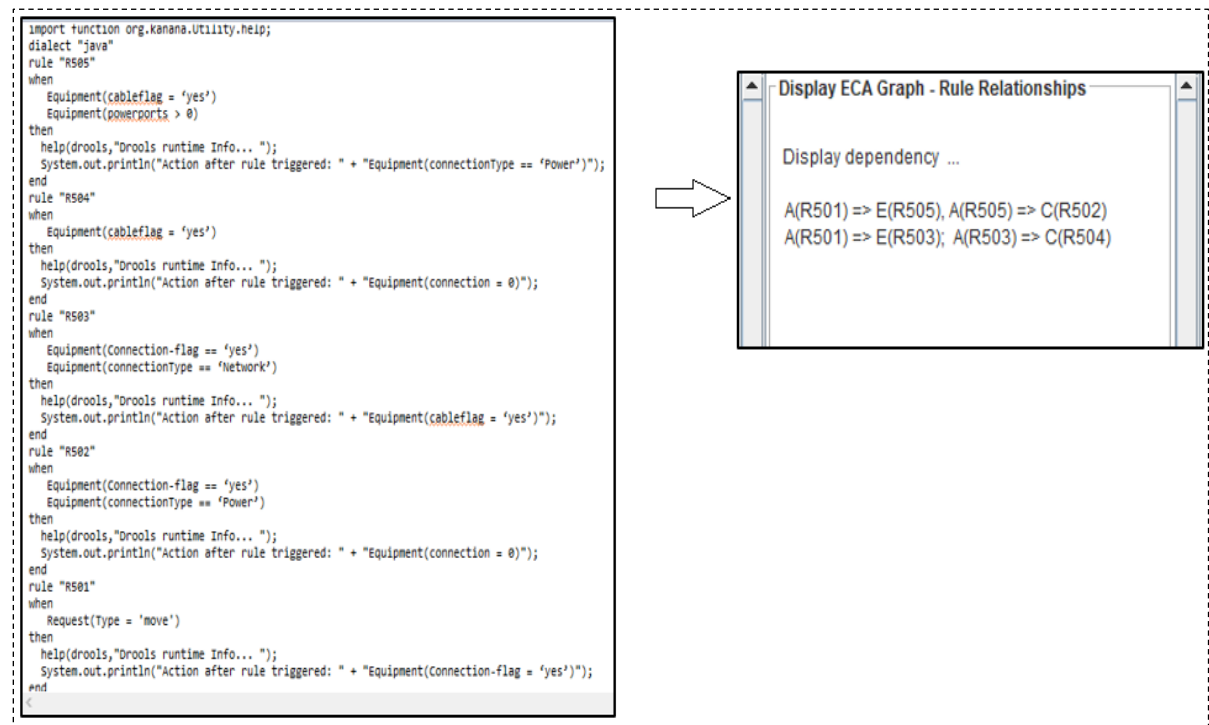


Figure 8.3.2. 42 Insert R305 and Relationships via ECA Model Test Client

By applying the business rules above, we can enable Parallel-AND Split flow patterns and insert a new process in a workflow. Two Parallel-AND Split paths are generated, (from P501 to P505, from P505 to P502 AND from P501 to P503 and from P503 to P504) as shown Figure 8.3.2.43: Parallel-AND Split paths 1) P501 → P505 → P502 AND 2) P501 → P503 → P504

9B) Modify business rule (changing properties of the of the business rule hence process)  
The proposed adaptation algorithm facilitates the modification of existing business rules to support the modification of business process in a Parallel-AND Split flow situation. Consider the Use case#9 scenario, whereby business rule (R503) event component is modified by renaming the object Equipment to Rack. The business rule R503 is modified through the ECA Model Test Client and then its relationships are updated using the dependency graph and a mapping table shown earlier in section 6.5. The R503 node is evaluated, to which all connected business rule nodes are accessed and updated accordingly. Also, the corresponding business process node and its relationships are updated accordingly. Figure 8.3.2.43 presents the business rules in DRL format, R503 is updated. The transformed business rules and their relationships are shown on the dependency graph.

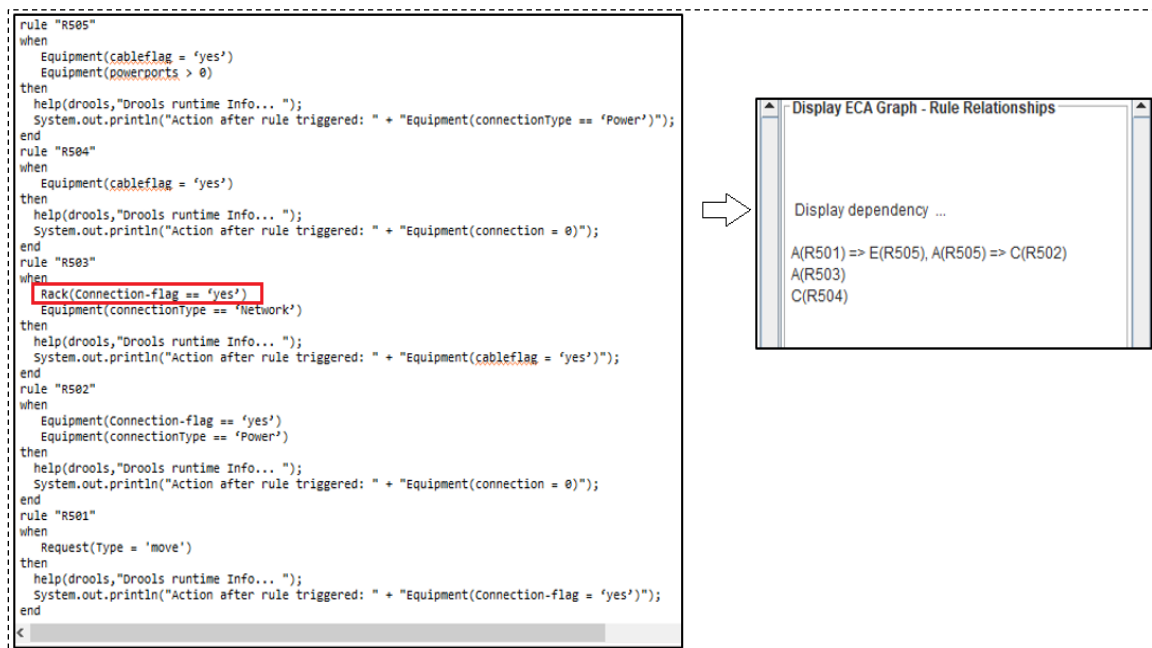


Figure 8.3.2. 43 Update R503 and Relationships via ECA Model Test Client

By applying the business rules above, we can enable Parallel-AND flow patterns and modify a process in a workflow. Two Parallel-AND Split paths are generated (from P501 to P505, P505 to P502, P503 AND P504) as shown Figure 8.3.2.44.

```

Start the ECA Model...

Use the ECA Model Test Client to load your rules...
Display the workflow configuration ...

[main] INFO org.drools.compiler.kie.builder.impl.ClasspathKieProject - Found kmodule: file:/C:/Users/Kanana/workspace/AdaptiveECARuleModel/1
[main] INFO org.drools.compiler.kie.builder.impl.KieRepositoryImpl - KieModule was added: FileKieModule[releaseId=ECARulesByGrace.group:Ada

Display process in the workflow
P505
P504
P503
P502
P501

Display relationships between generated business processes (workflow)...
P501 --> P505 --> P502
P503
P504

```

Figure 8.3.2. 44 Modification of R503 causing Parallel-AND Split Process Flows

Parallel-AND Split paths 1) P501 → P505 → P502 AND 2) P504

#### 9C) Deletion of existing business rules - disconnecting existing process and flows

The proposed adaptation algorithm facilitates the deletion of existing business rules to support the deletion of business process in a Parallel-AND Split flow situation. Consider the Use case #9 scenario, whereby the user would like to remove P502 from the workflow, which will involve deletion of R502. Business rule deletion is straight forward. Removing R502 will remove P502 as well as all source connections. The business rule R502 is deleted through the ECA Model Test Client and then its relationships are updated using the dependency graph and a mapping table shown earlier in section 6.5. The R502 node is evaluated to which all connected business rule nodes are removed accordingly. Also, the corresponding business process node and its relationship are updated accordingly. Figure 8.3.2.45 presents the business rules in DRL format, R502 is deleted. The transformed business rules and their relationships are shown on the dependency graph.

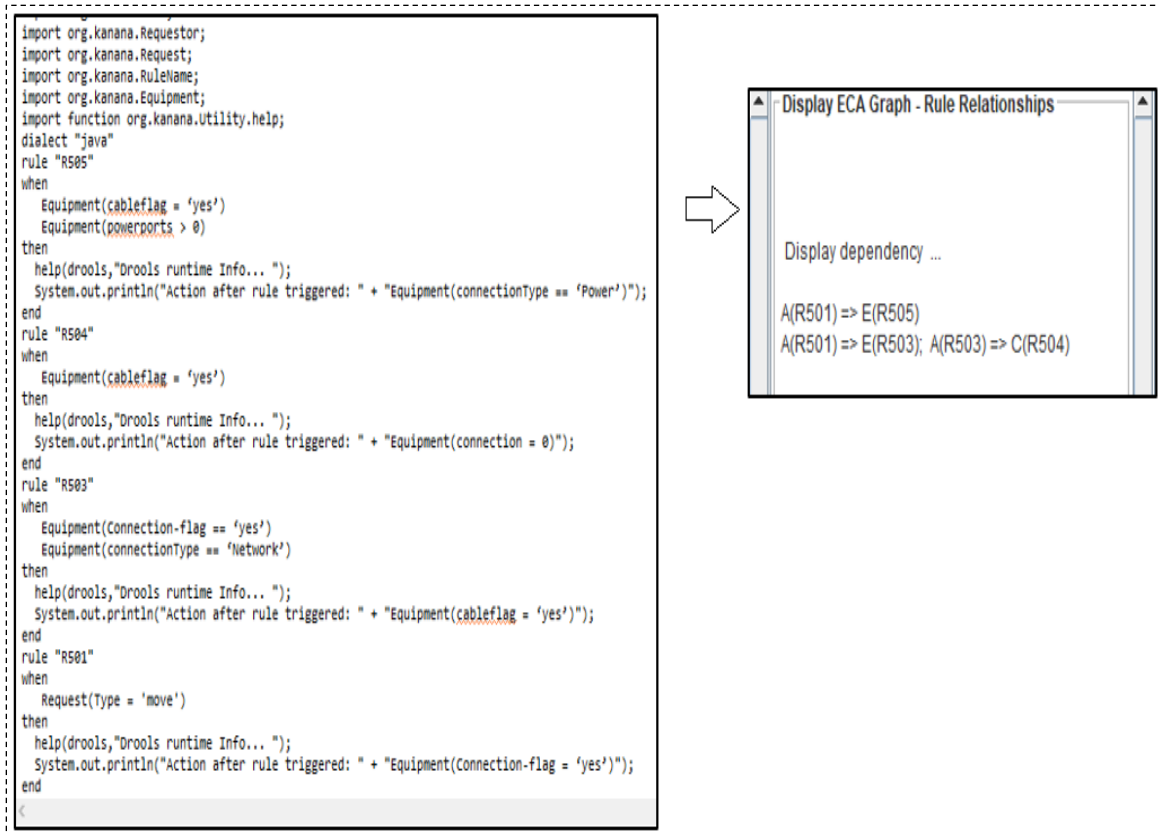


Figure 8.3.2. 45 Delete R502 and Relationships via ECA Model Test Client

After deleting R502, business process P502 is removed and P502 is no longer connected. P501 is connected P505 only as shown in Figure 8.3.2.46. Notice P502 and its dependencies have been removed.



Figure 8.3.2. 46 Deletion of R502 causing removal of P502 and connected paths

## **8.4 Summary**

The Chapter contributes to the area of research experimental. It introduced various scenarios (use cases) from data centre workflows have been presented to validate the proposed model prototype developed and presented in Chapter 7. The complexity of changing the structure of business rules and components, difficulty in propagating changes on related business rule components and workflow configurations are typical challenges facing workflow users. In this chapter, using research objectives identified in section 1.3, we have been able to demonstrate not only the ability to manage business rules and changes at component level but also to allow business rules to be used to govern processes of a workflow. There is a significant advantage for workflow users when adapting business rules to manage and control the flow of processes in real time. More general benefits for rule systems arise from being able to manage changes at business rule component level. Table 8.4.1 summarizes validation results.

| Research objectives                                                    | Scenarios                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | Validation Criteria                                                                                               | Actual Results                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | Results Analysis/Comments                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Creation of business rules & components<br>- <b>Objective 5(a)</b>     | <p>1- Model's ability to add entire business rules, which consists of three components (event, condition and action). As an example, Business rule R1 from Use case #1 is entered via ECA Model Test Client.</p> <p>2- Model's ability to add separate business rule components. As an example, Business rule R5 from Use case #1 is entered via ECA Model Test Client but this time only event and action components of the business rule are added.</p> <p>3- Model's ability to add separate business rule components. As an example, Business rule R6 from Use case #1 is entered via ECA Model Test Client but this time only condition and action components of the business rule are added.</p> | <p>(Ability to add business rules &amp; components on the fly)</p> <p>- Adaptation, Flexibility, Usability</p>    | <p>1- Business rule R1 and all components are added to Drools DRL as expected. Figure 9.3.1.1 captures the results of adding R1.</p> <p>2- Business rule R5 event and action added to Drools DRL via ECA Model Test Client. Figure 9.3.1.2 captures the results of adding R5's event and action components</p> <p>3- Business rule R6 condition and action components are added to Drools DRL Figure 9.3.1.3 captures the results of adding R6's condition and action components</p> | Using the proposed model, we can model business rules at components level. Users are free to enter any part/component of a business rule, any combination can be specified. The ECA Model satisfies the adaptability and flexibility of adding business rules and components. Furthermore, with a guided user interface (ECA Model Test Client), a non-technical user can add any number of business rules, which will then be converted into Drool rule language. |
| Modification of business rules & components<br>- <b>Objective 5(a)</b> | <p>1- Model's ability to modify entire business rules, which consists of three components (event, condition and action). As an example, Business rule R5 from Use case #2 is entered via ECA Model Test Client.</p> <p>2- Model's ability to modify separate business rule components. As an example, Business rule R5 from Use case #2 is entered via ECA Model Test Client but this time only action component of the business rule is modified.</p>                                                                                                                                                                                                                                                 | <p>(Ability to modify business rules &amp; components on the fly)</p> <p>- Adaptation, Flexibility, Usability</p> | <p>1- Business rule R5 with components are modified in Drools DRL Figure 9.3.1.5 captures results of modifying R5 and components</p> <p>2- Business rule R5 action is modified in Drools DRL. Figure 9.3.1.6 captures results of updating R5's action components</p>                                                                                                                                                                                                                 | Using the proposed model, we can modify business rules at components level. Users are free to modify any whole or part/component of a business rule, any combination can be specified. The ECA Model satisfies the adaptability and flexibility of modifying business rules and components. As mentioned before, with a guided user interface (ECA Model Test Client), a non-technical user can modify any number of business rules.                               |
| Deletion of business rules & components<br>- <b>Objective 5(a)</b>     | Model's ability to delete business rules and components. As an example, Business rule R5 from Use case #3 is entered via ECA Model Test Client for deletion.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | Ability to delete business rules & components on the fly                                                          | Business rule R5 is removed from Drools DRL file. Figure 9.3.1.7 captures results of removing R5 and components                                                                                                                                                                                                                                                                                                                                                                      | Business rules still are expressed in forms of simple statements. This is valuable, especially to avoid inconsistent syntax. As discussed above in ECA Model, business rules statements are formalized into business rules components (event, condition and action). They formatted and expressed in a simple way, easy to identify what part is event, condition or action for implementing them in the business rules management systems.                        |
| Automatic generation of                                                | Applying sample data from Use case #1 to demonstrate model's ability to generate business                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | Check change propagation                                                                                          | Figure 9.3.1.4 displays the result of adding business rules (R13) and components as well as                                                                                                                                                                                                                                                                                                                                                                                          | By using the dependency graphs to define new dependencies and regenerating existing relations of A(R13), the algorithm provides the ability to insert new                                                                                                                                                                                                                                                                                                          |

|                                                                                                                                                                                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |                                                                                                                                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>business rules components dependencies - <b>Objective 5(b)</b></p> <p>Change propagation during insertion, modification and deletion of business rule components - <b>Objective 5(c)</b></p> | <p>rules relationships and provide support for change propagation. In particularly looking at a scenario when new business rule (R13) and components are added.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | <p>(Accuracy, Usability and Simplicity): The interaction in particularly with chained business rules</p>                                 | <p>various dependency graphs to show which business rules will be affected by changing R13. At component level, business rules are linked or connected to each other, i.e. R13's action connects to R2's event. The dependency/change propagation graph is displayed in the "Display ECA Graph - Rule relationships" section of the ECA Model Test Client. R13 is linked to five business rules components. Business rules R2 has a direct dependency on R13's action component. This leads to indirect relation to R3 and R4 event components, R3's action component connects to R5's event component and R6's condition component. R13's change propagation to R2, R3, R4, R5 and R6 needs to be revised in order to guarantee the activation of all the rules</p> | <p>business rules at component level A(R13) and propagate changes by revising all related business rule components as seen in Figure 9.3.1.11. We also look at the change cost in order to measure performance or efforts needed to apply or modify the business rule change. For example, if business rule R13 is added and business rule R2 is changed in the previous example, so the effective change effort applicable to business rule R13 concerns the efforts to change business rules R3, R4, R5 and R6 plus the efforts to change business rule R2. It is important to estimate the maximum change cost before making any changes. This will help to determine and plan the change in advance hence giving a tangible estimation of the efforts needed to implement business rule changes. In our model, the cost of changes is based upon the business rule change dependency patterns in a graph. The arcs in a graph patterns are used as inputs to access the change. For example, the neighbour dependency's pattern will help to determine the effort required to change successors or predecessors of a give business rule component. The Level dependency pattern allows to determine the distance between business rule components.</p> |
| <p>Adaptation of business rules in a workflow to control creation/termination processes - <b>Objective 5(d)</b></p>                                                                             | <p>A typical data centre equipment move workflow is used to demonstrate how business rules can be used to create initiating and terminating processes. Consider the following scenario from Use case #4. When moving equipment from one data centre location to another, a requestor fills out a move form (request) to include equipment to be moved, current and new location, new power requirements, etc. Business rules exist to ensure power connected equipment are not moved around. The first business rule (R101) states that when request type is move then then set equipment power connections greater than zero. The second business rule (R102) states that if equipment power connections is greater than zero then request status is set to close. The third business rule (R103) states that if equipment power connection is less than zero then request status is set to power-provision and finally the fourth business rule (R104) states that if request status is set to power-provision then request status is set to close</p> | <p>Adaptability of ECA rules to control initiation of processes</p> <p>Adaptability of ECA rules to control termination of processes</p> | <p>When business rule R101, R102, R04 and their relationships were generated, R101 evaluated to a corresponding starting business process. Likewise, R102 and R104 evaluated to corresponding business processes and their links are formed. Figure 9.3.2.2 presents the business rules in DRL format and dependency graph. Workflow's start and end processes are enabled as shown Figure 9.3.2.3. As you can see, the same start and end processes as in the original workflow (Figure.9.2.2.3).</p>                                                                                                                                                                                                                                                               | <p>If we analyse the dependency graph of our business rules (Figure 9.3.2.2), we notice something interesting, the root node (R01) enables the "start process" (P101) and the leaf business rule nodes (R102 and R104) enable the terminating processes (P102 and P04). By identifying the root and leaf business rules, we can determine and enable the initiating and terminating processes.</p> <p>The ECA Model prototype offers the ability to auto generate the initiating and terminating business processes by using defined business rules. For the users, the adaptation of business rules to transform "start" and "end" business processes is literally a matter of entering all business rules via the ECA Model Test Client.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |

|                                                                                                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                                                                                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|--------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>Adaptation of business rules to enable sequential processes in a workflow. - <b>Objective (5d)</b></p>    | <p>1- Data centre equipment decommission workflow from Use case #5 is used to demonstrate how business rules can be used to enable sequential process flow patterns. The business processes and rules (R201, R202 and R203) are summarised in Figure.9.2.2.4</p> <p><b>Insertion of business rule components (new process and/or new relationships)</b></p> <p>2- Consider, a new business rule (R204) that is to be added to scenario in Use case #5. R204 is to ensure that when a request is set to decom-approve then power connections are disconnected from the equipment to be removed.</p> <p><b>Modify business rule (changing source or target process flows)</b></p> <p>3- Consider the Use case #5 scenario, whereby the user discovered that the business rule (R204) that has just been inserted to create a P204 process was wrongly positioned. The workflow was supposed to flow from P201 → P202 → P204 → P203 and not P201 → P204 → P202 → P203. So, they would like to be able to update the business rule R204</p> <p><b>Deletion of existing business rules (disconnecting existing process flows)</b></p> <p>4- Consider the Use case #5 scenario, whereby the user would like to remove P204 from the workflow, which will involve deletion of R204.</p> | <p>Adaptability of ECA rules to control the running of sequential processes</p>     | <p>1- By applying the business rules R201, R202 and R203 and their relationships, Processes P201, P203 and P03 are generated with sequential process flow from P201 to P202 and from P202 to P203 as shown Figure 9.3.2.6.</p> <p>2- By inserting the new business rule (R204), P204 process is generated with sequential process flow from P201 to P204, from P04 to P202 and from P202 to P203 as shown Figure 9.3.2.8</p> <p>3- After applying the changes to R204, business processes are connected sequentially from P201 to P202, from P202 to P204 and from P204 to P203 as shown in Figure 9.3.2.10.</p> <p>4- After deleting R204, business processes are connected sequentially from P201 to P202, from P202 to P204 and from P204 to P203 as shown in Figure 9.3.2.12.</p> | <p>The ECA Model prototype provides capability to enable sequential process flow by using defined business rules. For the users, the adaptation of business rules to enable sequential flow of business processes is literally a matter of entering all business rules via the ECA Model Test Client.</p> <p>The ECA Model prototype facilitates the insertion of a new business rules to support the insertion of business processes in a sequential flow situation.</p> <p>The ECA Model prototype facilitates the modification of existing business rules to support the modification of business process in a sequential flow situation</p> <p>The ECA Model prototype facilitates the deletion of existing business rules to support the deletion of business processes in a sequential flow situation</p> |
| <p>Adaptation of business rules to enable AND/OR Merged processes in a workflow. - <b>Objective (5d)</b></p> | <p>1- Consider the following decommission workflow (Figure.9.2.2.5) where business rules (R301, R302, R303 and R304) from Use case #6 have been added to ensure equipment is first disconnected by power or network provisioner before final decommission process is executed.</p> <p><b>Insertion of business rule components</b></p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | <p>Adaptability of ECA rules to control the running of parallel merge processes</p> | <p>1- By applying the business rules R301, R302, R303 and R304, two Parallel-OR Merge paths are generated, (from P301 to P302 and from P302 to P304 OR from P301 to P303 and from P303 to P304) as shown Figure 9.3.2.15.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | <p>The ECA Model prototype provides the Parallel Merge Flow Rule construct with an OR disjunction operator to enable workflow processes to form Parallel-OR Merge flow patterns.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |



|                                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |                                                                                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|-----------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                                                                                             | <p>2- Consider, a new business rule (R305) that is to be added to scenario in Use case#6. R305 is to schedule the equipment for decommission when a request is set to decom-approve then the equipment status is set to removed. This forms a scheduling decommission process (P305) that needs to be executed before the Request Close process (P304).</p> <p><b>Modify business rule<br/>(changing properties of the of the business rule hence process)</b></p> <p>3- Consider the Use case #6 scenario, whereby business rule (R303) is modified because P303 process is to consider not just network provisioner but also storage provisioner.</p> <p><b>Deletion of existing business rules<br/>(disconnecting existing process and flows)</b></p> <p>4- Consider the Use case #5 scenario, whereby the user would like to remove P302 from the workflow, which will involve deletion of R302.</p> |                                                                                     | <p>2- By inserting new business rule (R305), two Parallel-OR Merged paths are generated, (from P301 to P302, from P302 to P305 and from P305 to P304 OR from P301 to P303, from P303 to P305 and from P305 to P304) as shown Figure 9.3.2.17.</p> <p>3- By modifying business rules (R303), two parallel merged-or paths are generated, (from P301 to P302, from P302 to P305 and from P305 to P304 OR from P301 to P303, from P303 to P305 and from P305 to P304) as shown Figure 9.3.2.19.</p> <p>4- After deleting R302, business processes are connected sequentially from P301 to P303, from P303 to P305 and from P305 to P304 as shown in Figure 9.3.2.21. Notice P302 and its dependencies have been removed.</p> | <p>The ECA Model prototype facilitates the insertion of a new business rules to support the insertion of business processes in a parallel-or merged flow situation. The R305 node is evaluated to which all connected business rule nodes are accessed and updated accordingly. Then the new process is created, corresponding business process nodes and relationships are updated accordingly</p> <p>The ECA Model prototype facilitates the modification of existing business rules to support the modification of business process in a parallel-or merged flow situation. The R303 node is evaluated to which all connected business rule nodes are accessed and updated accordingly. Then, the corresponding business process node and its relationship are updated accordingly.</p> <p>The ECA Model prototype facilitates the deletion of existing business rules to support the deletion of business process in a parallel-or merged flow situation. Removing R302, removes P302 as well as all connections from source to destination</p> |
| <p>Adaptation of business rules to enable AND/OR Parallel Split processes in a workflow.</p> <p><b>- Objective (5d)</b></p> | <p>1- Model's ability to enable Parallel-AND Split flow patterns. Consider the following scenario from data centre move workflow (Figure.9.2.2.10). When moving equipment from one location to the other, both power and network connections must be disconnected to the equipment. Business rules R501, R502, R503, R504 and their relationships are maintained</p> <p><b>Insertion of business rule components</b></p> <p>2- Consider a new business rule (R505) is added to scenario in Use case #9. Business rule R505 is to be inserted between R501 and R502 to set connection type on equipment to power for equipment with power ports (no of power supplies). So, business rule R505 states that when</p>                                                                                                                                                                                       | <p>Adaptability of ECA rules to control the running of parallel split processes</p> | <p>1- By applying the business rules (R501, R502, R503, R504 and their relationships), two parallel split paths are generated, (from P501 to P502 AND from P501 to P503 and from P503 to P504) as shown Figure 9.3.2.41.</p> <p>2- By inserting the business rules (R505), two Parallel-AND Split paths are generated, (from P501 to P505, from P505 to P502 AND from P501 to P503 and from P503 to P504) as shown Figure 9.3.2.43.</p>                                                                                                                                                                                                                                                                                   | <p>The ECA Model prototype provides the Parallel Split Flow Rule construct with an AND conjunction operator to enable business processes to form parallel split flow patterns</p> <p>The ECA Model prototype facilitates the insertion of a new business rules to support the insertion of business processes in a parallel-or split flow situation. The new R505 node is evaluated to which all connected business rule nodes are accessed and updated accordingly. Then new process node is created, corresponding business process nodes and relationships are updated accordingly.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                          |

|  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|--|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  | <p>notified connection flag is set to yes and if power ports greater than zero then set connection type on equipment to power.</p> <p><b>Modify business rule (changing properties of business rule hence process)</b></p> <p>3- Consider the Use case #9 scenario, whereby business rule (R503) event component is modified by renaming the object Equipment to Rack.</p> <p><b>Deletion of existing business rules (disconnecting existing process and flows)</b></p> <p>4- Consider the Use case #9 scenario, whereby the user would like to remove P502 from the workflow, which will involve deletion of R502</p> |  | <p>Figure 9.3.2.43: Parallel-AND Split paths 1) P501 → P505 → P502 AND 2) P501 → P503 → P504</p> <p>3- By modifying business rules (R503), two Parallel-AND Split paths are generated (from P501 to P505, P505 to P502, P503 AND P504) as shown Figure 9.3.2.44.</p> <p>4- After deleting R502, business process P502 is removed and P502 is no longer connected. P501 is connected P505 only as shown in Figure 9.3.2.46. Notice P502 and its dependencies have been removed.</p> | <p>The ECA Model prototype facilitates the modification of existing business rules to support the modification of business process in a Parallel-AND Split flow situation.</p> <p>The ECA Model prototype facilitates the deletion of existing business rules to support the deletion of business process in a Parallel-AND Split flow situation. Business rule deletion is straight forward. Removing R502 will remove P502 as well as all source connections.</p> |
|--|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

Table 8.4. 1 Validation Results

## 9. Conclusion and Future Research

This research seeks to advance the development and use of workflows by introducing a two-level model for business rules to govern processes in workflows, which is based on a strict logical formalization of the business ontology. Using a set of descriptive primitives with strict logical semantics, the framework provides the basis for formal definitions of the structure of business workflows and the policies which control their execution. The approach adopted was to design and implement the framework by prototyping to provide visible evidence on the feasibility of the framework. In addition, this research allows the implementation of business rules indexing, change propagation and rule adaptation approaches to enhance the framework. This Chapter concludes the thesis with a closing remark on the problem statement (reflecting on research questions and objectives) as well as looking at the effectiveness of the proposed solution (contributions). Furthermore, recommendations and future research areas that can improve the business rules adaptation in workflows are discussed as well.

### 9.1 Reflection on Research Questions

The main research questions were presented in section 1.4. Next, Table 9.1.1 below provides an overview of how the research questions were addressed.

| Research questions                                                                                                                                                          | Research comments                                                                                                                                                                                                                                                                                                                          |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| What factors limit the adaptation of the business rules in workflows?                                                                                                       | This question was addressed with the research study conducted in chapter 2. The study concluded with a brief section describing the gaps and limitation of existing studies and applications. One of the difficulties being the lack of a consistent model to manage business rules at components level. For more information section 2.3. |
| How to develop an ontology of the business workflows, which allows to formalize the business rules using templates so that dependencies between the rules can be described. | Chapter 3 provides the foundation concepts and structures, which include Event, Condition, Action, Process, etc.<br><br>Chapter 4 provides a conceptual framework of a two-level model for business rules to govern processes in workflows, which is based on a strict logical formalization of the                                        |

|                                                                                                                                                                                                                                                                                                                                                                                                                                   |                                                                                                                                                                                                                                                                                                                          |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                                                                                                                                                                                                                                                                                                                                                                                                   | business ontology. Furthermore, section 4.4 addresses the question of managing complex business rule relationships by introducing the AND-OR graphs.                                                                                                                                                                     |
| How can we specify the dependencies between the rules on the base of the ontology model so that the rules can be adapted to the changing conditions in real-time and propagate the necessary changes? In more precise, is it possible to create an efficient algorithm for change propagation, which enable the run-time adaptation of the business workflows?<br>How efficiently the underlying business rules can be retrieved? | The dependency graph patterns introduced in Chapter 4 as well as mixed of algorithms (indexing and change propagation) covered in Chapters 5 and 6 provide the technique for computing business rules change propagation.                                                                                                |
| How can we optimise business rules to improve execution performance and provide runtime modification?                                                                                                                                                                                                                                                                                                                             | Chapter 5 provides the concept of Metarules to support runtime modification of business rules at runtime.                                                                                                                                                                                                                |
| How we can use the business rule dependencies to construct an efficient mechanism for adapting the rules in the case of changes?<br>How can we enable adaptation of the business rules in real-time with reasonable complexity?                                                                                                                                                                                                   | Chapter 4 provides the foundation framework of two-level model for business rules to govern processes in workflows and Chapter 6 presents the technique/algorithm implemented to support adaptation of business rules in workflow.                                                                                       |
| Can a proposed model structure be able to generalise to new business rules in a workflow not seen during prototype validation?                                                                                                                                                                                                                                                                                                    | Various implementation techniques have been applied for the development of the prototype. Specifically, chapter 8 covers the object-oriented implementation of business rule component classes including business rule template. The implementation provides generic reusable and adaptable objects that can be changed. |

Table 9.1. 1 Reflection of Research Questions

Overall, the successful development of the prototype proves the research hypothesis. The related theories of a two-level model for business rules, strict logical formalization of the business rules ontology using AND-OR dependency graphs, business rule adaptation and change propagation put forward in the theoretical part are feasible and were validated in Chapter 8. The prototype shows that the workflows from data centres can be efficiently implemented using the proposed model. Business workflows such as equipment install, equipment decommissions, equipment move can be managed through business rules to support the complex change and configuration problems.

## 9.2 Reflection on Aim and Objectives

Overall, the aim and objectives of this research study were met (Table 9.2. 1). The prototype developed by considering all important knowledge gained from the literature review to the development of the formal model, the rule indexing, rule adaptation and change propagation algorithms.

| Aim and objectives                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | Status | Description                                                                                                                                                                                                                                                                                                                                                          |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1. Study existing research works through literature review in the area of business rules and workflows.<br>a. Information gathering by identifying relevant published research papers, journals, articles, posters, etc.<br>b. Reviewing existing approaches and methods for accessing and modifying business rules reported in the research papers<br>c. Studying possible approaches and methods of formalizing business rules<br>d. Providing critical analysis and evaluation of the researched papers to establish real gaps and limitations to the existing business rules problem. | Met    | Details can be found in section 2.1 covering a survey of existing works. Based on research questions and objectives, we were able to investigate various research studies and identify some very real gaps. This helped to gain an understanding of the problem, how existing solutions work and highlight the work to be carried out in this research.              |
| 2. Study business rules and workflow systems and products in the market today<br>a. Identify and get familiar with relevant workflow business rules systems and products to understand the trends of what has been done in today's market.<br>b. Review existing approaches and methods for modifying business rules provided by these systems and products<br>c. Provide critical analysis of the systems and products to establish the real gaps and limitations to the existing business rules problem.                                                                                | Met    | Details can be found in section 2.2 covering a survey of existing works. Based on research questions and objectives, we were able to investigate various state of the art products and identify real gaps to highlight the work to be carried out in this research.                                                                                                  |
| 3. Using a suitable methodology to establish and design concepts necessary to support the management and administration of business rules in workflows<br>a. Define business rule structure<br>b. Define business rules concepts<br>c. Define business process concepts to be supported by business rule concepts                                                                                                                                                                                                                                                                         | Met    | Details can be found in Chapter 3. This chapter discusses the definition of a business rule, basic structure and concepts of business rules that are the building blocks of the proposed formal model. The EBNF definitions of the business rule concepts are presented to support the development of a formal model described in Chapter 4.                         |
| 4. Develop a formal model to define business rules concepts and relationships.<br>a. Define a methodology of proposed business rules model<br>b. Define the framework of proposed business rules model for formal business rules concepts definitions<br>c. Define business rules classifications<br>d. Define business rules relationships formal definitions and dependency graphs                                                                                                                                                                                                      | Met    | Details can be found in Chapter 4. The formal model is developed based on the understanding of existing business workflows as event-driven and as a constantly evolving process of incremental development, execution and control. Different business rules classifications are also discussed. The AND-OR graph is developed to manage business rules relationships |
| 5. Validate the proposed model by using prototype to demonstrate the following capabilities:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | Met    | Exhaustive use cases scenarios allow deep examination to provide a realism and richness of the proposed model. Several                                                                                                                                                                                                                                               |

|                                                                                                                                                                                                                                               |             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>5a. Provide runtime support for dynamic creation, modification and deletion of business rule and event, condition, action components</p> <p>5b. Provide support for managing business rules and components relationships in real time.</p> |             | <p>workflow use cases from data centres were used to validate and shape the prototype to compare how the research questions played out in the different contexts. Chapter 8 covers the model validation.</p> <p>Details on model design to support creation, modification and deletion of business rules and components (event, condition, action) can be found in Chapter 4. Chapter 5 covers business rule runtime modification using Metarules and business rules indexing to provide support for an efficient mechanism for runtime modification. Implementation can be found in Chapter 7.</p> |
| <p>5c. Provide support for managing change propagation between business rules and components.</p>                                                                                                                                             | Partial Met | <p>The change propagation approach is covered in Chapter 6 and the actual implementation is covered in Chapter 7. Based on AND-OR graph patterns described in Chapter 4, five change propagation patterns (Path, Direct-Node, Level, Neighbour and Indirect Node) were identified for propagation algorithm but only Path and Level Dependency propagations were implemented due to time limitation.</p>                                                                                                                                                                                            |
| <p>5d. Provide support for managing business rules adaptation to control govern a workflow, hence provide support for managing process flows within a workflow.</p>                                                                           | Met         | <p>The Adaptation algorithm is found in Chapter 6 and its implementation is covered in Chapter 7. The algorithm made it possible for business rules to be used to:</p> <ul style="list-style-type: none"> <li>✓ initiate and terminate workflow processes</li> <li>✓ execute Sequential flow patterns for workflow patterns</li> <li>✓ execute Parallel-OR Merged workflow patterns</li> <li>✓ execute Parallel-AND Merged workflow patterns</li> <li>✓ execute Parallel-OR Split workflow patterns</li> <li>✓ execute Parallel-AND Split workflow patterns</li> </ul>                              |

Table 9.2. 1 Status of Achieved Objectives

### **9.3 Contributions to the Knowledge**

This section gives a list of contributions as results of this research. The main contribution of this research is the formalisation and development of business rules framework to govern workflow. More precisely, the research contributions can be summarized as follows:

First - A conceptual model for rule adaptation based on incremental propagation of the changes across the network of inter-related business rules:

The establishment of the business rule model makes it possible for business rule components structure to be separated from workflow programming scripts. The user can create, modify and delete any component of a business rule at any time without updating and recompiling the programming codes. It becomes possible to use business rules and components in real time to control workflows. This provides support to the volatile and complex BRMSs, making them agile, dynamic and efficient.

Second - The multi-layered system architecture with clear separation of process ontology, rule policies and metarules:

The contribution is a loosely coupled framework integrating two-level paradigms for business rules governed process workflows, which is based on a strict logical formalization of the business ontology discussed in Chapter 4. The efficiency of workflows' operations could be largely improved. Processes creation and configuration which are currently done by technical workflow experts could be handled automatically by the businesspeople. Therefore, organisations can save time and the costs, such as the resource costs, would be extremely reduced. Moreover, the workflow will lead to fewer programming errors due to the programmer not understanding the requirements and can do much less coding.

Third - The algorithm for semantic indexing of the rules, which accounts for their structure and results in the AND-OR dependency graph:

The structuring of the business rules into AND-OR graphs provides support for more efficient implementation of business rules change propagation algorithms. Furthermore, the different patterns of inclusion of the business rules in the graphs are used inside the indexing algorithm to control the flow of execution and retrieving of the rules as the business processes progress in real

time. This provides a real support for the management of business rules dependencies and change propagation.

Fourth - Metarules to support runtime modification of business rules:

Developed the Metarules concept to support runtime modification of business rules and indexes. Metarules are business rules described on behalf of other business rules. The Metarules provide an important concept to manage existing business rules and their future accessibility or modification.

Fifth - The incremental algorithm for change propagation which uses the AND-OR graph and results in the actual rule adaptation:

The algorithm has been developed for handling of the business rule change propagation problem. It is important to understand that although the AND-OR graphs made it easier to realize the relationship between rules, the actual change propagation is translated through the propagation algorithm into rule language for workflow interpretation. The business rule adaptation algorithm facilitates the execution of business rules to govern processes in workflows. Another important contribution is that we use the business rules components (Event, Condition, Action) model to automatically detect and execute processes in workflows. We support a comprehensive set of business rules' flow patterns for automatic generation of workflow. The flow patterns rules include initiation rules, termination rules, sequential flow rules, parallel AND-Split flow rules, parallel OR-Split flow rules, parallel AND-Merge flow rules, parallel OR-Merge flow rules, etc., which for example allow us to start, stop, create and delete workflow processes.

On a final note, contributions from this research can be used to extend the concepts already developed as part of business process definition languages such as BPM to support the creation and design processes of workflows from business rules' event, condition and action constructs perspective. The design constructs created by this research extends the current knowledge for business process modelling.



## 9.4 *Limitations and Future Research*

Opportunities for further research are many and varied. This section presents known limitations of this research work, from which recommendations are proposed for future studies.

- Business processes consisting of a sequence of decisions in specific moments of time, like after completing the process steps. In the future we could extend the use of the approach to business processes where the changes can happen at any time (within the processes as well as outside the processes).
- Another possible limitation is the avoidance of parallelism - in the future we could consider simultaneous events, actions and decisions.
- Several algorithms have been developed in this research to support change propagation, adaptation of business rules in workflows and indexing of business rules to improve performance. Further implementation of a series of algorithms for logical analysis of the business rules could be implemented, leading to other applications of the framework, for example analysing the logical vulnerability in digital banking.
- We acknowledge that the implementation of fact classes requires future enhancement to handle the creation and deletion of various facts. Generic classes are required to handle different types of facts instead of the implementation of specific classes for individual fact classes. To enhance the implementation of Fact classes, Java Spring framework implementation could be used to handle creation and deletion of various facts. The implementation of bean classes [103] in Spring is important to use, allowing us to have Java fact classes that live within the application context without constantly creating new fact instances every time we need. Furthermore, the Spring framework can maintain the objects in main memory effectively reducing the risk of running out of memory [97]. Spring works in a way that it finds most inactive or passive objects in the main memory and then copies to the secondary storage to create space for new objects.
- In the proposed model, business rules components are constructed from business rules statements (English statements). They are formatted and expressed in a simple and easy way to identify what part is an event, a condition or an action and translated directly to the rule template described in Chapter 7. This is an important step, because it especially helps to avoid contradictions when using ambiguous English-like statements. Currently, the process of translating English-like statements to business rule components entered through

the ECA Test Client is done manually. A better programmatic approach needs to be developed for higher reliability and efficiency to support automatic generation of business rule components from phrases in Natural Language.

- The developed prototype does not consider low level workflow activities, such as sub-processes and tasks. Typically, a process may consist of several tasks to be performed before moving to the next processes. They need to be considered to further improve the use of business rules in workflow.
- The use of business rules and components could be further extended to control actor roles in workflows. Actor roles do play a fundamental part; a role concept is concerned with who is responsible for doing a process or activity in a workflow (Chapter 3). With the current prototype implementation, the creation, deletion and modification of roles are done through workflow. This creates additional work and reliance on a workflow engine or process management system. The implementation of workflow users' roles using business rules components is also vital. A business rule may spell out which actor role has to be selected, created, deleted or modified in a workflow. It is not necessary to introduce a separate business process management system to manage roles. Furthermore, the business rules will reduce the effort required to manage actor roles in a workflow.
- Currently business rules and components dependencies are represented in a graphical text format. The graphical visualisations of business rules and components dependencies could be further improved by integration with visual graphical tools, such as protégé [2]. Such tools could help to build knowledge-based solutions of business rules and dependencies that come from different areas as diverse as banking, e-commerce and education, insurance. Furthermore, a fully-fledged enterprise editor to load multiple business rules and components will need to be considered.
- Currently, the prototype provides integration with Drools to allow storage and execution of business rules as well as workflow. The integration with Drools itself is not a limitation but the use of Drools specific formats is a limitation. Future research can develop an abstract format and translate the specific formats into it using suitable adapters.

## References

- [1] Allemang Dean, Hendler James, "Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL" 2 Edition ISBN-13: 978-0123859655, (2011).
- [2] Al-zebari Adel Ali, et al., ELMS-DPU Ontology Visualization with Protégé VOWL and Web VOWL, Journal of Advanced Research in Dynamical and Control Systems 11:478-485 Project: e-learning system based on semantic web technologies, (2019).
- [3] Anantaram C., "A Framework to specify Declarative Rules on Objects, Attributes and Associations in the object model", in Journal of Object Technology, vol. 6, pp91-106, (2007).
- [4] Arkin A., et al. "Web Services Business Process Execution Language (WSBPEL)", Version 2.0, (2005), <http://www.oasis-open.org/committees/download.php/12791>
- [5] Barrera, L. F., Ramos, A. C., Florez Valencia, L., Pavlich-Mariscal, J. A., and Mejia-Molina, N. A., "Integrating Adaptation and HCI Concepts to Support Usability in User Interfaces-A Rule-based Approach". In WEBIST (2) - pp. 82-90, (2014).
- [6] Barrera L., Pavlich-Mariscal Jaime A., Carrillo-Ramos Á., Florez-Valencia L., Carrillo A., Runa-Kamachiy: Conceptual integration model between HCI and adaptation oriented to user interface usability, Published (2013), Corpus ID: 20493691.
- [7] Belhajjame K., Vargas-Solar, G. and Collet C., "A flexible workflow model for process-oriented applications," Proceedings of the Second International Conference on Web Information Systems Engineering, Kyoto, Japan, (2001), pp. 72-80 vol.1, doi: 10.1109/WISE.2001.996468.
- [8] Berretti S., Bimbo A. D. and Vicario E., "Efficient matching and indexing of graph models in content-based retrieval". In IEEE Trans. on Pattern Analysis and Machine Intelligence, vol 23, (2001).
- [9] Booch Grady, "Object-oriented analysis design with applications, design technology, computer automated design, computer science, computer engineering", (1994).
- [10] Booch Grady, "The Unified Modeling Language User Guide (Addison-Wesley Object Technology Series)"; Publisher: Addison-Wesley (2005), ISBN-10: 013485215X.
- [11] Boyer Jérôme & Mili Hafedh, "Agile Business Rule Development: Process, Architecture, and JRules Examples", (2011). Springer Science & Business Media 1 ISBN: 9783642190407.

- [12] Brod M, Tesler L, Christensen T. Qualitative research and content validity, “developing best practices based on science and experience”. *Qual Life Res* (2009); 18:1263–78.
- [13] Browne, P., JBoss Drools Business Rules, Packt Publishing, (2009), ISBN-10: 1847196063.
- [14] Business Rules Group, Organizing Business Plans: The Standard Model for Business Rule Motivation, v1.0. (2000), Available at [www.BusinessRulesGroup.org](http://www.BusinessRulesGroup.org).
- [15] Bruce Silver Associates, BPMN | Integrating Process and Rules; (2009).
- [16] Casati et al., “Using Patterns to Design Rules in Workflows.” *IEEE Trans. Software Eng.* 26, 760-785. (2000).
- [17] Christian de Sainte Marie, IBM Web Sphere ILOG BRMS, IBM. (2011), [available at <https://www-01.ibm.com/software/integration/business-rule-management/jrules-family/>].
- [18] Clement B, Roy D, Oudeyer P-Y, Lopes M, “Online optimization of teaching sequences with multi-armed bandits”. In: 7th international conference on educational data mining, (2014).
- [19] Cognizant, The Robot and I: How New Digital Technologies Are Making Smart People and Businesses Smarter by Automating Rote Work. White Paper [available online at <https://www.cognizant.com/whitepapers/the-robot-and-i-how-new-digital-technologies-are-making-smart-people-and-businesses-smarter-codex1193.pdf>], (2015).
- [20] Cubrilo, M., Malekovic. M., Business Rules Modelling by Means of F-logic, UML and Ontologies (problems and possible solutions), *Intelligent Systems at the Service of Mankind*, Volume I, Ubooks, Neusäß, (2004).
- [21] Dayal U., Buchmann A.P., McCarthy D.R., Rules are objects too: A knowledge model for an active, object-oriented database system. In: Dittrich K.R. (eds) *Advances in Object-Oriented Database Systems. OODBS (1988.) Lecture Notes in Computer Science*, vol 334. Springer, Berlin, Heidelberg. [https://doi.org/10.1007/3-540-50345-5\\_9](https://doi.org/10.1007/3-540-50345-5_9).
- [22] Desruelle, H., Isenberg, S., Botsikas, A., Vergori, P., and Gielen, F., “Accessible user interface support for multi-device ubiquitous applications: architectural modifiability considerations” *Universal Access in the Information Society*, 15(1), 5-19, (2016).
- [23] Dragos, A. M. and Ralph E. J. Dynamic Object Model and Adaptive Workflow, (1999).
- [24] Drools Latest final version: 7.38.0.Final License: Apache License 2.0, <https://www.drools.org/download/download.html>.
- [25] Embley David W, Understanding object-model concepts; Publication: ACM SIGPLAN OOPS; (1993) <https://doi.org/10.1145/260304.260364>.

- [26] Embury S. & Shao J. Analysing the Impact of Adding Integrity Constraints to Information Systems in Proc. of 15th Int. Conf. on Advanced Information Systems Engineering, LNCS vol. 2681, pages 175–192, (2003).
- [27] Ezekiel K., Vassilev, V., Ouazzane, K. and Patel, Y., "Adaptive business rules framework for workflow management", Business Process Management Journal, Vol. 25 No. 5, pp. 948-971 publisher by Emerald Publishing Limited, (2019), <https://doi.org/10.1108/BPMJ-08-2017-0219>.
- [28] Fair Isaac Corporation, Production Rule Representation, submitted to Business Modelling and Integration Domain Taskforce, ILOG SA 2007.
- [29] Feldman Jacob, "CTO Creating, Testing, and Executing Decision Models with OpenRules", (2011).
- [30] Feters Linda K., Handbook of Indexing Techniques: A Guide for Beginning Indexers Paperback, Publisher: Feters Info management Co; (2001), ISBN-10: 0929599055.
- [31] Forgy C. L., A fast algorithm for the many pattern/many object pattern match problem, Artificial Intelligence, 17-27. (1982).
- [32] Fossum Timothy V., Classes as first-class objects in an environment-passing interpreter: Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, ITiCSE, Portugal, (2005).
- [33] François Bry, Michael Eckert, Paula-Lavinia Pătrânjan, Inna Romanenko, "Realizing Business Processes with ECA Rules: Benefits, Challenges, Limits"4th International Workshop, PPSWR (2006), Budva, Montenegro, Print ISBN978-3-540-39586-7.
- [34] Frederic P. Miller, Agnes F. Vandome, McBrewhster John. Extended Backus-Naur Form, VDM Publishing, (2010), ISBN: 6130764871, 9786130764876.
- [35] Gartner Group, "Delivering IT's Contribution: CIO Agenda". Stamford, Connecticut, Gartner, Inc. (2005).
- [36] Geppert Andreas, Tombros Dimitrios and Dittrich Klaus R., Defining the semantics of reactive components in event-driven workflow execution with event histories, Information Systems Volume 23, Issues 3–4, May–June 1998, Pages 235-252.
- [37] Giarratano. J. C, "CLIPS User's Guide," (2002), Retrieved from: [www.ghg.ne/clips/download/documentation/usrguide.pdf](http://www.ghg.ne/clips/download/documentation/usrguide.pdf).

- [38] Giurca, A., Lukichev, S. and Wagner, G. "Modeling web services with URML", in Proceedings of SBPM 2006, 11 June, Budva, Montenegro.
- [39] Graham, I., "Business Rules Management and Service Oriented Architecture: A Pattern Language" 1st Edition, Publisher: Wiley, ISBN-13: 978-0470027219, (2007).
- [40] Graml, T., "Business Rules enable agile Business Process Management", Master thesis, Institute for Informatics, Der Ludwig-Maximilians-University Munchen, (2006).
- [41] Goh et al., "ECA rule-based support for workflows", Artificial Intelligence in Engineering Volume 15, Issue 1, January 2001, Pages 3-46.
- [42] Graml, T., Bracht, R. and Spies, M., "Patterns of Business Rules to Enable Agile Business Processes". In: Proceedings of 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC'07, Oct. 15-19, Annapolis, Maryland, USA), pages 365-378, (2007).
- [43] Grissa-Touzi, A., Ounally, H., Boulila, A., VISUAL JESS: An expandable visual generator of oriented object expert systems. Engineering and Technology; Pages 108–111, (2005).
- [44] Grumbach Lisa and Bergmann Ralph, "SEMAFLEX: A novel approach for implementing workflow flexibility by deviation based on constraint satisfaction problem solving", Published in Expert Systems (2019) <https://doi.org/10.1111/exsy.12385>.
- [45] Guenther C. W., Reichert M. and Van der Aalst W. M. P., "Supporting Flexible Processes with Adaptive Workflow and Case Handling," (2008), IEEE 17th Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, Rome, (2008), pp. 229-234, doi: 10.1109/WETICE.2008.15.
- [46] Hall J., "Semantics of Business Vocabulary and Business Rules (SBVR) -Model Systems", (2006).
- [47] Haley Paul, Confessions of a production rule vendor, Commercial Intelligence (2013), [available at <http://haleyai.com/wordpress/2013/06/22/confessions-of-a-production-rule-vendor-part-1>].
- [48] Harrison Gregory, "Dynamically configurable workflow in a mobile environment", (2019), <https://patents.google.com/patent/US20160132299>.
- [49] Heinz Lienhard and Urs-Martin Künzi, "Workflow and Business Rules: A Common Approach", The Workflow Handbook (2005), published in association with the Workflow Management Coalition (WfMC). [www.wfmc.org/information/handbook05.htm](http://www.wfmc.org/information/handbook05.htm).

- [50] Hilwa Al & Hendrick Stephen D., Competitive Analysis Worldwide Business Rules Management Systems, IDC. (2012) available online at [ftp://public.dhe.ibm.com/software/websphere/odm/2011\\_BRMS\\_MarketShare\\_Report.pdf](ftp://public.dhe.ibm.com/software/websphere/odm/2011_BRMS_MarketShare_Report.pdf)
- [51] Javier Cámara, Carlos Canal, Javier Cubo, Antonio Vallecillo, Formalizing WSBPEL Business Processes Using Process Algebra, Part of special issue: Proceedings of the 4th International Workshop on the Foundations of Coordination Languages and Software Architectures, FOCLASA: (2005).
- [52] JBoss Drools <http://www.jboss.org/drools/>.
- [53] Kambur Dalen, Roantree Mark: Storage of Complex Business Rules in Object Databases. ICEIS (1), 294-299, (2003).
- [54] Koch A, Burns J, Catchpole K, et al Associations of workflow disruptions in the operating room with surgical outcomes: a systematic review and narrative synthesis BMJ Quality & Safety 29:1033-1045, (2020).
- [55] Kumar, A. and R. Liu. "A Rule-Based Framework Using Role Patterns for Business Process Compliance." RuleML, 2008.
- [56] López-Jaquero, V., Montero, F., and Real, F. "Designing user interface adaptation rules with T: XML". Proceedings of the 14th international conference on Intelligent user interfaces - pp. 383-388, (2009).
- [57] Lovrenčić S., Rabuzin K., and Picek R., "Formal Modelling of business rules: what kind of tool to use?", Journal of Information and Organizational Sciences, JIOS, vol. 30, no. 2, 1.
- [58] Liu Di, Tao Gu and Xue Jiang-Ping., "Rule Engine based on improvement Rete algorithm," The 2010 International Conference on Apperceiving Computing and Intelligence Analysis Proceeding, Chengdu, pp. 346-349, doi:10.1109/ICACIA.2010.5709916, (2010).
- [59] Ludwig Ostermayer, Dietmar Seipel, Knowledge Engineering for Business Rules in PROLOG, Conference: 26th Workshop on Logic Programming, Bonn, Germany, September 24 - 25, 2012, Volume: Technical Report Nr. IAI-TR-2012-1, ISSN 0944-8535
- [60] Lukichev, S., Wagner, G., "UML-Based Rule Modelling with Fujaba" Proceedings of the 4th International Fujaba, Germany, pp: 31- 35, (2006).
- [61] Macdonald Andrew, The value of IBM WebSphere ILOG BRMS, IBM. (2010), [available at <https://www-01.ibm.com/software/integration/business-rule-management/jrules-family/>].

- [62] Mangan, PJ and Sadiq, S, "A constraint specification approach to building flexible workflows", Journal of Research and Practice in Information Technology (ERA 2010, 2012, 2015, 2018 Journal(s) Listed), ISSN 1443-458X, Publisher Australian Computer Soc Inc 2003.
- [63] Manning Ernest Friedman-Hill, Jess in Action, Rule Based Systems in Java., (2003).
- [64] Manning Ernest Friedman-Hill, Jess the rule engine for the java platform, URL <http://herzberg.ca.sandia.gov/jess/docs/70/> (2006).
- [65] Mathias Weske, "Business Process Management: Concepts, Languages, Architectures", Publisher: Springer (2012), ISBN-10: 3642286151.
- [66] McIver Annabelle and Morgan Carroll, "Programming Methodology", Publisher: Springer-Verlag New York, ISBN 978-0-387-95349-6, (2003).
- [67] Meir Doron (2018), "Workflow: A Practical Guide to the Creative Process" 1st Edition, Publisher: CRC Press; 1 edition, ISBN-10: 113805853X
- [68] Mezhoudi, N., PerezMedina, J. L., and Vanderdonckt, J., "Towards a Conceptual Model for UIs Context-Aware Adaptation". Proceedings of the 2nd World Congress, (2015).
- [69] Miller. Frederic P., et al., Logical Disjunction: Logic, Mathematics, Logical connective, Grammar, Grammatical conjunction, Exclusive or, Affirming a disjunct, Bitwise operation, (2010), Operator, Disjunctive syllogism Paperback, Alpha Script Publishing, ISBN-10: 6130293976.
- [70] Miñón, R., Paternò, F., Arrue, M., and Abascal, J., "Integrating adaptation rules for people with special needs in model-based UI development process". Universal Access in the Information Society, 15(1), 153-168, (2015).
- [71] Mnaouer Ben Adel et al., "A generic framework for rapid application development of mobile Web services with dynamic workflow management", Conference: Services Computing Proceedings of the 2004 IEEE International Conference on Services Computing (SCC'04).
- [72] Mulholland Ben, BPM Statistics to Help You Increase Efficiency in Your Business <https://www.process.st/bpm-statistics-increase-efficiency/> (2017).
- [73] Musen, M.A. The Protégé project: A look back and a look forward. AI Matters. Association of Computing Machinery Specific Interest Group in Artificial Intelligence, 1(4), June 2015. DOI: 10.1145/2557001.25757003.



- [74] NASA's Johnson Space Centre, CLIPS: A tool for building expert systems; (2008).  
<http://clipsrules.sourceforge.net/>.
- [75] Nhanle Thanh, Nhan Le Thanh, "An Ontology-based Approach for Business Process Compliance Checking", Publication: IMCOM '16: Proceedings of the 10th International Conference on Ubiquitous Information Management and Communication, January 2016  
Article No.: 56 Pages 1–6 <https://doi.org/10.1145/2857546.2857603>.
- [76] Nicola Jill, Mayfield Mark, Abney Mike; "Streamlined Object Modeling: Patterns, Rules, and Implementation" Published Sep 21, 2001 by Pearson. ISBN-10: 0-13-066839-7.
- [77] Nguyen THH., Le-Thanh N. (2014) An Ontology-Enabled Approach for Modelling Business Processes. In: Kozielski S., Mrozek D., Kasprowski P., Małysiak-Mrozek B., Kostrzewa D. (eds) Beyond Databases, Architectures, and Structures. BDAS 2014. Communications in Computer and Information Science, vol 424. Springer, Cham.  
[https://doi.org/10.1007/978-3-319-06932-6\\_14](https://doi.org/10.1007/978-3-319-06932-6_14).
- [78] Norman W. Paton (2012) Active Rules in Database Systems, Publisher: Springer; (September 5, 2012), ISBN-13: 978-1461264484.
- [79] Ocke Stefan, "A Metamodel-Based OCL-Compiler for UML and MOF", Publication: Electronic Notes in Theoretical Computer Science (ENTCS), November 2004  
<https://doi.org/10.1016/j.entcs.2003.09.003>.
- [80] Oguz Gizil, Proctor Mark, Kuncak Viktor, "Decision Tree Learning for Drools" Scientific production and competences I&C - School of Computer and Communication Sciences; IINFCOM LARA - Laboratory for Automated Reasoning and Analysis Work produced at EPFL, (2008).
- [81] OMG final adopted specification, BPMN 1.0, (2006), <http://www.omg.org/cgi-bin/doc?dtc/2006-02-01>.
- [82] OMG, Documents Associated with Business Process Model and Notation (BPMN), 2011,  
<http://www.omg.org/spec/BPMN>.
- [83] OMG, Semantics of Business Vocabulary and Business Rules (SBVR), 2017,  
<http://www.omg.org/spec/SBVR/1.4/PDF>.
- [84] OMG Business Process Model and Notation, January 2014, Version:2.0.2  
<http://www.omg.org/spec/BPMN/2.0.2/>.

- [85] Oracle BPM, Business Process Management, (2017).  
<http://www.oracle.com/us/technologies/bpm/overview/index.html>
- [86] Oracle Workflow User's Guide Release 2.6.3 Part Number B10285-02;  
[https://docs.oracle.com/cd/B12037\\_01/workflow.101/b10285/ugov.htm](https://docs.oracle.com/cd/B12037_01/workflow.101/b10285/ugov.htm).
- [87] Ouyang, Chun, van der Aalst, Wil, Dumas Menjivar, Marlon, & ter Hofstede, "Formal Semantics and Analysis of Control Flow in WS-BPEL", Science of Computer Programming, 67(2-3), pp. 162-198; 2007; ISSN:0167-6423.
- [88] Poornachandra Sarang, Matjaz Juric, Benny Mathew; Business Process Execution Language for Web Services: An Architects and Developers Guide to BPEL and BPEL4WS; Publisher: Packt Publishing; 2Rev Ed Edition (2006), ISBN-10: 1904811817.
- [89] PROCESOWCY.PL, Business process maturity in Polish organisations overview, 3rd Edition, (2016).
- [90] Proctor, et al., Drools Expert User Guide (2011).
- [91] Rajkumar Thirumalainambi, Pitfalls of JESS for Dynamic Systems (Artificial Intelligence and Pattern Recognition); Pages 491-494, (2007).
- [92] Ramakanth Kotha, Designing Business Rules with Oracle Business Process Management, (2018).
- [93] Raza Abdullah, Dynamic Partitioning and Task Scheduling for Complex Workflow Healthcare Application in Mobile Edge Cloud Architecture, (2019).
- [94] Rabova, I., "Business rules specification and business processes modeling," Agricultural Economics-Zemledelska Ekonomika (55:1), 20–24, (2009).
- [95] Regev Gil, Bider Ilia, Wegmann. Alain, "Defining Business Process Flexibility with the Help of Invariants Software Process": Improvement and Practice (SPIP), V12(1), pp. 65-79 (2007).
- [96] Richters Mark, "A Precise Approach to Validating UML Models and OCL Constraints" Publisher: Logos Verlag Berlin (30 Jan. 2002), ISBN-10: 3897228424.
- [97] Rod Johnson et al., "Professional Java Development with the Spring Framework", Publisher: Wrox, (2005), ISBN-10: 0764574833.
- [98] Ronald G. Ross, "Principles of the Business Rule Approach", Published February 15th, 2003 by Addison-Wesley Professional, ISBN0201788934 (ISBN13: 9780201788938).

- [99] Ronald G. Ross., Business Rule Concepts: Getting to the Point of Knowledge (4th Edition) ISBN: 0-941049-14-0; (2013).
- [100] Ronald G. Ross and Gladys S. W. Lam., BRSolutions, the BRS Business Rule Methodology, (2000).
- [101] Rosenberg, F., Nagl C. and Dustdar S., "Applying Distributed Business Rules - The VIDRE Approach," in 2006 IEEE International Conference on Services Computing, Chicago, IL, 2006 pp. 471-478. doi: 10.1109/SCC.2006.22 [Accessed Jan 26, 2018].
- [102] Rowe Anthony et al., "The use of Business Rules with Workflow Systems" [available online at <https://www.w3.org/2004/12/rules-ws/paper/105/>], (2004).
- [103] Rubinger Andrew Lee, Enterprise JavaBeans 3.1, Publisher: O'Reilly Media; 6 Edition, (2010), ISBN-13: 978-0596158026.
- [104] Rumbaugh James R, Blaha Michael R., Lorensen William, Eddy Frederick, Premerlani William, "Object-Oriented Modeling and Design", Publisher: Pearson; International Ed Edition (1 Mar. 1991), ISBN-10: 0136300545.
- [105] Salatino Mauricio, De Maio Mariano, Aliverti Esteban, Mastering JBoss Drools 6, Packt Publishing, (2016), ISBN-13: 978-1783288625.
- [106] Salesforce, The State of Sales Report - Insights and trends from over 2,900 sales professionals worldwide, third Edition (2020), <https://www.salesforce.com/research/>.
- [107] Sakr Sherif and Al-Naymat Ghazi, "An Overview of Graph Indexing and Querying Techniques" Graph Data Management: Techniques and Applications, Published 2012 DOI: 10.4018/978-1-61350-053-8.ch004.
- [108] Sedgewick Robert, "Algorithms in Java: Parts 1-4", Publisher: Addison Wesley, (2002), ISBN: 0-201-36120-5.
- [109] Hendrick Stephen D & Hendrick Kathleen E., "The Business Value of Business Rules Management Systems" (2012), Sponsored by IBM.
- [110] Tabebordbar, A., Beheshti, A., Benatallah, B. et al. Feature-Based and Adaptive Rule Adaptation in Dynamic Environments. Data Sci. Eng. 5, 207–223 (2020). <https://doi.org/10.1007/s41019-020-00130-4>.
- [111] Taentzer, G. "AGG: A Tool Environment for Algebraic Graph Transformation". Proceedings of the International Workshop on Applications of Graph Transformations with industrial Relevance. LNCS, vol. 1779. Springer, London, 481-488, (2000).

- [112] Taentzer Gabriele, "Adding Visual Rules to Object-Oriented Modeling Techniques", Technical University of Berlin Volume: 1, Pages: 275, Year: (1999).
- [113] Tan C.W. and Goh A, "Implementing ECA Rules in an Active Database," Knowledge-Based Systems, vol. 12, no. 4, pp. 137-144; (1999).
- [114] Thirumaran.M, Ilavarasan.E, Thanigaivel.K and Abarna.S, Business rule management framework for enterprise web services, (2010).
- [115] Thomas J., et al., "Automated epileptiform spike detection via affinity propagation-based template matching," 39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), (2017), Seogwipo, pp. 3057-3060.
- [116] Tombros Dimitrios, Geppert Andreas and Dittrich Klaus R., Semantics of reactive components in event-driven workflow execution, Conference: International Conference on Advanced Information Systems Engineering, (2006).
- [117] Tristan Yates, Enhanced Indexing Strategies: Utilizing Futures and Options to Achieve Higher Performance (Wiley Trading) Hardcover, Publisher: Wiley; 1 Edition, (2008), ISBN-13: 978-0470259252.
- [118] Uhlmann E., Gabriel C., and Raue, N. "An Automation Approach Based on Workflows and Software Agents for Industrial Product-Service Systems" Procedia CIRP, vol. 30, 2015. doi:10.1016/j.procir.2015.02.026.
- [119] Van der Aalst Wil and Van Hee Kees, "Workflow Management: Models, Methods, and Systems", Published in Cooperative information. Computer Science, Economics. (2002), ISBN 0-262-01189-1.
- [120] Van der Aalst, W.M.P; ter Hofstede, A.H.M and Weske M., Business Process Management: A Survey - The First International Conference of Business Process Management; (2003).
- [121] Van der Aalst W. M., ter Hofstede A., Kiepuszewski B., and Barros, A. Workflow patterns. Distributed and Parallel Databases, 14(3):5–51, July 2003.
- [122] Veerendra Kumar Rai, 'Systems Approach to Business Rules', Tata Research Development and Design Centre, Proceedings of the 20th System Dynamics Conference, 2002.
- [123] Wan M. N. Wan-Kadir, Pericles Loucopoulos: Relating Evolving Business Rules to Software Design. Software Engineering Research and Practice 2003: 129-134.

- [124] Waszkowski Robert and Kowalski Arkadiusz, Comparative Analysis of Business Process Management Frameworks, 2017, ISBN: 978-0-9860419-9-0  
<https://www.researchgate.net/publication/321826235>.
- [125] Wiegmann DA, Sundt TM Workflow disruptions and surgical performance: past, present and future BMJ Quality & Safety 2019;28:260-262.
- [126] WS-BPEL (Web Services Business Process Execution Language). In: ManagementMania.com [online]. Wilmington (DE) 2011-2020, 07/31/2015 [cit. 09/26/2020]. Available at: <https://managementmania.com/en/ws-bpel-web-services-business-process-execution-language>.
- [127] Xifeng Yan, Philip S. Yu, and Jiawei Han. Graph indexing: a frequent structure-based approach. In Proceedings of the 2004 ACM SIGMOD international conference.
- [128] Yan X., Han J., Graph Indexing. In: Aggarwal C., Wang H. (eds) Managing and Mining Graph Data. Advances in Database Systems, vol 40. Springer, Boston, MA. (2010), [https://doi.org/10.1007/978-1-4419-6045-0\\_5](https://doi.org/10.1007/978-1-4419-6045-0_5).
- [129] Yoder J., Federico B. and Ralph J., Adaptive Object-Models for Implementing Business Rules, (2001).
- [130] Yuan, Dayu and P. Mitra. "Lindex: a lattice-based index for graph databases." The VLDB Journal Vol 22, pp 229-252, (2012).
- [131] Yuyin Sun, L. Bo, D Fox, Attribute based object identification, Published 2013 IEEE International Conference on Robotics and Automation, Corpus ID: 8413785, (2013).
- [132] Zoet Martijn, Methods and Concepts for Business Rules Management; 2014. ISBN: 978-90-393-6130-6.

## Appendices

### ***Appendix I – Possible Validation Scenarios***

List of possible validation scenarios can be considered for experimentation.

| No | Scenarios                                                           |
|----|---------------------------------------------------------------------|
| 1  | Adding entire business rules & components                           |
| 2  | Adding event component on the fly                                   |
| 3  | Adding condition component on the fly                               |
| 4  | Adding action component on the fly                                  |
| 5  | Adding event and condition components on the fly                    |
| 6  | Adding event and action component on the fly                        |
| 7  | Adding condition and action component on the fly                    |
| 8  | Modifying entire business rules & components                        |
| 9  | Modifying event component on the fly                                |
| 10 | Modifying condition component on the fly                            |
| 11 | Modifying action component on the fly                               |
| 12 | Modifying event and condition components on the fly                 |
| 13 | Modifying event and action component on the fly                     |
| 14 | Modifying condition and action component on the fly                 |
| 15 | Deleting entire business rules & components                         |
| 16 | Deleting event component on the fly                                 |
| 17 | Deleting condition component on the fly                             |
| 18 | Deleting action component on the fly                                |
| 19 | Deleting event and condition components on the fly                  |
| 20 | Deleting event and action component on the fly                      |
| 21 | Deleting condition and action component on the fly                  |
| 22 | Adding entire business rules & components and change propagation    |
| 23 | Adding event component and change propagation                       |
| 24 | Adding condition component and change propagation                   |
| 25 | Adding action component and change propagation                      |
| 26 | Adding event and condition components and change propagation        |
| 27 | Adding event and action component and change propagation            |
| 28 | Adding condition and action component and change propagation        |
| 29 | Modifying entire business rules & components and change propagation |
| 30 | Modifying event component and change propagation                    |
| 31 | Modifying condition component and change propagation                |
| 32 | Modifying action component and change propagation                   |
| 33 | Modifying event and condition components and change propagation     |
| 34 | Modifying event and action component and change propagation         |
| 35 | Modifying condition and action component and change propagation     |

|    |                                                                                                                                                          |
|----|----------------------------------------------------------------------------------------------------------------------------------------------------------|
| 36 | Deleting entire business rules & components and change propagation                                                                                       |
| 37 | Deleting event component and change propagation                                                                                                          |
| 38 | Deleting condition component and change propagation                                                                                                      |
| 39 | Deleting action component and change propagation                                                                                                         |
| 40 | Deleting event and condition components and change propagation                                                                                           |
| 41 | Deleting event and action component and change propagation                                                                                               |
| 42 | Deleting condition and action component and change propagation                                                                                           |
| 43 | Ability to enable a business rule to initiate a process in a workflow                                                                                    |
| 44 | Ability to enable a business rule to terminate a process in a workflow                                                                                   |
| 45 | Ability to enable sequential process flow patterns                                                                                                       |
| 46 | Insertion of business rule components to generate sequential process flow patterns                                                                       |
| 47 | Modification of an existing business rule (changing source or target process flows) in the sequential workflow pattern disconnects process flows         |
| 48 | Deletion of an existing business rule in the sequential workflow pattern disconnects existing process flows                                              |
| 49 | Ability to enable Parallel-OR Merge flow patterns                                                                                                        |
| 50 | Insertion of a new business rule in the OR Merged rules flow pattern create a new process flow connection                                                |
| 51 | Modification of an existing business rule (changing source or target process flows) in the OR Merged rules flow pattern disconnects process flows        |
| 52 | Deletion of an existing business rule in the OR Merged rules flow pattern disconnects existing process flows                                             |
| 53 | Ability to enable Parallel-AND Merged flow patterns                                                                                                      |
| 54 | Insertion of a new business rule in the AND Merged rules flow pattern create a new process flow connection                                               |
| 55 | Modification of an existing business rule (changing source or target process flows) in the AND Merged rules flow pattern disconnects process flows       |
| 56 | Deletion of an existing business rule in the AND Merged rules flow pattern disconnects existing process flows                                            |
| 57 | Ability to enable Parallel-OR Split flow patterns                                                                                                        |
| 58 | Insertion of a new business rule in the OR Parallel Split workflow pattern create a new process flow connection                                          |
| 60 | Modification of an existing business rule (changing source or target process flows) in the OR Parallel Split workflow pattern disconnects process flows  |
| 61 | Deletion of an existing business rule in the OR Parallel Split workflow pattern disconnects existing process flows                                       |
| 62 | Ability to enable Parallel-AND Split flow patterns                                                                                                       |
| 63 | Insertion of a new business rule in the AND Parallel Split workflow pattern create a new process flow connection                                         |
| 64 | Modification of an existing business rule (changing source or target process flows) in the AND Parallel Split workflow pattern disconnects process flows |

|    |                                                                                                                     |
|----|---------------------------------------------------------------------------------------------------------------------|
| 65 | Deletion of an existing business rule in the AND Parallel Split workflow pattern disconnects existing process flows |
| 66 | Inserting index record for new business rules components relationships insertion or addition                        |
| 67 | Modifying index record for business rules components relationships modification and deletion                        |
| 68 | Deleting index record for business rules components relationships modification and deletion                         |



## Appendix II – Business Rules in XYZ Equipment Install Workflow

Summary of existing and new business rules configured for XYZ's equipment install workflows:

| Business Rule                                                                                                                                                                                                                                  | Notes                                                                                                                                                                                                    |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>When submit request, if requestor is a member of the Platform capacity team then go to 'Review' step else go to 'Approve' step for data centre area manager to approve and set install request</i>                                          | This business rule was added to ensure only valid requests are processed by assigned and approved resources enforcing security to company data centres.                                                  |
| <i>When install request, if rack utilization is greater than the rack space capacity, then count installed equipment and set the Rack is full and set process to manage data centre space else install the equipment in the available rack</i> | This business rule was added to ensure racks are not overloaded and new racks are order when there is enough space in racks.                                                                             |
| <i>When notify Rack is full then count total number of racks installed in the data centre</i>                                                                                                                                                  | This business rule was added to alert data centre managers number of installed equipment when racks are full.                                                                                            |
| <i>If the total number of installed racks is less data centre rack capacity, then order new rack</i>                                                                                                                                           | This business rule was added to alert data centre managers to order new racks when there is enough space in a rack.                                                                                      |
| <i>If the number of equipment power supplies is greater than zero, then set process name to provision power</i>                                                                                                                                | This business rule allows the workflow to flow to "Power Provisioning" process for powered equipment.                                                                                                    |
| <i>If the number of equipment network ports is greater than zero, then set process name to Provision Network</i>                                                                                                                               | This business rule allows the workflow to flow to "Network Provisioning" process for network equipment.                                                                                                  |
| <i>If the number of equipment power connections is equal to equipment power supplies, then set process name to Completing Power and Network Provisioning</i>                                                                                   | This business rule causes the workflow to flow to "Completing Power and Network Provisioning" process if equipment is fully connected i.e. all its power ports have connections.                         |
| <i>If equipment network cable is required, then set process name to Provision Network Cables</i>                                                                                                                                               | This business rule allows the workflow to flow to "Provision Network Cables" process if cable connection is required on equipment.                                                                       |
| <i>If equipment network connections and cables are configured, then set process name to Completing Power and Network Provisioning and request status is set to close</i>                                                                       | If equipment is configured for network and cables then this business rule causes the workflow to flow to "Completing Power and Network Provisioning" process and set the status of the request to close. |
| <i>If request status is set to close, then set process name to close request</i>                                                                                                                                                               | This is a termination rule where workflow flows to close process if request status is set to close.                                                                                                      |

## ***Appendix III – XYZ Business Rules in Drools DRL format***

Drools DRL displaying business rules configured for XYZ's equipment install workflows:

```
package org.kanana;
import org.kanana.MessageFact;
import org.kanana.Rack;
import org.kanana.Request;
import org.kanana.RuleName;
import function org.kanana.Utility.help;
dialect "java"

rule "R12"
when
 Request(Status == 'Close')
then
 help(drools,"Drools runtime Info... ");
 System.out.println("Action after rule triggered: " + "Process(Name == 'P12-Close Request')");
end

rule "R11"
when
 Equipment(Network and Cablling Configured == 'Yes')
then
 help(drools,"Drools runtime Info... ");
 System.out.println("Action after rule triggered: " + "Process(Name == 'P10-Complete Power and Network Provision')");
end

rule "R10"
when
 Equipment(NetworkCablling == 'Yes')
then
 help(drools,"Drools runtime Info... ");
 System.out.println("Action after rule triggered: " + "Process(Name == 'P11-Network Cables Provision')");
end

rule "R9"
when
 Equipment(Power Connections == 'PowerSupplies')
then
 help(drools,"Drools runtime Info... ");
 System.out.println("Action after rule triggered: " + "Process(Name == 'P10-CompletePower and Network Provision')");
end

rule "R8"
when
 Equipment(Network Ports > '0')
then
 help(drools,"Drools runtime Info... ");
 System.out.println("Action after rule triggered: " + "Process(Name == 'P9-Network Provision')");
end

rule "R7"
when
 Equipment(Power Supplies > '0')
then
 help(drools,"Drools runtime Info... ");
 System.out.println("Action after rule triggered: " + "Process(Name == 'P8-Power Provision')");
end

rule "R6"
when
 Rack(Volume < '10')
then
 help(drools,"Drools runtime Info... ");
 System.out.println("Action after rule triggered: " + "Process(Name == 'P6-Order Rack')");
end
```

```

rule "R5"
when
 Rack(Space == 'isfull')
then
 help(drools,"Drools runtime Info... ");
 System.out.println("Action after rule triggered: " + "Rack(availableRacks == 'installedRacks - 1')");
end

rule "R4"
when
 Request(Type == 'Install')
 Rack(Utilization < '2000')
then
 help(drools,"Drools runtime Info... ");
 System.out.println("Action after rule triggered: " + "Process(Name == 'P7-Install Equipment')");
end

rule "R3"
when
 Request(Type == 'Install')
 Rack(Utilization >= '2000')
then
 help(drools,"Drools runtime Info... ");
 System.out.println("Action after rule triggered: " + "Process(Name == 'P5-Manage DC Space')");
end

rule "R2"
when
 Request(Status == 'Submit')
 Requestor(Rolename != 'Platform Capacity')
then
 help(drools,"Drools runtime Info... ");
 System.out.println("Action after rule triggered: " + "Process(Name == 'P3-Approve')");
end

rule "R1"
when
 Request(Status == 'Submit')
 Requestor(Rolename == 'Platform Capacity')
then
 help(drools,"Drools runtime Info... ");
 System.out.println("Action after rule triggered: " + "Process(Name == 'P2-Review')");
end

rule "R0"
when
 Workflow(Activity == 'Start')
then
 help(drools,"Drools runtime Info... ");
 System.out.println("Action after rule triggered: " + "Process(Name == 'P1-Create Request')");
End

```

## ***Appendix IV – XYZ Business Rules Insertion via Drools DRL***

Drools DRL displaying inserted business rules (R13) configured for XYZ's equipment install workflows:

```
package org.kanana;
import org.kanana.MessageFact;
import org.kanana.Rack;
import org.kanana.Request;
import org.kanana.RuleName;
import function org.kanana.Utility.help;
dialect "java"

rule "R13"
when

 Request(Status == 'Install')
then
 help(drools,"Drools runtime Info... ");
 System.out.println("Action after rule triggered: " + "Request(Status == 'Submit')");
End

rule "R12"
when

 Request(Status == 'Close')
then
 help(drools,"Drools runtime Info... ");
 System.out.println("Action after rule triggered: " + "Process(Name == 'P12-Close Request')");
end

rule "R11"
when

 Equipment(Network and Cablling Configured == 'Yes')
then
 help(drools,"Drools runtime Info... ");
 System.out.println("Action after rule triggered: " + "Process(Name == 'P10-Complete Power and Network Provision')");
end

rule "R10"
when

 Equipment(NetworkCablling == 'Yes')
then
 help(drools,"Drools runtime Info... ");
 System.out.println("Action after rule triggered: " + "Process(Name == 'P11-Network Cables Provision')");
end

rule "R9"
when

 Equipment(Power Connections == 'PowerSupplies')
then
 help(drools,"Drools runtime Info... ");
 System.out.println("Action after rule triggered: " + "Process(Name == 'P10-CompletePower and Network Provision')");
end

rule "R8"
when

 Equipment(Network Ports > '0')
then
```

```

 help(drools,"Drools runtime Info... ");
 System.out.println("Action after rule triggered: " + "Process(Name == 'P9-Network Provision')");
end

rule "R7"
when

 Equipment(Power Supplies > '0')
then
 help(drools,"Drools runtime Info... ");
 System.out.println("Action after rule triggered: " + "Process(Name == 'P8-Power Provision')");
end

rule "R6"
when

 Rack(Volume < '10')
then
 help(drools,"Drools runtime Info... ");
 System.out.println("Action after rule triggered: " + "Process(Name == 'P6-Order Rack')");
end

rule "R5"
when
 Rack(Space == 'isfull')

then
 help(drools,"Drools runtime Info... ");
 System.out.println("Action after rule triggered: " + "Rack(availableRacks == 'installedRacks - 1')");
end

rule "R4"
when
 Request(Type == 'Install')
 Rack(Utilization < '2000')
then
 help(drools,"Drools runtime Info... ");
 System.out.println("Action after rule triggered: " + "Process(Name == 'P7-Install Equipment')");
end

rule "R3"
when
 Request(Type == 'Install')
 Rack(Utilization >= '2000')
then
 help(drools,"Drools runtime Info... ");
 System.out.println("Action after rule triggered: " + "Process(Name == 'P5-Manage DC Space')");
end

rule "R2"
when
 Request(Status == 'Submit')
 Requestor(Rolename != 'Platform Capacity')
then
 help(drools,"Drools runtime Info... ");
 System.out.println("Action after rule triggered: " + "Process(Name == 'P3-Approve')");
end

rule "R1"
when
 Request(Status == 'Submit')
 Requestor(Rolename == 'Platform Capacity')

```

```
then
 help(drools,"Drools runtime Info... ");
 System.out.println("Action after rule triggered: " + "Process(Name == 'P2-Review')");
end

rule "R0"
when
 Workflow(Activity == 'Start')

then
 help(drools,"Drools runtime Info... ");
 System.out.println("Action after rule triggered: " + "Process(Name == 'P1-Create Request')");
End
```

## Appendix V – Level-Based Dependency Pattern Index Algorithm

Level Based Dependency Pattern Indexed below:

```
//Creation of indexes for dependency patterns
public List<ECAModel> IndexingGraphPatterns(ECAGraph ecaRuleG){
//variables declaration
int UniqueIndex =0;
List<ECAModel> DependencyPatterns = new ArrayList<ECAModel>();
ECAModel rootcomponent = new ECAModel();
Queue<ECAModel> queue;
PathBasedPattern<ECAModel> pattenIndeces = new PathBasedPattern<ECAModel>(UniqueIndex, rootcomponent);

//loop through graph to identify dependency patterns
for (int index=0; index < ecaRuleG.ruleRelations.size();index++){
 rootcomponent = ecaRuleG.getNode(index);

 //Check the root rule
 if (rootcomponent == null) return null;

 //Create an empty stack and push the root rule to it
 Stack<ECAModel> nodeStack=new Stack<ECAModel>();
 nodeStack.push(rootcomponent);
 rootcomponent.visited=true;

 //Create a map to store parent pointers of graph nodes
 HashMap<ECAModel,ECAModel> parent=new HashMap<ECAModel, ECAModel>();
 //Parent of root is NULL
 parent.put(rootcomponent,null);

 //Traverse through Path Dependency Pattern then generate indexes
 while (!nodeStack.isEmpty()) {
 //Pop the top item from stack
 ECAModel current = nodeStack.pop();
 if(current.hasChildren()) {

 //Convert to object array
 ECAModel[] temppatterns = new ECAModel[current.children().size()];

 //ArrayList to Array Conversion to allow generation of indexes for each path
 for (int pindex=0; pindex < current.children().size();pindex++){
 temppatterns[pindex] = current.children().get(pindex);

 for (ECAModel linkedIndex : temppatterns) {
 //pattenIndeces contains index for the ECA component node and pattern indexes (combining linked ids)
 pattenIndeces = new PathBasedPattern<ECAModel>(pindex,linkedIndex);

 }
 }
 //Create indexes for path dependency
 DependencyPatterns.add(new ECAModel(RuleList.get(pindex).getruleName(), current, pattenIndeces));
 }
 }

 //Traverse through Level-Based Dependency pattern then generate indexes
 queue = new LinkedList<ECAModel>();
 queue.add(rootcomponent);
 rootcomponent.visited=true;
 while (!queue.isEmpty()){

 ECAModel element=queue.remove();
 List<ECAModel> neighbours=element.children();

 //Convert to object array
 ECAModel[] temppatterns = new ECAModel[neighbours.size()];

 for (int lindex = 0; lindex < neighbours.size(); lindex++) {
```

```

tempPatterns[lindex] = neighbours.get(lindex);

ECAModel n = neighbours.get(lindex);
if(n!=null && !n.visited){
 queue.add(n);
 n.visited=true;
 for (ECAModel linkedIndex : tempPatterns) {
 //pattenIndices contains index for the ECA component node and pattern indexes (combining linked ids)
 pattenIndices = new PathBasedPattern<ECAModel>(lindex,linkedIndex);
 }
 //Create indexes for level-based dependency
 DependencyPatterns.add(new ECAModel(RuleList.get(lindex).getruleName(), n, pattenIndices)); }}}
}
return DependencyPatterns;
}

```



## **Appendix VI – JBoss Drools Setup and Installation**

The easiest way to setup Drools is to install Drools Eclipse Plugin. The Eclipse IDE version used in this research is Eclipse Java EE IDE for Web Developers. To install it, use the following built-in update steps:

1. *Start Eclipse*
2. *Go to Help menu -> Install New Software*
3. *In the work with or site: input field, enter:  
"http://download.jboss.org/drools/release/<VERSION>.Final/org.drools.updatesite/", replace  
"<VERSION>" with appropriate version and click the "Add" button*
4. *Enter name details*
5. *Check the Drools and jBPM checkbox and next follow the instructions to get it installed.*
6. *Click next and accept licensing term. Click "Finish". DROOLS plugin will start installing into eclipse.*
7. *After the installation eclipse will restart.*

Once installation is completed, follow steps below to create a Drool project:

1. *Open eclipse*
2. *Go to File → New → Other (pop up appears)*
3. *Select Drools project from DROOLS folder.*
4. *After selection click on next button. A dialog appears*
5. *Enter a project name and click on next button. A new window appears, select first two options for a simple rule else uncheck the options.*
6. *Click on configure workspace settings, a pop up appears*
7. *Click on "add" button, a window appears*
8. *Click on create a new DROOLS 7 runtime button.*
9. *From folder dialog, browse a drive and select a blank folder and click on "Ok".*
10. *Click on "OK" button of DROOLS runtime window.*
11. *Click on "Ok" button of install drools runtime window.*
12. *Click finish to create a project.*

To add rules:

1. *Create a package*
2. *Right click on the package*
3. *Select new → Other. A dialog appears*
4. *Select Rule Resource from Drools folder*
5. *Click next. A dialog appears.*
6. *Enter Rule Name*
7. *Click on finish.*

To add other java class files:

1. *Create a package*
2. *Right click on the package*
3. *Select new → Class. A dialog appears*
4. *Select source folder*
5. *Enter class name and*
6. *Click finish. Class will be created.*

Drools provides APIs to allow provider implementations to be connected to its library of dependency modules that are required during rule development/compiling, and some are required at runtime. A maven project can be created to specify the Drools dependencies in POM.xml file. The POM.xml contains information about the project and configuration details used by maven to build the project. The following is a description of the important libraries that make up JBoss Drools:

- Knowledge-api: This provides the interfaces and factories (Example: org.drools.KnowledgeBase, org.drools.builder.KnowledgeBuilder, org.drools.runtime.StatefulKnowledgeSession, org.drools.runtime.StatelessKnowledgeSession, org.drools.agent.KnowledgeAgent, org.drools.KnowledgeBaseFactory, org.drools.builder.KnowledgeBuilderFactory, etc.,).
- Drools-core: This is the core engine, runtime component. this is the core engine, runtime component. It contains both the RETE and LEAPS engines.
- Drools-compiler: This dependency contains the compiler components to take rule source and build executable rule-bases. This is the main package to load rules and a runtime dependency of an application. This depends on drools-core.
- RuntimeManager: This manages RuntimeEngines that are built with KieSession and TaskService to provide an executable environment for processes and user tasks.
- Drools-jsr94: This is an essential layer to the drools-compiler component.
- Drools-decisiontables: This is the decision tables 'compiler' component to support excel and CSV inputs.

For the latest information on dependencies in a release, use POM release details, which can be found on the maven repository website. It is also possible to rely on maven to configure dependencies using configuration XML (POM) file instead of setting it programmatically. Below shows the screenshot of how to add knowledge-api (kie – knowledge is everything) and other dependencies to the pom.xml:

```

<?xml version="1.0" encoding="UTF-8">

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">

 <modelVersion>4.0.0</modelVersion>
 <groupId>ECARulesByGrace.group</groupId>
 <artifactId>AdaptiveECARuleModel</artifactId>
 <version>1.0</version>

 <dependencies>
 <!-- Start dependencies for the other Kie Modules -->
 <dependency>
 <groupId>org.drools</groupId>
 <artifactId>drools-templates</artifactId>
 <version>6.4.0.Final</version>
 </dependency>
 <dependency>
 <groupId>org.jbpm</groupId>
 <artifactId>jbpm-runtime-manager</artifactId>
 <version>6.4.0.Final</version>
 </dependency>
 <dependency>
 <groupId>org.kie</groupId>
 <artifactId>kie-api</artifactId>
 <version>6.4.0.Final</version>
 </dependency>
 <dependency>
 <groupId>org.kie</groupId>
 <artifactId>kie-internal</artifactId>
 <version>6.4.0.Final</version>
 <scope>compile</scope>
 </dependency>
 <dependency>
 <groupId>org.jbpm</groupId>
 <artifactId>jbpm-persistence-ipa</artifactId>
 <version>6.4.0.Final</version>
 </dependency>
 </dependencies>

</project>

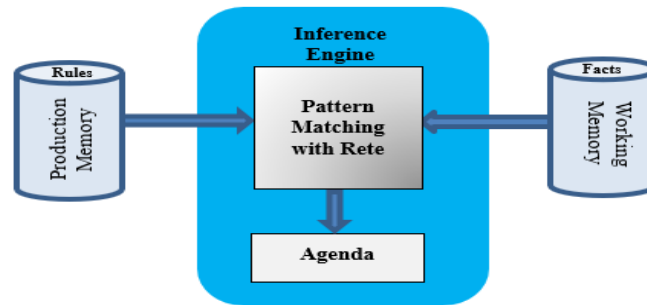
```

POM file – Maven dependencies and configuration

## Appendix VII – JBoss Drools Components

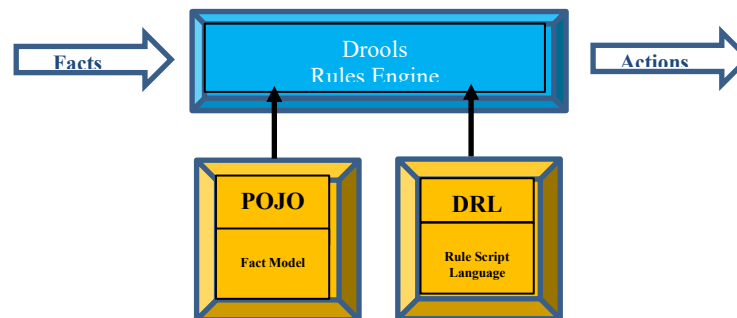
### Drools Expert: Rule Engine

Drools began as a specific type of rule engine called a Production Rule System (PRS) and it was based on the Rete algorithm for pattern matching [51]. Rules are stored in the production memory, while facts are maintained in the working memory, see Figure 7.1.2. During the execution session, facts are added into the working memory where they are updated or removed. The Agenda manages the execution order of conflicting rules during execution.



High level view of a rule engine

Figure 7.1.3 shows how Drools rule engine applies the rules to the facts. The facts are the data to be processed while the fact model tells the engine how to interpret the facts. Rules in the DRL (Drools Rules Language) tell the rule engine what actions to take when certain conditions are met, and in turn fires specified actions.



The main parts of a rule engine

### Rule Script Language

Drools offers four different ways to define rules. The first is to use the native rules language (Drools Rules Language - DRL), which is very easy to implement for most developers. For example, the following represent the business rule “when install request if rack utilization is above two thousand then process manage data centre space”.

```

1 package org.kanana;
2 import org.kanana.Rack;
3 import org.kanana.Request;
4 import org.kanana.RuleName;
5 import function org.kanana.Utility.help;
6
7 rule "R3"
8 when
9 Request(Type == 'Install')
10 Rack(Utilization >= '2000')
11 then
12 help(drools,"Drools runtime Info... ");
13 System.out.println("Action after rule triggered: " + "Process(Name == 'P5-Manage DC Space')");
14 end

```

The second way to define rules, is to use the template language DSL (see below), which is translated into the native DRL at real-time.

**When there is an install request**  
**Rack Utilization >= 200**  
**Then**  
**PS-Manage DC Space**

The third way is to use spreadsheets and a fourth way is to use a Rule Template (see section 7.2). At runtime, the second to fourth ways will need to be translated into the rule script language, e.g. DRL.

### **Drools Flow (jBPM)**

Drools Flow executes business process or workflow for the Drools platform. A workflow or business process shows the flow of execution of several processes. Processes are useful in describing activities or tasks status or states. Drools jBPM allows users to define, execute and monitor their business processes.

### **Drools Guvnor**

Drools Guvnor is a web and network components. It provides user-friendly interfaces to a business rules manager, which allows managing and changing rules in a multi-user environment.

### **Drools Fusion**

Drools Fusion is responsible for enabling of an event processing for the Drools platform. An event processing concept is concerned with the processing of multiple events to identify meaningful events.

### **Drools Planner (OptaPlanner)**

Drools Planner is the planning engine written in Java to solve constraint satisfaction problems efficiently. It can optimize planning in order to execute more rules with less resource.

### **Drools Eclipse Java Plugin (IDE)**

Another important part of Drools is the Eclipse IDE (Integrated Development Environment). Eclipse is an open-source environment for developing applications for the most popular platforms. It helps with creating and compiling rules and processes and, also offers creation of facts as POJO classes and has lots of other valuable features. The Eclipse IDE version used in this research is Eclipse Java EE IDE for Web Developers,

## ***Appendix VIII – Editing Business Rules via Test Client***

### **Adding business rule components via the ECA Model Test Client:**

Steps to add and execute business rule components via the ECA Model Test Client:

1. Enter Rule Name
2. Select “Create Rule” Option (Checkbox)
3. On the New EVENT form panel, enter event object, property, value and operator
4. On the New CONDITION form panel, enter condition object, property, value and operator
5. On the New ACTION form panel, enter action object, property, value and operator
6. Click “Add Rule” button to add the rule and components
7. Repeat Steps 1-6 for each rule
8. Execute by clicking on “Execute Rules” button
9. Click “Display All” button to show some statistics on business rules and process information

### **Modifying business rule components via the ECA Model Test Client:**

Steps to modify and execute business rule components via the ECA Model Test Client:

1. Enter Rule Name
2. Select “Create Rule” Option (Checkbox)
3. On the New EVENT form panel, enter event object, property, value and operator
4. On the New CONDITION form panel, enter condition object, property, value and operator
5. On the New ACTION form panel, enter action object, property, value and operator
6. Click “Add Rule” button to add the rule and components
7. Repeat Steps 1-6 for each rule
8. Execute by clicking on “Execute Rules” button
9. Click “Display All” button to show some statistics on business rules and process information

### **Deleting business rule components via the ECA Model Test Client:**

Steps to delete and execute business rule components via the ECA Model Test Client:

1. Enter Rule Name
2. Select “Create Rule” Option (Checkbox)
3. On the New EVENT form panel, enter event object, property, value and operator
4. On the New CONDITION form panel, enter condition object, property, value and operator
5. On the New ACTION form panel, enter action object, property, value and operator
6. Click “Add Rule” button to add the rule and components
7. Repeat Steps 1-6 for each rule
8. Execute by clicking on “Execute Rules” button
9. Click “Display All” button to show some statistics on business rules and process information